Thesis

Development of Algorithms for the Direct Multi-Configuration Self-Consistent Field (MCSCF) Method

Submitted to Imperial College London
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
by

Shaopeng Li

May 2011
Department of Chemistry
Imperial College London
To my Parents
The work in this thesis has not been submitted previously for a degree at this or any other university. Except where explicitly mentioned, the work is wholly my own.

Shaopeng Li
Abstract

In order to improve the performance of the current parallelized direct multi-configuration self-consistent field (MCSCF) implementations of the program package Gaussian [42], consisting of the complete active space (CAS) SCF method [43] and the restricted active space (RAS) SCF method [44], this thesis introduces a matrix multiplication scheme as part of the CI eigenvalue evaluation of these methods. Thus highly optimized linear algebra routines, which are able to use data in a sequential and predictable way, can be used in our method, resulting in a much better performance overall than the current methods. The side effect of this matrix multiplication scheme is that it requires some extra memory to store the additional intermediate matrices. Several chemical systems are used to demonstrate that the new CAS and RAS methods are faster than the current CAS and RAS methods respectively.

This thesis is structured into four chapters. Chapter One is the general introduction, which describes the background of the CASSCF/RASSCF methods. Then the efficiency of the current CASSCF/RASSCF code is discussed, which serves as the motivation for this thesis, followed by a brief introduction to our method. Chapter Two describes applying the matrix multiplication scheme to accelerate the current direct CASSCF method, by reorganizing the summation order in the equation that generates non-zero Hamiltonian matrix elements. It is demonstrated that the new method can perform much faster than the current CASSCF method by carrying out single point energy calculations on pyracylene and pyrene molecules, and geometry optimization calculations on anthracene$^+$ / phenanthrene$^+$ molecules. However, in the RASSCF method, because an arbitrary number of doubly-occupied or unoccupied orbitals are introduced into the CASSCF reference space, many new orbital integral cases arise. Some cases are suitable for the matrix multiplication scheme, while others are not. Chapter Three applies the scheme to those suitable integral cases that are also the most time-consuming cases for the RASSCF calculation. The coronene molecule - with different sizes of orbital active space - has been used to demonstrate that the new RASSCF method can perform significantly faster than the current Gaussian method. Chapter Four describes an attempt to modify the other integral cases, based on a review of the method developed by Saunders and Van Lenthe [95]. Calculations on coronene molecule are used again to test whether this implementation can further improve the performance of the RASSCF method developed in Chapter Three.
Acknowledgements

I would like to present my sincere thanks for all the support and encouragement I have received from many people.

First of all, in particular I would like to thank Prof. Mike A Robb and Dr. Mike J Bearpark for offering me the opportunity to carry out this work and most importantly for their supervision and patient guidance.

Thanks are also due to all past and present members of the Imperial College Computational Photochemistry Group, for a very memorable and pleasant time especially for the tradition of the various topics discussed during lunchtime.

I would like to thank Mr. Abdihakim Hassan, Mr. Benjamin PC Jones, Mr. Jordan L Muscatello, and Mr. Lee M Thompson for proofreading different parts of the manuscript. In particular I would like to thank Dr. Mike J Bearpark for reprooofreading some parts of the manuscript.

For financial support, thanks are due to ORSAS and to the KWOK Foundation.

Finally, I would like to acknowledge my family. My family has always been a source of strength and plays a very important part in my life, offering me my sense of identity and keeping my feet firmly on the floor. Thanks for the whole support they have given me.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>4</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>5</td>
</tr>
<tr>
<td>List of Tables</td>
<td>10</td>
</tr>
<tr>
<td>List of Figures</td>
<td>16</td>
</tr>
<tr>
<td>List of Charts</td>
<td>22</td>
</tr>
<tr>
<td>List of Algorithms</td>
<td>23</td>
</tr>
<tr>
<td>1. General Introduction</td>
<td>24</td>
</tr>
<tr>
<td>1.1 Perspective</td>
<td>25</td>
</tr>
<tr>
<td>1.2 The Hartree-Fock Self-Consistent-Field Method</td>
<td>29</td>
</tr>
<tr>
<td>1.3 Electronic Correlation</td>
<td>32</td>
</tr>
<tr>
<td>1.4 Configuration</td>
<td>34</td>
</tr>
<tr>
<td>1.5 The CAS/RAS-SCF Method Overview</td>
<td>36</td>
</tr>
<tr>
<td>1.5.1 One Example</td>
<td>37</td>
</tr>
<tr>
<td>1.5.2 General Theory</td>
<td>38</td>
</tr>
<tr>
<td>1.5.3 Active Space Selection</td>
<td>38</td>
</tr>
<tr>
<td>1.5.4 Full Configuration Interaction (Full CI)</td>
<td>44</td>
</tr>
<tr>
<td>1.6 Configuration Interaction (CI) Method</td>
<td>45</td>
</tr>
<tr>
<td>1.6.1 The CI Method</td>
<td>45</td>
</tr>
<tr>
<td>1.6.2 CI Eigenvalue Problem</td>
<td>48</td>
</tr>
<tr>
<td>1.7 Our Matrix-matrix Multiplication Method</td>
<td>50</td>
</tr>
<tr>
<td>1.7.1 The String Concept</td>
<td>51</td>
</tr>
<tr>
<td>1.7.2 Factorization vs. Non-factorization Concept</td>
<td>54</td>
</tr>
<tr>
<td>1.7.2.1 The Factorized Methods</td>
<td>54</td>
</tr>
<tr>
<td>1.7.2.2 The Unfactorized Methods</td>
<td>57</td>
</tr>
<tr>
<td>1.7.3 Our Matrix-matrix Multiplication Method</td>
<td>58</td>
</tr>
<tr>
<td>1.7.4 Motivation for this Thesis</td>
<td>60</td>
</tr>
<tr>
<td>1.7.5 Inspiration for this Thesis</td>
<td>61</td>
</tr>
<tr>
<td>1.7.5.1 Siegbahn-Knowles-Handy Method</td>
<td>62</td>
</tr>
<tr>
<td>1.7.5.2 Olsen-Roos Method</td>
<td>64</td>
</tr>
</tbody>
</table>
1.8 Scope of this Thesis ............................................................... 65

2. A New Matrix-Matrix Multiplication Approach to the Direct Full CI / CASSCF Method and Its Parallel Implementation 67
   2.1 Introduction ........................................................................ 68
   2.2 Theory ................................................................................ 75
   2.3 Algorithm for Matrix-Matrix Multiplication Method .............. 76
      2.3.1 General Algorithm for Bilinear Contribution ................. 77
      2.3.2 General Algorithm for Linear Contribution .................... 78
      2.3.3 Algorithm for Hartree-Waller Functions Basis ............... 81
   2.4 The Numerical Values .......................................................... 82
      2.4.1 The Addresses of the Configurations ............................... 82
      2.4.2 Reduced List Concept Review ....................................... 83
      2.4.3 The Sign .................................................................... 87
   2.5 Implementation Details ....................................................... 87
      2.5.1 Matrix $C_{\alpha \beta}$ Generation ..................................... 89
      2.5.2 Matrix $\mathbf{X}$ Generation ............................................ 91
      2.5.3 The Elements Density in $\mathbf{X}$ .................................... 95
      2.5.4 Matrix Multiplication .................................................. 97
      2.5.5 Memory Requirement .................................................. 99
      2.5.6 CI Vector Updating Operation ....................................... 100
   2.6 Parallelism ........................................................................ 102
   2.7 Test Calculations and Performance ..................................... 113
      2.7.1 Verification of the New Method ..................................... 113
      2.7.2 Performance Investigation ............................................ 117
         2.7.2.1 Serial Performance ............................................... 117
         2.7.2.2 Parallel Performance ............................................. 118
      2.7.3 Performance on Real Applications .................................. 126
   2.8 Summary .......................................................................... 128

3. A New Matrix-Matrix Multiplication Approach to the Direct RASSCF Method and Its Parallel Implementation 130
   3.1 Introduction .................................................................... 131
PhD Thesis: Algorithm Development for Direct MCSCF Method

3.2 CASSCF Matrix Multiplication Algorithm Review.......................... 144

3.3 RASSCF Notation and Theory.............................................. 146
  3.3.1 RASSCF Configurations Notation...................................... 146
  3.3.2 RASSCF CI Vector Updating........................................... 150
  3.3.3 Concepts of Model Space and Expanding............................ 152
  3.3.4 Orbital Integral Types.................................................. 156
  3.3.5 Contribution Weight of These Model Integral Types.............. 158

3.4 Algorithm for the Selected 7 Cases...................................... 164
  3.4.1 Current Unfactorized Direct Method................................. 165
  3.4.2 Factorized Matrix Multiplication Approach......................... 168
    3.4.2.1 \((ij|yz)\) Model Orbital Integral Cases....................... 169
    3.4.2.2 \((wx|ij)\) Model Orbital Integral Cases....................... 171

3.5 Numerical Values.......................................................... 172
  3.5.1 String Category, Addresses of Strings and Configurations...... 172
  3.5.2 The Sign of Symbolic Matrix Elements.............................. 177
  3.5.3 Brief Review and Examples of the Propagation Rules
        \((^P(RASX))\).............................................................. 178

3.6 Implementation Details.................................................... 183
  3.6.1 CI Vector Restructure.................................................. 186
    3.6.1.1 CI Vector Restructure for Slater Determinant Basis....... 186
    3.6.1.2 Normalization of the CI Vector for Hartree-Waller
            Functions........................................................... 187
  3.6.2 Extension of the Model Subspace.................................... 191
  3.6.3 Dimension of the X Array (Eqs. 3.4.6a and 3.4.8a).............. 192
  3.6.4 \(\sigma\) Vector Updating............................................... 193
  3.6.5 Memory Requirement.................................................... 194

3.7 Parallelism........................................................................ 198

3.8 Timing Data for the Modified Cases..................................... 204
  3.8.1 New Method Verification................................................ 205
  3.8.2 New Method Performance............................................... 211

3.9 Summary.......................................................................... 219
4. An Attempted Implementation for Modifying the Rest Integral Cases of the Direct RASSCF Method

4.1 Introduction ............................................................................................................ 222
4.2 Review of Saunders and Van Lenthe’s Method .................................................... 223
   4.2.1 \((i\delta|\epsilon)\) Model Orbital Integral Cases .................................................... 225
   4.2.2 \((\delta|\epsilon')\delta')\) Model Orbital Integral Cases ............................................ 228
4.3 An Attempted Implementation to the Remaining Cases and Its Performance .......................................................... 232
4.4 Summary ................................................................................................................ 237

Conclusion .................................................................................................................. 239

A. The Second Quantization Formalism and Hamiltonian Matrix Diagonalization

A.1 The Second Quantization Formalism .................................................................... 242
   A.1.1 The Fock Space .................................................................................................... 242
   A.1.2 Annihilation and Creation Operators ................................................................. 243
   A.1.3 The Excitation Operator ...................................................................................... 244
A.2 Hamiltonian Matrix Diagonalization .................................................................... 245

B. CASSCF Implementation Details ............................................................................. 247

C. RASSCF Implementation Details ............................................................................ 266

Bibliography .................................................................................................................. 289
List of Tables

2.1 The 11 integral classes constructed under the conditioning index loop given in Eq. 2.3.11. Column $f(I_{ijkl})$ is the factor corresponding to the integral classes $I_{ijkl}$, and the Coefficient column gives the value of $E_iE_l - \delta_{ik}E_{il}$ corresponding to the particular integral class. In this table the coefficients indicate that in the cases of integral classes 1, 3, 5, 7, 8, 9, and 10, there are linear contributions, which can be calculated together with the one-electron linear contributions.................................................. 80

2.2 The addresses and the binary representation ($\phi_{\xi}$) of the $\xi$-strings ($\xi \in \{\alpha, \beta\}$) for a system with $M = 6$ and $N = 3$............................................................ 84

2.3 Generating the excitation list in the case of $i \neq j$ by the $E_{43}^{\alpha}$ operation.......... 86

2.4 Comparison of the lengths of the full string list and the reduced string lists ($\xi \in \{\alpha, \beta\}$)................................................................. 86

2.5 String pairs generated and the associated sign from reduced list $L_{n-1}^{M-2}$ for the orbital indices ........................................................................ 92

2.6 $X_{lin}$ elements generation for 1-e contribution when the $ij$ index is $i = 6, j = 4$ ........................................................................ 93

2.7 Matrix elements generation for 2-electron linear contribution when the contribution is from integral class 9 (Table 2.3, in this example this is $i = 6, l = 3$) ....................................................... 94

2.8 The Linda Operations.................................................................. 105

2.9 Verification test on the new method. We only list the result of serial CASSCF single point energy Slater determinants basis calculations on three systems (naphthalene, acenaphthylene, and pyracylene) by using both the new method and the current method. The computation cost of the current method on pyrene is too high to be carried out here. The outputs of these three systems, e.g. the
List of Tables

2.10 The excited states calculation of Hartree-Waller (HW) singlet, HW triplet, Slater determinant (SD) singlet, triplet of naphthalene and ground state optimization of naphthalene$^+$ (doublet) by using both of the new method (New) and the current method (Old). The Slater determinant singlet calculation generates 4 states, which are singlet S$_0$, triplet T$_1$, triplet T$_2$, and singlet S$_1$. Since the HW singlet calculation will not result any triplet energy, we will only compare the singlet energies generated by both SD singlet basis and HW singlet basis. We can see the SD singlet S$_0$ and S$_1$ energies are the same as HW singlet S$_0$ and S$_1$ energies. By contrast, SD triplet calculation will only give triplet energies. Thus we can see SD triplet and HW triplet calculations generate identical results...

2.11a Serial timing data for the 4$^{th}$ Davidson iteration when using different bases on the same system of pyracylene (CASSCF(14,14)). We can see from this table that no matter what basis is used, the matrix multiplication method is always much faster than the current method...

2.11b Serial timing data for the 4$^{th}$ Davidson iteration when using different bases on the same system of pyrene (CASSCF(16,16)). We can see from this table that no matter what basis is used, the matrix multiplication method is always much faster than the current method...

2.12a $ij$ level OpenMP timing data of the 4$^{th}$ Davidson iteration for the system of pyracylene (SD CAS(14,14) calculation)...

2.12b $ij$ level OpenMP timing data of the 4$^{th}$ Davidson iteration for the system of pyrene (HW singlet CAS(16,16) calculation)...

2.13 The shared memory timing data for the matrix multiplication level ($kl$ level) parallelization. Since the time required to finish one matrix multiplication is too short, the parallel performance on the matrix multiplication will be strongly affected by the parallel overhead...

2.14a Linda only timing data of the 4$^{th}$ Davidson iteration for the system of pyracylene. We can obtain a linear scaling of this type of calculation because in
this type of parallelization, each processor can access its own memory without any overhead occurring.\\[\text{124}\]

2.14b Linda only timing data of the 4th Davidson iteration for the system of pyrene. We can obtain a linear scaling of this type of calculation because in this type of parallelization, each processor can access its own memory without any overhead occurring.\\[\text{124}\]

2.15a Linda+OpenMP timing data of the 4th Davidson iteration for the system of pyracylene. In this calculation, the number of nodes is fixed at 2 while the number of shared memory processors varies.\\[\text{125}\]

2.15b Linda+OpenMP timing data of the 4th Davidson iteration for the system of pyrene. In this calculation, the number of nodes is fixed at 4 while the number of shared memory processors varies.\\[\text{125}\]

2.16 The calculation results of the ground state geometry optimization of two hydrocarbon cations (anthracene$^+$ / phenanthrene$^+$) carried out using both current and the new methods. The initial geometry structure data is obtained from a MMVB calculation. In order to compare the results with ref.[47], the same basis sets (6-31G* that includes polarization $d$ functions on the carbon atoms) is used. From this table we can see the new method works properly as well as performs much faster than the current method when used for real problems. The timing data of ref.[47] is not listed because the reference calculations were carried out by using a different type of computer hardware...\\[\text{128}\]

3.1 The model integral types and their expanding to the full RAS space orbital integrals based on the number of model space indices dropped in the model subspaces.\\[\text{159}\]

3.2 Possible combinations of excitations between RAS subspaces. ($x \rightarrow w$ indicates excitation from orbital $x$ to orbital $w$).\\[\text{161}\]

3.3 Number of model space excitations $a^\dagger_{\gamma\gamma}a_{\sigma\sigma}$ between RAS model spaces, provided $\text{MxHole} \geq 2$ and $\text{MxElec} \geq 2$; analytically and numerically for different RAS2 orbital subspace sizes. ($x \rightarrow w$ indicates excitation from orbital $x$ to orbital $w$).\\[\text{161}\]

3.4 Number of double excitations $a^\dagger_{\gamma\gamma}a_{\sigma\sigma}(x \rightarrow w)$, $a^\dagger_{\alpha\alpha}(z \rightarrow y)$ between RAS model subspaces; analytically and numerically for different RAS2 orbital subspaces (Constructed using Tables 3.2 and 3.3 with Eq. 3.3.24; $M$ represents $M(\text{RAS2})$, $\gamma, \sigma \in \{\alpha, \beta\}$). This is equivalent to the integral cases listed in Table 3.1.\\[\text{162}\]
3.5 General overview of the total numbers of distinctive \( w, x, y, z \) quadruples for \( MxHole=2, MxElec=2 \), analytically and numerically for a few RAS2 subspace sizes (\( M \) represents \( M(RAS2) \)). .......................................... 163

3.6 The percentage of the dimension for the dominating cases over the entire dimension for \( MxHole = 2, MxElec = 2 \), analytically and numerically for several different RAS2 subspace sizes (\( M \) represents \( M(RAS2) \)). ................. 163

3.7 The number of independent parallel tasks, defined by the orbital indices of the model space for different sizes of the RAS2 subspace with \( MxHole = MxElec = 2 \). ......................................................... 200

3.8 Single point energy Slater determinant singlet calculations on systems Naphthalene, Acenaphthalene, and Pyracylene by using both the new method (indicated as New in table) and the current method (indicated as Old in table). The results show us that the new method works properly ....................... 207

3.9 Single point energy calculations on Naphthalene by using both the new method and the current method with different basis functions. HW stands for Hartree-Waller. SD is for Slater determinant. And configs. means configurations. We can see in this table that the outputs of the SD singlet / triplet calculations are identical to the outputs of the HW singlet / triplet calculations. The result shown in this table again proves the new method works properly ....................... 207

3.10a The convergence of a series of RASSCF calculation to a reference CASSCF calculation (HW singlet basis is used). The CASSCF(12,12) single point energy calculation on Acenaphthylene is carried out as a reference calculation. Then a series of RASSCF calculations with different combinations of \( M(RAS1)+M(RAS2)+M(RAS3) \), and (\( MxHole,MxElec \)) are carried out to show the convergence of the accuracy of these RASSCF calculations. In this table, the number of RAS1, RAS2, and RAS3 subspace orbitals is set fixed while \( MxHole \) and \( MxElec \) vary. We will use \( RAS_{i,j}(a,b,c,d) \) to indicate a calculation with \( MxHole = a, M(RAS1) = b, MxElec = c, \) and \( M(RAS3) = d \) in the table (1 hartree = 27.2114 eV). ................................................. 209

3.10b In this table we will set \( MxHole = MxElec = 1 \) while the number of RAS2 subspace orbitals varies. We will still use \( RAS_{i,j}(a,b,c,d) \) to indicate a calculation as in Table 3.10a. 1 hartree = 27.2114 eV ................................................. 210

3.10c In this table we will set \( MxHole = MxElec = 2 \) while the number of \( M(RAS2) \) varies ................................................................. 210

3.11 Timing data of the 7 cases only for the cases of the total number of orbitals
remains the same while the number of RAS2 subspace varies (single excitation only: $M_{\text{Hole}}=M_{\text{Elec}}=1$)............................. 213

3.12 Timing data of the 7 cases only for the cases of the total number of orbitals remains the same while the number of RAS2 subspace varies (double excitation only: $M_{\text{Hole}}=M_{\text{Elec}}=2$).................................................. 213

3.13 Timing data of the 7 cases only for the cases of the number of RAS2 orbitals remains the same while the number of RAS3 subspace varies (single excitation only: $M_{\text{Hole}}=M_{\text{Elec}}=1$).................................................. 213

3.14 Timing data of the 7 cases only for the cases of the number of RAS2 orbitals remains the same while the number of RAS3 subspace varies (double excitation only: $M_{\text{Hole}}=M_{\text{Elec}}=2$).................................................. 214

3.15 Serial timing data for the 4th Davidson iteration when using different basis on the same calculation of coronene (RAS(24,7+10+7)[1,1]). We can see from this table that no matter what basis is used, the RASSCF matrix multiplication method always performs faster than the current method.......................... 216

3.16 Shared memory parallel timing data for the 4th Davidson iteration of both the current method and the new method with only the first 7 time consuming cases are modified (OpenMP only)................................................................. 217

3.17 Timing data of the 4th Davidson iteration of both the current method and the new method when using Linda only.................................................. 217

3.18 Hybrid parallel timing data when using 2 Linda workers and the number of shared memory threads varies of both methods. From the speedup rows of both methods, we can see both methods scale perfect linearly with the increasing number of processors................................................................. 218

4.1 The model integral types in the SV method that excludes the RAS1 subspace and their expanding to the full orbital space ........................................ 224

4.2 The configuration blocks applied in the SV method ........................................ 225

4.3 The different types of interaction between configurations for the last 3 model orbital integral cases in Table 4.1................................................................. 225

4.4 Timing data of calculations on the cases of the total number of orbitals remains the same while the number of RAS2 subspace varies (single excitation only).................................................. 234

4.5 Timing data of calculations on the cases of the total number of orbitals remains the same while the number of RAS2 subspace varies (double excitation only).................................................. 234

4.6 Timing data of calculations on the cases of the number of RAS2 orbitals............. 234
remains the same while the number of RAS3 subspace varies (single excitation only)

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7</td>
<td>Timing data of calculations on the cases of the number of RAS2 orbitals remains the same while the number of RAS3 subspace varies (double excitation only)</td>
</tr>
<tr>
<td>4.8</td>
<td>Shared memory parallel timing data for the 4th Davidson iteration of the current method (Old), the method developed in Chapter 3 (New₀), and the modified method developed in this chapter (New₁) using OpenMP only</td>
</tr>
<tr>
<td>4.9</td>
<td>Distributed memory parallel timing data for the 4th Davidson iteration of the current method (Old), the method developed in Chapter 3 (New₀), and the modified method developed in this chapter (New₁) using Linda only</td>
</tr>
<tr>
<td>4.10</td>
<td>Hybrid parallel timing data when using 2 Linda workers and the number of shared memory threads varies of the current method (Old), the method developed in Chapter 3 (New₀), and the modified method developed in this chapter (New₁)</td>
</tr>
</tbody>
</table>
**List of Figures**

1.1 Possible splitting of different orbitals in a general MCSCF formalism .................. 44
1.2 A brief visualized version of the CI vector updating process of the current CAS/RAS implementation in *Gaussian*. From this figure we can see each time when a non-zero matrix element is obtained, the elements in the CI vectors are used directly for updating. But these elements in the CI vectors are not usually sequentially ordered for different non-zero Hamiltonian matrix elements. The gap between these elements will cause a frequent main memory access, which will slow down the calculation performance significantly.............................. 62
2.1 The bits-operation to the reduced list in order to generate the corresponding full strings of the excitation list (using the system of 6 active electrons acting in the space of 6 active orbitals) ......................................................... 85
2.2 General structure of shared memory multiprocessors........................................ 103
2.3 General structure of distributed memory multiprocessors................................. 103
2.4 Program flow in an OpenMP model .................................................................. 104
2.5 Communication between processing units when using Linda parallel model........ 105
2.6 The data flowchart for the parallelization as described in the text .................. 109
2.7 When using shared memory parallelism only, there will be three ways to carry out the CI vector updating calculation. i) parallelizing on $ij$ level, means each processor will be allocated one $ij$ index. ii) parallelizing on $kl$ level, indicates all the 9 CPUs will be used for parallelizing the matrix multiplication corresponding to one $ij$ index. Or iii) a combination of both. For example, if we set 3 processors to be used for $kl$ level parallelization, then we know 3 $ij$ index pairs will be executed at the same time ($N_{Procij} = \text{Int} \left( \frac{N_{ProcS}}{N_{Prockl}} \right)$). Therefore, 3 processors will act as masters and each uses 3 processors (including the master itself) for parallelizing the matrix multiplication corresponding to the $ij$ index ...................... 111
2.8 The visualized result of the timing data given in Table 2.9. From this figure we can see the new method can always perform faster than the current method. Particularly when the investigated system becomes larger and larger, the matrix multiplication method can perform faster and faster .............................................. 115
2.9 Speedup vs. the number of processors used for shared memory parallelization
using both methods on the systems of pyracylene (left hand side) and pyrene (right hand side)……………………………………………………………………………………………... 120

2.10 kl level shared memory parallelization performance of the new method. We can see due to the execution time for each ij index pair being too short, the parallelization on the kl level performs very poorly……………………………………… 122

2.11 The performance of both methods when using distributed memory parallelization (Linda) only. We can see when using only Linda, the speedup scales linearly due to each node accesses its own memory without the restrictions of memory-CPU bandwidth and overhead……………………………………………………………………………………………….. 124

2.12 The speedup vs. number of processors for using Linda+OpenMP parallel scheme. We can see for the CAS(16,16) calculation, the performance of the new method is worse than the old method. This is because when using Linda only the performance of the new method is slightly worse than the current method (Figure 2.11)…………………………………………………………………………………………………………… 125

2.13 The geometry structure of the ground states optimization (using the newly-developed CASSCF method) of the investigated systems. (a) gives the final optimized structure of the anthracene cation, while (b) is the optimized structure of the ground state of phenanthrene cation with 1B_{1} symmetry species [132] of D_{2h} symmetry group respectively. The unit of the values (bond length) is Å. The result agrees with the reference [47]……………………………………………………………………………………………….. 127

3.1 The construction of the model external space. Since the occupation number of the external space is at most 2, the external space that contains a large number of virtual orbitals can be replace by two model external orbitals. This operation reduces the size of the external space dramatically and thus more easier to generate the external coupling coefficients. Moreover, the obtained model symbolic matrix elements will remain constant for all the full space configuration pairs K,L with the same occupation and spin pattern as the corresponding model configurations……………………………………………………………………………………………….. 135

3.2 The model RAS space generation in the Klene’s method. The numbers of orbitals in the model RAS1 and RAS3 subspaces are determined by the maximum number of holes (MxHole) and the maximum number of electrons (MxElec) allowed in RAS1 and RAS3 subspaces, respectively. Therefore the size of the model space is determined by the size of the RAS2 subspace and the values of MxHole and MxElec……………………………………………………………………………………………….. 136

3.3 A visualized graph to demonstrate the first 7 cases in Table 3.1 are the most time
3.4 The model string generating process from a global string, \(K_{n}^{11,32}\), when \(\text{MxHole} = \text{MxElec} = 1\). The generated model string can represent all the global strings given in Section 3.3.1.  

3.5 Example 1: the expansion process of any of the global orbital indices is in neither RAS1 subspace nor RAS3 subspace. From this figure we can see once the first substring address of RAS1/RAS3 subspace is known, it is easy to get the addresses of the rest substrings within this subspace without using Eq. 3.5.6.  

3.6 Example 2: the excitation from RAS1 subspace to RAS3 subspace. Since in this excitation both RAS1 and RAS3 subspaces are involved, the expanded orbital integral, \((i|j)\), and the symbolic matrix elements, \(\text{sgn}_{ij}^{K_{\alpha\beta}}\), will not remain constant any more.  

3.7 Unwanted double contributions may arise when two model string pairs generated by different \(wx\) pairs result in the same global string pair through the application of the propagation rules concept. In this figure we can see both model string pairs \(K_{\xi}^{01,14}a_{14}\xi a_{14}\xi L_{01,14}^{01,14}\) and \(K_{\xi}^{01,14}a_{14}\xi a_{14}\xi L_{01,14}^{01,14}\) give the global string pair \(K_{\xi}^{01,14}a_{14}\xi a_{14}\xi L_{01,14}^{01,14}\).  

3.8 General structure of parallel RASSCF matrix multiplication method.  

3.9a When the number of orbitals in RAS1, RAS2, and RAS3 subspaces is set fixed, the result of the RASSCF calculation can become closer and closer to the reference CASSCF calculation when the number of \(\text{MxHole}/\text{MxElec}\) increases. The 0.0 line is the result given by the reference CASSCF calculation.  

3.9b When the values of \(\text{MxHole}\) and \(\text{MxElec}\) are both set as 1, we can see the results of RASSCF calculations are getting closer and closer to the result of the reference CAS calculation when the number of RAS2 subspace orbitals increases. The 0.0 line is the result given by the reference CASSCF calculation.  

3.9c When the values of \(\text{MxHole}\) and \(\text{MxElec}\) are both set as 2, we can see the results of RASSCF calculations are getting closer and closer to the result of the reference CAS calculation when the number of RAS2 subspace orbitals increases. The 0.0 line is the result given by the reference CASSCF calculation.  

3.10 The shared memory parallelism only performance of both the old and the new methods. From this figure we can see the speedup of the parallelism is almost linearly scaled.
List of Figures

3.11 The distributed memory parallelism only performance of both the old and the new methods. From this figure we can also see the speedup of the parallelism is linearly scaled. .................................................................................................................. 218

3.12 The performance of the hybrid parallelism (Linda+OpenMP). The number of Linda workers is set as 2 while the number of shared memory processors varies. Since both the OpenMP only performance and the Linda only performance are linearly scaled, the hybrid parallelism performance is also linearly scaled .......... 219

B.1 Subroutine calling sequence for CASSCF matrix multiplication CI vector updating algorithm .......................................................................................................................... 251

B.2a Linda communication for the Slater determinant basis when \( N_\alpha \neq N_\beta \) of the CASSCF matrix multiplication algorithm for CI vector updating .................................................................................................................. 252

B.2b Linda communication for the Slater determinant basis when \( N_\alpha = N_\beta \), and Hartree-Waller functions basis of the CASSCF matrix multiplication algorithm for CI vector updating .................................................................................................................. 253

B.3a CI vector updating process for the Slater determinant basis when \( N_\alpha \neq N_\beta \) in the CASSCF matrix multiplication algorithm .................................................................................................................. 254

B.3b CI vector updating process for the Slater determinant basis when \( N_\alpha = N_\beta \) and the Hartree-Waller functions basis in the CASSCF matrix multiplication algorithm .................................................................................................................. 255

B.4a Subroutine UpSLin generates the matrix elements for the linear contribution to the CI vector updating (1e linear). If using Slater determinants basis, the matrix elements for \( \beta \) -electron contribution must be generated .................................................................................................................. 256

B.4b (continued from Figure B.4a): Subroutine UpSLin generates the matrix elements for the linear contribution to the CI vector updating (2e linear). If using Slater determinants basis, the matrix elements for \( \beta \) -electron contribution must be generated .................................................................................................................. 257

B.4c Matrix multiplication for the linear contribution to the CI vector updating is carried out in subroutines UpdWrX and UpdXWr .................................................................................................................. 258

B.5a In subroutine UpdSX, the matrix \( X \) for all possible \((kl)\) index pairs corresponding to one \((ij)\) index is generated. Then this matrix is used for matrix multiplication to generate the intermediate matrix \( M \). The CI vector is then updated by carrying out scalar times a vector style calculation on \( M \) (shown in Figure B.5b) .................................................................................................................. 259

B.5b (continued from Figure B.5a): After the matrix \( M \) for certain \((ij)\) index is obtained, scalar times a vector type calculation is used for updating the CI vector of this \((ij)\) index. The excitation list, \( \chi_i \), in this figure is generated from subroutine UpXLin / UpSLin (Figure B.4a) .................................................................................................................. 260
PhD Thesis: Algorithm Development for Direct MCSCF Method

B.6 Subroutine calling sequence for density matrix algorithm............................. 261
B.7 Subroutine calling sequence for diagonal Hamiltonian algorithm...................... 262
B.8 Subroutine calling sequence for reduced Hamiltonian algorithm........................ 263
B.9 Linda communication for density matrix algorithm...................................... 264
B.10 Linda communication for diagonal Hamiltonian algorithm.............................. 264
B.11 Linda communication for reduced Hamiltonian algorithm.............................. 265

C.1 Subroutine calling sequence for RASSCF matrix multiplication CI vector updating algorithm. For Hartree-Waller functions, simply replace the subroutines RJABC1/RUABC1 and RJABC2/RUABC2 to RJHWC1/ RUHWC1 and RJHWS2 / RUHWS2, respectively................................................................. 271
C.2a Linda communication for the Slater determinant basis when $N_{\alpha} \neq N_{\beta}$ of the RASSCF matrix multiplication algorithm for CI vector updating......................... 272
C.2b Linda communication for the Slater determinant basis when $N_{\alpha} = N_{\beta}$, Hartree-Waller functions basis of the RASSCF matrix multiplication algorithm for CI vector updating........................................................................ 273
C.3 CI vector updating process for the 7 time consuming cases in the RASSCF matrix multiplication algorithm. This is for the general cases of Hartree-Waller functions basis (RJHWC1/RJHWS2) and / or Slater determinant basis when $N_{\alpha} = N_{\beta}$ (RJABC1/RJABC2). For Slater determinant basis when $N_{\alpha} \neq N_{\beta}$, simply replace the subroutines RJbXAA as RJbDAA, RJABC1/RJABC2 as RJABD1/RJABD2, RBiUp3 as RBiUp2, and RXbJAB as RJobAB............................................................. 274
C.4 Linear contribution of the RASSCF method is carried out in RUpXAA. In this subroutine, the full model strings are generated from the reduced model strings, then the list of RAS1 and RAS3 subspaces expansion is generated. If Slater determinants basis is used and $N_{\alpha} \neq N_{\beta}$, the routine UpXSAA is replaced by UpSXAA, and the procedure is repeated for $\beta$ -model strings by using routine UpSXBB............................................................. 275
C.5 Subroutine GetKLs provides the RAS1 and RAS3 subspace excitations lists. Before proceeding with excitation list generation it performs validity tests for the model strings and their corresponding categories, Cat$(i_{h}, i_{r})$ ........................................ 276
C.6 The subroutine UpXSAA represents all other subroutines (UpSSAA, UpSXAA, and UpSXBB) for the linear contribution to the CI vector updating......................... 277
C.7 Subroutine GeOLst generates the excitation list for the model orbital index pair \(wx\). This list is only used for bilinear contribution of the RASSCF matrix multiplication algorithm. The process of list elements generation is very similar to the way of elements generation of the linear contribution shown in Figure C. 6 but much simpler.

C.8a Subroutine RUABC1 generates the elements of the intermediate matrix \(M\). This routine operates the matrix generation for the excitation cases of (2->2,1->1), (2->2,1->2), and (2->2,2->2). Routines RUHWCS and RUABD1 do similar job.

C.8b Subroutine RUABC1 generates the elements of the intermediate matrix \(M\). This routine operates the matrix generation for the excitation cases of (2->2,1->1), (2->2,1->2), and (2->2,2->2). Routines RUHWCS and RUABD1 do similar job.

C.9a Subroutine RUABC2 generates the elements of the intermediate matrix \(M\). This routine operates the matrix generation for the excitation cases of (1->2,2->2), (1->3,2->2), (2->3,2->2), and (3->3,2->2). Routines RUHWS2 and RUABD2 do similar job.

C.9b Subroutine RUABC2 generates the elements of the intermediate matrix \(M\). This routine operates the matrix generation for the excitation cases of (1->2,2->2), (1->3,2->2), (2->3,2->2), and (3->3,2->2). Routines RUHWS2 and RUABD2 do similar job.

C.10 Subroutine RBiUp3 carries out the CI vector updating from the list generated from the \(wx\) model index pair. Similar to the CASSCF matrix multiplication method, this matrix multiplication is carried out once per \(wx\) pair.

C.11 Subroutine calling sequence for density matrix algorithm

C.12 Subroutine calling sequence for diagonal Hamiltonian algorithm

C.13 Subroutine calling sequence for reduced Hamiltonian algorithm

C.14 Linda communication for density matrix algorithm

C.15 Linda communication for diagonal Hamiltonian algorithm

C.16 Linda communication for reduced Hamiltonian algorithm
List of Charts

2.1 The systems used for testing the new method performance. Naphthalene is used as a CASSCF (10,10) system, acenaphthylene is used as a CASSCF (12,12) system, pyraerylene is used as a CASSCF (14,14) system, and pyrene is a CASSCF(16,16) system. Among these systems, the first three are used to verify the new method works while pyraerylene and pyrene are used to demonstrate the performance of the new method vs. the performance of the current method............................................. 114

2.2 The structure of cations anthracene+ and phenanthrene+. The CASSCF calculations of these systems have been reported in reference [47]. And to compare the results, we have used the same reported geometry structure and basis sets ................................................................. 126

3.1 Systems that are used to verify the new developed RASSCF method........ 206

3.2 Coronene structure .............................................................. 212
List of Algorithms

2.1 Matrix multiplication method implementation of the CASSCF method (for the non-zero elements generation please refer to [43]). From this algorithm we see the bilinear matrix multiplication is carried out once per \( ij \) index pair, while the linear matrix multiplication is carried out once after all the orbital indices are looped over. Parallel level 1 indicates the parallel scheme used in the current method [43], while parallel level 2 denotes the parallel scheme for the matrix multiplication………………………………………………………… 88

3.1 RAS linear contribution to the \( \sigma \)-vector updating. \( P \) indicates the propagation rules applied to \( \text{RAS} X(X \in \{1,3\}) \) subspace …………………………… 184

3.2 RAS bilinear contribution to the \( \sigma \)-vector updating (\( w \) and \( x \) are both in RAS2 subspace. If \( w \) and \( x \) are not both in RAS2 subspace, the current method will be used)…………………………………………………………………………… 185
Chapter 1

General Introduction
1. General Introduction

The more progress physical sciences make, the more they tend to enter the domain of mathematics, which is a kind of centre to which they all converge. We may even judge the degree of perfection to which a science has arrived by the facility with which it may be submitted to calculation.

Adolphe Quetelet, in *Instructions Populaires sur le Calcul des Probabilities* (1828)

1.1 Perspective

Computational chemistry is a subfield of theoretical chemistry that implements the mathematical descriptions of chemistry on computers to solve chemical problems. The purpose of this subject is to explain and predict properties and reactivity of chemical compounds by calculating the expectation values of the time-independent Schrödinger equation [1]. Since the middle of the last century, with the rapid developments in computer hardware and software, routines for accurate calculations on a variety of chemically relevant systems have been developed. As well as the ability to provide experimentally comparable values of physical observables for real systems, e.g. [2-10], computational chemistry is capable of giving the mechanisms of chemical processes by allowing one to carry out reaction path calculations on the electronic states of a chemical system, e.g. [11-15]. Computational chemistry is thus not only an important field for its contribution to the theoretical understanding of chemistry, but also an indispensable resource to complement experimental work in other disciplines, such as biochemistry, e.g. [16].

Since the exact solution of the time-independent molecular electronic Schrödinger equation for an N-electron system is almost impossible to obtain\(^1\) (except for a few simple one-electron systems such as the hydrogen atom and the \(\text{H}_2^+\) molecule[1]), there are many different methods (“levels of theory”) developed to find approximations to the

---

\(^1\) This is due to the correlation effect [26] (more detail will be given in the following sections) between the \(N\) electrons. Nevertheless, as we will see later, theoretically, if the full configuration interaction (full CI) method [24,25] is used with an infinite basis set [49], the exact solution to the \(N\)-electron time-independent molecular electronic Schrodinger equation can be achieved. However, since the full CI method is too expensive to carry out, it is not possible to use this method for any but the simplest systems.
exact solution. Apart from density functional theory (DFT) \([17,18]\), which describes the properties of a many-electron system via functionals of the spatially dependent electron density rather than the molecular orbitals (MOs), all other non-empirical methods are based on the MO concept (these methods are called \textit{ab initio} methods \([19,20]\)). An MO describes the “motion” of one electron in the electric field generated by the nuclei and some average distribution of the other electrons. The occupation number of an orbital can be 0 (which means the orbital is empty), 1 (the orbital is singly occupied) or 2 (the orbital is doubly occupied). In the case of a doubly occupied orbital, the Pauli principle requires that the two electrons take the opposite spin.

In practice, different methods require vastly different computational costs (the resources, such as time, used for completing a computation) for the treatment of certain chemical problems. For example, the Hartree-Fock (HF) method \([21-23]\) has low computational cost because it is a single configurational\(^2\) method. But it wouldn’t be a good starting-point when a system has multiconfigurational features. By contrast, some other methods can provide more accurate results but with a very high computational cost, e.g. the full configuration interaction (FCI) method \([24,25]\), which deals with systems with multiconfigurational features very well. The choice of method will depend on the suitability of a method for the problem at hand, the required accuracy of the results and the computational cost.

However, although the HF method is a very cheap method that can reasonably well describe the end points of an organic reaction, e.g. the reactants or products (because these are ground energy states that can usually be described by a single configuration), it is not capable of describing the reaction itself, e.g. the transition structures, reactive intermediates, and excited electronic states, etc., correctly. The main reason for this deficiency of the HF method is because there is only one configuration in the HF method. Another drawback of the HF method is that the effect of the so-called “electron correlation energy” \([26]\) is not included. This effect will be described in more detail in the

---

\(^2\) ‘Configuration’ describes the arrangement of electrons in MOs (more details about this will be given in the following sections).
1. General Introduction

following sections, and is often described by higher-level theories that are designed to correct the HF errors, e.g. those methods that include more than one electronic configuration to construct the wavefunction. The Configuration Interaction (CI) method [27,28] and the Multi-Configuration Self-Consistent-Field (MCSCF) method [29-32] are examples of this type of theory. The MCSCF methods especially, specifically the complete active space (CAS) SCF method [33] and the restricted active space (RAS) SCF method [34], are very powerful methods for studying the excited states of molecules and chemical reaction paths etc [35-41] in which the occupation of active orbitals changes significantly during the process being investigated. However, the current implementations of this powerful method in the program package Gaussian [42] perform poorly, especially for chemical systems with large active spaces. In other words, the computational timing cost of the current MCSCF implementations in Gaussian is too high to be used for systems requiring a large active space. Thus the practical limit for the active space of a CASSCF calculation in Gaussian is currently 14 orbitals [43]. This is primarily because the non-zero Hamiltonian elements are used directly once they are obtained to evaluate the CI eigenvector, resulting in the data in the CI vector being accessed non-sequentially during the CI vector updating process. This thesis introduces a matrix multiplication scheme (so the data in the CI coefficient vector can be used sequentially and highly optimized linear algebra routines can be applied) to the CI eigenvector evaluation process in order to improve the performance of the current MCSCF implementations in Gaussian. In Chapters Two and Three we will present this scheme to improve the performance of the current CASSCF [33] method and the current RASSCF [34,44] variants of the MCSCF methods respectively.

Modern supercomputer architectures are mostly massively parallel machines, although very expensive. Nowadays, even the much cheaper PC clusters also include multi-core parallel processors. Parallel computing uses multiple processing elements, known as multi-processors, simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be diverse and include resources such as a single computer with multiple processors, known as shared
memory multiprocessors, or several networked computers or specialized hardware, known as distributed memory multiprocessors, or any combination of these two. Since the current MCSCF implementations in Gaussian are parallelized, in order to take full advantage of the newly developed method in this thesis, efficient parallel implementations have also been designed to maximize parallel efficiency. More details about how to parallelize the newly developed CASSCF and RASSCF methods will be given in Chapters Two and Three respectively.

In Chapter Two, the molecules of pyracylene [45] (with 14 active electrons in 14 active orbitals) and pyrene [46] (with 16 active electrons in 16 active orbitals) are used to demonstrate the newly-developed CASSCF method performs much faster than the current CASSCF implementation in Gaussian. Then a representative real application - geometry optimization of anthracene+ / phenanthrene+ [47] (with 13 active electrons in 14 active orbitals) - is carried out, using both the new method and the current method to demonstrate the new method is capable of being used in real problems with an improved performance. In Chapter Three, to demonstrate the new RASSCF method performs much faster than the current RASSCF method, the coronene molecule [48] with different sizes of orbital active spaces is used.

However, it is not easy to understand why the current CASSCF/RASSCF methods in Gaussian perform slowly directly without introducing all the background of such a type of method. Therefore, in this chapter, before we investigate the performance itself, the background of this type of method will be introduced first.

In the following, Section 1.2 reviews the most basic method, the Hartree-Fock (HF) SCF method [21-23], to show what a typical SCF / orbital optimization process would be. From Section 1.2 we know that the HF method doesn’t include the correlation between electrons. In order to get a more accurate wavefunction, e.g. the MCSCF wavefunction, the electronic correlation must be included. Thus the electronic correlation is described in Section 1.3. Section 1.4 introduces the types of configurations that are widely used in the multiconfigurational methods, e.g. the MCSCF and CI methods. Then in Section 1.5 the
MCSCF theory (both CASSCF and RASSCF) that recovers the electronic correlation effect is reviewed. Since the time-consuming part of the MCSCF calculation is on the configuration interaction (CI) coefficients optimization (the CI eigenvalue problem), the CI method [27,28] itself and the CI eigenvalue problem are briefly reviewed in Section 1.6. In this section, the notation we will use in this thesis is also introduced. Then we will briefly introduce our method (the matrix multiplication scheme) for improving the performance of the current MCSCF implementations in Gaussian in Section 1.7. The reasons we developed the method in this way and where it is derived from are presented in this section too. And finally in Section 1.8, the general scope of this thesis is described.

1.2 The Hartree-Fock Self-Consistent-Field Method

As mentioned in Section 1.1, the purpose of most quantum mechanical methods is to find an approximate solution of the non-relativistic time-independent Schrödinger equation,

$$H \Psi = E \Psi, \quad (1.2.1)$$

where $\Psi$ is the wavefunction, $E$ is the energy of the system under investigation, and $H$ is the Hamiltonian operator for this $N$-electron system:

$$H = \sum_{k=1}^{N} h_k + \sum_{k \neq l}^{N} \frac{1}{r_{kl}}, \quad (1.2.2)$$

where the one electron operator, $h_k$, including the kinetic energy and the Coulomb attractions from all the nuclei in the system (in atomic units) is,

$$h_k = -\frac{1}{2} \nabla_k^2 - \sum_i Z_i \frac{1}{r_{ki}}, \quad (1.2.3)$$

with $i$ as the index of the nuclei. The term $1/r_{kl}$ in Eq. 1.2.2 is defined as the repulsions between electron $k$ and electron $l$. In practice, Eq. 1.2.1 is impossible to solve analytically, as mentioned in Section 1.1, so some suitable approximations must be introduced.

One very important approximation is the Born-Oppenheimer approximation [49]. This approximation decouples the nuclear motions from those of the electrons in a molecule and computes electronic energies for fixed nuclear positions. This is usually valid due to
the much higher mass of the nuclei. Without considering the effect of the nuclei in the HF SCF method, another approximation makes the electronic structure computations tractable: the Hartree approximation. This approximation replaces the \( N \)-electron problem with \( N \) 1-electron problems. Thus the two-electron repulsion is replaced by the interaction between one electron and the “average field” created by all of the other electrons and vice-versa.

In the HF method, for a given closed-shell molecule, the wavefunction is analysed in terms of MOs. It is convenient to expand the MOs in a set of basis wave functions (the atomic orbitals (AOs)) as,

\[
\varphi_i = \sum_{\mu=1}^{N} \psi_\mu C_{\mu i},
\]

where the set of \( N \) basis functions, \( \psi_\mu \), is called the “basis set” [50] and each has associated with it some coefficient \( C_{\mu i} \). The HF wavefunction, \( \Psi_{\text{HF}} \), is usually constructed in the form of a single Slater determinant (SD) [51], which satisfies the Pauli exclusion principle (also termed the anti-symmetry requirement: in a system of identical fermions, no two particles can occupy the same state). The general form of a Slater determinant for an \( N \)-electron system is:

\[
\Psi_{\text{SD}} = \frac{1}{\sqrt{N!}} \begin{vmatrix}
\chi_i (1) & \chi_j (1) & \cdots & \chi_k (1) \\
\chi_i (2) & \chi_j (2) & \cdots & \chi_k (2) \\
\vdots & \vdots & \ddots & \vdots \\
\chi_i (N) & \chi_j (N) & \cdots & \chi_k (N)
\end{vmatrix},
\]

where \( N \) is the total number of electrons and \( \chi \) is a spin-orbital, i.e. a product of a spatial orbital, e.g. \( \phi_i \), and an electron spin eigenfunction, e.g. \( \alpha(1) \). Note that the rows of the SD are labelled by electrons, e.g. first row 1, second row 2, etc; and the columns are labelled by spin orbitals, e.g. first column \( \chi_i \), second column \( \chi_j \), etc. Interchanging the coordinates of two electrons means interchanging two rows of the Slater determinant, which changes the sign of the determinant. Thus SD meets the requirements of the anti-symmetry principle. Having two electrons occupying the same spin orbital corresponds to having two columns of SD equal, which makes the determinant zero. Thus no more than
one electron can occupy a spin orbital (the Pauli principle). Therefore, it is convenient to introduce a shorthand notation of normalized SD that includes the normalization constant and only shows the diagonal elements of the determinants as:

$$\Psi_{SD} = \left| \chi_i \chi_j \cdots \chi_k \right|. \quad (1.2.6)$$

The MO expansion coefficients, $C_{\mu i}$, of Eq. 1.2.4 is optimised in an iterative “self-consistent field” (SCF) way that was first proposed by Hartree in 1928 [21]. Then this procedure is extended to Slater determinantal wavefunctions by Fock [22]. Recall that the Hartree approximation replaces the $N$-electron problem with $N$ 1-electron problems: the HF SCF method has replaced the many-electron Schrodinger equation with a set of effective one-electron Schrodinger equations. Therefore the associated Hamiltonian operator has been replaced by a set of one-electron (Fock) operators. To minimise the energy of Eq. 1.2.1, in the HF method the best eigenfunctions of the Fock operator must be found. The Fock operator is dependent on the molecular orbitals themselves via the two-electron repulsion term, which contains the one-electron density matrix:

$$\hat{F}(\mathbf{1}) = \hat{h}(\mathbf{1}) + \sum_{i=1}^{N} \left( 2\hat{j}_i(\mathbf{1}) - \hat{K}_i(\mathbf{1}) \right), \quad (1.2.7)$$

where the summation is over all MOs. The Coulomb operator, $\hat{J}_i(\mathbf{1})$, and exchange operator, $\hat{K}_i(\mathbf{1})$, are defined as,

$$\hat{J}_i(\mathbf{1}) f(1) = f(1) \int \frac{\varphi_i(2) \varphi_i(2)}{r_{12}} \, dr_2, \quad (1.2.8a)$$

$$\hat{K}_i(\mathbf{1}) f(1) = \varphi_i(1) \int \frac{\varphi_i(2) f(2)}{r_{12}} \, dr_2, \quad (1.2.8b)$$

where $f$ is an arbitrary function and the integrals are definite integrals over all space. When replacing $f(1)$ in terms of basis functions, $\psi_i(1)$, Eq. 1.2.8a becomes,

$$\hat{j}_i(1) \psi_i(1) = \psi_i(1) \int \frac{\varphi_i(2) \varphi_i(2)}{r_{12}} \, dr_2 = \psi_i(1) \sum_{t} \sum_{u} C_{u i} C_{u i} \int \frac{\psi_t^*(2) \psi_u(2)}{r_{12}} \, dr_2. \quad (1.2.9a)$$

Multiplying Eq.1.2.9a by $\psi_k^*(1)$ and integrating over the coordinates of electron 1 gives,
\[ \langle \psi_k(1) | \hat{J}_i(1) | \psi_i(1) \rangle = \sum_{t} \sum_{u} C_{it}^* C_{ui} \int \int \frac{\psi_k^*(1) \psi_i^*(1) \psi_i^*(2) \psi_u(2)}{r_{12}} \, dr_1 dr_2 , \]  
\[ = \sum_{t} \sum_{u} C_{it}^* C_{ui} (kl | tu) \]  
(1.2.9b)

where \( k, l, t, u \) are atomic orbital indices. Thus we know the two-electron repulsion integral is defined as,

\[ (kl | tu) = \int \int \frac{\psi_k^*(1) \psi_i^*(1) \psi_i^*(2) \psi_u(2)}{r_{12}} \, dr_1 dr_2 . \]  
(1.2.10a)

When similar operation is applied to \( \hat{K}_i(1) \), we can get,

\[ \langle \psi_k(1) | \hat{K}_i(1) | \psi_i(1) \rangle = \sum_{t} \sum_{u} C_{it}^* C_{ui} (ku | tl) . \]  
(1.2.10b)

Based on this operation, we know the Fock matrix elements can be written as,

\[ F_{kl} = \langle \psi_k(1) | \hat{F}(1) | \psi_i(1) \rangle = \langle \psi_k(1) | \hat{h}(1) | \psi_i(1) \rangle + \sum_{t} \sum_{u} \sum_{i} C_{it}^* C_{ui} \left[ 2(ku | tl) - (ku | tl) \right], \]  
(1.2.11)

where \( P_{iu} \equiv 2 \sum_i C_{it}^* C_{ui} \) are called density matrix elements. The eigenfunctions of the Fock operator (Eq. 1.2.7) must be found iteratively: a guess of the density matrix is made. This matrix usually describes the degree to which individual basis functions contribute to the \( N \)-electron wavefunction. Then the Fock matrix is constructed and diagonalized. The MOs obtained are then used to construct a new density matrix. This process is iterated until the change in the density matrix is below a certain threshold. The solution is then said to be self-consistent.

### 1.3 Electronic Correlation

As we stated in Section 1.1, the HF method is not capable of describing the energy of a chemical reaction, e.g. the binding energies or ionisation energies etc. This is because the HF theory makes the approximation that each electron moves in the static electric field created by all of the other electrons and then carries out the orbital optimization in a self-
consistent way. The interaction between two electrons is replaced by the interaction between one electron and the average field generated by all other electrons. In other words, in the HF SCF, the correlation [26] between electrons is neglected. Therefore it is possible to find two electrons, 1 and 2, which are very close to each other. However, in reality, electron 1 defines a region in space that electron 2 won’t access due to the Coulomb repulsion. In other words, in reality, the two electrons will avoid each other. Thus, to obtain a more accurate energy of the system, the electron correlation effect must be included.

The electron correlation energy is usually defined as the difference between the exact nonrelativistic energy of the system and the HF energy [52]. Electron correlation is critical for the accurate and quantitative evaluation of molecular energies [20,53,54]. Therefore, in order to get more accurate energies of systems, the correlation energy should be added to the wavefunction. However, with a single determinant, one cannot do better than the HF wavefunction. Thus one way to carry this out is to construct a wavefunction as a linear combination of multiple determinants,

\[ \Psi = c_0 \Psi_0 + c_1 \Psi_1 + c_2 \Psi_2 + \cdots, \] (1.3.1)

where the coefficients, \( c_i \), indicate the weight of each determinant in the expansion and also ensure normalization.

According to whether the determinants in Eq. 1.3.1 take similar weight because of near (or exact) degeneracy of frontier orbitals, or one determinant dominates, the electronic correlation is divided into two types: non-dynamical correlation (also known as static correlation) and dynamical correlation [55,56]. Dynamical correlation reflects the dynamical character of the electron-electron interactions. Empirically, it is observed that for most systems the first determinant of Eq. 1.3.1 dominates in the linear combination, e.g. \( c_0 \) is much larger than any other coefficients. The correlation is generated from summing up many individually small contributions of other determinants. However, in some cases, one or more coefficients of the other determinants may have similar magnitude to that for the first determinant (usually HF wavefunction) because of near (or
exact) degeneracy of frontier orbitals. This type of correlation is called non-dynamical correlation, to distinguish it from dynamical correlation.

As pointed out, the HF method is a single determinant method that does not include the electron correlation in the wavefunction. Many post-HF methods have been developed to treat the electronic correlation. For example, the single reference configuration interaction (e.g. CISD, which uses HF wavefunction as reference) methods recover dynamical correlation due to the HF reference which takes the dominant weight in Eq. 1.3.1. By contrast, the multi-reference CI (MRCI that uses MCSCF wavefunction as references) can recover both the dynamical correlation and the non-dynamical correlation due to the multi-reference taking similar weight in Eq. 1.3.1. The CASSCF method is mainly used to recover non-dynamical correlation (although it can also recover dynamical correlation by adding a large number of virtual orbitals to the active space). There are also many other methods that recover electronic correlation, such as the perturbation theory [57] (e.g. the MPn methods [58]) that treats electron correlation as a small perturbation to the Hamiltonian operator, the density functional theory (DFT) [17,18] that includes electron correlation approximately with a functional, and the coupled cluster theory [59-61]. In this thesis we will mainly investigate the MCSCF methods, as indicated in Section 1.1. We therefore will not pay too much attention to the other post-HF methods but only on CI and MCSCF methods.

1.4 Configuration

So far we have mentioned the term “configuration” many times. Since it is one fundamental concept of the CI and MCSCF methods, it is necessary to explain what a configuration is. In general, a “configuration” or “configuration state function” (CSF) refers to the molecular spin state and the occupation numbers of the orbitals. There are many ways to choose the basis functions (configurations) that are included in the CI or MCSCF wavefunctions. For example, the Slater determinants we have used so far are one choice of basis functions. However, the exact nonrelativistic wave function is an eigenfunction of both $S^2$ (the operator for the square of the magnitude of the total spin
angular momentum) and \( S_z \) (the operator for the z-component of the spin angular momentum) but the Slater determinants are usually eigenfunctions of \( S_z \) only. There are only two cases where the Slater determinants are eigenfunctions of both \( S^2 \) and \( S_z \): in *closed-shell systems* where all orbitals are doubly occupied and *high-spin systems* where all orbitals are singly occupied by electrons with the same spin [62]. For other cases, in order to be eigenfunctions of \( S^2 \), a linear combination of determinants will be required. Based on this linear combination, we can thus set up a basis of functions that are simultaneously eigenfunctions of both \( S^2 \) and \( S_z \). Such spin-adapted functions are called configuration state functions (CSFs) [63].

Based on the above description, we can see it is very convenient to choose CSFs as basis functions in the CI or MCSCF wavefunctions. This is because the principle advantage of CSFs is that their use imposes the correct spin symmetry on the approximate wavefunction. Moreover, the use of CSFs leads to a shorter expansion of the wavefunction because, for a fixed \( S_z : M \geq 0 \), the number of CSFs with a given total spin \( S = M \) is always less than or equal to the total number of Slater determinants. This feature means that CSFs were originally the more commonly-used type of basis, especially in the Unitary Group Approach (UGA) and Graphical UGA (GUGA) [64-69].

So far we have introduced two alternative sets of basis function, SDs and CSFs, from which we may proceed to calculate approximate wavefunctions. There are also other choices of basis functions, such as the Hartree-Waller (HW) functions [70-74]. This type of basis was once called Clifford algebra spinor by Paldus and Sarma [75]. In their paper, when the Clifford algebra spinor is used as basis, they showed the relationship between the generator of the unitary group \( U(n) \) and the “generator matrix elements for the totally symmetric two-box representation of \( U(2^n) \)” [75]. This effectively reduces an \( N \)-electron problem to “a number of two-boson problems”. In general, this type of basis function is, similarly to the SD basis, eigenfunction of \( S_z \) but not of \( S^2 \). However, since it contains spin functions that represent singlet basis or triplet basis (thus partly spin-
adapted), this type of basis function can lead to a shorter expansion of the wavefunction than the SD basis. For example, for a problem with 14 active electrons in 14 active orbitals, the total number of SDs is 11,768,624 while the total number of HW is 5,891,028 - about half of the number of SD. Also, since the HW function is similar to the SD basis, when using this type of basis function the algorithms that calculate approximate wavefunctions will be as simple as using the SD basis. Therefore, the partly spin-adapted HW function is also a good choice of basis function that we can use to calculate approximate wavefunctions.

Thanks to the fast development of the modern computer hardware, calculations with very large determinant spaces have been reported, e.g. Rossi et al. [76] reported a frozen core calculation for the N$_2$ molecule using almost 10$^{10}$ determinants in 1999. A calculation on the CN anion in cc-pVDZ basis performed by Thøgersen and Olsen [77] in 2004 included about 2×10$^{10}$ determinants. A series of calculations published by Gan et al. [78] in 2006 treated 4.5×10$^{10}$ determinants in some cases. Also, as mentioned above, using a determinant basis allows one to develop very high efficiency algorithms (this will be discussed in more detail later). Moreover, the current direct MCSCF implementations in Gaussian use both the SD and HW functions basis rather than the CSF basis. Therefore, we will mainly focus on the determinant based direct CI / MCSCF method in this thesis (indeed, the methods developed in both Chapter Two and Chapter Three are SD / HW function based). As we have a general idea about the configurations used in the CI / MCSCF wavefunction, we will now review the CI / MCSCF wavefunction.

1.5 The CAS/RAS-SCF Method Overview

The MCSCF methods - both CASSCF and RASSCF - are powerful tools: they give a reliable first-order descriptions of molecular excited states [79]; and have analytic energy gradients for excited-state geometry optimization and locating crossing points [35] (conical intersections [e.g.36,80-83]) between potential energy surfaces, which are important for understanding rapid radiationless deactivation of molecular excited states. There are also other methods that can calculate excitation energies, e.g. the time-
1. General Introduction

dependent DFT method (TDDFT) [84a]. However, the TDDFT is dependent on the available functionals: with “standard” functionals, it cannot describe all geometries for all excited states equally well [84b,84c]. Consequently TDDFT is less systematic at doing this than CASSCF, which can be controlled by changing the active space (the active space selection will be discussed later in this section). The excited state geometry optimizations of butadiene and hexatriene [10a] demonstrate the above point. In the following, we will start with one example to review this powerful MCSCF method.

1.5.1 One Example

As stated in Section 1.2, the HF method is a single determinant method. In this method only the MO expansion coefficients are optimized iteratively. However, it is unable to describe excited states or potential energy surfaces where more than one determinant is required to represent the wavefunction. It will be convenient to illustrate this by considering the classic example of the hydrogen molecule. For this molecule, the non-dynamic correlation effect refers to the tendency that when one electron is near the first hydrogen atom, the other tends to be near the second hydrogen [20]. This cannot be reflected by the HF method because in this method the two electrons are uncorrelated. When increasing the inter-atomic distances, the accuracy of the HF method becomes worse. An improvement to the energy of H$_2$ can be obtained by adding a second electronic configuration, $\Psi^\ast$, where both electrons occupy the anti-bonding orbital. The wavefunction, $\Psi$, can then be written as a linear combination of the two configurations,

$$\Psi = c_0 \Psi_{\text{HF}} + c_1 \Psi^\ast.$$  

(1.5.1)

The coefficients, $c_0$ and $c_1$, are determined variationally. This is an example of configuration interaction (CI), which will be introduced in more detail later. In principle, by increasing the number of configurations included in the CI expansion, the CI method is able to provide accurate approximations to the exact wavefunction. If the expansion coefficients of the configurations are optimized together with the orbitals in a variational way, this type of wavefunction is a so-called multiconfiguration self-consistent field (MCSCF) method.
1.5.2 General Theory

In MCSCF theory, the wavefunction is written as a linear combination of configurations, as indicated in Eq. 1.3.1,

$$\Psi_{MCSCF} = \sum_{K} c_{K} \Psi_{K},$$

where the coefficients $c_{K}$ reflect the weight of each configuration in the expansion and also ensure normalization. The ground state MCSCF wavefunction is obtained by minimizing the energy:

$$E_{MCSCF} = \min \left\{ \frac{\langle \Psi_{MCSCF} | H | \Psi_{MCSCF} \rangle}{\langle \Psi_{MCSCF} | \Psi_{MCSCF} \rangle} \right\}.$$  \hfill (1.5.3)

Clearly, if there is only one configuration in Eq. 1.5.2, the wavefunction becomes identical to the HF wavefunction. This model provides a highly flexible description of the electronic system because expansions in both the one-electron functions (the MOs) and the $N$-electron function (the configurations) may adapt to the physical problem. This feature makes the MCSCF wavefunction a very powerful method, as described at the beginning of this section. However, the active space selection in the MCSCF method is a critical step that determines whether the wavefunction to be built is a good one or a bad one.

1.5.3 Active Space Selection

Selecting orbitals that will be included in a MCSCF wavefunction is always the first thing to consider before investigating a chemical problem. As stated above, the accuracy of the MCSCF method depends critically on the proper selection of the active space. A review of how to select the active space for the CAS wavefunction is given by Roos [85]. There are currently many ways to select the active space. One way is to carry out a natural orbital analysis [86-88]. Thus the orbital occupations that change during a chemical process can be studied to guide the selection of the active space for a MCSCF calculation. Thus it is necessary to know what the natural orbitals are. The set of natural orbitals, introduced by Löwdin [26], provides a one-electron basis for which the CI expansion is
more rapidly convergent than it is with the HF basis, thus giving equivalent results with a smaller number of configurations. Natural orbitals are obtained by diagonalizing the spinless first-order reduced density matrix [26]. The occupation numbers ($\eta$) of the natural orbitals are not restricted to integer 2, 1 or 0. Instead, they fulfill the condition:

$$0 \leq \eta \leq 2. \quad (1.5.4)$$

The importance of natural orbitals is that they give the most rapidly convergent CI expansion. That is to obtain a given accuracy one requires fewer configurations formed from natural orbitals than configurations formed from any other orthonormal basis. It turns out that only configurations constructed from natural orbitals with occupation numbers that significantly differ from 2 or 0 make a large contribution to the energy. Thus a natural orbital with an occupation number of either very close to 2 or very close to 0 may be omitted from the CI expansion without affecting the accuracy. For example, in the azulene calculation in reference [87], after the natural orbital analysis using the unrestricted HF method [89], the occupation numbers of the 7 HOMOs and 7 LUMOs are 2.0, 2.0, 1.9566, 1.9428, 1.9265, 1.6514, 1.6500, 0.3500, 0.3486, 0.0735, 0.0572, 0.0434, 0.0, 0.0. From this analysis, we can see the occupation numbers of 10 orbitals (5 HOMOs and 5 LUMOs) are significantly different from 2 or 0. Therefore, in their CASSCF calculation [87], they set the number of the active orbitals as 10. Alternatively, one can always choose the valence MOs as the active space. For example, for aromatic systems, one usually chooses the $\pi$ orbitals as the active space.

Usually, the CASSCF active space choices are written as $(m,M)$ with $m$ indicating the number of active electrons and $M$ giving the number of active orbitals. The number of the configurations that included in the MCSCF wavefunction is a function of the active space. If we define the number of electrons with $\alpha$-spin as $N_\alpha$ and $N_\beta$ as the number of $\beta$-spin electrons ($m = N_\alpha + N_\beta$), the number of Slater determinants can be generated as,

$$N_{SD} = \binom{M}{N_\alpha} \binom{M}{N_\beta}, \quad (1.5.5)$$

where the quantities in parentheses are binominal coefficients defined as,
\[
\binom{M}{N} = \frac{M!}{N!(M-N)!}.
\]  

(1.5.6)

By contrast, if the HW functions singlet basis is chosen, the number of configurations is then given by,

\[
N_{\text{HW}}^{\text{sin}} = \binom{M}{m/2} \left[ \left( \binom{M}{m/2} + 1 \right) \right]^{2},
\]

(1.5.7)

while the number of configurations for the HW triplet basis is given as,

\[
N_{\text{HW}}^{\text{tri}} = \binom{M}{m/2} \left[ \left( \binom{M}{m/2} - 1 \right) \right]^{2}.
\]

(1.5.8)

For the azulene example mentioned above, according to Eqs. 1.5.5, 1.5.7, and 1.5.8, we can see for the CASSCF(10,10) calculation there will be in total 65,536 SDs, 31,878 HW singlet determinants and 31,626 HW triplet determinants respectively. When the active space becomes slightly larger the number of determinants will increase dramatically, e.g. when the azulene example active space changes from 10 orbitals to 12 orbitals (with 12 active electrons), the number of configurations changes to 853,776 SDs, 427,350 HW singlet determinants and 426,426 HW triplet determinants respectively. When the number of configurations becomes large, e.g. more than 1 million configurations, the CASSCF calculations will be extraordinarily demanding of resources. Therefore, the practical limit for the current CASSCF implementation in \textit{Gaussian} is up to 14 active orbitals \cite{43} with the number of active electrons similar to the number of orbitals.

However, if one wants to recover the dynamical correlation by using the CASSCF method, an active space is larger than e.g. the minimal \(\pi\) space \cite{10a, 90}. Therefore, for the azulene example, if one would like to recover the dynamical correlation, the number of active orbitals should be larger than 10, e.g. double the size of the current active space for the 10 active electrons (thus a CASSCF(10,20) calculation). But in this case, the length of the CASSCF expansion grows factorially with the number of active orbitals, e.g. for the azulene case, the number of SDs changes from 65,536 to 240,374,016. To carry out such a CASSCF calculation will not be practical. Therefore various schemes
exist to try to reduce the number of configurations in the expansion of CASSCF wavefunction. One of these schemes is to shrink the size of the CAS calculation but to allow a limited number of excitations from/to orbitals outside of the CAS space. This secondary space is then called a “restricted active space” (RAS). And the scheme is called RASSCF method [34,44].

In the RASSCF wavefunction, the active space is divided into three sub-spaces: RAS1, RAS2, and RAS3. Occupation restrictions are imposed on the RAS1 and RAS3 subspaces. The RAS1 subspace typically contains all doubly occupied MOs. A maximum number of electrons are allowed to be excited out from this subspace to form some “holes” (empty spin orbitals). The RAS3 subspace contains weakly occupied MOs with an allowed maximum number of electrons excited into it. There is no occupancy restriction imposed on the RAS2 subspace. Equally, the selection of the RASX (X= 1, 2, 3) subspaces becomes critical to the accuracy of a RASSCF calculation. In order to divide the CASSCF active space in to RASSCF subspaces, the orbitals that included in each subspace need to be chosen appropriately. The natural orbital analysis method described above is mainly used to carry out this RAS subspace selection. Still using the azulene example, among the 10 active orbitals of the CASSCF wavefunction listed above, six orbitals have more significant occupation number changes than the other orbitals (1.9265, 1.6514, 1.6500, 0.3500, 0.3486, 0.0735). The occupation numbers of the other orbitals are either close to 2 or close to 0. Thus based on this we can define the subspaces of a RASSCF calculation for this as \( M(RAS1) = 2 \), \( M(RAS2) = 6 \), and \( M(RAS3) = 2 \).

