Cooperating Sparse Simplex Algorithm for a Distributed Memory Multiprocessor[1]

I. Maros
Imperial College, London, UK,
i.maros@doc.ic.ac.uk,

G. Mitra
Brunel University, London, UK,
gautam.mitra@brunel.ac.uk

Departmental Technical Report DOC 98/12
Imperial College, London, UK.

December 1998

[1]This paper is based on an invited lecture of the authors at the XVI Mathematical Programming Symposium, Lausanne, Switzerland, August 1997.

Contents

Abstract

We undertake a computational analysis of the algorithmic components of the sparse simplex (SSX) method and consider the performance of alternative SSX solutions strategies. We then combine these alternative strategies within an asynchronous control structure of a parallel algorithm implemented on a distributed memory multiprocessor machine. Our experimental results not only show (limited) speedup and robust performance they also give new insight into the internal operation of the SSX and its adaptation on a parallel platform.

1 Introduction and Background

1.1 Computational solution