The RASSCF wavefunction can effectively reduce the number of configurations compared to the expansion for the CASSCF wavefunction having the same size of the active space as the RASSCF wavefunction. This is because while all possible configurations of electrons in the CAS space are permitted, only a limited number of RAS configurations is possible. Due to the occupation restrictions imposed on the RAS1 and RAS3 subspaces, a given maximum number of “holes” (empty spin orbitals) in RAS1 and a maximum number of electrons in RAS3 cannot be exceeded. This restriction results in a large reduction in the number of configurations. If we define \( MxHole \) as the
maximum number of “holes” in RAS1, MxElec as the maximum number of electrons in RAS3, the number of orbitals in the RAS1, RAS2, and RAS3 subspaces as \( M(RAS1) \), \( M(RAS2) \), and \( M(RAS3) \), respectively, for a RASSCF calculation with \( M = M(RAS1) + M(RAS2) + M(RAS3) \) active orbitals and \( m = N_\alpha + N_\beta \) active electrons, we can have the total number of the SD included in the RASSCF expansion as,

\[
N_{SD}^{\text{RAS}} = \sum_{i_h=0}^{M_{\text{Hole}}} \sum_{i_e=0}^{M_{\text{Elec}}} \left( \begin{array}{c}
M(RAS1) \\
M(RAS2) \\
M(RAS3)
\end{array} \right) \left( \begin{array}{c}
i_h \\
N_\alpha - M(RAS1) + i_h - i_e \\
i_e
\end{array} \right) \left( \begin{array}{c}
M(RAS2) \\
M(RAS3) \\
M(RAS3)
\end{array} \right) ,
\]

(1.5.9)

where \( i_h (i'_h) \) and \( i_e (i'_e) \) indicate the number of electrons excited out of RAS1 subspace and excited in to RAS3 subspace respectively. For the azulene example above, if we set \( M(RAS1) = 2 \), \( M(RAS2) = 6 \), \( M(RAS3) = 2 \), and \( M_{\text{Hole}} = M_{\text{Elec}} = 2 \), according to Eq. 1.5.9, the total number of Slater determinants (SDs) for the calculation of 10 active electrons in 10 active orbitals becomes 33,672, which is about half the total number of SDs for the CASSCF problem with the same size of the active space. This is a very large reduction in the number of configurations. Moreover, although the number of configurations is largely reduced, if the occupancy restrictions imposed on the RAS1 and RAS3 subspaces are chosen properly the RASSCF wavefunction has the potential to successfully approximate the CASSCF wavefunction [91]. However, although the RASSCF method can reduce the number of configurations dramatically and is capable of approximating the CASSCF wavefunction in a successful way, the introduction of the occupation restrictions in the RAS1 and RAS3 result in a more complicated implementation, which affects the performance of the RASSCF calculations compared to the CASSCF calculation with the same active space size (when the CASSCF is applicable). This is because when dividing the CASSCF active space into three subspaces with occupation restrictions in RAS1 and RAS3 subspaces, for the RASSCF wavefunction the indexing of the configurations becomes much more complicated (thus the computation of the non-zero symbolic matrix elements becomes more complicated too). Moreover, in CASSCF wavefunction there is only one type of orbital integral. But
1. General Introduction

in RASSCF calculations, due to the dividing of the three subspaces, many orbital integral types raise (among orbitals in different subspaces or in the same subspace). These increase the complication of the RASSCF implementation, resulting a slower performance of the RASSCF calculations compared to the CASSCF calculation with the same active space size. This low efficiency becomes our main motivation to carry out the work described in Chapter Three.

Another advantage of the RASSCF calculation is that it can recover the dynamical correlation energy without constructing a huge number of configurations. For example, for the above azulene case, when recovering the dynamical correlation by using the CASSCF method, the number of configurations corresponding to the CASSCF(10,20) is 240,374,016. However, when 10 virtual orbitals are added to the RAS3 subspace (so $M(RAS3)=12$) with all other sets kept unchanged ($M(RAS1)=2$, $M(RAS2)=6$, $MxHole=MxElec=2$), the total number of configurations for the corresponding RASSCF calculation, according to Eq. 1.5.9, becomes 873,652, which is far less than the number of configurations included in the corresponding CAS calculation. There are also other methods that can recover the dynamical correlation. For example, the dynamical correlation can also be included either by carrying out a subsequent step of calculation after a CASSCF calculation, e.g. adding a calculation treating dynamic correlation perturbatively, e.g. the CASPT2 method [92,93], or performing a Multi-reference CI (MRCI) calculation\(^3\) where selected configurations from the CASSCF wave function are used as reference configurations [94,95]. However, this treatment assumes the reference configurations are those obtained without taking dynamic correlation into account. For systems such as negative ions and excited states [96], that have strongly occupied orbitals that depend on dynamical correlation effects, this treatment will break down. For these cases, an extended active space maybe essential for an adequate description of the electronic structure problem, which can be achieved by using the RASSCF method. Due to these two advantages, the RASSCF method is very useful for large systems when the CASSCF calculations are too demanding, although there are not many RASSCF

\(^3\) The MRCI method cannot be carried out routinely by Gaussian. Since in this thesis we mainly focus on the development of the MCSCF methods in Gaussian, we won’t discuss the MRCI method in detail.
calculations so far in literature [e.g. 10a,45,91,97-99]. Figure 1.1 shows the orbital types of a CAS/RAS calculation.

![Diagram showing possible splitting of different orbitals in a general MCSCF formalism.](image)

**Figure 1.1 Possible splitting of different orbitals in a general MCSCF formalism.**

### 1.5.4 Full Configuration Interaction (Full CI)

Since we have so far discussed how to reduce the CI expansion of the CASSCF calculation, it is necessary to introduce another extreme case of the CASSCF method. That is if all of the orbitals of a system are selected to be the active space and all of the electrons of this system are selected to be the active electrons, then the method becomes identical to the so-called “full configuration interaction” (full CI) method. With an infinite basis set [50], the full CI method produces the exact solution to the time-independent nonrelativistic electronic Schrödinger equation. In this method, no optimization of the HF orbitals is needed, because the set of all possible configurations is
1. General Introduction

complete. However, for any but the simplest systems, it is not possible to carry out such a full CI calculation due to the unmanageably large number of configurations. However, due to the ability of the full CI to provide accurate results, full CI is mainly used as a benchmark for other, lower levels of theory, e.g. [100,101].

Based on the features of full CI, a CASSCF wavefunction may be regarded as a full CI wavefunction in a limited but variationally optimized active orbital space. As stated in Section 1.1, the performance of the current MCSCF methods in Gaussian is poor when solving the CI eigenvalue problem. In order to understand this problem clearly, it is necessary for us to review the CI theory and the CI eigenvalue problem. Therefore in the following sections, we will first briefly review the CI wavefunction, followed by the review of the CI eigenvalue problem.

1.6 Configuration Interaction (CI) Method

1.6.1 The CI Method

As shown in the last section, if we include all of the configurations that are generated from all electrons in all orbitals, we will have a full CI. But such a calculation is too demanding to accomplish. As we know that RASSCF can reduce the number of CASSCF configurations dramatically, if we truncate the CI expansion so that only a subset of the full configuration space is included, we can also reduce the number of configurations dramatically. In other words, only a subset of the configurations will be included in the wavefunction rather than all possible configurations. The truncated CI expansion should recover a large part of the correlation energy.

As indicated in the previous sections, we know that the single configurational HF method is not capable of describing systems with multiconfigurational features, e.g. the excited states, because the electron correlation effect is not included in the HF wavefunction. Several more configurations must be added to the wavefunction to recover the electronic
correlation energy. To extend the HF theory and include the electronic correlation effect in the wavefunction, it is usual to expand the wavefunction as a linear combination of a set of “excited” determinants that are formed from the ground HF determinant and the HF determinant itself,

\[
\Psi_{CI} = C_H \Psi_{HF} + \sum_a \sum_r C_{ar} \Psi_{ar} + \sum_a \sum_b \sum_r \sum_s C_{abrs} \Psi_{abrs} + \ldots = \sum_i C_i \Psi_i
\]

(1.6.1)

where \(a,b\) indicate occupied orbitals in the HF “reference” wavefunction, \(r\) and \(s\) are virtual MOs in \(\Psi_{HF}\), and the additional configurations appearing in Eq. 1.6.1 are generated by exciting an electron from the occupied orbital(s) indicated by subscripts into the virtual orbital(s) indicated by superscripts. This leads us to the single reference configuration interaction (CI) method, in which only the configuration expansion coefficients are variationally optimized. The coefficients of Eq. 1.6.1 are determined by minimizing the energy of the ground state:

\[
E_{CI} = \min_c \frac{\langle \Psi_{CI} | H | \Psi_{CI} \rangle}{\langle \Psi_{CI} | \Psi_{CI} \rangle},
\]

(1.6.2)

which is equivalent to a set of eigenvalue equations,

\[
H \Psi_{CI} = E_{CI} \Psi_{CI},
\]

(1.6.3)

where \(H\) is the Hamiltonian matrix with elements,

\[
H_{KL} = \langle \Psi_K | H | \Psi_L \rangle = \langle K | H | L \rangle.
\]

(1.6.4)

The labels \(K\) and \(L\) are used to describe arbitrary configurations. Solving Eq. 1.6.3 is equivalent to diagonalizing the Hamiltonian matrix \(H\), and permits determination of the CI coefficients associated with the energy. The matrix elements of Eq. 1.6.4 are determined by the so-called Condon-Slater rules [102], which apply to Slater determinants.

In the CI theory, only the configuration expansion is variationally optimized. The orbitals are generated separately from either a HF calculation (leads to a single reference CI) or an MCSCF calculation (leads to a multi-reference CI) and are held fixed during the optimization of the configuration expansion. In the single reference CI as described
above, the CI wavefunction is generated from the HF configuration by adding a set of “excited” configurations that form from this configuration (as shown in Eq. 1.6.1). Thus the determinants in the CI expansion related to the HF configuration are characterized as single (S), double (D), triple (T), quadruple (Q), quintuple (5) and higher excitations. Based on this, according to how many excited configurations are included in the wavefunction, the single reference truncated CI methods are expressed as: CIS, CISD, CISDT, CISDTQ, and so on (this truncation may be continued until the full CI wavefunction is recovered). Since the HF determinant is the reference, surely in Eq. 1.6.1 the weight of the HF configuration will be dominant. Therefore, according to Section 1.3, the single reference CI method recovers the dynamical correlation energy.

However, for cases where the non-dynamical correlation effect must be included, the single-reference CI calculations become unreliable. In order to overcome this and recover both the dynamical correlation effect and the non-dynamical correlation effect, the multireference CI (MRCI) wavefunction is introduced by including in the wavefunction all configurations belonging to a given reference space (usually obtained from a MCSCF calculation) as well as all excitations up to a given level from each reference configuration. In practice, the most important MRCI wavefunction is the multireference singles-and-doubles CI (MRSDCI) [94,95] wavefunction. Unfortunately, the large reference space generated from the MCSCF calculation usually makes the MRCI calculations time-consuming. This is the main bottleneck of the MRCI method. Moreover, as mentioned above, the MRCI treatment assumes the reference configurations are those obtained without taking dynamic correlation into account. For systems such as negative ions and excited states [96], that have strongly occupied orbitals that depend on dynamical correlation effects, this treatment will break down. To overcome this, the RASSCF method can be applied, which also recovers both the dynamical correlation effect and the non-dynamical correlation effect. As now we have a brief idea of the CI theory, now we should move on to the CI eigenvalue problem, which is the core calculation of both the CI method and the MCSCF methods.
1.6.2 CI Eigenvalue Problem

The CI wavefunctions are solutions to the eigenvalue problem (Eq. 1.6.3). For large CI expansions, it is impossible to construct and store the Hamiltonian matrix because of the large number of configurations. The eigenvalues of Eq. 1.6.3 can be obtained by diagonalizing the Hamiltonian matrix. However, a complete diagonalization of the Hamiltonian matrix is usually not required because only a few of its lowest eigenvalues are of interest. Selected eigenvalues can be determined by iterative methods, e.g. the methods of Lanczos [103] or Davidson [104] (a brief review of these iterative methods is given in Appendix A). In these methods, the eigenvectors and eigenvalues are instead generated by a sequence of linear transformations or contractions as,

$$\sigma = HC,$$  \hspace{1cm} (1.6.5)

where C is some trial vector and $\sigma$ the linearly transformed trial vector. However, as described above, it is not necessary to carry out a full diagonalization of the Hamiltonian matrix although nowadays we can construct and store the CI Hamiltonian matrix for certain sizes of CI problem. The major step in CI technique was taken by Roos in 1972 [105] with the introduction of the direct method that avoids the explicit construction of the Hamiltonian matrix $H$. With this, one starts with a guess vector, $C_0$, and improves it iteratively (thus a subset of eigenvalues, the $\sigma^n_K$ vector, can be updated) until convergence,

$$\sigma^n_K = \sum_L H_{KL}C_{L}^{n-1},$$  \hspace{1cm} (1.6.6)

where $C^{n-1}$ is an approximate eigenvector from the previous iteration (the first one is a guess vector), $H_{KL}$ is defined by Eq. 1.6.4, and $|K\rangle$ and $|L\rangle$ denote configurations. The Hamiltonian matrix element, $H_{KL}$, is usually written as,

$$H_{KL} = \sum_{ij} h_{ij} A^{KL}_{ij} + \frac{1}{2} \sum_{ijkl} (ij|kl) B^{KL}_{ijkl},$$  \hspace{1cm} (1.6.7)

where $h_{ij} = (i|j) = \langle i(1)|h|j(1)\rangle$ and $(ij|kl) = \langle i(1)j(1)|(1/r_{12})k(2)l(2)\rangle$ are the usual one- and two-electron repulsion integrals ($i$, $j$, $k$, and $l$ are orbital indices). The $A^{KL}_{ij}$ and
1. General Introduction

$B_{ijkl}^{KL}$ are numerical vector coupling coefficients (also termed as symbolic matrix elements) that depend on the nature of the configurations. Based on the second quantization formalism described in Appendix A, these symbolic matrix elements correspond to the creation and annihilation operators,

$$A_{ij}^{KL} = \langle K | \sum_{\xi} a_{i\xi}^\dagger a_{j\xi} | L \rangle, \quad (1.6.8a)$$

$$B_{ijkl}^{KL} = \langle K | \sum_{\xi\gamma} a_{i\xi}^\dagger a_{k\gamma}^\dagger a_{l\gamma} a_{j\xi} | L \rangle, \quad (1.6.8b)$$

where $\xi, \gamma$ denote spin ($\xi, \gamma \in \{\alpha, \beta\}$). If the SD or HW function basis is chosen, the value of these symbolic matrix elements is either 0 or ±1. The operator $a_{i\xi}^\dagger$ creates an electron in a spin orbital $|i\xi\rangle$ ($|i\xi\rangle = |i\rangle |\xi\rangle$ with $\xi \in \{\alpha, \beta\}$). Similarly, operator $a_{j\xi}$ destroys an electron from spin orbital $|j\xi\rangle$. In this thesis, the creation and annihilation operators, $a_{i\xi}^\dagger$ and $a_{j\xi}$, always occur in pairs, e.g. $a_{i\xi}^\dagger a_{j\xi}$ (there are cases that they don’t occur in pair, e.g. ionisation). Because of this feature, we can define an excitation operator as,

$$E_{ij} = E_{ij}^\alpha + E_{ij}^\beta = \sum_{\xi} a_{i\xi}^\dagger a_{j\xi}. \quad (1.6.9)$$

Therefore it is easy for us to rewrite the one-electron symbolic matrix elements, $A_{ij}^{KL}$, as,

$$A_{ij}^{KL} = \langle K | E_{ij} | L \rangle. \quad (1.6.10)$$

However, as we can see from Eq. 1.6.8b, the creation and annihilation operators do not appear as a pair. In order to use the excitation operators to express Eq. 1.6.8b, the anticommutation relationship between the creation and annihilation operators (details in appendix A) is used,

$$a_{i\xi}^\dagger a_{i\xi} a_{j\xi} = -a_{i\xi} a_{i\xi}^\dagger a_{j\xi} = -a_{i\xi}^\dagger \left(\delta_{ik} - a_{i\xi} a_{j\xi}^\dagger\right) a_{i\xi} = -\delta_{ik} a_{i\xi}^\dagger a_{i\xi} + a_{i\xi}^\dagger a_{i\xi} a_{j\xi} = E_{ij} E_{kl} - \delta_{jk} E_{il}, \quad (1.6.11)$$

where $\delta_{jk}$ is the Kronecker delta (equals to one if $j = k$ and zero otherwise). Based on Eq. 1.6.11, we can rewrite Eq. 1.6.8b as,

$$B_{ijkl}^{KL} = \langle K | E_{ij} E_{kl} - \delta_{jk} E_{il} | L \rangle = \langle K | E_{ij} E_{kl} | L \rangle - \delta_{jk} \langle K | E_{il} | L \rangle. \quad (1.6.12)$$

Applying Eqs. 1.6.10 and 1.6.12 into Eq. 1.6.6, we can get,
\[\sigma^n_K = \sum_L \sum_{ij} \langle i|j \rangle \langle K | E_{ij} | L \rangle C_L^{n-1} + \frac{1}{2} \sum_L \sum_{ijkl} \langle ij|kl \rangle \langle K | E_{ij} E_{kl} - \delta_{ik} E_{ij} | L \rangle C_L^{n-1}, \]  

where the two summations (over configuration labels \(L\) and over the integral indices \(ij\) and \(ijkl\)) update the \(\sigma\) vector directly. According to the above descriptions, now we know that in order to implement Eq. 1.6.13 in an efficient way, one needs to implement two things in an efficient way. One is to directly generate the non-zero symbolic matrix elements (Eqs. 1.6.10 and 1.6.12) in an efficient way; the other is to use these obtained non-zero elements in the CI eigenvalue iterations in an efficient way (Eq. 1.6.13). So far many different methods, e.g. [43,44,64,65,106-119], have been developed for generating the non-zero elements efficiently. Since in the current MCSCF implementations given by Klene et al [43,44] in Gaussian already generate non-zero elements very efficiently, we will directly use this efficient non-zero matrix elements generation scheme in this thesis. Nevertheless, a brief review of this scheme for CASSCF wavefunction and RASSCF wavefunction will be given in Chapters Two and Three respectively. By contrast, for updating the CI vector in an efficient way by using the obtained non-zero elements (Eq. 1.6.13), although many robust methods have been developed, e.g. [34,43,44,65,95,107,109,115,119,120], there is still space for improving the efficiency, especially when the current MCSCF implementations of Gaussian performs slowly. Therefore, to improve the CI vector updating efficiency of the current MCSCF implementations in Gaussian becomes our main motivation for carrying out the work in this thesis. In the following we will overview our method developed in this thesis.

### 1.7 Our Matrix-matrix Multiplication Method

As stated above, to develop a new efficient method to update the CI vector (Eq. 1.6.13) when the non-zero matrix elements are efficiently obtained is our main motivation for carrying out the work of this thesis. In this section we will overview the method, the matrix-matrix multiplication scheme, we have developed for improving the performance of the current MCSCF implementations in Gaussian. A brief scope of the content of this thesis will be given in the following section. However, before starting to overview our
I. General Introduction

method, two important concepts must be introduced first. One is the string concept given by Knowles and Handy [107,108], and the other is the factorizing (first suggested by Paldus in 1974 [106]) vs. unfactorizing of the $B$ symbolic matrix elements (Eq. 1.6.12). We shall start with the review of the string concept.

1.7.1 The String Concept

By definition, a string describes the occupation pattern of a number of electrons that have the same spin type in a number of otherwise empty orbitals (usually the active orbitals). For example, when $M$ active orbitals are occupied only by $N_\alpha$ $\alpha$-electrons ($N_\alpha \leq M$), each occupation pattern of these electrons in these orbitals forms a $\alpha$-string. $\beta$-strings are obtained in the similar fashion. The combinations of $\alpha$-strings with $\beta$-strings generate the corresponding Slater determinants,

$$\left| K \right> = \left| K_\alpha K_\beta \right>,$$

(1.7.1)

where $\left| K_\alpha \right>$ and $\left| K_\beta \right>$ are the $\alpha$- and $\beta$-strings, respectively. Since in a string the electrons are with the same type of spin, according to Pauli’s principle, we know each orbital will at most be occupied by only one electron. Therefore the occupation number of the orbitals in a string will be either 0 or 1. In other words, for a string that contains $N_\xi$ $\xi$-electrons, termed as $N_\xi$ ($\xi \in \{\alpha,\beta\}$), and $M$ orbitals, this string can be expressed by a binary word of length $M$, containing $N_\xi$ 1s and $(M - N_\xi)$ 0s, which is an analogue of the binary strings used for storing data in computers. Usually, when writing a string, the orbitals are written in descending order, i.e. from highest LUMO to lowest HOMO chosen for the active space. For example, for the CAS(10,10) azulene calculation described above, if the active orbitals 1,2,4,5, and 9 are occupied by the 5 $\alpha$-electrons, then the $\alpha$-string formed by this occupation pattern can be written as, 0100011011. In this thesis, we will use $\phi_{K_\xi}$ to represent the binary form of a string $K_\xi$, and use notation $b_i$ to indicate the $i^{th}$ bit of $\phi_{K_\xi}$. This binary representation leads to a very simple configuration-indexing scheme. Still using the CAS(10,10) azulene calculation as an
example, when the occupation pattern of a $\xi$-string is $\phi_{k_\xi} = 0000011111$, of which all the electrons are not excited to other orbitals (the five 0s indicate the LUMOs are empty, and the five 1s mean the HOMOs are all singly occupied), we can index this string as 1 (the ground state string). Once the electron in the highest HOMO excited into the lowest LUMO, e.g. $\phi'_{k_\xi} = 0000101111$, the string is then indexed as 2. Thus there is a one-to-one mapping between an integer and an occupation pattern of the string. Since the determinants are products of the combinations of $\alpha$- and $\beta$-strings, the indices of strings will simply lead to a one-to-one mapping between an integer and the obtained determinant (more numerical details about this will be given in the following chapters). For example, the combination of $\phi_{k_\alpha} = 0000011111$ and $\phi_{k_\beta} = 0000011111$ will generate the first configuration in the CI expansion, which is the ground state configuration. This simple way of indexing the strings and the corresponding configurations provides a way to develop a high-efficiency method for generating non-zero symbolic matrix elements directly. Indeed, the high efficient non-zero symbolic matrix elements generation methods used in the current MCSCF implementations developed by Klene et al [43,44] are string-based.

Another significant contribution of the string concept is that it replaces the summation over the Slater determinants in Eq. 1.6.13 by two summations over strings, the number of which is only the square root of the number of determinants. As pointed out above, a configuration can be split into a combination of a $\alpha$-string and a $\beta$-string (Eq.1.7.1). Therefore, taking the first term of Eq. 1.6.13 (the one-electron contribution) as example, the CI vector updating for this term can also be split into two parts according to the configuration splitting,

$$
\sigma_{k_\alpha,k_\beta}^{1} = \sum_{L_\alpha L_\beta} \sum_{i<j} (i|j) \langle K_\alpha K_\beta | E_{ij}^{\alpha} + E_{ij}^{\beta} | L_\alpha L_\beta \rangle C_{L_\alpha L_\beta}^{n-1}
$$

$$
= \sum_{L_\alpha L_\beta} \sum_{i<j} (i|j) \left[ \langle K_\alpha | E_{ij}^{\alpha} | L_\alpha \rangle \langle K_\beta | L_\beta \rangle + \langle K_\beta | E_{ij}^{\beta} | L_\beta \rangle \langle K_\alpha | L_\alpha \rangle \right] C_{L_\alpha L_\beta}^{n-1}, \quad (1.7.2)
$$

$$
= \sum_{L_\alpha} \sum_{i<j} (i|j) \langle K_\alpha | E_{ij}^{\alpha} | L_\alpha \rangle C_{L_\alpha K_\beta}^{n-1} + \sum_{L_\beta} \sum_{i<j} (i|j) \langle K_\beta | E_{ij}^{\beta} | L_\beta \rangle C_{K_\alpha L_\beta}^{n-1}
$$

52
where the superscript, 1, of $\sigma_{K\alpha,K\beta}$ means the first term of Eq. 1.6.13, the integral $\langle K_\xi | L_\xi \rangle$ equals 1 so it can interchange the label of $C_{L_\gamma K_\xi}^{n-1}$ from $L_\xi$ to $K_\xi$ ($C_{K_\xi L_\gamma}^{n-1}$).

Comparing Eq. 1.7.2 with the first term of Eq. 1.6.13 we can see the summation over the full configuration has been replaced by two summations over the corresponding strings and the length of the two summations is dramatically shorter because the number of strings usually is the square root of the number of the configurations (for the cases of $N_\alpha = N_\beta$). According to the definition of the string we know that certain $\xi$-string ($\xi \in \{\alpha,\beta\}$) is generated from distributing $N_\xi$ electrons in $M$ orbitals, therefore the number of $\xi$-strings is generated as,

$$N_{str}^{\xi} = \binom{M}{N_\xi}. \quad (1.7.3)$$

Compare Eq. 1.7.3 with Eq. 1.5.5, we know if $N_\alpha = N_\beta$, the number of the $\xi$-strings will be equal to the square root of the number of Slater determinants. Therefore this string concept makes it possible to calculate large systems that were too time-demanding for the methods without applying the string concept, because it replaces the summation over the Slater determinants by two summations over the strings, which is much shorter.

Moreover, to split the determinants into combinations of strings (Eq. 1.7.1) can also lead to highly efficient algorithms that are based on linear algebra. This is because after splitting the determinants, the CI vectors, $\sigma_{K\alpha}^{n}$ and $C_{L}^{n-1}$, can be represented as two-dimensional matrices, $\sigma^{n}(K_\alpha,K_\beta)$ and $C^{n-1}(L_\alpha,L_\beta)$, respectively. Obviously, the row and column indices are the indices of the corresponding strings. Based on this idea, we can see that Eq. 1.7.2 can be easily carried out by a matrix multiplication using basic linear algebra routines. For example, one two-dimensional matrix can be built as $X(K_\alpha,L_\alpha) = \sum_{\xi} \langle i | j \rangle \langle K_\alpha | E_{ij}^{\alpha} | L_\alpha \rangle$, which is also indexed by the indices of strings. Then a matrix multiplication between this $X$ and the CI vector matrix $C$ can be carried out as, $\sigma^{1}(K_\alpha,K_\beta) = \sum_{L_\alpha} X(K_\alpha,L_\alpha) C(L_\alpha,K_\beta)$. Highly optimized linear algebra routines, e.g. the
basic linear algebra subprograms (BLAS) [121], can thus very easily be applied for this type of calculation. Moreover, when the updating process is parallelized by using parallel machines, the columns of these matrices can also be distributed among different processors of such type of machine, and thus the matrix multiplication process is easily parallelized. Therefore, based on the above description, we know this string concept is a very powerful technique. Moreover it also allows one to develop highly efficient algorithms to solve the CI eigenvalue problem due to the matrix feature of the CI vectors. Therefore, so far many methods are string based [34,43,44,87,107-119], in which the typical methods are the ones developed by Olsen et al [34] and Klene et al [43,44]. Now we should introduce the second important concept, which is the factorization vs. non-factorization of the $B$ symbolic matrix elements (Eq. 1.6.12).

1.7.2 Factorization vs. Non-factorization Concept

As stated in Section 1.6.2, the way that Eq. 1.6.13 is implemented will directly affect the efficiency of calculations. And there are many robust methods developed to implement Eq. 1.6.13, e.g. [34,43,44,65,95,107,109,115,119,120]. By comparing all these methods, we find there are mainly two ways to carry out Eq. 1.6.13. And the difference between these two ways is whether to factorize the $B$ symbolic matrix elements (Eq. 1.6.12) or not. Since in this thesis we have developed our method using the factorization way, it is necessary to review what the factorization of the $B$ symbolic matrix elements is first.

1.7.2.1 The Factorized Methods

By summing over an intermediate configuration, $J$, between the configurations $K$ and $L$ in Eq. 1.6.12, we can get the factorized $B$ symbolic matrix elements as,

$$B_{ijkl}^{KL} = \langle K | \sum_j E_{ij} | J \rangle \langle J | E_{kl} | L \rangle - \delta_{jk} \langle K | E_{ij} | L \rangle. \tag{1.7.4}$$

This factorization idea was first suggested by Paldus in 1974 [106]. Many methods have been developed to implement Eq. 1.6.13 using this idea ever since, e.g. [34,64,65, 107,108,119,120,122] etc, except a few methods, e.g. [43,44] that employ the
unfactorized way. This idea simplifies the complicated two-electron contribution to a product of two simpler one-electron contributions. As pointed out in Section 1.7.1, the one-electron contribution can be easily converted into matrix multiplication by applying basic linear algebra routines. Thus the contribution to the CI vector updating generated by Eq. 1.7.4 can also be converted into matrix multiplications using basic linear algebra routines (this idea was first suggested by Siegbahn [120]). Now we will look into this in detail a bit more. If we apply Eq. 1.7.4 to Eq. 1.6.13, we can get,

\[
\sigma_K^n = \sum_{L} \sum_{ij} \left[ (i|j) \langle K|e_{ij}|L \rangle - \frac{1}{2} \sum_{kl} (ij|kl) \delta_{jk} \langle K|e_{ij}|L \rangle \right] C_{L}^{n-1} \\
+ \frac{1}{2} \sum_{L} \sum_{j} \sum_{ijkl} (ij|kl) \langle K|e_{ij}|J \rangle \langle J|e_{kl}|L \rangle C_{L}^{n-1}.
\]

(1.7.5)

From Eq. 1.7.5 we can see the contribution to the CI vector updating is now a combination of several products of one-electron contributions. To convert the first term of Eq. 1.7.5 into a matrix multiplication using basic linear algebra routines based on the description in Section 1.7.1 is simple. So we will mainly use the second term here as an example to demonstrate the matrix multiplication idea. Since after factorization the two-electron contribution has been converted into a product of two one-electron contributions, the basic linear algebra routines are also applicable to implement the second term of Eq. 1.7.5. This would become clear if the order of the summations of the second term in Eq. 1.7.5 is rearranged as,

\[
\sigma'_K = \sum_{j} \sum_{ij} \langle K|e_{ij}|J \rangle \sum_{kl} \sum_{L} (ij|kl) \langle J|e_{kl}|L \rangle C_{L}^{n-1}.
\]

(1.7.6)

There are many ways to construct the matrices to store the directly obtained non-zero \( \langle K|e_{ij}|J \rangle \) and \( \langle J|e_{kl}|L \rangle \) that will be used for the matrix multiplication to carry out Eq. 1.7.6. The way of building matrices depends on how the summations in Eq. 1.7.6 are organized. Different summation arrangements can lead to different matrix multiplication implementations. We will leave the discussion about this in more detail until later when our method is briefly outlined. Another advantage of using basic linear algebra routines to carry out Eq. 1.7.6 is that the elements in the CI vector, \( C_{L}^{n-1} \), can be accessed in sequential order, which also leads to good efficiency. The disadvantage of this scheme is that some extra memory is required to store the constructed matrices used for matrix
multiplication. However, as described in the last section, the number of strings is only the square root of the number of the configurations, so if we apply the string concept to Eq. 1.7.6, the memory requirement for storing the non-zero $\langle K | E_{ij} | J \rangle$ and $\langle J | E_{kl} | L \rangle$ would be reduced dramatically. For example, applying the string concept to $\langle K | E_{ij} | J \rangle$ leads to $\langle K_{\alpha} | E_{ij}^\alpha | J_{\alpha} \rangle \langle J_{\beta} | E_{kl}^\beta | K_{\beta} \rangle$, which reduces the memory requirement dramatically from the number of configuration level to the number of string level. Thus this idea is capable to be used for calculating systems with a large active space efficiently. The string expression of Eq. 1.7.6 is written as,

$$
\sigma'_{K_{\alpha}K_{\beta}} = \sum_{I_{\alpha},I_{\beta}} \sum_{ij} \langle K_{\alpha} | E_{ij}^\alpha | J_{\alpha} \rangle \langle J_{\beta} | E_{kl}^\beta | L_{\beta} \rangle C_{I_{\alpha}I_{\beta}}^{n-1} 
= \sum_{I_{\alpha}} \sum_{ij} \langle K_{\alpha} | E_{ij}^\alpha | J_{\alpha} \rangle \langle J_{\beta} | E_{kl}^\beta | L_{\beta} \rangle C_{I_{\alpha}I_{\beta}}^{n-1} 
+ \sum_{I_{\beta}} \sum_{ij} \langle K_{\alpha} | E_{ij}^\alpha | J_{\alpha} \rangle \langle J_{\beta} | E_{kl}^\beta | L_{\beta} \rangle C_{I_{\alpha}I_{\beta}}^{n-1} 
+ \sum_{I_{\alpha}} \sum_{ij} \langle K_{\alpha} | E_{ij}^\alpha | J_{\alpha} \rangle \langle J_{\beta} | E_{kl}^\beta | L_{\beta} \rangle C_{I_{\alpha}I_{\beta}}^{n-1} 
+ \sum_{I_{\beta}} \sum_{ij} \langle K_{\alpha} | E_{ij}^\alpha | J_{\alpha} \rangle \langle J_{\beta} | E_{kl}^\beta | L_{\beta} \rangle C_{I_{\alpha}I_{\beta}}^{n-1} .
$$

As mentioned above, different arrangement of the summations in these terms will lead to different ways to carry out the matrix multiplication via basic linear algebra routines. Nevertheless, as indicated in the end of Section 1.7.1, we can see after factorization the rows and columns of the matrices built for basic linear algebra routines in Eq. 1.7.7 will be labelled by the string indices. This feature will make the factorized matrix multiplication method, as stated above, suitable for calculating systems with large active space in an efficient way. In the following, we will use one term, e.g. the second term, of Eq. 1.7.7 as example to briefly outline our matrix multiplication method, and to review the methods that inspired us to develop the method in this thesis. As we have a rough idea of how the basic linear algebra routines can be applied to the factorized method, now it is necessary to review the unfactorized method in general.
1.7.2.2 The Unfactorized Methods

By contrast, for the unfactorized methods, by applying the string concept to Eq. 1.6.13, we can get (taking the two-electron contribution as example),

$$2e^2\sigma^n(K) = \frac{1}{2} \sum L \sum_{ijkl} \langle ij|kl \rangle \langle K_\alpha | a_{ia}^\dagger a_{ke}^\dagger a_{ia} a_{ja} | L_\alpha \rangle \langle K_\beta | L_\beta \rangle$$

$$+ \langle K_\beta | a_{ia}^\dagger a_{id}^\dagger a_{id} a_{ja} | L_\beta \rangle \langle K_\alpha | L_\alpha \rangle$$

$$+ \langle K_\alpha | E_{ij}^{\alpha} | L_\alpha \rangle \langle K_\beta | E_{kl}^{\beta} | L_\beta \rangle$$

$$+ \langle K_\beta | E_{ij}^{\beta} | L_\beta \rangle \langle K_\alpha | E_{kl}^{\alpha} | L_\alpha \rangle C^{n-1}(L),$$

(1.7.8)

which is the current CAS/RAS implementation in *Gaussian* [43,44]. In this type of method, the CI vector is updated directly once the non-zero matrix elements, e.g. $$\langle K_\alpha | E_{ij}^{\alpha} | L_\alpha \rangle \langle K_\beta | E_{kl}^{\beta} | L_\beta \rangle$$, are obtained. Therefore the extra memory that is required in the factorized methods for storing these elements is eliminated. However, in this type of method, the application of the highly optimized linear algebra routines would become impossible since the non-zero elements are used directly rather than stored. This is because if one wants to use basic linear algebra routines to carry out this unfactorized method, the rows and columns of the matrix built for matrix multiplication would be labeled by the indices of configurations. It would require a huge memory to store this matrix. For example, for the CASSCF(10,10) calculation using SD basis case in the previous sections, the number of configurations is 65536. If one wants to use basic linear algebra to carry out an unfactorized calculation like this, a matrix has the dimension of 65536 x 65536 (~34GBytes) will be required for this calculation. Therefore, it is impossible to use basic linear algebra routines to carry out unfactorized calculations on systems with large active space. Moreover, when using the non-zero elements directly, the indices of the configurations correspond to two sequentially obtained non-zero elements that would not necessarily be sequential, resulting in the data in the CI vector being accessed in non-sequential order (more details about the review of the current methods will be given later in the motivation section). Thus we can see that compared to the unfactorized method, the factorized scheme clearly has its advantages: that it can apply highly optimized linear algebra routines to carry out the CI vector updating via
matrix multiplication of which the dimensions are at the number of strings level. Inspired by this, we have developed our string-based matrix-matrix multiplication method to improve the performance of the current CASSCF and RASSCF implementations in *Gaussian*. In the following, we will briefly overview this matrix-matrix multiplication method. Then the reasons of why this method has been developed (the motivation) and where it is derived from (the inspiration) are discussed.

### 1.7.3 Our Matrix-matrix Multiplication Method

Since our method is developed in the factorization way, some extra memory to store the obtained non-zero elements as matrices will be required. However, as our method is a string-based method, the newly added matrices would have the dimensions at the number of string level: at most the square of the number of strings. Moreover, as 48GBytes of memory has become standard for modern commodity PCs, this extra memory requirement would not be a bottleneck for applying our method to systems with large active spaces that the current MCSCF implementations cannot afford.

As stated in Section 1.7.2.1, for carrying out Eq. 1.7.7, according to the way of organizing the summations in the equation, many different matrix multiplication algorithms can be developed. For the purpose of applying highly optimized linear algebra routines, e.g. the BLAS level 3 (deals with two-dimensional matrix-matrix multiplication) [121], and ease of parallelization, in this thesis, we have organized the summations of, e.g. the second term of Eq. 1.7.7 as,

\[
\sigma'(K_\alpha, K_\beta) = \sum_{ij} \sum_{J_\alpha} \langle K_{\alpha} | E_{ij} | J_\alpha \rangle \langle K_{\beta} | J_\beta \rangle \sum_{L_{\beta}} \sum_{kl} (ij|kl) \langle J_{\beta} | E_{kl} | L_{\beta} \rangle C_{L_{\beta}}^{|n|},
\]

(1.7.9)

where the superscript, 2, of \( \sigma'(K_\alpha, K_\beta) \) means the second term. Based on this summation organization, we can build one matrix \( X \) as,

\[
X_{ij}(J_\beta, L_{\beta}) = \sum_{kl} (ij|kl) \langle J_{\beta} | E_{kl} | L_{\beta} \rangle,
\]

(1.7.10)
of which the dimension is defined by \( N_{sr}^\beta \times N_{sr}^\alpha \) (\( N_{sr}^\beta \) is given by Eq. 1.7.3). Then a matrix multiplication between this two-dimensional matrix \( \mathbf{X} \) and the two-dimensional matrix \( \mathbf{C} \) derived from the CI vector can be carried out as,

\[
M_{ij}(J_{\beta},J_{\alpha}) = \sum_{L_{\beta}} X_{ij}(J_{\beta},L_{\beta}) \left[ C(J_{\alpha},L_{\beta}) \right]^T,
\]  
(1.7.11)

where the highly optimized BLAS level 3 routines can be applied. The dimension of this matrix product is defined by \( N_{sr}^\beta \times N_{sr}^\alpha \). Thus the \( \sigma \) vector for this \( ij \) index pair can be updated by using the obtained matrix multiplication product as,

\[
^2\sigma_{ij}' \left( K_{\alpha},K_{\beta} \right) = \sum_{J_{\alpha}} \langle K_{\alpha} | E_{ij}^\alpha | J_{\alpha} \rangle \left[ M_{ij}(K_{\beta},J_{\alpha}) \right]^T.
\]  
(1.7.12)

And by summing up all the \( \sigma_{ij} \) matrices corresponding to different \( ij \) index pairs, we can get the CI vector updated,

\[
^2\sigma'(K_{\alpha},K_{\beta}) = \sum_{ij}^2\sigma_{ij}' \left( K_{\alpha},K_{\beta} \right).
\]  
(1.7.13)

Organizing the summations in the way of Eq. 1.7.9 and carry out the matrix multiplication in the way of Eqs. 1.7.10-1.7.13 has the following advantages:

1. We can use the highly optimized linear algebra routines, e.g. the BLAS level 3 routines, to carry out Eq. 1.7.9 in our method, resulting in significant improvement of the performance of the current MCSCF implementations in Gaussian.

2. As described above, when splitting the determinants into string pairs, the CI vectors can be represented as two-dimensional matrices. When carrying out the matrix-matrix multiplication (Eq. 1.7.9), in order to take full advantage of modern parallel computers, the columns of the matrices can be distributed among different processors of these computers. Thus based on the existed parallel scheme of the current MCSCF implementation in Gaussian, a new parallel scheme, which is to parallelize the program at the matrix multiplication level, can be introduced resulting in more flexibility when using parallelism. Also, the highly optimised BLAS routines for carrying out parallelizing matrix-matrix multiplication can be used to make the performance much better.
Therefore we have every reason to expect that the method developed in this thesis will improve the performance of the current MCSCF implementations in *Gaussian* significantly, serving as the motivation for this thesis.

### 1.7.4 Motivation for this Thesis

As we briefly stated in Section 1.7.2.2, the current MCSCF implementations [43,44] in *Gaussian* [42] update the CI vector via Eq. 1.7.8, which is an unfactorized method. This type of method performs slowly as it is impossible to apply the string-based highly optimized linear algebra routines (e.g. Eq. 1.7.11) to carry out the CI vector updating. This is because the current implementations use the obtained non-zero elements directly rather than store them, which require huge extra memory to do. But the advantage of this unfactorized method is that it can eliminate this huge extra memory requirement for storing the obtained non-zero elements in an array of which the dimension is at the number of configurations level rather than at the number of strings level.

Moreover, apart from not being able to use string based basic linear algebra routines to carry out CI vector updating, the direct using of the non-zero elements to update the CI vector will cause the current MCSCF implementations in *Gaussian* to perform poorly too. The reason for this, as stated in Section 1.7.2.2, is when using the obtained non-zero elements directly, the indices of the configurations corresponding to two sequentially obtained non-zero symbolic matrix elements are usually not necessarily sequentially ordered, resulting in the CI vector itself being accessed frequently. Since non-local data access (means the main computer memory will be accessed frequently) will be slow, the frequent access of the CI vector would restrict the performance of the current method at a significant level. We will use one example to illustrate this.

For example, for a CASSCF(12,12) calculation that contains 853776 determinants, all the elements in the CI coefficients vector, $C^{\text{n-1}}_L$, are indexed according to the indices of the configurations, e.g. from 1 to 853776. Now, assume for a certain orbital index $\{i,j,k,l\}$
1. General Introduction

set in Eq. 1.7.8, the first directly obtained non-zero symbolic matrix element corresponds to a pair of configurations having the indices of $K_1 = 18256$ and $L_1 = 1554$ respectively. This means the $18256^{th}$ element in the $\sigma_K^n$ vector is updated by the $1554^{th}$ element in the $C_L^{n-1}$ vector (Eq. 1.7.8). Then the next directly obtained non-zero symbolic matrix element yields a pair of configurations corresponding to the same orbital index set that has the indices of $K_2 = 110656$ and $L_2 = 55146$ respectively, which means the $110656^{th}$ element in the $\sigma_K^n$ vector is updated by the $55146^{th}$ element in the $C_L^{n-1}$ vector. Obviously, after updating the element in $\sigma_K^n$ at position $K_1$ by using the element in $C_L^{n-1}$ at position $L_1$, the CI vectors, $\sigma_K^n$ and $C_L^{n-1}$, have to be accessed again for the newly required data. This will also affect the performance of the current MCSCF implementations. Figure 1.2 illustrates the CI vector updating process of this type of method in a general way.

By contrast, from Section 1.7.3, we can see the matrix multiplication method we have developed can achieve both the use of the string based highly optimized linear algebra routines to carry out the CI vector updating and the use of data in the CI vectors in a sequential order. Therefore, we know the method developed in this thesis can improve the performance of the current MCSCF implementations dramatically. However, as indicated in Section 1.7.2.1, the method developed in this thesis is inspired by several previously-developed factorized matrix multiplication methods, e.g. [34,107,108,119, 120], so it is necessary to briefly review the key concept of these methods. That is, how they organized the summations order in Eq. 1.7.7 to carry out the matrix multiplication.

1.7.5 Inspiration for this Thesis

In this section we will review some typical methods that apply basic linear algebra in the process of CI vector updating. As stated in Section 1.7.2.1, these methods serve as the inspiration for our method developed in this thesis. The reason why we develop our method in the way shown in Section 1.7.3 rather than directly using these methods is also given after the review.
\[ \sigma^n_K = \left( \begin{array}{cccc} K_1 & K_2 & \cdots & K_n \end{array} \right) \]

\[ \sigma^n_K = \sum_L H_{KL} C^{n-1}_L \]

Directly obtained Hamiltonian matrix elements:

- first: \( H_{K_1L_1} \)
- second: \( H_{K_2L_2} \)
- third: \( H_{K_3L_3} \)

\[ C^{n-1}_L = \left( \begin{array}{cccc} L_1 & L_2 & \cdots & L_n \end{array} \right) \]

Figure 1.2 A brief visualized CI vector updating process of the current CAS/RAS implementations in Gaussian. From this figure we can see each time when a non-zero matrix element is obtained, the elements in the CI vectors are used directly for updating. But these elements in the CI vectors are not usually sequentially ordered for different non-zero Hamiltonian matrix elements. The gap between these elements will cause a frequent CI vector access, which will affect the calculation performance significantly.

1.7.5.1 Siegbahn-Knowles-Handy Method

Siegbahn [120] is the first one that introduced basic linear algebra algorithms for matrix multiplication in implementing Eq. 1.6.13. However, in his method, the CSFs basis was used. In order to present this method in the string-based fashion, the method developed by Knowles and Handy [107,108], which is an extension of the Siegbahn’s method from CSFs basis to Slater determinant basis (thus the string based Eq. 1.7.7 is implemented), should be reviewed instead.

In the Knowles and Handy method [107,108], the summations of, i.e. the second term of Eq. 1.7.7 are organized in the following way,

\[ 2 \sigma^i(K_{\alpha}, K_{\beta}) = \sum_{ij} \sum_{K_{\alpha}} \left| E_{ij}^{\alpha} \right| J_{\alpha} \left( K_{\beta} \right) J_{\beta} \sum_{kl} \langle ij | kl \rangle \sum_{L_{\alpha}} \left| E_{kl}^{\alpha} \right| L_{\alpha} \right) C^{n-1}_{J_{\alpha}L_{\beta}}. \quad (1.7.14) \]

Based on this summation arrangement, one matrix \( X \) can be built as,

\[ ij X_{kl}(J_{\beta}, J_{\alpha}) = \sum_{L_{\beta}} \left| E_{kl}^{\beta} \right| L_{\beta} \right) C^{n-1}_{J_{\alpha}L_{\beta}}, \quad (1.7.15) \]
1. General Introduction

which has a dimension defined by the number of \( kl \) orbital index pairs (depends on the values of \( ij \) index pair but maximums as \( \frac{M(M+1)}{2} \), \( M \) as the number of active orbitals) times \( N_{\beta \sigma}^\beta \times N_{\alpha \sigma}^\alpha \cdot \frac{M(M+1)}{2} \times N_{\alpha \sigma}^\beta \times N_{\beta \sigma}^\alpha \). Therefore, if one presents this matrix in string-based way, Eq. 1.7.15 gives a three-dimensional matrix. Then a matrix multiplication is carried out as,

\[
M_{ij}(J_{\beta}, J_{\alpha}) = \sum_{kl} (ij|kl) \left[ \langle J_{\beta}, J_{\alpha} | X_{kl} \rangle \right] = (IX)_{ij}, \tag{1.7.16}
\]

where \( I_{ij} \) is a vector that stores all the orbital integrals corresponding to one \( ij \) index pair, and matrix \( M_{ij} \) is a two-dimensional matrix with the dimension as \( N_{\beta \sigma}^\beta \times N_{\alpha \sigma}^\alpha \). This step “can be formulated as a matrix multiplication” [107] via applying basic linear algebra routines. Then based on the obtained matrix \( M_{ij} \), the CI vector that corresponding to this \( ij \) index pair can be updated as given by Eq. 1.7.12. By summing up this product for all \( ij \) orbital index pairs, the second term of Eq. 1.7.7 (Eq. 1.7.14) can be updated, which is given in Eq. 1.7.13.

Comparing this method with our method described in Section 1.7.3, we can see the processes of generating the matrix \( X \) (Eq. 1.7.10 vs. Eq. 1.7.15) and the process of the matrix multiplication (Eq. 1.7.11 vs. Eq. 1.7.16) are quite different. The reason for not using the matrix multiplication method suggested by Siegbahn-Knowles-Handy [107, 108,120] is because the matrix \( X \) generated here will be a three-dimensional matrix, which will take a much larger dimension than the matrix \( X \) used in our method. Thus it will require much more memory space to store the matrix \( X \). Nevertheless, this pioneering idea of using basic linear algebra algorithms to carry out CI vector updating indeed inspired us to develop our method.
1.7.5.2 Olsen-Roos Method

Another very important typical matrix multiplication method should be reviewed is the method developed by Olsen et al [34]. In their method, the summations of, e.g. the second term is organized in the way as,

\[
2 \sigma' (K_\alpha, K_\beta) = \sum_{ij} \sum_{\ell \beta} \sum_{kl} (ij|kl) \langle J_\beta | E_{ij}^\beta | L_\beta \rangle \sum_{J_\alpha} \langle K_\alpha | E_{ij}^\alpha | J_\alpha \rangle \langle K_\beta | J_\beta \rangle C_{J_\alpha \ell \beta}^{n-1} \\
= \sum_{ij} \sum_{\ell \beta} \sum_{kl} (ij|kl) \langle K_\beta | E_{ij}^\beta | L_\beta \rangle \sum_{J_\alpha} \langle K_\alpha | E_{ij}^\alpha | J_\alpha \rangle C_{J_\alpha \ell \beta}^{n-1} . \tag{1.7.17}
\]

This arrangement of the summations leads to two matrices generation as,

\[
C'_{ij} (K_\alpha, L_\beta) = \sum_{J_\alpha} \langle K_\alpha | E_{ij}^\alpha | J_\alpha \rangle C_{J_\alpha \ell \beta}^{n-1}, \tag{1.7.18}
\]

and

\[
X_{ij} (K_\beta, L_\beta) = \sum_{kl} (ij|kl) \langle K_\beta | E_{ij}^\beta | L_\beta \rangle . \tag{1.7.19}
\]

We can see the generation of the matrix X is the same as in our method (Eq. 1.7.10 vs. Eq. 1.7.19). But instead of carrying out the matrix multiplication between X and the C matrix derived from the CI vector as indicated in Eq. 1.7.10, the matrix multiplication in Olsen’s method is carried out between X and the C’ matrix to generate the \( \sigma \) vector for this ij index pair directly as,

\[
2 \sigma'_{ij} (K_\alpha, K_\beta) = \sum_{\ell \beta} C'_{ij} (K_\alpha, L_\beta) \left[ X_{ij} (K_\beta, L_\beta) \right]^T . \tag{1.7.20}
\]

The highly optimized linear algebra routines can be used in this matrix multiplication process. And the \( \sigma \) vector for the second term of Eq. 1.7.7 is updated by summing over all the \( 2 \sigma'_{ij} (K_\alpha, K_\beta) \) for all the ij index pairs as given in Eq. 1.7.13.

This is a very elegant matrix multiplication method that has been used for method development ever since, e.g. the methods developed by Pulay et al [87] and Rolik et al [119]. However, the reason that we have not used this way to arrange the summations in Eq. 1.7.7 in our method is as follows. For a certain ij index pair, the number of non-zero \( \langle K_\alpha | E_{ij}^\alpha | J_\alpha \rangle \) is very limited. Indeed, from Chapter Two we will know this number will be at most half of the number of the corresponding strings. Therefore, the matrix C'
generated from Eq. 1.7.18 will not be dense, although the matrix $C_{J_{a}L_{g}}^{\pi-1}$, which is the CI vector matrix, is dense. Moreover, from Chapter Two we will also know that the matrix $X$ generated from Eq. 1.7.10 or Eq. 1.7.19 will be a sparse matrix. Therefore, the matrix multiplication between $X$ and $C'$ will be between two sparse matrices. It won’t be very efficient to use highly optimized linear algebra routines to carry out a matrix multiplication between two sparse matrices (because it would be unavoidable to deal with some zeros). However, since the matrix elements in $X$ are generated from all $kl$ index pairs, the number of non-zero elements will certainly much larger than the number of the corresponding strings. Thus the matrix multiplication between $X$ and the CI vector matrix $C$ (Eq. 1.7.11) would produce a dense matrix $M$. Then the efficiency of updating the $\sigma$ vector by using this dense matrix should be good. Considering this reason, we have developed our method as shown in Section 1.7.3, although the Olsen’s method does inspire us about the matrix $X$ generation.

In the following two chapters we will implement the matrix multiplication concept described in Section 1.7.3 to the CASSCF method and RASSCF method respectively. However, it is necessary to give a scope of this thesis before we start the following chapters.

1.8 Scope of This Thesis

In this thesis we will focus on improving the performance of the current MCSCF implementations in Gaussian described above. In Chapter Two, the matrix multiplication scheme described in Section 1.7.3 is implemented to increase the efficiency of the current CASSCF method. This is programmed as a part of the Gaussian development program package. In order to take full advantage of modern parallel computers, this implementation is parallelized too; supporting both shared memory and distributed memory architectures. The performance of the new method has been demonstrated by carrying out a CASSCF(14,14) calculation ($\sim 10^7$ determinants) on the system of pyracylene [45] by using both the current CASSCF implementation in Gaussian and the
new method. Another CASSCF(16,16) calculation (~ $2 \times 10^8$ determinants) on the pyrene [46] system is also carried out to illustrate that the new method can provide a much better performance than the current implementation. In order to demonstrate the new method is good for “real world” problems, e.g. geometry optimization, the ground state geometry optimization of two hydrocarbon cations (anthracene$^+$ / phenanthrene$^+$) [47] is carried out. The result shows that our method saves a lot time compared to the current method.

Chapter Three describes applying the matrix multiplication scheme given in Section 1.7.3 to improve the performance of the current RASSCF implementation in Gaussian. Since the introduction of the occupation restrictions in the RAS1 and RAS3 subspaces, many different integral types (among orbitals in different subspaces or in the same subspace) arise. Some of these integral types are not only the time-consuming cases, but they can also be suitably modified using the factorized matrix multiplication scheme described in Section 1.7.3. By implementing the matrix multiplication algorithm to these cases, the method developed in Chapter Three can already perform significantly faster than the current RASSCF method in Gaussian. Similar to the CASSCF case, the new implementation is also parallelized for modern parallel computers. The performance of this new RASSCF method is demonstrated by using coronene [48] as the test system. For this system, the active electrons are set as 24 always, while the number of orbital in the 3 subspaces varies as well as the value of the MxHole and MxElec. The result shows that the new method can always perform better than the current implementation. For the rest of the integral cases, it is proved that using the unfactorized way for implementation is the best choice. An attempted modification to these integral cases of the RASSCF method using the unfactorized way is implemented in Chapter Four. The timing result of the same coronene calculations used in Chapter 3 shows this attempt at modification can further improve the performance of the new RASSCF method.
Chapter 2

A New Matrix-Matrix Multiplication Approach to the Direct Full Configuration Interaction (Full CI) / Complete Active Space Self-Consistent-Field (CASSCF) Method and Its Parallel Implementation
2.1 Introduction

In order to study the excited-state reactivity, minima and transition structures etc. of molecules, the complete active space self-consistent-field (CASSCF) method [33], which is a computational electronic structure method, is widely used [35-41]. As described in the last chapter (Section 1.5.3), in this method a set of active molecular orbitals (MOs) is selected, of which the occupancy will change significantly during the process of investigation. The occupation number of all other orbitals is kept frozen, e.g. either 2 or 0. Apart from the electrons that are in the inactive orbitals, all other electrons are defined as active electrons. And each occupation pattern of these active electrons in the active orbitals forms a configuration in the CI expansion (e.g. one term in Eq. 1.5.1). In other words, a full configuration interaction (FCI) calculation is carried out in the range of these active orbitals, named the active CI space, to obtain the CI eigenvector at each iteration of the iterative multi-configuration SCF (MCSCF [29-32]) process. During this calculation process, the Hamiltonian matrix elements are usually recomputed when they are needed in the eigensolver iteration. Methods of this type are called the direct methods [105].

As indicated in Section 1.6.2, the key step in the direct methods is to solve the CI eigenvalue problem (Eq. 1.6.13) in an iterative way efficiently. In order to achieve this, two things must be considered. One is to efficiently generate the non-zero Hamiltonian matrix elements directly and the other is to efficiently evaluate the CI eigenvector from the obtained matrix elements. As stated in Section 1.6.2, since the non-zero matrix elements generation scheme in the current CASSCF implementation [43] in Gaussian [42] is already efficient enough, we will use this scheme in our method directly. However, also according to Section 1.6.2, there is still space for improving the efficiency of the process of evaluating the CI vector from the obtained non-zero matrix elements. In Section 1.7, we have identified that the current CASSCF implementation [43] in Gaussian [42] performs poorly. Also in Section 1.7, we briefly outlined our matrix-matrix multiplication approach, which uses highly optimized linear algebra routines for updating the CI vector, to improve the performance of the current CASSCF method.
significantly. In this chapter, we will describe this approach in more detail and to implement it for the CASSCF wavefunction. However, the efficient non-zero matrix elements generation scheme of the current CASSCF method should be reviewed first (briefly here, more detail will be given in the following sections).

In Klene’s method [43], after applying the string concept [107,108] to the determinants, we can get, taking the one-electron symbolic matrix elements as an example,

\[ \langle K | E_{ij} | L \rangle = \langle K_{\alpha} | E_{ij}^\alpha | L_{\alpha} \rangle \langle K_{\beta} | L_{\beta} \rangle + \langle K_{\beta} | E_{ij}^\beta | L_{\beta} \rangle \langle K_{\alpha} | L_{\alpha} \rangle. \] (2.1.1)

According to Chapter 1 and Appendix A we know that the excitation operator, \( E_{ij}^\xi \), annihilates an electron from orbital \( j \) of the initial string, \( |L_{\xi}\rangle \), and generates the final string, \( |K_{\xi}\rangle \), by creating an electron on orbital \( i \) of this string. Therefore, in order to have non-zero \( \langle K_{\xi} | E_{ij}^\xi | L_{\xi} \rangle \), before the excitation operator operates, the occupation number of orbitals \( i \) and \( j \) in string \( |L_{\xi}\rangle \) should be 0 and 1 and the occupation number of these orbitals in string \( |K_{\xi}\rangle \) should be 1 and 0, respectively. The occupation pattern of all other orbitals must be identical between strings \( |K_{\xi}\rangle \) and \( |L_{\xi}\rangle \). Therefore, a list of strings having all possible occupation patterns of \( N_{\xi} - 1 \) electrons (the reduced one electron is used for the annihilation and creation) in \( M - 2 \) orbitals (with orbitals \( i \) and \( j \) are reduced) should be generated. This list is called the reduced string list, of which the number of the active orbitals and the number of active electrons are reduced. According to Eq. 1.7.3, the length of the reduced string list (the number of reduced strings) can be defined as (for the \( i \neq j \) cases),

\[ L_{N_{\xi}-1}^{M-2} = \binom{M - 2}{N_{\xi} - 1}. \] (2.1.2)

Comparing Eq. 2.1.2 with Eq. 1.7.3, we can see the length of the reduced string list is much shorter than the length of the full string list. More conveniently, all the reduced string lists are only needed to be generated once before the iterative calculation of CI vector evaluation starts (called pre-computed). These pre-computed reduced string lists
are stored in memory. Then during the CI vector evaluation process, based on the elements in the corresponding lists, one can generate the non-zero symbolic matrix elements as well as the indices of the corresponding strings dynamically. Now we shall briefly review how this works.

The non-zero $\langle K_\xi E_\xi^i|L_\xi \rangle$ is obtained dynamically by applying the so-called “bits operation” on the elements in the reduced string list according to the given orbital index set. When a pair of orbital index is given, e.g. $i$ and $j$, in order to get non-zero $\langle K_\xi E_\xi^i|L_\xi \rangle$, one only needs to insert the occupation number 1 and 0 (the “bits”) to the $i^{th}$ and $j^{th}$ positions and the occupation number 0 and 1 to the $i^{th}$ and $j^{th}$ positions to each element in the reduced list to obtain a pair of full strings, $K_\xi$ and $L_\xi$. For example, if the given orbital index pair is $i=4$ and $j=1$, and the first element in the reduced list is (written in binary form) $\phi_\xi = 000111$ ($\phi_\xi$ denotes a binary reduced string), then by inserting the occupation number 1 and 0 to the 4$^{th}$ and 1$^{st}$ positions of this reduced string, we can get the binary form of $K_\xi$ as $\phi_{K_\xi} = 0001110$ (the inserted bits are bold). And by inserting the occupation number 0 and 1 to the 4$^{th}$ and the 1$^{st}$ positions of the reduced string, one can get the binary form of $L_\xi$ as $\phi_{L_\xi} = 00010111$. Therefore, at the same time one gets the non-zero $\langle K_\xi E_\xi^i|L_\xi \rangle$ directly, the indices of the corresponding string pairs are also obtained, which can be used to locate the element position in the CI vectors as in the current CASSCF implementation or to index the matrices will be used in the matrix multiplication method developed in this chapter.

Now we should move to how to implement the matrix multiplication approach. However, before we proceed to the detail of our method itself, we should remind ourselves why the current CASSCF method performs slowly.

The CI eigenvector evaluation process (Eq. 1.6.13) is written as,
2. Algorithm for the Direct Full CI / CASCCF Method

\[ \sigma_K^n = \sum_{L} \sum_{ij} (i|j) \langle K | E_{ij} | L \rangle C_{L}^{n-1} + \frac{1}{2} \sum_{L} \sum_{ijkl} (ij|kl) \langle K | E_{ij} E_{kl} - \delta_{jk} \delta_{il} | L \rangle C_{L}^{n-1}, \]  

(2.1.3)

where the two-particle symbolic matrix elements are given by,

\[ B_{ijkl}^{KL} = \langle \sigma_K^n | (K_{E_{ij} E_{kl}} - \delta_{jk} \delta_{il}) | C_{L}^{n-1} \rangle. \]

(2.1.4)

In the current CASSCF implementation, after applying the string concept \([107,108]\) to the Slater determinants, the complicated two-electron contribution (the second term of Eq. 2.1.3) is written as,

\[ 2e \sigma_K^n = \sum_{L} \sum_{ijkl} (ij|kl) \left( \langle K_{\alpha} | a_{ia}^+ a_{ia}^+ | L_{\alpha} \rangle \langle K_{\beta} | L_{\beta} \rangle + \langle K_{\beta} | a_{ja}^+ a_{ja}^+ | L_{\alpha} \rangle \langle K_{\alpha} | L_{\beta} \rangle \right) \]

+ \left( \langle K_{\alpha} | E_{ij}^\alpha | L_{\alpha} \rangle \langle K_{\beta} | E_{kl}^\beta | L_{\beta} \rangle + \langle K_{\beta} | E_{ij}^\beta | L_{\beta} \rangle \langle K_{\alpha} | E_{kl}^\alpha | L_{\alpha} \rangle \right) C_{L}^{n-1}(L). \]

(2.1.5)

In this method, the non-zero element, e.g. \( \langle K_{\alpha} | E_{ij}^\alpha | L_{\alpha} \rangle \langle K_{\beta} | E_{kl}^\beta | L_{\beta} \rangle \), is used directly to update the CI vector once it is obtained, e.g. the \( K_{\alpha} K_{\beta} \) \( \text{th} \) element in \( 2e \sigma_K^n (K) \) is updated by the \( L_{\alpha} L_{\beta} \) \( \text{th} \) element in \( C_{L}^{n-1}(L) \). This operation doesn’t require memory to store the obtained non-zero elements. However, as stated in Section 1.7, due to these non-zero elements are not stored, the basic linear algebra routines will not be possible to be used in implementing Eq. 2.1.5, because if storing these elements, the memory requirement would be at the number of configurations level rather than at the number of strings level.

Moreover, although the reduced list method is very efficient to obtain the non-zero symbolic matrix elements and to locate the elements in the CI vectors, the indices of the full strings that are obtained directly will not be predictable. This is due to the “bits operation” applied to the elements in the reduced string lists will lead to non-sequentially ordered full strings. For example, suppose we have three sequentially ordered reduced strings, written in binary as, 000111, 001011, and 010011. The given orbital index pair is \( i=4 \) and \( j=2 \). By carrying out the “bits operation” to these three reduced strings, we will have three pairs of full strings as \{00011101, 0001011\}, \{00101101, 0010011\}, and \{01001101, 0100011\}. Obviously, we can see the indices of these three string pairs are not sequentially ordered (more details about how to index strings like this will be
reviewed later). These non-sequentially ordered string indices would result large gap between the indices of the configurations. Therefore the CI vector has to be accessed frequently resulting in the current CASSCF method in Gaussian performing poorly.

According to Section 1.7 in Chapter 1, we know that the current CASSCF method in Gaussian belongs to the so-called unfactorized method, which doesn’t factorize the two-particle symbolic matrix elements (Eq. 2.1.4) by inserting an intermediate configuration, \( J \), between the configurations \( K \) and \( L \),

\[
B_{ijk\ell}^{KL} = \sum_j \langle K | E_{ij} | J \rangle \langle J | E_{kl} | L \rangle - \delta_{j\ell} \langle K | E_{ij} | L \rangle. \tag{2.1.6}
\]

Also from Section 1.7 in Chapter 1, we know that the factorized method can implement Eq. 2.1.3 in a way of matrix multiplication using string based basic linear algebra routines, which are very efficient. This is because in Eq. 2.1.6 the complicated two-electron contribution has been simplified to a product of two one-electron contributions. If we apply Eq. 2.1.6 to Eq. 2.1.3,

\[
\sigma_k^s = \sum_{i,j} \sum_{L} \left[ (i|j) \langle K | E_{ij} | L \rangle - \frac{1}{2} \sum_{kl} (ij|kl) \delta_{jk} \langle K | E_{iL} | L \rangle \right] C_{L}^{n-1}, \tag{2.1.7}
\]

\[
+ \frac{1}{2} \sum_{i,j} \sum_{kl} (ij|kl) \langle K | E_{ij} | J \rangle \langle J | E_{kl} | L \rangle C_{L}^{n-1}
\]

then simply by reorganizing the orders of the summations of the second term of Eq. 2.1.7, matrix multiplication scheme can be applied. This would be clear if we first reorganize the summations of the second term of Eq. 2.1.7 as (Eq. 1.7.6),

\[
\sigma_k^s = \sum_{i,j} \sum_{L} \langle K | E_{ij} | J \rangle \sum_{kl} (ij|kl) \langle J | E_{kl} | L \rangle C_{L}^{n-1}, \tag{2.1.8}
\]

and then applying the string concept [107,108] to the determinants of Eq. 2.1.8 as,

\[
\sigma_{k_s,k_p}^{ij} = \sum_{J_s,J_p} \sum_{I_s,I_p} \langle K_a | E_{ij} | J_a \rangle \langle J_a | E_{kl} | L_a \rangle \sum_{l_s,l_p} (ij|kl) \langle J_s | E_{i}^{l_s} | I_s \rangle \delta_{l_s,l_p} C_{l_s,l_p}^{n-1}
\]

\[
+ \sum_{J_s,J_p} \sum_{I_s,I_p} \langle K_a | E_{ij} | J_a \rangle \langle J_a | E_{kl} | L_a \rangle \sum_{l_s,l_p} (ij|kl) \langle J_s | E_{i}^{l_s} | I_s \rangle C_{l_s,l_p}^{n-1}
\]

\[
+ \sum_{J_s,J_p} \sum_{I_s,I_p} \langle K_b | E_{ij} | J_b \rangle \langle J_b | E_{kl} | L_b \rangle \sum_{l_s,l_p} (ij|kl) \langle J_s | E_{i}^{l_s} | I_s \rangle C_{l_s,l_p}^{n-1}
\]

\[
+ \sum_{J_s,J_p} \sum_{I_s,I_p} \langle K_b | E_{ij} | J_b \rangle \langle J_b | E_{kl} | L_b \rangle \sum_{l_s,l_p} (ij|kl) \langle J_s | E_{i}^{l_s} | I_s \rangle C_{l_s,l_p}^{n-1}
\]

72
Then we can see each term of Eq. 2.1.9 can be easily implemented in the way of matrix multiplication using basic linear algebra. The method developed by Olsen et al [34] is a typical method of this type. In the following we will use the second term of Eq. 2.1.9 to illustrate how the Olsen’s method implements this into matrix multiplication. Then we will also use the second term of Eq. 2.1.9 as example to introduce our method, which organizes the summations in the term in a slightly different way from Olsen’s method.

In Olsen’s method, the order of the summations of the second term of Eq. 2.1.9 is organized as,

\[
2 \sigma'^\prime(K_\alpha, K_\beta) = \sum_{ij} \sum_{kl}(ij|kl) \langle J_\beta | E_{ij}^\beta | L_\beta \rangle \sum_{J_\alpha} \langle K_\alpha | E_{ij}^\alpha | J_\alpha \rangle \langle K_\beta | J_\beta \rangle C_{J_\alpha L_\beta}^{n-1}.
\]

(2.1.10)

The matrix multiplication of this method to update the CI vector for this term is implemented as,

1. Matrix Generation

\[
C_i^j(K_\alpha, L_\beta) = \sum_{J_\alpha} \langle K_\alpha | E_{ij}^\alpha | J_\alpha \rangle C_{J_\alpha L_\beta}^{n-1},
\]

(2.1.11a)

\[
X_i^j(K_\beta, L_\beta) = \sum_{kl}(ij|kl) \langle K_\beta | E_{ij}^\beta | L_\beta \rangle,
\]

(2.1.11b)

2. Matrix Multiplication

\[
2 \sigma'^\prime(K_\alpha, K_\beta) = \sum_{L_\beta} C_i^j(K_\alpha, L_\beta) [X_i^j(K_\beta, L_\beta)]^T,
\]

(2.1.11c)

3. CI Vector Updating

\[
2 \sigma^\prime(K_\alpha, K_\beta) = \sum_j 2 \sigma'^\prime(K_\alpha, K_\beta).
\]

(2.1.11d)

We can see the matrix multiplication (Eq. 2.1.11c) is carried out between two two-dimensional matrices. Therefore, highly optimized linear algebra routines, e.g. the BLAS level 3 routines [121] that are very efficient, can be used.

However, due to the reason described towards the end of Section 1.7.5, we will organize the order of the summations of the second term in a different way from Olsen’s method, which is the key concept of our matrix multiplication method,
\[ \sigma'(K_\alpha,K_\beta) = \sum_{ij} \sum_{\alpha} \langle K_\alpha | E_\alpha^\alpha | J_\alpha \rangle \langle J_\alpha | K_\beta \rangle \sum_{il} \sum_{\beta} \langle ij|kl \rangle \langle J_\beta | E_\beta^\beta | L_\beta \rangle C_{\alpha \beta}^{-1}. \] (2.1.12)

And the matrix multiplication method for this kind of summation arrangement is implemented as,

1. **Matrix Generation**
   \[ X_0(J_\beta,L_\beta) = \sum_{il} \langle ij|kl \rangle \langle J_\beta | E_\beta^\beta | L_\beta \rangle, \] (2.1.13a)

2. **Matrix Multiplication**
   \[ M_0(J_\beta,J_\alpha) = \sum_{L_a} X_{lj} \langle J_\beta | E_\beta^\beta | L_\beta \rangle [C(J_\alpha,L_\beta)]^T, \] (2.1.13b)

3. **CI Vector Updating**
   \[ \sigma'(K_\alpha,K_\beta) = \sum_{\alpha} \langle K_\alpha | E_\alpha^\alpha | J_\alpha \rangle [M_\alpha(J_\beta,J_\alpha)]^T, \] (2.1.13c)

\[ \sigma'(K_\alpha,K_\beta) = \sum_{\beta} \sigma'_0(K_\alpha,K_\beta). \] (2.1.13d)

Comparing Eqs. 2.1.10 and 2.1.11 with Eqs. 2.1.12 and 2.1.13, we can see that our arrangement of the order of the summations of the second term of Eq. 2.1.9 is different from the arrangement of Olsen’s method. One can refer to Section 1.7 in Chapter 1 for the reason this occurs. In our matrix multiplication step (Eq. 2.1.13b), the highly optimized BLAS level 3 routines can also be used. Moreover, by carrying out the CI vector updating in this way, based on the existing parallel scheme in the current CASSCF implementation in Gaussian, a new parallel scheme that is to parallelize on the matrix multiplication level can be introduced too, thus adding more flexibility for parallelism. Because of these features of the new method, we can predict that the new method will perform much better than the current CASSCF implementation in Gaussian. In the rest of this chapter, we will present the details of how this new method is implemented and a number of chemical systems that are used to demonstrate that the new method indeed performs much faster than the current method. Moreover, in order to demonstrate that the new method is capable to be used in “real world” problems, e.g. geometry optimization, the ground state geometry optimization of two hydrocarbon cations (anthracene$^+$ / phenanthrene$^+$) [47] are carried out, and the result is discussed towards the end of this chapter.
This chapter is structured as: Section 2.2 overviews the general theory that our method will use. Then the algorithms of our method are mathematically described in Section 2.3. The numerical values of our new method are discussed in Section 2.4. Section 2.5 introduces the implementation details of this new algorithm. The parallelism scheme of our method is given in Section 2.6. And in Section 2.7 the new method is verified and the performance of this new method is tested using two relatively large systems: pyracylene [45] (CAS(14,14) calculations, $\sim 10^7$ determinants) and pyrene [46] (CAS(16,16) calculations, $\sim 2 \times 10^8$ determinants). The data of the calculations on the two cations described above are also discussed in this section. Finally, a brief summary of this chapter is issued in Section 2.8.

### 2.2 Theory

As described in Chapter 1, large-scale configuration interaction (CI) calculations require the computation of only a few eigenvalues and eigenvectors of large, real-symmetric matrices: to solve the CI eigenvalue problem in an iterative way,

$$\sigma^a_k = \sum_L H_{KL} C^{a-1}_L,$$  \hspace{1cm} (2.2.1)

where $C^{a-1}_L$ is an approximate eigenvector from the previous iteration (the first one is a guess vector), $H_{KL}$ is Hamiltonian matrix element, and $|K\rangle$ and $|L\rangle$ denote configurations. In second quantization (Appendix A), $H_{KL}$ is represented as,

$$H_{KL} = \sum_i h_{ij} \langle K | E_{ij} | L \rangle + \frac{1}{2} \sum_{ijkl} (ij|kl) \langle K | E_{ij} E_{kl} - \delta_{jk} E_{il} | L \rangle.$$  \hspace{1cm} (2.2.2)

As defined in the last chapter, $h_{ij}$ and $(ij|kl)$ are the usual one- and two-electron repulsion integrals ($i$, $j$, $k$, and $l$ are orbital indices). The $A_{ij}^{KL} = \langle K | E_{ij} | L \rangle$ and $B_{ijkl}^{KL} = \langle K | E_{ij} E_{kl} - \delta_{jk} E_{il} | L \rangle$ are numerical vector coupling coefficients (also termed as symbolic matrix elements) that depend on the nature of the configurations. The excitation operator, e.g. $E_{ij}$, is given by $E_{ij} = E_{ij}^\alpha + E_{ij}^\beta = \sum_\xi a^{\dagger}_{ij\xi} a_{j\xi}$. Apply Eq. 2.1.6 to Eq. 2.2.2, we can get,
We find that the two-electron contribution is actually a combination of a bilinear\(^1\) contribution (\(E_{ij}E_{kl}\) part in the second term of Eq. 2.2.3) and a linear contribution (\(\delta_{jk}E_{il}\) part in the second term of Eq. 2.2.3). Because the 1-electron contribution is a linear contribution, in our method, we will use linear and bilinear contributions to the CI vector updating to express Eq. 2.2.1. Thus we define linear contributions by using the coefficients as \(A_{ij}^{KL} = \langle K|E_{ij}|L\rangle\) and bilinear contributions by using the coefficients as \(B_{ij}^{KJ} = \langle K|E_{ij}|J\rangle\). Based on this definition, we can easily rewrite the CI vector construction procedure as the combination of linear contribution and bilinear contribution:

\[
\sigma_k = \sum_L H_{KL}C_L^{n-1} = \sum_L \left( \sum_{ij} A_{ij}^{KL} (i|j) - \frac{1}{2} \sum_{ijkl} \delta_{jk} A_{ij}^{KL} (ij|kl) \right) C_L^{n-1} + \frac{1}{2} \sum_j \sum_{Lijkl} (ij|kl) B_{ij}^{KJ} B_{ij}^{KL} C_L^{n-1}. \quad (2.2.4)
\]

The first term in Eq. 2.2.4 gives all the linear contributions to the CI vector updating and the second term indicates the bilinear contributions. In the following section we will describe the algorithms for the bilinear and linear contribution separately.

### 2.3 Algorithm for Matrix-Matrix Multiplication Method

In this section, the implementation of Eq. 2.2.4 described in Section 2.2 will be discussed. From Eq. 2.2.4 we know the contribution to the CI vector updating is from both linear and bilinear parts. Since so far we have mainly used the bilinear contribution as example to introduce our method (both in Section 1.7.3 and Section 2.1), we will start from the algorithm for the bilinear contribution first.

---

\(^1\) If we consider the excitation operator as a variable \(x\) in algebra, then the first part of the second term in Eq. 2.2.3 will yield an \(x^2\). Thus it is an analogue of bilinear term. Similarly, we know the first term of Eq. 2.2.3 and the second part of the second term of Eq. 2.2.3 are both linear terms.
2.3.1 General Algorithm for Bilinear Contribution

The bilinear contribution to the CI vector updating, separated from Eq. 2.2.4, is,

\[ \sigma'_K = \frac{1}{2} \sum_j \sum_{i,j|kl} (ij|kl) B_{ij}^{KJ} B_{kl}^{IL} C_L . \]  

(2.3.1)

In Eq. 2.3.1, since \( B_{ij}^{KJ} \) does not operate on both the configurations \( L \) and the orbital index pair \( kl \), we can move it out of the \( L \) and \( kl \) summations as:

\[ \sigma'_K = \frac{1}{2} \sum_j \sum_{i,j} B_{ij}^{KJ} \sum_L \sum_{kl} (ij|kl) B_{kl}^{IL} C_L . \]  

(2.3.2)

By splitting the determinants into \( \alpha \) - and \( \beta \)-strings [107,108], we can rewrite Eq. 2.3.2 in a spin-based form as:

\[ \sigma'_{K_a K_b} = \frac{1}{2} \sum_{J_{a \beta}} \sum_{i,j} B_{ij}^{K_a K_b J_{a \beta}} \sum_{L_{a \beta}} \sum_{kl} (ij|kl) B_{kl}^{J_{a \beta} L_{a \beta}} C_{L_{a \beta}} . \]  

(2.3.3)

To make Eq. 2.3.3 easy to calculate, we need to arrange Eq. 2.3.3 into a matrix-matrix multiplication form as:

\[ X_{ij}^{J_{a \beta}} = \sum_{kl} (ij|kl) \cdot B_{kl}^{J_{a \beta} L_{a \beta}} C_{L_{a \beta}} , \]  

(2.3.4a)

\[ M_{ij}^{J_{a \beta}} = \sum_{L_{a \beta}} X_{ij}^{J_{a \beta}} C_{L_{a \beta}} , \]  

(2.3.4b)

\[ \sigma'_{K_a K_b} = \sum_{J_{a \beta}} \sum_{i,j} B_{ij}^{K_a K_b J_{a \beta}} M_{ij}^{J_{a \beta}} , \]  

(2.3.4c)

where Eq. 2.3.4a is a matrix generation process, and Eq. 2.3.4b is the matrix-matrix multiplication step. From the definition of the bilinear coupling coefficient, the coefficients in Eqs. 2.3.4a and 2.3.4c can be obtained as:

\[ B_{kl}^{J_{a \beta} L_{a \beta}} = \left \langle J_{a \beta} L_{a \beta} \bigg| E_{kl} \bigg| L_{a \beta} \right \rangle , \]

\[ B_{ij}^{K_a K_b J_{a \beta}} = \left \langle K_a K_b \bigg| E_{ij} \bigg| J_{a \beta} \right \rangle . \]  

(2.3.5)

Applying \( E_{ij} = E_{ij}^\alpha + E_{ij}^\beta \), Eq. 2.3.5 can be rewritten as:

\[ B_{kl}^{J_{a \beta} L_{a \beta}} = \left \langle J_{a \beta} L_{a \beta} \bigg| E_{ij}^\alpha + E_{ij}^\beta \bigg| L_{a \beta} \right \rangle = \left \langle J_{a \beta} L_{a \beta} \bigg| E_{ij}^\alpha \bigg| L_{a \beta} \right \rangle + \left \langle J_{a \beta} L_{a \beta} \bigg| E_{ij}^\beta \bigg| L_{a \beta} \right \rangle , \]

\[ B_{ij}^{K_a K_b J_{a \beta}} = \left \langle K_a K_b \bigg| E_{ij}^\alpha + E_{ij}^\beta \bigg| J_{a \beta} \right \rangle = \left \langle K_a K_b \bigg| E_{ij}^\alpha \bigg| J_{a \beta} \right \rangle + \left \langle K_a K_b \bigg| E_{ij}^\beta \bigg| J_{a \beta} \right \rangle , \]  

(2.3.6)
where the operators, e.g. $E_{ij}^\alpha$, only act on $\alpha$-strings, leaving $\beta$-strings unaltered, and vice versa. Therefore applying Eq. 2.3.6 to Eqs. 2.3.4, we can get:

\[
\langle J_\gamma | L_\gamma \rangle \cdot X_{ij}^{J_\gamma} = \langle J_\gamma | L_\gamma \rangle \cdot \sum_{kl} (ij|kl) \cdot \langle J_\xi | E_{ij}^\xi | L_\xi \rangle , \tag{2.3.7a}
\]

\[
M_{ij}^{J_\alpha J_\beta} = \sum_{L_\alpha} X_{ij}^{J_\alpha} C_{L_\alpha J_\beta} + \left[ \sum_{L_\beta} X_{ij}^{J_\beta} \left( C_{J_\beta L_\beta} \right)^T \right], \tag{2.3.7b}
\]

\[
\sigma_{ij}^{\alpha} (K_\alpha, K_\beta) = \sum_{ij} \sum_{L_\alpha} \langle K_\alpha | E_{ij}^\alpha | J_\alpha \rangle \langle K_\beta | J_\beta \rangle M_{ij}^{J_\alpha J_\beta} + \sum_{ij} \sum_{L_\beta} \langle K_\beta | E_{ij}^\beta | J_\beta \rangle \langle K_\alpha | J_\alpha \rangle M_{ij}^{J_\alpha J_\beta} = \sum_{ij} \sum_{L_\alpha} \langle K_\alpha | E_{ij}^\alpha | J_\alpha \rangle M_{ij}^{J_\alpha J_\beta} + \sum_{ij} \sum_{L_\beta} \langle K_\beta | E_{ij}^\beta | J_\beta \rangle M_{ij}^{J_\alpha J_\beta}. \tag{2.3.7c}
\]

Note that the term $\langle J_\beta | L_\beta \rangle$ has changed the column index of matrix $C(L_\alpha, L_\beta)$ from $L_\beta$ to $J_\beta$. This is because the integral, $\langle J_\beta | L_\beta \rangle$, is 1. The changing of the row or column indices of other matrices is because of the same reason. Also, we notice that the generation of the intermediate matrix $M$ requires two matrix multiplications. This is because the matrix $C$ when using Slater determinant basis is not symmetrical ($C_{L_\alpha L_\beta} \neq \left( C_{L_\alpha L_\beta} \right)^T$). Now we shall move to the algorithm for the linear contribution to the CI vector updating.

### 2.3.2 General Algorithm for Linear Contribution

In Eq. 2.2.4, the first term gives the linear contribution to the CI vector updating:

\[
\sigma_K^{lin} = \sum_{L} \left( \sum_{ij} E_{ij}^{KL} (ij) - \frac{1}{2} \sum_{ijkl} \delta_{ik} E_{ij}^{KL} (ij|kl) \right) C_L. \tag{2.3.8}
\]

Since the orbital indices $ij$ are the outermost loop for both terms in Eq. 2.3.8, we can take the summation out and rewrite Eq. 2.3.8 as:

\[
\sigma_K^{lin} = \sum_{L,ij} \left[ E_{ij}^{KL} (ij) - \frac{1}{2} \sum_{kl} \delta_{ik} E_{ij}^{KL} (ij|kl) \right] C_L. \tag{2.3.9}
\]
2. Algorithm for the Direct Full CI / CASSCF Method

According to Shavitt [65], if we apply the symmetry properties to the two-electron integrals then there will be 11 integral classes \( I_{ijkl} \) and the symmetry properties will give rise to the addition of a factor \( f(I_{ijkl}) \) due to possible permutations of orbital indices (Table 2.1). The factor can be evaluated as:

\[
f(I_{ijkl}) = 4 \times \left( \frac{1}{2} \right)^{\delta_{ij}} \times \left( \frac{1}{2} \right)^{\delta_{kl}},
\]

(2.3.10)

with the restrictions:

\[
i \geq j, \ k \geq l, \ ij \geq kl
\]

with \((ij) = \frac{i(i-1)}{2} + j; \)

\((kl) = \frac{k(k-1)}{2} + l\)

Based on the Coefficient column given in Table 2.1, we can successfully eliminate the \( \delta \) function in Eq. 2.3.9 and thus can make it easy to implement.

Similar to the process of generating Eqs. 2.3.7, the spin form of Eq. 2.3.9 can be obtained as:

\[
\sigma_{K_{\alpha}}^{lin} = \sum_{I_{\alpha}, j} \left[ \langle K_{\alpha} | E_{ij}^{\alpha} | L_{\alpha} \rangle (ij) - \frac{1}{2} \sum_{kl} \delta_{jk} \langle K_{\alpha} | E_{il}^{\alpha} | L_{\alpha} \rangle (ij|kl) \right] C_{I_{\alpha} K_{\beta}},
\]

(2.3.12)

where the \( \delta \) function can be avoided as given in Table 2.1. To calculate this contribution, we can also use the matrix-matrix multiplication method. Similar to the bilinear cases, we can rearrange Eq. 2.3.12 into a matrix multiplication form as,

\[
X_{K_{\alpha}, I_{\alpha}}^{lin} = \sum_{ij} \left[ \langle K_{\alpha} | E_{ij}^{\alpha} | L_{\alpha} \rangle (ij) - \frac{1}{2} \sum_{kl} \delta_{jk} \langle K_{\alpha} | E_{il}^{\alpha} | L_{\alpha} \rangle (ij|kl) \right] \langle K_{\beta} | L_{\beta} \rangle,
\]

(2.3.13a)
\[
\sigma_{\alpha l} = \sum_{l_{\alpha}} \left( X_{K\alpha} + X_{K\beta} \right) C_{L\alpha} + \left[ \sum_{l_{\beta}} X_{K\beta} \left( C_{L\alpha} \right)^T \right].
\]

In Eq. 2.3.13a, it is clear that the matrix \( X \) generation is summed over all the orbital indices, which means the matrix multiplication only needs to be carried out once per iteration. In addition, in the linear contribution, the matrix multiplication step (Eq. 2.3.13b) is already the CI vector-updating step, which dramatically increases the calculation efficiency.

Table 2.1 The 11 integral classes constructed under the conditioning index loop given in Eq. 2.3.11. Column \( f(I_{ijkl}) \) is the factor corresponding to the integral classes \( I_{ijkl} \), and the Coefficient column gives the value of \( E_i E_k - \delta_{ij} E_{il} \) corresponding to the particular integral class. In this table the coefficients indicate that in the cases of integral classes 1, 3, 5, 7, 8, 9, and 10, there are linear contributions, which can be calculated together with the one-electron linear contributions.

| \( I_{ijkl} \) | \( f(I_{ijkl}) \) | Integral (ij|kl) | Coefficient \( E_i E_k - \delta_{ij} E_{il} \) |
|---------------|----------------|----------------|------------------------------------------|
| 1             | \( \frac{1}{2} \) | (ii|ii)         | \( E_i E_i - E_{ij} \)                  |
| 2             | 1              | (ii|kk)         | \( E_i E_k \)                           |
| 3             | 2              | (ij|jj)         | \( (E_{ij} + E_{ji}) E_{jj} - E_{ij} \) |
| 4             | 2              | (ij|kk)         | \( (E_{ij} + E_{ji}) E_{kk} \)          |
| 5             | 2              | (ii|il)         | \( E_{ij} (E_{il} + E_{il}) - E_{il} \) |
| 6             | 2              | (ii|kl)         | \( E_{ij} (E_{kl} + E_{ik}) \)          |
| 7             | 1              | (ij|ij)         | \( (E_{ij} + E_{ji}) (E_{ij} + E_{ji}) - E_{ij} - E_{ij} \) |
| 8             | 2              | (il|kl)         | \( (E_{il} + E_{ij}) (E_{il} + E_{ij}) - E_{ik} \) |
| 9             | 2              | (ik|kl)         | \( (E_{ik} + E_{il}) (E_{ik} + E_{il}) - E_{il} \) |
| 10            | 2              | (ij|il)         | \( (E_{ij} + E_{ji}) (E_{il} + E_{ij}) - E_{il} \) |
| 11            | 2              | (ij|kl)         | \( (E_{ij} + E_{ji}) (E_{kl} + E_{ik}) \) |

80
2.3.3 Algorithm for Hartree-Waller Functions Basis

So far, we have introduced the general theory and algorithm of our matrix multiplication method based on the Slater determinants basis. If the configurations are chosen to be spin-adapted Hartree-Waller functions \[70-74\], the process will be similar but much simpler. This is because, for the singlet state \((S=0)\) system with \(2N\) electrons, we have \(N = N_\alpha = N_\beta\) and therefore all string lists are the same for \(\alpha\)- and \(\beta\)-strings, hence we can have the relation,

\[
\begin{aligned}
|K_\alpha K_\beta\rangle &= |K_\beta K_\alpha\rangle,
\end{aligned}
\]  

(2.3.14)

which is a symmetrical relation that is not applicable for the Slater determinant cases. We only need to use one spin type strings, e.g. \(\alpha\)-strings, to carry out the CI vector updating. Thus the linear contribution (Eq. 2.3.13b) for a singlet contribution becomes:

\[
\sigma_{lin}^{HW}(K_\alpha, K_\beta) = \sum_{L} X_{K_\alpha L \alpha}^{lin} C_{L, K_\beta}^{HW},
\]  

(2.3.15)

where the matrix \(X_{K_\alpha L \alpha}^{lin}\) is the same matrix for the Slater determinant case, but the matrix \(C_{L, K_\beta}^{HW}\) generated from the CI vector will be different. The superscript of \(\sigma_{lin}^{HW}\) and \(C_{L, K_\beta}^{HW}\) indicates the calculation use Hartree-Waller functions as basis. And the bilinear contribution (Eqs. 2.3.7) becomes:

\[
\begin{aligned}
X_{ij}^{L_\alpha L_\beta} &= \sum_{kl} (ij|kl) \cdot \langle J_\beta | E_{kl}^\beta | L_\beta \rangle, \\
M_{ij}^{L_\alpha L_\beta} &= \sum_{L_\beta} X_{ij}^{L_\alpha L_\beta} \left( C_{L_\beta}^{HW} \right)^T, \\
\sigma_{HW}'(K_\alpha, K_\beta) &= \sum_{ij} \sum_{L_\alpha} \langle K_\alpha | E_{ij}^{\alpha} | J_\alpha \rangle \langle K_\beta | J_\beta \rangle \left( M_{ij}^{L_\alpha L_\beta} \right)^T \\
&= \sum_{ij} \sum_{L_\alpha} \langle K_\alpha | E_{ij}^{\alpha} | J_\alpha \rangle \left( M_{ij}^{L_\alpha K_\beta} \right)^T.
\end{aligned}
\]  

(2.3.16a, 2.3.16b, 2.3.16c)

From Eqs. 2.3.15 and 2.3.16, we can see, the linear contribution for the singlet state case can still be carried out once per iteration, while the intermediate matrix \(M\) (Eq. 2.3.16b) is still regenerated for every new \(ij\) index for the bilinear contribution. But the operation of the matrix multiplication is halved compared to Eq. 2.3.7b where the matrix \(M\) is a product of two matrix multiplications. This is due to when the HW function basis is used,
we will have $C_{L_\alpha L_\beta}^{\text{HW}} = \left(C_{L_\alpha L_\beta}^{\text{HW}}\right)^T$. As we have introduced the general algorithm for the matrix-matrix multiplication method, now we should introduce the practical details regarding the implementation of the algorithms described in this section, starting from the numerical values that are used in this method.

2.4 The Numerical Values

2.4.1 The Addresses of the Configurations

So far we have the theoretical idea of how the matrix multiplication method will work. However, in order to practically implement this idea, we need to know how the determinants and the corresponding strings are indexed in the calculation. As stated in Chapter 1, the string concept given by Knowles and Handy [107,108] simplifies the string and configuration indexing. Therefore, we will use the method that addressed by them to index the strings and configurations. An addressing (or indexing) array, $Z$, is defined, separately for $\alpha$- and $\beta$-strings, as:

$$Z(k,l) = \sum_{m=M-l+1}^{M-k} \left[ \begin{array}{c} m \\ N_\xi - k \end{array} - \begin{array}{c} m-1 \\ N_\xi - k-1 \end{array} \right],$$

$$\left( M-N_\xi + k \geq l \geq k; k < N_\xi \right),$$

$$Z(N_\xi,l) = l - N_\xi, \quad \left( M \geq l \geq N_\xi \right),$$

(2.4.1)

where $k$ refers to an electron, $l$ is an orbital, $M$ denotes the number of orbitals, and $N_\xi$ is the number of electrons of spin type $\xi$ ($\xi \in \{\alpha, \beta\}$). The quantities in parentheses are binomial coefficients, defined as $\binom{m}{n} = \frac{m!}{n!(m-n)!}$ (Eq. 1.5.6). So the address (or index) of any string, identified by a list of occupied orbitals in strictly ascending order, e.g. an $\alpha$ string, is given by:

$$\text{Addr}\{K_\alpha\} = 1 + \sum_{k=1}^{N_\alpha} Z(k,l(k)).$$

(2.4.2)
2. Algorithm for the Direct Full CI / CASSCF Method

This gives the addresses in a sequential order based on Eq. 2.4.1. The combination of the addresses of the \( \alpha \) - and \( \beta \) -strings gives an address of a Slater determinant:

\[ \text{Addr} \{ K \} = (\text{Addr} \{ K_\alpha \} - 1) \times \binom{M}{N_\beta} + \text{Addr} \{ K_\beta \}. \quad (2.4.3) \]

Note that this addressing scheme is not limited to Slater determinants. If the HW function basis is chosen, Eq. 2.4.3 can be rewritten, taking HW singlet functions as an example,

\[ \text{Addr} \{ K \} = \frac{\text{Addr} \{ K_\alpha \} (\text{Addr} \{ K_\alpha \} - 1)}{2} + \text{Addr} \{ K_\beta \}. \quad (2.4.4) \]

One example would make this addressing scheme clear. Assuming we have a \( \alpha \) -string, \( \phi_{K_\alpha} = 111000 \), of a system of \( M = 6 \), \( N_\alpha = 3 \), of which the 4\(^{\text{th}}\), 5\(^{\text{th}}\), and 6\(^{\text{th}}\) orbitals are occupied by the 3 \( \alpha \) -electrons, the summation of Eq. 2.4.2 goes from 1 to 3. Since the first electron occupies the fourth orbital, second occupies the fifth, and the third in the sixth orbital, the calculation in Eq. 2.4.1 comes out with the pairs of \( (k = 1, l = 4) \), \( (k = 2, l = 5) \), and \( (k = 3, l = 6) \). Thus we can obtain the address of this string as (from Eq. 2.4.2): \( \text{Add} \{111000\} = 1 + 10 + 6 + 3 = 20 \). Table 2.2 gives the addresses of the full string list of this example system.

2.4.2 Reduced Lists Concept Review

As described in Section 2.1, we will adopt the reduced string list concept developed by Klene \textit{et al} [43] to generate the non-zero matrix elements. A brief review of this concept is given already in Section 2.1. In this section we will review this concept in more detail with some examples. For the purpose of simplicity, we will use the linear coupling coefficient defined in Section 2.2,

\[ A_{ij}^{KL} = \langle K | E_{ij} | L \rangle = \langle K | E_{ij}^\alpha + E_{ij}^\beta | L \rangle, \quad (2.4.5) \]

as an example to illustrate this concept. To calculate all the non-zero values corresponding to certain \( ij \) orbital index pair of Eq. 2.4.5 efficiently, the method of reduced string lists [43] must be used. Based on the string splitting concept [107,108], we can rewrite Eq. 2.4.5 as,
Table 2.2. The addresses and the binary representation ($\phi_{K_{\xi}}$) of the $\xi$-strings ($\xi \in \{\alpha, \beta\}$) for a system with $M = 6$ and $N_{\xi} = 3$.

<table>
<thead>
<tr>
<th>Add ${K_{\sigma}}$</th>
<th>$\phi_{K_{\xi}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000111</td>
</tr>
<tr>
<td>2</td>
<td>001011</td>
</tr>
<tr>
<td>3</td>
<td>010011</td>
</tr>
<tr>
<td>4</td>
<td>100011</td>
</tr>
<tr>
<td>5</td>
<td>001101</td>
</tr>
<tr>
<td>6</td>
<td>010101</td>
</tr>
<tr>
<td>7</td>
<td>100101</td>
</tr>
<tr>
<td>8</td>
<td>011001</td>
</tr>
<tr>
<td>9</td>
<td>101001</td>
</tr>
<tr>
<td>10</td>
<td>110001</td>
</tr>
<tr>
<td>11</td>
<td>001110</td>
</tr>
<tr>
<td>12</td>
<td>010110</td>
</tr>
<tr>
<td>13</td>
<td>100110</td>
</tr>
<tr>
<td>14</td>
<td>011010</td>
</tr>
<tr>
<td>15</td>
<td>101010</td>
</tr>
<tr>
<td>16</td>
<td>110010</td>
</tr>
<tr>
<td>17</td>
<td>011100</td>
</tr>
<tr>
<td>18</td>
<td>101100</td>
</tr>
<tr>
<td>19</td>
<td>110100</td>
</tr>
<tr>
<td>20</td>
<td>111000</td>
</tr>
</tbody>
</table>

$$A_{ij}^{KL} = \langle K | E_{ij}^\alpha + E_{ij}^\beta | L \rangle$$
$$= \langle K_{\alpha} | E_{ij}^\alpha | L_{\alpha} \rangle \langle K_{\beta} | L_{\beta} \rangle + \langle K_{\beta} | E_{ij}^\beta | L_{\beta} \rangle \langle K_{\alpha} | L_{\alpha} \rangle \quad (2.4.6)$$

Taking the first term on the right hand side of Eq. 2.4.6 as an example, assuming the current $ij$ index pair is $\{i = 4, j = 3\}$, we can thus consider the element:

$$\langle K_{\alpha} | E_{ij}^\alpha | L_{\alpha} \rangle = \langle K_{\alpha} | a_{4\alpha}^\dagger a_{3\alpha} | L_{\alpha} \rangle \neq 0 \quad (2.4.7)$$

This means one electron has been annihilated from the $3^{rd}$ $\alpha$-spin orbital and created at the $4^{th}$ $\alpha$-spin orbital, which was virtual (empty) before the operation. Thus for a non-zero contribution, a set of the string pairs $\{K_{\alpha}, L_{\alpha}\}$ (for $\phi_{K_{\alpha}}$, we must have bits $b_4 = 1, b_3 = 0$, and for $\phi_{L_{\alpha}}$, it must be $b_4 = 0, b_3 = 1$) must be found. The set of all the final strings, $K_{\xi}$, and initial strings, $L_{\xi}$, for a certain orbital index pair of $ij$ that makes Eq. 2.4.7 be non-zero is called an excitation list, $\chi_{ij}^{\xi}$ (where the subscript denotes the orbital where a $\xi$-spin electron is annihilated, and the super script indicates the orbital where the
2. Algorithm for the Direct Full CI / CASSCF Method

electron is created). The full excitation list for the single excitation can thus be calculated in a shorter list, which is called the reduced list, of \( N_{\xi} - 1 \) electrons and \( M - 2 \) orbitals (\( i \neq j \) cases, written as \( L_{N_{\xi}-1}^{M-2} \)). Based on this, in our example, the reduced string list is thus generated from arranging 2 electrons in 4 orbitals freely, which reduces the length of the string list from 20 (as given in Table 2.2) to 6 \( \left( \begin{array}{c} 4 \\ 2 \end{array} \right) = 6 \). Then the relevant strings \( K_\alpha \) and \( L_\alpha \) of the excitation list \( \chi_{3\alpha}^{4\alpha} \) can be generated by inserting the known bits \( b_4 \) and \( b_3 \) into the 6 elements of the reduced string list (as shown in Figure 2.1). Table 2.3 lists the elements of the reduced string list, and the obtained string pairs after inserting the bits \( b_4 \) and \( b_3 \) (shown in bold) into the elements of this reduced string list. The crucial observation is that the same reduced list \( L_{N_{\xi}-1}^{M-2} \) can be used for any index pair \( i \neq j \) to generate all the excitation lists \( \chi_{i\xi}^{j\xi} \). Similarly, one can easily find out all the single excitation lists for the case of \( i = j \) (\( \chi_{i\xi}^{i\xi} \)) from the reduced list \( L_{N_{\xi}-1}^{M-1} \), and all double excitation lists, \( \chi_{i\xi j\xi}^{i\xi j\xi} \), from the reduced lists \( L_{N_{\xi}-2}^{M-2}, L_{N_{\xi}-3}^{M-3} \), and \( L_{N_{\xi}-4}^{M-4} \) for different orbital index sets. Table 2.4 shows some examples of the lengths-comparison of the full string list and the reduced string lists.

\[ \begin{array}{c}
\text{Reduced strings} \\
\hline
0011 \\
0101 \\
1001 \\
0110 \\
1010 \\
1100
\end{array} \]

Figure 2.1 The bits-operation to the reduced list in order to generate the corresponding full strings of the excitation list (using the system of 6 active electrons acting in the space of 6 active orbitals).
Table 2.3 Generating the excitation list in the case of $i \neq j$ by the $E_{43}^{\alpha}$ operation.

<table>
<thead>
<tr>
<th>$\phi_{\alpha}$</th>
<th>$\phi_{K_{\alpha}}$</th>
<th>Add${K_{\alpha}}$</th>
<th>$\phi_{L_{\alpha}}$</th>
<th>Add${L_{\alpha}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>001011</td>
<td>2</td>
<td>000111</td>
<td>1</td>
</tr>
<tr>
<td>0101</td>
<td>01001</td>
<td>8</td>
<td>010101</td>
<td>6</td>
</tr>
<tr>
<td>1001</td>
<td>101001</td>
<td>9</td>
<td>100101</td>
<td>7</td>
</tr>
<tr>
<td>0110</td>
<td>011010</td>
<td>14</td>
<td>010110</td>
<td>12</td>
</tr>
<tr>
<td>1010</td>
<td>100110</td>
<td>15</td>
<td>1000110</td>
<td>13</td>
</tr>
<tr>
<td>1100</td>
<td>110000</td>
<td>20</td>
<td>110000</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 2.4 Comparison of the lengths of the full string list and the reduced string lists ($\xi \in \{\alpha, \beta\}$)

<table>
<thead>
<tr>
<th>$M$</th>
<th>$N_{\xi}$</th>
<th>$L_{M-1}^{N-1}$</th>
<th>$L_{M-2}^{N-2}$</th>
<th>$L_{M-2}^{N-2}$</th>
<th>$L_{M-3}^{N-2}$</th>
<th>$L_{M-4}^{N-2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>3</td>
<td>56</td>
<td>21</td>
<td>15</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>792</td>
<td>330</td>
<td>210</td>
<td>120</td>
<td>84</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>11440</td>
<td>5005</td>
<td>3003</td>
<td>2002</td>
<td>1287</td>
</tr>
<tr>
<td>20</td>
<td>9</td>
<td>167960</td>
<td>75582</td>
<td>43758</td>
<td>31824</td>
<td>19448</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>70</td>
<td>35</td>
<td>20</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>924</td>
<td>462</td>
<td>252</td>
<td>210</td>
<td>126</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>3432</td>
<td>1716</td>
<td>924</td>
<td>792</td>
<td>462</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>12870</td>
<td>6435</td>
<td>3432</td>
<td>3003</td>
<td>1716</td>
</tr>
</tbody>
</table>

From the description above and Table 2.4, we can see the lengths of the reduced lists are dramatically shorter than the full string list. Comparing the memory requirement for storing the CI vector in a CASSCF calculation using Slater determinants basis (with a dimension of $\binom{M}{N_{\alpha}} \binom{M}{N_{\beta}}$), the memory required for storing the reduced string list is negligible. According to Eq. 2.4.7, all the non-zero elements can be generated directly from the reduced string list and is thus very efficient. According to the Pauli principle, the excitation cases like $\langle K_{\alpha} | a_{\alpha i}^\dagger a_{\alpha i}^\dagger a_{\alpha i} a_{\alpha i} | L_{\alpha} \rangle$ do not exist.
2.4.3 The Sign

The non-zero values of \( \langle K_\alpha | E_{ij}^\alpha | J_\alpha \rangle \) (for linear cases this is \( \langle K_\alpha | E_{ij}^\alpha | L_\alpha \rangle \)) are 1 or -1. The associated sign, \( \text{sgn}_{ij}^{K_\alpha} \), is defined through:

\[
\left| K_\alpha \right> = \text{sgn}_{ij}^{K_\alpha} E_{ij}^\alpha \left| J_\alpha \right>
\]
and is easily evaluated from \( \phi_{K_\alpha} [116] \):

\[
\text{sgn}_{ij}^{K_\alpha} = \left\{ \begin{array}{lr} +1 & \text{for } \sum_{a=i+1}^{j-1} b_a \text{ even} \\ -1 & \text{for } \sum_{a=i+1}^{j-1} b_a \text{ odd} \end{array} \right. 
\]

where \( b_i \) indicates the \( i^{th} \) bit of the binary string. Thus the numerical values of the matrix elements for any given \( i \) and \( j \) can be easily generated from the corresponding reduced list dynamically when generating the strings for the corresponding excitation list.

As we now have the basic theory and method of calculating the numerical values in mind, we can move on to the implementation details of how our matrix multiplication method is carried out.

2.5 Implementation Details

In this section we will discuss the implementation details of the matrix multiplication method described above. For the sake of simplicity, we will only outline the general algorithm using Slater determinants. In appendix B we list a detailed subroutine calling sequence of this implementation. This implementation is inherently parallelized but we will postpone this discussion until Section 2.6. Before we start to discuss the details, we shall have an overview of the general algorithm of our matrix multiplication method (given in the Algorithm 2.1 below). We can see since the introduction of the matrix-matrix multiplication, a new parallel scheme can be introduced, which is the parallel level 2 indicated in Algorithm 2.1. Parallel level 1 denotes the parallel scheme existed in the current CASSCF implementation [43].
for 2N electrons and M orbitals
(parallel level 1) loop over \{i, j\}

( generate matrix elements from \( E_\ell^\xi \) )
\[ \Rightarrow |K_{ij}\rangle = \text{sgn}_{ij}^{K_\ell} \cdot E_\ell^\xi |L_\xi\rangle \]
\[ X_{\text{lin}} \left( K_{ij}, L_\xi \right) = X_{\text{lin}} \left( K_{ij}, L_\xi \right) + (i|j)\text{sgn}_{ij}^{K_\ell} \]
\[ X_{\text{lin}} \left( L_\xi, K_{ij} \right) = X_{\text{lin}} \left( L_\xi, K_{ij} \right) + (i|j)\text{sgn}_{ij}^{K_\ell} \]

loop over \( \{k,l\} \)

( generate matrix elements from \( E_\ell^\xi \) )
\[ \Rightarrow |K_{ij}\rangle = \text{sgn}_{kl}^{K_\ell} \cdot E_\ell^\xi |L_\xi\rangle \]
\[ X_{\text{lin}} \left( K_{ij}, L_\xi \right) = X_{\text{lin}} \left( K_{ij}, L_\xi \right) + f(I_{ijkl})(ij|kl)\text{sgn}_{kl}^{K_\ell} \]
\[ X_{\text{lin}} \left( L_\xi, K_{ij} \right) = X_{\text{lin}} \left( L_\xi, K_{ij} \right) + f(I_{ijkl})(ij|kl)\text{sgn}_{kl}^{K_\ell} \]

(bilinear contribution)
matrix elements generation
\[ \Rightarrow |J_{\gamma}\rangle = \text{sgn}_{kl}^{J_\gamma} \cdot E_\ell^\xi |L_\xi\rangle \]
\[ X_{ij} \left( J_{\gamma}, L_\xi \right) = X_{ij} \left( J_{\gamma}, L_\xi \right) + f(I_{ijkl})(ij|kl)\text{sgn}_{kl}^{J_\gamma} \]
\[ X_{ij} \left( L_\xi, J_{\gamma} \right) = X_{ij} \left( L_\xi, J_{\gamma} \right) + f(I_{ijkl})(ij|kl)\text{sgn}_{kl}^{J_\gamma} \]

end loop \( \{k,l\} \)

(bilinear contribution)
\( \text{(parallel level 2) } M_{ij}^{L_\alpha L_\beta} = \sum_{L_\alpha} X_{ij}^{L_\alpha L_\beta} C_{L_\alpha L_\beta} + \sum_{L_\beta} C_{L_\alpha L_\beta} X_{ij}^{L_\beta L_\alpha} \)
\[ \sigma(K_\alpha, K_\beta) = \sigma(K_\alpha, K_\beta) + \sum_{L_\alpha} \langle K_\alpha | E_{ij}^{L_\alpha} | J_\alpha \rangle M_{ij}^{L_\alpha K_\beta} + \sum_{L_\beta} M_{ij}^{K_\beta L_\beta} \left( \langle K_\beta | E_{ij}^{L_\beta} | J_\beta \rangle \right)^T \]

end loop \( \{i, j\} \)

(linear contribution)
\( \text{(parallel level 2) } \sigma(K_\alpha, K_\beta) = \sigma(K_\alpha, K_\beta) + \sum_{L_\alpha} X_{ij}^{L_\alpha K_\beta} C_{L_\alpha K_\beta} + \left[ \sum_{L_\beta} X_{ij}^{K_\beta L_\beta} \left( C_{K_\alpha L_\beta} \right)^T \right]^T \)

Algorithm 2.1: Matrix multiplication method implementation of the CASSCF method (for the non-zero elements generation please refer to [43]). From this algorithm we see the bilinear matrix multiplication is carried out once per \( ij \) index pair, while the linear matrix multiplication is carried out once after all the orbital indices are looped over. Parallel level 1 indicates the parallel scheme used in the current method [43], while parallel level 2 denotes the parallel scheme for the matrix multiplication.
2. Algorithm for the Direct Full CI / CASSCF Method

2.5.1 Matrix $C_{L_{\alpha}L_{\beta}}$ Generation

As stated in Section 2.2, the CI vector $C_L^{n-1}$ is an eigenvector generated from the previous iteration. From Eq. 2.2.1 we can see this vector will remain unchanged during one iteration. Therefore, the two-dimensional matrix corresponding to the CI vector only needs to be generated once per iteration before the loop over the orbital indices starts. In our method, this matrix is indexed by the addresses of full $\alpha$- and $\beta$-strings for the Slater determinant case, e.g. addresses of $\alpha$-strings are used to be the row index, and the column index is defined by the addresses of $\beta$-strings. When HW functions basis is used, the rows and columns of the matrix are both indexed by the addresses of the full $\alpha$-strings. According to Eq. 2.4.3, each string pair defines a Slater determinant. This means we can have a one-to-one mapping between the matrix elements of the CI vector, $C_L^{n-1}$, and the elements of matrix $C \left( L_\alpha, L_\beta \right)$. Based on this, we can have the matrix $C \left( L_\alpha, L_\beta \right)$ for the Slater determinant case as:

\[
C_{L_{\alpha}L_{\beta}} = \begin{pmatrix}
C(1,1) = C_1 & C(1,2) = C_2 & \ldots & C(1,L_\beta) = C_{L_\beta} \\
C(2,1) = C_{\left\{ M \right\}_{L_{\beta}}^{\left\{ L_{\beta} \right\}} + 1} & C(2,2) = C_{\left\{ M \right\}_{L_{\beta}}^{\left\{ L_{\beta} \right\}} + 2} & \ldots & C(2,L_\beta) = C_{\left\{ M \right\}_{L_{\beta}}^{\left\{ L_{\beta} \right\}} + L_\beta} \\
\vdots & \vdots & \ddots & \vdots \\
C(L_\alpha,1) = C_{\left\{ M \right\}_{L_{\alpha}}^{L_{\alpha}} + 1} & C(L_\alpha,2) = C_{\left\{ M \right\}_{L_{\alpha}}^{L_{\alpha}} + 2} & \ldots & C(L_\alpha,L_\beta) = C_{\left\{ M \right\}_{L_{\alpha}}^{\left\{ L_{\beta} \right\}} + L_\beta}
\end{pmatrix}.
\]

From Eq. 2.5.1a we can see the dimension of the matrix derived from the CI vector is the same as the dimension of the CI vector for the Slater determinant basis. For the Hartree-Waller functions basis, we need to normalize the CI vector to generate the matrix $C$. As described above, the calculation for the Hartree-Waller functions is simpler than the calculation for the Slater determinants case as the number of $\alpha$-strings is equal to the number of $\beta$-strings. Therefore in generating the matrix $C$ for the Hartree-Waller functions basis, we only need to use the addresses of the $\alpha$-strings as the rows and columns index. However, in order to avoid confusion, we will still use $\left( L_{\alpha}, L_{\beta} \right)$ to label the matrix $C$ for Hartree-Waller functions cases.
In the case of when the Hartree-Waller Singlet basis is used, since the singlet two-body creation operator is symmetric in the matrix indices [123], when the indices of a \( \alpha \)-string and a \( \beta \)-string are identical, \( \text{Addr}\{L_\alpha\} = \text{Addr}\{L_\beta\} \), the matrix element of \( \mathbf{C} \) at this position needs to be multiplied by \( \sqrt{2} \). Based on Eq. 2.4.4, we can thus generate the matrix \( \mathbf{C} \) for the Hartree-Waller singlet function case from normalizing the CI vector as:

\[
C_{\text{HW,sin}}^{L_\alpha L_\beta} = \begin{pmatrix}
C(1,1) = \sqrt{2} \cdot C_1 & C(1,2) = C_2 & \cdots & C(1,L_\beta) = C_{\frac{(L_\alpha-1)L_\beta}{2}+1} \\
C(2,1) = C_2 & C(2,2) = \sqrt{2} \cdot C_3 & \cdots & C(2,L_\beta) = C_{\frac{(L_\alpha-1)L_\beta}{2}+2} \\
\vdots & \vdots & \ddots & \vdots \\
C(L_\alpha,1) = C_{\frac{(L_\alpha-1)L_\beta}{2}+1} & C(L_\alpha,2) = C_{\frac{(L_\alpha-1)L_\beta}{2}+2} & \cdots & C(L_\alpha, L_\beta) = \sqrt{2} \cdot C_{\frac{(L_\alpha-1)L_\beta}{2}+L_\beta}
\end{pmatrix}
\]

(2.5.1b)

Obviously, in this case, we will always have \( \mathbf{C}_{L_\alpha L_\beta} = (\mathbf{C}_{L_\alpha L_\beta})^T \). Thus in this case, the dimension of the normalized matrix will be roughly two times larger than the dimension of the CI vector (calculated by Eq. 1.5.7) itself of the Hartree-Waller singlet cases. If we define \( D_{\text{HW}}^{\text{sin}} \) as the dimension of the \( \mathbf{C} \) matrix generated for the HW singlet basis, we will have \( D_{\text{HW}}^{\text{sin}} = \left(\frac{M}{N_\alpha}\right)^2 \).

When HW triplet basis is used, for the diagonal positions of the matrix \( \mathbf{C} \) (\( \text{Addr}\{L_\alpha\} = \text{Addr}\{L_\beta\} \)), the matrix element vanishes because of the Pauli principle. Moreover, the triplet two-body creation operator is anti-symmetric. Therefore, we can generate the matrix \( \mathbf{C} \) for the HW triplet basis from normalizing the CI vector as:

\[
C_{\text{HW,tri}}^{L_\alpha L_\beta} = \begin{pmatrix}
C(1,1) = 0 & C(1,2) = C_1 & \cdots & C(1,L_\beta) = C_{\frac{(L_\alpha-1)(L_\beta-2)}{2}+1} \\
C(2,1) = -C_1 & C(2,2) = 0 & \cdots & C(2,L_\beta) = C_{\frac{(L_\alpha-1)(L_\beta-2)}{2}+2} \\
\vdots & \vdots & \ddots & \vdots \\
C(L_\alpha,1) = -C_{\frac{(L_\alpha-1)(L_\beta-2)}{2}+1} & C(L_\alpha,2) = -C_{\frac{(L_\alpha-1)(L_\beta-2)}{2}+2} & \cdots & C(L_\alpha, L_\beta) = 0
\end{pmatrix}
\]

(2.5.1c)
Thus for the Hartree-Waller triplet basis, we will always have $C_{L\alpha L\beta} = -\left(C_{L\alpha L\beta}\right)^T$. Again, if we define the dimension of this matrix as $D_{H,W}^{\text{tr}}$, it is calculated as $D_{H,W}^{\text{tr}} = MN^\alpha \binom{M}{N^\alpha}^2$.

### 2.5.2 Matrix X generation

After generating the matrix C for the matrix multiplication method, we now focus on generating the matrix X that will be used to multiply the matrix C. Since the processes of generating $X_{\text{lin}}$ and $X_{ij}$ are similar (as shown in Algorithm 2.1), we will mainly describe the $X_{ij}$ generation in the following. However, an example of the $X_{\text{lin}}$ construction is also given to make the process of matrix X generation as clear as possible.

According to Eq. 2.3.7a, we know matrix $X_{ij}$ is generated by summing over all the orbital indices $k$ and $l$ that correspond to one $ij$ index pair based on the conditions given in Eq. 2.3.11. Thus we can see when the value of the $ij$ index decreases, the number of the corresponding indices pair $kl$ will reduce, which means the loop over the $kl$ pairs will become shorter and shorter. In other words, when the value of the $ij$ index decreases, the computation becomes cheaper.

For a certain $ij$ index pair, in the loop over the corresponding $kl$ indices, one-particle reduced lists are utilized to generate the addresses of the corresponding strings, Add\{J_\sigma\} and Add\{L_\sigma\} ($\sigma \in \{\alpha, \beta\}$), and the associated sign ($\text{sgn}_{kl}^\alpha$). These addresses will be used as the row indices and the column indices of matrix $X_{ij}$, respectively. Since the loop over $k$ and $l$ is based on the orbital integral symmetry characters (Eq. 2.3.11), we need to take the symmetry factor, $f(I_{ijkl})$ (Eq. 2.3.10), into consideration when generating the elements of $X_{ij}$. Therefore, in one step of the loop over $k$ and $l$, by inserting bits $k$ and $l$ into the elements in the corresponding reduced list, we can have several elements in the matrix $X_{ij}$ as:
\[ X_{ij} \left( \text{Add}\{J_\sigma\}, \text{Add}\{L_\sigma\}\right) = X_{ij} \left( \text{Add}\{J_\sigma\}, \text{Add}\{L_\sigma\}\right) + \frac{1}{2} \cdot \text{sgn}_{kl} I_{ijkl} \cdot (ij|kl). \] 

(2.5.2)

In Table 2.1, we can see that the coefficient of the matrix elements is the combination of both \( E_{kl} \) and \( E_{lk} : \hat{E}_{kl} = E_{kl} + E_{lk} \). Since Eq. 2.5.2 gives the elements corresponding to \( E_{kl} \), we can easily generate the elements in the symmetrical position in \( X_{ij} \) for \( E_{lk} \) as:

\[ X_{ij} \left( \text{Add}\{L_\sigma\}, \text{Add}\{J_\sigma\}\right) = X_{ij} \left( \text{Add}\{L_\sigma\}, \text{Add}\{J_\sigma\}\right) + \frac{1}{2} \cdot \text{sgn}_{kl} I_{ijkl} \cdot (ij|kl). \] 

(2.5.3)

Thus after looping over all the orbital indices \( k \) and \( l \) corresponding to this \( ij \) index, a symmetrical matrix \( X_{ij} \) for this \( ij \) index can be obtained. For a new \( ij \) index, e.g. \( (ij)' \), a new matrix, \( X_{(ij)'} \), will be constructed. Moreover, from Eqs. 2.3.7a and 2.3.13a we can see that the row index and the column index of matrix \( X \) are always the indices of the strings with the same spin type. Therefore we can define the dimension of the matrix \( X \) as:

\[ \begin{pmatrix} M \\ N_\xi \end{pmatrix} \begin{pmatrix} M \\ N_\xi \end{pmatrix} (\xi \in \{\alpha, \beta\}). \]

The following examples illustrate the matrix \( X_{ij} \) generation process (Eqs. 2.5.2 and 2.5.3) by using the chemical system with \( M = 6, N_\sigma = 3 \).

Assuming the inner loop is at the step of \( k=4, l=3 \) corresponding to the \( i = j = 6 \) indices, the reduced list, \( L_{n_{\alpha-1}}^{M-2} \), will be used \((k \neq l)\) to yield the following string pairs (Table 2.5).

<table>
<thead>
<tr>
<th>( \Phi_\xi )</th>
<th>( \Phi_{J_\xi} )</th>
<th>Add{( J_\xi )}</th>
<th>( \Phi_{L_\xi} )</th>
<th>Add{( L_\xi )}</th>
<th>sgn( I_{ijkl}^{J_\xi} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>001011</td>
<td>2</td>
<td>000111</td>
<td>1</td>
<td>+1</td>
</tr>
<tr>
<td>0101</td>
<td>011001</td>
<td>8</td>
<td>010101</td>
<td>6</td>
<td>+1</td>
</tr>
<tr>
<td>1001</td>
<td>101001</td>
<td>9</td>
<td>100101</td>
<td>7</td>
<td>+1</td>
</tr>
<tr>
<td>0110</td>
<td>011010</td>
<td>14</td>
<td>010110</td>
<td>12</td>
<td>+1</td>
</tr>
<tr>
<td>1010</td>
<td>101010</td>
<td>15</td>
<td>100110</td>
<td>13</td>
<td>+1</td>
</tr>
<tr>
<td>1100</td>
<td>111000</td>
<td>20</td>
<td>110100</td>
<td>19</td>
<td>+1</td>
</tr>
</tbody>
</table>

According to Table 2.5, the following elements in matrix \( X_{ij} \) can be obtained:
2. Algorithm for the Direct Full CI / CASSCF Method

\[ X_{ij}(2,1) = X_{ij}(2,1) + \frac{1}{2} \cdot f(I_{ij}) \cdot (66|43) \]
\[ X_{ij}(8,6) = X_{ij}(8,6) + \frac{1}{2} \cdot f(I_{ij}) \cdot (66|43) \]
\[ \vdots \]
\[ X_{ij}(15,13) = X_{ij}(15,13) + \frac{1}{2} \cdot f(I_{ij}) \cdot (66|43) \]
\[ X_{ij}(20,19) = X_{ij}(20,19) + \frac{1}{2} \cdot f(I_{ij}) \cdot (66|43) \]

Because of the symmetry properties, the elements in the symmetrical positions in \( X_{ij} \) are:

\[ X_{ij}(1,2) = X_{ij}(1,2) + \frac{1}{2} \cdot f(I_{ij}) \cdot (66|43) \]
\[ X_{ij}(6,8) = X_{ij}(6,8) + \frac{1}{2} \cdot f(I_{ij}) \cdot (66|43) \]
\[ \vdots \]
\[ X_{ij}(13,15) = X_{ij}(13,15) + \frac{1}{2} \cdot f(I_{ij}) \cdot (66|43) \]
\[ X_{ij}(19,20) = X_{ij}(19,20) + \frac{1}{2} \cdot f(I_{ij}) \cdot (66|43) \]

When all the \( kl \) indices have been looped over, one matrix \( X_{ij} \) for this \( ij \) index can be generated.

Since the \( X_{ln} \) generation is very similar to the bilinear case (as shown in Algorithm 2.1), here we will only give an example to illustrate how to generate the matrix for the linear contribution. According to Algorithm 2.1 we know the elements in \( X_{ln} \) come from two parts: 1-electron contribution, and 2-electron linear contribution. We will start from the 1-electron contributions. Assuming now the \( ij \) index is \( i = 6, j = 4 \), the string pairs and the associated sign generated from reduced list \( L_{n-1}^{M-2} \) for this \( ij \) pair are given in Table 2.6.

<table>
<thead>
<tr>
<th>( \phi_{\xi} )</th>
<th>( \phi_{\zeta} )</th>
<th>( \text{Add}{K_{\xi}} )</th>
<th>( \phi_{L_{\xi}} )</th>
<th>( \text{Add}{L_{\xi}} )</th>
<th>( \text{sgn}<em>{ij}^{K</em>{\xi}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>100011</td>
<td>4</td>
<td>001011</td>
<td>2</td>
<td>+1</td>
</tr>
<tr>
<td>0101</td>
<td>100101</td>
<td>7</td>
<td>001101</td>
<td>5</td>
<td>+1</td>
</tr>
<tr>
<td>1001</td>
<td>110001</td>
<td>10</td>
<td>011001</td>
<td>8</td>
<td>-1</td>
</tr>
<tr>
<td>0110</td>
<td>100110</td>
<td>13</td>
<td>001110</td>
<td>11</td>
<td>+1</td>
</tr>
<tr>
<td>1010</td>
<td>110010</td>
<td>16</td>
<td>011010</td>
<td>14</td>
<td>-1</td>
</tr>
<tr>
<td>1100</td>
<td>110100</td>
<td>19</td>
<td>011100</td>
<td>17</td>
<td>-1</td>
</tr>
</tbody>
</table>
Thus according to Table 2.6, for the 1-electron linear contribution, the elements in $X_{in}$ are:

$$X_{in}(4,2) = X_{in}(4,2) + (6|4)$$
$$X_{in}(7,5) = X_{in}(7,5) + (6|4)$$
$$\vdots$$
$$X_{in}(16,14) = X_{in}(16,14) - (6|4)$$
$$X_{in}(19,17) = X_{in}(19,17) - (6|4)$$

and the elements at the symmetrical positions in $X_{in}$ are:

$$X_{in}(2,4) = X_{in}(2,4) + (6|4)$$
$$X_{in}(5,7) = X_{in}(5,7) + (6|4)$$
$$\vdots$$
$$X_{in}(14,16) = X_{in}(14,16) - (6|4)$$
$$X_{in}(17,19) = X_{in}(17,19) - (6|4)$$

Now we shall move to the matrix elements generation for 2-electron linear contribution case. Assuming the $ij$ index is still $i = 6, j = 4$, and the inner loop over the orbital indices $k$ and $l$ for this $ij$ index is at step $k=4, l=3$, according to Table 2.1, this is the case of integral class 9. For this integral class, only the excitation operator $E_{il}$ contributes to the 2-electron linear contribution which means the string pairs and the associated signs for this case are generated from $i=6, l=3$ only (Table 2.7).

Table 2.7 Matrix elements generation for 2-electron linear contribution when the contribution is from integral class 9 (Table 2.3, in this example this is $i=6, l=3$).

| $\phi_{K|i}$ | $\phi_{K|l}$ | Add${\{K_{i}}|j\}$ | $\phi_{L|i}$ | Add${\{L_{i}}|j\}$ | $-\text{sgn}_{ij}^{K_{i}}$ |
|---------------|--------------|-------------------|--------------|-------------------|----------------|
| 0011          | 100011       | 4                 | 000111       | 1                 | +1             |
| 0101          | 101001       | 9                 | 001101       | 5                 | +1             |
| 1001          | 110001       | 10                | 010101       | 6                 | +1             |
| 0110          | 101010       | 15                | 001110       | 11                | +1             |
| 1010          | 110010       | 16                | 010110       | 12                | +1             |
| 1100          | 111000       | 20                | 011100       | 17                | -1             |

After applying the symmetry factor, $f(I_{ijl})$, in constructing the 2-electron linear contribution matrix elements, we can obtain the elements in $X_{in}$ as:
2. Algorithm for the Direct Full CI / CASSCF Method

According to the integral class 9 given in Table 2.1, the contribution to the $X_{lin}$ elements is only from $E_{il}$, thus the elements for the symmetrical positions will not be generated. Following the process given in the above example, after looping over all the $ijkl$ indices, one matrix $X_{lin}$ for this iteration can be obtained.

2.5.3 The Elements Density in Matrix X

After the matrix $X$ is generated, we would like to know whether it is a dense matrix as we expected. In the following, we will use the bilinear case as an example to identify this. The generation of matrix $X_{ij}$, as indicated above, is achieved by looping over all the orbital indices $kl$ corresponding to one pair of indices $ij$. In the loop over $kl$, the reduced lists are used dynamically to obtain string pairs, of which the addresses are used as the index of matrix $X_{ij}$. If $k = l$, the reduced list $L_{N_k - 1}^{M-1}$ is used, if $k \neq l$, then $L_{N_k - 1}^{M-2}$ is evolved. $L_{N_k - 1}^{M-1}$ can generate $\binom{M-1}{N_k - 1}$ string pairs while $L_{N_k - 1}^{M-2}$ can define $\binom{M-2}{N_k - 1}$ pairs of non-zero-contribution-string. When $L_{N_k - 1}^{M-1}$ is used, since the string pairs generated from this reduced list are the same, $\text{Add}\{J_{\xi}\} = \text{Add}\{L_{\xi}\}$, these elements are usually the diagonal elements in $X_{ij}$. Thus the reduced list $L_{N_k - 1}^{M-2}$ will generate the off-diagonal elements. For a CASSCF calculation that has $M$ active orbitals, when generating $X_{ij}$ for the first $ij$ index which is the largest index, there will be $M$ $k = l$ cases and $\frac{M(M-1)}{2}$ $k \neq l$ cases (according to Eq. 2.3.11). Thus for this $ij$ index, all the diagonal elements in $X_{ij}$ are non-zero elements. Then the question becomes how many non-zero off-diagonal elements can
be obtained. As stated in Section 2.5.2, each string pair generated from $L^{M-2}_{N_{\xi}-1}$ can give 2 non-zero elements (Eqs. 2.5.2 and 2.5.3). So the number of non-zero off-diagonal elements for the first $ij$ index becomes:

$$\left( \frac{M - 2}{N_{\xi} - 1} \right) \cdot \left( \frac{M(M - 1)}{2} \right) \cdot 2, \quad (2.5.4)$$

plus all the non-zero diagonal elements, $\binom{M}{N_{\xi}}$, the total number of non-zero elements in matrix $X_{ij}$ for the largest $ij$ index pair would be:

$$\left( \frac{M - 2}{N_{\xi} - 1} \right) \cdot M(M - 1) + \binom{M}{N_{\xi}}. \quad (2.5.5)$$

Since the dimension of matrix $X_{ij}$ is $\binom{M}{N_{\xi}} \times \binom{M}{N_{\xi}}$, we can easily get the non-zero elements ratio of this matrix $X_{ij}$ corresponding to the largest value of the $ij$ index pair as:

$$\frac{\left( \frac{M - 2}{N_{\xi} - 1} \right) \cdot M(M - 1) + \binom{M}{N_{\xi}}}{\binom{M}{N_{\xi}} \times \binom{M}{N_{\xi}}} \times 100\%. \quad (2.5.6)$$

Take a CASSCF(14,14) calculation (with $M = 14, N_{\xi} = 7$) as an example, for the first $ij$ index which is $i = j = 14$ (the largest $ij$ index pair), the non-zero elements ratio of the corresponding matrix $X_{ij}$ is (according to Eq. 2.5.6):

$$\frac{\left( \frac{14 - 2}{7 - 1} \right) \cdot 14(14 - 1) + 14}{\left( \frac{14}{7} \right) \times \left( \frac{14}{7} \right)} \times 100\% \approx 1.5\%,$$

which is already a very sparse matrix. When the value of $(ij)$ decreases, the sparseness of matrix $X_{ij}$ will increase, because the number of the corresponding $kl$ index pairs decreases. If such a sparse matrix is used to operate a matrix-matrix multiplication with a dense matrix $C$, the computation efficiency will be inefficient, because a lot of zeros will
cause redundancy. Thus we must develop a strategy to deal with this matrix sparseness when carrying out the matrix multiplication.

### 2.5.4 Matrix Multiplication

According to Eqs. 2.3.7 and 2.3.13 we know the matrix multiplication is always carried out between matrices $X$ and $C$. However, as indicated in the above section, the matrix $X$ is very sparse. It would be unwise to use some well-optimised subprograms such as the Basic Linear Algebra Subprogram (BLAS) level 3 subprograms [121] to carry out the matrix multiplication, because these subprograms are developed for dense matrix multiplication. Thus it is necessary to develop a subprogram in this method, based on the concept of the BLAS routines, to carry out the matrix multiplication between the sparse matrix $X$ and the dense matrix $C$:

$$M_{ij}^{L_{\alpha}L_{\beta}} = \sum_{L_{\alpha}} X_{ij}^{L_{\alpha}L_{\alpha}} C_{L_{\alpha}L_{\beta}} + \sum_{L_{\beta}} X_{ij}^{L_{\alpha}L_{\beta}} \left( C_{L_{\alpha}L_{\beta}} \right)^T. \quad (2.5.7)$$

Before carrying out the matrix multiplication, several lists should be made for the sparse matrix so that all the non-zero elements in the sparse matrix can be listed in a sequential order. We will demonstrate this with an example.

Assuming we have a sparse $5 \times 5$ matrix $B$:

$$B = \begin{pmatrix}
1 & -5 & 0 & -1 & 0 \\
-3 & 5 & 0 & 0 & 0 \\
0 & 0 & 2 & 8 & 4 \\
-6 & 0 & 1 & 7 & 0 \\
0 & 8 & 0 & 0 & -5
\end{pmatrix},$$

when carrying out a matrix multiplication in the form of:

$$C = AB,$$

in the core calculation, each element in matrix $B$ will be used as a scalar to time a column of matrix $A$, then the result is added to the corresponding column of matrix $C$ (the scalar times a vector type of calculation [124]):

$$C(n, J) = C(n, J) + B(K, J) \cdot A(n, K).$$
However, since matrix $B$ is a sparse matrix, the multiplication between matrices $B$ and $A$ will be inefficient due to the zeros in matrix $B$ will also be used as the scalar to carry out the multiplication that causes redundancy. To eliminate this redundancy, four one-dimensional arrays are introduced (using column-major format for $B$) to mark the positions and values of the non-zero elements in matrix $B$:

- $XV$: An array that contains the non-zero elements of $B$. Values of the non-zero elements of $B$ are mapped to the $XV$ array using the column-major storage mapping.
- $IDR$: The value of the $I$-th element of this array is the row index in matrix $B$ that corresponds to the $I$-th value in the $XV$ array. This array has the same dimension as array $XV$.
- $IDS$: Row index starting point for a column. The value of the $J$-th element of this integer array gives the starting point of arrays $XV$ and $IDR$ that indicates the first non-zero element of column $J$ of $B$. This array has the dimension of the full number of the columns of $B$.
- $IDE$: Row index ending point for a column. The value of the $J$-th element of this integer array gives the ending point of arrays $XV$ and $IDR$ that indicates the last non-zero element of column $J$ of $B$. For example, $IDE(J) - IDS(J)$ gives the number of all the non-zero elements in column $J$. And $IDS(J)$ gives the starting point in arrays $XV$ and $IDR$ (see example below).

Using the above sparse matrix $B$ as an example, one can generate the four arrays as:

$$
XV = (1 \ -3 \ -6 \ -5 \ 5 \ 8 \ 2 \ 1 \ -1 \ 8 \ 7 \ 4 \ -5)
$$

$$
IDR = (1 \ 2 \ 4 \ 1 \ 2 \ 5 \ 3 \ 4 \ 1 \ 3 \ 4 \ 3 \ 5)
$$

$$
IDS = (1 \ 4 \ 7 \ 9 \ 12)
$$

$$
IDE = (4 \ 7 \ 9 \ 12 \ 14)
$$

Therefore all the non-zero elements in matrix $B$ and its corresponding information e.g. the row and column indices are ordered sequentially in these lists. Then these lists are used for the scalar times a vector type calculation:

$$
C(n, J) = C(n, J) + XV(\{IDS(J), IDE(J) - 1\}) \cdot A(n, IDR(\{IDS(J), IDE(J) - 1\})),
$$
where $\{IDS(J), IDE(J) - 1\}$ indicates one of the values between $IDS(J)$ and $IDE(J)-1$.

Applying this scheme to our matrix multiplication method, the non-zero elements in the sparse matrix $X$ will be distributed in the 4 lists described above. Then the matrix multiplication (Eq. 2.5.7) is carried out by using the sequentially ordered non-zero elements in these lists to update the CI vector.

### 2.5.5 Memory Requirement

As stated in Chapter 1, the factorized matrix multiplication methods will require some extra memory to store the obtained non-zero symbolic matrix elements as the matrix $X$, and some extra memory for the intermediate matrix $M$. Here we will describe how much extra memory will be required for these matrices used in our method. We know in the current CASSCF implementation [43], only the $\sigma$-vector and the $C$ vector take very large memory space. The memory requirement for all other short lists, e.g. the reduced string lists and excitation lists (Section 2.4.2), can be neglected. By contrast, in our matrix multiplication method, since we have introduced several intermediate matrices, more memory will be required. The extra memory requirement depends on the configuration basis one chooses. For example, when the Hartree-Waller functions basis is chosen, the added matrices are the normalized CI vector $C^{fHW}$, the $X$ matrices (one for linear contribution, Eq. 2.3.13a, and one for bilinear contribution, Eq. 2.3.7a), the intermediate matrix $M$, and the arrays ($XV$, ID$R$, IDS, and IDE) that are used for storing the non-zero elements of matrix $X$. As described in previous sections, the dimensions of these added arrays are defined as:

$$\text{Dim}(C^{fHW}) = \text{Dim}(X) = \text{Dim}(M) = \begin{pmatrix} M \\ N_\xi \end{pmatrix} \begin{pmatrix} M \\ N_\xi \end{pmatrix},$$

$$\text{Dim}(XV) = \text{Dim}(IDR) = \begin{pmatrix} M - 2 \\ N_\xi - 1 \end{pmatrix} \cdot M (M - 1) + \begin{pmatrix} M \\ N_\xi \end{pmatrix},$$

and

$$\text{Dim}(IDS) = \text{Dim}(IDE) = \begin{pmatrix} M \\ N_\xi \end{pmatrix}.$$
Therefore, apart from the basic memory requirement of the current method, we can see the extra memory requirement for a Hartree-Waller function basis calculation will be about:

\[ 4 \left( \begin{array}{c} M \\ N_x \end{array} \right) \left( \begin{array}{c} M \\ N_y \end{array} \right) + 2 \cdot \left( \begin{array}{c} M - 2 \\ N_x - 1 \end{array} \right) \cdot M \left( M - 1 \right) + \left( \begin{array}{c} M \\ N_x \end{array} \right) \] + 2 \cdot \left( \begin{array}{c} M \\ N_y \end{array} \right) \cdot \frac{100}{(i = j)} \text{ or } 2 \times \left( \begin{array}{c} M - 2 \\ n_x - 1 \end{array} \right) \times 100\% \left( i \neq j \right). \]

For example, for a Hartree-Waller singlet calculation of a system with 14 active electrons in 14 active orbitals, the extra memory requirement is,

\[ 4 \cdot \left( \begin{array}{c} 14 \\ 7 \end{array} \right) \left( \begin{array}{c} 14 \\ 7 \end{array} \right) + 2 \cdot \left( \begin{array}{c} 12 \\ 6 \end{array} \right) \cdot 14 \left( 14 - 1 \right) + \left( \begin{array}{c} 14 \\ 7 \end{array} \right) \] + 2 \cdot \left( \begin{array}{c} 14 \\ 7 \end{array} \right) = 47464560 \text{ words,} \]

which is about 379Mbytes. For Slater determinant basis, since it involves the introduction of the matrix for the \( \beta \)-strings (one more matrix \( M^T \) will be required), slightly more memory will be required.

### 2.5.6 CI Vector Updating Operation

From Algorithm 2.1, we can see the linear contribution to the CI vector comes directly from the matrix multiplication product (Eq. 2.3.13b). Therefore, the main concern in this section is the CI vector updating process from the bilinear contribution. For this contribution, the CI vector-updating step is carried out via implementing Eq. 2.3.7c in which the matrix \( M \) generated from the matrix multiplication step is used. First, let us still consider applying a matrix multiplication to carry out Eq. 2.3.7c. Since only a few elements will be generated for one \( ij \) pair \( \left( \begin{array}{c} M - 1 \\ n_x - 1 \end{array} \right) \) elements if \( i=j \), or \( 2 \times \left( \begin{array}{c} M - 2 \\ n_x - 1 \end{array} \right) \) elements if \( i \neq j \), the matrix formed for these elements will be very sparse. The non-zero elements density of such a sparse matrix is calculated as:

\[ \frac{\left( \begin{array}{c} M - 1 \\ n_x - 1 \end{array} \right)}{\left( \begin{array}{c} M \\ n_x \end{array} \right) \times \left( \begin{array}{c} M \\ n_x \end{array} \right)} \times 100\% \left( i = j \right), \text{ or } \frac{2 \times \left( \begin{array}{c} M - 2 \\ n_x - 1 \end{array} \right)}{\left( \begin{array}{c} M \\ n_x \end{array} \right) \times \left( \begin{array}{c} M \\ n_x \end{array} \right)} \times 100\% \left( i \neq j \right). \]
Thus to store such a super sparse matrix would be a waste of memory. However, in Section 2.5.4 we also pointed out the core calculation in our matrix multiplication is actually a scalar-times-a-vector style calculation [124]. Therefore, we can build a short list that stores the non-zero elements generated from the given $ij$ index pair. Then the elements in this list are used as scalar to multiple a column of the obtained matrix $M_{ij}$ to update $\sigma(K_{\alpha},K_{\beta})$. We will use the first term of Eq. 2.3.7c as an example to illustrate this.

For certain $ij$ index pair, depending on whether $i=j$ or $i\neq j$, the corresponding reduced list is chosen. From this, an excitation list containing the indices of the corresponding full string pairs and their associated sign, $\{\text{Add}\{K_{\alpha}\},\text{Add}\{J_{\alpha}\},\text{sgn}^{K_{\alpha}}_{ij}\}$, can be obtained. The dimension of this list is only $3 \times \left(\begin{array}{c} M-1 \\ n_{\alpha}-1 \end{array}\right)(i=j)$ (storing the indices of the strings and the associated sign) or $3 \times \left(\begin{array}{c} M-2 \\ n_{\alpha}-1 \end{array}\right)(i \neq j)$. The addresses of the strings, $\text{Add}\{K_{\alpha}\}$ and $\text{Add}\{J_{\alpha}\}$, generated from this $ij$ index pair are used to locate the row indices of matrix $\sigma'_{ij}(K_{\alpha},K_{\beta})$ and the row indices of $M_{ij}(J_{\alpha},J_{\beta})$, respectively. And the associated sign, $\text{sgn}^{K_{\alpha}}_{ij}$, is used as the scalar to the scalar-times-a-vector style calculation. Thus the CI vector updating can be operated in the way of:

$$\sigma'_{ij}(K_{\alpha},n) = \sigma'_{ij}(K_{\alpha},n) + \text{sgn}^{K_{\alpha}}_{ij} \cdot M_{ij}(J_{\alpha},n). \quad (2.5.8)$$

Similarly, the updating process of the second term of Eq. 2.3.7c can be written as:

$$\sigma'_{ij}(n,K_{\beta}) = \sigma'_{ij}(n,K_{\beta}) + \text{sgn}^{K_{\beta}}_{ij} \cdot M_{ij}(n,J_{\beta}). \quad (2.5.9)$$
Based on this, we know the memory requirement for storing such an excitation list,

\[3 \times \binom{M - 1}{n_z - 1} \text{ or } 3 \times \binom{M - 2}{n_z - 1},\]

can be neglected compared to the memory required for storing a very sparse matrix,

\[\binom{M}{n_z} \times \binom{M}{n_z}.\]

### 2.6 Parallelism

So far the serial implementation details about our algorithm have been introduced. In this section, we describe the parallel implementation of the above algorithm. Parallel computing uses multiple processing elements, multiprocessors, simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can diverse and include resources such as a single computer with multiple processors, known as shared memory multiprocessors, several networked computers or specialized hardware, know as distributed memory multiprocessors, or any combination of the above.

The architectural differences between shared memory multiprocessors and distributed memory multiprocessors have implications on how each is programmed. With a shared memory multiprocessor, different processors can access the same variables (Figure 2.2). This makes referencing data stored in memory similar to traditional single processor programs, but adds the complexity of shared data integrity. For this architecture, we have implemented our solution by following the OpenMP model\[125\]. A distributed memory system introduces a different problem: how to distribute a computational task to multiple processors with distinct memory spaces and reassemble the results from each processor into one solution (Figure 2.3). We have implemented our solution within the Linda model \[126\] for the distributed architecture. We will begin with a brief summary of the key concepts of both models and then discuss our implementation.
2. Algorithm for the Direct Full CI / CASSCF Method

OpenMP is an open standard for providing parallelization mechanisms on shared memory multiprocessors. The standard provides a specification of compiler directives, library routines, and environment variables that control the parallelization and runtime characteristics of a program. The code written with OpenMP is portable to other shared memory multiprocessors. The compiler directives defined by OpenMP tell a compiler which region (or regions) of code should be parallelized and define specific options for parallelization. OpenMP is based on a thread paradigm. A thread is an active execution sequence of instructions within a process. A running program, or a process, is allocated its own memory by the operating system when it is loaded into memory. Within a process, multiple threads may exist. Threads within a process share the same memory.
space and have access to the same variables. They have the advantage of allowing a process to perform multiple tasks seemingly simultaneously. Practically, the concept of this thread paradigm is that a master thread can run serially until it encounters a directive to fork off new threads. These new threads can then be distributed and executed on different processors, reducing execution time because different pieces of the process are run simultaneously. Results of each threads execution can then be combined. Figure 2.4 gives an execution flow chart for a simple OpenMP model.

Linda is a distributed memory parallel model. One of the key concepts in the Linda coordination model is the shared, content-addressed, virtual memory called tuple space. Tuples are defined as sequences of data of different types. All the inter-process communication is carried out via operations on tuple space (the virtual memory). Thus coordination in Linda is un-coupled: the acts of sending and receiving data are independent. Figure 2.5 shows the technique of communication between nodes via the tuple space. The data is moved from/to tuple space by using tuples. Linda interacts with the tuple space using six basic operations. Five operations can be used to add/read/remove tuples from the tuple space and a sixth operation is capable of creating new processes. The entire 6 operations are given in Table 2.8. Linda allows for distinguishing between the available nodes and the tasks to be computed. If a worker-process (worker) on a given node finishes a given task, it may look for the next task in the tuple space and continue working, without getting it pre-assigned explicitly.
Table 2.8 The Linda Operations

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in(s)</strong></td>
</tr>
<tr>
<td>Withdraw a tuple matching $s$ from tuple space. If no matching tuple is available, execution suspends until one is. If more than one matching tuple exists, one is chosen arbitrarily. When a match is found, the actuals in the matching tuples are assigned to the formals in the corresponding fields of $s$.</td>
</tr>
<tr>
<td><strong>rd(s)</strong></td>
</tr>
<tr>
<td>Look for a tuple matching $s$ in tuple space. If a matching tuple is found, actual-to-formal assignment occurs. If no matching tuple exists, the process blocks until one becomes available.</td>
</tr>
<tr>
<td><strong>rdp(s)</strong> &amp; <strong>inp(s)</strong></td>
</tr>
<tr>
<td>Predicate forms of <strong>rd</strong> and <strong>in</strong> respectively. They do not block if no matching tuple exists, but return 0/.FALSE. and exit. If a match is found, they return 1/.TRUE. and perform actual-to-formal assignment.</td>
</tr>
<tr>
<td><strong>eval(s)</strong></td>
</tr>
<tr>
<td>Each field of $s$ containing a simple function call results in the creation of a new process to evaluate that field. All other fields are evaluated synchronously prior to process creation. When all field values have become available, the tuple $s$ is placed into tuple space.</td>
</tr>
<tr>
<td><strong>out(s)</strong></td>
</tr>
<tr>
<td>Synchronously evaluates the fields of the tuple $s$ and then places it into tuple space.</td>
</tr>
</tbody>
</table>

The performance of parallelism is defined in terms of scalability. Since the workers communicate heavily with the virtual shared memory – the tuple space that is applied in the distributed memory parallelization, surely the network connection will restrict the scalability of distributed memory parallelization strongly. There are also other factors that can affect the scalability of parallelism. For example the first factor is called the memory-CPU bandwidth, which describes the rate at which data can be read from or stored into memory by a processor. This effect is directly from the computer hardware. So to avoid this effect, more advanced hardware should be chosen. Another factor that can affect the scalability of a parallel program is the parallel algorithm, e.g. if there are serial part in the program, the scalability will fulfil the Amdahl’s law [127]. This law indicates that if there
is $x\%$ of serial component in a program, the parallel speedup cannot be better than \( \frac{100}{x} \).

This effect can be avoided by minimizing the serial sections of the program. The third factor is called the parallel overhead, which is defined as the amount of time required to coordinate parallel tasks, as opposed to doing useful work. This is related directly to the size of the problem that will be divided for the selected shared memory processors. If the problem is too small, e.g. not having massive data to be executed, and is parallelized, then the time used to read from and write to memory of each processor will be larger than the time the data is executed by the processor. In this case, probably running the program serially will perform better than running the program in parallel. Also, load balancing is another factor that affects the scalability of parallelization for both shared memory and distributed memory parallelism. Load balancing is defined as to distribute workload evenly across the processors that are used for parallel so that no single processor is overwhelmed. To achieve a good load balancing, a dynamic load balancing is suggested. That is to distribute the pieces of works to the processors implicitly so that once a processor finishes its current task; it loads dynamically a new task until all pieces of tasks are done.

Now we should discuss our parallel implementation. There are two key issues for parallelizing the algorithm we have developed: splitting the total work into sub-tasks that is to be done by different processors and load balancing, as these factors will affect the scalability of the parallelism strongly.

The creation of sub-tasks essentially involves a decision about the parallel loop, where each cycle of the loop defines a sub-task that gets carried out in parallel on a node. However, the sub-tasks may be of different lengths, so load balancing becomes important to ensure that each processor is kept busy, i.e. one processor may carry out several sub-tasks while another processor does only a single task. Furthermore, simple explicit allocation of equal numbers of tasks to all nodes would result in poor load balancing in cases where the processing elements are not equally loaded or have a different architecture.
As indicated in Algorithm 2.1, we can see our matrix multiplication method is parallelized in two levels. The first level (Parallel level 1 in Algorithm 2.1), which is the current CASSCF parallelization scheme, is to parallelize (both shared memory and distributed memory architectures) the program on the level of loops over the orbital indices $i$ and $j$. This is because to parallelize the program at this level can keep a good balance between the number of tasks so to minimize the communication overhead\(^2\) and load balancing. Each processor will be distributed a unique $ij$ index pair to carry out the CI vector updating for this pair serially. Moreover, new tasks are allocated dynamically each time a processor finishes its current one, which results very good load balancing. Reference [43] provides more details about the parallelization on this level. In this section, we will focus on the new parallel scheme, which is to parallelize the program on the matrix multiplication level (Parallel level 2 in Algorithm 2.1), and the combination with the level 1 parallelization.

The matrix multiplication algorithm we have developed involves the outer loops over the orbital indices $i$ and $j$ (the parallel scheme used in the current CASSCF implementation). Inside the $ij$ level nested loops, all corresponding $kl$ indices are looped over and the $X$ matrices are generated. Then, still inside the $ij$ level loops, the updating of the CI vector takes place (Eqs. 2.3.7b and 2.3.7c). In other words, the matrix multiplication and CI vector updating are carried out once per $ij$ index pair inside the $ij$ level loops (bilinear contributions only). As we can see from Algorithm 2.1, apart from the $ij$ level parallelization, the matrix multiplication can also be parallelized. In the following, we will denote this matrix multiplication parallelization as the $kl$ level parallelization as to indicate this is for the matrix multiplication parallel within the $ij$ index loops for the bilinear contributions.

In our algorithm, as stated above, sub-tasks do not get pre-allocated to particular workers from the outset. By contrast, load balancing is achieved by allocating tasks dynamically.

\(^2\) Each Linda worker will read data from and write data to the tuple space. If the number of tasks is too large, the communication time will be longer than the time used for useful executions, which results communication overhead. To minimize this effect, a relatively small number of tasks should be generated.
This means each time a worker finishes its current task, it gets a new task automatically. The communication overhead mainly appears in distributed memory parallelism because each worker takes some time to read tasks and data from, and write results to the tuple space. In order to keep overheads as low as possible and keep a good load balancing, we have decided to keep on using indices $i, j$ to define the parallel loop for distributed memory parallelism as the current method does. The communication cost can be reduced by using shared memory parallelism because in this case all the processors share the same memory space and all the static data and the combined results have to be passed only once to these processors. Therefore, on the basis of $ij$ level parallelization for shared memory parallel, we have introduced the new $kl$ level parallelization for shared memory parallelism. In the following, we will describe the implementation of this strategy.

The flowchart of distributed memory parallelism of our method is given in Figure 2.6. We assume each node is a symmetric multi-processor (SMP) machine. The number of processors on a SMP is $NProcS$ (for single processor nodes: $NProcS = 1$). As described above, the shared memory parallelism can be implemented on $ij$ level, or $kl$ level, or a combination of both. If we define the number of shared memory processors used for $kl$ level parallelization as $NProckl$ (set by user explicitly), and the number of shared memory processors used for $ij$ level parallelization as $NProci(j)$ ($NProci(j) \times NProckl \leq NProcS$), then for shared memory parallelization, one has three options: 1. to parallelize on $ij$ level only (by setting $NProckl = 1$ explicitly); 2. on $kl$ level only (by setting $NProckl = NProcS$ explicitly); or 3. a combination of both ($NProci(j) > 1$ and $NProckl > 1$ with $NProci(j) \times NProckl \leq NProcS$). As all the necessary parallel notations are defined, we shall discuss this in details.

In distributed memory parallelism, the main process is called the master process (master). The master retrieves the first index $ij$ (usually the most expensive task which equals $ij = \frac{M(M + 1)}{2}$) and puts a new index $ij' = ij - NProci(j)$ into tuple space, which will be retrieved by the next process and so on. If $NProckl$ is set as 1 explicitly, then we have $NProci(j) = NProcS$ (thus identical to the existed parallel scheme). In this case, $NProcS - 1$
shared memory processes are created. Each process will be distributed one unique $ij$ index pair.

Figure 2.6. The data flowchart for the parallelization as described in the text.
Thus the original loop over $ij$ is parallelized (only using Parallel level 1 in Algorithm 2.1). If $NProckl$ is set greater than 1 by a user, then $(NProcS/NProckl)-1$ shared memory processes are created and each of the $(NProcS/NProckl)$ processes is assigned one unique $ij$ index. And the $NProckl$ processors are used for $kl$ level parallelism corresponding to the $ij$ index. Each processor carries out the computation of updating the CI vector using the algorithm described in Sections 2.3-2.5. On finishing, the next available index $ij$ is retrieved from tuple space and this procedure is continued until the index $ij = 0$ is found. Then the intermediate results are passed through tuple space to be combined to give the final result of the calculation.

As mentioned above, there are three ways to carry out a parallel computation: 1) using Linda only, 2) using shared memory only, or 3) a combination of both. When using Linda only, one Linda worker is created on each node. Network performance, as stated above, is a factor that can affect the performance of Linda parallelism. Fortunately, as the development of the modern computer network, the network effect to Linda can be neglected. Furthermore, using Linda takes full advantage of the dynamical load balancing mechanism (most expensive task executed first) described above. However, the static data must be replicated for each Linda worker.

When using shared memory parallelism only, as discussed above, there are also three ways to operate: 1) on $ij$ level only, 2) on $kl$ level (matrix multiplication level) only, or 3) a combination of both. The advantage of this type of parallelism is all the static data gets replicated only once per SMP and are shared by all processors. Therefore the communication overhead that appears in the distributed memory parallelism can be eliminated. When the parallelization is carried out on $ij$ level only ($NProckl = 1$), the parallelization concept is the same as the distributed memory parallelization concept described above. This means each processor will be distributed one unique $ij$ index and carries out the corresponding matrix multiplication of this $ij$ index serially. All result vectors (one per processor) are summed before the result of this worker is passed back to tuple space. By contrast, when parallelization is on $kl$ level only ($NProckl = NProcS$), a SMP will be assigned only one $ij$ index. The corresponding matrix multiplication of this
2. Algorithm for the Direct Full CI / CASSCF Method

The $ij$ index is parallelized on the $NProcS$ processors. Then the result vector (only one) is passed back to tuple space. The third option is to use a combination of both the $ij$ level and the $kl$ level parallelization ($NProcij = \frac{NProcS}{NProckl} > 1$ and $NProckl > 1$). In this case, $NProcij$ $ij$ indices are distributed to $NProcij$ processors on a node. These processors will behave like masters and use $NProckl$ processors to parallelize the corresponding matrix multiplication. Figure 2.7 shows an example when $NProcij$ and $NProckl$ are both set as 3 the combination of the shared memory parallelism using 9 processors. The tasks defined by the $NProcij$ indices are of slightly different length, and defined by the $NProckl$ indices are of the same length. In practice this means that parallel efficiency will decrease if $NProcij$ becomes very large. To avoid this, one can set $NProckl$ large and keep $NProcij$ small.

Figure 2.7. When using shared memory parallelism only, there will be three ways to carry out the CI vector updating calculation. i) parallelizing on $ij$ level, means each processor will be allocated one $ij$ index. ii) parallelizing on $kl$ level, indicates all the 9 CPUs will be used for parallelizing the matrix multiplication corresponding to one $ij$ index. Or iii) a combination of both. For example, if we set 3 processors to be used for $kl$ level parallelization, then we know 3 $ij$ index pairs will be executed at the same time ($NProcij = \text{Int}\left(\frac{NProcS}{NProckl}\right)$). Therefore, 3 processors will act as masters and each uses 3 processors (including the master itself) for parallelizing the matrix multiplication corresponding to the $ij$ index.
The third way is to combine the distributed memory parallelism and the shared memory parallelism together. In this case, one can set Linda workers on several separated SMPs for distributed memory parallelization and on these SMPs shared memory parallelization is carried out.

The parallelism implementation described above provides great flexibility in choosing the number of processors and, due to the relatively high number of tasks (higher than the number of processors used for \( ij \) level parallelization), leads to a good load balancing. Furthermore, since the tasks become considerably cheaper as \( ij \) gets smaller (indicates a smaller number of data to be executed), the implicit order of computation is an essential feature in order to ensure efficient usage of all processors. The dynamical load-balancing scheme implemented also allows for the fact that, in many environments, the CPU-time on different nodes of a parallel machine may be shared among a number of running programs and thus automatically uses the resources as they become available.

One more thing needs to be pointed out is the memory requirement for the shared memory parallelism. As described above, in our method, the shared memory parallelism can be carried out in three ways: on \( ij \) level, or on \( kl \) level, or a combination of both. If we parallelize our method at the \( ij \) level, then each processor will be assigned a unique \( ij \) index pair. Eqs. 2.3.7 will be carried out serially for this \( ij \) index pair on this thread. Thus we can see for \( ij \) level shared memory parallelism, a copy of matrices \( X_{ij}^{J_L \xi_L} \), \( M_{ij}^{J_{\alpha_{\beta}}} \), and \( \sigma_{ij}^{K_{\alpha_{K_{\beta}}}} \) that corresponding to a certain \( ij \) index pair will be necessary for each processor.

The dimension of the matrix \( X_{ij}^{J_L \xi_L} \) is the square of the number of \( \xi \)-strings \( \left( \frac{M}{n_{\xi}} \right)^2 \), and the dimensions of the matrices \( M_{ij}^{J_{\alpha_{\beta}}} \) and \( \sigma_{ij}^{K_{\alpha_{K_{\beta}}} \xi_L} \) are the same as the CI vector. Therefore, for \( ij \) level shared memory parallelism, the memory requirement will increase when the number of shared memory processors increases. By contrast, for the \( kl \) level shared memory parallelism, since all the processor share the same \( ij \) index pair, and the matrix multiplication (Eq. 2.3.7b) is parallelized, only one copy of the matrices of \( X_{ij}^{J_L \xi_L} \),
Algorithm for the Direct Full CI / CASSCF Method

$M_{ij}^{\alpha \beta}$ and $\sigma_{ij}^{K \lambda} \sigma_{ij}^{K \beta}$ is needed. Therefore, the memory requirement for the $kl$ level parallelism will remain constant no matter how many shared memory processors are used. However, as we will see in the following section soon that the new method has a much better performance than the current method, the execution time for each $ij$ index pair in the new method becomes very short (or cheap). If we parallel on the $kl$ level and the task is very cheap, as described above, we won’t get too much benefit from the parallelism because of the parallel overhead. Therefore it is recommended that to use $ij$ level parallelism to carry out the shared memory CASSCF calculations when the system is with a relatively small active space, e.g. a CASSCF(14,14) calculation, although the memory requirement for the $ij$ level shared memory parallel is larger than the $kl$ level shared memory parallel.

2.7 Test Calculations and Performance

In this section we will investigate the performance of the newly-developed matrix multiplication method compared to the current implementation of Gaussian. In this section, PC clusters are mainly used for our tests. In the following, all the timing data is obtained by using Intel Nehalem servers (2 Intel Nehalem Quad-core 2.5 GHz processors per node with a peak speed of $\sim 80$ GFlops [128]) on the PC cluster. However, before we test the performance of the new method, we need to make sure it works properly, i.e. produces the same CI vector elements as the current method. Thus we will start with a method verification sub-section. In the tables that will appear in this section, we will label the newly developed matrix multiplication method as “New”, and the current method as “Old”.

2.7.1 Verification of the New Method

So far we have developed the method described in the above sections. However, we would like to know whether this new method works properly, e.g. gives the same result as the current method. In order to test this, the same type of test jobs should be carried out
on the same type of hardware using both methods. In the following, single point full ground state energy calculations are carried out on three systems: naphthalene, acenaphthylene, and pyracylene (Chart 2.1). Table 2.9 gives the single point ground state energies using Slater determinant basis (Figure 2.8 visualizes Table 2.9). From the result of Table 2.9 we can see the final results generated by both the new method and the old method are identical. Moreover, by comparing the CI vectors produced from each iteration of both methods, we found the elements in the two corresponding CI vectors are identical. This means that the new method works correctly. Moreover, from the Job CPU time column of Table 2.9 we can see that the new method always performs faster than the current method. And as the system becomes bigger, the new method can perform faster and faster than the current method.

Chart 2.1. The systems used for testing the new method performance. Naphthalene is used as a CASSCF (10,10) system, acenaphthylene is used as a CASSCF (12,12) system, pyracylene is used as a CASSCF (14,14) system, and pyrene is a CASSCF(16,16) system. Among these systems, the first three are used to verify the new method works while pyracylene and pyrene are used to demonstrate the performance of the new method vs. the performance of the current method.
2. Algorithm for the Direct Full CI / CASSCF Method

Table 2.9 Verification test on the new method. We only list the result of serial CASSCF single point energy Slater determinants basis calculations on three systems (naphthalene, acenaphthylene, and pyracylene) by using both the new method and the current method. The computation cost of the current method on pyrene is too high to be carried out here. The outputs of these three systems, e.g. the diagonal elements of the final density matrix, the lowest eigenvalue of the CI vector, and the job CPU time of both methods are compared with each other. The CPU time is equal to the wall clock time for a serial calculation. In this table, the CPU time of link 510 (the link that carries out the MCSCF calculation in program package Gaussian) is listed as taking ~99.99% time of the entire job CPU time.

<table>
<thead>
<tr>
<th>System</th>
<th>Number of Configurations</th>
<th>Eigenvalues / Final Energy (Hartree)</th>
<th>Job CPU Time for L510 (s)</th>
<th>New Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naphthalene CAS(10,10) SD serial</td>
<td>63504</td>
<td>New: -382.9117239</td>
<td>100.9</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old: -382.9117239</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acenaphthylene CAS(12,12) SD serial</td>
<td>853776</td>
<td>New: -458.5173472</td>
<td>2528.7</td>
<td>3.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old: -458.5173472</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pyracylene CAS(14,14) SD serial</td>
<td>11778624</td>
<td>New: -534.1895791</td>
<td>32928.5</td>
<td>4.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old: -534.1895791</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.8 The visualized result of the timing data given in Table 2.9. From this figure we can see the new method can always perform faster than the current method. Particularly when the investigated system becomes larger and larger, the matrix multiplication method can perform faster and faster.

Apart from the most general Slater determinant cases, as described in the previous sections, we also have developed simplifications of singlets and triplets using the
Hartree-Waller functions basis. We have tested the Slater determinants cases above (Table 2.9), we now would like to know whether the Hartree-Waller functions implementations give the same results as the current method when they both use HW functions basis. In order to determine this, several single point excited states energy calculations on system naphthalene alone are carried out (Table 2.10). For these calculations, we expect the outputs of the HW singlet (/triplet) will give the same outputs as the outputs of SD singlet (/triplet). From the result given in Table 2.10, we can see our expectation is achieved. This means again that the newly-developed method also works correctly if the HW functions basis is used.

Table 2.10 The excited states calculation of the Hartree-Waller (HW) singlet, HW triplet, Slater determinant (SD) singlet, triplet of naphthalene and ground state optimization of naphthalene$^+$ (doublet) by using both of the new method (New) and the current method (Old). The Slater determinant singlet calculation generates 4 states, which are singlet S_0, triplet T_1, triplet T_2, and singlet S_1. Since the HW singlet calculation will not result in any triplet energy, we will only compare the singlet energies generated by both SD singlet basis and HW singlet basis. We can see the SD singlet S_0 and S_1 energies are the same as HW singlet S_0 and S_1 energies. By contrast, SD triplet calculation will only give triplet energies. Thus we can see SD triplet and HW triplet calculations generate identical results.

<table>
<thead>
<tr>
<th>Calculation: Naphthalene CAS(n,10)</th>
<th>No. α-electron</th>
<th>No. β-electron</th>
<th>No. config.</th>
<th>Nroot=</th>
<th>$E$(hartree)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HW singlet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old</td>
<td>5</td>
<td>5</td>
<td>31878</td>
<td>2</td>
<td>$E_0$ = .382.782165665 $E_1$ = .382.735639012</td>
</tr>
<tr>
<td>New</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HW triplet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old</td>
<td>5</td>
<td>5</td>
<td>31626</td>
<td>2</td>
<td>$E_0$ = .382.782165665 $E_1$ = .382.735639012</td>
</tr>
<tr>
<td>New</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SD singlet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old</td>
<td>5</td>
<td>5</td>
<td>63504</td>
<td>4</td>
<td>$E_0$ = .382.782165665 $E_1$ = .382.735639012</td>
</tr>
<tr>
<td>New</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SD triplet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old</td>
<td>6</td>
<td>4</td>
<td>44100</td>
<td>2</td>
<td>$E_0$ = .382.782165666 $E_1$ = .382.735639012</td>
</tr>
<tr>
<td>New</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n = 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$E_0$ = .382.64838871</td>
</tr>
</tbody>
</table>

$n = 9$ doublet Old New 5 4 52920 1 $E_0$ = .382.64838871
2.7.2 Performance Investigation

2.7.2.1 Serial Performance

As we have proved that the new method works properly, now we would like to investigate the performance of the new method. As described previously, our implementation is mainly about the CI vector updating. Thus in order to test the performance of our implementation, we should investigate the performance difference between the new method and the old method on the CI vector updating by using the timing data of certain full Davidson iteration. We choose the timing data elapsed by the $4^{th}$ Davidson iteration of the calculation on pyracylene and pyrene respectively, to investigate the serial and parallel performance of the new method. The reason for choosing this iteration is because when the calculation gets into this iteration the CI vector generated from the previous iterations becomes completely dense (the initial guess CI vector for the first iteration is sparse). All timing data using different bases is listed in Table 2.11a for pyracylene and in Table 2.11b for pyrene. From Tables 2.11, we can see no matter what basis is used, the new method always performs much faster (~an order of magnitude) than the old method. Moreover, with the increasing of the active space, the new method can perform faster. This proves that we have successfully improved the performance of the current method. From the timing data of the new method listed in Table 2.11a, we know the Slater determinant basis calculation on pyracylene takes the longest time. This is because this type of calculation has the largest configuration number (thus the biggest CI vector) compared to the other bases. In order to investigate the performance of the parallelization of the new method, we will choose the Slater determinant basis for testing calculation on pyracylene, because the timing data listed in Table 2.11a indicates that such a calculation can be long enough to show the performance differences between serial jobs and parallel jobs. However, since the computational cost on pyrene of the current method is too high when using SD basis, we would like to choose a relatively cheaper basis to investigate the parallel performance of the CAS(16,16) calculation. From Table 2.11b, we can see the HW singlet calculation on pyrene of the current method is the cheapest basis, but the timing data is already long.
enough to show the performance differences between the two methods of serial and parallel jobs respectively. Thus we will use HW singlet basis calculations on pyrene as our test calculation.

Table 2.11a Serial timing data for the 4th Davidson iteration when using different bases on the same system of pyracylene (CASSCF(14,14)). We can see from this table that no matter what basis is used, the matrix multiplication method is always much faster than the current method.

<table>
<thead>
<tr>
<th>System</th>
<th>Pyracylene (CASSCF(14,14))</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Basis used</td>
<td>SD</td>
<td>HW singlet</td>
<td>HW triplet</td>
<td>SD triplet</td>
<td>doublet</td>
</tr>
<tr>
<td>No. of Configurations</td>
<td>11,778,624</td>
<td>5,891,028</td>
<td>5,887,596</td>
<td>9,018,009</td>
<td>10,306,296</td>
</tr>
<tr>
<td>No. of α -strings</td>
<td>3432</td>
<td>3432</td>
<td>3432</td>
<td>3003</td>
<td>3432</td>
</tr>
<tr>
<td>No. of β -strings</td>
<td>3432</td>
<td>3432</td>
<td>3432</td>
<td>3003</td>
<td>3003</td>
</tr>
<tr>
<td>Timing (s)</td>
<td>Old 1009.87</td>
<td>499.82</td>
<td>675.53</td>
<td>766.44</td>
<td>803.23</td>
</tr>
<tr>
<td></td>
<td>New 132.07</td>
<td>66.24</td>
<td>64.45</td>
<td>113.39</td>
<td>112.3</td>
</tr>
<tr>
<td>New Method Speedup</td>
<td>7.65</td>
<td>7.55</td>
<td>10.48</td>
<td>6.76</td>
<td>7.15</td>
</tr>
</tbody>
</table>

Table 2.11b Serial timing data for the 4th Davidson iteration when using different bases on the same system of pyrene (CASSCF(16,16)). We can see from this table that no matter what basis is used, the matrix multiplication method is always much faster than the current method.

<table>
<thead>
<tr>
<th>System</th>
<th>Pyrene (CASSCF(16,16))</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Basis used</td>
<td>SD</td>
<td>HW singlet</td>
<td>HW triplet</td>
<td>SD triplet</td>
<td>doublet</td>
</tr>
<tr>
<td>No. of Configurations</td>
<td>165,636,900</td>
<td>82,824,885</td>
<td>82,812,015</td>
<td>130,873,600</td>
<td>147,232,800</td>
</tr>
<tr>
<td>No. of α -strings</td>
<td>12,870</td>
<td>12,870</td>
<td>12,870</td>
<td>11,440</td>
<td>12,870</td>
</tr>
<tr>
<td>No. of β -strings</td>
<td>12,870</td>
<td>12,870</td>
<td>12,870</td>
<td>11,440</td>
<td>11,440</td>
</tr>
<tr>
<td>Timing (s)</td>
<td>Old 42000.35</td>
<td>14813.40</td>
<td>16000.98</td>
<td>26276.97</td>
<td>32415.24</td>
</tr>
<tr>
<td></td>
<td>New 3503.82</td>
<td>1454.07</td>
<td>1431.2</td>
<td>3055.13</td>
<td>3496.85</td>
</tr>
<tr>
<td>New Method Speedup</td>
<td>11.99</td>
<td>10.19</td>
<td>11.18</td>
<td>8.60</td>
<td>9.27</td>
</tr>
</tbody>
</table>

2.7.2.2 Parallel Performance

Now we will investigate the parallel performance of the new method. As described above, we will use the 4th Davidson iteration of Slater determinant calculations on pyracylene and HW singlet calculations on pyrene as our test calculations. In Section 2.6 we discussed the parallel scheme that is applied in our new method. Compared to the current method, our method has one new feature in the shared memory parallelization, which is the parallelization on matrix multiplication (kl level). Since the current method
2. Algorithm for the Direct Full CI / CASSCF Method

carries out both shared memory and distributed memory parallelization on $ij$ level only, in order to compare the performance of the new method with the current method, we will start from the investigation of the performance of the shared memory $ij$ level parallelism of both methods. Table 2.12a gives the timing data for the 4th Davidson iteration of calculations using both methods on pyracylene (CASSCF(14,14)), where wall time gives the real elapsed time that this iteration takes and the CPU time indicates the CPU time used by this iteration. Table 2.12b lists the $ij$ level parallel timing data of both methods on pyrene calculation (CASSCF(16,16)). Figure 2.9 shows the visualized version of speedup vs. the number of processors used in Tables 2.12.

Table 2.12a $ij$ level OpenMP timing data of the 4th Davidson iteration for the system of pyracylene (SD CAS(14,14) calculation).

<table>
<thead>
<tr>
<th>System</th>
<th>Pyracylene, CAS(14,14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Determinants</td>
<td>11,778,624</td>
</tr>
<tr>
<td>No. of strings</td>
<td>3432</td>
</tr>
<tr>
<td>Number of processors</td>
<td>1</td>
</tr>
<tr>
<td>Old</td>
<td></td>
</tr>
<tr>
<td>CPU time (s)</td>
<td>1009.87</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>1009.87</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
</tr>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>CPU time (s)</td>
<td>132.07</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>132.07</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2.12b $ij$ level OpenMP timing data of the 4th Davidson iteration for the system of pyrene (HW singlet CAS(16,16) calculation).

<table>
<thead>
<tr>
<th>System</th>
<th>Pyrene, CAS(16,16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of configurations</td>
<td>82,824,885</td>
</tr>
<tr>
<td>No. of strings</td>
<td>12870</td>
</tr>
<tr>
<td>Number of processors</td>
<td>1</td>
</tr>
<tr>
<td>Old</td>
<td></td>
</tr>
<tr>
<td>CPU time (s)</td>
<td>14813.40</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>14813.40</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
</tr>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>CPU time (s)</td>
<td>1454.07</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>1454.07</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
</tr>
</tbody>
</table>

From Tables 2.12 and Figure 2.9 we can see although the speedup still goes up when the number of processors increases, we cannot get a linear scaling. Especially for the CAS(16,16) test job we can see the shared memory parallel speedup only changes slightly when the number of processors changes from 4 to 8. As stated in Section 2.6, there are many factors that can affect the scalability of a parallel implementation, e.g. memory-CPU bandwidth, parallel algorithm, and parallel overhead. Since on $ij$ level
parallelization, both the new method and the old method use the same parallel structure, there will be no difference in the parallel algorithm (otherwise the outcome of the pyracylene would not be good). Thus the reason for this performance result would be either the memory-CPU bandwidth or parallel overhead, or even a combination of both.

The hardware we chose has a relatively good memory-CPU bandwidth, which is the rate at which data can be read from or stored into memory by a processor. However, comparing the parallel results given in Tables 2.12a and 2.12b, we can see when the size of data that flows between memory and CPU is relatively small, e.g. the CAS(14,14) calculation, we can reach relatively good speedup when using up to 8 shared memory processors. The memory-CPU bandwidth has relatively little effect on the speedup. However, once the size of the data becomes large, e.g. the CAS(16,16) calculation, the parallel performance will be strongly affected by the memory-CPU bandwidth. This is why we can see from Table 2.12b, when the number of shared memory processors changes from 4 to 8, the speedup only changes slightly for both methods.

Also from Tables 2.12 we can see the current method shows slightly better parallel performance than the new method. However, this doesn’t mean the shared memory
Algorithm for the Direct Full CI / CASSCF Method

The parallel performance of the new method is worse than the current method. The reason for this scalability difference between the two methods should be the parallel overhead. As described in Section 2.6, parallel overhead is defined as “the amount of time required to coordinate parallel tasks, as opposed to doing useful work.” Thus if the time used for doing the useful work is very short, then the parallel overhead will take a considerable amount of time during the total elapsed time. For the testing systems we used above, the shared memory parallelization is carried out at $ij$ index level. The pyracylene system has 14 active electrons in 14 active orbitals. Thus the total number of $ij$ index pairs, according to Section 2.5.3, is calculated as: \( \frac{14(14+1)}{2} = 105 \). Similarly, there will be 136 $ij$ index pairs for the pyrene system. We know from Tables 2.12 that the total amount of time used serially for the new method to calculate one iteration of all the 105 (136) $ij$ indices of pyracylene (pyrene) is about 132 (1454) seconds. Thus on average, the amount of time that is required to complete one $ij$ index calculation is around 1.25 (10.69) seconds. By contrast, this amount of time that is required in the current method is around \( \frac{1010}{105} = 9.62 \) (\( \frac{14813}{136} = 108.92 \)) seconds. Nevertheless, 1.25 (10.69) seconds is still much larger than the overhead time because each processor will be assigned one unique $ij$ index pair. This is why we can see a good scaling when using up to 4 shared processors for pyracylene and pyrene. However, when the number of processors becomes larger, the traffic on the shared memory-CPU path is increased too, which adds more overheads. Therefore it is not easy for us to get a good scaling when using more than 4 shared processors.

In Section 2.6 we also introduced a new shared memory parallel scheme that is designed to parallelize the new code on the matrix multiplication level ($kl$ level). We also would like to investigate the performance of this kind of parallel. Since the current method doesn’t parallel on this level, we will only list the parallel timing data for this scheme of the new method (Table 2.13 visualized in Figure 2.10). As mentioned above, the average time used for calculating one $ij$ index pair is around 1.25 (10.69) seconds in the new method. Thus the shared memory on $kl$ level is to split this 1.25 (10.69) seconds to the
shared memory processors. On one hand, the more shared memory processors one has used, the shorter time that each processor elapses to carry out the “useful work”. On the other hand, the more shared memory processors are used, the more amount of time of overhead is required. Therefore we can predict that when the number of shared memory processors increases to certain level, the actual performance of the parallel will become slower when increasing the number of threads. From Table 2.13 and Figure 2.10 we can see that the performance of $kl$ level parallel is very poor. This matches our prediction. Therefore the parallel overhead is the main bottleneck for the $kl$ level parallel.

Table 2.13 The shared memory timing data for the matrix multiplication level ($kl$ level) parallelization. Since the time required to finish one matrix multiplication is too short, the parallel performance on the matrix multiplication will be strongly affected by the parallel overhead.

<table>
<thead>
<tr>
<th>NProcShared</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pyracylene</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAS(14,14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$kl$ level</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU time (s)</td>
<td>132.07</td>
<td>172.16</td>
<td>286.25</td>
<td>404.23</td>
<td>610.46</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>132.07</td>
<td>86</td>
<td>72</td>
<td>67</td>
<td>76</td>
</tr>
<tr>
<td>Parallel speedup</td>
<td>-</td>
<td>1.54</td>
<td>1.83</td>
<td>1.97</td>
<td>1.76</td>
</tr>
<tr>
<td>Pyrene</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAS(16,16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$kl$ level</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU time (s)</td>
<td>1454.07</td>
<td>2475.39</td>
<td>3593.5</td>
<td>5545.81</td>
<td>6335.2</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>1454.07</td>
<td>1238</td>
<td>899</td>
<td>924</td>
<td>792</td>
</tr>
<tr>
<td>Parallel speedup</td>
<td>-</td>
<td>1.17</td>
<td>1.62</td>
<td>1.57</td>
<td>1.84</td>
</tr>
</tbody>
</table>

Figure 2.10 $kl$ level shared memory parallelization performance of the new method. We can see due to the execution time for each $ij$ index pair being too short, the parallelization on the $kl$ level performs very poorly.
So far we have demonstrated the timing data of the shared memory parallelization. As stated in Section 2.6, apart from the shared memory parallelization, we can also parallelize our algorithm by using distributed memory parallelism via Linda and/or a combination of both. In Tables 2.14 the timing data using Linda only is listed (Figure 2.11 shows the speedup vs. number of Linda nodes). From this table we can see when using Linda only, the speedup scales linearly. This is because in this case each node has its own memory system, there is no memory-CPU bandwidth restriction and shared memory parallel overhead. However, if the number of Linda workers becomes large, a price of communication time between the Linda workers and the tuple space must be paid. The reason the speedup goes down of the pyrene Linda only parallelization (Table 2.14b) of the new method is because the linear contribution is carried out once per iteration via matrix multiplication outside the distributed memory parallel region. Thus, if only using Linda for parallelization, the linear contribution will always be carried out serially. The time used for the linear contribution in the CASSCF(14,14) calculation is about 1% of the total elapsed time of one Davidson iteration. Thus we can see there is a linear scaling for the distributed memory parallelization of the CAS(14,14) calculation, due to the small effect of the serially linear contribution. However, the time used for the linear contribution in the CASSCF(16,16) calculation has risen to about 8% of the total elapsed time of one Davidson iteration. The serially running linear contribution will thus affect the parallel performance at a considerable level (shown in Table 2.14b and Figure 2.11). Therefore for the CASSCF(16,16) calculation, when only using Linda for parallelization, the speedup becomes less linear when the number of Linda nodes increases. Tables 2.15 give the timing data using a combination of Linda parallel and OpenMP parallel (on ij level only). In these two tables, for the CAS(14,14) calculation, the number of nodes are set as 2 for representative while the number of shared memory processors varies. And for the CAS(16,16) calculation, the number of nodes has been set as 4 as representative while the number of shared memory processors varies. The speedup vs. number of processors used is shown in Figure 2.12.
Table 2.14a Linda only timing data of the 4th Davidson iteration for the system of pyracylene. We can obtain a linear scaling of this type of calculation because in this type of parallelization, each processor can access its own memory without any overhead occurring.

<table>
<thead>
<tr>
<th>System</th>
<th>Pyracylene CAS(14,14)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Determinants</td>
<td>11,778,624</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of strings</td>
<td>3432</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NProcLinda =</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>NProcShared =</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old</td>
<td>CPU time (s)</td>
<td>1009.87</td>
<td>1008.18</td>
<td>1014.42</td>
<td>1035.32</td>
</tr>
<tr>
<td></td>
<td>Wall time (s)</td>
<td>1009.87</td>
<td>505</td>
<td>338</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>Parallel speedup</td>
<td>-</td>
<td>2.00</td>
<td>2.99</td>
<td>3.90</td>
</tr>
<tr>
<td>New</td>
<td>CPU time (s)</td>
<td>110.24</td>
<td>106.95</td>
<td>107.52</td>
<td>106.46</td>
</tr>
<tr>
<td></td>
<td>Wall time (s)</td>
<td>110.24</td>
<td>54</td>
<td>37</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Parallel speedup</td>
<td>-</td>
<td>2.04</td>
<td>2.98</td>
<td>3.94</td>
</tr>
</tbody>
</table>

Table 2.14b Linda only timing data of the 4th Davidson iteration for the system of pyrene. We can obtain a linear scaling of this type of calculation because in this type of parallelization, each processor can access its own memory without any overhead occurring.

<table>
<thead>
<tr>
<th>System</th>
<th>Pyrene CAS(16,16)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of configurations</td>
<td>82,824,885</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of strings</td>
<td>12870</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NProcLinda =</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>NProcShared =</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old</td>
<td>CPU time (s)</td>
<td>14813.40</td>
<td>14797.82</td>
<td>14779.56</td>
<td>14779.86</td>
</tr>
<tr>
<td></td>
<td>Wall time (s)</td>
<td>14813.40</td>
<td>7400</td>
<td>3695</td>
<td>2463</td>
</tr>
<tr>
<td></td>
<td>Parallel speedup</td>
<td>-</td>
<td>2.00</td>
<td>4.01</td>
<td>6.01</td>
</tr>
<tr>
<td>New</td>
<td>CPU time (s)</td>
<td>1454.07</td>
<td>1379.54</td>
<td>1338.6</td>
<td>1374.94</td>
</tr>
<tr>
<td></td>
<td>Wall time (s)</td>
<td>1454.07</td>
<td>762</td>
<td>408</td>
<td>294</td>
</tr>
<tr>
<td></td>
<td>Parallel speedup</td>
<td>-</td>
<td>1.91</td>
<td>3.56</td>
<td>4.95</td>
</tr>
</tbody>
</table>

Figure 2.11 The performance of both method when using distributed memory parallelization (Linda) only. We can see when using only Linda, the speedup scales linearly due to each node accessing its own memory without the restrictions of memory-CPU bandwidth and overhead.
2. Algorithm for the Direct Full CI / CASSCF Method

Table 2.15a Linda+OpenMP timing data of the 4th Davidson iteration for the system of pyracylene. In this calculation, the number of nodes is fixed at 2 while the number of shared memory processors varies.

<table>
<thead>
<tr>
<th>System</th>
<th>CAS(14,14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Determinants</td>
<td>11,778,624</td>
</tr>
<tr>
<td>No. of strings</td>
<td>3432</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>NProcL = 2</td>
</tr>
<tr>
<td>Number of shared memory processors</td>
<td>serial</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>1009.87</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>1009.87</td>
</tr>
<tr>
<td>Parallel Speedup</td>
<td>-</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>526.98</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>263</td>
</tr>
<tr>
<td>Parallel Speedup</td>
<td>-</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>836.97</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>209</td>
</tr>
<tr>
<td>Parallel Speedup</td>
<td>-</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>670.28</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>112</td>
</tr>
<tr>
<td>Parallel Speedup</td>
<td>-</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>784.71</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>98</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Pyrene, CAS(16,16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of configurations</td>
<td>82,824,885</td>
</tr>
<tr>
<td>No. of strings</td>
<td>12870</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>NProcL = 4</td>
</tr>
<tr>
<td>Number of shared memory processors</td>
<td>serial</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>14813.40</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>14813.40</td>
</tr>
<tr>
<td>Parallel Speedup</td>
<td>-</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>15864.52</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>1969</td>
</tr>
<tr>
<td>Parallel Speedup</td>
<td>-</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>18051.64</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>1129</td>
</tr>
<tr>
<td>Parallel Speedup</td>
<td>-</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>25332.16</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>1056</td>
</tr>
<tr>
<td>Parallel Speedup</td>
<td>-</td>
</tr>
<tr>
<td>CPU time (s) per node</td>
<td>31559.44</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>986</td>
</tr>
<tr>
<td>Parallel Speedup</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 2.12 The speedup vs. number of processors for using Linda+OpenMP parallel scheme. We can see for the CAS(16,16) calculation, the performance of the new method is worse than the old method. This is because when using Linda only the performance of the new method is slightly worse than the current method (Figure 2.11).
2.7.3 Performance on Real Applications

Based on the above description, we can see the new method works properly and it does perform much faster than the current method. So far we have only presented data of test calculations. Now we would like to know whether this newly-developed method is suitable for real problems, e.g. the ground state geometry optimization. In this section, we choose planar conjugated hydrocarbon cations (anthracene$^+$ / phenanthrene$^+$, Chart 2.2) to demonstrate our method can be used for real applications. The calculation on these systems is to distribute 13 $\pi$ electrons into 14 valence $\pi$ orbitals (thus a CASSCF (13,14) doublet calculation). In order to compare the performance and the final result, we have used the same geometry ($D_{2h}$ and $C_{2v}$ symmetry point group [129] for anthracene$^+$ and phenanthrene$^+$ respectively) and the same basis sets (6-31G* basis sets [130,131]) reported in reference [47]. From Sections 2.7.1 and 2.7.2, we know our new method works properly as well as faster. We thus expect getting the same optimized structure compared with the reference with a better performance. For more details about the two cation-systems, reference [47] is recommended. We have carried out the geometry optimization calculations by using OpenMP parallelization (with 8 shared memory processors). The timing data and the final result of the calculations are listed in Table 2.16 while the final structures of these systems after optimization are listed in Figure 2.13.

![Anthracene$^+$ and Phenanthrene$^+$](chart22.png)

Chart 2.2 The structure of cations anthracene$^+$ and phenanthrene$^+$. The CASSCF calculations of these systems have been reported in reference [47]. And to compare the results, we have used the same reported geometry structure and basis sets.
Figure 2.13 The geometry structure of the ground states optimization (using the newly-developed CASSCF method) of the investigated systems. (a) gives the final optimized structure of the anthracene cation, while (b) is the optimized structure of the ground state of phenanthrene cation with $1B_1$ symmetry species [132] of $D_{2h}$ symmetry group respectively. The unit of the values (bond length) is Å. The result agrees with the reference [47].
Table 2.16 The calculation results of the ground state geometry optimization of two hydrocarbon cations (anthracene$^+$ / phenanthrene$^+$) carried out using both current and the new methods. The initial geometry structure data is obtained from a MMVB calculation. In order to compare the results with ref.[47], the same basis sets (6-31G* that includes polarization $d$ functions on the carbon atoms) is used. From this table we can see the new method works properly as well as performs much faster than the current method when used for real problems. The timing data of ref.[47] is not listed because the reference calculations were carried out by using a different type of computer hardware.

<table>
<thead>
<tr>
<th>CASSCF(13,14) doublet ground state geometry opt. / 6-31G*</th>
<th>No. of Conf.</th>
<th>Geometry Energy (Hartree)</th>
<th>Job CPU Time (s)</th>
<th>New Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1510</td>
<td>overall</td>
</tr>
<tr>
<td>Anthracene$^+$ (B$<em>3g$ energy, D$</em>{2h}$ symmetry)</td>
<td>10,306,296</td>
<td>Ref.: -535.936464</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New: -535.9364637</td>
<td>119793</td>
<td>173380.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old: -535.9364637</td>
<td>568100.4</td>
<td>621754</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.60</td>
</tr>
<tr>
<td>Phenanthrene$^+$ (1B$<em>1$ energy, C$</em>{2v}$ symmetry)</td>
<td>10,306,296</td>
<td>Ref.: -535.926834</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New: -535.926834</td>
<td>135286.7</td>
<td>189062.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old: -535.926834</td>
<td>471937.9</td>
<td>525721.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.76</td>
</tr>
</tbody>
</table>

From Table 2.16 and Figure 2.13 we can see that the new method works properly as well as performing faster than the current method. Therefore, all our expectations described above are fulfilled.

2.8 Summary

The direct reduced list based matrix multiplication CASSCF algorithm presented in this chapter represents a much better performing method than the current widely used method. Its main feature is that by combining the efficient non-zero symbolic matrix elements generation scheme (the reduced string lists concept) used in the current method and matrix-matrix multiplication concept together, data in the CI vectors can be used in a more predictable order, thus increasing the performance at a significant level. The pyrene test jobs illustrate this feature (the new method is about 10 times faster than the current method). From these test jobs, we can see the new method can now be used for some big
systems that the current method cannot afford. Another feature of this new method is that based on the parallel scheme of the current method, a new shared memory scheme is introduced that is to parallelize on the matrix multiplication level \((kl \text{ level})\). Therefore, more flexibility for using parallel computation is added, e.g. on the shared memory parallel level, one can choose to parallelize the program at \(ij\) level when the available memory is large, or to parallelize on \(kl\) level when the available memory is limited, or a combination of both. The algorithm is implemented in the current development version of the \textit{Gaussian} [42] program package. As well as the most general case of Slater determinants basis we also implement the straightforward simplifications for singlets and triplets using the Hartree-Waller functions basis.
Chapter 3

A New Matrix-Matrix Multiplication Approach to the Direct Restricted Active Space Self-Consistent-Field (RASSCF) Method and Its Parallel Implementation
3. Algorithm for the Direct RASSCF Method (Part I)

3.1 Introduction

The complete active space self-consistent field (CASSCF) method [33] has been very successfully applied to study excited-state reactivity, minima, transition structures and potential energy surfaces etc. [35-41] of molecules. It has many advantages over other truncated CI methods, e.g. the fact that it is size consistent. Interpretation of the resulting orbitals is also straightforward, since orbital rotations within the active space are invariant with respect to the energy [33], which allows one to select the orbitals for the active space. Unfortunately, applicability of the CASSCF method is restricted to relatively small chemical systems. The CASSCF configuration number, which defines the dimension of the CI vector, increases drastically when the number of active orbitals increases slightly. For example, the number of configurations of a system that has 14 active electrons distributed in 14 active orbitals is 11,778,624. Once the number of active orbitals is increased to 16 with 16 active electrons, the number of the configurations becomes 165,636,900. However, the occupation number of several orbitals in the active space is frequently either close to 2 or close to 0 during the entire investigation. Thus we can mimic the CASSCF configurations by applying certain occupancy restrictions on groups of orbitals. This allows us to include all configurations that contribute to non-dynamical electron correlation while reducing the size of the CI vector compared to the CAS expansion.

Dynamical correlation effects, as described in Section 1.3, are also very important to include in an electronic structure computation. We also stated in Chapter 1 that the CASSCF method could also recover dynamical correlation when it has an active space larger than e.g. the minimal π space [10a, 90]. But since the number of configurations increases dramatically when the active space increases slightly, as indicated above, CASSCF is not usually used for recovering dynamical correlation effects. Instead, the dynamical correlation effect is included either by carrying out a subsequent calculation after a CASSCF calculation, e.g. adding a calculation treating dynamic correlation perturbatively [e.g. 92,93], or performing a multi-reference CI (MRCI) calculation where selected configurations from the CASSCF wave function are used as reference
configurations [94,95], although this type of calculation cannot be routinely carried out by Gaussian [42]. However, this treatment assumes the reference configurations are those obtained without taking dynamical correlation into account. For systems, such as negative ions and excited states [96], that have strongly occupied orbitals, which depend on dynamical correlation effects, this treatment will break down. Therefore, an extended space may be essential for an adequate description of the electronic structure problem at hand.

Both of the above two cases can be treated with the Restricted Active Space SCF (RASSCF) method [34,44,96,133]. The RASSCF method may be considered a logical extension to the CASSCF method. In the RASSCF formalism, the configuration space is specified by dividing the molecular orbitals into five subsets and imposing restrictions on the allowed configurations based upon occupations within those subsets. The first subset consists of the lowest lying (inactive) MOs, which interact only weakly with the other MOs, and are treated as frozen. They are always doubly occupied for any allowed configurations. The second subset, named RAS1, typically includes all doubly occupied MOs in some accepted reference. Allowed configurations must contain a minimum of \( p \) electrons (or maximum holes: MxHole) within RAS1. The third subset, called RAS2, includes MOs believed to be particularly important for the system under investigation. No occupancy restrictions are imposed on RAS2. The fourth subset, RAS3, consists of weakly occupied MOs, which contribute relatively less to the description of the system of interest. Any allowed configuration can only have a maximum of \( q \) electrons (MxElec) in RAS3. The final subset includes all remaining MOs. They are unoccupied in all configurations of the RASSCF wavefunction (Figure 1.1).

Similarly to the CASSCF case discussed in the last chapter, there are two factors to be considered in solving the RASSCF eigenvalue problem. The first one is how to generate the non-zero symbolic matrix elements,

\[
A_{ij}^{KL} = \langle K \mid E_{ij} \mid L \rangle, \tag{3.1.1}
\]

and

\[
B_{ijkl}^{KL} = \langle K \mid E_{ij} E_{kl} - \delta_{ik} E_{jl} \mid L \rangle, \tag{3.1.2}
\]
3. Algorithm for the Direct RASSCF Method (Part I)

in a efficient way. In Eqs. 3.1.1 and 3.1.2, \( |K\rangle\text{and}|L\rangle\) denote configurations and \( E_{ij} \) indicates the excitation operator, taking the form as 

\[
E_{ij} = \sum \xi_i^\dagger \xi_j^\dagger \xi_j \xi_i .
\]

Then, based on the non-zero symbolic matrix elements obtained, the non-zero Hamiltonian matrix elements can be generated,

\[
H_{KL} = \sum_{ij} h_{ij} A_{ij}^{KL} + \frac{1}{2} \sum_{ijkl} (ij|kl) B_{ijkl}^{KL},
\]

(3.1.3)

where \( h_{ij} \) and \( (ij|kl) \) are the usual one- and two-electron integrals. As indicated in Chapter 1, the current method for the generation of non-zero elements [44] of the RASSCF wavefunction used in Gaussian is already efficient enough. Thus we will directly use this scheme in this chapter. However, a brief review of this scheme will be given in this section and a more detailed review will be given in the following sections.

For the non-zero matrix elements generation, due to the introduction of the orbital occupation restrictions in the RAS1 and RAS3 subspaces of the RASSCF wavefunction, the computation of the elements becomes more complicated than the unrestricted CASSCF wavefunction. This is because there was only one type of orbital integral (within the full active space) in the CASSCF wavefunction while the occupation restrictions in the RASSCF wavefunction raise many different types of orbital integrals (among orbitals in different RAS subspaces or in the same RAS subspace). Moreover, most of the configurations that were used for generating the non-zero symbolic matrix elements in the CASSCF wavefunction will be invalid in the RASSCF wavefunction because of the orbital occupation restrictions. Thus the highly efficient “reduced string list” concept that we used in Chapter 2 for the CASSCF wavefunction cannot be applied to the full RAS space in the RASSCF wavefunction directly. To solve this, the current method has adopted and extended the “model space” concept proposed by Saunders and van Lenthe [95]. A model space can be regarded as an orbital space that has no occupation restrictions, but is representative of the entire RASSCF active orbital space (RAS1+RAS2+RAS3), which does have occupation restrictions in RAS1 and RAS3 subspaces. Since there are no occupation restrictions in the model space, one can apply the well-developed “reduced string list” concept to this space, resulting a highly efficient
way to generate the non-zero symbolic matrix elements for the RASSCF wavefunction directly. Before we review the method of generating non-zero matrix elements, it is necessary to review what a model space looks like.

The restricted occupation of orbitals concept was first given by Siegbahn [94] for the single and double replacements from a set of reference configurations, which is the multi-reference CISD (MRCISD) implementations. In this method, the entire active molecular orbital space has been divided into ‘internal space’ and ‘external space’. The internal set “contains the few orbitals occupied in the reference configurations”[94] (i.e. they can be treated as complete active orbitals) while the external set “contains the large number of unoccupied orbitals”[94] (i.e. virtual orbitals). Therefore the symbolic matrix elements (Eqs. 3.1.1 and 3.1.2) can also be written as a product of an internal and an external coupling coefficients:

\[ A_{ij}^{KL} = C_{ij}^{KL} \times D_{ij}^{KL}, \]
\[ B_{ijkl}^{KL} = C_{ijkl}^{KL} \times D_{ijkl}^{KL}, \]

where \( C_{ij}^{KL} (C_{ijkl}^{KL}) \) comes from the internal active space and \( D_{ij}^{KL} (D_{ijkl}^{KL}) \) comes from the external virtual space. The efficiency of this separation into two products comes from the fact that the external vector coupling coefficients, \( D_{ij}^{KL} \) and \( D_{ijkl}^{KL} \), are comparatively “easily evaluated and take on very simple values”[94]. And the more complicated internal coupling coefficients can be recycled. However, although the external coupling coefficients are relatively easy to evaluate, some amount of time is still required to calculate these coefficients, since the orbital indices run over the entire external space. Moreover, for MRCISD, the set of external coupling coefficients actually share the same value for certain internal coupling coefficients. Thus there will be redundancy if calculating the external coupling coefficients by running over the entire external space.

The model space concept introduced by Saunders and van Lenthe [95] has made a great improvement to Siegbahn’s method [94]. In this method, Saunders and Van Lenthe (denoted SV hereafter) discovered that for the MRCISD implementation since the occupation number of the external space is at most 2, it is possible to replace the external
3. Algorithm for the Direct RASSCF Method (Part I)

space with two model external orbitals (Figure 3.1). Model configurations are in this scheme constructed by assigning 0, 1 or 2 electrons (opposite spin or the same spin) to the model external space. The coupling coefficients are then computed in the model orbital space and remain constant for all the full space configuration pairs, $|K\rangle$ and $|L\rangle$, with the same occupation and spin pattern in the internal space as the corresponding model configurations, which simplifies the computation of the non-zero symbolic matrix elements dramatically.

![Diagram](image)

**Figure 3.1** The construction of the model external space. Since the occupation number of the external space is at most 2, the external space that contains a large number of virtual orbitals can be replace by two model external orbitals. This operation reduces the size of the external space dramatically and thus more easier to generate the external coupling coefficients. Moreover, the obtained model symbolic matrix elements will remain constant for all the full space configuration pairs $K,L$ with the same occupation and spin pattern as the corresponding model configurations.

The model space concept described above, however, is restricted to only a special case of the RASSCF wavefunction, e.g. cases with no RAS1 subspace and where MxElec in the RAS3 subspace is fixed as 2. Thus this type of methods may be used for MRCISD only. A more general model space concept is required for RASSCF implementations that are not covered by the special case described above. Klene et al. [44] have extended the
model space concept developed by SV to include the RAS1 model space and remove the restrictions of the values of MxHole and MxElec (Figure 3.2).

![Figure 3.2 The model RAS space generation in the Klene’s method. The numbers of orbitals in the model RAS1 and RAS3 subspaces are determined by the maximum number of holes (MxHole) and the maximum number of electrons (MxElec) allowed in RAS1 and RAS3 subspaces, respectively. Therefore the size of the model space is determined by the size of the RAS2 subspace and the values of MxHole and MxElec.](image)

From Figure 3.2 we can see that the model RAS space reduces the full RAS orbital space dramatically. The size of the model space keeps a constant once the size of RAS2 and the values of MxHole and MxElec are fixed no matter how the sizes of RAS1 and RAS3 subspaces vary. For example, for a system that has $M(RAS1)=4$, $M(RAS2)=6$, $M(RAS3)=12$, and MxHole=MxElec=2, the size of the model space is $6+2+2 = 10$. This size will remain constant if $M(RAS2)$ and the values of MxHole and MxElec are fixed. Therefore, even if the size of the RAS3 subspace doubles, e.g. $M(RAS3)=24$, the size of the model space will still be 10. Moreover, since there are no occupation restrictions in the RAS model space, one can fully adopt the “reduced string list” method used in the CASSCF method to the RAS model space. In Klene’s method, based on the model space concept, after a pair of model strings and its associated model symbolic matrix element is obtained by applying the “reduced string list” method on the model space, it is directly expanded to the full RAS space. Since the RAS model space is a representation of the full RAS space, one pair of model strings and its associated model symbolic matrix element will
3. Algorithm for the Direct RASSCF Method (Part I)

represent a set of full RAS space string pairs and the corresponding full RAS space symbolic matrix elements. Then the indices of the full RAS space string pairs and the values of the corresponding full RAS space symbolic matrix elements are used directly to update the CI vector of the RASSCF wavefunction. More details about how the model string pairs are generated and expanded to the full RAS space in the current RASSCF implementation will be given in the following sections.

Now we move to the second factor for solving the CI eigenvalue problem efficiently, which is how to use the non-zero matrix elements that have been obtained to update the RASSCF CI vector in an efficient way,

\[ \sigma^n_K = \sum_L H_{KL} C^{n-1}_L, \]  

(3.1.6)

where \( C^{n-1} \) is an approximate eigenvector from the previous iteration. As indicated in Chapter 1, although many robust RASSCF methods (e.g. [34,44,96]) have been developed for evaluating the CI eigenvectors, there is still space to improve performance. Before discussing how to achieve this, a brief review of the present methods should be given. In these methods, as pointed out in Chapter 1, there are two distinct ways to carry out the RASSCF CI vector evaluation. One is to factorized the \( B \) symbolic matrix elements (Eq. 3.1.2) via inserting an intermediate configuration, \( J \), as,

\[ B^{KL}_{ijkl} = \sum_J \langle K | E_{ij} | J \rangle \langle J | E_{kl} | L \rangle - \delta_{jk} \langle K | E_{ij} | L \rangle. \]  

(3.1.7)

The second term of Eq. 3.1.7 can be classified as a linear contribution and the first term is a bilinear contribution, which is a product of two linear contributions. As stated in Section 1.7.2, by factorizing the bilinear contribution and applying the string concept [107,108] to the determinants, one can use a matrix-multiplication approach via string-based basic linear algebra routines to update the CI vector \( \sigma^n_K \) from the vector \( C^{n-1}_L \). The disadvantage of this scheme, however, is that extra memory will be required for storing the matrices used for the matrix multiplication step. The representative method for this scheme is that developed by Olsen et al [34].
The alternative, termed as the unfactorized method, is to calculate the two-electron symbolic matrix elements (Eq. 3.1.2) directly without factorization. The advantage of this type of method is that no extra memory is required other than the memory requirement for storing the CI vectors $\sigma_k^n$ and $C_{L_{n-1}}^n$. The CI vector can be updated directly once the non-zero element, $B$, is obtained. The disadvantage of this method is, as described in Chapter 1, that it is impossible to use string-based basic linear algebra routines to carry out the CI vector updating because if these routines are used, the memory requirement for storing the matrices for matrix multiplication would be at the number of configuration level rather than at the number of string level. This direct usage of the non-zero matrix elements that have been obtained causes the indices of the data in the CI vector to be generated in an unpredictable and non-sequential order. Thus the CI vector would be accessed frequently resulting in slow performance. The representative method of this scheme is the method developed by Klene et al [44], which is the current RASSCF implementation in Gaussian. In the following we will briefly review these two typical methods and develop a method to improve the performance of the current RASSCF implementation in Gaussian.

We shall start with a review of the current RASSCF implementation [44] in Gaussian. As described above, the current method updates the CI vector directly once the non-zero matrix elements are obtained. When applying the string concept developed by Knowles and Handy [107, 108] to the determinants, the CI vector evaluation formulae becomes a combination of a 1-electron contribution and a two-electron contribution,

$$
\begin{align*}
{1e} & \sigma_{K_\alpha L_\alpha K_\beta L_\beta}^n = \sum_{l_\alpha l_\beta} \sum_{i,j} (i|j) \langle K_\alpha | E_{ij} | L_\alpha \rangle + \langle K_\beta | L_\beta \rangle C_{l_\alpha l_\beta}^{n-1} \\
&= \sum_i \sum_{j} (i|j) \left[ \sum_{l_\alpha} \langle K_\alpha | E_{ij} | L_\alpha \rangle C_{l_\alpha l_\beta}^{n-1} + \sum_{l_\beta} \langle K_\beta | L_\beta \rangle C_{l_\alpha l_\beta}^{n-1} \right], \quad (3.1.8a)
\end{align*}
$$

and
Algorithm for the Direct RASSCF Method (Part I)

\[ \sigma_{\alpha,\beta}^{n} = \frac{1}{2} \sum_{i,j} \sum_{k,l} (ij|kl) \left( K_{\alpha} K_{\beta} \right) E_{ij} - \delta_{ik} E_{jl} \left| L_{\alpha} L_{\beta} \right| C_{\alpha,\beta}^{n-1} \]

\[ = \sum_{i,j} \sum_{k,l} \left( \langle K_{\alpha} | a_{i_{\alpha}}^{\dagger} a_{k_{\alpha}} a_{j_{\alpha}} a_{l_{\alpha}} | L_{\alpha} \rangle C_{\alpha,\beta}^{n-1} \right. \]
\[ + \sum_{l_{\beta}} \left. \langle K_{\beta} | a_{i_{\beta}}^{\dagger} a_{k_{\beta}} a_{j_{\beta}} a_{l_{\beta}} | L_{\beta} \rangle C_{\alpha,\beta}^{n-1} \right) \]
\[ + \sum_{l_{\beta}} \langle K_{\alpha} | E_{ij} | L_{\alpha} \rangle \sum_{l_{\gamma}} \langle K_{\beta} | E_{kl} | L_{\beta} \rangle C_{\alpha,\beta}^{n-1} \]
\[ + \sum_{l_{\beta}} \langle K_{\beta} | E_{ij} | L_{\beta} \rangle \sum_{l_{\gamma}} \langle K_{\alpha} | E_{kl} | L_{\alpha} \rangle C_{\alpha,\beta}^{n-1} \] \quad \forall K_{\alpha}, K_{\beta} \tag{3.1.8b}

Comparing Eq. 3.1.8b with Eq. 2.1.5, we can see in the RASSCF CI vector evaluation formulae that the two-electron contribution is true for all \( \{ K_{\alpha}, K_{\beta} \} \) pairs that generate valid combinations of configurations, \( |K\rangle \). This is because of the occupation restrictions introduced to the RAS1 and RAS3 subspaces. Due to this restriction, in the RASSCF wavefunction, \( \alpha \)-strings are no longer freely combined with any \( \beta \)-strings. In the RASSCF wavefunction, the orbital space has been divided into 3 subspaces. Since a \( \xi \)-string (\( \xi \in \{ \alpha, \beta \} \)) expands all the electrons with \( \xi \)-spin in this orbital space, for a \( \xi \)-string the orbital space can also be divided into 3 subspaces. Each subspace can only contain electrons with \( \xi \) spin. If we define the number of holes in the RAS1 subspace of a \( \alpha \)-string as \( i_{h} \) (\( i'_{h} \) for a \( \beta \)-string) and the number of electrons in the RAS3 subspace of the \( \alpha \)-string as \( i_{e} \) (\( i'_{e} \) for the \( \beta \)-string), the condition of the combination of an \( \alpha \)-string and a \( \beta \)-string to generate a valid configuration would be defined as,

\[
\left\{ \begin{array}{l}
  i_{h} + i'_{h} \leq \text{MxHole} \\
  i_{e} + i'_{e} \leq \text{MxElec}
\end{array} \right. \tag{3.1.9}
\]

Thus we can see the \( \forall K_{\alpha}, K_{\beta} \) term in Eqs. 3.1.8 represents Eq. 3.1.9. As indicated above, updating the CI vector in this way can avoid the extra memory requirement for storing the obtained non-zero matrix elements, but will produce unpredictable and non-sequential indices of the configurations that index the CI vectors, \( \sigma^{n} \) and \( C^{n-1} \). These gaps between the unpredictable indices of the configurations will require the CI vectors to be accessed
frequently. Thus the performance of the current RASSCF implementation in *Gaussian* is strongly affected.

We also stated in Chapter 1 that the string-based factorized method developed by Olsen *et al* [34] could use basic linear algebra routines to carry out the CI vector updating. Taking the second term of Eq. 1.7.7 as an example, this process for the RASSCF wavefunction is written as,

\[ 2'\sigma'(K_\alpha, K_\beta) = \sum_{ij} \sum_{L_\beta} (ij|kl) \left< J_\beta | E_{kl}^\beta | L_\beta \right> \sum_{J_\alpha} \left< K_\alpha | E_{ij}^\alpha | J_\alpha \right> \left< K_\beta | J_\beta \right> C_{J_\alpha L_\beta}^{n-1} \]

\[ = \sum_{ij} \sum_{L_\beta} (ij|kl) \left< K_\beta | E_{kl}^\beta | L_\beta \right> \sum_{J_\alpha} \left< K_\alpha | E_{ij}^\alpha | J_\alpha \right> C_{J_\alpha L_\beta}^{n-1} \quad \forall K_\alpha, K_\beta, \tag{3.1.10} \]

where the condition of Eq. 3.1.9 is also applied. Except for the introduction of the string combination condition (Eq. 3.1.9) to generate the index of the corresponding determinants, all other steps of updating the CI vector in Olsen’s RASSCF method are the same as described in Section 2.1 (Eqs. 2.1.11). That is,

1. **Matrix Generation**

\[ C'_{ij}(K_\alpha, L_\beta) = \sum_{J_\alpha} \left< K_\alpha | E_{ij}^\alpha | J_\alpha \right> C(J_\alpha, L_\beta), \quad \forall K_\alpha, L_\beta, \tag{3.1.11a} \]

\[ X_{ij}(K_\beta, L_\beta) = \sum_{kl} (ij|kl) \left< K_\beta | E_{kl}^\beta | L_\beta \right>, \tag{3.1.11b} \]

2. **Matrix Multiplication**

\[ 2'\sigma'_{ij}(K_\alpha, K_\beta) = \sum_{L_\beta} C'_{ij}(K_\alpha, L_\beta) \left[ \left< X_{ij}(K_\beta, L_\beta) \right> \right]^T \quad \forall K_\alpha, K_\beta, \tag{3.1.11c} \]

3. **CI Vector Updating**

\[ 2'\sigma'(K_\alpha, K_\beta) = \sum_{ij} 2'\sigma'_{ij}(K_\alpha, K_\beta) \quad \forall K_\alpha, K_\beta. \tag{3.1.11d} \]

We have noticed in Eq. 3.1.11b that the condition given in Eq. 3.1.9 is not applied. This is because this vector is built from the strings with only one spin type. It doesn’t consider the combination between \( \alpha \)- and \( \beta \)-strings. As indicated in Chapter 1, string-based basic linear algebra routines that are very efficient can be used to carry out the matrix multiplication step (Eq. 3.1.11c). However, we noticed in Olsen’s RASSCF method, the advantage of the model space concept is not used. The matrix multiplication is carried out
3. Algorithm for the Direct RASSCF Method (Part I)

for the full RAS active space. Therefore, although there are many different types of orbital integrals in the RASSCF wavefunction (as mentioned above), the same matrix multiplication scheme is used for every orbital integral type.

According to the review above, we know the current RASSCF implementation in Gaussian updates the CI vector in the same way as it did for CASSCF wavefunction. We therefore would like to know whether it is possible to convert the method developed in the last chapter into the RASSCF wavefunction to improve the performance of the current RASSCF method [44]. Since both the current CASSCF and RASSCF methods are string-based methods, it is absolutely possible to convert the matrix multiplication method developed in Chapter 2 to the RASSCF wavefunction. In the last chapter, by reorganizing the order of the summation in a slightly different way from Olsen’s method, the newly developed CASSCF matrix multiplication method improves the performance of the current CAS implementation to a significant degree. Still using the second term of Eq. 1.7.7 as example, by converting the factorized CASSCF matrix multiplication method of Chapter 2 to the RASSCF wavefunction, the CI vector is updated as,

\[ 2\sigma'(K_\alpha, K_\beta) = \sum_{ij} \sum_{J_{\beta}} \langle K_\alpha | E_{ij}^{\alpha} | J_{\alpha} \rangle \langle K_{\beta} | J_{\beta} \rangle \sum_{kl} \langle ij | kl \rangle \langle J_{\beta} | E_{kl}^{\beta} | L_{\beta} \rangle C_{J_{\alpha} L_{\beta}}^{n-1} \forall K_\alpha, K_\beta. \]  

(3.1.12)

And the matrix multiplication method for this kind of summation arrangement is implemented as,

1. Matrix Generation

\[ X_{ij}(J_{\beta}, L_{\beta}) = \sum_{kl} \langle ij | kl \rangle \langle J_{\beta} | E_{kl}^{\beta} | L_{\beta} \rangle, \]  

(3.1.13a)

2. Matrix Multiplication

\[ M_{ij}(J_{\beta}, J_{\alpha}) = \sum_{L_{\beta}} X_{ij}(J_{\beta}, L_{\beta}) \left[ C(J_{\alpha}, L_{\beta}) \right]^T \forall J_{\alpha}, J_{\beta}, \]  

(3.1.13b)

3. CI Vector Updating

\[ 2\sigma'_g(K_\alpha, K_\beta) = \sum_{J_{\alpha}} \langle K_\alpha | E_{ij}^{\alpha} | J_{\alpha} \rangle \left[ M_{ij}(K_{\beta}, J_{\alpha}) \right]^T \forall K_{\alpha}, K_{\beta}, \]  

(3.1.13c)

\[ 2\sigma'(K_\alpha, K_\beta) = \sum_{ij} 2\sigma'_g(K_\alpha, K_\beta) \forall K_\alpha, K_{\beta}, \]  

(3.1.13d)
The reason for why we organize the order of the summations in a different way to Olsen 
*et al* in Chapter 2 is outlined towards the end of Section 1.7. Basic linear algebra routines 
are used to compute Eq. 3.1.13b.

In Chapter 2, only one type of orbital integral is used, such as within the orbitals of the 
CASSCF active space. Thus the matrix multiplication approach is fully applicable for 
wavefunctions with a single type of orbital integral. The RASSCF wavefunction, as 
stated above, has many more types of orbital integrals due to the division of the CAS 
active space into three subspaces. There is, therefore, a possibility that the factorized 
matrix multiplication scheme is not suitable for some orbital integral cases and this 
should be analysed to determine whether this is the case. Thus in this chapter, along with 
the introduction of the basic background of the RASSCF theory, the applicability of the 
factorized matrix multiplication scheme to the many different RASSCF orbital integral 
types is analyzed first. Indeed we have discovered that only several integral types are 
suitable to be modified by using the factorized matrix multiplication approach. It would 
be better to implement the unfactorized method for all other integral types. Thus a hybrid 
method, which combines factorized method and unfactorized method, will be required to 
improve the performance of the current method, which uses unfactorized method for all 
the integral types.

In this chapter, based on the non-zero matrix elements generation scheme used in the 
current method, we have mainly focused on converting the matrix multiplication method 
developed in Chapter 2 to the suitable RASSCF orbital integral types. The 
implementation for the other integral types using the unfactorized method is given in the 
following chapter. This hybrid method should perform much better than the current 
RASSCF method due to the elements in the CI vector will be accessed in a more 
predictable and sequential order in the new method. Thus we can predict the new method 
would perform much faster than the current RASSCF implementation in *Gaussian*.

Since we will focus on applying the matrix multiplication method developed in Chapter 2 
to suitable orbital integral types in the RASSCF wavefunction in this chapter, it is
organized in the following way. In Section 3.2 the factorized CASSCF matrix-multiplication method is reviewed. In order to illustrate the concept that is used in the RASSCF method above, we have used similar notation as the CASSCF wavefunction for the strings and configurations. However, if we continue to use this notation to present RASSCF strings and configurations, it will cause confusions because of the introduction of occupation restrictions. Therefore, a notation for the strings and configurations used in the RASSCF wavefunction based on the CASSCF notation is defined in Section 3.3. Based on this notation, in this section, the RASSCF theory, and more details about the model space concept are also introduced. As described above, due to the occupation restrictions the RASSCF wavefunction has many more different types of orbital integrals than the CASSCF wavefunction that has only one type of orbital integral. These different integral types, as well as the analysis of the applicability of the factorized matrix multiplication method to these types, are also given in Section 3.3. The suitable integral types for factorized matrix multiplication method are hence determined. Since the weights of the contribution to the CI vector updating of these integral types are different from each other, this is discussed at the end of Section 3.3. We have found that the suitable cases are also the most time-consuming integral cases of the current RASSCF method. With all the above background knowledge, the new algorithm, which is to improve the performance of these time consuming cases is described in Sections 3.4. Section 3.5 describes the details of the numerical values that will be used in this new method. Then the implementation details of the algorithms introduced in Section 3.4 can be found in Section 3.6. The parallel scheme of the new algorithms is discussed in Section 3.7. Since the new method is focused on improving the performance of the current method, in Section 3.8 the performance analysis of the newly developed method by using one relatively large testing system is given. A brief summary of what has been done in this chapter can be found in Section 3.9.
3.2 CASSCF Matrix Multiplication Algorithm Review

We will start with a brief review of the factorized matrix multiplication developed for the CASSCF wavefunction first. The CASSCF wavefunction can be regarded as a special case of the RASSCF wavefunction, of which no RAS1 and RAS3 subspaces are given. As described in Chapter 1 the main step of the MCSCF method is to solve the CI eigenvalue problem (Eq. 3.1.6) Therefore, for both of the CASSCF and the RASSCF method, in order to find the lowest eigenvalues and the eigenvectors of the CI eigenvalue problem, some iterative eigenvector procedures, such as the methods of Lanczos [103] or Davidson [104], are used. In this iterative process, one should not only consider how to generate the non-zero Hamiltonian matrix elements (Eq. 3.1.3) efficiently, but also how to use the obtained elements in an efficient way to implement Eq. 3.1.6. As the non-zero element generation process of the current MCSCF implementations is already very efficient, then the performance of how to use the non-zero elements that are obtained becomes our main concern. As described above and discussed in Chapter 1, based on the way of implementing the two-electron symbolic matrix elements, \( B \) (Eq. 3.1.2), there are two types of methods to carry out Eq. 3.1.6. One is called the unfactorized method, which is the current MCSCF implementations in Gaussian (Eqs. 3.1.8). The other one is called the factorized method, which is represented by the Olsen’s method [34] (Eqs. 3.1.10 and 3.1.11).

However, in Chapter 2, we have reorganized the summations in Eq. 3.1.10 to construct our matrix multiplication. The advantages of this reorganization has been summarised at the end of Section 1.7. Here we will only review this newly developed method for the CASSCF wavefunction. In the CASSCF matrix multiplication method developed in Chapter 2, Eq. 3.1.2 has been factorized by introducing an intermediate state \( |J\rangle \) as,

\[
B_{ijkl}^{KL} = \langle K | E_{ij} E_{kl} - \delta_{jk} E_{il} | L \rangle = \langle K | \sum_J E_{ij} | J \rangle \langle J | E_{kl} | L \rangle - \delta_{jl} \langle K | E_{il} | L \rangle ,
\]

(3.2.1)

where the first term of Eq.3.2.1 is a bilinear contribution, which is a product of two linear contributions, and the second term is a linear contribution to the CI vector updating.
Based on this, we can write Eq. 3.1.6 as a combination of linear and bilinear contributions:

$$\sigma_K^n = \sigma_K^{\text{linear}} + \sigma_K^{\text{bilinear}}. \quad (3.2.2)$$

After applying the string concept [107,108] to the configurations, $$|K\rangle = |K_\alpha K_\beta\rangle$$, the two terms of Eq. 3.2.2 can be written as,

$$\sigma_{K_\alpha K_\beta}^{\text{linear}} = \sum_{ij} \langle i \mid j \rangle \left( \sum_{L_\alpha} \langle K_\alpha | a_{ia}^\dagger a_{ja} | L_\alpha \rangle \langle K_\beta | L_\beta \rangle + \sum_{L_\beta} \langle K_\beta | a_{i\beta}^\dagger a_{j\beta} | L_\beta \rangle \langle K_\alpha | L_\alpha \rangle \right) + \frac{1}{2} \sum_{ijkl} \langle ij \mid kl \rangle \left( \sum_{L_\alpha} \langle K_\alpha | a_{ia}^\dagger a_{ja} a_{ia}^\dagger a_{ja} | L_\alpha \rangle \langle K_\beta | L_\beta \rangle + \sum_{L_\beta} \langle K_\beta | a_{i\beta}^\dagger a_{j\beta} a_{i\beta}^\dagger a_{j\beta} | L_\beta \rangle \langle K_\alpha | L_\alpha \rangle \right) C(L_\alpha, L_\beta). \quad (3.2.3)$$

and

$$\sigma_{K_\alpha K_\beta}^{\text{bilinear}} = \frac{1}{2} \sum_{ijkl} \langle ij \mid kl \rangle \left( \sum_{L_\alpha} \langle K_\alpha | a_{ia}^\dagger a_{ja} | L_\alpha \rangle \sum_{L_\beta} \langle K_\beta | a_{i\beta}^\dagger a_{j\beta} | L_\beta \rangle \right) C(L_\alpha, L_\beta). \quad (3.2.4)$$

By reorganizing the order of summations in Eq. 3.2.4, the bilinear contribution can be rewritten as:

$$\sigma_{K_\alpha K_\beta}^{\text{bilinear}} = \frac{1}{2} \sum_{ij} \sum_{L_\alpha} \langle K_\alpha | a_{ia}^\dagger a_{ja} | L_\alpha \rangle \sum_{L_\beta} \sum_{kl} \langle ij \mid kl \rangle \langle K_\beta | a_{i\beta}^\dagger a_{j\beta} | L_\beta \rangle C(L_\alpha, L_\beta) + \frac{1}{2} \sum_{ij} \sum_{L_\alpha} \langle K_\alpha | a_{ia}^\dagger a_{ja} | L_\alpha \rangle \sum_{L_\beta} \sum_{kl} \langle ij \mid kl \rangle \langle K_\beta | a_{i\beta}^\dagger a_{j\beta} | L_\beta \rangle C(L_\alpha, L_\beta), \quad (3.2.5)$$

which can be split into the following three parts:

1. **Matrix generation**:

$$X_{ij}(K_\xi, L_\zeta) = \sum_{kl} \langle ij \mid kl \rangle \langle K_\xi | a_{k\xi}^\dagger a_{l\xi} | L_\zeta \rangle, \quad (3.2.6a)$$

2. **Matrix multiplication**:

$$M_{ij}(K_\xi, L_\eta) = \sum_{L_\zeta} X_{ij}(K_\xi, L_\zeta) C(L_\zeta, L_\eta), \quad (3.2.6b)$$


3. CI vector updating:

\[
\sigma^\text{bilinear}(K_\xi, K_\gamma) = \sum_{ij} \sum_{L_\gamma} \langle K_\gamma | a_i^\dagger a_j^\dagger | L_\gamma \rangle M_{ij} \langle K_\xi, L_\gamma \rangle \quad (\xi, \gamma \in \{\alpha, \beta\}). \tag{3.2.6c}
\]

Equations 3.2.6 are the key equations of the matrix multiplication method that is implemented to speed up the current CASSCF method based on the factorization idea. Having this idea in mind, we can now add an occupation restricted RAS1 subspace and an occupation restricted RAS3 subspace to the CASSCF active space (the occupation unrestricted RAS2 subspace) to form the RASSCF wavefunction. The basic CI vector updating theory will thus be similar as the CASSCF wavefunction (Eq. 3.1.6) but with the occupation restrictions in the added RAS1 and RAS3 subspaces, which can be reflected in the notation of the strings. Therefore, in order not cause confusion, the notations for the RASSCF wavefunction must be defined first.

3.3 RASSCF Notation and Theory

In this section, we will first define the notation of the strings and configurations used in the RASSCF wavefunction. Based on these we can derive the \( \sigma \) vector updating process for the RASSCF wavefunction as well as the RAS model space concept. Therefore, we will start with the key point to the RAS theory – the notation of the RASSCF configurations.

3.3.1 RASSCF Configurations Notation

In the CASSCF wavefunction, since there are no occupation restrictions of the configurations, we can simply denote the strings as \( |K_\alpha\rangle, |K_\beta\rangle \). The \( \alpha \)-strings can be freely combined with the \( \beta \)-strings to form the corresponding configurations \( |K\rangle = |K_\alpha K_\beta\rangle \). However, in the RASSCF wavefunction, since there are occupation restrictions in the RAS1 and RAS3 subspaces, the \( \alpha \)-strings can no longer be freely combined with the \( \beta \)-strings. For a valid configuration to the RASSCF wavefunction, the
3. Algorithm for the Direct RASSCF Method (Part I)

RAS1 subspace can contain a maximum number of holes, denoted as MxHole, and the RAS3 subspaces can contain a maximum number of electrons, marked as MxElec. Because of these occupation restrictions, for certain RASSCF strings, there will also be a certain number of holes in the RAS1 subspace and a certain number of electrons in RAS3 subspace. Based on this, we can define:

- \( i_h \): the number of holes in the RAS1 subspace of one RASSCF string, and
- \( i_e \): the number of electrons in the RAS3 subspace of the RASSCF string.

Obviously, the values of \( i_h \) and \( i_e \) fulfill \( 0 \leq i_h \leq \text{MxHole} \) and \( 0 \leq i_e \leq \text{MxElec} \) respectively. Thus we can mark a RASSCF string as \( K_{\xi}^{i_h,i_e} \) (\( \xi \in \{\alpha,\beta\} \)), where the subscript indicates the spin type of the electrons of this string and the superscript gives the number of holes and electrons in the RAS1 and RAS3 subspaces of this string. This notation can be treated as a combination of the occupation pattern of all three subspaces:

\[
K_{\xi}^{i_h,i_e} = K_{\xi}^{\text{RAS1},i_h} + K_{\xi}^{\text{RAS2}} + K_{\xi}^{\text{RAS3},i_e},
\]

(3.3.1)

where \( K_{\xi}^{\text{RAS1},i_h} \) indicates the occupation pattern of the \( M (\text{RAS1}) - i_h \) \( \xi \)-electrons in the RAS1 subspace, \( K_{\xi}^{\text{RAS2}} \) denotes the occupation pattern of the \( N_{\xi} - M (\text{RAS1}) + i_h - i_e \) electrons in the \( M (\text{RAS2}) \) orbitals (\( N_{\xi} \) is the overall number of \( \xi \)-electrons), and \( K_{\xi}^{\text{RAS3},i_e} \) gives the occupation pattern of the \( i_e \) \( \xi \)-electrons in the RAS3 subspace. For example, assuming we have one \( \alpha \)-string that has \( N_{\alpha} = 6 \), \( M (\text{RAS1}) = M (\text{RAS2}) = M (\text{RAS3}) = 4 \) and \( i_h = i_e = 1 \), we can have the binary representation of the corresponding substrings as:

\[
K_{\alpha}^{\text{RAS1},1} \in \begin{pmatrix} 1110 \\ 1101 \\ 1011 \\ 0111 \end{pmatrix}, \quad K_{\alpha}^{\text{RAS2}} \in \begin{pmatrix} 0011 \\ 0101 \\ 1001 \\ 0110 \\ 1010 \\ 1100 \end{pmatrix}, \quad \text{and} \quad K_{\alpha}^{\text{RAS3},1} \in \begin{pmatrix} 0001 \\ 0010 \\ 0100 \\ 1000 \end{pmatrix}.
\]

The combination of these substrings can generate in total 96 global strings, \( K_{\alpha}^{1,1} \).

According to this definition, we can treat the CASSCF wavefunction as a special case of
the RASSCF wavefunction that has no RAS1 and RAS3 subspaces defined \((i_h = 0, i_e = 0)\). Thus we can also index the strings in the CASSCF wavefunction as \(K^{0,0}_\xi = K^{RAS^2}_\xi\).

Since there are occupation restrictions in the RAS wavefunction, according to the values of \(i_h\) and \(i_e\), we can classify the RASSCF strings into several categories \([87]\). The total number of categories for RASSCF strings is a function of MxHole and MxElec and is defined as,

\[
n_{\text{Cat}} = (\text{MxHole} + 1) \cdot (\text{MxElec} + 1).
\]

Each string category is labelled by the values of \(i_h\) and \(i_e\) via \(\text{Cat}_\xi(i_h, i_e)\) \((\xi \in \{\alpha, \beta\})\).

For example, for a RAS system that with MxHole = MxElec = 1, there will be in total 4 \(\xi\)-string categories: \(\text{Cat}_\xi(0,0)\), \(\text{Cat}_\xi(0,1)\), \(\text{Cat}_\xi(1,0)\), and \(\text{Cat}_\xi(1,1)\). And for an MxHole = MxElec = 2 RAS system, the number of string categories becomes 9:

\(\text{Cat}_\xi(0,0)\), \(\text{Cat}_\xi(0,1)\), \(\text{Cat}_\xi(0,2)\), \(\text{Cat}_\xi(1,0)\), \(\text{Cat}_\xi(1,1)\), \(\text{Cat}_\xi(1,2)\), \(\text{Cat}_\xi(2,0)\), \(\text{Cat}_\xi(2,1)\), and \(\text{Cat}_\xi(2,2)\). Therefore, the allowed combinations between \(\alpha\)- and \(\beta\)-strings becomes the allowed combinations between the corresponding string categories.

The combination between a RAS \(\alpha\)-string, \(K^{i_h,i_e}_\alpha\), and a RAS \(\beta\)-string, \(K^{i_h',i_e'}_\beta\), have the conditions:

\[
\begin{align*}
i_h + i'_h &\leq \text{MxHole} \\
i_e + i'_e &\leq \text{MxElec}
\end{align*}
\]

These conditions are also valid for the combination between a \(\alpha\)-string category, \(\text{Cat}_\alpha(i_h, i_e)\), and a \(\beta\)-string category, \(\text{Cat}_\beta(i'_h, i'_e)\). Therefore, if we write the CI vector as a matrix where the rows are indexed by the \(\alpha\)-string categories and the columns are indexed by the \(\beta\)-string categories, we can have:
3. Algorithm for the Direct RASSCF Method (Part I)

Single excitation (MxHole=MxElec=1):

\[
C = \begin{pmatrix}
\begin{array}{c}
\text{Cat}_\alpha(0,0) \\
\text{Cat}_\alpha(0,1) \\
\text{Cat}_\alpha(1,0) \\
\text{Cat}_\alpha(1,1)
\end{array}
\end{pmatrix}
\begin{pmatrix}
\begin{pmatrix}
K_{\alpha}^{0,0}K_{\beta}^{0,0} \\
K_{\alpha}^{0,0}K_{\beta}^{0,1} \\
K_{\alpha}^{0,1}K_{\beta}^{0,0} \\
K_{\alpha}^{0,1}K_{\beta}^{0,1}
\end{pmatrix} & \begin{pmatrix}
K_{\alpha}^{0,0}K_{\beta}^{1,0} \\
K_{\alpha}^{0,1}K_{\beta}^{1,0}
\end{pmatrix}
\end{pmatrix}, \quad (3.3.4a)
\]

each element in Eq. 3.3.4a is a sub-block matrix of the CI vector (dimensions are defined by the length of the corresponding string categories); and the CI vector matrix for a Double excitation (MxHole = MxElec = 2) is:

\[
C = \begin{pmatrix}
\begin{array}{c}
\text{Cat}_\alpha(0,0) \\
\text{Cat}_\alpha(0,1) \\
\text{Cat}_\alpha(1,0) \\
\text{Cat}_\alpha(1,1)
\end{array}
\end{pmatrix}
\begin{pmatrix}
\begin{pmatrix}
K_{\alpha}^{0,0}K_{\beta}^{0,0} \\
K_{\alpha}^{0,0}K_{\beta}^{0,1} \\
K_{\alpha}^{0,1}K_{\beta}^{0,0} \\
K_{\alpha}^{0,1}K_{\beta}^{0,1}
\end{pmatrix} & \begin{pmatrix}
K_{\alpha}^{0,0}K_{\beta}^{1,0} \\
K_{\alpha}^{0,1}K_{\beta}^{1,0} \\
K_{\alpha}^{0,2}K_{\beta}^{0,0} \\
K_{\alpha}^{0,2}K_{\beta}^{0,1}
\end{pmatrix} & \begin{pmatrix}
K_{\alpha}^{0,0}K_{\beta}^{1,1} \\
K_{\alpha}^{0,1}K_{\beta}^{1,0} \\
K_{\alpha}^{0,2}K_{\beta}^{1,0}
\end{pmatrix} & \begin{pmatrix}
K_{\alpha}^{0,0}K_{\beta}^{2,0} \\
K_{\alpha}^{0,1}K_{\beta}^{2,0} \\
K_{\alpha}^{0,2}K_{\beta}^{2,0} \\
K_{\alpha}^{0,2}K_{\beta}^{2,1}
\end{pmatrix}
\end{pmatrix}, \quad (3.3.4b)
\]

Similarly, each element in Eq. 3.3.4b is also a block matrix of the CI vector of which the dimension is defined by the length of the corresponding string categories. The numerical values of the strings and the length of the corresponding string categories will be discussed in detail later.

As mentioned above, the CASSCF wavefunction can be regarded as a special case of the RASSCF wavefunction, thus when the number of the active space of a CAS calculation equals the number of the RAS2 orbitals and the number of the active \( \xi \) -electrons of this CAS calculation is the same as \( N_\xi - M(RAS1) + i_h - i_e \), the CI vector of the CAS calculation becomes a block (the \( K_{\alpha}^{0,0}K_{\beta}^{0,0} \) block) of the CI vector of the RASSCF

149
As we divided the CI vector updating into linear contribution and bilinear contribution in the CASSCF method, but with the notations defined above for the RASSCF configuration and strings, the following, all the equations will be in the same form as in the CASSCF method, but with the notations defined above for the RASSCF configuration and strings.

### 3.3.2 RASSCF CI Vector Updating

Similar to the theory described in Section 3.2, the basic theory of the RASSCF CI vector updating is also to solve the CI eigenvalue problem in an iterative way. The only difference between the RASSCF wavefunction and the CASSCF wavefunction is that in the RASSCF method, the occupation restrictions are introduced. Therefore, in the following, all the equations will be in the same form as in the CASSCF method, but with the notations defined above for the RASSCF configuration and strings.

The \( \sigma \) vector updating process is still the same as in CASSCF:

\[
\sigma^n_k = \sum_L \left( \sum_{ij} (i|j) A_{ij}^{KL} + \frac{1}{2} \sum_{ijkl} (ij|kl) B_{ijkl}^{KL} \right) C_{L}^{n-1}.
\] (3.3.5)

But the symbolic matrix elements in the RASSCF wavefunction now become:

\[
A_{ij}^{K_{ij}, \alpha, \beta} = \begin{pmatrix} \langle K_{ij}^{h_i, i^e} | E_{ij}^\alpha | L_{ij}^{\alpha, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\alpha, \sigma} \rangle + \langle K_{ij}^{h_i, i^e} | E_{ij}^\beta | L_{ij}^{\beta, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\beta, \sigma} \rangle \end{pmatrix},
\] (3.3.6)

\[
B_{ijkl}^{K_{ij}, \alpha, \beta, \alpha', \beta'} = \begin{pmatrix} \langle K_{ij}^{h_i, i^e} | a_{ij}^+ a_{kj}^+ a_{ia} a_{ja} | L_{ij}^{\alpha, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\alpha, \sigma} \rangle + \langle K_{ij}^{h_i, i^e} | a_{ij}^+ a_{kj}^+ a_{ia} a_{ja} | L_{ij}^{\beta, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\beta, \sigma} \rangle \\ + \langle K_{ij}^{h_i, i^e} | a_{ij}^+ a_{kj}^+ a_{ia} a_{ja} | L_{ij}^{\alpha, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\alpha, \sigma} \rangle \langle E_{ij}^\alpha | L_{ij}^{\alpha, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\alpha, \sigma} \rangle + \langle K_{ij}^{h_i, i^e} | a_{ij}^+ a_{kj}^+ a_{ia} a_{ja} | L_{ij}^{\beta, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\beta, \sigma} \rangle \langle E_{ij}^\alpha | L_{ij}^{\alpha, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\alpha, \sigma} \rangle \end{pmatrix}.
\] (3.3.7)

As we divided the CI vector updating into linear contribution and bilinear contribution in the CASSCF method, we can also separate the RASSCF CI vector updating process as linear and bilinear contributions:

\[
\sigma^{linear}_{K_{ij}, \alpha, \beta} = \sum_{L, \alpha, \beta} \left( \sum_{ij} (i|j) \left( \langle K_{ij}^{h_i, i^e} | E_{ij}^\alpha | L_{ij}^{\alpha, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\alpha, \sigma} \rangle + \langle K_{ij}^{h_i, i^e} | E_{ij}^\beta | L_{ij}^{\beta, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\beta, \sigma} \rangle \right) \right)
\]

\[
+ \frac{1}{2} \sum_{ijkl} (ij|kl) \left( \langle K_{ij}^{h_i, i^e} | a_{ij}^+ a_{kj}^+ a_{ia} a_{ja} | L_{ij}^{\alpha, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\alpha, \sigma} \rangle \right) + \left( \langle K_{ij}^{h_i, i^e} | a_{ij}^+ a_{kj}^+ a_{ia} a_{ja} | L_{ij}^{\beta, \sigma} \rangle \langle K_{ij}^{h_i, i^e} | L_{ij}^{\beta, \sigma} \rangle \right) \right] C_{L}^{n-1},
\] (3.3.8)
3. Algorithm for the Direct RASSCF Method (Part I)

$$\sigma_{\text{Bilinear}}^{\text{K_{\alpha} \rightarrow \beta} \ K_{\beta} ^{\alpha \rightarrow \beta}} = \frac{1}{2} \sum_{ijkl} \sum_{ijkl} (ij|kl) \left( \langle K_{\alpha} ^{i,j} | E_{ij} ^{\alpha} | L_{\alpha} ^{i,j} \rangle \langle K_{\beta} ^{i,j} | E_{ij} ^{\beta} | L_{\beta} ^{i,j} \rangle \right) +$$

$$\left( \langle K_{\beta} ^{i,j} | E_{ij} ^{\beta} | L_{\beta} ^{i,j} \rangle \langle K_{\alpha} ^{i,j} | E_{ij} ^{\alpha} | L_{\alpha} ^{i,j} \rangle \right) C_{i,j,k,l} ^{\alpha,\beta},$$

(3.3.9)

where $i,j,k,l$ are the full RAS space (global) orbital indices, $i_h,i_e$ indicate the number of holes and electrons in the RAS1 and RAS3 subspaces respectively with the following relations,

$$i_h + i_e \leq \text{MxHole}, \quad i_e + i_e' \leq \text{MxElec}.$$  

(3.3.10)

As we can see, with the introduction of the occupation restriction (Eq. 3.3.10), the process of the CI vector updating of the RASSCF wavefunction becomes more complicated than the process of the CASSCF wavefunction. However, if we do not consider the occupation restriction, Eq. 3.3.9 becomes identical to Eq. 3.2.4. Thus we can also factorize Eq. 3.3.9 in the way of matrix multiplication as,

$$\sigma_{\text{Bilinear}}^{\text{K_{\alpha} \rightarrow \beta} \ K_{\beta} ^{\alpha \rightarrow \beta}} = \frac{1}{2} \sum_{ij} \sum_{ij} (ij|kl) \left( \langle K_{\alpha} ^{i,j} | E_{ij} ^{\alpha} | L_{\alpha} ^{i,j} \rangle \sum_{kl} (ij|kl) \langle K_{\beta} ^{i,j} | E_{ij} ^{\beta} | L_{\beta} ^{i,j} \rangle \right) C_{i,j,k,l} ^{\alpha,\beta} +$$

$$\frac{1}{2} \sum_{ij} \sum_{ij} (ij|kl) \left( \langle K_{\beta} ^{i,j} | E_{ij} ^{\beta} | L_{\beta} ^{i,j} \rangle \sum_{kl} (ij|kl) \langle K_{\alpha} ^{i,j} | E_{ij} ^{\alpha} | L_{\alpha} ^{i,j} \rangle \right) C_{i,j,k,l} ^{\alpha,\beta},$$

(3.3.11)

with

$$X_{ij} \left( K_{\xi} ^{i,j} , L_{\xi} ^{i,j} \right) = \sum_{kl} (ij|kl) \langle K_{\xi} ^{i,j} | a_{i,j} ^{\alpha} a_{i,j} ^{\beta} | L_{\xi} ^{i,j} \rangle ,$$

(3.3.12a)

$$M_{ij} \left( K_{\xi} ^{i,j} , L_{\gamma} ^{i,j} \right) = \sum_{kl} X_{ij} \left( K_{\xi} ^{i,j} , L_{\xi} ^{i,j} \right) C \left( L_{\xi} ^{i,j} , L_{\gamma} ^{i,j} \right) ,$$

(3.3.12b)

$$\sigma ^{\text{Bilinear}} \left( K_{\xi} ^{i,j} , K_{\gamma} ^{i,j} \right) = \sum_{ij} \sum_{ij} \left( a_{i,j} ^{\alpha} a_{i,j} ^{\beta} | L_{\xi} ^{i,j} \rangle \right) M_{ij} \left( K_{\xi} ^{i,j} , L_{\gamma} ^{i,j} \right) \left( \xi, \gamma \in \{ \alpha, \beta \} \right).$$

(3.3.12c)

All the orbital indices in Eqs. 3.3.12 are over the full RAS space. If we carry out the matrix multiplication as given in Eqs. 3.3.12, the number of $ij$ index pairs in Eq. 3.3.12c will be a function of the size of the full RAS space. This might become a bottleneck to the performance when the full RAS space is larger than certain level. However, if the
model space concept is used, this potential bottleneck could be eliminated, because the size of the model space is only dependent on the size of the RAS2 subspace and the values of MxHole and MxElec. No matter how large the full RAS space is, if the size of RAS2 subspace and the values of MxHole and MxElec don’t change, the size of the model space would be constant. We have briefly reviewed what a model space would look like. Now we should have a more detailed overview of the model space concept.

### 3.3.3 Concepts of Model Space and Expanding

Because the orbital occupation restriction in the RAS1 and RAS3 subspaces makes the RAS calculation more complicated, to simplify this, we seek a space that can represent the full RAS space but with no orbital restrictions applied. As introduced in Section 3.1, the model space concept can be used for this purpose. Thus we will describe the model space concept [44,95] in more details in this section.

Suppose we would like to construct a RAS wavefunction using the determinantal basis, with a maximum number of holes (MxHole) allowed in the RAS1 space and a maximum number of electrons (MxElec) allowed in the RAS3 space, a model RAS space may then be defined with $M^m$ orbitals (the superscript $m$ stands for model) as,

$$M^m = M \left( RAS2 \right) + MxHole + MxElec \, , \tag{3.3.13}$$

and $N^m = N^m_\alpha + N^m_\beta$ electrons where

$$N^m_\alpha = N_\alpha - M \left( RAS1 \right) + MxHole \, , \tag{3.3.14a}$$

$$N^m_\beta = N_\beta - M \left( RAS1 \right) + MxHole \, . \tag{3.3.14b}$$

The remaining orbitals excluded from this model space will always be doubly occupied in the RAS1 subspace and empty in the RAS3 subspace. In order to avoid confusion, we label the orbitals in the model space as,

- model space: $w, x, y, z$.

The most striking feature of the model string space, as we mentioned in Section 3.1, is the fact that, although it represents restricted string spaces, it is entirely unrestricted and
in fact identical to those used in a CASSCF problem with \( M(CAS) = M'(RAS) \) orbitals and \( N(CAS) = N'(RAS) \) electrons. Therefore, one can use the well defined “reduced string list” concept described in Chapter 2 to generate all the non-zero model string pairs \( \{ K^m_\xi, L^m_\xi \} \) and their associated model symbolic matrix elements. Thus the CI vector updating process for the model space becomes identical to a CAS problem,

\[
\sigma_{K^m_\xi K^m_\eta} = \sum_{L^m_\alpha L^m_\beta} \left[ \sum_{wx} \langle w | x \rangle A_{wx}^{K^m_\alpha K^m_\beta L^m_\alpha L^m_\beta} + \frac{1}{2} \sum_{wxyz} \langle w xy | z \rangle B_{wxyz}^{K^m_\alpha K^m_\beta L^m_\alpha L^m_\beta} \right] C_{L^m_\alpha L^m_\beta} , \quad (3.3.15)
\]

\[
\sigma_{linear}^{\text{Bilinear}} = \frac{1}{2} \sum_{L^m_\alpha L^m_\beta} \sum_{wx} \sum_{wxyz} \langle w | x \rangle \left( \langle K^m_\alpha | E^\alpha_{wx} | L^m_\alpha \rangle \langle K^m_\beta | E^\beta_{wy} | L^m_\beta \rangle + \langle K^m_\beta | E^\beta_{wx} | L^m_\beta \rangle \langle K^m_\alpha | E^\alpha_{wy} | L^m_\alpha \rangle \right) C_{L^m_\alpha L^m_\beta} , \quad (3.3.16)
\]

\[
\sigma_{Bilinear}^{\text{Bilinear}} = \frac{1}{2} \sum_{L^m_\alpha L^m_\beta} \sum_{wxyz} \langle w | x \rangle \left( \langle K^m_\alpha | E^\alpha_{wx} | L^m_\alpha \rangle \langle K^m_\beta | E^\beta_{wy} | L^m_\beta \rangle + \langle K^m_\beta | E^\beta_{wx} | L^m_\beta \rangle \langle K^m_\alpha | E^\alpha_{wy} | L^m_\alpha \rangle \right) C_{L^m_\alpha L^m_\beta} , \quad (3.3.17)
\]

where \( |K^m_\xi\rangle \) and \( |L^m_\xi\rangle \) are the model \( \xi\)-strings, and \( \langle wx | yz \rangle \) indicates the model space orbital integral. There is no orbital occupation restriction for this model space. Thus all the concepts that were used in the CASSCF wavefunction problem in Chapter 2 would be applicable here. Similar to the factorized CASSCF matrix multiplication method, we can also factorize Eq. 3.3.17 as,

\[
\sigma_{Bilinear}^{\text{Bilinear}} = \frac{1}{2} \sum_{wx} \sum_{L^m_\alpha} \langle K^m_\alpha | E^\alpha_{wx} | L^m_\alpha \rangle \sum_{L^m_\beta} \sum_{YZ} \langle L^m_\beta | E^\beta_{wy} | L^m_\beta \rangle \sum_{L^m_\gamma} \langle w | x \rangle \left( \langle K^m_\alpha | E^\alpha_{wy} | L^m_\alpha \rangle \langle K^m_\beta | E^\beta_{wy} | L^m_\beta \rangle \right) C_{L^m_\alpha L^m_\beta L^m_\gamma} . \quad (3.3.18)
\]

And the matrix multiplication scheme can be built based on this as,

\[
X^m_{wx} \left( K^m_\xi, L^m_\xi \right) = \sum_{YZ} \langle w | x \rangle \left( \langle K^m_\xi | E^\xi_{YZ} | L^m_\xi \rangle \right) , \quad (3.3.19a)
\]

\[
M^m_{wx} \left( K^m_\xi, L^m_\gamma \right) = \sum_{L^m_\xi} X^m_{wx} \left( K^m_\xi, L^m_\xi \right) C \left( L^m_\xi, L^m_\gamma \right) , \quad (3.3.19b)
\]

\[
\sigma^{\text{Bilinear}} \left( K^m_\xi, K^m_\gamma \right) = \sum_{wx} \sum_{L^m_\xi} \langle K^m_\xi | E^\gamma_{wx} | L^m_\gamma \rangle M^m_{wx} \left( K^m_\xi, L^m_\gamma \right) \left( \xi, \gamma \in \{ \alpha, \beta \} \right) . \quad (3.3.19c)
\]
We have already mentioned that the unrestricted character of the model string space may be exploited. The problem of finding non-zero contribution model string pairs closely resembles the corresponding problem for full CI. Thus the efficient reduced string concept used for the CASSCF method can be used here. Based on this, the generation of the model string pairs and their associated model symbolic matrix elements is simple. The current RASSCF implementation uses Eq. 3.3.17, which is unfactorized, to generate the bilinear contribution. And Eqs. 3.3.19 will be the main method we are going to implement in this chapter. However, in order to construct a valid contribution to the CI vector updating of the RASSCF wavefunction, the model orbital indices appearing in either Eqs. 3.3.16 and 3.3.17 or Eqs. 3.3.19 must be expanded to the full RAS space. In other words, it must be determined how to generate the full RAS space orbital integrals, \( (i|j) \) and \( (ij|kl) \), from the model orbital indices, \( \{w,x,y,z\} \), the full RAS string pairs, and their associated full RAS space symbolic matrix elements, \( A^b_{ij}K^h_{il} \) and \( B^b_{ijkl}L^h_{lm} \), from the model string pairs and their associated model symbolic matrix elements respectively. To achieve this, we have adopted the efficient expanding method, termed as propagation rules, of Klene’s method [44], to carry out the expansion:

\[
K^m_\gamma \rightarrow K^{h,i'}_\gamma, \quad L^m_\gamma \rightarrow L^{k,k'}_\gamma, \quad B^{m,m'}_\omega \rightarrow B^{KL}_{ijkl}, \quad \omega \rightarrow (ij|kl). \quad (3.3.20)
\]

The details about this concept can be found in [44]. However, for the purpose of easy understanding, several examples of this concept are given in the numerical discussion section later in this chapter. Here we will only investigate the sign of the global symbolic matrix elements after the expansion process (Eq. 3.3.20).

As indicated in Eq. 3.3.20, the model symbolic matrix elements must be expanded to the full RAS space with the expansion process of the orbital integrals and the model string pairs. The sign of the full RAS space symbolic matrix elements after the expansion of the model symbolic matrix elements is defined, taking the 2-electron linear contribution for example, as:

\[
\langle K^{h,i'}_\gamma | a^\dagger_{lz}a^\dagger_{lz}a_{iz}a_{iz} | L^{k,k'}_\xi \rangle = B^{KL}_{ijkl} \Big| \omega_{xyz} \Big| \rightarrow (ij|kl) \times sgn^{RASS}_{ijkl} \times sgn^{RASS}_{ijkl} \quad (3.3.21)
\]
where the orbital indices in Eq. 3.3.21, \( i j k l \), denote full RAS space indices, and the factors \( \text{sgn}_{jkl}^{RAS} \) are potentially variables that can take the values \( \pm 1 \). Thus,

\[
B_{ijkl}^{KL} = (-1)^n B_{wxyz}^{k'n'} = (3.3.22)
\]

However, when all four model orbital indices, \( \{ w, x, y, z \} \), are in the RAS2 subspace, we will always have,

\[
B_{ijkl}^{KL} = B_{wxyz}^{k'n'} \  \{ w, x, y, z \} \text{in RAS2}. \quad (3.3.23)
\]

The reason for this is because the whole RAS2 subspace is included in the model space. When expanding the model RAS2 subspace to the full RAS space, only one result can be obtained, which is the model RAS2 orbital indices themselves. Thus when all four model orbital indices are in the RAS2 subspace, the extension of RAS1 and RAS3 subspaces will simply result in a list of consecutive subspace indices, and both model symbolic matrix elements, \( B_{wxyz}^{k'n'} \), and integrals, \( (ij)|kl) \), will remain constant for all contributions derived from a model space string pair. However, once any of the model orbital indices drops in either RAS1 model space or RAS3 model space, the sign of the symbolic matrix elements might change and Eq. 3.3.22 applies. The details about how to calculate the numerical values of the signs will be discussed later with the numerical values of the strings and the string categories. As we now have a rough idea of how the model string pairs are expanded, we shall move back to the CI vector updating process again.

As described above, the current RASSCF method uses Eqs. 3.3.16 and 3.3.17, which is the unfactorized way, to update the CI vector directly. That is once a pair of model strings is obtained, it is directly expanded to the full RAS space to generate the corresponding set of full RAS space string pairs and the associated non-zero symbolic matrix elements. Then these non-zero elements are used directly to update the elements in the CI vector according to the obtained indices of the full RAS string pairs (more details about this process will be given later). Therefore no matter what the orbital integral type is, the current method will use the same direct way to update the CI vector.
However, the RASSCF orbital integral types become crucial in the factorized matrix multiplication method. The reason for this lies in Eq. 3.3.19c,

$$\sigma^{bilinear}(K^m_{\xi}, K^m_{\gamma}) = \sum_{w} \sum_{x} \langle K^m_{\gamma} | E_{ws} | L^m_{\xi} \rangle \sum_{\xi \gamma \in \{\alpha, \beta\}} M^m_{ws}(K^m_{\xi}, L^m_{\gamma}) \xi \gamma \in \{\alpha, \beta\}. \quad (3.3.19c)$$

From this equation we can see that the outer most loop is over the model orbital indices $w$ and $x$. For each $wx$ model orbital index pair, there will be one corresponding intermediate matrix $M^m_{wx}$, which is the product of the matrix multiplication (Eq. 3.3.19b). Since the model orbital indices will be expanded to the full RAS space to generate the real contribution to the CI vector updating, the expanding of $wx$ model orbital index pair will lead to a set of full RAS space orbital index pairs. Each element in this set corresponds to its unique intermediate matrix $M$ (Eq. 3.3.19c). Therefore, the number of matrix multiplication operations for a certain $wx$ model orbital index pair is solely dependent on the number of full RAS orbital index pairs expanded from this model orbital index pair. In order to achieve a good performance, our aim is to keep the number of matrix multiplication operations as low as possible. From Eq. 3.3.13 we can see the RAS2 subspace is fully included in the model space. Therefore, if the $w$ and $x$ model orbital indices are both in the RAS2 subspace, we know the number of expanded full RAS orbital indices is minimized to 1 pair, which is $w$ and $x$ themselves. Based on this, we know the types of expansion of the model orbital integral to the full RAS space are crucial to the matrix multiplicaion method, because we would like to find out if the integral types that have both $w$ and $x$ are in the RAS2 subspace. Thus for carrying out the factorized matrix multiplication method, it is necessary to investigate what the orbital integral types are present in the RASSCF wavefunction.

### 3.3.4 Orbital Integral Types

As described in Section 3.3.3, we know the model orbital integral can represent the full RAS space orbital integral. We also mentioned in Section 3.1 that, due to the introduction of the occupation restrictions in the RAS1 and RAS3 subspaces, more than one type of orbital integrals must be considered for the RASSCF wavefunction. Therefore, in this section, we will mainly focus on investigating the integral types of the RASSCF method.
and their corresponding expansion to the full RAS space. First we should define some labels for the orbitals of the model space and the full RAS space to avoid confusion:

- RAS1: \( p, q, r, s \)
- RAS2: \( i, j, k, l \)
- RAS3: \( a, b, c, d \)
- full model RAS space: \( w, x, y, z \)
- full RAS space (global space): \( i, j, k, l \)
- model RAS1 space: \( \mu, \nu \)
- model RAS3 space: \( \delta, \epsilon \).

Note that the orbital indices of RAS2 are the same as the orbital indices of the full RAS space. This is because after expanding the model space to the full RAS space, the orbital indices of the RAS2 orbitals in the full RAS space will remain the same as they were in the model RAS2 space.

With this orbital label, we can thus define certain specific strings that show the occupation pattern in the RAS1 and RAS3 subspaces. For example, the string \( K^{i_h,i_e}_\xi \) indicates the general case of a string in which there are \( i_h \) holes in the RAS1 subspace and \( i_e \) electrons in the RAS3 subspace. But how do we distinguish two specific strings like \( K^{1,1}_\xi \) and \( K^{1,1}_{\xi'} \)? Are they the same strings? Or are they both representations of model strings or not? To eliminate this confusion, for these specific cases, we need to introduce the orbital labels in the general string expression. So we can label the two specific strings as \( K^{11,\mu\delta}_\xi \) and \( K^{11,\nu\epsilon}_\xi \) to denote these two model strings as different. For presenting certain global string, one only needs to change the model RAS space label to the full RAS space label, e.g. \( K^{11,\mu\delta}_\xi \) and \( K^{11,\nu\epsilon}_\xi \). However, as indicated in Section 3.3.1, when \( i_h = i_e = 0 \), the model string already represents the global string. Thus for this case, the specific string will still be represented as \( K^{0,0}_\xi \). The reason for developing this specific notation is to avoid confusions when expressing a set of global strings, e.g. \( \left\{ K^{11,\mu\delta}_\xi, \cdots, K^{11,\nu\epsilon}_\xi \right\} \), that is expanded from one model string, e.g. \( K^{11,\mu\delta}_\xi \). In the
following both the general notation (to describe algorithms and theory) and the specific notation (to describe examples) will be used.

The orbital integral types (in total 22 cases) of the model space and their corresponding expansion are given in Table 3.1. From this, we can see that only when all the four model orbital indices are in the RAS2 subspace, the expansion of the model orbital integral results in only one global orbital integral, which is the model integral itself. This again proves to us that the CASSCF problem can be treated as a special case of the RASSCF problem. Once any one of the model orbital indexes is dropped in the model RAS1 or model RAS3 subspaces, the expansion of the model orbital integral will lead to a set of full RAS space orbital integrals.

From Table 3.1 we can also see that the first 7 integral cases would be our target cases for implementing the factorized matrix multiplication method. This is because in these 7 cases, either the \(wx\) model index pair or the \(yz\) model index pair are both in the RAS2 subspace. Based on the above description, the modification of these cases should provide the best performance. However, as there are in total 22 integral cases in a RAS problem, we would like to find out the weight of the contribution to the CI vector updating of each of these 22 cases in order to see whether these 7 cases are the most time consuming cases.

### 3.3.5 Contribution Weight of These Model Integral Types

As indicated in the last Section, whether the 7 integral cases are the most time consuming cases among the 22 integral types should be identified. In the CASSCF case, the symmetry properties of the two-electron integrals \((ij|kl)\) mean that it is possible to reduce the loops over \(i,j,k,l\) to unique integrals with summation over \(i \geq j, k \geq l\) and \((ij) \geq (kl)\). Since there is no occupation restriction in the RAS model space, by introducing the model space we can replace summation over orbital indices \(i,j,k,l\) with the model space orbital indices \(w,x,y,z\). A single set of model space indices potentially
### 3. Algorithm for the Direct RASSCF Method (Part I)

Table 3.1 The model integral types and their expanding to the full RAS space orbital integrals based on the number of model space indices dropped in the model subspaces.

| Integral cases | The possible values for the full model RAS space orbital indices | Expanding to full space integral: $(wx|yz) \xrightarrow{\text{expanding}} (ij|kl)$ |
|----------------|---------------------------------------------------------------|------------------------------------------------------------------|
| 1              | $i$ $j$ $k$ $l$                                              | $(ij|kl) \xrightarrow{\text{expanding}} (ij|kl)$                  |
| 2              | $i$ $j$ $k$ $\mu$                                           | $(ij|k\mu) \xrightarrow{\text{expanding}} \{ (ij|kp), \ldots, (ij|kp'') \}$ |
| 3              | $i$ $j$ $\mu$ $\nu$                                         | $(ij|\mu\nu) \xrightarrow{\text{expanding}} \{ (ij|pq), \ldots, (ij|p''q'') \}$ |
| 4              | $i$ $\mu$ $j$ $k$                                           | $(i\mu|jk) \xrightarrow{\text{expanding}} \{ (ip|jk), \ldots, (ip''|jk) \}$ |
| 5              | $\delta$ $\mu$ $i$ $j$                                       | $(\delta\mu|ij) \xrightarrow{\text{expanding}} \{ (ap|ij), \ldots, (a''p''|ij) \}$ |
| 6              | $\delta$ $i$ $j$ $k$                                         | $(\delta i|jk) \xrightarrow{\text{expanding}} \{ (ai|jk), \ldots, (a''i|jk) \}$ |
| 7              | $\delta$ $\epsilon$ $i$ $j$                                 | $(\delta\epsilon|ij) \xrightarrow{\text{expanding}} \{ (ab|ij), \ldots, (a''b''|ij) \}$ |
| 8              | $\mu$ $\nu$ $\mu$ $\nu$                                    | $(\mu\nu|\mu\nu) \xrightarrow{\text{expanding}} \{ (pq|rs), \ldots, (p''q''r''s'') \}$ |
| 9              | $i$ $\mu$ $\mu$ $\nu$                                       | $(i\mu|\mu\nu) \xrightarrow{\text{expanding}} \{ (ir|pq), \ldots, (ir''p''q'') \}$ |
| 10             | $i$ $\mu$ $j$ $\nu$                                         | $(i\mu|j\nu) \xrightarrow{\text{expanding}} \{ (ip|jq), \ldots, (ip''|jq'') \}$ |
| 11             | $\delta$ $\mu$ $\mu$ $\nu$                                 | $(\delta\mu|\mu\nu) \xrightarrow{\text{expanding}} \{ (ar|pq), \ldots, (a''r''|p''q'') \}$ |
| 12             | $\delta$ $\mu$ $i$ $\nu$                                    | $(\delta\mu|i\nu) \xrightarrow{\text{expanding}} \{ (ap|iq), \ldots, (a''p''|iq'') \}$ |
| 13             | $\delta$ $\mu$ $\epsilon$ $\nu$                            | $(\delta\mu|\epsilon\nu) \xrightarrow{\text{expanding}} \{ (ap|bq), \ldots, (a''p''|b''q'') \}$ |
| 14             | $\delta$ $i$ $\mu$ $\nu$                                    | $(\delta i|\mu\nu) \xrightarrow{\text{expanding}} \{ (ai|pq), \ldots, (a''i|p''q'') \}$ |
| 15             | $\delta$ $i$ $j$ $\mu$                                       | $(\delta i|j\mu) \xrightarrow{\text{expanding}} \{ (ai|jp), \ldots, (a''i|jp'') \}$ |
| 16             | $\delta$ $i$ $\epsilon$ $\mu$                               | $(\delta i|\epsilon\mu) \xrightarrow{\text{expanding}} \{ (ai|bp), \ldots, (a''i|b''p'') \}$ |
| 17             | $\delta$ $i$ $\epsilon$ $j$                                 | $(\delta i|\epsilon j) \xrightarrow{\text{expanding}} \{ (ai|bj), \ldots, (a''i|b''j'') \}$ |
| 18             | $\delta$ $\epsilon$ $\mu$ $\nu$                            | $(\delta\epsilon|\mu\nu) \xrightarrow{\text{expanding}} \{ (ab|pq), \ldots, (a''b''|p''q'') \}$ |
| 19             | $\delta$ $\epsilon$ $i$ $\mu$                               | $(\delta\epsilon|i\mu) \xrightarrow{\text{expanding}} \{ (ab|ip), \ldots, (a''b''|ip'') \}$ |
| 20             | $\delta$ $\epsilon$ $\delta$ $\mu$                         | $(\delta\epsilon|\delta\mu) \xrightarrow{\text{expanding}} \{ (ab|cp), \ldots, (a''b''|c''p'') \}$ |
| 21             | $\delta$ $\epsilon$ $\delta$ $i$                            | $(\delta\epsilon|\delta i) \xrightarrow{\text{expanding}} \{ (ab|ci), \ldots, (a''b''|c''i) \}$ |
| 22             | $\delta$ $\epsilon$ $\delta$ $\epsilon$                   | $(\delta\epsilon|\delta\epsilon) \xrightarrow{\text{expanding}} \{ (ab|cd), \ldots, (a''b''|c''d'') \}$ |
represents a set of global orbital indices. Nevertheless, the symmetry condition of the CAS orbital indices is also suitable for the RAS model space:

\[
\begin{align*}
  w & \geq x, \quad y \geq z, \quad (wx) \geq (yz) \\
  \text{with} \quad (wx) & = \frac{w(w-1)}{2} + x, \quad (yz) = \frac{y(y-1)}{2} + z.
\end{align*}
\]  (3.3.24)

Because of these restrictions we will only find a restricted number of excitation combinations between RAS subspaces, which is reflected in Table 3.1. However, the integral types listed in Table 3.1 indicate the interactions between the orbitals in the model RAS subspaces. If we replace the interaction between the orbitals in a certain model RAS subspace with the interaction between the model RAS subspaces that the orbital index drops in, we can define the following terms to represent the excitations reflected in the integrals of Table 3.1:

- **1→1**: an excitation within the RAS1 subspace \((a_{\mu\xi}^\dagger a_{\mu\xi})\) or \((a_{\mu\xi}^\dagger a_{\nu\xi})\)
- **1→2**: an excitation from RAS1 subspace to RAS2 subspace \((a_{\mu\xi}^\dagger a_{\mu\xi})\)
- **2→2**: an excitation within the RAS2 subspace \((a_{\mu\xi}^\dagger a_{\mu\xi})\) or \((a_{\mu\xi}^\dagger a_{\nu\xi})\)
- **1→3**: an excitation from RAS1 subspace to RAS3 subspace \((a_{\mu\xi}^\dagger a_{\mu\xi})\)
- **2→3**: an excitation from RAS2 subspace to RAS3 subspace \((a_{\mu\xi}^\dagger a_{\mu\xi})\)
- **3→3**: an excitation within the RAS3 subspace \((a_{\mu\xi}^\dagger a_{\mu\xi})\) or \((a_{\mu\xi}^\dagger a_{\nu\xi})\)

The possible combinations of these excitations are shown in Table 3.2. Assuming \(MxHole \geq 2\) and \(MxElec \geq 2\), how many excitations \(a_{\mu\nu}^\dagger a_{\nu\gamma}\) (\(\gamma \in \{\alpha, \beta\}\)) are possible? There are only two possibilities for excitations of type \(1\rightarrow 1\), namely \(a_{\mu\nu}^\dagger a_{\mu\nu}\) and \(a_{\nu\gamma}^\dagger a_{\nu\gamma}\). Likewise there are only two realizations for excitations of type \(3\rightarrow 3\). Excitations from model RAS1 to model RAS3 are all represented by only one model space excitation, \(a_{\mu\nu}^\dagger a_{\nu\gamma}\). We have \(M(RAS2)\) possible model space excitations of type \(1\rightarrow 2\) (one for each RAS2 orbital), and also \(M(RAS2)\) excitations of type \(2\rightarrow 3\) (also one for each RAS2 orbital). Finally, there are \(\left(\frac{M(RAS2)}{2}\right) + M(RAS2)\) possibilities for the excitation type \(2\rightarrow 2\).

Table 3.3 shows these numbers analytically and numerically for \(M(RAS2) = 4, 6\) and 10.
3. Algorithm for the Direct RASSCF Method (Part I)

Table 3.2 Possible combinations of excitations between RAS subspaces. \((x \rightarrow w)\) indicates excitation from orbital \(x\) to orbital \(w\)

<table>
<thead>
<tr>
<th>(x \rightarrow w)</th>
<th>1 (\rightarrow) 1</th>
<th>1 (\rightarrow) 2</th>
<th>2 (\rightarrow) 2</th>
<th>1 (\rightarrow) 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z \rightarrow y)</td>
<td>1 (\rightarrow) 1</td>
<td>1 (\rightarrow) 1</td>
<td>1 (\rightarrow) 2</td>
<td>1 (\rightarrow) 1</td>
</tr>
<tr>
<td></td>
<td>1 (\rightarrow) 2</td>
<td>2 (\rightarrow) 2</td>
<td>2 (\rightarrow) 2</td>
<td>1 (\rightarrow) 1</td>
</tr>
</tbody>
</table>

Table 3.3 Number of model space excitations \(a^\gamma_w a^\gamma_x\) between RAS model spaces, provided \(\text{MxHole} \geq 2\) and \(\text{MxElec} \geq 2\); analytically and numerically for different RAS2 orbital subspace sizes. \((x \rightarrow w)\) indicates excitation from orbital \(x\) to orbital \(w\)

<table>
<thead>
<tr>
<th>(x \rightarrow w)</th>
<th>(2 \rightarrow 3)</th>
<th>(3 \rightarrow 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z \rightarrow y)</td>
<td>1 (\rightarrow) 1</td>
<td>1 (\rightarrow) 1</td>
</tr>
<tr>
<td></td>
<td>1 (\rightarrow) 2</td>
<td>2 (\rightarrow) 2</td>
</tr>
<tr>
<td></td>
<td>1 (\rightarrow) 1</td>
<td>1 (\rightarrow) 3</td>
</tr>
<tr>
<td></td>
<td>2 (\rightarrow) 3</td>
<td>1 (\rightarrow) 1</td>
</tr>
<tr>
<td></td>
<td>2 (\rightarrow) 2</td>
<td>2 (\rightarrow) 2</td>
</tr>
<tr>
<td></td>
<td>1 (\rightarrow) 3</td>
<td>2 (\rightarrow) 3</td>
</tr>
<tr>
<td></td>
<td>3 (\rightarrow) 3</td>
<td>3 (\rightarrow) 3</td>
</tr>
</tbody>
</table>

Table 3.4 summarises the number of unique index quadruples corresponding to the integrals in Table 3.1, due to the restriction on possible combinations of \((wx)\) with \((yz)\) (Eq. 3.3.24). These results show that

- The contribution of \(\left(a^\gamma_w a^\gamma_x, a^\gamma_y a^\gamma_z\right) = (2 \rightarrow 2, 2 \rightarrow 2)\) is the largest one and its relative importance grows with increasing size of the RAS2 subspace \(M(\text{RAS2})\).
  This means that in most cases the extension of RAS1 and RAS3 subspaces will simply result a list of consecutive subspace addresses, and that both the symbolic matrix element, \(B_{wx}^{\kappa m} L^m\), and orbital integral, \((ij|kl)\), will remain constant for all contributions derived from a model space string pair.

- All other relatively large contributions stem from excitations with at least one of the two excitations being a \(2 \rightarrow 2\).

In summary, the most time consuming part will be in the case where all four model orbital indices are in the RAS2 subspace (the first case in Table 3.1), and at least one of
the two excitations, $a_w^i a_x^j$ or $a_y^i a_z^j$, is an excitation from RAS2 to RAS2, $a_t^i a_j^j$ (the second to the seventh cases in Table 3.1). These cases become more dominant in terms of computation time when the RAS2 subspace becomes larger (Table 3.6 and Figure 3.3). Thus we can confirm that the performance can be improved to a significant degree by modifying these target integral cases because they are the most time consuming cases.

Table 3.4 Number of double excitations $a_w^i a_x^j (x \rightarrow w)$, $a_y^i a_z^j (z \rightarrow y)$ between RAS model subspaces; analytically and numerically for different RAS2 orbital subspaces (Constructed using Tables 3.2 and 3.3 with Eq. 3.3.24; $M$ represents $M$(RAS2), $\gamma, \sigma \in \{\alpha, \beta\}$). This is equivalent to the integral cases listed in Table 3.1.

<table>
<thead>
<tr>
<th>$x \rightarrow w$</th>
<th>$z \rightarrow y$</th>
<th>Number of combinations</th>
<th>$M$(RAS2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \rightarrow 1$</td>
<td>$1 \rightarrow 1$</td>
<td>Analytically</td>
<td>$4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$6$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$10$</td>
</tr>
<tr>
<td>$1 \rightarrow 2$</td>
<td>$1 \rightarrow 1$</td>
<td>$2 \times M$</td>
<td>$4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$4$</td>
</tr>
<tr>
<td>$1 \rightarrow 2$</td>
<td>$1 \rightarrow 2$</td>
<td>$(M^2 + M)/2$</td>
<td>$20$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$42$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$110$</td>
</tr>
<tr>
<td>$2 \rightarrow 2$</td>
<td>$1 \rightarrow 1$</td>
<td>$M^3 + M$</td>
<td>$10$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$30$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$91$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$385$</td>
</tr>
<tr>
<td>$2 \rightarrow 2$</td>
<td>$1 \rightarrow 2$</td>
<td>$(2M^3 + 3M^2 + M)/6$</td>
<td>$55$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$231$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$1540$</td>
</tr>
<tr>
<td>$1 \rightarrow 3$</td>
<td>$1 \rightarrow 1$</td>
<td>$2$</td>
<td>$2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$2$</td>
</tr>
<tr>
<td>$1 \rightarrow 2$</td>
<td>$1 \rightarrow 2$</td>
<td>$M$</td>
<td>$4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$6$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$10$</td>
</tr>
<tr>
<td>$2 \rightarrow 2$</td>
<td>$1 \rightarrow 2$</td>
<td>$(M^2 + M)/2$</td>
<td>$10$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$21$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$55$</td>
</tr>
<tr>
<td>$1 \rightarrow 3$</td>
<td>$1 \rightarrow 2$</td>
<td>$M^3$</td>
<td>$16$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$36$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$100$</td>
</tr>
<tr>
<td>$2 \rightarrow 3$</td>
<td>$1 \rightarrow 1$</td>
<td>$2 \times M$</td>
<td>$8$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$12$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$20$</td>
</tr>
<tr>
<td>$1 \rightarrow 2$</td>
<td>$1 \rightarrow 2$</td>
<td>$M^2$</td>
<td>$16$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$36$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$100$</td>
</tr>
<tr>
<td>$2 \rightarrow 2$</td>
<td>$1 \rightarrow 2$</td>
<td>$(M^3 + M^2)/2$</td>
<td>$40$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$126$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$550$</td>
</tr>
<tr>
<td>$1 \rightarrow 3$</td>
<td>$1 \rightarrow 3$</td>
<td>$M$</td>
<td>$4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$6$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$10$</td>
</tr>
<tr>
<td>$2 \rightarrow 3$</td>
<td>$1 \rightarrow 3$</td>
<td>$(M^2 + M)/2$</td>
<td>$10$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$21$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$55$</td>
</tr>
<tr>
<td>$3 \rightarrow 3$</td>
<td>$1 \rightarrow 1$</td>
<td>$2 \times M$</td>
<td>$8$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$12$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$20$</td>
</tr>
<tr>
<td>$1 \rightarrow 3$</td>
<td>$1 \rightarrow 2$</td>
<td>$2 \times M$</td>
<td>$20$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$42$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$110$</td>
</tr>
<tr>
<td>$1 \rightarrow 3$</td>
<td>$2 \rightarrow 2$</td>
<td>$2 \times M$</td>
<td>$8$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$12$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$20$</td>
</tr>
<tr>
<td>$3 \rightarrow 3$</td>
<td>$3 \rightarrow 3$</td>
<td>$4$</td>
<td>$4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$4$</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td></td>
<td></td>
<td>$278$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$743$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$3242$</td>
</tr>
</tbody>
</table>
3. Algorithm for the Direct RASSCF Method (Part I)

Table 3.5 General overview of the total numbers of distinctive \(w, x, y, z\) quadruples for \(Mx\text{Hole}=2, MxElec=2\), analytically and numerically for a few RAS2 subspace sizes \((M \text{ represents } M(\text{RAS2}))\).  

<table>
<thead>
<tr>
<th>Number of model indices (w, x, y, z) in RAS2</th>
<th>Total dimension (M(\text{RAS2}))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Analytically</td>
</tr>
<tr>
<td>4 ((M^4 + 2M^3 + 3M^2 + 2M) / 8)</td>
<td>55</td>
</tr>
<tr>
<td>3 (M^4 + M^2)</td>
<td>80</td>
</tr>
<tr>
<td>(2 \text{ (without } 2 \rightarrow 3, 1 \rightarrow 2))</td>
<td>70</td>
</tr>
<tr>
<td>(2 \text{ (} 2 \rightarrow 3, 1 \rightarrow 2 \text{ only)})</td>
<td>16</td>
</tr>
<tr>
<td>1 (10 M)</td>
<td>40</td>
</tr>
<tr>
<td>0 (17)</td>
<td>17</td>
</tr>
<tr>
<td>(\sum)</td>
<td>278</td>
</tr>
</tbody>
</table>

Table 3.6 The percentage of the dimension for the dominating cases over the entire dimension for \(Mx\text{Hole} = 2, MxElec=2\), analytically and numerically for several different RAS2 subspace sizes \((M \text{ represents } M(\text{RAS2}))\).

<table>
<thead>
<tr>
<th>Dimension</th>
<th>(M(\text{RAS2}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytically</td>
<td>4</td>
</tr>
<tr>
<td>Total cases</td>
<td>((M^4 + 2M^3 + 3M^2 + 2M) / 8) + ((M^3 + M^2) + 7(M^2 + M) / 2) + (M^2 + 10 M + 17)</td>
</tr>
<tr>
<td>First 7 cases in Table 3.1</td>
<td>((M^4 + 2M^3 + 3M^2 + 2M) / 8) + ((M^3 + M^2) + 3(M^2 + M))</td>
</tr>
<tr>
<td>%</td>
<td>62.95%</td>
</tr>
</tbody>
</table>
3.4 Algorithm for the Selected 7 Cases

As stated in Section 3.3.4, we would like to apply the factorized matrix multiplication method to the first 7 integral cases of Table 3.1. In Section 3.3.5, we have proved that these 7 integral cases are the most time consuming cases among the entire 22 cases. Therefore in this section we will mainly focus the factorized matrix multiplication algorithms for these 7 integral cases. For the other cases, according to Section 3.3.3, we know it would not be suitable to implement the matrix multiplication scheme. Instead, for these remaining cases the unfactorized method is better suited. In the next chapter, we will discuss the algorithm for these remaining cases in more details. In the following, the algorithms of the matrix multiplication scheme for the 7 cases are described, starting with a review of how the current method works for these cases.
3. Algorithm for the Direct RASSCF Method (Part I)

3.4.1 Current Unfactorized Direct Method

For the purpose of simplicity, we will use the model linear $\alpha \alpha$ -contribution to the CI vector updating of the current method to demonstrate how the current method works on the first 7 cases:

$$\alpha\alpha B^{K \beta}_{wxyz} = \left\langle K_{\alpha}^{m} \right| a_{w\alpha} a_{y\alpha} a_{z\alpha} \left| L_{\alpha}^{m} \right\rangle \left\langle K_{\beta}^{m} \right| L_{\beta}^{m} \right\rangle.$$  \hspace{1cm} (3.4.1)

In the current method, although the model orbital integral type is not crucial to the current method because it uses the unfactorized direct way to update the CI vector, the treatment of the first case and the other six cases in Table 3.1 are slightly different. This is because for the first case (the $2 \rightarrow 2, 2 \rightarrow 2$ excitation) both the symbolic matrix element, $B^{K \beta}_{wxyz}$, and the orbital integral, $\langle ij|kl \rangle$, will remain constant for all contributions derived from a model space string pair. Here we will use one example to illustrate how the current method updates the CI vector for the first case.

For a RAS calculation with MxHole = MxElec = 2, assuming the model excitation we have by hand is $B^{K^{11,\mu\delta}L^{11,\mu\delta}}_{ijkl}$, we have

$$B^{K^{11,\mu\delta}L^{11,\mu\delta}}_{ijkl} = \left\langle K_{\alpha}^{11,\mu\delta} \right| a_{i\alpha} a_{k\alpha} a_{l\alpha} \left| L_{\alpha}^{11,\mu\delta} \right\rangle \left\langle K_{\beta}^{m} \right| L_{\beta}^{m} \right\rangle,$$

where the superscript of the RAS model strings indicates there is one hole at orbital $\mu$ in RAS1 model space and one electron at orbital $\delta$ in model RAS3 space for both of the initial and final state, $\left| L_{\alpha}^{11,\mu\delta} \right\rangle$ and $\left| K_{\alpha}^{11,\mu\delta} \right\rangle$, respectively. The expanding process for this case, as described in Section 3.3.5, will simply relate to the expansion of the model RAS1 and model RAS3 orbital indices to full RAS1 and RAS3 orbital indices,

$$\mu \Rightarrow \left( \ p \ p' \ \cdots \ p''' \right), \text{ and}$$

$$\delta \Rightarrow \left( \ a \ a' \ \cdots \ a''' \right).$$

Based on this expansion, the first full RAS space string pair would be $\left\{ K_{\alpha}^{11,\mu\delta}, L_{\alpha}^{11,\mu\delta} \right\}$. According to Eq. 3.3.3, we know the $\beta$ -strings that are allowed to combine with this $\alpha$ -string pair are in the categories of $\text{Cat}_{\beta}(0,0)$, $\text{Cat}_{\beta}(0,1)$, $\text{Cat}_{\beta}(1,0)$, and $\text{Cat}_{\beta}(1,1)$. 

165
Since the orbital integral, \((ij|kl)\), and the model symbolic matrix element, \(B_{ijkl}^{K^1_{\alpha},L^1_{\alpha}}\), will remain constant, the CI vector updating process for this \(\alpha\)-string pair will be:

\[
\sigma(K_{\alpha}^{11,pa},K_{n,\beta}^{0,0}) \leftarrow (ij|kl)B_{ijkl}^{K_{\alpha}^{11,pa},L_{\alpha}^{0,0}}C(L_{\alpha}^{11,pa},K_{n,\beta}^{0,0})
\]
\[
\sigma(K_{\alpha}^{11,pa},K_{n,\beta}^{01,b}) \leftarrow (ij|kl)B_{ijkl}^{K_{\alpha}^{11,pa},L_{\alpha}^{01,b}}C(L_{\alpha}^{11,pa},K_{n,\beta}^{01,b}) \quad (3.4.2)
\]
\[
\sigma(K_{\alpha}^{11,pa},K_{n,\beta}^{10,q}) \leftarrow (ij|kl)B_{ijkl}^{K_{\alpha}^{11,pa},L_{\alpha}^{10,q}}C(L_{\alpha}^{11,pa},K_{n,\beta}^{10,q})
\]
\[
\sigma(K_{\alpha}^{11,pa},K_{n,\beta}^{11,gb}) \leftarrow (ij|kl)B_{ijkl}^{K_{\alpha}^{11,pa},L_{\alpha}^{11,gb}}C(L_{\alpha}^{11,pa},K_{n,\beta}^{11,gb})
\]

where \(n\) varies from 1 to the length of the category that the \(\beta\)-string drops in. For another string pair, e.g. \(K_{\alpha}^{11,pa'},L_{\alpha}^{11,pa'}\), the CI vector updating process becomes:

\[
\sigma(K_{\alpha}^{11,pa'},K_{n,\beta}^{0,0}) \leftarrow (ij|kl)B_{ijkl}^{K_{\alpha}^{11,pa'},L_{\alpha}^{0,0}}C(L_{\alpha}^{11,pa'},K_{n,\beta}^{0,0})
\]
\[
\sigma(K_{\alpha}^{11,pa'},K_{n,\beta}^{01,b}) \leftarrow (ij|kl)B_{ijkl}^{K_{\alpha}^{11,pa'},L_{\alpha}^{01,b}}C(L_{\alpha}^{11,pa'},K_{n,\beta}^{01,b}) \quad (3.4.3)
\]
\[
\sigma(K_{\alpha}^{11,pa'},K_{n,\beta}^{10,q}) \leftarrow (ij|kl)B_{ijkl}^{K_{\alpha}^{11,pa'},L_{\alpha}^{10,q}}C(L_{\alpha}^{11,pa'},K_{n,\beta}^{10,q})
\]
\[
\sigma(K_{\alpha}^{11,pa'},K_{n,\beta}^{11,gb}) \leftarrow (ij|kl)B_{ijkl}^{K_{\alpha}^{11,pa'},L_{\alpha}^{11,gb}}C(L_{\alpha}^{11,pa'},K_{n,\beta}^{11,gb})
\]

From Eqs. 3.4.2 and 3.4.3 we can see the scalar, \((ij|kl)B_{ijkl}^{K_{\alpha}^{11,pa},L_{\alpha}^{11,gb}}\), remains the same but different rows in the same block of the CI vector (e.g. the same element in Eq. 3.3.4b) are accessed for one set of RAS2 orbital indices \(\{i,j,k,l\}\). Thus these blocks of the CI vector are accessed repeatedly, which is inefficient. Moreover, for another set of \(\{i,j,k,l\}\), e.g \(\{i,j,k',l''\}\), the same rows of these blocks of the CI vector will be accessed again.

For the 2\(^{nd}\) to the 7\(^{th}\) cases in Table 3.1, most of the CI vector updating process will be similar to the process of the first case. The only difference will be the orbital integrals will change accordingly when expanding the model string pairs to full RAS space.

Taking the second case of Table 3.1 as an example, the integral class is \((ij|k\mu)\), which indicates that there will be one electron excited out from the model RAS1 subspace into the RAS2 subspace. Assuming now we have a set of model orbital integral, a model string pair and its associated model symbolic matrix element as:

\(\{(ij|k\mu),\{K_{\alpha}^{11,gb},L_{\alpha}^{01,b}\},B_{ijkl}^{K_{\alpha}^{11,gb},L_{\alpha}^{01,b}}\}\), we notice that the values of \(i_h\) for the initial and final
3. Algorithm for the Direct RASSCF Method (Part I)

states, \( |L_{\xi}^{01,\delta}\rangle \) and \( |K_{\xi}^{11,\mu\delta}\rangle \) respectively, are different now. The model RAS1 subspace has been involved in the orbital integral. Thus the expansion process will not only be the orbital index expansion, but also be the integral expansion and the RAS1 subspace sign factor expansion:

\[
\mu \Rightarrow \left( p \ p' \ \cdots \ p^{'''} \right), \quad \text{sgn}_{\xi}^{\text{RAS1}} \Rightarrow \left( \text{sgn}_{\xi}^{\text{p,RAS1}} \ \text{sgn}_{\xi}^{\text{p',RAS1}} \ \cdots \ \text{sgn}_{\xi}^{\text{p''',RAS1}} \right),
\]

\[
(ij|k\mu) \Rightarrow \{(ij|kp) \ (ij|kp') \ \cdots \ (ij|kp^{'''})\},
\]

\[
\delta \Rightarrow \left( a \ a' \ \cdots \ a^{'''}\right).
\]

For orbital \( p \) of the RAS1 subspace, the orbital integral will be \( (ij|kp) \), the full RAS space symbolic matrix element is: \( B_{ijkp}^{\alpha \beta \mu \gamma} = B_{ijkp}^{\alpha \beta} \times \text{sgn}_{\xi}^{\alpha \beta} \), and the corresponding full RAS space string pairs will be: \( \{K_{\xi}^{11,\mu\delta}, L_{\xi}^{01,\alpha}\}, \{K_{\xi}^{11,\mu\delta}, L_{\xi}^{01,\alpha'}\}, \cdots, \{K_{\xi}^{11,\mu\delta}, L_{\xi}^{01,\alpha^{''''}}\} \), from which we can see a set of full RAS string pairs share the same orbital integral and the symbolic matrix element. The allowed \( \gamma \) -string categories to combine with these \( \xi \) -strings would also be: \( \text{Cat}_{\gamma}(0,0), \text{Cat}_{\gamma}(0,1), \text{Cat}_{\gamma}(1,0) \), and \( \text{Cat}_{\gamma}(1,1) \). The CI vector for this orbital \( p \) is thus updated as (assuming \( \xi = \alpha, \gamma = \beta \)):

\[
\sigma(K_{\alpha}^{11,\mu\delta}, K_{\beta}^{01,0}) \leftarrow (ij|kp)B_{ijkp}^{\alpha \beta \mu \gamma} C(L_{\alpha}^{01,\alpha}, K_{\beta}^{01,0})
\]

\[
\vdots
\]

\[
\sigma(K_{\alpha}^{11,\mu\delta}, K_{\beta}^{01,1}) \leftarrow (ij|kp)B_{ijkp}^{\alpha \beta \mu \gamma} C(L_{\alpha}^{01,\alpha'}, K_{\beta}^{01,1})
\]

\[
\vdots
\]

\[
(3.4.4)
\]

\[
\sigma(K_{\alpha}^{11,\mu\delta}, K_{\beta}^{01,0}) \leftarrow (ij|kp)B_{ijkp}^{\alpha \beta \mu \gamma} C(L_{\alpha}^{01,\alpha''}, K_{\beta}^{01,0})
\]

\[
\vdots
\]

\[
\sigma(K_{\alpha}^{11,\mu\delta}, K_{\beta}^{01,1}) \leftarrow (ij|kp)B_{ijkp}^{\alpha \beta \mu \gamma} C(L_{\alpha}^{01,\alpha'''}, K_{\beta}^{01,1})
\]

For another extended orbital \( p' \) in RAS1, the orbital integral and symbolic matrix elements in Eq. 3.4.4 become \( (ij|kp') \) and \( B_{ijkp'}^{\alpha \beta \mu \gamma} = B_{ijkp}^{\alpha \beta \mu \gamma} \times \text{sgn}_{\alpha}^{\gamma \beta \text{RAS1}} \). There will also

167
be a set of full RAS space string pairs corresponding to the integral and symbolic matrix element. Eq. 3.4.4 will be carried out again for this new extended orbital $p'$ with the new generated orbital integral and the symbolic matrix elements. Thus we can see, the same rows of the same blocks of the CI vector will be accessed again and again for different extended orbitals $p$s in RAS1. This will be inefficient and we need to find a way to solve this.

### 3.4.2 Factorized Matrix Multiplication Approach

In Section 3.3.3 we have briefly discussed that by factorizing the model bilinear contribution we can convert the factorized matrix multiplication method used for the CASSCF wavefunction to the RASSCF wavefunction (Eq. 3.3.18 and Eqs. 3.3.19):

\[
\sigma^{\text{Bilinear}}_{K^m \xi^m, K^m \gamma^m} = \frac{1}{2} \sum_{wx} \sum_{L^m_{\xi \xi}} \langle K^m_{\xi^m} | E_{wx}^\alpha | L^m_{\xi \xi} \rangle \sum_{yx} \sum_{L^m_{\xi \xi}} \langle K^m_{\gamma^m} | E_{yx}^{\beta} | L^m_{\xi \xi} \rangle C(L^m_{\xi \xi}, L^m_{\xi \xi}),
\]

and

\[
X^m_{wx}(K^m_{\xi^m}, L^m_{\xi^m}) = \sum_{yx} \langle wx | yz \rangle \langle K^m_{\xi^m} | E_{yz}^\xi | L^m_{\xi^m} \rangle,
\]

\[
M^m_{wx}(K^m_{\xi^m}, L^m_{\xi^m}) = \sum_{L^m_{\xi^m}} X^m_{wx}(K^m_{\xi^m}, L^m_{\xi^m}) C(L^m_{\xi^m}, L^m_{\xi^m}),
\]

\[
\sigma^{\text{Bilinear}}(K^m_{\xi^m}, K^m_{\gamma^m}) = \sum_{wx} \sum_{L^m_{\xi^m \xi^m}} \langle K^m_{\xi^m} | E_{wx}^\alpha | L^m_{\xi^m \xi^m} \rangle M^m_{wx}(K^m_{\xi^m}, L^m_{\xi^m}) (\xi, \gamma \in \{\alpha, \beta\}).
\]

As stated in Section 3.3.3, the model orbital index pair $wx$ must be expanded to the full RAS space to construct the real contribution to the CI vector updating. From Eq. 3.3.19c we can see, for a certain $wx$ model index pair, we would like to keep the number of full RAS space orbital index pairs expanded from this model index pair as low as possible to achieve the best matrix multiplication performance. Therefore the first 7 model orbital integral cases in Table 3.1 would be our target cases to apply the factorized matrix multiplication method, because these cases have the least number (only 1) of full RAS space orbital index pairs expanded from a certain $wx$ model index pair. In this subsection,
we will discuss the algorithm given in Eqs. 3.3.19 for the 7 model integral cases in more details.

For the first 7 cases in Table 3.1, there are two types of orbital integrals. One type is that the left hand side of the model orbital integral is a $2 \rightarrow 2$ excitation ($w$ and $x$ are both in RAS2 subspace) while the right hand side varies (the $2 \rightarrow 2, 2 \rightarrow 2$ integral is included in this type): $(ij|yz)$; the other type is the other way round, the right hand side of the model orbital integral is a $2 \rightarrow 2$ excitation ($y$ and $z$ are both in RAS2 subspace) while the left hand side varies (excluding the $2 \rightarrow 2, 2 \rightarrow 2$ integral): $(wx|ij)$. We will discuss the factorized matrix multiplication modification for these two types separately in the following based on Eq. 3.3.18.

### 3.4.2.1 $(ij|yz)$ Model Orbital Integral Cases

This integral type corresponds to the first 3 cases in Table 3.1, where both the $w$ and $x$ model orbital indices are in RAS2: $(ij|kl)$, $(ij|\mu \nu)$, and $(ij|\kappa \mu)$. Thus the expansion of $wx$ index pair to the full RAS space will result only one index pair, which is the $wx$ pair itself. The global $kl$ orbital index pairs are changed according to the expansion of the $yz$ model orbital index pairs. Thus by exchanging the model orbital indices in Eq. 3.3.18 to full RAS space orbital indices, the bilinear contribution to the CI vector updating for these model orbital integral cases becomes,

$$
\sigma_{K_\xi K_\eta L_\gamma}^{\text{Bilinear}} = \frac{1}{2} \sum_{ij} \sum_{\kappa \lambda} B_{ij}^{\{K_\xi \kappa, L_\gamma \lambda\}} \sum_{kl} (ij|kl) B_{kl}^{\{K_\kappa \lambda, K_\eta \mu\}} C_{L_\gamma L_\lambda L_\mu L_\lambda},
$$

(3.4.5)

where $B_{ij}^{\{K_\xi \kappa, L_\gamma \lambda\}} = B_{ij}^{\{\kappa \lambda, \gamma \lambda\}}$ and $B_{kl}^{\{K_\kappa \lambda, K_\eta L_\gamma L_\lambda\}} = \langle K_\kappa^{\gamma \lambda} L_\gamma | a_{k\kappa}^\dagger a_{l\lambda} | L_\lambda \rangle$; the orbital indices $i,j,k,l$ indicate the full RAS space orbital indices with $i,j$ are always in RAS2 subspace. For these model orbital integral cases, $kl$ can both be in the RAS2 subspace ($2 \rightarrow 2, 2 \rightarrow 2$ excitation in Table 3.4), or both in the RAS1 subspace ($2 \rightarrow 2, 1 \rightarrow 1$ excitation), or have $k$ in the RAS2 subspace while $l$ is in the RAS1 subspace ($2 \rightarrow 2, 1 \rightarrow 2$ excitation). Moreover, the orbital symmetry conditions that are applied in the CASSCF method...
wavefunction, $i \geq j, k \geq l$ and $(ij) \geq (kl)$, are also suitable for the expanded full RAS space orbital indices. Based on this, we can break up Eq. 3.4.5, as with Eqs. 3.3.19, for the first three cases of Table 3.1 as,

$$X_{ij}^{\text{bilinear}}(K_y^{i',j'}, L_y^{\alpha',\beta'}) = \sum_{kl \in ij} (ij|kl) B_{ij}^{\{K_y^{i',j'}, L_y^{\alpha',\beta'}\}},$$  \hspace{1cm} (3.4.6a)

where $kl$ represents the full RAS space orbital indices after expanding the $yz$ model orbital indices and fulfilling the orbital symmetry conditions. Then the matrix multiplication step for these three cases is,

$$M_{ij}^{K_y^{i',j'}, L_y^{\alpha',\beta'}} = \sum_{\ell \in \ell'} X_{ij}^{K_y^{i',j'}, \ell^{\alpha',\beta'}} C^T(L_{\ell}^{\alpha',\beta'}, L_{\ell'}^{\alpha',\beta'}),$$  \hspace{1cm} (3.4.6b)

which fulfills the condition of

$$\begin{cases}
i_h + i'_h \\ i_h + i'_h
\end{cases} \leq \text{MxHole,} \quad \begin{cases}
i_e + i'_e \\ i_e + i'_e
\end{cases} \leq \text{MxElec}.$$

The generated matrix $M$ is then used to obtain the final updated bilinear CI vector:

$$\sigma(K_{\ell}^{h,i}, K_{\ell'}^{h,i'}) \leftarrow \sum_{ij} \sum_{\ell \in \ell'} B_{ij}^{\{K_y^{i',j'}, \ell^{\alpha',\beta'}\}} (M_{ij}^{K_y^{i',j'}, \ell^{\alpha',\beta'}})^T.$$

(3.4.6c)

The conditions of $i_h + i'_h \leq \text{MxHole}$ and $i_e + i'_e \leq \text{MxElec}$ should also be abided. From Eq. 3.4.6c we can see for the first 3 cases in Table 3.1, the matrix multiplication is only carried out once per model $wx$ pair, because this pair is in the RAS2 subspace and the expansion of this pair only results in one full RAS space orbital index pair. Thus for these three cases, according to the result of Chapter 2, we know the performance after the modification (Eqs. 3.4.6) would be much better than the current method. As with the CASSCF method, the RASSCF matrix multiplication process will be much simpler in the Hartree-Waller functions basis as only one spin type strings, e.g. $\alpha$-strings, are used to update the CI vector. This is because of the symmetry property of the determinants,

$$\begin{vmatrix}K_y^h,i_y^j & K_y^h,i_y^{j'} \\ K_y^\alpha,i_{\gamma} & K_y^\beta,i_{\gamma}
\end{vmatrix} = \begin{vmatrix}K_y^h,i_y^{j'} & K_y^h,i_y^j \\ K_y^{\alpha},i_{\gamma} & K_y^{\beta},i_{\gamma}
\end{vmatrix} \forall \begin{cases}
i_h + i'_h \leq \text{MxHole} \\ i_e + i'_e \leq \text{MxElec}.
\end{cases}$$
3. Algorithm for the Direct RASSCF Method (Part I)

3.4.2.2 \((\mathbf{wx}|i\mathbf{j})\) Model Orbital Integral Cases

For the fourth to the seventh integral types in Table 3.2, we can see the situation is slightly different. This time the orbital index pair of \(yz\) will remain constant under expansion. And the extension of the \(wx\) pair will give a set of full RAS space orbital index pairs. In order to keep the expansion of the outer most loop of Eq. 3.3.19c as computationally efficient as possible for these integral cases, we need to swap the summation over \(wx\) and \(yz\) extensions in Eq. 3.4.5 to achieve the best performance as,

\[
\sigma_{\text{Bilinear}}^{\mathbf{K}_{x}^{h}\mathbf{K}_{y}^{h},\mathbf{L}_{x}^{h}\mathbf{L}_{y}^{h}} = \frac{1}{2} \sum_{ij} \sum_{kl} B_{ij}^{k_{x}^{h}\mathbf{K}_{y}^{h},l_{x}^{h}\mathbf{L}_{y}^{h}} \left( i j | k l \right) B_{ij}^{k_{y}^{h}\mathbf{K}_{x}^{h},l_{y}^{h}\mathbf{L}_{x}^{h}} \mathbf{C}_{e}\mathbf{L}_{x}^{h}\mathbf{L}_{y}^{h}, (3.4.7)
\]

where the orbital indices are also the full RAS space indices. This time the \(kl\) indices are always in the RAS2 subspace while the \(ij\) indices can both be in the RAS3 subspace (3\(\rightarrow\)3,2\(\rightarrow\)2 excitation in Table 3.4), or have index \(i\) in RAS3 and index \(j\) in the RAS2 or RAS1 subspace (2\(\rightarrow\)3,2\(\rightarrow\)2 or 1\(\rightarrow\)3,2\(\rightarrow\)2 excitation), or index \(i\) in RAS2 while index \(j\) is in RAS1 (1\(\rightarrow\)2,2\(\rightarrow\)2 excitation). The orbital symmetry condition is applied here too. Similar to the \((ij)|yz\) model orbital integral cases, we can also split Eq. 3.4.7 into 3 terms as,

\[
X_{kl}^{\text{bilinear}}\left(\mathbf{K}_{x}^{i_{r},i_{s}},\mathbf{L}_{x}^{i_{r},i_{s}}\right) = \sum_{ij} \left( i j | k l \right) B_{ij}^{k_{x}^{i_{r},i_{s}}\mathbf{K}_{y}^{i_{r},i_{s}},l_{x}^{i_{u},i_{v}}\mathbf{L}_{y}^{i_{u},i_{v}}}, (3.4.8a)
\]

\[
M_{kl}^{k_{x}^{i_{r},i_{s}}\mathbf{K}_{y}^{i_{r},i_{s}},l_{x}^{i_{u},i_{v}}\mathbf{L}_{y}^{i_{u},i_{v}}} = \sum_{ij} X_{kl}^{k_{x}^{i_{r},i_{s}}\mathbf{K}_{y}^{i_{r},i_{s}},l_{x}^{i_{u},i_{v}}\mathbf{L}_{y}^{i_{u},i_{v}}} \mathbf{C}\left(L_{x}^{i_{u},i_{v}}\mathbf{L}_{y}^{i_{u},i_{v}},\right), (3.4.8b)
\]

where the conditions : \(i_{r}+i_{u}\) \(\leq\) MxHole and \(i_{r}+i_{u}\) \(\leq\) MxElec are applied, and the bilinear contribution to the CI vector updating is:

\[
\sigma(\mathbf{K}_{x}^{i_{x},i_{y}},\mathbf{K}_{y}^{i_{r},i_{s}}) \leftarrow \sum_{ij} \sum_{kl} B_{ij}^{k_{x}^{i_{x},i_{y}}\mathbf{K}_{y}^{i_{x},i_{y}},l_{x}^{i_{u},i_{v}}\mathbf{L}_{y}^{i_{u},i_{v}}} \left( M_{kl}^{k_{x}^{i_{x},i_{y}}\mathbf{K}_{y}^{i_{x},i_{y}},l_{x}^{i_{u},i_{v}}\mathbf{L}_{y}^{i_{u},i_{v}}} \right)^{T}. (3.4.8c)
\]

From Eqs. 3.4.8a-3.4.8c we can see for the fourth to seventh cases in Table 3.1, the summation over the \(yz\) model orbital index pairs and the summation over the \(wx\) model index pairs are switched and all the rest will be similar to the cases of \((ij)|yz\). Again, in the Hartree-Waller functions basis, only one spin type of strings is used for the CI vector.
updating because of the configuration symmetry property given above. Now as we have
the general algorithm for these 7 model orbital integral cases in mind we should move to
how to implement these algorithms, starting from all the numerical values that will be
used in the implementation.

3.5 Numerical Values

In the previous sections, we have introduced many definitions, e.g. the RAS global string,
the RAS model string, the RAS string categories, the sign of the symbolic matrix
elements, etc. without giving the corresponding numerical values. In this section, we will
mainly focus on the numerical values of these variables.

3.5.1 String Category, Addresses of Strings and Configurations

Kozlowski and Pulay [87] proposed a two level addressing scheme for the strings used in
the RASSCF wavefunction. The first level is a string category determined by the number
of holes, $i_h$, in RAS1 and the number of electrons, $i_e$, in RAS3, while the second gives
the local string address within a given category. We will largely adopt this scheme for the
string indexing here.

Eq. 3.3.2 only gives the total number of the $\xi$-string category without defining a single
category. According to [87], each category, $\text{Cat}_\xi(i_h,i_e)$, takes a number as its index and
is defined as,

$$
\text{idCat}_\xi(i_h,i_e) = (i_h + 1) + (\text{MxHole} + 1) \times i_e.
$$

(3.5.1)

Obviously, the index of a certain category is a function of the values of $i_h$ and $i_e$. Thus we
know the strings $K^{0,0}_\xi$ will always be in the first category ($\text{idCat}_\xi(0,0) = 1$), and so on.
The length of a category, which is the total number of the global $\xi$-string in this
category, is thus defined as,
3. Algorithm for the Direct RASSCF Method (Part I)

\[ L[i\text{dCat}_\xi(i_h, i_e)] = \begin{pmatrix} M(\text{RAS1}) \\ i_h \end{pmatrix} \begin{pmatrix} M(\text{RAS2}) \\ N_\xi - M(\text{RAS1}) + i_h - i_e \end{pmatrix} \begin{pmatrix} M(\text{RAS3}) \\ i_e \end{pmatrix}. \] (3.5.2)

Obviously, the total number of the global \( \xi \)-strings will be the sum of the lengths of all the \( \xi \)-string categories,

\[ N_{\xi \text{-str}} = \sum_{i_h=0}^{\text{MxHole}} \sum_{i_e=0}^{\text{MxElec}} L[i\text{dCat}_\xi(i_h, i_e)]. \] (3.5.3)

Therefore, according to the occupation restriction (Eq. 3.3.3), we know the total number of the determinants of certain RAS problems (the dimension of the CI vector) can be calculated as (Eq. 1.5.9),

\[ N_{SD}^{\text{RAS}} = \sum_{i_h=0}^{\text{MxHole}} \sum_{i_e=0}^{\text{MxElec}} L[i\text{dCat}_\alpha(i_h, i_e)] \sum_{i'_h=0}^{\text{MxHole}} \sum_{i'_e=0}^{\text{MxElec}} L[i\text{dCat}_\beta(i'_h, i'_e)]. \] (3.5.4)

If we treat each string category as an independent block of the total global strings, then each string in this block will have its own address starting from 1 to \( L[i\text{dCat}_\xi(i_h, i_e)] \).

This address is thus defined as the local address, termed as \( \text{Addr}\{K_\xi\} \) (\( \xi \in \{\alpha, \beta\} \)), of a string within the category,

\[
\text{Addr}\{K_\xi\} = \left( \text{Addr}\{K_\xi^{\text{RAS1}}\} - 1 \right) \begin{pmatrix} M(\text{RAS1}) \\ i_h \end{pmatrix} + \left( \text{Addr}\{K_\xi^{\text{RAS3}}\} - 1 \right) \begin{pmatrix} M(\text{RAS3}) \\ i_e \end{pmatrix} + \text{Addr}\{K_\xi^{\text{RAS3}}\},
\] (3.5.5)

where each term \( K_\xi^{\text{RAS1}}, K_\xi^{\text{RAS2}}, K_\xi^{\text{RAS3}} \) denote the corresponding subspace strings, as indicated in Eq. 3.3.1. The addresses of these subspace strings are defined via the indexing formula given by Knowles and Handy [107,108],

\[
\text{Addr}\{K_\xi^{\text{RASX}}\} = 1 + \sum_{k=1}^{N_\xi} Z(k, l(k)) \] (3.5.6)

with
According to Eq. 3.5.2, we know the length of each category is: 

\[ Z(k,l) = \sum_{m=M^{(RASX)}-k}^{M^{(RASX)}-k+1} \left[ \binom{m}{N^{RASX}_\xi - k} - \binom{m-1}{N^{RASX}_\xi - k-1} \right] \]

\[ = \left( M^{(RASX)} - N^{RASX}_\xi + k \geq l \geq k; k < N^{RASX}_\xi \right) \]

\[ Z(N^{RASX}_\xi , l) = l - N^{RASX}_\xi \left( M^{(RASX)} \geq l \geq N^{RASX}_\xi \right) \]  

(3.5.7)

where \( M^{(RASX)} \) is the number of orbitals in the subspace \( RASX \) \( (X \in \{1,2,3\}) \), and \( N^{RASX}_\alpha \) the number of electrons in \( RASX \) subspace.

However, the string categories are not actually independent of the entire global string space. The local string addresses must be mapped to the global string addresses, 

\[ \text{Addr}\left\{ K^{i_h,i_d}_\xi \right\}, \]

\[ \text{Addr}\left\{ K^{i_h,i_d}_\alpha \right\} = \text{Addr}\left\{ K_\alpha \right\} + \sum_{\text{Cat}_\alpha=(i_h,i_d)=1}^{\text{Cat}_\alpha=(i_h,i_d)-1} L^{[\text{id Cat}_\alpha]}, \]  

(3.5.8)

where the second term indicates the sum of all the lengths of the previous string categories.

One example can make this addressing scheme clear. Still using the same example given in Section 3.3.1, we choose one global string \( K^{11,32}_\alpha \) = 0010 0101 1011 out of the 96 global strings to illustrate the addressing scheme. Assuming for this RAS problem, \( \text{MxHole} = \text{MxElec} = 1 \), thus there will be four string categories: \( \text{Cat}_\xi(0,0) \) \( (i_{\text{Cat}_\xi(0,0)} = 1) \), \( \text{Cat}_\xi(0,1) \) \( (i_{\text{Cat}_\xi(0,1)} = 2) \), \( \text{Cat}_\xi(1,0) \) \( (i_{\text{Cat}_\xi(1,0)} = 3) \), and \( \text{Cat}_\xi(1,1) \) \( (i_{\text{Cat}_\xi(1,1)} = 4) \). According to Eq. 3.5.2, we know the length of each category is: \( L[1] = 6 \), \( L[2] = 16 \), \( L[3] = 16 \), \( L[4] = 96 \). The addresses of the subspace strings for the global string \( K^{11,32}_\alpha \), are calculated via Eq. 3.5.6: \( \text{Addr}\{0010\} = 2 \), \( \text{Addr}\{0101\} = 2 \), and \( \text{Addr}\{1011\} = 2 \). Thus according to Eq. 3.5.5 the local address of this string, \( K^{11,32}_\alpha \), in string category \( \text{Cat}_\alpha(1,1) \) is calculated as,

\[ \text{Addr}\{ K_\alpha \} = (2-1) \left( \begin{array}{c} 4 \\ 1 \end{array} \right) \left( \begin{array}{c} 4 \\ 1 \end{array} \right) + (2-1) \left( \begin{array}{c} 4 \\ 1 \end{array} \right) + 2 = 22 \]
3. Algorithm for the Direct RASSCF Method (Part I)

and the global address of this string \( K^{11,32}_\alpha \) is calculated via Eq. 3.5.8,

\[
\text{Addr} \{ K^{11,32}_\alpha \} = \text{Addr} \{ K_\alpha \} + L[1] + L[2] + L[3]
\]
\[= 22 + 6 + 16 + 16 = 60 \]

Recall that in the CASSCF wavefunction, the addresses of the determinants are generated by combining \( \alpha \)- and \( \beta \)-strings together freely. However, in the RASSCF wavefunction, as pointed out in Section 3.3.1, the \( \alpha \)-strings cannot be freely combined with all \( \beta \)-strings any more because of the occupation restrictions (Eq. 3.3.3). The resulting expression for valid determinants in the RAS expansion can be obtained by two steps: first the local address for a string pair in a given set of string categories, \( \text{Addr} \{ K_\alpha, K_\beta \} \), is given by,

\[
\text{Addr} \{ K_\alpha, K_\beta \} = (\text{Addr} \{ K_\alpha \} - 1) \times L \left[ \text{id Cat}_\beta \left(i_h',i_e'\right) \right] + \text{Addr} \{ K_\beta \}
\]

(3.5.9)

where \( L \left[ \text{id Cat}_\beta \left(i_h',i_e'\right) \right] \) is the length of the category that corresponds to the \( \beta \)-string. The corresponding global address of the determinant \( |K\rangle = |K^{i_h,i_e}_\alpha, K^{i_h',i_e'}_\beta \rangle \) is then calculated as,

\[
\text{Addr} \{ K \} \equiv \text{Addr} \left\{ K^{i_h,i_e}_\alpha, K^{i_h',i_e'}_\beta \right\} = \text{Addr} \{ K_\alpha, K_\beta \} + F_{\text{Cat'}}
\]

(3.5.10)

where \( F_{\text{Cat'}} \) is the offset accounting for all determinants prior to \( K \). Eq. 3.5.10 is an analogue of Eq. 3.5.8. This can be understood in the following way: according to the description given in Section 3.3.1, if we write the CI vector in a matrix format, we can have for example Eqs. 3.3.4a and 3.3.4b. In Eq. 3.5.8, the first term gives the local address of a string in certain string category. Thus we can treat the first term of Eq. 3.5.10 as the local address of the determinant in a sub-block of the CI vector. Similarly, in Eq. 3.5.8 the second term gives the sum of all the lengths of the previous string categories. And the second term of Eq. 3.5.10 denotes the sum of all the lengths of the previous sub-blocks of the CI vector. Thus \( F_{\text{Cat'}} \) can be defined as,

\[
F_{\text{Cat'}} = \sum_{\text{id Cat}' = 1}^{\text{id Cat}(i_h',i_e')-1} \sum_{\text{id Cat}' = 1}^{\text{id Cat}(\text{MxHole,MxElec})} L \left[ \text{id Cat}' \right] \times L \left[ \text{id Cat} \right] + \sum_{\text{id Cat}' = 1}^{\text{id Cat}(i_h',i_e')-1} L \left[ \text{id Cat}(i_h,i_e) \right] \times L \left[ \text{id Cat}' \right]
\]

(3.5.11)
\( F^{\text{Cat}^{'}} \) may be precomputed for every allowed graph combination. The length of each sub-block of the CI vector is defined as,

\[
L \left[ K^\alpha_{\delta, \iota} K^\beta_{\iota', \iota''} \right] = L \left[ \text{id} \text{Cat}_\alpha (i_h, i_e) \right] \times L \left[ \text{id} \text{Cat}_\beta (i'_e, i'_e) \right] \forall \begin{cases} i_h + i'_h \leq \text{MxHole} \\ i_e + i'_e \leq \text{MxElec} \end{cases} \tag{3.5.12}
\]

Still using the above example, we have an \( \alpha \)-string at hand at the moment. Since the RAS problem is \( \text{MxHole} = \text{MxElec} = 1 \), according to the occupation restriction (Eq. 3.3.3), the only possible \( \beta \)-strings that can be combined with this \( \alpha \)-string are in the category \( \text{Cat}_\beta (0,0) \). Thus the formed determinants fall in the block of \( \left| K^{1,1}_{\alpha} K^{0,0}_{\beta} \right| \) of Eq. 3.3.4a, which is the last block. Assuming the \( \beta \)-string that is combined with the \( \alpha \)-string is the first string in the category \( \text{Cat}_\beta (0,0) \) (thus \( \text{Addr} \left| K_{\beta} \right| = 1 \)), according to Eq. 3.5.9, the local address of the determinant in the block \( \left| K^{1,1}_{\alpha} K^{0,0}_{\beta} \right| \) is calculated as,

\[
\text{Addr} \left| K_{\alpha}, K_{\beta} \right| = (22 - 1) \times L \left[ \text{id} \text{Cat}_\beta (0,0) \right] + 1
= 21 \times 6 + 1 = 127
\]

Since block \( \left| K^{1,1}_{\alpha} K^{0,0}_{\beta} \right| \) is the last block of the CI vector, the value of \( F^{\text{Cat}^{'}} \) is calculated by adding up all the lengths of the previous blocks (Eq. 3.5.11),

\[
F^{\text{Cat}(0,0)}_{\text{Cat}(1,1)} = L \left[ K^{0,0}_{\alpha} K^{0,0}_{\beta} \right] + L \left[ K^{0,0}_{\alpha} K^{0,1}_{\beta} \right] + L \left[ K^{0,0}_{\alpha} K^{1,0}_{\beta} \right] + L \left[ K^{0,0}_{\alpha} K^{1,1}_{\beta} \right]
+ L \left[ K^{1,0}_{\alpha} K^{0,0}_{\beta} \right] + L \left[ K^{1,0}_{\alpha} K^{0,1}_{\beta} \right] + L \left[ K^{1,0}_{\alpha} K^{1,0}_{\beta} \right] + L \left[ K^{1,0}_{\alpha} K^{1,1}_{\beta} \right]
= 36 + 96 + 96 + 576 + 96 + 256 + 96 + 256
= 1508
\]

Thus the determinant \( \left| K \right| = \left| K^{11,32}_{\alpha} K^{0,0}_{\beta} \right| \) will be the 1508+127=1635^{th} element in the CI vector of this RAS problem.

So far we have only focused on the Slater determinant basis, if the basis is changed to spin-adapted Hartree-Waller functions, the process of generating the address of a certain string will be similar. Only the way of generating the addresses of the configurations will be different. In the basis of Hartree-Waller functions, Eq. 3.5.9 turns into (singlet: \( S = 0 \); triplet: \( S = 1 \),

176
3. Algorithm for the Direct RASSCF Method (Part I)

In Section 3.3.3, we have introduced the sign of the symbolic matrix elements. Here we will discuss the numerical values of these signs in details. The model symbolic matrix elements are defined as,

\[
\text{Addr}\{K_\alpha, K_\beta\} = \begin{cases} 
(\text{Addr}\{K_\alpha\} - 1) \times L[\text{id Cat}(i_\alpha', i_\beta')] + \text{Addr}\{K_\beta\} & \text{if } F_{\text{Cat}}' < F_{\text{Cat}}' \\
\frac{2}{2} \text{Addr}\{K_\alpha\} \times \text{Addr}\{K_\alpha\} - 2 + \text{Addr}\{K_\beta\} & \text{if } F_{\text{Cat}}' = F_{\text{Cat}}', S = 0 \\
(\text{Addr}\{K_\beta\} - 1) \times L[\text{id Cat}(i_\beta, i_\beta')] + \text{Addr}\{K_\alpha\} & \text{if } F_{\text{Cat}}' = F_{\text{Cat}}', S = 1 \\
\end{cases}
\]

where the offset \( F_{\text{Cat}}' \) has been redefined as,

\[
F_{\text{Cat}}' = \sum_{\text{id Cat}(i_\alpha, i_\beta) = 1}^{\text{id Cat}-1} \frac{\sum_{\text{id Cat}(i_\alpha, i_\beta) = 1}^{\text{id Cat}-1} L[\text{id Cat}] \times (L[\text{id Cat}]+1)}{2} \text{ if } S = 0 \\
\frac{\sum_{\text{id Cat}(i_\alpha, i_\beta) = 1}^{\text{id Cat}-1} L[\text{id Cat}] \times (L[\text{id Cat}]-1)}{2} \text{ if } S = 1.
\]

3.5.2 The Sign of the Symbolic Matrix Elements

In Section 3.3.3, we have introduced the sign of the symbolic matrix elements. Here we will discuss the numerical values of these signs in details. The model symbolic matrix elements are defined as,

\[
B_{\text{wx}}^{K_{\sigma}l_{\alpha}^\dagger} = \langle K_{\sigma} | a_{\alpha\sigma}^\dagger a_{\sigma\sigma} | l_{\alpha}^m \rangle = \text{sgn}_{\text{wx}}^{K_{\sigma}}
\]

with

\[
\text{sgn}_{\text{wx}}^{K_{\sigma}} = \begin{cases} 
+1 & \text{if } \sum_{n=x+1}^{w-1} b_n \text{ is even} \\
-1 & \text{if } \sum_{n=x+1}^{w-1} b_n \text{ is odd}
\end{cases}
\]

and for the \( a_{\alpha\sigma}^\dagger a_{\gamma\sigma}^\dagger a_{\sigma\sigma} a_{\alpha\sigma} \) operator, we have

\[
B_{\text{wxz}}^{K_{\sigma}l_{\alpha}^\dagger} = \langle K_{\sigma} | a_{\alpha\sigma}^\dagger a_{\gamma\sigma}^\dagger a_{\sigma\sigma} a_{\alpha\sigma} | l_{\sigma}^m \rangle = \text{sgn}_{\text{wxz}}^{K_{\sigma}}
\]

with

\[
\text{sgn}_{\text{wxz}}^{K_{\sigma}} = \begin{cases} 
-\text{sgn}_{\text{wx}}^{K_{\sigma}} \times \text{sgn}_{\text{yz}}^{K_{\sigma}} & \text{if } w > y > x > z \\
+\text{sgn}_{\text{wx}}^{K_{\sigma}} \times \text{sgn}_{\text{yz}}^{K_{\sigma}} & \text{otherwise}
\end{cases}
\]
As indicated in Section 3.3.3, we know when expanding the sign of the model symbolic matrix elements to the full RAS space, a phase factor, \((-1)^n\), should be introduced if one or more of the model orbital indices is dropped in either RAS1 or RAS3 subspaces (Eq. 3.3.22). For example, for the sign of \(B_{ij}^{K_{\xi}^{L_{\xi}}}\), when the index \(i\) lies in the RAS3 subspace and \(j\) is in the RAS2 subspace, \(\text{sgn}_{wx}^{K_{\xi}^{L_{\xi}}}\) must be multiplied by -1 every time the bit value of \(b_{i-1}\) equals 1. The expanded sign of the global symbolic matrix elements is thus calculated as,

\[
B_{ij}^{K_{\xi}^{L_{\xi}}} = \left( K_{\xi}^{L_{\xi}} | a^\dagger_{i2} a_{j2} | L_{\xi}^{L_{\xi}} \right) = \text{sgn}_{ij}^{K_{\xi}^{L_{\xi}}} \times \text{sgn}_{i2}^{K_{\xi}^{L_{\xi}}} \times \text{sgn}_{j2}^{K_{\xi}^{L_{\xi}}},
\]

\[
B_{ijkl}^{K_{\xi}^{L_{\xi}}} = \left( K_{\xi}^{L_{\xi}} | a^\dagger_{i2} a^\dagger_{k2} a_{j2} a_{l2} | L_{\xi}^{L_{\xi}} \right) = \text{sgn}_{ijkl}^{K_{\xi}^{L_{\xi}}} \times \text{sgn}_{i2}^{K_{\xi}^{L_{\xi}}} \times \text{sgn}_{j2}^{K_{\xi}^{L_{\xi}}} \times \text{sgn}_{k2}^{K_{\xi}^{L_{\xi}}} \times \text{sgn}_{l2}^{K_{\xi}^{L_{\xi}}},
\]

where \(\text{sgn}_{ij}^{K_{\xi}^{L_{\xi}}} \times \text{sgn}_{ijkl}^{K_{\xi}^{L_{\xi}}} = (-1)^n\) (\(X \in \{1,3\}\)) is the phase factor.

### 3.5.3 Brief Review and Examples of the Propagation Rules (\(P^{(\text{RAS}X)}\))

As we have mentioned above, we have to expand the model string pair to the full RAS space. In this section we will present the process of the expansion, known as the propagation rules (\(P^{(\text{RAS}X)}\)). This approach is given by Klene et al [44] to efficiently compute all full RAS space string pairs, \(\{ K_{\xi}^{L_{\xi}}, L_{\xi}^{L_{\xi}} \} \), that give non-zero contribution from a given model string pair, \(\{ K_{\xi}^{m}, L_{\xi}^{m} \} \). As shown in Section 3.4.1, the sub string of the RAS2 space is always kept fixed during the expansion while the model RAS1 substrings and model RAS3 substrings are expanded. This is because after expansion, the model RAS2 substring will remain the same as the full RAS space RAS2 substring. Thus the propagation rules focuses mainly on the expansion of the model RAS1 and model RAS3 subspaces. When the first RAS1/RAS3 substring address, \(\text{Addr}\{ K_{\xi}^{\text{RASX}}(1) \} \), is known, this approach derives all the following substring addresses efficiently without
applying Eq. 3.5.6 to calculate these addresses. The detail about this approach can be found in reference [44]. Here we will only give a couple of examples to show how this approach works.

There are in total five cases representing 0, 1, 2, 3, or all 4 global orbital indices, which are in either the RAS1 or RAS3 subspace in this approach. In the following, we will choose the excitations of $2 \rightarrow 2$ and $1 \rightarrow 3$ as examples to illustrate how the propagation rules works. However, before we start the examples, we need to remind ourselves of the concept of the model space. In Section 3.3.3, the number of model space orbitals and $\xi$-electrons is defined by Eq. 3.3.13 and Eqs. 3.3.14, respectively. Thus for example for the global string, $K^{11,32}_\alpha$ (when $\text{MxHole} = \text{MxElec} = 1$), the model space can be constructed as Figure 3.4 shows.

Obviously, the propagation rules operate the reverse process that Figure 3.4 shows. However, if we set $\text{MxHole} = \text{MxElec} = 1$, as shown in Figure 3.4, the propagation process will not be very clear to us. Thus in the following examples, although we will still use the global string, $K^{11,32}_\alpha$, in order to clearly explain how the propagation rules works, we will set $\text{MxHole} = \text{MxElec} \geq 2$. In this case, the model space shown in Figure 3.4 now becomes $[01 \ 0101 \ 10]$.

**Example 1:** $2 \rightarrow 2$ The first example is the model space excitation $a_{wa}^\dagger a_{wa}$ with $w=5$ and $x=4$. We start with the reduced model string 010110. By inserting the bits $b_4, b_5$ into the appropriate positions of the reduced model string ($K^m_{\alpha}: b_4 = 0, b_5 = 1; L^m_{\alpha}: b_4 = 1, b_5 = 0$), a pair of model strings, $K^m_{\alpha} = \begin{bmatrix} 01 & 0101 & 10 \end{bmatrix}$ and $L^m_{\alpha} = \begin{bmatrix} 01 & 0011 & 10 \end{bmatrix}$, is obtained. In both model strings, the number of holes in the RAS1 model subspace is $i_h = 1$, and the number of electrons in the RAS3 model subspace is $i_e = 1$. Thus we know
the expanded string pairs are all in the string category $\text{Cat}_\alpha(1,1)$. Since the occupation pattern of RAS2 subspace is fixed, we can get in total 

$$\begin{pmatrix} M(\text{RAS1})_i \\ M(\text{RAS3})_i \end{pmatrix} = 16 \text{ global } \alpha \text{-string pairs, } \{K_{\alpha}^{1,1}, L_{\alpha}^{1,1}\}.$$  

Figure 3.5 gives the process of obtaining the 16 string pairs. $i$ and $j$ indicate the full RAS space orbital indices. Since the excitation type is $2 \rightarrow 2$, the model orbital integral, $(w|x) = (i|j)$, and the model symbolic matrix elements, $\text{sgn}_{ij}^{K_{\alpha}^{m}}$, will remain constant during the process of the expansion.

Figure 3.4 The model string generating process from a global string, $\left|K_{\alpha}^{11,32}\right>$, when $M_{\text{Hole}} = M_{\text{Elec}} = 1$. The generated model string can represent all the global string given in Section 3.3.1.
3. Algorithm for the Direct RASSCF Method (Part I)

**Figure 3.5 Example 1:** the expansion process of any of the global orbital indices is in neither RAS1 subspace nor RAS3 subspace. From this Figure we can see once the first substring address of RAS1/RAS3 subspace is known, it is easy to get the addresses of the rest substrings within this subspace without using Eq. 3.5.6.

From Figure 3.5 we can conclude that when 0 global orbital indices are in either RAS1 or RAS3 subspaces, the rest of the substring addresses within the RAS1 or RAS3 subspaces can be calculated by using the first address of the substring within this subspace,

\[
\text{Addr}\left\{K^{\text{RASX}}_{\alpha}(n)\right\} = \text{Addr}\left\{K^{\text{RASX}}_{\alpha}(n-1)\right\} + 1 \left[ 2 \leq n \leq \left\lfloor \frac{M(\text{RASX})}{N_\xi \text{ in RASX}} \right\rfloor \right]. \tag{3.5.21}
\]

The application of Eq. 3.5.6 for the addresses of the rest of the substrings has been completely avoided. Only the address of the first string should be calculated via Eq. 3.5.6.

**Example 2:** 1 → 3 For our second example let w=8, x=1. Inserting \(b_w\) and \(b_x\) into the reduced model string 001011 yields \(K^m_{\alpha} = \text{010110}\) and \(L^m_{\alpha} = \text{001110}\), which represent the string categories of Cat\(_{\alpha}(1,1)\) and Cat\(_{\alpha}(0,0)\), respectively. Since the excitation involve both the RAS1 and RAS3 subspaces, the expanded orbital integral, \((i|j)\), and the symbolic matrix elements, \(\text{sgn}_{ij}^{K_{\alpha}}\), won’t be constant any more. Figure 3.6

---

<table>
<thead>
<tr>
<th>(i)</th>
<th>(j)</th>
<th>RAS3</th>
<th>RAS2</th>
<th>RAS1</th>
<th>Addr({K_{\alpha}^{\text{RAS1}}})</th>
<th>Addr({K_{\alpha}^{\text{RAS2}}})</th>
<th>RAS3</th>
<th>RAS2</th>
<th>RAS1</th>
<th>Addr({L_{\alpha}^{\text{RAS1}}})</th>
<th>Addr({L_{\alpha}^{\text{RAS2}}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>0001</td>
<td>0101</td>
<td>0111</td>
<td>1</td>
<td>1</td>
<td>0001</td>
<td>0011</td>
<td>0111</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>0010</td>
<td>0111</td>
<td></td>
<td>2</td>
<td>1</td>
<td>0010</td>
<td>0111</td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>0100</td>
<td>0111</td>
<td></td>
<td>3</td>
<td>1</td>
<td>0100</td>
<td>0111</td>
<td></td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>0001</td>
<td>0111</td>
<td></td>
<td>4</td>
<td>1</td>
<td>0001</td>
<td>1011</td>
<td></td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>1000</td>
<td>1110</td>
<td></td>
<td>4</td>
<td>2</td>
<td>1000</td>
<td>1110</td>
<td></td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

181
shows the process of the expansion. From Figure 3.6 we can see the orbital integral changes all the time.

![Reduced model string](image)

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>RAS3</th>
<th>RAS2</th>
<th>RAS1</th>
<th>Addr${K_{\alpha}^{RAS1}}$</th>
<th>Addr${K_{\alpha}^{RAS2}}$</th>
<th>Addr${L_{\alpha}^{RAS1}}$</th>
<th>Addr${L_{\alpha}^{RAS2}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>0001</td>
<td>0101</td>
<td>0111</td>
<td>1</td>
<td>1</td>
<td>0000</td>
<td>0101</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>0010</td>
<td></td>
<td>0111</td>
<td>2</td>
<td>1</td>
<td>0000</td>
<td>1111</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>0100</td>
<td></td>
<td>0111</td>
<td>3</td>
<td>1</td>
<td>0000</td>
<td>1111</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>1000</td>
<td></td>
<td>0111</td>
<td>4</td>
<td>1</td>
<td>0000</td>
<td>1111</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>0001</td>
<td>1011</td>
<td></td>
<td>1</td>
<td>2</td>
<td>0000</td>
<td>1111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1000</td>
<td>1110</td>
<td></td>
<td>4</td>
<td>4</td>
<td>0000</td>
<td>1111</td>
</tr>
</tbody>
</table>

Figure 3.6 Example 2: the excitation from RAS1 subspace to RAS3 subspace. Since in this excitation both RAS1 and RAS3 subspaces are involved, the expanded orbital integral, $\langle i \mid j \rangle$, and the symbolic matrix elements, $\text{sgn}_{ij}^{K_{\alpha}^{RAS1, RAS2}}$, will not remain constant any more.

One can certainly calculate the addresses of the substrings, $\text{Addr}\{K_{\alpha}^{RAS1}\}$ and $\text{Addr}\{K_{\alpha}^{RAS2}\}$, by using Handy’s index equation (Eq. 3.5.6). But from the addresses for consecutive strings one may find there is a simpler way of producing them. For this example, one can calculate the addresses of the rest of the substrings by using the address of the first substring as,

$$
\text{Addr}\{K_{\alpha}^{RASX} (n)\} = \text{Addr}\{K_{\alpha}^{RASX} (n-1)\} + 1
$$

$$
\text{Addr}\{L_{\alpha}^{RASX} (n)\} = \text{Addr}\{L_{\alpha}^{RASX} (n-1)\}
$$

where $2 \leq n \leq M(\text{RASX}) $. (3.5.22)

The propagation approach provides a very simple way to generate the addresses of the substrings when the address of the first substring is available. This avoids using Handy’s index scheme (Eq. 3.5.6) frequently, which is less efficient than the propagation rules.
3.6 Implementation Details

In this section, we will describe the implementation details of the RASSCF matrix multiplication method for the 7 selected cases (Eqs. 3.4.6 and 3.4.8). All reduced lists of model space and RAS subspaces and the associated Handy’s index matrices are precomputed once per iteration. When the Slater determinant basis is used, the transposed form of the CI vector, $C_{L}^{n-1}$ in Eq. 3.3.5, is generated once per iteration. However, when the Hartree-Waller spin adapted functions (singlet or triplet) basis is used, a normalized matrix of the CI vector must be generated instead of a transposed CI vector. The outer loop is over the model space orbital indices $w$ and $x$ and the inner loop is over $y$ and $z$. The computation of model space string pairs is analogous to string pair generation for the CASSCF / Full CI method described in Chapter 2. Once a model string pair, $\{K_{\xi}^{m}, L_{\xi}^{m}\}$, is obtained, the information about its category and propagation rules for both the RAS1 and RAS3 subspaces is identified. This information is then used to assemble the list of integrals into a vector, which will be accessed in a sequential order during the matrix multiplication process to update the $\sigma$-vector. Then the excitation lists, denoted as $E_{\xi}^{\text{RASX}}$, for a $wx$ model orbital index pair is generated via applying the propagation rules to the RAS subspace model strings, RAS1$^{m}$ and RAS3$^{m}$. With the generated elements of the excitation lists for the $yz$ model orbital index pairs that correspond to the $wx$ index pair (Eq. 3.3.24), the corresponding elements in the integral vector are used to generate the intermediate matrix $M$ (Eqs. 3.4.6b and 3.4.8b). Then the generated excitation list from the model orbital index pair $wx$ is used to multiply the obtained matrix $M$ to update the CI vector. This new implementation is also parallelized. But we will postpone the discussion about this until the parallelism section. Algorithms 3.1 and 3.2 show the appropriate algorithms for the linear and bilinear contributions respectively. The implementation details and discussions follow. A detailed description of the subroutine calling sequence of this algorithm can be found in Appendix C.
Algorithm 3.1: RAS linear contribution to the $\sigma$-vector updating.\( \mathcal{P} \) indicates the propagation rules applied to $\text{RAS}(X \in \{1,3\})$ subspace.

$$C \text{ vector restructuring: } C^T \text{ for SD, } C^{HW} \text{ for HW functions}$$

(parallel) loop over $w,x$

(1e linear contribution)

$$\Rightarrow K^m_\xi, L^m_\xi, \text{sgn}_\xi, \text{Cat}_\xi(i_h,i_e), \text{Addr}\{K^\text{RAS2}_\xi\}, \text{Cat}_\xi(i_h',i_e'), \text{Addr}\{L^\text{RAS2}_\xi\},$$

$$\mathcal{P}(\text{RAS1}), \mathcal{P}(\text{RAS3}), \text{and integral list } \{(i|j)\}$$

apply $\mathcal{P}(\text{RAS1})$ to assemble $\mathcal{I}^\text{RAS1}_\xi$

apply $\mathcal{P}(\text{RAS3})$ to assemble $\mathcal{I}^\text{RAS3}_\xi$

$$\Rightarrow (i|j) B_{ij}^{K^\text{RAS2}_\xi,K^\text{RAS2}_\xi}$$

loop over all allowed string categories $\text{Cat}_\gamma(i''_h,i''_e)$

loop over all $\gamma$-strings in $\text{Cat}_\gamma(i''_h,i''_e)$

$$\sigma^{\text{linear}}(K^\text{RAS2}_\xi,K^\text{RAS3}_\gamma) \leftarrow \sum_{i''_h,i''_e} (i|j) B_{ij}^{K^\text{RAS2}_\xi,K^\text{RAS3}_\gamma} C(L^\text{RAS3}_\xi,K^\text{RAS3}_\gamma)$$

end loop $\gamma$-strings

end loop $\text{Cat}(i''_h,i''_e)$

(2e linear contribution)

loop over $y,z$

insert bits $b_w,b_s,b_y,b_z \Rightarrow K^m_\xi, L^m_\xi, \text{sgn}_\xi, \text{Cat}_\xi(i_h,i_e), \text{Addr}\{K^\text{RAS2}_\xi\}, \text{Cat}_\xi(i_h',i_e'),$\n
$$\text{Addr}\{L^\text{RAS2}_\xi\}, \mathcal{P}(\text{RAS1}), \mathcal{P}(\text{RAS3}), \text{and integral list } \{(i|j|k)\}$$

apply $\mathcal{P}(\text{RAS1})$ to assemble $\mathcal{I}^\text{RAS1}_\xi$

apply $\mathcal{P}(\text{RAS3})$ to assemble $\mathcal{I}^\text{RAS3}_\xi$

$$\Rightarrow (i|j|k) B_{ijkl}^{K^\text{RAS2}_\xi,K^\text{RAS3}_\gamma}$$

loop over all allowed string categories $\text{Cat}_\gamma(i''_h,i''_e)$

loop over all $\gamma$-strings in $\text{Cat}_\gamma(i''_h,i''_e)$

$$\sigma^{\text{linear}}(K^\text{RAS2}_\xi,K^\text{RAS3}_\gamma) = \frac{1}{2} \sum_{i''_h,i''_e} (i|j|k) B_{ijkl}^{K^\text{RAS2}_\xi,K^\text{RAS3}_\gamma} C(L^\text{RAS3}_\xi,K^\text{RAS3}_\gamma)$$

end loop $\gamma$-strings

end loop $\text{Cat}(i''_h,i''_e)$

end loop $y,z$

end (parallel) loop $w,x$
3. Algorithm for the Direct RASSCF Method (Part I)

(parallel) loop over \( w, x \)

when \( w, x \) are both in RAS2 subspace, then

insert bits \( b_w, b_x \Rightarrow K^m_w, L^m_x \)

\( \Rightarrow \sgn_x \cdot \text{Cat}_x (i_x', i_x) \cdot \text{Addr} \{K^{\text{RAS2}}_x\}, \text{Cat}_x (i_h, i_e) \cdot \text{Addr} \{L^{\text{RAS2}}_x\}, \)

\( \mathcal{P}(\text{RAS1}), \mathcal{P}(\text{RAS3}) \)

apply \( \mathcal{P}(\text{RAS1}) \) to assemble \( T^{\text{RAS1}}_x \)

apply \( \mathcal{P}(\text{RAS3}) \) to assemble \( T^{\text{RAS3}}_x \)

\( \Rightarrow B_{ij}^{K^{\text{RAS1}}_w, L^{\text{RAS1}}_x} = \{ \text{Addr} \{K^{\text{RAS1}}_w\}, \text{Addr} \{L^{\text{RAS1}}_x\}, \sgn_y = \sgn_y^m \times \sgn_{\text{RAS1}}^y \times \sgn_{\text{RAS3}}^y \} \)

(matrix \( M \) generation)

loop over \( y, z (yz \leq wx \) or \( yz \geq wx \) \)

insert bits \( b_y, b_z \Rightarrow K^m_y, L^m_z \)

\( \Rightarrow \sgn_y \cdot \text{Cat}_y (i_y', i_y') \cdot \text{Addr} \{K^{\text{RAS2}}_y\}, \text{Cat}_y (i_h', i_e') \cdot \text{Addr} \{L^{\text{RAS2}}_y\}, \)

\( \mathcal{P}(\text{RAS1}), \mathcal{P}(\text{RAS3}), \text{integral list} \{ \{ij|kl\} \} \)

apply \( \mathcal{P}(\text{RAS1}) \) to assemble \( T^{\text{RAS1}}_y \)

apply \( \mathcal{P}(\text{RAS3}) \) to assemble \( T^{\text{RAS3}}_y \)

\( \Rightarrow (ij|kl) \cdot B_{kl}^{K^{\text{RAS1}}_y, L^{\text{RAS1}}_z} = \{ \text{Addr} \{K^{\text{RAS1}}_y\}, \text{Addr} \{L^{\text{RAS1}}_z\}, \sgn_y = \sgn_y^m \times \sgn_{\text{RAS1}}^y \times \sgn_{\text{RAS3}}^y \} \)

loop over all allowed string categories : \( \text{Cat}_z (i_z', i_z') \)

loop over all string \( L_z \) in category \( \text{Cat}_z (i_z', i_z') \)

\[ M_{ij}^{K^{\text{RAS1}}_y, L^{\text{RAS1}}_z} = \frac{1}{2} \sum_{l_x' \in \delta_x'} (ij|kl) \cdot B_{kl}^{K^{\text{RAS1}}_y, L^{\text{RAS1}}_z} \cdot C^T (L^{\text{RAS1}}_z, K^{\text{RAS1}}_y) \]

end loop all \( L_z \) in \( \text{Cat}_z (i_z', i_z') \)

end loop allowed \( \text{Cat}_z (i_z', i_z') \)

end loop \( y, z \)

(\( \sigma \) bilinear updating)

\[ \sigma^{\text{bilinear}} (K^{\text{RAS1}}_x, K^{\text{RAS1}}_y) = \sigma^{\text{bilinear}} (K^{\text{RAS1}}_x, K^{\text{RAS1}}_y) + \sum_{l_x' \in \delta_x'} B_{ij}^{K^{\text{RAS1}}_x, L^{\text{RAS1}}_y} \left[ M_{ij}^{K^{\text{RAS1}}_x, L^{\text{RAS1}}_y} \right]^T \forall K^{\text{RAS1}}_x, K^{\text{RAS1}}_y \]

end (parallel) loop \( w, x \) (in RAS2 subspace)

Algorithm 3.2: RAS bilinear contribution to the \( \sigma \)-vector updating \( (w \) and \( x \) are both in RAS2 subspace. If \( w \) and \( x \) are not both in RAS2 subspace, the current method will be used)
3.6.1 CI Vector Restructure

In the CASSCF matrix multiplication method, in order to carry out the matrix multiplication, the CI vector has been converted into a 2-dimensional matrix format. The same operation is also required for the RASSCF matrix multiplication method. However, because of the introduction of the occupation restriction in the RAS subspaces, the process of the CI vector restructure is completely different from the process of the CASSCF method. As we stated in Section 3.3.1, the RASSCF CI vector is divided into many sub-blocks, of which the dimension is determined by the lengths of the corresponding string categories. The restructuring of the CI vector of the RASSCF method is actually a process of reorganizing the data in these sub-blocks. Moreover, since the values in the CI vector, $C_{l}^{n-1}$, do not change during a Davidson iteration, the restructuring of the CI vector needs to be carried out only once per iteration before the loop over the model orbital indices starts. In the following we will use the CI vector of a RAS problem with MxHole = MxElec = 2 as example to demonstrate how to reorganize the CI vector into a proper vector that can be used in the matrix multiplication method. For other RAS systems with different numbers of MxHole and MxElec, the process is similar.

3.6.1.1 CI Vector Restructure for Slater Determinant Basis

As indicated in Section 3.3.1, for Slater determinant basis, the CI vector used in the current method, written in a matrix format, is given in Eq. 3.3.4b (MxHole = MxElec = 2),
In CASSCF wavefunction, given the symmetrical property of the configurations, \(|K_\alpha K_\beta\rangle = \langle K_\beta K_\alpha|\), the dimension of the CI vector for the HW function basis is about half.
of the dimension of the CI vector for the Slater determinant basis (e.g. Eqs. 1.5.7 and 1.5.8 vs. Eq. 1.5.5 in Chapter 1). Similarly, if we write the CI vector for the Hartree-Waller functions in a matrix format, the CI vector matrix for Hartree-Waller function basis will be,

\[
C = \begin{pmatrix}
|K^0_0 K^0_0 \rangle & |K^0_0 K^0_1 \rangle & |K^0_0 K^{1,0} \rangle & |K^0_0 K^{2,0} \rangle & |K^0_0 K^2_1 \rangle & |K^0_0 K^2_2 \rangle \\
|K^0_1 K^0_0 \rangle & |K^0_1 K^0_1 \rangle & |K^0_1 K^{1,0} \rangle & |K^0_1 K^{2,0} \rangle & |K^0_1 K^2_1 \rangle & |K^0_1 K^2_2 \rangle \\
|K^0_2 K^0_0 \rangle & |K^0_2 K^0_1 \rangle & |K^0_2 K^{1,0} \rangle & |K^0_2 K^{2,0} \rangle & |K^0_2 K^2_1 \rangle & |K^0_2 K^2_2 \rangle \\
|K^{1,0}_0 K^0_0 \rangle & |K^{1,0}_0 K^0_1 \rangle & |K^{1,0}_0 K^{1,0} \rangle & |K^{1,0}_0 K^{2,0} \rangle & |K^{1,0}_0 K^2_1 \rangle & |K^{1,0}_0 K^2_2 \rangle \\
|K^{1,0}_1 K^0_0 \rangle & |K^{1,0}_1 K^0_1 \rangle & |K^{1,0}_1 K^{1,0} \rangle & |K^{1,0}_1 K^{2,0} \rangle & |K^{1,0}_1 K^2_1 \rangle & |K^{1,0}_1 K^2_2 \rangle \\
|K^{1,0}_2 K^0_0 \rangle & |K^{1,0}_2 K^0_1 \rangle & |K^{1,0}_2 K^{1,0} \rangle & |K^{1,0}_2 K^{2,0} \rangle & |K^{1,0}_2 K^2_1 \rangle & |K^{1,0}_2 K^2_2 \rangle \\
|K^{2,0}_0 K^0_0 \rangle & |K^{2,0}_0 K^0_1 \rangle & |K^{2,0}_0 K^{1,0} \rangle & |K^{2,0}_0 K^{2,0} \rangle & |K^{2,0}_0 K^2_1 \rangle & |K^{2,0}_0 K^2_2 \rangle \\
|K^{2,0}_1 K^0_0 \rangle & |K^{2,0}_1 K^0_1 \rangle & |K^{2,0}_1 K^{1,0} \rangle & |K^{2,0}_1 K^{2,0} \rangle & |K^{2,0}_1 K^2_1 \rangle & |K^{2,0}_1 K^2_2 \rangle \\
|K^{2,0}_2 K^0_0 \rangle & |K^{2,0}_2 K^0_1 \rangle & |K^{2,0}_2 K^{1,0} \rangle & |K^{2,0}_2 K^{2,0} \rangle & |K^{2,0}_2 K^2_1 \rangle & |K^{2,0}_2 K^2_2 \rangle \\
|K^{2,1}_0 K^0_0 \rangle & |K^{2,1}_0 K^0_1 \rangle & |K^{2,1}_0 K^{1,0} \rangle & |K^{2,1}_0 K^{2,0} \rangle & |K^{2,1}_0 K^2_1 \rangle & |K^{2,1}_0 K^2_2 \rangle \\
|K^{2,1}_1 K^0_0 \rangle & |K^{2,1}_1 K^0_1 \rangle & |K^{2,1}_1 K^{1,0} \rangle & |K^{2,1}_1 K^{2,0} \rangle & |K^{2,1}_1 K^2_1 \rangle & |K^{2,1}_1 K^2_2 \rangle \\
|K^{2,2}_0 K^0_0 \rangle & |K^{2,2}_0 K^0_1 \rangle & |K^{2,2}_0 K^{1,0} \rangle & |K^{2,2}_0 K^{2,0} \rangle & |K^{2,2}_0 K^2_1 \rangle & |K^{2,2}_0 K^2_2 \rangle \\
|K^{2,2}_1 K^0_0 \rangle & |K^{2,2}_1 K^0_1 \rangle & |K^{2,2}_1 K^{1,0} \rangle & |K^{2,2}_1 K^{2,0} \rangle & |K^{2,2}_1 K^2_1 \rangle & |K^{2,2}_1 K^2_2 \rangle \\
|K^{2,2}_2 K^0_0 \rangle & |K^{2,2}_2 K^0_1 \rangle & |K^{2,2}_2 K^{1,0} \rangle & |K^{2,2}_2 K^{2,0} \rangle & |K^{2,2}_2 K^2_1 \rangle & |K^{2,2}_2 K^2_2 \rangle 
\end{pmatrix},
\tag{3.6.3}
\]

which is about half the dimension of the CI vector for the SD basis (Eq. 3.3.4b). This is because of the symmetry property of the configurations of the RAS wavefunction: \[|K^{i, \beta}_a K^{\beta, i}_a \rangle = |K^{\beta, i}_a K^{i, \beta}_a \rangle\]. In the matrix multiplication method, however, we should normalize this matrix into a full matrix format. For example, for the Hartree-Waller singlet basis, the normalized matrix is,
where \( |K_{\alpha i_h, i_e}^h, K_{\beta i_h, i_e}^h\rangle = |K_{\alpha i_{h'}, i_{e'}}^h, K_{\beta i_{h'}, i_{e'}}^h\rangle^T\) (\(i_h \neq i_h'\) or \(i_e \neq i_e'\)). For the diagonal blocks (\(i_h = i_h'\) and \(i_e = i_e'\)), when the global \(\alpha\)-strings and global \(\beta\)-strings are identical, \(|K_{\alpha i_h, i_e}^h\rangle = |K_{\beta i_h, i_e}^h\rangle\), the elements at the position of \(C_{\mathrm{sin}} \left| K_{\alpha i_h, i_e}^h, K_{\beta i_h, i_e}^h \right\rangle\), which are the diagonal elements in the diagonal blocks, should be multiplied by \(\sqrt{2}\). This is because the singlet two-body creation operator is symmetric in the matrix indices [123]. Thus for the diagonal blocks, when carrying out the normalization, all diagonal elements will be multiplied by \(\sqrt{2}\), and all off-diagonal elements are symmetrically ordered. For example the elements in the first block of the normalized matrix will be,

\[
\begin{pmatrix}
\sqrt{2} C_1 & C_2 & \cdots & C_{K_{\mathrm{u}}(K_{\mathrm{u}}-1)+1} \frac{1}{2} \\
C_2 & \sqrt{2} C_3 & \cdots & C_{K_{\mathrm{u}}(K_{\mathrm{u}}-1)+2} \frac{1}{2} \\
\vdots & \vdots & \ddots & \vdots \\
C_{K_{\mathrm{u}}(K_{\mathrm{u}}-1)+1} \frac{1}{2} & C_{K_{\mathrm{u}}(K_{\mathrm{u}}-1)+2} \frac{1}{2} & \cdots & \sqrt{2} C_{K_{\mathrm{u}}(K_{\mathrm{u}}-1)+K_{\beta}} \\
\end{pmatrix}
\]  

(3.6.4)

The subscript of the elements in Eq. 3.6.4 indicates the local address of the determinant (Eq. 3.5.13) in the corresponding sub-block of the CI vector, e.g. the first block in Eq. 3.6.2. From Eq. 3.6.4 we can see after normalization all diagonal elements have been multiplied by \(\sqrt{2}\), and the block has been normalized as a full matrix rather than a lower triangular matrix in the first block of Eq. 3.6.2. Obviously, comparing Eq. 3.6.3 with Eq. 3.6.2, we can see the dimension of the normalized matrix is about two times of the dimension of the original CI vector.

For the Hartree-Waller triplet basis, according to the Pauli principle, the elements of the diagonal positions of the normalized matrix will vanishes. Moreover, the triplet two-body creation operator is anti-symmetrical. Thus we can generate the normalized CI vector in a matrix format as,
the normalized matrix as we define the dimension of the current CI vector as

\[
C_{\text{HW}} = \begin{pmatrix}
K^{0,0}_a K^{0,0}_\beta & K^{0,0}_a K^{0,1}_\beta & K^{0,0}_a K^{0,2}_\beta & K^{0,0}_a K^{1,0}_\beta & K^{0,0}_a K^{1,1}_\beta & K^{0,0}_a K^{1,2}_\beta & K^{0,0}_a K^{2,0}_\beta & K^{0,0}_a K^{2,1}_\beta & K^{0,0}_a K^{2,2}_\beta \\
-K^{0,1}_a K^{0,0}_\beta & K^{0,1}_a K^{0,1}_\beta & K^{0,1}_a K^{0,2}_\beta & K^{0,1}_a K^{1,0}_\beta & K^{0,1}_a K^{1,1}_\beta & K^{0,1}_a K^{1,2}_\beta & K^{0,1}_a K^{2,0}_\beta & K^{0,1}_a K^{2,1}_\beta & K^{0,1}_a K^{2,2}_\beta \\
-K^{0,2}_a K^{0,0}_\beta & -K^{0,2}_a K^{0,1}_\beta & K^{0,2}_a K^{0,2}_\beta & K^{0,2}_a K^{1,0}_\beta & K^{0,2}_a K^{1,1}_\beta & K^{0,2}_a K^{1,2}_\beta & K^{0,2}_a K^{2,0}_\beta & K^{0,2}_a K^{2,1}_\beta & K^{0,2}_a K^{2,2}_\beta \\
-K^{1,0}_a K^{0,0}_\beta & -K^{1,0}_a K^{0,1}_\beta & -K^{1,0}_a K^{0,2}_\beta & K^{1,0}_a K^{1,0}_\beta & K^{1,0}_a K^{1,1}_\beta & K^{1,0}_a K^{1,2}_\beta & K^{1,0}_a K^{2,0}_\beta & K^{1,0}_a K^{2,1}_\beta & K^{1,0}_a K^{2,2}_\beta \\
-K^{1,1}_a K^{0,0}_\beta & -K^{1,1}_a K^{0,1}_\beta & -K^{1,1}_a K^{0,2}_\beta & -K^{1,1}_a K^{1,0}_\beta & K^{1,1}_a K^{1,1}_\beta & K^{1,1}_a K^{1,2}_\beta & K^{1,1}_a K^{2,0}_\beta & K^{1,1}_a K^{2,1}_\beta & K^{1,1}_a K^{2,2}_\beta \\
-K^{2,0}_a K^{0,0}_\beta & -K^{2,0}_a K^{0,1}_\beta & -K^{2,0}_a K^{0,2}_\beta & -K^{2,0}_a K^{1,0}_\beta & -K^{2,0}_a K^{1,1}_\beta & K^{2,0}_a K^{1,2}_\beta & K^{2,0}_a K^{2,0}_\beta & K^{2,0}_a K^{2,1}_\beta & K^{2,0}_a K^{2,2}_\beta \\
-K^{2,1}_a K^{0,0}_\beta & -K^{2,1}_a K^{0,1}_\beta & -K^{2,1}_a K^{0,2}_\beta & -K^{2,1}_a K^{1,0}_\beta & -K^{2,1}_a K^{1,1}_\beta & -K^{2,1}_a K^{1,2}_\beta & K^{2,1}_a K^{2,0}_\beta & K^{2,1}_a K^{2,1}_\beta & K^{2,1}_a K^{2,2}_\beta \\
-K^{2,2}_a K^{0,0}_\beta & -K^{2,2}_a K^{0,1}_\beta & -K^{2,2}_a K^{0,2}_\beta & -K^{2,2}_a K^{1,0}_\beta & -K^{2,2}_a K^{1,1}_\beta & -K^{2,2}_a K^{1,2}_\beta & -K^{2,2}_a K^{2,0}_\beta & K^{2,2}_a K^{2,1}_\beta & K^{2,2}_a K^{2,2}_\beta \\
\end{pmatrix}, \quad (3.6.5)
\]

where \( \begin{pmatrix} K^{i'_h,i'_c} K^{i_h,i_c}_\beta \end{pmatrix} = - \begin{pmatrix} K^{i'_h,i'_c} K^{i_h,i_c}_\alpha \end{pmatrix}^T \) (\( \ell_h \neq \ell_h' \) or \( \ell_c \neq \ell_c' \)). And the normalized diagonal blocks, e.g. the first block will be,

\[
\begin{pmatrix} K^{0,0}_a K^{0,0}_\beta \end{pmatrix}_{\text{HW}} = \begin{pmatrix}
0 & C_1 & \cdots & C_{(K_a-2)(K_a-1)}/2,1 \\
-C_1 & 0 & \cdots & C_{(K_a-2)(K_a-1)}/2,2 \\
\vdots & \vdots & \ddots & \vdots \\
-\frac{C_{(K_a-2)(K_a-1)}}{2} & -\frac{C_{(K_a-2)(K_a-1)}}{2} & \cdots & 0
\end{pmatrix}. \quad (3.6.6)
\]

Again, the subscript of the elements in Eq. 3.6.6 indicates the local addresses of the determinants in this block given by Eq. 3.5.13. From Eq. 3.6.6 we can see the diagonal elements in the diagonal blocks are all zeros, and the block matrix is anti-symmetrical. Similarly, comparing Eq. 3.6.5 with Eq. 3.6.2 we know the dimension of the normalized matrix for the Hartree-Waller triplet basis is about two times larger than the dimension of the CI vector of the current method using HW triplet basis. This is because after normalization, the upper triangular matrix and the diagonal elements are added. Thus if we define the dimension of the current CI vector as \( D_{\text{HW}} \), we can have the dimension of the normalized matrix as \( 2 \cdot D_{\text{HW}} + N_{\text{global}}^{\alpha} \), where \( N_{\text{global}}^{\alpha} \) indicates the total number of global \( \alpha \) -strings.
3.6.2 Extension of the Model Subspaces

So far we have mentioned many times that when a pair of model strings is available, it is necessary to expand this pair to the full RAS space. However, care must be taken to ensure each term \( K_\xi^m \xi | a_\xi^\dagger a_\xi | L_\xi^m \) in the full RAS space is represented by only one model space term \( K_\xi^m \xi | a_\xi^\dagger a_\xi | L_\xi^m \), in order to avoid multiple contributions. Figure 3.7 illustrates the problem. From Figure 3.7 we can see that the same global string pair can be generated from different model string pairs. Each model string pair is related to a different excitation operator in the model space, e.g. \( a_8^\dagger a_7^\dagger \) and \( a_{10}^\dagger a_7^\dagger \), respectively.

\[
\begin{align*}
K_\alpha^m (1) & \quad \leftrightarrow \quad K_\alpha^{01,(14)} \\
L_\alpha^m (1) & \quad \leftrightarrow \quad L_\alpha^{01,(9)} \\
K_\alpha^m (2) & \quad \leftrightarrow \quad K_\alpha^{m (2)}
\end{align*}
\]

Figure 3.7 Unwanted double contributions may arise when two model string pairs generated by different \( wx \) pairs result in the same global string pair through the application of the propagation rules concept. In this figure we can see both model string pairs \( K_\xi^m | a_8^\dagger a_7^\dagger | L_\xi^m \) and \( K_\xi^m | a_{10}^\dagger a_7^\dagger | L_\xi^m \) give the global string pair \( K_\xi^{01,(14)} | a_{14}^\dagger a_9^\dagger | L_\xi^{01,(9)} \).

This problem is easily avoided by requiring the model space indices, \( \{w,x,y,z\} \), to be flush-right in the RASX\(^m\) model subspaces. In other words, the orbital indices in the model RAS1 and model RAS3 subspaces will take the lowest possible orbital level to represent the global string and integral set. It is then straightforward to see that the only restrictions on the values of MxHole and MxElec are,

\[
0 \leq \text{MxHole} \leq 2 \cdot M (\text{RAS1}) \\
0 \leq \text{MxElec} \leq 2 \cdot M (\text{RAS3})
\]

In the special case where both MxHole and MxElec assume their maximum (or minimum) values the active spaces of CAS and RAS (or RAS2) become identical.
3.6.3 Dimension of the X Array (Eqs. 3.4.6a and 3.4.8a)

In this section we will discuss the memory requirement of generating the matrix $X$ (Eqs. 3.4.6a and 3.4.8a) that is multiplied with the CI vector matrix to generate the intermediate matrix $M$. However, it is necessary to review the memory requirement of the $X$ array in the CASSCF matrix multiplication method before we discuss the memory requirement of $X$ in RASSCF matrix multiplication method.

We remember that in the CASSCF matrix multiplication method reviewed in Section 3.2, to generate the intermediate matrix $M$, a matrix $X$ is generated first (Eq. 3.2.6a). Then a linear algebra routine is used to carry out a matrix multiplication between the matrix $X$ and the CI vector matrix (Eq. 3.2.6b). In the CASSCF method, the dimension of the matrix $X$ is defined by the square of the number of full $\xi$-strings, e.g. \[
\begin{pmatrix}
M \\
N_{\xi}
\end{pmatrix} \begin{pmatrix}
M \\
N_{\xi}
\end{pmatrix}.
\]

However, if we also define the dimension of the $X$ array (Eqs. 3.4.6a and 3.4.8a) that is used for the RASSCF cases by using the square of the full global $\xi$-strings, the memory requirement will be very large. For example, for a relatively large system that has 24 electrons in 24 orbitals with $M(RAS1) = M(RAS2) = M(RAS3) = 8$ and $M(xHole) = M(xElec) = 2$, the number of total global $\xi$-strings is 86,982. If we define the matrix $X$ in the RAS matrix multiplication method in the same way as in CASSCF matrix multiplication method, the memory requirement for storing such a matrix will be $86982^2$ words (~60.5 GBytes), which is incredibly large. This will seriously restrict the applications of the RASSCF method and prevent its application to the calculation of large systems.

Fortunately, from the CASSCF cases, we have discovered the following two factors:

1. The matrix $X$ generated for certain $ij$ index pair is sparse.

2. For one $ij$ index pair, when the orbital indices $k = l$, the diagonal elements in matrix $X$ will be generated. For different $kl$ index pairs ($k=l$), some elements might be generated at the same diagonal position in the matrix. The dimension of
the diagonal elements will not exceed the number of full $\xi$-strings, \[
\begin{pmatrix}
M \\
N_\xi
\end{pmatrix}.
\]

However, when $k \neq l$, no two elements will be generated at the same position in the matrix.

According to factor 2, we know only the diagonal positions of the matrix $X$ will be used frequently for different elements. Thus a summation operation for these positions is necessary. But for the off-diagonal positions, since no two elements will be generated at the same position for all the $kl$ orbital index pairs corresponding to one $ij$ pair, no summation operation is needed. Beside, the matrix $X$ is very sparse. We can thus define the $X$ array for the RASSCF matrix multiplication method as a much smaller dimension to store only those diagonal elements for a certain $wx$ model orbital index pair. If we define $N_{\xi}^{\text{Global}}$ as the total number of global $\xi$-strings, the dimension of the $X$ array in our RAS matrix multiplication method will therefore become $N_{\xi}^{\text{Global}}$. All the off-diagonal elements will then be used directly when they are obtained to generate the intermediate matrix $M$ (Algorithm 3.2). Therefore, the total memory requirement for storing the matrix $X$ now becomes $N_{\xi}^{\text{Global}}$ only.

After all the possible $yz$ index pairs that correspond to this $wx$ pair are looped over, we can obtain an array $X$ that contains all non-zero elements and an intermediate matrix $M$ that is generated from all the non-zero off-diagonal elements dynamically. Then the $X$ array, if it contains non-zero elements, is used to carry out a matrix multiplication operation with the CI vector $C$ and $C^T$ (Eqs. 3.4.6b and 3.4.8b) to generate the rest of the elements in $M$.

### 3.6.4 $\sigma$ Vector Updating

As the intermediate matrix $M$ for a certain $wx$ model index pair is obtained, it is necessary to update the $\sigma$-vector from it (Eqs. 3.4.6c and 3.4.8c). This is achieved by
using the excitation list generated from the \( wx \) model orbital index pair \( \{ B_i \} \) in Algorithm 3.2) to multiply with the intermediate matrix \( M \) (towards the end in Algorithm 3.2).

According to Eqs. 3.4.6c and 3.4.8c, in order to update the \( \sigma \)-vector, the intermediate matrix \( M \) must be transposed. Then the excitation list generated from the model orbital index pair \( w \) and \( x \), containing the elements of \( \{ K^{i_h,i_e}, L^{i_h,i_e}, \text{sgn}^{i_h,i_e} \} \), is looped over. Note that the superscripts, \( (i_h,i_e) \), of the string addresses for both of the initial and final states are identical. This is due to when a certain \( wx \) pair is in the RAS2 subspace the excitation doesn’t involve the RAS1 and RAS3 subspaces. The occupation pattern in the RAS1 and RAS3 subspaces of the global string pairs, \( \{ K^{i_h,i_e}, L^{i_h,i_e} \} \), will always be the same, as indicated in Figure 3.5. This excitation list is generated once per \( wx \) pair before starting the loop over \( yz \) pairs. Based on the value of \( i_h \) and \( i_e \) we can find out two points: one is in which \( \xi \)-string category the obtained global string pairs are, and the other is which \( \gamma \)-string categories are allowed to combine with this string category. According to the first point, we can obtain the local addresses of these strings in the corresponding string category. Then by looping over all the strings in the allowed \( \gamma \)-string categories that are found according to the second point, one can obtain all the local addresses of the determinants corresponding to the global string pair \( \{ K^{i_h,i_e}, L^{i_h,i_e} \} \). Thus the determinants in the \( \sigma \)-vector can be updated.

### 3.6.5 Memory Requirement

We have discussed the memory requirement for the \( X \) array specifically in Section 3.6.3. In this section, we will investigate in general what the memory requirement can be in the RASSCF matrix multiplication method. As indicated in Section 3.4.2, we know there are several arrays added on the basis of the current RAS method to carry out the factorized matrix multiplication method for the 7 model integral cases. Therefore, along with the
basic memory requirement of the current RASSCF method, some extra memory must be added.

We shall start from the discussion of how much memory the current method needs. As indicated in the previous sections, the current method updates the CI vector directly once the global string pairs and the associated symbolic matrix elements are available. In this process, the arrays that store the reduced model string lists, expanded global orbital integrals from one model orbital integral, excitation lists generated from the propagation rules, and the $\sigma$-vector and C vector require their own memory. Similar to the CASSCF cases, the memory requirement for storing the reduced model string lists and the excitation lists is negligible compared to the memory requirement for storing the CI vectors. We now would like to know whether the memory requirement for the integral is also negligible. From Table 3.1 we can see that once any of the four model space orbital indices are not in the RAS2 subspace, the integral must be extended to the full RAS space. Thus the size of the expanded integral list is fully determined by the number of model orbital indices in RAS1 and/or RAS3 subspaces,

$$D_{\text{integral}} = \binom{M(\text{RAS1})}{n_1} \binom{M(\text{RAS3})}{n_3} (0 \leq n_1 + n_3 \leq 4),$$

(3.6.7)

where $n_X$ indicates the number of model orbital indices dropped in model RAS$X$ subspaces ($X \in \{1, 3\}$). Obviously, when $n_1 = n_3 = 0$, which is all four model orbital indices are in RAS2 subspace, there will be only one orbital integral (the first term in Table 3.1). Then we can see the size of the integral list depends on the sizes of RAS1 and RAS3 subspaces. For example, with $M(\text{RAS1}) = M(\text{RAS3}) = 10$ and $n_1 = n_3 = 2$ the maximum length of the integral list will be 2025. At any stage the length of the integral vector will be small when compared to the CI vector. Thus the dominant memory requirement becomes the vectors $\sigma$ and C. According to Eq. 3.5.4, the dimension of the CI vector for the Slater determinant cases is calculated as Eq. 1.5.9,
\[ N_{SD}^{\text{RAS}} = \sum_{i_h=0}^{\text{MxHole}} \sum_{i_e=0}^{\text{MxElec}} \left( M(\text{RAS}1)_{i_h} \right) \left( N_\alpha - M(\text{RAS}1) + i_h - i_e \right) \left( M(\text{RAS}3)_{i_e} \right) \times \sum_{i_h' = 0}^{\text{MxHole}-i_h} \sum_{i_e' = 0}^{\text{MxElec}-i_e} \left( M(\text{RAS}1)_{i_h'} \right) \left( N_\beta - M(\text{RAS}1) + i_h' - i_e' \right) \left( M(\text{RAS}3)_{i_e'} \right) \]  

\[ \frac{M(\text{RAS}1)}{N_\alpha} \left( \frac{M(\text{RAS}2)}{N_\beta} \right) \left( \frac{M(\text{RAS}3)}{M_{\text{total}}} \right) \]  

\[ N_{SD}' = \left( \frac{M_{\text{total}}}{N_\alpha} \right) \left( \frac{M_{\text{total}}}{N_\beta} \right) \]  

However, if the problem at hand is a CASSCF calculation with the same size of the active space, the number of the Slater determinant will be,

where \( M_{\text{total}} = M(\text{RAS}1) + M(\text{RAS}2) + M(\text{RAS}3) \). Comparing Eq. 1.5.9 with Eq. 3.6.8, we can see the dimension of the RASSCF CI vector is far less than the dimension of the CASSCF CI vector when the sizes of the active space of both methods are identical. For example, for a RASSCF problem with \( M(\text{RAS}1)=M(\text{RAS}2)=M(\text{RAS}3)=8 \), \( N_\alpha=N_\beta=12 \), and \( \text{MxHole} = \text{MxElec} = 2 \), according to Eq. 1.5.9, the dimension of the CI vector is 66,187,044 words (~530MB). By contrast, the dimension of the CASSCF CI vector for the same size of the active space will become 7,312,459,672,336 words (~58TB). Further more, if the values of MxHole and MxElec change to 1, the dimension of the RASSCF CI vector then becomes 1,158,948 words (~9MB), which is ~60 times smaller than the dimension of the CI vector when MxHole = MxElec = 2. Nevertheless, the dimension of the RASSCF CI vector is completely negligible compared to the dimension of the CASSCF CI vector with the same size of the active space. In Hartree-Waller functions basis, the dimension of the CI vector is about half the dimension compared to the Slater determinant basis. This is due to the fact that the CI vector for the Hartree-Waller functions basis is a symmetrical matrix (HW singlet basis) or an anti-symmetrical matrix (HW triplet basis). Only the lower triangular matrix of the CI vector (Eq. 3.6.2) is necessary to be stored. This will save more memory than the Slater determinant basis. However, if the working memory is not large enough to accommodate the entire RASSCF CI vector, it is straightforward to further develop the algorithm to use only certain blocks of the CI vector, by taking advantage of the string categories. This
indicates that the combinations of the $\alpha$-string categories with the $\beta$-string categories also define many blocks of the CI vector.

For carrying out the RASSCF matrix multiplication method, as described in Section 3.4, several temporary arrays must be added. Since the 7 cases we have modified all involve at least one $2 \rightarrow 2$ excitation, all possible string category combinations will be required. Therefore, for the 7 cases, all blocks of the CI vector will be read into memory.

In the RASSCF matrix multiplication method, we need to add two restructured CI vectors (one for the $\sigma$-vector and one for the $C$ vector, denoted as $\sigma^T$ and $C^T$), an intermediate matrix $M$ having the same dimension as $\sigma^T$, one vector, $M^T$, that is the transposed form of $M$, and several other arrays, e.g. the $X$ array. The dimensions of some of these added vectors are defined previously. For example, the restructured CI vectors will take either the same dimension as the CI vector (Slater determinant basis), or about twice the dimension of the CI vector (Hartree-Waller functions basis). The number of the full global $\xi$-strings determines the size of the $X$ array. As we indicated above, the size of the CI vector of the RASSCF wavefunction is relatively negligible compared to the size of the corresponding CASSCF CI vector. Thus in the new method the extra memory taken by the newly added temporary arrays will be acceptable. If we define $N_{\text{Conf.}}^{\text{RASSCF}}$ as the dimension of $\sigma^T$ and $N_{\xi}^{\text{Global}}$ as the total number of the global $\xi$-strings, then the total dimension of the newly added arrays for the RASSCF matrix multiplication becomes

$$m \cdot N_{\text{Conf.}}^{\text{RASSCF}} + 19 \cdot N_{\xi}^{\text{Global}}$$

words. In Eq. 3.6.9 $m$ is an integer that take different values for different basis used. For the Slater determinant basis, $m = 5$, while in the Hartree-Waller functions basis, $m = 4$. For large system, the number of global $\xi$-strings can be neglected compared to the dimension of the CI vector ($N_{\text{Conf.}}^{\text{RASSCF}} \gg N_{\xi}^{\text{Global}}$). Therefore for large system Eq. 3.6.9 can be rewritten as,

$$m \cdot N_{\text{Conf.}}^{\text{RASSCF}}$$

(3.6.10)
For example, if the Hartree-Waller singlet basis is chosen, on the basis of the memory requirement of the current method, the extra memory requirement for a system with \(M(RAS1) = M(RAS2) = M(RAS3) = 8\), \(N_\alpha = N_\beta = 12\), and \(\text{MxHole} = \text{MxElec} = 2\) in the new method will be 
\[4 \cdot N_{\text{Conf}}^{\text{RAS}} + 19 \cdot N_{\sigma}^{\text{STr}} = 4 \cdot 66187044 + 19 \cdot 86982 = 266,400,834 \text{ words} \]
(\(~2\text{ GBytes}\)). This amount of memory is widely available in modern computer hardware. The extra memory requirement will always be acceptable because we will soon see from the following performance section that the timing cost of the new method will become very expensive before the available memory runs out (details see in Section 3.8). Therefore the new method is still applicable for large systems. However, this implementation is only the very first version. If the working memory is really not large enough to accommodate the extra required storage, since the newly added \(\sigma^T\), \(C^T\), \(M\), and \(M^T\) are structured in the same way as the \(\sigma\)-vector in the current method, in the future, the matrix multiplication algorithm can be modified by either reading in the required block of the CI vectors from distributed memory, or by restricting contributions to a given CI vector block at a time.

### 3.7 Parallelism

We will now describe the parallel implementation of the direct RASSCF factorized matrix multiplication algorithm (Section 3.4). In this case we will assume a scalable parallel distributed memory computer architecture consisting of nodes is available. Each node has its own local memory and may be a symmetric multi-processor (SMP) machine. Similar to the CASSCF matrix multiplication method, the parallelization of the RASSCF algorithm proposed in this chapter again is implemented following the OpenMP model [125] for the shared memory parallelism and the Linda model [126] for the distributed memory parallelism. For the details about these two models, one can refer back to Section 2.6 in Chapter 2.

As indicated in the last chapter, two key issues must be considered when implementing parallelization: splitting the total work into sub-tasks and load balancing. A flexible load
balancing scheme can help to achieve optimum scaling and is essential in heterogeneous environments, where a cluster may consist of nodes with different architectures.

Generally, load balancing can be achieved when the total number of sub-tasks is larger than the number of available processing elements (PEs, or processors: CPUs or threads). The overhead of splitting the total work into sub-tasks should be kept small compared to total execution time. Thus we can see whether we can achieve a good load balancing depends on how we will split the entire work into sub-tasks.

Recall that in the CASSCF method, the outer most loop is over the orbital indices $i$ and $j$. For a certain $ij$ index pair, all possible $kl$ index pairs that fulfil the condition, $i \geq j, k \geq l$, and $(ij) \geq (kl)$, will be looped over to generate the elements of matrix $X$ (Eq. 3.2.6a), and a matrix multiplication operation is carried out for this $ij$ index pair (Eq. 3.2.6b). Thus on the distributed memory parallelism level, the sub-tasks are executed on parallel nodes correspond to the orbital index pairs $ij$ ($ij$ level). On the shared memory parallelism level, one can either build the sub-tasks that correspond to the orbital index pairs $i$ and $j$ as for the distributed memory parallelism ($ij$ level), or simply parallelize the matrix multiplication for one $ij$ index pair ($kl$ level), or a combination of both. Obviously, one can always choose to use a hybrid scheme to carry out the parallelization, e.g. a combination of both the shared memory and distributed memory parallelization. Thus splitting the sub-tasks at the $ij$ orbital index level is a good choice for the CASSCF method. With this sub-tasks splitting mechanism of the CASSCF matrix multiplication method in mind, we shall start the discussion of the sub-tasks splitting of our RASSCF matrix multiplication method.

As indicated in Section 3.6, similar to the CASSCF matrix multiplication method, in the RASSCF matrix multiplication method, the outer most loops are over the model orbital indices $w$ and $x$ (as shown in Algorithms 3.1 and 3.2). According to the symmetry property (Eq. 3.3.24), all possible $yz$ indices are looped over for a certain $wx$ pair. Thus, similar to the CASSCF method, there are also four ways to split the entire job into sub-tasks: on the level of $w$, $wx$, $wxy$, or $wxyz$. This means, for example, if we define the
parallel loop over \(wx\), then each thread will be distributed to one unique \(wx\) pair and carry out the calculations of all possible \(yz\) pairs corresponding to this \(wx\) pair. Table 3.7 gives the number of sub-tasks when size of the RAS2 subspace varies with \(M_{x\text{Hole}} = M_{xElec} = 2\). The number of tasks is independent of the size of the RAS1 and RAS3 subspaces. When the number of the tasks is very small, e.g. the tasks are defined by index \(w\), load balancing will become impossible. However, on the other hand, when the number of tasks is very large and the sizes of the subtasks are similar, e.g. the tasks are defined by indices \(w,x,y,z\), which is the parallel scheme used by the current method, there will be a cost of communication overhead when using distributed memory parallelism. We may thus consider the parallel loops over \(w,x\) or \(w,x,y\). From Section 3.3.3 we know the factorized method requires the outer most loop is over the \(wx\) model orbital index pair and the inner loop is over the \(yz\) model orbital index pair for carrying out the matrix multiplication. Thus we know in the factorized matrix multiplication method, it is convenient for us to set our parallel loops over the \(wx\) model orbital index pair. In our algorithm sub-tasks do not get pre-allocated to particular workers from the outset. Instead, load balancing is achieved by dynamically allocating tasks to workers. This means each time when a worker finishes its current task, a new one will be automatically loaded. Therefore, in order to achieve a good load balancing, we use an implicit order of computation (the largest \(wx\) pair, which is the most expensive task, is calculated first).

Table 3.7 The number of independent parallel tasks, defined by the orbital indices of the model space for different sizes of the RAS2 subspace with \(M_{x\text{Hole}} = M_{xElec} = 2\).

<table>
<thead>
<tr>
<th>(M(\text{RAS2}))</th>
<th>(M'')</th>
<th>(w)</th>
<th>(w,x)</th>
<th>(w,x,y)</th>
<th>(w,x,y,z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>8</td>
<td>36</td>
<td>204</td>
<td>666</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>10</td>
<td>55</td>
<td>385</td>
<td>1540</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>12</td>
<td>78</td>
<td>650</td>
<td>3081</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>14</td>
<td>105</td>
<td>1015</td>
<td>5565</td>
</tr>
</tbody>
</table>

Inside the parallel loop over the orbital indices \(w,x\), certain blocks of the \(\sigma\)-vector are updated. However, we notice that except for the \(\sigma\)-vector, the transposed \(\sigma\)-vector (or the normalized \(\sigma\)-vector), and several newly added arrays of the matrix multiplication method, e.g. the \(X\) array, all other data inside the loops are static, e.g. the elements of the \(C\) vector will remain unchanged during the entire iteration. Therefore, we can minimise
the communication overhead for the distributed memory parallelisation by passing the static data to all workers only once before the start of each iteration. For the shared memory parallelisation, the static data needs to exist only once.

Now we discuss the implementation of the general strategy discussed above for distributed memory architectures using Linda. Figure 3.8 shows a flowchart of this algorithm. We assume each node is either a single processor machine or a symmetrical multi-processor (SMP) machine. The number of processors on each node is denoted as \( NProcS \). The main process is called the master process (master). The process spawned on the first processing node retrieves the index \((wx)\) (with \((wx)\) = \(M^m(M^m+1)\) \(2\) from tuple space. Then it returns a new index \((wx)'+(wx)−NProcS\) to tuple space, which is retrieved by the next process, etc. If \(NProcS > 1\) then \(NProcS − 1\) shared memory processes are created. Each process will be assigned a unique \((wx)\) index. Every process will check the validity of the \((wx)\) index. If it is valid, then the job of updating the \(\sigma\)-vector takes place, if not, another \((wx)\) index will be retrieved until it is valid. The reason for not implementing the shared memory parallelism on the \(yz\) level as we did in the CASSCF case is because we carry out the matrix multiplication directly when the elements are obtained in order to avoid the huge memory requirement for storing the off diagonal elements of the matrix \(X\). Thus in our RASSCF implementation, both distributed and shared memory parallelism are parallelized on the \(wx\) level. Each processor then operates all the work for all the inner loop over all the valid \(y,z\) indices and updates the \(\sigma\)-vector using the serial algorithm (Section 3.4). This procedure is continued until the index \((wx)=0\) is found. Then the results of nodes are passed through tuple space to be added up to generate the final result of the computation.

As stated in the CASSCF chapter, SMP architectures are supported in three ways: 1. using Linda parallelism only, 2. using shared memory parallelism only, or 3. using a combination of both. In the first case, one Linda worker is created on each node. Since the communication overhead is relatively small in our parallel scheme, the performance
of the parallelization should not be affected significantly unless the network performance is a restriction. However, the static data (the C vector, the transposed C vector or the normalized C vector, the orbital integrals, etc.) must be copied to each Linda worker before the iteration starts.

The second way is to use shared memory parallelism only (where \( NProcS > 1 \)). In this case all the static data needs to be replicated only once per node and are shared by all processors (or cores if the processor is multi-core). All result vectors (one per thread) are summed up before the result is transferred back to the tuple space. However, the price to be paid in this case is that there is no load balancing between threads on a node. The tasks defined by the \( NProcS \) indices are of very similar sizes, because the values of the \((wx)\) indices retrieved by the threads of one node are very close. In practise this means that the efficiency will decrease if \( NProcS \) takes a very large number.

The third way is to combine the distributed memory parallelism and the shared memory parallelism together. In this case, one can set Linda workers on several separated SMPs for distributed memory parallelization and on these SMPs shared memory parallelization is carried out. Using this option the mode of parallelism may be adapted to any available architecture.

The parallel approach described above has great flexibility in choosing the number of processors and, due to the relatively high number of tasks (higher than the number of processors used for \((wx)\) level parallelization), leads to a good load balancing. Furthermore, since the tasks become considerable cheaper as the value of the \((wx)\) index gets smaller (indicates a smaller total number of allowed \(yz\) index pairs thus a smaller vector updating operations), the implicit order of computation is an essential feature in order to ensure efficient use of all processors. The load-balancing scheme implemented also allows for the fact that, in many environments, the CPU-time on different nodes of a parallel machine may be shared among a number of running programs and thus automatically uses the resources as they become available.
Now we should briefly discuss the memory requirement for the shared memory parallelization. As we indicated above, the shared memory for the parallel RASSCF
method is very similar to the \(ij\) level shared memory (Chapter 2) of the parallel CASSCF method. Therefore memory requirement of the shared memory parallelization will be similar too. When our method is parallelized on the model index pair \(wx\) level, each processor will be assigned a unique \(wx\) pair. Then Eqs. 3.4.6 and 3.4.8 will be carried out serially for this \(wx\) pair. Based on these equations, we can see each processor will have its own copy of the vectors \(M_{ij}^{K_h L_i G}\) and \(\sigma_{ij}^{K_h L_i G}\). Thus when the number of shared memory parallel processors increases, the number of the copies of the vectors will increase too. In other words, the memory requirement will increase when the number of shared memory parallel threads increases. For example, based on the discussion in Section 3.6, for a Hartree-Waller singlet computation, on a SMP with \(NProcS = 2\) the memory requirements are for 10 CI vectors, i.e. one shared CI vector \((C)\) for the other integral cases, one shared normalized CI vector for the 7 time consuming integral cases \((C^{HW})\), one result vector for each processor \((\sigma\) vector) for the other integral cases, one normalized result vector for each processor \((\sigma^{HW}\) vector) for the 7 time consuming cases, one intermediate vector for each processor \((M_{ij}^{K_h L_i G})\), and one transposed intermediate vector for each processor \((M_{ij}^{K_h L_i G})^T\). The advantage of this option is that this can reduce the communication between the Linda workers by a factor of \(NProcS\), since all static data and the results have to be passed only once per \(NProcS\) workers. However, again, this is only the very first version of implementation. If the advantage of the blocked structure of the CI vector is adapted in future, very large systems can be still calculated even if the available memory is very limited.

### 3.8 Timing data for the modified cases

The algorithms given in Section 3.4 are developed within the development package of *Gaussian* [42] to improve the performance of the current method. Thus in this section we will investigate the performance of the newly developed method for the 7 selected cases, which are also the most time consuming cases. However, before we look at the performance of the new method compared to the current method, it is very important to
confirm that the new method works, e.g. gives the same result as the current method. Hence this section starts with a sub-section that verifies the new method. The performance investigation of the new method is then followed. All the test calculations are carried out on PC cluster with the hardware property as Intel Nehalem servers (2 Intel Nehalem Quad-core 2.5 GHz processors per node with a peak speed of \( \sim 80 \) GFlops [128]).

3.8.1 New Method Verification

In the previous sections, we have presented a matrix multiplication approach to improve the performance of the current RASSCF method implemented in Gaussian. However, we would like to know whether this method works or not. Thus it is very important to verify whether the new method works properly through a series of test calculations. For this purpose, some relatively small chemical systems have been chosen (Chart 3.1). These systems were also used to verify the new CASSCF method we have developed. However, due to the introduction of the RAS1 and RAS3 subspaces in the RASSCF method, the following verification will be divided into two parts. One is simply to compare the output of both the current RASSCF method (denoted as Old hereafter) and the new method (denoted as New hereafter) by carrying out the same calculation on the same type of computer hardware. The current RASSCF has been proved to successfully approximate the CASSCF wavefunction if the occupation restrictions in RAS1 and RAS3 subspaces are chosen properly. Thus in this section, the second part would be to use a CASSCF, e.g. CAS(12,12), calculation as a reference and to carry out a series of RAS calculations by using the newly developed method with a fixed total number of active orbitals but different combinations of \( M(\text{RAS1})+M(\text{RAS2})+M(\text{RAS3}) \) and varying \( M\text{xHole} \) and \( M\text{xElec} \). This is to show the convergence of the RASSCF calculations to the corresponding CASSCF calculation.
In the following, a single point energy Slater determinant singlet calculation on the three systems shown in Chart 3.1 is carried out to show the new developed method works properly (Table 3.8). The test calculations will be indicated as $\text{RAS}(N, \text{RAS}1 + \text{RAS}2 + \text{RAS}3)[\text{MxHole}, \text{MxElec}]$ to represent a calculation that the total number of electrons and orbitals are $N$ and $\text{RAS}1 + \text{RAS}2 + \text{RAS}3$ respectively where at most $\text{MxHole}$ electrons can be annihilated from the $\text{RAS1}$ subspace and at most $\text{MxElec}$ electrons can be created in the $\text{RAS3}$ subspace. For calculations in which $\text{RAS}1 + \text{RAS}2 + \text{RAS}3$ remains constant, we will use a simpler way to represent a calculation as $\text{RAS}_N(a, b, c, d)$, with $\text{MxHole} = a$, $M(\text{RAS1}) = b$, $\text{MxElec} = c$, and $M(\text{RAS3}) = d$.

As mentioned in previous sections, along with the most general Slater determinant cases, we have also developed algorithms for the basis of spin adapted Hartree-Waller functions. In order to test whether these algorithms work properly, several single point energy calculations on system Naphthalene alone are carried out (Table 3.9). From the data listed in both Tables 3.8 and 3.9 we can see, the outputs of both the current method and the new method for the same calculation are identical. Moreover, we expect the output of Hartree-Waller singlet / triplet calculation should give the same result as the Slater determinant singlet / triplet calculation. From Table 3.9, by comparing the outputs of Slater determinants singlet / triplet calculation with Hartree-Waller singlet / triplet calculations, our expectation is confirmed. All these simply prove that our new method works properly as in the current method.
Table 3.8 Single point energy Slater determinant singlet calculations on systems Naphthalene, Acenaphthalene, and Pyracylene by using both the new method (indicated as New in table) and the current method (indicated as Old in table). The results show us that the new method works properly.

<table>
<thead>
<tr>
<th>Calculation:</th>
<th>No. α-strings</th>
<th>No. β-strings</th>
<th>No. configurations</th>
<th>( E ) (hartree)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS(10,2+6+12)[2,2]</td>
<td>Old 4592</td>
<td>Old 4592</td>
<td>873652</td>
<td>-382.9384189</td>
</tr>
<tr>
<td>RAS(12,4+4+12)[2,2]</td>
<td>Old 4150</td>
<td>Old 4150</td>
<td>238588</td>
<td>-382.5403585</td>
</tr>
<tr>
<td>RAS(12,2+6+4)[1,3]</td>
<td>Old 701</td>
<td>Old 701</td>
<td>125461</td>
<td>-382.5146168</td>
</tr>
<tr>
<td>RAS(14,4+6+10)[2,2]</td>
<td>Old 10336</td>
<td>Old 10336</td>
<td>2089780</td>
<td>-382.9384189</td>
</tr>
</tbody>
</table>

Table 3.9 Single point energy calculations on Naphthalene by using both the new method and the current method with different basis functions. HW stands for Hartree-Waller. SD is for Slater determinant. And configs. means configurations. We can see in this table that the outputs of the SD singlet / triplet calculations are identical to the outputs of the HW singlet / triplet calculations. The result shown in this table again proves the new method works properly.

<table>
<thead>
<tr>
<th>Calculation: RAS(n,2+6+12)[2,2]</th>
<th>No. α-electrons</th>
<th>No. β-electrons</th>
<th>No. α-strings</th>
<th>No. β-strings</th>
<th>No. configs.</th>
<th>( E ) (hartree)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW singlet</td>
<td>Old 5 5</td>
<td>Old 4592</td>
<td>Old 4592</td>
<td>Old 4592</td>
<td>437181</td>
<td>-382.9384189</td>
</tr>
<tr>
<td>HW triplet</td>
<td>Old 5 5</td>
<td>Old 4592</td>
<td>Old 4592</td>
<td>Old 4592</td>
<td>436471</td>
<td>-382.8187673</td>
</tr>
<tr>
<td>SD singlet</td>
<td>Old 5 5</td>
<td>Old 4592</td>
<td>Old 4592</td>
<td>Old 4592</td>
<td>873652</td>
<td>-382.9384189</td>
</tr>
<tr>
<td>SD triplet</td>
<td>Old 6 4</td>
<td>Old 5320</td>
<td>Old 5320</td>
<td>Old 5320</td>
<td>644457</td>
<td>-382.8187673</td>
</tr>
<tr>
<td>Doublet</td>
<td>Old 5 4</td>
<td>Old 4592</td>
<td>Old 2590</td>
<td>Old 2590</td>
<td>632468</td>
<td>-382.6621729</td>
</tr>
</tbody>
</table>

For testing the convergence of the RASSCF calculations to the corresponding CASSCF calculation, a series of RASSCF test calculations for acenaphthylene with different combinations of \( M(RAS1) \), \( M(RAS2) \), \( M(RAS3) \), MxHole, and MxElec are carried out. Since the total number of active electrons and active orbitals remains constant for this series of RAS test calculations, we can use the simpler way to represent these calculations as mentioned above. Also, according to Tables 3.8 and 3.9, we know the new method works properly, the convergence tests will thus be carried out by using the new method only. The performance of the RASSCF method depends critically on the proper selection of the RAS subspaces. From the natural orbital analysis of the CAS(12,12)
reference calculation, we know the occupation number of the 12 active orbitals are: 1.9945, 1.9922, 1.9494, 1.9034, 0.1126, 0.0708, 0.0532, 0.0735, 0.00003, 0.00007. We can see that the occupation of the first, second, and third orbitals is very close to 2, while the occupation of the ninth, eleventh, and twelfth orbitals is very close to 0. Therefore we know the 6 orbitals (the fourth, fifth, sixth, seventh, eighth, and tenth orbitals) show relatively large occupation change. Thus a natural splitting of the active space will be \( M(RAS1) = M(RAS3) = 3 \) and \( M(RAS2) = 6 \). Tables 3.10 give the results of these test calculations. Table 3.10a gives the results of the calculations when the sizes of the RAS1, RAS2, and RAS3 subspaces are fixed while \( MxHole \) and \( MxElec \) vary (in this case, \( M(RAS2) \) is set as 6). Table 3.10b gives the results of the calculations when \( MxHole \) and \( MxElec \) are set as 1 with the sizes of RASX subspace vary. Table 3.10c lists all the results with \( MxHole = MxElec = 2 \) when the sizes of RASX subspace vary. From Tables 3.10 we can find the following factors:

1. If we set \( M(RAS2)=12 \) we will obviously get identical result as the \( CAS(12,12) \) reference calculation.

2. When the values of \( MxHole \) and \( MxElec \) are set fixed, the RASSCF calculation can be more and more accurate when \( M(RAS2) \) increases (thus \( M(RAS1) \) and/or \( M(RAS3) \) decreases).

3. When the values of \( M(RAS1), M(RAS2), \) and \( M(RAS3) \) are fixed, the RASSCF calculation can be more and more accurate when \( MxHole \) and/or \( MxElec \) increases.

Figures 3.9 visualize the above 3 factors. According to the data given above, we can say that we have successfully verified that the new method works. Now we should start our investigation of the performance of the newly developed method.
3. Algorithm for the Direct RASSCF Method (Part I)

Table 3.10a The convergence of a series of RASSCF calculation to a reference CASSCF calculation (HW singlet basis is used). The CASSCF(12,12) single point energy calculation on acenaphthylene is carried out as a reference calculation. Then a series of RASSCF calculations with different combinations of \( M(RAS1)+M(RAS2)+M(RAS3) \), and \((\text{MxHole, MxElec})\) are carried out to show the convergence of the accuracy of these RASSCF calculations. In this table, the number of RAS1, RAS2, and RAS3 subspace orbitals is set fixed while MxHole and MxElec vary. We will use \( \text{RAS}_{12}(a,b,c,d) \) to indicate a calculation with \( \text{MxHole} = a, M(RAS1) = b, \text{MxElec} = c, \) and \( M(RAS3) = d \) in the table (1 hartree \( \approx 27.2114 \) eV).

<table>
<thead>
<tr>
<th>Calculations</th>
<th>( M(RAS2) = )</th>
<th>( E ) (hartree)</th>
<th>( \Delta E_{\text{RAS-CAS}} ) (eV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS(12,12)</td>
<td>-</td>
<td>-458.5173469</td>
<td>-</td>
</tr>
<tr>
<td>RAS_{12}(1,3,1,3)</td>
<td>6</td>
<td>-458.5028286</td>
<td>0.3951</td>
</tr>
<tr>
<td>RAS_{12}(2,3,2,3)</td>
<td>6</td>
<td>-458.5170937</td>
<td>6.89E-3</td>
</tr>
<tr>
<td>RAS_{12}(3,3,3,3)</td>
<td>6</td>
<td>-458.5173020</td>
<td>1.22E-3</td>
</tr>
<tr>
<td>RAS_{12}(4,3,4,3)</td>
<td>6</td>
<td>-458.5173466</td>
<td>8.16E-6</td>
</tr>
<tr>
<td>RAS_{12}(0,0,0,0)</td>
<td>12</td>
<td>-458.5173469</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Figure 3.9a When the number of orbitals in RAS1, RAS2, and RAS3 subspaces is set fixed, the result of the RASSCF calculation can become closer and closer to the reference CASSCF calculation when the number of MxHole/MxElec increases. The 0.0 line is the result given by the reference CASSCF calculation.
Table 3.10b In this table we will set \( M_{\text{XHole}} = M_{\text{XElec}} = 1 \) while the number of RAS2 subspace orbitals varies. We will still use \( \text{RAS}_{12}(a,b,c,d) \) to indicate a calculation as in Table 3.10a. 1 hartree \( \approx 27.2114 \) eV

<table>
<thead>
<tr>
<th>Calculations</th>
<th>( M(\text{RAS2}) )</th>
<th>( E ) (hartree)</th>
<th>( \Delta E_{\text{RAS-CAS}} ) (eV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS(12,12)</td>
<td>-</td>
<td>-458.5173469</td>
<td>-</td>
</tr>
<tr>
<td>RAS(_{12}(1,5,1,5))</td>
<td>2</td>
<td>-458.4496558</td>
<td>1.8420</td>
</tr>
<tr>
<td>RAS(_{12}(1,4,1,4))</td>
<td>4</td>
<td>-458.4809290</td>
<td>0.9910</td>
</tr>
<tr>
<td>RAS(_{12}(1,3,1,3))</td>
<td>6</td>
<td>-458.5028286</td>
<td>0.3951</td>
</tr>
<tr>
<td>RAS(_{12}(1,2,1,2))</td>
<td>8</td>
<td>-458.5148772</td>
<td>0.0672</td>
</tr>
<tr>
<td>RAS(_{12}(1,1,1,1))</td>
<td>10</td>
<td>-458.5164510</td>
<td>0.0244</td>
</tr>
<tr>
<td>RAS(_{12}(0,0,0,0))</td>
<td>12</td>
<td>-458.5173469</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Figure 3.9b When the values of \( M_{\text{XHole}} \) and \( M_{\text{XElec}} \) are both set as 1, we can see the results of RASSCF calculations are getting closer and closer to the result of the reference CAS calculation when the number of RAS2 subspace orbitals increases. The 0.0 line is the result given by the reference CASSCF calculation.

Table 3.10c In this table we will set \( M_{\text{XHole}} = M_{\text{XElec}} = 2 \) while the number of \( M(\text{RAS2}) \) varies.

<table>
<thead>
<tr>
<th>Calculations</th>
<th>( M(\text{RAS2}) = )</th>
<th>( E ) (hartree)</th>
<th>( \Delta E_{\text{RAS-CAS}} ) (eV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS(12,12)</td>
<td>-</td>
<td>-458.5173469</td>
<td>-</td>
</tr>
<tr>
<td>RAS(_{12}(2,5,2,5))</td>
<td>2</td>
<td>-458.5039699</td>
<td>0.3640</td>
</tr>
<tr>
<td>RAS(_{12}(2,4,2,4))</td>
<td>4</td>
<td>-458.5136535</td>
<td>0.1005</td>
</tr>
<tr>
<td>RAS(_{12}(2,3,2,3))</td>
<td>6</td>
<td>-458.5170937</td>
<td>6.89E-3</td>
</tr>
<tr>
<td>RAS(_{12}(2,2,2,2))</td>
<td>8</td>
<td>-458.5173226</td>
<td>6.61E-4</td>
</tr>
<tr>
<td>RAS(_{12}(2,1,2,1))</td>
<td>10</td>
<td>-458.5173469</td>
<td>0.00</td>
</tr>
<tr>
<td>RAS(_{12}(0,0,0,0))</td>
<td>12</td>
<td>-458.5173469</td>
<td>0.00</td>
</tr>
</tbody>
</table>
3.8.2 New Method Performance

In this sub-section, we will investigate the performance of the newly developed method for the selected 7 cases described in Section 3.4. Since our method focuses mainly on improving the efficiency of the CI vector evaluation, the timing data of one Davidson iteration of both the new method and the current method will be crucial for the performance investigation. In order to obtain reliable timing data of one Davidson iteration, we will choose a larger system (coronene, Chart 3.2) than the systems used in the last sub-section. With an increased number of active electrons (24), the process of choosing the sizes of RAS1 and RAS3 subspaces for this system will become more flexible. Thus a single point RAS (24, RAS1+RAS2+RAS3) [MxHole, MxElec] calculation on this system is carried out where the total number of active electrons is fixed as 24 and the numbers of RAS1, RAS2, RAS3, MxHole, and MxElec vary.
In order to obtain a better performance data, we choose to use the Slater determinant singlet basis in our main performance test calculations. This is because the Slater determinant singlet basis has the largest CI vector compared to other bases. In the following, similar to the CASSCF timing data, the timing data of the 4\(^{th}\) Davidson iteration of these calculations (both the New method and the Old method) are recorded, where the timing data of the current method is used as our benchmark. In the following tables, the timing data of only the changed part (the 7 cases) and the overall of the 4\(^{th}\) Davidson iteration are given. The percentage of the timing of the changed part in the overall timing is also listed. Table 3.11 shows the single excitation timing data of the total number of orbital remains constant while the size of RAS2 increases (thus the size of RAS1 and RAS3 subspaces decreases). Table 3.12 gives the double excitation timing data of these cases. Table 3.13 lists the single excitation timing data where the sizes of the RAS1 and RAS2 subspaces remain constant while the number of orbitals in the RAS3 subspace is changed. In Table 3.14 one can find the double excitation timing data of these cases.

Chart 3.2 Coronene structure.
### 3. Algorithm for the Direct RASSCF Method (Part I)

Table 3.11 Timing data of the 7 cases only for the cases of the total number of orbitals remains the same while the number of RAS2 subspace varies (single excitation only: MxHole=MxElec=1).

<table>
<thead>
<tr>
<th>Calculation</th>
<th>RAS(24,10+4+10)</th>
<th>RAS(24,9+6+9)</th>
<th>RAS(24,8+8+8)</th>
<th>RAS(24,7+10+7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>11,396</td>
<td>112,450</td>
<td>1,158,948</td>
<td>12,090,456</td>
</tr>
<tr>
<td>No. of full strings</td>
<td>686</td>
<td>1,910</td>
<td>5,446</td>
<td>15,540</td>
</tr>
</tbody>
</table>

| Old | 1 iter. overall | 0.81 | 18.18 | 468.40 | 11943.46 |
|     | Changed part    | 0.28 | 9.6   | 303.89 | 8733.53  |
|     | %               | 35%  | 52.81%| 64.88% | 73.12%   |

| New | 1 iter. overall | 0.54 | 8.88 | 165.15 | 3139.59 |
|     | Changed part    | 0.03 | 0.49 | 5.85   | 138.83  |
|     | %               | 5.56%| 5.52%| 5.20%  | 4.42%   |

| Changed part speedup | 9.33 | 19.59 | 35.42 | 62.91 |
| Overall Speedup      | 1.48 | 2.05  | 2.84  | 3.80  |

Table 3.12 Timing data of the 7 cases only for the cases of the total number of orbitals remains the same while the number of RAS2 subspace varies (double excitation only: MxHole=MxElec=2).

<table>
<thead>
<tr>
<th>Calculation</th>
<th>RAS(24,10+4+10)</th>
<th>RAS(24,9+6+9)</th>
<th>RAS(24,8+8+8)</th>
<th>RAS(24,7+10+7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>959,926</td>
<td>7,925,332</td>
<td>66,187,044</td>
<td>540,193,248</td>
</tr>
<tr>
<td>No. of strings</td>
<td>16,526</td>
<td>37,982</td>
<td>86,982</td>
<td>193,452</td>
</tr>
</tbody>
</table>

| Old | 1 iter. overall | 30.18 | 439.94 | 7243.08 | 139236.76 |
|     | Changed part    | 5.84  | 145.51 | 3289.52 | 80384.47  |
|     | %               | 19.35%| 33.07%| 45.42% | 58.06%   |

| New | 1 iter. overall | 14.99 | 189.37 | 2134.8 | 31337.55 |
|     | Changed part    | 2.1   | 46.36 | 577.57 | 9957.56 |
|     | %               | 14.01%| 24.84%| 27.05% | 31.78%   |

| Changed part speedup | 2.78 | 3.14 | 5.70 | 8.12 |
| Overall Speedup      | 2.01 | 2.32 | 3.39 | 4.44 |

Table 3.13 Timing data of the 7 cases only for the cases of the number of RAS2 orbitals remains the same while the number of RAS3 subspace varies (single excitation only: MxHole=MxElec=1).

<table>
<thead>
<tr>
<th>Calculation</th>
<th>RAS(24,8+8+8)</th>
<th>RAS(24,8+8+12)</th>
<th>RAS(24,8+8+16)</th>
<th>RAS(24,8+8+20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>1,158,948</td>
<td>1,704,612</td>
<td>2,250,276</td>
<td>2,795,940</td>
</tr>
<tr>
<td>No. of full strings</td>
<td>5,446</td>
<td>7,910</td>
<td>10,374</td>
<td>12,838</td>
</tr>
</tbody>
</table>

| Old | 1 iter. overall | 468.40 | 501.17 | 523.39 | 550.90 |
|     | Changed part    | 303.89 | 322.26 | 330.26 | 343.08 |
|     | %               | 64.88% | 64.30% | 63.10% | 62.28% |

| New | 1 iter. overall | 165.15 | 182.43 | 195.67 | 201.73 |
|     | Changed part    | 8.58   | 16.99 | 21.85 | 20.48 |
|     | %               | 5.20%  | 9.31% | 11.17% | 10.15% |

| Changed part speedup | 35.42 | 18.97 | 15.11 | 16.75 |
| Overall Speedup      | 2.84  | 2.75  | 2.67  | 2.73  |
Table 3.14 Timing data of the 7 cases only for the cases of the number of RAS2 orbitals remains the same while the number of RAS3 subspace varies (double excitation only: MxHole=MxElec=2).

<table>
<thead>
<tr>
<th>Calculation</th>
<th>No. of Slater Det.</th>
<th>No. of strings</th>
<th>Timing (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAS(24,8+8+8)</td>
<td>RAS(24,8+8+12)</td>
<td>RAS(24,8+8+16)</td>
</tr>
<tr>
<td>1 iter. overall</td>
<td>66,187,044</td>
<td>145,209,540</td>
<td>254,977,380</td>
</tr>
<tr>
<td>Changed part</td>
<td>86,982</td>
<td>188,286</td>
<td>328,566</td>
</tr>
<tr>
<td>%</td>
<td>7243.08</td>
<td>17350.55</td>
<td>46401.73</td>
</tr>
<tr>
<td>New</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 iter. overall</td>
<td>2134.8</td>
<td>4740.75</td>
<td>10695.54</td>
</tr>
<tr>
<td>Changed part</td>
<td>577.57</td>
<td>1511.92</td>
<td>3150.63</td>
</tr>
<tr>
<td>%</td>
<td>27.05%</td>
<td>31.89%</td>
<td>29.46%</td>
</tr>
<tr>
<td>Changed part speedup</td>
<td>5.70</td>
<td>4.77</td>
<td>5.41</td>
</tr>
<tr>
<td>Overall Speedup</td>
<td>3.39</td>
<td>3.66</td>
<td>4.34</td>
</tr>
</tbody>
</table>

From the data listed in Tables 3.11-3.14, we can see after we have modified the most time consuming 7 cases the new method can always perform faster. Moreover, as the size of RAS2 subspace increases (Tables 3.11 and 3.12), the new method can perform better and better. However, this timing data come from the new method that only modifies 7 integral cases among the total 22 cases. In the next chapter, the remaining integral cases will be investigated and modified in order to get a better overall performance.

Another point that should be mentioned here is the competition between the memory requirement and the computation timing cost. According to the memory requirement discussion given in Section 3.6.5, the memory requirement is mainly determined by the size of the CI vector. Based on the data listed above, we can see the largest CI vector is the calculation of RAS(24,7+10+7)[2,2], which is 540,193,248 words (~4.3GBytes) long. In the new method, to carry out such a calculation, as we stated in Section 3.6.5, the extra memory requirement will be \( \sim 7 \cdot N_\text{Conf}^{RAS} \) (5 added vectors plus the 2 original CI vectors) for the Slater determinant basis, which is around 30GBytes. For Hartree-Waller basis, this becomes \( \sim 5 \cdot N_\text{CSFs}^{RAS} \) (4 added arrays in the new method and the rest from the current method), which is about 22GBytes. However, as indicated in Table 3.12, although the new method for carrying out such a large job is \( \sim 4 \) times faster than the current method, the timing cost for a single Davidson iteration is still too expensive (31337.55 seconds, about 10 hours) The memory requirement, however, is achievable on modern commercial
PC clusters. Nevertheless, as stated above, this version of method is only the very first version and, in the future, when the blocked structure algorithm for the CI vector is adapted, the memory requirement will be dramatically reduced.

So far we have only given the timing data of Slater determinant singlet calculations by using our new method. We would like to know what the performance of the new method would be when the calculations are carried out by using the Hartree-Waller functions (singlet and triplet) basis or doublet / triplet calculations using the Slater determinants basis. From Tables 3.11-3.14 we can see the timing data of the calculation of RAS(24,7+10+7)[1,1] is good for demonstrating not only serially the new method performs better, but also the parallel performance as will be given later. This is because, while the execution time is not too long, it is enough to show the difference between the two methods. Moreover, this calculation is also a good example to show the parallel performance of both the new and the current methods. As we know the new method can change the time consuming part successfully to non-time consuming cases, in the following, only the overall timing data of the 4th Davidson iteration will be given to illustrate the performance of the RAS matrix multiplication method when using the different basis (Table 3.15). From the data given in Table 3.15 we know no matter what basis is used, the RAS factorized matrix multiplication method can always perform faster than the current method. This again proves that the matrix multiplication method has successfully changed the time consuming cases into non-time consuming cases and thus performs faster.
Table 3.15 Serial timing data for the 4th Davidson iteration when using different basis on the same calculation of coronene (RAS(24,7+10+7)[1,1]). We can see from this table that no matter what basis is used, the RASSCF matrix multiplication method always performs faster than the current method.

| Calculation: RAS(n,7+10+7)[1,1] | No. α-electrons | No. β-electrons | No. α-strings | No. β-strings | No. configs. | 1 iteration timing (s) | New method speedup |
|-------------------------------|------------------|------------------|.executeUpdate();} | | | | | |
| HW singlet | Old | New | 12 | 12 | 15540 | 15540 | 6045354 | 6395.14 | 3.87 |
| HW triplet | Old | New | 12 | 12 | 15540 | 15540 | 6045102 | 6229.87 | 3.80 |
| SD singlet | Old | New | 12 | 12 | 15540 | 15540 | 12090456 | 11943.46 | 3.80 |
| SD triplet | Old | New | 12 | 11 | 13104 | 13104 | 9276876 | 8899.73 | 3.69 |
| Doublet | Old | New | 12 | 11 | 15540 | 13104 | 10340568 | 10062.93 | 3.67 |

Now we will investigate the performance of parallelization of the new method. We again use the Slater determinant calculation of coronene RAS(24,7+10+7)[1,1] to demonstrate. As we stated in Section 3.7 that the parallelism can be carried out in three ways, using shared memory parallelization (OpenMP) only, using distributed memory parallelization (Linda) only, or a combination of both. In the following we will investigate the performance of parallelism for these three types separately. The timing data of using shared memory parallelization only are recorded in Table 3.16 with \( N_{ProcS} = \{1,2,4,6,8\} \). Table 3.17 gives the timing data of using Linda only (up to 5 Linda workers as representative). The timing data of using a hybrid parallelization that uses 2 Linda workers while the number of shared memory processors on each node varies can be found in Table 3.18. From the speedup rows of Table 3.16-3.18 we can see the parallel performance of both the current method and the new method are almost scaled linearly with the increasing number of processors (Figures 3.10-3.12). The parallelization of the new method performs as well as we expected. This proves to us that the flexible parallel scheme described in Section 3.7 works well.
3. Algorithm for the Direct RASSCF Method (Part I)

Table 3.16 Shared memory timing data for the 4th Davidson iteration of both the current method and the new method with only the first 7 time consuming cases are modified (OpenMP only).

<table>
<thead>
<tr>
<th>System</th>
<th>Coronene: RAS(24,7+10+7)[1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>12,090,456</td>
</tr>
<tr>
<td>No. of Global string</td>
<td>15,540</td>
</tr>
<tr>
<td>NProcShared =</td>
<td>1     2     4     6     8</td>
</tr>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>CPU time</td>
<td>3164.92 3206.91 3239.93 3251.25 3332.62</td>
</tr>
<tr>
<td>Wall time</td>
<td>3164.92 1603 810 542 413</td>
</tr>
<tr>
<td>Speedup</td>
<td>- 1.97 3.91 5.84 7.66</td>
</tr>
<tr>
<td>Old</td>
<td></td>
</tr>
<tr>
<td>CPU time</td>
<td>11943.46 12083.80 12155.64 12426.75 12549.53</td>
</tr>
<tr>
<td>Wall time</td>
<td>11943.46 6043 3040 2073 1569</td>
</tr>
<tr>
<td>Speedup</td>
<td>- 1.98 3.93 5.76 7.61</td>
</tr>
</tbody>
</table>

Figure 3.10 The shared memory parallelism only performance of both the old and the new methods. From this figure we can see the speedup of the parallelism is almost linearly scaled.

Table 3.17 Timing data of the 4th Davidson iteration of both the current method and the new method when using (Linda only).

<table>
<thead>
<tr>
<th>System</th>
<th>Coronene: RAS(24,7+10+7)[1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>12,090,456</td>
</tr>
<tr>
<td>No. of Global string</td>
<td>15,540</td>
</tr>
<tr>
<td>NProcLinda =</td>
<td>1     2     3     4     5</td>
</tr>
<tr>
<td>NProcShared =</td>
<td>1     1     1     1     1</td>
</tr>
<tr>
<td>Old</td>
<td></td>
</tr>
<tr>
<td>CPU time (s)</td>
<td>11943.46 12041.50 12004.59 11911.28 11922.75</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>11943.46 6021 4002 2978 2385</td>
</tr>
<tr>
<td>Speedup</td>
<td>- 1.98 2.98 4.01 5.01</td>
</tr>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>CPU time (s)</td>
<td>3164.92 3144.60 3154.50 3140.92 3156.10</td>
</tr>
<tr>
<td>Wall time (s)</td>
<td>3164.92 1573 1052 786 631</td>
</tr>
<tr>
<td>Speedup</td>
<td>- 2.01 3.01 4.03 5.02</td>
</tr>
</tbody>
</table>

217
Figure 3.11 The distributed memory parallelism only performance of both the old and the new methods. From this figure we can also see the speedup of the parallelism is linearly scaled.

Table 3.18 **Hybrid parallel** timing data when using 2 Linda workers and the number of shared memory threads varies of both methods. From the speedup rows of both methods, we can see both methods scale perfect linearly with the increasing number of processors.

<table>
<thead>
<tr>
<th>System</th>
<th>Coronene: RAS(24,7+10+7)[1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>12,090,456</td>
</tr>
<tr>
<td>No. of Global string</td>
<td>15,540</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NProcShared =</th>
<th>Serial</th>
<th>NProcLinda = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>New</td>
<td>CPU time</td>
<td>3164.92</td>
</tr>
<tr>
<td></td>
<td>Wall time</td>
<td>3164.92</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
<td><strong>3.99</strong></td>
</tr>
<tr>
<td>Old</td>
<td>CPU time</td>
<td>11943.46</td>
</tr>
<tr>
<td></td>
<td>Wall time</td>
<td>11943.46</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
<td><strong>3.96</strong></td>
</tr>
</tbody>
</table>
3.9 Summary

In this chapter, by adopting the highly efficient non-zero symbolic matrix elements generation method used in the current method and converting the factorized matrix multiplication method used for the CASSCF method, we have developed an efficient RASSCF method for improving the performance of the targeted 7 cases which are also the most time consuming cases in the current method. The main feature in the new method is the capability to use the data in the RASSCF CI vector in a sequential order, which improves the performance of the current method to a significant level (illustrated in Tables 3.11-3.15). Another feature of this new RAS method is the introduction of the matrix multiplication in the nested loops over model orbital indices $w$ and $x$ (integral driven) that allows us to carry out the parallelization in an implicit order. Thus the sub-
tasks defined by the $wx$ pairs will not be pre-distributed to the processors. A good load balancing can be achieved and the communication overhead is minimal. However, since the factorized method requires some extra memory to store the generated matrices, when the available memory is limited, the application of the new RAS method will be restricted. In the future, when the blocked structure algorithm of the CI vector is adapted, the memory requirement will be dramatically reduced.

The algorithm is implemented in the current development version of the Gaussian program package [42]. An option of parallel execution is implemented in this algorithm. Running in parallel may be done using distributed memory (Linda) or shared memory, or a combination of these. As well as the most general basis of Slater determinants we also implement the straightforward simplifications for singlets and triplets using Hartree-Waller functions.
Chapter 4

An Attempted Implementation for Modifying the Rest Integral Cases of the Direct RASSCF Method
4.1 Introduction

As described in the last chapter, due to the introduction of the occupation restrictions in the RAS1 and RAS3 subspaces, in total 22 different orbital integral cases arise (Table 3.1). Among these cases, is the current RASSCF method, 7 of them are time consuming and are suitable to be modified by using the factorized matrix multiplication method (Section 1.7.2.1). In the last chapter we applied this matrix multiplication scheme to these cases resulting in better performance of the new method over the current one. Thereby, leaving the 15 unchanged cases time consuming in the newly developed RASSCF method. Although the modification of the 7 cases described in the last chapter has already made the new method perform faster than the current method, we would like to seek a way to modify these remaining cases to achieve an overall outstanding performance.

However, as discussed in Section 3.3.3, when implementing Eqs. 3.3.19 to carry out the CI vector updating, the number of matrix multiplication operation will not be 1 if at least one of the model orbital indices \( w \) and \( x \) is not in RAS2 subspace (Eq. 3.3.19c), which is the 15 remaining integral types. This is because once one model orbital index is dropped in either model RAS1 subspace or model RAS3 subspace, the expansion of the model orbital index will lead to a set of full RAS space orbital indices. Each of these indices will correspond to its own intermediate matrix \( M \) (Eq. 3.3.19c). In other words, the number of the matrix multiplication operation (Eq. 3.3.19b) will not be 1 any more. And this many number of matrix multiplication operation might become a bottleneck to the performance of the factorized matrix multiplication method. Hence, if we apply the factorized matrix multiplication scheme to these 15 integral cases, at certain level, the overall performance of the modified method would be slower than the current method. Therefore, the factorized matrix multiplication method that was suitable for the 7 targeted integral cases (first 7 cases in Table 3.1) will not be the solution to modify the remaining 15 integral cases. In this chapter we will seek a possible way to modify these remaining cases. However, as indicated in Section 1.7.2, there are only two ways to carry out the CI vector updating, factorized or unfactorized method. As the factorized matrix multiplication
method is not suitable for the 15 cases, we assume the unfactorized method would be the solution to these cases. We noticed that the method developed by Saunders and Van Lenthe [95] (abbreviated as SV method hereafter) deals with different integral cases in different ways, e.g. some integral cases are carried out via factorized matrix multiplication method and others use unfactorized method. Therefore it is necessary to review this method first.

In the following, we assume one has already read through Chapter 3 and had the background knowledge, e.g. the model space concept and the notation etc., of the RASSCF wavefunction in mind, and we will use these concepts directly in this chapter. This chapter is organized in the following way. Section 4.2 reviews the SV method in details. Based on this review, we know the assumption above is fully applicable. From Chapter 3 we know that the current RASSCF implementation for the remaining 15 cases is already in the unfactorized method. However, according to the review there should still be room to improve upon the performance of these cases. Based on the review, in Section 4.3 an attempt at optimizing these 15 integral cases of the current RASSCF method is undertaken. The performance of this optimized method is demonstrated by using the same calculations for generating the timing data in Tables 3.11-3.14 and 3.16-3.18 of coronene. The result shows that this optimized method can further improve the performance of the RASSCF method developed in the last chapter. Thus an overall improvement of performance is achieved.

4.2 Review of Saunders and Van Lenthe’s Method

As described in Chapter 3, the SV method [95] doesn’t contain the RAS1 subspace. Moreover, at most 2 electrons can be excited into the RAS3 subspace (the so called “external space” in the method). Therefore, the SV method can be treated as a special case of the general RASSCF method with $M(RAS1) = 0$ (thus $MxHole = 0$) and $MxElec = 2$. In this case, the number of model orbital integral cases can be reduced from 22 (Table 3.1) to 6 (Table 4.1). Since the SV method is a special case of the RASSCF

---

223
PhD Thesis: Algorithm Development for Direct MCSCF Method

method, all the orbital labels defined for the RASSCF method in Chapter 3 can be used here too except the RAS1 subspace labels should be excluded,

- RAS2 (internal space): $i,j,k,l$
- RAS3 (external space): $a,b,c,d$
- full model RAS space: $w,x,y,z$
- full RAS space (global space): $i,j,k,l$
- model RAS3 space: $\delta,\epsilon$.

Table 4.1 The model integral types in the SV method that excludes the RAS1 subspace and their expanding to the full orbital space.

| The possible values for the full model RAS space orbital indices in SV method | Expanding to full space integral: $(wx|yz)$ expanding to $(ij|kl)$ |
|---|---|
| $w$, $x$, $y$, $z$ | $(ij|kl)$ expanding to $(ij|kl)$ |
| $i$, $j$, $k$, $l$ | $(ij|k\delta)$ expanding to $(ij|ka),\ldots (ij|ka'')$ |
| $i$, $j$, $k$, $\delta$ | $(ij|\delta\epsilon)$ expanding to $(ia|jb),\ldots (ia''|jb'')$ |
| $i$, $\delta$, $j$, $\epsilon$ | $(i\delta|je)$ expanding to $(ia|bc),\ldots (ia''|b''c'')$ |
| $i$, $\delta$, $\delta$, $\epsilon$ | $(i\delta|\delta\epsilon)$ expanding to $(ab|cd),\ldots (a''|b''c'd'')$ |

Comparing the model orbital integral cases listed in Table 3.1 and Table 4.1, we can see the first 3 model integral cases in Table 4.1 belong to the 7 time consuming cases that we have already studied in the last Chapter. So we won’t consider these integral cases in this chapter. Instead, we will review how the SV method deals with the rest 3 model integral cases.

As mentioned above, we can treat the SV method as a special case of the general RASSCF method. When $M_{x\text{Hole}} = 0$ (thus $i_{\text{c}} = 0$) and $M_{xElec} = 2$ (thus $i_{\text{c}} = \{0,1,2\}$), there will be 6 types of configurations: $K_{\alpha}^{0,0} K_{\beta}^{0,0}$, $K_{\alpha}^{0,1} K_{\beta}^{0,0}$, $K_{\alpha}^{0,0} K_{\beta}^{1,0}$, $K_{\alpha}^{0,1} K_{\beta}^{0,1}$, $K_{\alpha}^{0,2} K_{\beta}^{0,0}$, and $K_{\alpha}^{0,0} K_{\beta}^{0,2}$. In the SV method, according to the number of particles in the RAS3 subspace, they divided the full configuration space into 4 blocks (Table 4.2). For different model orbital integral cases, the types of interactions between configuration
blocks will be different. For example, for the \((i\delta|j\varepsilon)\) model orbital integral, the excitation type is \((2 \rightarrow 3, 2 \rightarrow 3)\). Therefore the initial state (initial configuration: \(|L\rangle = |L^{0.0}_{\alpha} L^{0.0}_{\beta}\rangle\)) should have no particle in the external space. After excitation, two particles are excited from the internal space to the external space to form the final state (final configurations: \(|K\rangle = |K^{0.1}_{\alpha} K^{0.1}_{\beta}\rangle\), \(|K\rangle = |K^{0.2}_{\alpha} K^{0.0}_{\beta}\rangle\), or \(|K\rangle = |K^{0.0}_{\alpha} K^{0.2}_{\beta}\rangle\)). Thus the interactions between \(|K\rangle\) and \(|L\rangle\) will either be singlet-vacuum interaction, e.g. \(\langle K^{0.1}_{\alpha} K^{0.1}_{\beta} | e_{i\delta j\varepsilon} | L^{0.0}_{\alpha} L^{0.0}_{\beta} \rangle\) (where \(e_{i\delta j\varepsilon} = \sum_{\xi\gamma} a^{\dagger}_{\xi} a^{\dagger}_{\gamma} a_{\gamma} a_{\xi}\)), or triplet-vacuum interaction, e.g. \(\langle K^{0.2}_{\alpha} K^{0.0}_{\beta} | e_{i\delta j\varepsilon} | L^{0.0}_{\alpha} L^{0.0}_{\beta} \rangle\).

The interaction types that corresponding to certain model orbital integral class (only the last 3 integral cases in Table 4.1) are listed in Table 4.3.

<table>
<thead>
<tr>
<th>Configuration blocks</th>
<th>Occupation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vacuum: (</td>
<td>K^{0.0}<em>{\alpha} K^{0.0}</em>{\beta}\rangle)</td>
</tr>
<tr>
<td>Doublet: (</td>
<td>K^{0.1}<em>{\alpha} K^{0.0}</em>{\beta}\rangle) or (</td>
</tr>
<tr>
<td>Singlet: (</td>
<td>K^{0.2}<em>{\alpha} K^{0.0}</em>{\beta}\rangle)</td>
</tr>
<tr>
<td>Triplet: (</td>
<td>K^{0.2}<em>{\alpha} K^{0.0}</em>{\beta}\rangle) or (</td>
</tr>
</tbody>
</table>

Table 4.3 The different types of interaction between configurations for the last 3 model orbital integral cases in Table 4.1

<table>
<thead>
<tr>
<th>Model orbital integral types</th>
<th>Configuration interaction types</th>
</tr>
</thead>
<tbody>
<tr>
<td>((i\delta</td>
<td>j\varepsilon))</td>
</tr>
<tr>
<td>((i\delta'</td>
<td>j\varepsilon))</td>
</tr>
<tr>
<td>((i\delta'</td>
<td>j\varepsilon))</td>
</tr>
<tr>
<td>((i\delta'</td>
<td>j\varepsilon))</td>
</tr>
<tr>
<td>((\delta'</td>
<td>\delta\varepsilon))</td>
</tr>
<tr>
<td>((\delta'</td>
<td>\delta\varepsilon))</td>
</tr>
</tbody>
</table>

4.2.1 \((i\delta|j\varepsilon)\) Model Orbital Integral Cases

We will mainly use the singlet-vacuum configuration interaction type of the \((i\delta|j\varepsilon)\) model integral class, which corresponds to the 17th integral case in Table 3.1, in the SV method as example to illustrate how they managed to implement this integral type. For
the singlet-vacuum interaction of this type of integral, one \( \alpha \)-electron and one \( \beta \)-electron are excited from the orbitals \( i \) and \( j \) (assuming \( i \neq j \)) of the initial configuration \( |L^{0,0}_{i,j}\rangle = |L^{0,0}_{\alpha,\beta}\rangle \) (superscript of \( L \) means the electrons in these orbitals will be annihilated) to the orbitals \( a \) and \( b \) of the final configuration, e.g. \( |K^{ab}_{\alpha,\beta}\rangle = |K^{01,01}_{\alpha,\beta}\rangle \). In other words, orbitals \( i \) and \( j \) each carry one more electron in \( L \) than \( K \). The occupation and spin patterns of the other orbitals in the internal space should be the same between \( K \) and \( L \). In the SV method, the \( \sigma \)-vector is updated for the singlet-vacuum interaction case as,

\[
\sigma_{K_{ab}} = P^i_j C_{L_{i,j}} \left( B_{K_{i,j}}^a b \right)^T, \tag{4.4.1}
\]

where the \( P^i_j \) is an “column vector” built for the full RAS space orbital integral corresponding to certain \( ij \) index pair,

\[
P^i_j = \left( \begin{array}{c} (b_i |a_j) + (a_i |b_j) \\ (b'_i |a_j') + (a_i |b'_j) \\ \vdots \\ (b''_i |a''_j) + (a''_i |b''_j) \end{array} \right)^T, \tag{4.4.2}
\]

where \( b \leq a \) and \( i \leq j \). The dimension of this vector is given as \( \left( \frac{M(\text{RAS}3)}{2} \right) \) when \( b < a \) and \( M(\text{RAS}3) \) when \( b = a \). Since there are no electrons in the external space for the configurations, \( |L^{i,j}_n\rangle \), according to the occupation and spin patterns in the orbitals except \( i \) and \( j \), one block vector can be generated from the vacuum configuration block,

\[
C_{L^i_j} = \left( |L^i_1\rangle \right. \left. |L^i_2\rangle \cdots |L^j_n\rangle \right), \tag{4.4.3}
\]

where \( |L^i_1\rangle \) indicates the index of the configuration in the vacuum configuration block of the CI vector \( C \). The subscript \( n \) denotes the occupation and spin pattern of internal orbitals other than orbitals \( i \) and \( j \) (the superscript). The dimension of this block vector is defined as \( \left( \frac{M(\text{RAS}2)}{N_a - 1} \right) \left( \frac{M(\text{RAS}2)}{N_\beta - 1} \right) \), where orbitals \( i \) and \( j \) are deducted from the internal space. The “-1” in the dimension expression comes from: since the final state is a singlet configuration, the orbitals \( i \) and \( j \) should be \( \alpha \)-spin orbital and \( \beta \)-spin orbital, respectively. Thus only one orbital is deducted from the internal space for the \( \alpha \) - or \( \beta \)-electrons. The elements in the matrix \( B_{K_{i,j}}^a b \) are defined as,
The dimension of this result block is where 

Updated, based on this, we can see Eq. 4.4.1 will result in a set of elements in the matrix (Eq. 4.4.4) will be the square of the dimension of the full RAS space symbolic matrix elements, e. Therefore all creation and annihilation operators of the second quantization (Appendix A). Also, as mentioned in Chapter 3, in the SV method, the model symbolic matrix elements remain constant when expanding the model orbital indices to the full RAS space. Therefore all the full RAS space symbolic matrix elements, e.g. 

, , will share the same value, e.g. . The dimension of this matrix (Eq. 4.4.4) will be the square of the dimension of the vector . 

Based on this, we can see Eq. 4.4.1 will result in a set of elements in the -vector being updated, 

where indicates the element index in the singlet CSFs block of the CI vector . 

The dimension of this result block is when \( b < a \) or when \( b = a \). Comparing the dimensions of Eq.
4.4.3 and Eq. 4.4.5, we can see a large number of elements in the $\sigma$-vector are updated by using a relatively small number of elements in the $C$ vector.

However, as we can see from Eq. 4.4.4 that the symbolic matrix elements, $B_{K_m^L i_j^L}$, are not factorized, because the $e_{i\delta_j \epsilon} = \sum_{j\xi} a_{i\xi}^\dagger a_{j\xi} a_{j\epsilon} a_{i\delta}$ operator is not factorized. Therefore, the updating of the CI vector for this $(i\delta|j\epsilon)$ model orbital integral type is through the unfactorized method (Section 1.7.2.2). That is once the non-zero $B_{K_m^L i_j^L}$ is obtained, it is used directly to update the $\sigma$-vector via Eq. 4.4.1.

### 4.2.2 $(\delta'\epsilon'|\delta\epsilon)$ Model Orbital Integral Cases

For the $(i\delta'|\delta\epsilon)$ model orbital integral, in the SV method, the implementation is similar to the $(i\delta|j\epsilon)$ model orbital integral cases described above, which is also an unfactorized method. Therefore, we should move to the $(\delta'\epsilon'|\delta\epsilon)$ model orbital integral cases and see how the SV method carries out the CI vector updating for this type of integral. The $(\delta'\epsilon'|\delta\epsilon)$ model orbital integral indicates that there are always two electrons in the RAS3 space and thus the configuration interaction type for this integral, as given in Table 4.3, can be only singlet-singlet or triplet-triplet. The occupation patterns of orbitals of the initial configurations, $|L\rangle$, and the final configurations, $|K\rangle$, only differ in the RAS3 subspace (external space), e.g. two electrons occupying orbitals $b$ and $d$ in $|L\rangle$ and orbitals $a$ and $c$ in $|K\rangle$. Again, the model symbolic matrix elements, e.g. $B_{\delta'\epsilon'|\delta\epsilon}^{KL} = \langle K^{\delta\epsilon'}|e_{\delta'\epsilon'|\delta\epsilon}\rangle |L^{\delta\epsilon}\rangle$, can represent the symbolic matrix elements of the full space, e.g. $B_{\delta'\epsilon'|\delta\epsilon}^{KL} = \langle K^{\delta\epsilon'}|e_{\delta'\epsilon'|\delta\epsilon}\rangle |L^{\delta\epsilon}\rangle$. Since for non-zero model symbolic matrix elements the occupation and spin patterns in the internal space are the same between $|K\rangle$ and $|L\rangle$, for certain occupation and spin patterns in the internal space, we can write the model symbolic matrix element as $B_{\delta'\epsilon'|\delta\epsilon}^{KL} = \langle K^{\delta\epsilon'}|e_{\delta'\epsilon'|\delta\epsilon}\rangle |L^{\delta\epsilon}\rangle$. Similar to the $(i\delta|j\epsilon)$ case,
different internal occupation and spin patterns between the \(|K\rangle\) and \(|L\rangle\) configurations will result zero, e.g. \(\langle K_1^{\delta e} | e^{\delta e} | L_2^{\delta e} \rangle = 0\). And most importantly, this symbolic matrix element is also not factorized. Now we shall investigate how the SV method updates the CI vector for this type of model orbital integral in a bit more detail.

In the SV method, for the \((\delta e'|\delta e)\) model orbital integral case the CI vector is updated via a matrix-matrix multiplication operation:

\[
\sigma(K_n^{ac}) = P^{ba,dc} C'(K_n^{bd}). \tag{4.4.6}
\]

The \(P^{ba,dc}\) in Eq. 4.4.6 is a matrix indexed by the possible combinations of orbital indices \(a\) and \(c\), and \(b\) and \(d\) in the RAS3 subspace as row and column indices respectively,

\[
P^{ba,dc} = \begin{pmatrix}
(ab|cd) & (ab|cd') & \cdots & (ab|cd'') & (ab'|cd) & \cdots & (ab''|cd'') \\
(ab|c'd) & (ab|c'd') & \cdots & (ab|c'd'') & (ab'|c'd) & \cdots & (ab''|c'd'') \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
(ab|c''d) & (ab|c''d') & \cdots & (ab|c''d'') & (ab'|c''d) & \cdots & (ab''|c''d'') \\
(a'b|cd) & (a'b|cd') & \cdots & (a'b|cd'') & (a'b'|cd) & \cdots & (a'b''|cd'') \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
(a''b|c''d) & (a''b|c''d') & \cdots & (a''b|c''d'') & (a''b'|c''d) & \cdots & (a''b''|c''d'')
\end{pmatrix}. \tag{4.4.7}
\]

The dimension of this matrix is defined as \(\left[\frac{M(\text{RAS3})(M(\text{RAS3})+1)}{2}\right]^2\) for singlet-singlet interaction cases and as \(\left[\frac{M(\text{RAS3})(M(\text{RAS3})-1)}{2}\right]^2\) for triplet-triplet interaction cases.

When the orbital symmetry property is applied \(\left\{ \begin{array}{ll}
b \leq a,d \leq c & \text{singlet} \\
b < a,d < c & \text{triplet} \end{array} \right.\), the dimension of the matrix given by Eq. 4.4.7 will be halved and thus the matrix of Eq. 4.4.7 becomes a triangular matrix. However, for the purpose of showing clearly how the SV method updates the CI vector for this case by using matrix multiplication method, we will use the full square matrix to demonstrate in the following.

229
σ\(K_n^{ac}\) and \(C'(K_n^{bd})\) are two matrices that are indexed by the possible combinations of orbital indices \(a\) and \(c\) (and \(b\) and \(d\)) in the RAS3 subspace as row index and the possible occupation and spin patterns in the RAS2 subspace as column index,

\[
\sigma(K_n^{ac}) = \begin{pmatrix}
K_1^{ac} & K_2^{ac} & \cdots & K_n^{ac} \\
K_1^{ac'} & K_2^{ac'} & \cdots & K_n^{ac'} \\
\vdots & \vdots & \ddots & \vdots \\
K_1^{ac''} & K_2^{ac''} & \cdots & K_n^{ac''} \\
K_1^{ac'''} & K_2^{ac'''} & \cdots & K_n^{ac'''}
\end{pmatrix}, \quad (4.4.8a)
\]

and

\[
C'(K_n^{bd}) = \begin{pmatrix}
K_1^{bd} & K_2^{bd} & \cdots & K_n^{bd} \\
K_1^{bd'} & K_2^{bd'} & \cdots & K_n^{bd'} \\
\vdots & \vdots & \ddots & \vdots \\
K_1^{bd''} & K_2^{bd''} & \cdots & K_n^{bd''} \\
K_1^{bd'''} & K_2^{bd'''} & \cdots & K_n^{bd'''}
\end{pmatrix}, \quad (4.4.8b)
\]

The dimension of these matrices (they share the same dimension) is defined as

\[
\begin{pmatrix}
\frac{M(RAS3)(M(RAS3)+1)}{2} & M(RAS2) \\
N_a - 1 & N_b - 1
\end{pmatrix}
\]

for singlet-singlet interaction case and

\[
\begin{pmatrix}
\frac{M(RAS3)(M(RAS3)-1)}{2} & M(RAS2) \\
N_a - 2 & N_b - 1
\end{pmatrix}
\]

or

\[
\begin{pmatrix}
\frac{M(RAS3)(M(RAS3)-1)}{2} & M(RAS2) \\
N_a & N_b - 2
\end{pmatrix}
\]

for triplet-triplet interaction cases. The matrix \(C'\) given by Eq. 4.4.8b is generated by multiplying the model symbolic matrix elements with the corresponding elements in the CI vector \(C\),
4. Algorithm for the Direct RASSCF Method (Part II)

\[ C'(K_{bd}^{nl}) = \begin{pmatrix} L_1^{bd} \\ L_1^{bd'} \\ \vdots \\ L_1^{b'd'} \\ L_1^{b'd''} \end{pmatrix} B_{\delta'\delta'\delta\delta'}^{K_{bd}^{\delta'\delta}} L_1^{\delta'} B_{\delta'\delta'\delta\delta'}^{L_2^{\delta'} \delta \delta \delta} L_2^{\delta'} \ldots \ldots B_{\delta'\delta'\delta\delta'}^{L_n^{\delta'} \delta \delta \delta} L_n^{\delta'}, \quad (4.4.9) \]

where there will be \( \frac{M(RAS3)(M(RAS3)+1)}{2} \) (or \( \frac{M(RAS3)(M(RAS3)-1)}{2} \)) pairs of \( bd \) indices, and \( \begin{pmatrix} M(RAS2) \\ N_{\alpha} - 1 \end{pmatrix} \begin{pmatrix} M(RAS2) \\ N_{\beta} - 1 \end{pmatrix} \) (or \( \begin{pmatrix} M(RAS2) \\ N_{\alpha} - 2 \end{pmatrix} \begin{pmatrix} M(RAS2) \\ N_{\beta} \end{pmatrix} \)).

possible internal occupation and spin patterns. From Eq. 4.4.9 we can see that the model symbolic matrix elements, \( B_{\delta'\delta'\delta\delta'}^{K_{bd}^{\delta'\delta}} \), are not factorized either.

As we can see in the above example, we know in the SV method for carrying out the CI vector updating from the last 3 model integral cases in Table 4.1 the unfactorized method is used. But since in this MRCISD case the model symbolic matrix elements will remain constant when the model external orbitals are expanded to the full external space, it is still possible to carry out matrix multiplication operation between the integral matrix and the matrix derived from certain block(s) of the CI vector. However, since the introduction of the RAS1 subspace in the general RASSCF method and the values of MxHole and MxElec are no longer restricted to 0 and 2 respectively, the symbolic matrix elements after expanding may not be a constant anymore (Eq. 3.3.22). Thus the CI vector updating process becomes more complicated. However, the idea of the SV method, which is to use the unfactorized method to carry out the CI vector updating for integral types that at least one model orbital index of \( wx \) pair or \( yz \) pair is not in the RAS2 subspace, confirms our assumption made in Section 4.1. Thus based on this idea, we can see the best way of implanting the remaining 15 integral cases in Table 3.1 is to use the unfactorized method.
4.3 An Attempted Implementation to the Remaining Cases And Its Performance

As reviewed in Section 4.2 the SV method [95], for the last 3 orbital integral cases the model symbolic matrix elements are unfactorized. But because in the SV method, the model symbolic matrix elements will remain constant when the model external orbital indices are expanded to the full external space, it is still possible to apply a matrix multiplication scheme between the integral array and the corresponding blocks of the CI vector. However, once the RAS1 subspace is introduced and the value of MxHole is no longer 0, when expanding the model symbolic matrix elements to the full RAS space, a phase factor, \((-1)^n\), should be introduced. This phase factor makes it not possible to carry out a matrix multiplication between the integral array and the corresponding blocks of the CI vector because in this case a new matrix that stores the non-zero expanded symbolic matrix elements is required. As stated in Chapter 1, to store this symbolic matrix elements matrix requires large computer memory capacity; it would thus not be ideal to carry out matrix multiplication in this case. Instead, these obtained non-zero elements should be used directly to update the CI vector (as indicated in Section 1.7.2.2), which is the unfactorized method.

As we can see from Section 4.2, once there is one model orbital index in the model external space, the unfactorized method is used, we may thus carry out the CI vector updating process for the remaining 15 model orbital integral cases in the unfactorized fashion, in the same way as the current RASSCF implementation. Therefore, it becomes very clear that a hybrid method should be used to carry out the implementation of the RASSCF wavefunction. That is when the expansion of the model \(wx\) pair leads to only one full RAS space orbital index pair (the first 7 cases in Table 3.1), we should apply the factorized matrix multiplication method (Eqs. 3.3.19) that is capable to use data in a sequential order. And when the expansion of the model \(wx\) pair leads to more than 1 full RAS space orbital index pairs (the rest cases in Table 3.1), the unfactorized method should be used, because in this case all model orbital indices only need to be expanded
once for all the different expansions from the \(wx\) pair. Based on this idea, we can see the method we have developed in Chapter 3 is already a hybrid method.

However, in the SV method [95], before they analyzed the orbital integral cases, they renormalized the CI vectors. These renormalized CI vectors are used for all the integral cases, irrespective of using the factorized method or unfactorized method. Recall in Chapter 3, we restructured the CI vector (Section 3.6.1) for different configuration basis types, e.g. the Slater determinant basis and the Hartree-Waller basis. The reason for this is to make the data in the CI vector be accessed in a more predictable way. We would like to apply the restructured CI vectors developed in Chapter 3 to the current unfactorized RASSCF implementation for the remaining 15 cases and thus expect a better performance. This implementation is only to replace the CI vectors used in the current implementation; all other steps would be the same as the current method. Thus for the implementation details and subroutine calling sequence, one can refer to reference [44]. In the following we will only report the result of this modified method after applying the restructured CI vectors. Again, all the calculations are carried out on PC cluster with the hardware property as Intel Nehalem servers (2 Intel Nehalem Quad-core 2.5 GHz processors per node with a peak speed of \(~80\) GFlops [128]).

For the verification of the newly modified method, the same calculations used in Section 3.8.1 are used. Since the output of this modified method is identical to the result given in Tables 3.8 and 3.9, we will focus on the performance comparison among the current method (termed as “Old” hereafter), the method developed in Chapter 3 (termed as “\(\text{New}_0\)” hereafter), and the modified method (termed as “\(\text{New}_1\)” thereafter). To carry out this comparison, the same molecule, Coronene (Chart 3.2), is used. And the same calculations used to generate the data in Tables 3.11-3.14 and 3.16-3.18 are carried out by using the \(\text{New}_1\) method again. The corresponding results are given in Tables 4.4-4.10, where Tables 4.4 – 4.7 list the serial performance timing data (the overall timing data only) of these methods and Tables 4.8-4.10 list the parallel timing data of using OpenMP only, using Linda only, and using a hybrid parallelism (2 Linda workers while the number of shared processors varies) of calculation \(\text{RAS}(24,7+10+7)[1,1]\).
### Table 4.4 Timing data of calculations on the cases of the total number of orbitals remains the same while the number of RAS2 subspace varies (single excitation only).

<table>
<thead>
<tr>
<th>Calculation</th>
<th>RAS(24,10+4+10) [1,1]</th>
<th>RAS(24,9+6+9) [1,1]</th>
<th>RAS(24,8+8+8) [1,1]</th>
<th>RAS(24,7+10+7) [1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>11,396</td>
<td>112,450</td>
<td>1,158,948</td>
<td>12,090,456</td>
</tr>
<tr>
<td>No. of full strings</td>
<td>686</td>
<td>1,910</td>
<td>5,446</td>
<td>15,540</td>
</tr>
<tr>
<td>Old Timing</td>
<td>0.81</td>
<td>18.18</td>
<td>468.40</td>
<td>11943.46</td>
</tr>
<tr>
<td>Speedup vs. Old</td>
<td>1.48</td>
<td>2.05</td>
<td>2.84</td>
<td>3.80</td>
</tr>
<tr>
<td>New Timing</td>
<td>0.54</td>
<td>8.88</td>
<td>165.15</td>
<td>3139.59</td>
</tr>
<tr>
<td>Speedup vs. New0</td>
<td>2.53</td>
<td>3.73</td>
<td>5.27</td>
<td>7.21</td>
</tr>
<tr>
<td>Speedup vs. New1</td>
<td>1.69</td>
<td>1.82</td>
<td>1.86</td>
<td>1.90</td>
</tr>
</tbody>
</table>

### Table 4.5 Timing data of calculations on the cases of the total number of orbitals remains the same while the number of RAS2 subspace varies (double excitation only).

<table>
<thead>
<tr>
<th>Calculation</th>
<th>RAS(24,10+4+10) [2,2]</th>
<th>RAS(24,9+6+9) [2,2]</th>
<th>RAS(24,8+8+8) [2,2]</th>
<th>RAS(24,7+10+7) [2,2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>959,926</td>
<td>7,925,332</td>
<td>66,187,044</td>
<td>540,193,248</td>
</tr>
<tr>
<td>No. of full strings</td>
<td>16,526</td>
<td>37,982</td>
<td>86,982</td>
<td>193,452</td>
</tr>
<tr>
<td>Old Timing</td>
<td>30.18</td>
<td>439.94</td>
<td>7243.08</td>
<td>139236.76</td>
</tr>
<tr>
<td>Speedup vs. Old</td>
<td>2.01</td>
<td>2.32</td>
<td>3.39</td>
<td>4.44</td>
</tr>
<tr>
<td>New Timing</td>
<td>14.99</td>
<td>189.37</td>
<td>2134.8</td>
<td>31337.55</td>
</tr>
<tr>
<td>Speedup vs. New0</td>
<td>2.39</td>
<td>3.21</td>
<td>4.47</td>
<td>6.37</td>
</tr>
<tr>
<td>Speedup vs. New1</td>
<td>1.19</td>
<td>1.38</td>
<td>1.32</td>
<td>1.43</td>
</tr>
</tbody>
</table>

### Table 4.6 Timing data of calculations on the cases of the number of RAS2 orbitals remains the same while the number of RAS3 subspace varies (single excitation only).

<table>
<thead>
<tr>
<th>Calculation</th>
<th>RAS(24,8+8+8) [1,1]</th>
<th>RAS(24,8+8+12) [1,1]</th>
<th>RAS(24,8+8+16) [1,1]</th>
<th>RAS(24,8+8+20) [1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>1,158,948</td>
<td>1,704,612</td>
<td>2,250,276</td>
<td>2,795,940</td>
</tr>
<tr>
<td>No. of full strings</td>
<td>5,446</td>
<td>7,910</td>
<td>10,374</td>
<td>12,838</td>
</tr>
<tr>
<td>Old Timing</td>
<td>468.40</td>
<td>501.17</td>
<td>523.39</td>
<td>550.90</td>
</tr>
<tr>
<td>Speedup vs. Old</td>
<td>2.84</td>
<td>2.75</td>
<td>2.67</td>
<td>2.73</td>
</tr>
<tr>
<td>New Timing</td>
<td>165.15</td>
<td>182.43</td>
<td>195.67</td>
<td>201.73</td>
</tr>
<tr>
<td>Speedup vs. New0</td>
<td>5.27</td>
<td>5.12</td>
<td>4.85</td>
<td>4.62</td>
</tr>
<tr>
<td>Speedup vs. New1</td>
<td>1.86</td>
<td>1.86</td>
<td>1.81</td>
<td>1.69</td>
</tr>
</tbody>
</table>
Table 4.7 Timing data of calculations on the cases of the number of RAS2 orbitals remains the same while the number of RAS3 subspace varies (double excitation only).

<table>
<thead>
<tr>
<th>Calculation</th>
<th>RAS(24,8+8+8)</th>
<th>RAS(24,8+8+12)</th>
<th>RAS(24,8+8+16)</th>
<th>RAS(24,8+8+20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>66,187,044</td>
<td>145,209,540</td>
<td>254,977,380</td>
<td>395,490,564</td>
</tr>
<tr>
<td>No. of full strings</td>
<td>86,982</td>
<td>188,286</td>
<td>328,566</td>
<td>507,822</td>
</tr>
<tr>
<td>Overall timing (s) and speedup</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old</td>
<td>Timing</td>
<td>7243.08</td>
<td>18443.47</td>
<td>39641.14</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>New&lt;sub&gt;0&lt;/sub&gt;</td>
<td>Timing</td>
<td>2134.8</td>
<td>4740.75</td>
<td>10695.54</td>
</tr>
<tr>
<td>Speedup vs. Old</td>
<td>3.39</td>
<td>3.89</td>
<td>3.71</td>
<td>4.95</td>
</tr>
<tr>
<td>New&lt;sub&gt;1&lt;/sub&gt;</td>
<td>Timing</td>
<td>1620.88</td>
<td>4663.92</td>
<td>8226.53</td>
</tr>
<tr>
<td>Speedup vs. Old</td>
<td>4.47</td>
<td>3.95</td>
<td>4.82</td>
<td>5.16</td>
</tr>
<tr>
<td>Speedup vs. New&lt;sub&gt;0&lt;/sub&gt;</td>
<td>1.32</td>
<td>1.02</td>
<td>1.30</td>
<td>1.04</td>
</tr>
</tbody>
</table>

From Tables 4.4-4.7 we can see that after applying the restructured CI vector to the remaining 15 cases via the unfactorized method, the modified method can further improve the performance of the method developed in Chapter 3. Moreover, since we only use the restructured CI vectors generated in Chapter 3 without introducing any other arrays, the memory requirement would remain the same as the method developed in Chapter 3. However, the data provided in Tables 4.4-4.7 are only serial data, we would like to see whether this modified method will have as good scalability when it is parallelized as the method developed in Chapter 3. Thus the same calculations that were used for testing the parallel scalability of the method developed in the last chapter are used again here. Tables 4.8-4.10 shows the parallel results of using OpenMP only, Linda only, and hybrid parallelism (i.e. 2 Linda workers while number of shared memory processors varies), respectively. Once again, we can see the modified method developed in this chapter performs as good as the method developed in Chapter 3. Thus we know this attempted implementation to the remaining 15 cases has successfully improved the performance of the RASSCF method developed in Chapter 3, which is already faster than the current RASSCF method.
Table 4.8 Shared memory timing data for the 4\textsuperscript{th} Davidson iteration of the current method (Old), the method developed in Chapter 3 (New\textsubscript{0}), and the modified method developed in this chapter (New\textsubscript{1}) using OpenMP only.

<table>
<thead>
<tr>
<th>System</th>
<th>Coronene: RAS(24,7+10+7)[1,1]</th>
<th>12,090,456</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>12,090,456</td>
<td></td>
</tr>
<tr>
<td>No. of Global string</td>
<td>15,540</td>
<td></td>
</tr>
<tr>
<td>NProcShared =</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Old</td>
<td>CPU time</td>
<td>11943.46</td>
</tr>
<tr>
<td></td>
<td>Wall time</td>
<td>11943.46</td>
</tr>
<tr>
<td></td>
<td>Speedup</td>
<td>-</td>
</tr>
<tr>
<td>New\textsubscript{0}</td>
<td>CPU time</td>
<td>3164.92</td>
</tr>
<tr>
<td></td>
<td>Wall time</td>
<td>3164.92</td>
</tr>
<tr>
<td></td>
<td>Speedup</td>
<td>-</td>
</tr>
<tr>
<td>New\textsubscript{1}</td>
<td>CPU time</td>
<td>1656.74</td>
</tr>
<tr>
<td></td>
<td>Wall time</td>
<td>1656.74</td>
</tr>
<tr>
<td></td>
<td>Speedup</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.9 Distributed memory timing data for the 4\textsuperscript{th} Davidson iteration of the current method (Old), the method developed in Chapter 3 (New\textsubscript{0}), and the modified method developed in this chapter (New\textsubscript{1}) using Linda only.

<table>
<thead>
<tr>
<th>System</th>
<th>Coronene: RAS(24,7+10+7)[1,1]</th>
<th>12,090,456</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Slater Det.</td>
<td>12,090,456</td>
<td></td>
</tr>
<tr>
<td>No. of Global string</td>
<td>15,540</td>
<td></td>
</tr>
<tr>
<td>NProcLinda=</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>NProcShared =</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Old</td>
<td>CPU time</td>
<td>11943.46</td>
</tr>
<tr>
<td></td>
<td>Wall time</td>
<td>11943.46</td>
</tr>
<tr>
<td></td>
<td>Speedup</td>
<td>-</td>
</tr>
<tr>
<td>New\textsubscript{0}</td>
<td>CPU time</td>
<td>3164.92</td>
</tr>
<tr>
<td></td>
<td>Wall time</td>
<td>3164.92</td>
</tr>
<tr>
<td></td>
<td>Speedup</td>
<td>-</td>
</tr>
<tr>
<td>New\textsubscript{1}</td>
<td>CPU time</td>
<td>1656.7</td>
</tr>
<tr>
<td></td>
<td>Wall time</td>
<td>1656.74</td>
</tr>
<tr>
<td></td>
<td>Speedup</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 4.10 Hybrid parallel timing data when using 2 Linda workers and the number of shared memory threads varies of the current method (Old), the method developed in Chapter 3 (New₀), and the modified method developed in this chapter (New₁).

<table>
<thead>
<tr>
<th>System</th>
<th>No. of Slater Det.</th>
<th>No. of Global string</th>
<th>Serial</th>
<th>NProcLinda = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12,090,456</td>
<td>15,540</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old</td>
<td></td>
<td></td>
<td>CPU time</td>
<td>11943.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Wall time</td>
<td>11943.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Speedup</td>
<td>-</td>
</tr>
<tr>
<td>New₀</td>
<td></td>
<td></td>
<td>CPU time</td>
<td>3164.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Wall time</td>
<td>3164.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Speedup</td>
<td>-</td>
</tr>
<tr>
<td>New₁</td>
<td></td>
<td></td>
<td>CPU time</td>
<td>1656.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Wall time</td>
<td>1656.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Speedup</td>
<td>-</td>
</tr>
</tbody>
</table>

4.4 Summary

In this chapter we mainly focus on the development of the remaining 15 integral types in order to further improve the performance of the current RASSCF method. As discussed in Section 3.3.3, the factorized matrix multiplication method will not be suitable for these cases. Thus leaving the unfactorized method as the ideal solution to these cases. As the MRSDCI method developed by Saunders and Van Lenthe [95] deals with different integral types in different ways, i.e. some integral types are treated by factorized matrix multiplication method and others are implemented unfactorized method, a review of this method is given.

Based on the review, we have found that for implementing the RASSCF wavefunction, a hybrid method should be applied. That is for the integral cases when there is at least one $2 \rightarrow 2$ excitation (the first 7 cases in Table 3.1), the factorized matrix multiplication method should be used. When there is no $2 \rightarrow 2$ excitation indicated in the model integral cases (the $8^{th}$ – $22^{nd}$ integral cases in Table 3.1), the unfactorized method, which is the current RASSCF method, should be chosen. However, still based on the review, it is found that the restructured CI vector would be useful. Therefore we developed an
attempted implementation by replacing the CI vector in the current method by the restructured CI vectors developed in Chapter 3. The performance result shows that this attempted implementation can further improve the performance of the RASSCF method developed in Chapter 3. The improvement in performance comes from that the data of the restructured CI vector can be accessed in a more predictable way compared to the CI vector used in the current method. Thus we now have a much better performing RASSCF hybrid implementation for calculating expensive systems that were not affordable under the current method.
Conclusion

The MCSCF methods are powerful tools for studying electronic structure problems of a multi-configurational nature. In the MCSCF wavefunction, both the coefficients of the configurations and the coefficients of the orbitals are optimized simultaneously. The computational bottleneck lies within the CI part of the MCSCF formalism, which is the CI eigenvalue problem. To solve this problem, in modern implementations the full diagonalization of the Hamiltonian matrix is replaced by the iterative diagonalization methods of Lanczos [103] or Davidson [104] (Appendix A). For large MCSCF calculations, it proves more efficient to recompute the non-zero Hamiltonian matrix elements when they are needed. This strategy is termed direct CI [105]. For direct CI, the efficiency of solving the CI eigenvalue problem is affected by two factors: the efficiency of directly computing the non-zero matrix elements when they are needed; and the efficiency of evaluating the CI eigenvector via these obtained non-zero elements. To date, the efficiency of the first factor is already high enough. However, there is still space to improve the efficiency of the second factor in Gaussian [42].

This thesis addresses two ideas to improve the performance of the current MCSCF implementations in Gaussian [42], one for the CASSCF implementation and one for the RASSCF implementation. By factorizing and reorganizing the summations when generating the non-vanishing Hamiltonian matrix elements in the CASSCF Hamiltonian, it is possible to store the directly obtained non-zero matrix elements into a temporary matrix, with rows and columns labeled by strings. Then some highly optimized linear algebra routines can be used to carry out a matrix-matrix multiplication between this smaller temporary matrix and the matrix generated from the CI vector. The improved efficiency comes from the basic linear algebra routines, which are impossible to use in the current CASSCF method in Gaussian due to a huge memory requirement. The newly-implemented CASSCF program can perform faster than the current method when the investigated problem size increases. However, although this new method provides better efficiency, extra memory is required to store the matrices for matrix multiplication. Nevertheless, the extra storage
requirement of the matrices is acceptable due to 1. the dimensions of the matrices are at the numbers of strings level, and 2. developments in modern computer hardware.

The new code has also been parallelized, allowing for distributed memory, shared memory, and hybrid (distributed and shared memory) parallel architectures. Care has been taken to maximize efficiency and to achieve good scaling behavior where possible. Moreover, based on the parallel structure of the current method (single integer task-definition for parallel tasks with an implicit task order), we have introduced one more scheme to the shared memory parallelism, which is to parallelize the program on the matrix multiplication level. This results in more flexible choices being available to users for carrying out parallel calculations. One can either choose to run shared memory parallel on the single integer level, or on the matrix multiplication level for one single integer, or on a hybrid level. The algorithm has been implemented as part of the Gaussian development set of programs [42]. The performance of the newly-developed method has been tested by running both serial and parallel calculations on pyracylene and pyrene, using both the new method and the current method. Pyracylene is a 14 electrons in 14 active orbitals system and pyrene has a 16 electrons in 16 active orbitals. When the same type of computer hardware and calculation condition are used, since the two methods have similar parallel performance, both serially and in terms of parallelism, the new method can perform between ~7 and ~11 times faster than the current method on the pyracylene and pyrene systems respectively. In order to demonstrate the newly developed method is capable of use for “real world” problems, ground state geometry optimization calculations on two hydrocarbon cations (anthracene$^+$ and phenanthrene$^+$, both CASSCF(13,14)) are carried out. The result shows that the new method performs much faster than the current method.

The second methodological contribution of this thesis, which is a hybrid method that combines the factorized algorithm together with the unfactorized algorithm, relates to the RASSCF method. This wavefunction allows much larger active spaces than the CASSCF wavefunction to be calculated, thus enabling studies of large systems. This type of wavefunction can also be used to enhance qualitative and quantitative accuracy by partially including the dynamical electron correlation effects that the CASSCF wavefunction usually doesn’t recover (because a larger active space is
required). By adapting the direct non-zero symbolic matrix elements generation scheme of the current method, we have reorganized the evaluation of the CI eigenvector so that the factorized matrix multiplication algorithm developed for the CASSCF calculation can be used for RASSCF. However, because there are more integral cases for RASSCF than for CASSCF, this factorized scheme is only suitable for modifying the seven most time-consuming integral types. For the other integral cases, the CI eigenvector evaluation is better carried out via the unfactorized algorithm but using a restructured CI vector that is generated for the seven integral cases. The improved efficiency also comes from the data in the restructured CI vector being accessible in a more predictable order. The newly-developed RASSCF algorithm has been implemented as part of the Gaussian development set of programs [42]. The parallel version allows for distributed, shared and hybrid parallelism. The performance of the new algorithm has been evaluated by carrying out a series of calculations on coronene. The results show that the new RASSCF code can perform four to seven times faster than the current method under the same condition, e.g. the same type of calculation on the same type of hardware. The memory requirement for the temporary matrix is also acceptable due to the development of modern computer hardware. However, in the RASSCF wavefunction, the CI vector is divided into blocks according to the occupation restrictions in the RAS1 and RAS3 subspaces. Future RASSCF development can take the advantage of this feature of the CI vector to reduce the memory requirement from storing the entire CI vector to storing only a few blocks of the CI vector, thus meaning RASSCF calculations be carried out when the available memory is limited.

The developments of the MCSCF methods described in this thesis are capable of being used for calculating larger systems than the current MCSCF methods. Thus the computation limits of the current MCSCF methods have been successfully pushed further.
Appendix A.
The Second Quantisation Formalism and Hamiltonian Matrix Diagonalization

A.1 The Second Quantisation Formalism

A.1.1 The Fock space

In the first quantisation formalism, the observables are represented by operators and the wavefunctions are normal functions. In the second quantisation formalism, the wave functions are also expressed in terms of operators. The formalism starts with the introduction of an abstract vector space, the Fock space, which depends on the occupation numbers of orbitals in the basis functions. Since the definition of the Slater determinant is already given in chapter 1, we will use it directly here. The occupation of a given Slater determinant can be written as an occupation number vector, \( \{ n \} \),

\[
\{ n \} = |n_1, n_2, \ldots, n_m \rangle, \quad n_i = 0, 1 \quad \text{for } i = 1, 2, \ldots, m, \quad (A.1)
\]

where \( n_i \) is one if spin orbital \( \phi_i \) is occupied and zero if \( \phi_i \) is unoccupied. For a given spin-orbital basis, there is a one-to-one mapping between a Slater determinant and an occupation number vector in the Fock space. Thus we can write the Slater determinant in the form of an occupation number vector:

\[
|K\rangle = |\phi_1, \phi_2, \ldots, \phi_p \rangle \nonumber \\
= |i, j, \ldots, p \rangle. \quad (A.2)
\]

A Fock space contains all the vectors:

\[
|\text{vac}\rangle, |i\rangle, |ij\rangle, \ldots, |ij\ldots p\rangle, \ldots \quad (A.3)
\]
A. The Second Quantisation Formalism and Hamiltonian Matrix Diagonalization

where $|\text{vac}\rangle$ defines the true vacuum state containing no particles:

$$|\text{vac}\rangle = |0,0,\ldots,0\rangle,$$

(A.4)

and $|i\rangle,|j\rangle$ indicate 1 electron space, etc.

A.1.2 Annihilation and Creation Operators

We now define an annihilation operator, $a_i$, acting on the elements of the Fock space, that it removes an electron from the $i^{th}$ spin-orbital to give a ket with N-1 electrons occupied. However, if the $i^{th}$ spin-orbital of the ket was not occupied but the annihilation operator, $a_i$, is acting on this ket, then the determinant will vanish:

$$a_i |rj\ldots pi\rangle = |rj\ldots p\rangle$$

(A.5)

If $i$ is contained within the ket, but not at its end, we need to add a factor:

$$a_i |de\ldots p\rangle = \begin{cases} (-1)^{v_i} |de\ldots f\ldots p\rangle & i \in \{d,e,\ldots,p\} \\ 0 & i \notin \{d,e,\ldots,p\} \end{cases},$$

(A.6)

where $v_i$ is the number of permutations required to move $i$ to the end. Therefore it is clear that:

$$a_i |ji\rangle = |j\rangle, \quad a_j |j\rangle = |\text{vac}\rangle.$$  

(A.7)

The anti-symmetry property of the Slater determinant implies a change of sign of the ket: $|ij\rangle = -|ji\rangle$, so using Eq. A.7 we get:

$$a_ia_j |ij\rangle = |\text{vac}\rangle = a_ja_i |ji\rangle = -a_ia_j |ij\rangle,$$

(A.8)

Therefore the annihilation operator is be said to ‘anti-commute’:

$$a_ia_j + a_ja_i = [a_i,a_j] = 0.$$  

(A.9)

The creation operator can be similarly defined:

$$a_i^\dagger |jk\ldots p\rangle = \begin{cases} |jk\ldots pi\rangle & i \notin \{j,k,\ldots,p\} \\ 0 & i \in \{j,k,\ldots,p\} \end{cases}. $$

(A.10)

Similarly, we know the creation operator is anti-commuted too;

$$[a_i^\dagger,a_j^\dagger] = 0.$$  

(A.11)
The combination of the creation and annihilation operators changes an occupied spin orbital from \( r \) to \( s \) by:

\[
\begin{align*}
a^\dagger_r a_r |ij \cdots p\rangle &= a^\dagger_r a_r (-1)^{\gamma_r} |ij \cdots pr\rangle \\
&= a^\dagger_r (-1)^{\gamma_r} |ij \cdots p\rangle \\
&= (-1)^{\gamma_r} |ij \cdots ps\rangle \\
&= (-1)^{\gamma_r} (-1)^{\gamma_r} |ij \cdots sp\rangle \\
&= (-1)^{2\gamma_r} |ij \cdots sp\rangle.
\end{align*}
\]

(A.12)

The factor \((-1)^{\gamma_r}\) is multiplied by a second factor, \((-1)^{\gamma_s}\), which moves \( s \) from the end of the list to the position it should be in. If \( r = s \), this moves \( s \) from the end of the list to the position previously occupied by \( r \), and thus \((-1)^{2\gamma_r} = 1\). Therefore:

\[
\begin{align*}
a^\dagger_r a_r |ij \cdots p\rangle &= \begin{cases} 
|i\cdots p\rangle & r \in \{i,j,\cdots,p\} \\
0 & r \notin \{i,j,\cdots,p\}
\end{cases} \\
a_r a^\dagger_r |ij \cdots p\rangle &= \begin{cases} 
|i\cdots p\rangle & r \notin \{i,j,\cdots,p\} \\
0 & r \in \{i,j,\cdots,p\}
\end{cases}.
\end{align*}
\]

(A.13)

So \((a^\dagger_r a_r + a_r a^\dagger_r)\) is the unit operator for any ket. The anti-commutation relationship for these operators can be therefore generated as:

\[
[a^\dagger_r, a_s]_\alpha = \delta_{\alpha s}.
\]

(A.14)

### A.1.3 The Excitation operator

In the derivation of the MCSCF energy expression we will assume that the Hamiltonian does not contain any spin-dependent terms. It is then possible to formulate the theory in terms of spin summed excitation operators, \( E_{ij} \):

\[
E_{ij} = E_{ij}^G + E_{ij}^B = a^\dagger_\alpha a_\beta + a^\dagger_\beta a_\alpha.
\]

(A.15)

The indices \( i \) and \( j \) in the far left hand side of Eq. A.15 now refer to the \( n \) molecular orbitals without the spin factor. These operators fulfill the same commutation relation as
the generators of the unitary group of dimension \(n\), \(U(n)\), and are often referred to as generators. By applying Eq. A.15 and Eq. A.14, the commutator relation is:

\[
\begin{bmatrix} E_{ij} & E_{kl} \end{bmatrix} = E_{ij}E_{kl} - E_{kl}E_{ij}
= a_i^\dagger a_j^\dagger a_l a_i - a_i^\dagger a_j a_i^\dagger a_l
= a_i^\dagger (\delta_{jk} - \delta_{kl}) a_l - a_i^\dagger (\delta_{il} - \delta_{ij}) a_l.
\]

\(A.16\)

The definition in Eq. A.15 leads to the following relation for the adjoint operator:

\[
E_{ij}^\dagger = E_{ji}.
\]

\(A.17\)

We find that if \(E_{ij}\) is operating on a determinant where orbital \(j\) is unoccupied, the result 0 will be produced. Similarly, if the orbital \(i\) is doubly occupied, 0 is obtained when \(i \neq j\).

When \(i = j\) the result is two times the ket:

\[
E_{ii} |m\rangle = n_i |m\rangle.
\]

\(A.18\)

where \(n_i\) is the occupation number for the molecular orbital \(i\) (0, 1, or 2).

**A.2 Hamiltonian Matrix Diagonalization**

When the matrix representation of the Hamiltonian of the configuration basis,

\[
H_{KL} = \langle K | H | L \rangle
= \sum_{ij} (i|j) \langle K | \sum_\xi \sum_\gamma a_{\xi i}^\dagger a_{\gamma j} \| L \rangle + \frac{1}{2} \sum_{ijkl} (ij|kl) \langle K | \sum_\xi \sum_\gamma a_{\xi i}^\dagger a_{\gamma k} a_{\gamma l}^\dagger a_{\gamma j} \| L \rangle \ (\xi, \gamma \in \{\alpha, \beta\}),
\]

\(A.19\)

is formed, it must be diagonalized to obtain the eigenvalues and eigenvectors. There are many methods of diagonalizing matrices. Two important methods of these will be reviewed here. The first one is the full diagonalization, or the Jacobi method. This method requires the full matrix to be explicitly built before the off-diagonal elements are reduced to zero by iterative operation of plane rotations. The advantages of this method are that it is simple and it gives the complete list of eigenvalues and eigenvectors. The disadvantage is that the storage of the full \(N_{\text{conf}} \times N_{\text{conf}}\) (\(N_{\text{conf}}\) gives the number of
configurations) matrix requires sufficient computer memory. Even if the symmetry of the matrix is used and only one triangle of this matrix is stored, the memory requirement would still be $N_{\text{conf}} \times (N_{\text{conf}} + 1)/2$ words of space. Since the number of configurations increases rapidly with the number of active orbitals, this will be impossible for even modest active spaces.

Another method is the Lanczos tri-diagonalization method [103]. The algorithm can be summarized as,

$$H C^n = \varepsilon_n C^{n+1} + \alpha_n C^n + \beta_{n-1} C^{n-1}. \quad (A.20)$$

The Hamiltonian is operated on a trial vector $C$, producing a sum of three vectors, scaled by the constants $\alpha$, $\beta$, and $\varepsilon$. This sum is sassily resolved into its components to give the constants, which form the elements of the tri-diagonal representation of the Hamiltonian, and a new trial vector. The procedure is then reoperated on this new vector. After $n$ iterations, estimates for the lowest $n$ eigenvalues of the Hamiltonian can be obtained from the tri-diagonal matrix. It is found that the lowest eigenvalues converge fast, so the procedure is continued until the selected number of eigenvalues has converged. The eigenvectors corresponding to these eigenvalues can be back-transformed from the list of the vectors produced at each iteration.

The advantages of this method are: 1. One can use the Hamiltonian matrix implicitly. From the left hand side of Eq. A.20 we can see it is a matrix-vector multiplication. The individual multiplications that comprise this matrix-vector multiplication can be performed in any order. Thus the assembling of the Hamiltonian matrix explicitly is avoided. 2. All that is necessary is a list of the matrix elements. Convergence depends on how close the trial vector is to the eigenvector. Failure to converge is rare but may occur if the final eigenvector is deficient in the configurations present in the trial vector. And 3. The procedure only requires enough memory to store a few vectors of length $N_{\text{conf}}$, and enough disc space to store the trial vectors and the elements of the tri-diagonal form. Therefore, this is the method of choice for computing systems with large active space.
Appendix B:  
CASSCF Implementation Details

In the new CASSCF implementation in the *Gaussian* program package [42], a new Iop overlay, 5/139, is introduced:

\[ \text{Iop}(139) \quad \cdots \quad \text{When using Iop}(39) \text{ to carry out On-Fly calculation, this is an option tag for choosing to use the CAS/RAS method developed by Klene et al [43,44] (the current implementations) or the matrix multiplication method. When this is set greater than 0, the matrix multiplication method will be used. For the new CAS method, the value of this decides the number of processors used for } k_l \text{ level (matrix multiplication level) shared memory parallelism. (The distributed memory parallelism is always on } i_j \text{ level)} \]

This can be set as:

\[ \text{Null or 0} \quad \cdots \quad \text{Using the CAS/RAS method developed by Klene.} \]

\[ 1 \quad \cdots \quad \text{For CAS: using } i_j \text{ level shared memory parallelism only.} \]

\[ \text{For RAS: using RASSCF matrix multiplication method.} \]

\[ \text{NProcS} \quad \cdots \quad \text{CAS only: using } k_l \text{ level shared memory parallelism only.} \]

\[ N \leq \frac{1}{2} \times \text{NProcS} \quad \cdots \quad \text{CAS only: } N \text{ processors are used for shared memory } k_l \text{ level parallelism and } Trunc(\text{NProcS}/N) \text{ (in integer arithmetic) processors are used for shared memory } i_j \text{ level parallelism.} \]

The subroutine calling sequence flow chart of the CASSCF matrix multiplication implementation is given in Figure B.1. The distributed memory Linda parallel subroutine calling sequence can be found in Figures B.2.

We have implemented the CASSCF matrix-matrix multiplication algorithm, described in Chapter 2, as part of the *Gaussian* program package [42]. Our implementation accelerates the set of routines previously performing the same tasks within the direct CASSCF part.
of link l510, which performs the MCSCF computations in *Gaussian*. A brief description of each subroutine is given. For simplicity, we use the Slater determinant basis as main example.

**FlyUp / MCAXMl** Top level routines. Perform memory allocation for different methods. FlyUp is used for Lanczos iteration method and MCAXMl is used for Davidson iteration method. In these subroutines, call subroutine UpdatM for the method developed by Mike Klene [43], call subroutine UpdaMX for Slater determinant basis when \( N_\alpha \neq N_\beta \) (\( N_\xi \) gives the number of \( \xi \)-electrons \( \xi \in \{\alpha, \beta\} \)), and call UpdaXM for all other cases.

**UpdaMX / UpdaXM** In these subroutines, the Handy’s index matrix, IIIZ, is computed. All the reduced lists \((L_{N\xi-1}^{M-1}, L_{N\xi-1}^{M-2}, L_{N\xi-2}^{M-2}, L_{N\xi-2}^{M-3}, L_{N\xi-2}^{M-4})\) are generated. In the matrix-matrix multiplication method, for Slater determinant basis, we only need the one particle reduced lists \((L_{N\xi-1}^{M-1} \text{ and } L_{N\xi-1}^{M-2})\). But since all the other reduced lists are necessary for the Hartree-Waller function cases, we keep the two-particle reduced lists in these subroutines. Linda parallelization is setup in these subroutines. The integral list is read from file. In UpdaMX, call subroutine UpdWrX; and in UpdaXM call subroutine UpdXWr.

**UpdWrX / UpdXWr (Parallel)** parallelism of the CASSCF matrix multiplication method is carried out in these subroutines. As mentioned above, the distributed memory parallel is always carried out on the level of \(ij\) index pairs (each Linda worker has one unique \(ij\) index pair) while the shared memory parallel is carried out in two levels: one is on \(ij\) level (each shared memory processor has one unique \(ij\) index pair), which is the same parallel structure as
the method developed by Klene et al [43]; the other is on the level of matrix multiplication (all shared memory processors share the same \(ij\) index pair). A hybrid of both levels can also be used. The matrix multiplication process for linear contribution is carried out in these routines too (Figure B.4c). In UpdWrX, call subroutine UpdWrk; and in UpdXWr, call subroutine UpTask.

- Serial version (no parallelism): loop over orbital index \((ij) = \frac{i(i-1)}{2} + j\)
- Shared memory parallelism only: Create \(NProcij\) \(ij\) level shared memory processes and \(NProckl\) matrix multiplication level shared memory processes. Let several threads compute the sub-task \((ij)^a\), \((ij)^a - NProcij\), \((ij)^a - 2NProcij\), \(\cdots\), where \((ij)^a\) is the first task-index allocated to the thread labelled as \(a\); threads \(a, b, \cdots\) will be allocated first sub-task indices \((ij)^a = \frac{M(M+1)}{2}\), \((ij)^b = (ij)^a - 1\), \(\cdots\). Each of these threads will use \(NProckl\) threads (including the thread itself) to carry out the matrix multiplication parallelism. If the total number of the threads on one node is defined as \(NProcS\), the relation between \(NProcij, NProckl\), and \(NProcS\) is: \(NProcij \times NProckl \leq NProcS\)
- Distributed memory parallelism (Linda) only: read \((ij)\) from tuple space, and return \((ij)' = (ij) - 1\) to tuple space
- Mixed (shared and distributed) parallel jobs are allocated to nodes as blocks of \(NProcij\) sub-tasks, i.e. \((ij)' = (ij) - NProcij\) is returned to tuple space. One block of sub-task is allocated to threads on one node based on the shared memory parallelism described above.

**UpdWrk** Umbrella routine to update Slater determinants basis when \(N_a \neq N_b\). See Figure B.3a. Call UpXLin and UpdXS.
UpTask Umbrella routine to update Slater determinant cases when $N_\alpha = N_\beta$, Hartree-Waller singlet and triplet cases. See Figure B.3b. Call UpSLin and UpdSX for Slater determinants basis when $N_\alpha = N_\beta$, and call UpHWLi and UpdHWX for Hartree-Waller functions cases.

Ind2IJ Translate the task index $(ij)$ to orbital indices $i$ and $j$.

UpXLin / UpSLin Using UpSLin as representative in Figure B.4. Generating linear contribution to the CI vector updating for the Slater determinants basis.

UpHWLi Like UpSLin, but for Hartree-Waller functions basis

UpdXS / UpdSX Using UpdSX as representative in Figure B.5. Generating bilinear contribution to the CI vector updating for the Slater determinants basis.

UpdHWX Like UpdSX, but for Hartree-Waller functions basis
Figure B.1: Subroutine calling sequence for CASSCF matrix multiplication CI vector updating algorithm.
Figure B.2a: Linda communication for the *Slater determinant basis* when $N_\alpha \neq N_\beta$ of the CASSCF matrix multiplication algorithm for CI vector updating.
Figure B.2b: Linda communication for the *Slater determinant basis* when \( N_\alpha = N_\beta \), and *Hartree-Waller functions basis* of the CASSCF matrix multiplication algorithm for CI vector updating.
Figure B.3a: CI vector updating process for the Slater determinant basis when $N_\alpha \neq N_\beta$ in the CASSCF matrix multiplication algorithm.
Figure B.3b: CI vector updating process for the Slater determinant basis when $N_\alpha = N_\beta$ and the Hartree-Waller functions basis in the CASSCF matrix multiplication algorithm.
Figure B.4a: Subroutine UpSLin generates the matrix elements for the linear contribution to the CI vector updating (1e linear). If using Slater determinants basis, the matrix elements for $\beta$-electron contribution must be generated.
Figure B.4b (continued from Figure B.4a): Subroutine UpSLin generates the matrix elements for the linear contribution to the CI vector updating (2e linear). If using Slater determinants basis, the matrix elements for \( \beta \)-electron contribution must be generated.
Figure B.4c: Matrix multiplication for the linear contribution to the CI vector updating is carried out in subroutines UpdWrX and UpdXWr.

\[
\mathbf{M}^{\text{lin}}(K_\alpha, K_\beta) = \mathbf{X}^{\text{lin}}(K_\alpha, L_\alpha)\mathbf{C}(L_\alpha, K_\beta) + \left[ \mathbf{X}^{\text{lin}}(K_\beta, L_\beta)\mathbf{C}(K_\alpha, L_\beta)^T \right]^T
\]

If UpdXWr, \(\mathbf{X}^{\text{lin}}(K_\alpha, L_\alpha) = \mathbf{X}^{\text{lin}}(K_\beta, L_\beta)\)

Updating CI vector from linear contribution:

\[
\sigma(K_\alpha, K_\beta) = \sigma(K_\alpha, K_\beta) + \mathbf{M}^{\text{lin}}(K_\alpha, K_\beta)
\]
B. CASSCF Implementation Details

Figure B.5a: In subroutine UpdSX, the matrix $X$ for all possible $(kl)$ index pairs corresponding to one $(ij)$ index is generated. Then this matrix is used for matrix multiplication to generate the intermediate matrix $M$. The CI vector is then updated by carrying out scalar times a vector style calculation on $M$ (shown in Figure B.5b).
Figure B.5b (continued from Figure B.5a): After the matrix $\mathbf{M}$ for certain $(ij)$ index is obtained, scalar times a vector type calculation is used for updating the CI vector of this $(ij)$ index. The excitation list, $\chi^j_i$, in this figure is generated from subroutine UpXLin / UpSLin (Figure B.4a).
Figure B.6: Subroutine calling sequence for density matrix algorithm.
Figure B.7: Subroutine calling sequence for diagonal Hamiltonian algorithm
Figure B.8: Subroutine calling sequence for reduced Hamiltonian algorithm.
Figure B.9: Linda communication for density matrix algorithm.

Figure B.10: Linda communication for diagonal Hamiltonian algorithm.
Figure B.11: Linda communication for reduced Hamiltonian algorithm.
Appendix C: RASSCF Implementation Details

We have implemented the RASSCF matrix multiplication algorithm for the most time consuming 7 cases of the CI vector updating process described in Chapter 3 as part of the Gaussian program package [42]. Our implementation accelerates the set of routines previously performing the same tasks within the direct RASSCF part of the existing MCSCF program (link l510). In order to achieve an overall better performance on the basis of the method developed in Chapter 3, for the other 15 integral types, we replaced the CI vector that is used in the current method by restructured CI vectors. However, since the modification only involves CI vector replacement, here we will only give the implementation details of the method developed in Chapter 3. The serial and parallel executions are both implemented. Figures C.1 and C.2 show the subroutine-tree for serial and parallel execution, respectively. A brief description of each subroutine is given. For simplicity, we will also use the Slater determinant cases as main example.

FlyRas / MCAXMR Top level routines. Perform memory allocation for different methods. FlyRas is used for Lanczos iteration method and MCAXMR is used for Davidson iteration method. In these subroutines, call subroutine UpdRas for the method developed by Mike Klene (the original method [44]), call subroutine UpSRas for Slater determinant basis when $N_\alpha \neq N_\beta$, and call UpXRas for all other cases.

UpSRas / UpXRas Compute the lengths of all the string categories. Compute all permissible combinations of categories. For Slater determinant cases, generate the transposed form of the CI vector, $C^T$, and for Hartree-Waller functions cases, generate normalized matrix from the CI vector. Compute reduced string lists for model spaces. Build Handy’s index array IIZ.
C. RASSCF Implementation Details

Parallelization is set up in these subroutines (put all the necessary data into tuple space). In UpSRas, call subroutine RasWok; and in UpXRas call subroutine RasWrk.

**RasWok / RasWrk (Parallel)**

Parallelism of the RASSCF matrix multiplication method is carried out in these subroutines. Define the task list by the model space orbital pairs $w_x$. In RasWok, call subroutine RasTsk; and in RasWrk, call subroutine RasJob. For Slater determinant cases, the transpose of the CI vector $\sigma^T$ (obtained from $C^T$) is added to the updated CI vector $\sigma$ ($\sigma = \sigma + (\sigma^T)^T$); for Hartree-Waller functions cases, scatter the elements of the updated CI vector (obtained from normalized CI vector) to the CI vector for the Hartree-Waller functions.

- **Serial version (no parallelism):** simple loop over model orbital index

$$ (w_x) = \frac{w(w-1)}{2} + x $$

- **Shared memory parallelism only:** Create $NProcS$ shared memory processes and let each thread compute the sub-task $(w_x)^a_0$, $(w_x)^a_0 - NProcS$, $(w_x)^a_0 - 2NProcS$, ..., where $(w_x)^a_0$ is the first task-index allocated to the thread labelled as $a$; threads $a$, $b$, ... will be allocated first sub-task indices $(w_x)^a_0 = \frac{M^m(M^m + 1)}{2}$, $(w_x)^b_0 = (w_x)^a_0 - 1$, ...

  ($M^m$ indicates the number of model orbitals).

- **Distributed memory parallelism (Linda) only:** read $(w_x)$ from tuple space, and return $(w_x)' = (w_x) - 1$ to tuple space

- **Mixed (shared and distributed) parallel jobs are allocated to nodes as blocks of $NProcS$ sub-tasks, i.e. $(w_x)' = (w_x) - NProcS$ is returned to tuple space. One block of sub-task is allocated to threads on one node based on the shared memory parallelism described above.
I2wx Check if task corresponding to model space indices $w, x$ is valid.

RasTsk Umbrella routine to update Slater determinants basis when $N_\alpha \neq N_\beta$. If the model orbital indices $w$ and $x$ are both in RAS2 subspace, then matrix multiplication method is used. Call RJbDAA for linear contribution, call RJABD1 / RJABD2 for bilinear intermediate matrix generation, and call RBiUp2 for matrix multiplication and CI vector updating. The original method is used otherwise. Call RJbDAA for linear contribution and call RJobAB for bilinear contribution (the original method). This is represented by RasJob in Figure C.3.

RasJob See Figure C.3. Umbrella routing to update Hartree-Waller functions basis and Slater determinants basis when $N_\alpha = N_\beta$. If the model orbital indices $w$ and $x$ are both in RAS2 subspace, then matrix multiplication method is used (all possible $yz$ pairs corresponding to this $wx$ pair is looped over). Call RJbXAA for linear contribution. Call RJABC1 and RJABC2 for Slater determinants bilinear intermediate matrix generation. Call RJHWCS and RJHWS2 for Hartree-Waller functions bilinear intermediate matrix generation. Call RUpHWS for Hartree-Waller functions matrix multiplication and CI vector updating. And call RBiUp3 for Slater determinant matrix multiplication and CI vector updating. The original method for the bilinear contribution but with a replaced CI vector is used otherwise. Call RJbXAA for linear contribution and call RXJbAB for bilinear contribution (the original method with a replaced CI vector).

IPBack Map task index to model space index pair $w, x$.

Srtl1e Assemble 1e integral list, $(i | j)$, corresponding to orbital indices $i, j$ ($w \Rightarrow i, x \Rightarrow j$) for the 1e linear contribution when calling RJbDAA / RJbXAA.

GetICl Compute integral class.

Srtl2e Assemble 2e integral list, $(ij | kl)$, based on the global orbital indices $i, j, k, l$ obtained from model orbital indices $w, x, y, z$. This list is used for 2e linear
C. RASSCF Implementation Details

contribution when calling RJbDAA / RJbXAA and bilinear contribution when calling RJABD1 / RJABC1 and RJABD2 / RJABC2.

**RJbDAA / RJbXAA** Get propagation rule. Decide 1e linear or 2e linear contribution. In RJbDAA call RUpSAA and in RJbXAA call RUpXAA.

**GetPrp** Compute propagation rules for RAS1 and RAS3.

**RUpSAA / RUpXAA** See Figure C.4 (RUpXAA as representative)

**GetSgn** Compute sign factors corresponding to propagators in RAS1 and RAS3.

**Insrt** Insert bits into reduced model string.

**GetKLs** Provide RAS1 and RAS3 subspaces excitation lists. Before the list generation, a valid test of the model strings and their corresponding string categories is carried out (Figure C.5).

**Prpgt1** Apply propagation rule to the appropriate RAS substring.

**UpSSAA, UpXSAA, UpSXAA, and UpSXBB** Carry out the linear contribution for the Hartree-Waller functions (UpSSAA), Slater determinant basis when \( N_\alpha = N_\beta \) (UpXSAA), and Slater determinant basis when \( N_\alpha \neq N_\beta \) (UpSXAA and UpSXBB). See Figure C.6 (UpXSAA as representative).

**GeOLst / GeOtLt** Generate the outer excitation list from the model orbital index pair \( w, x \) for bilinear contributions (Figure C.7).

**RJHWCS / RJABC1, RJABD1** Generate the intermediate matrix \( M \) for the excitation cases of \( \{2 \rightarrow 2, 1 \rightarrow 1\}, \{2 \rightarrow 2, 1 \rightarrow 2\}, \text{ and } \{2 \rightarrow 2, 2 \rightarrow 2\} \) (where \( X \rightarrow Y \) indicates an excitation from RASX subspace to RASY subspace). RJHWCS is for Hartree-Waller functions basis, RJABC1 is for Slater determinant basis when \( N_\alpha = N_\beta \), and RJABD1 is for Slater determinant basis cases when \( N_\alpha \neq N_\beta \). In RJHWCS call RUHWCS, in RJABC1 call RUABC1, and in RJABD1 call RUABD1.
RUHWCS / RUABC1, RUABD1 Generate the intermediate matrix $M$ for the excitation cases of $\{2 \rightarrow 2, 1 \rightarrow 1\}$, $\{2 \rightarrow 2, 1 \rightarrow 2\}$, and $\{2 \rightarrow 2, 2 \rightarrow 2\}$. See Figure C.8 (RUABC1 as representative).

RJHWS2 / RJABC2, RJABD2 Generate the intermediate matrix $M$ for the excitation cases of $\{1 \rightarrow 2, 2 \rightarrow 2\}$, $\{1 \rightarrow 3, 2 \rightarrow 2\}$, $\{2 \rightarrow 3, 2 \rightarrow 2\}$, $\{3 \rightarrow 3, 2 \rightarrow 2\}$. In RJHWS2 call RUHWS2, in RJABC2 call RUABC2, and in RJABD2 call RUABD2.

RUHWS2 / RUABC2, RUABD2 Generate the intermediate matrix $M$ for the excitation cases of $\{1 \rightarrow 2, 2 \rightarrow 2\}$, $\{1 \rightarrow 3, 2 \rightarrow 2\}$, $\{2 \rightarrow 3, 2 \rightarrow 2\}$, $\{3 \rightarrow 3, 2 \rightarrow 2\}$. RUHWS2 is for Hartree-Waller functions basis, RUABC2 is for Slater determinant cases when $N_\alpha = N_\beta$, and RUABD2 is for Slater determinant cases when $N_\alpha \neq N_\beta$. See Figure C.9 (RUABC2 as representative).

RUpHWS / RBiUp3, RBiUp2 Update the CI vector from the outer excitation list generated from the model orbital index pair $wx$. RUpHWS is for Hartree-Waller functions basis, RBiUp3 is for Slater determinant cases when $N_\alpha = N_\beta$, and RBiUp2 is for Slater determinant cases when $N_\alpha \neq N_\beta$. See Figure C.10 (RBiUp3 as representative).
C. RASSCF Implementation Details

Figure C.1: Subroutine calling sequence for RASSCF matrix multiplication CI vector updating algorithm. For Hartree-Waller functions, simply replace the subroutines RJABC1/RUABC1 and RJABC2/RUABC2 to RJHWCS/RUHWCS and RJHWS2 / RUHWS2, respectively.
Figure C.2a: Linda communication for the *Slater determinant basis* when $N_\alpha \neq N_\beta$ of the RASSCF matrix multiplication algorithm for CI vector updating.
Figure C.2b: Linda communication for the Slater determinant basis when $N_\alpha = N_\beta$, Hartree-Waller functions basis of the RASSCF matrix multiplication algorithm for CI vector updating.
Figure C.3: CI vector updating process for the 7 time consuming cases in the RASSCF matrix multiplication algorithm. This is for the general cases of Hartree-Waller functions basis (RJHWCS/RJHWS2) and / or Slater determinant basis when $N_\alpha = N_\beta$ (RJABC1/RJABC2). For Slater determinant basis when $N_\alpha \neq N_\beta$, simply replace the subroutines RJbXAA as RJbDAA, RJABC1/RJABC2 as RJABD1/RJABD2, RBiUp3 as RBiUp2, and RXJbAB as RJofAB.
C. RASSCF Implementation Details

Figure C.4: Linear contribution of the RASSCF method is carried out in RU\textsc{p}X\textsc{a}A. In this subroutine, the full model strings are generated from the reduced model strings, then the list of RAS1 and RAS3 subspaces expansion is generated. If Slater determinants basis is used and $N_\alpha \neq N_\beta$, the routine Up\textsc{x}s\textsc{a}A is replaced by Up\textsc{x}s\textsc{a}A, and the procedure is repeated for $\beta$-model strings by using routine Up\textsc{x}XB.
Figure C.5: Subroutine GetKLs provides the RAS1 and RAS3 subspace excitations lists. Before proceeding with excitation list generation it performs validity tests for the model strings and their corresponding categories, Cat\((i_n, i_e)\).
Figure C.6: The subroutine UpXSAA represents all other subroutines (UpSSAA, UpSXAA, and UpSXBB) for the linear contribution to the CI vector updating.
Figure C.7: Subroutine GeOLst generates the excitation list for the model orbital index pair \( w_x \). This list is only used for bilinear contribution of the RASSCF matrix multiplication algorithm. The process of list elements generation is very similar to the way of elements generation of the linear contribution shown in Figure C. 6 but much simpler.
C. RASSCF Implementation Details

Figure C.8a: Subroutine RUABC1 generates the elements of the intermediate matrix \( \mathbf{M} \). This routine operates the matrix generation for the excitation cases of \( (2 \rightarrow 2, 1 \rightarrow 1), (2 \rightarrow 2, 1 \rightarrow 2), \) and \( (2 \rightarrow 2, 2 \rightarrow 2) \). Routines RUHWCS and RUABD1 do similar job.

\[ y = z \quad ? \]

Loop over reduced model string list

Insert bits \( h_y, h_z \)

(Incret)

Compute \( \text{sgn}_y \).

(INumGt)

Get complete list of

\( \text{Addr} \{ \mathbf{K}_{\mathbf{RASSX}} \}, \text{Addr} \{ \mathbf{L}_{\mathbf{RASSX}} \} \)

and \( i_y, i_z \) (GetKLs)

Yes

No

\[ y, z \text{ both in RAS2?} \]

Reset integral counter

\( \text{sgn} = \text{sgn}_y \)

\( f = f (I) \cdot \text{sgn-integral} \)

Loop over \( \mathbf{Z}_{\mathbf{RASSX}} \)

Loop over \( \mathbf{Z}_{\mathbf{RASSX}} \)

\( \text{Addr} \{ \mathbf{K}_{\mathbf{RASSX}} \} = \text{Addr} \{ \mathbf{L}_{\mathbf{RASSX}} \} \)

\( \text{LID}(1, \mathbf{K}_{\mathbf{RASSX}}) = i_y \)

\( \text{LID}(2, \mathbf{K}_{\mathbf{RASSX}}) = i_z \)

\( \text{XID}(\mathbf{K}_{\mathbf{RASSX}}) = \text{XID}(\mathbf{K}_{\mathbf{RASSX}}) + f \)

Loop: \( \mathbf{Z}_{\mathbf{RASSX}} \) offset

Reset integral counter

Loop: \( \mathbf{Z}_{\mathbf{RASSX}} \) sublist

\( \Rightarrow \mathbf{K}_{\mathbf{RASSX}}, \mathbf{L}_{\mathbf{RASSX}}, \text{sgn}_{\mathbf{RASSX}} \)

Increment integral counter

\( \Rightarrow \text{sgn} = \text{sgn}_y, \text{sgn}_{\mathbf{RASSX}} \)

\( f = f (I) \cdot \text{sgn-integral} \)

Loop over \( \mathbf{Z}_{\mathbf{RASSX}} \)

\( \text{Addr} \{ \mathbf{K}_{\mathbf{RASSX}} \} = \text{Addr} \{ \mathbf{L}_{\mathbf{RASSX}} \} \)

\( \text{LID}(1, \mathbf{K}_{\mathbf{RASSX}}) = i_y \)

\( \text{LID}(2, \mathbf{K}_{\mathbf{RASSX}}) = i_z \)

\( \text{XID}(\mathbf{K}_{\mathbf{RASSX}}) = \text{XID}(\mathbf{K}_{\mathbf{RASSX}}) + f \)

Return

Continued in Figure C.8b
Figure C.8b: Subroutine RUABC1 generates the elements of the intermediate matrix $M$. This routine operates the matrix generation for the excitation cases of $(2\rightarrow 2, 1\rightarrow 1)$, $(2\rightarrow 2, 1\rightarrow 2)$, and $(2\rightarrow 2, 2\rightarrow 2)$. Routines RUHWCS and RUABD1 do similar job.
Figure C.9a: Subroutine RUABC2 generates the elements of the intermediate matrix \( \mathbf{M} \). This routine operates the matrix generation for the excitation cases of \((1\rightarrow2,2\rightarrow2), (1\rightarrow3,2\rightarrow2), (2\rightarrow3,2\rightarrow2), \) and \((3\rightarrow3,2\rightarrow2)\). Routines RUHWS2 and RUABD2 do similar job.
PhD Thesis: Algorithm Development for Direct MCSCF Method

Figure C.9b: Subroutine RUABC2 generates the elements of the intermediate matrix \( M \). This routine operates the matrix generation for the excitation cases of (1\( \rightarrow \)2,2\( \rightarrow \)2), (1\( \rightarrow \)3,2\( \rightarrow \)2), (2\( \rightarrow \)3,2\( \rightarrow \)2), and (3\( \rightarrow \)3,2\( \rightarrow \)2). Routines RUHWS2 and RUABD2 do similar job.
Figure C.10: Subroutine RBiUp3 carries out the CI vector updating from the list generated from the $wx$ model index pair. Similar to the CASSCF matrix multiplication method, this matrix multiplication is carried out once per $wx$ pair.
Figure C.11: Subroutine calling sequence for density matrix algorithm.
Figure C.12: Subroutine calling sequence for diagonal Hamiltonian algorithm.
Figure C.13: Subroutine calling sequence for reduced Hamiltonian algorithm.
C. RASSCF Implementation Details

Figure C.14: Linda communication for density matrix algorithm.

Figure C.15: Linda communication for diagonal Hamiltonian algorithm.
Figure C.16: Linda communication for reduced Hamiltonian algorithm.
Bibliography


71. Pople JA, Brickstock A, *Phil. Mag.*, 44(1953), 697

292
Bibliography

96. Malmqvist PA, Rendell A, and Roos BO, J. Phys. Chem. 94(1990), 5477
104. Davidson ER, J. Comp. Phys., 17(1975), 87
Bibliography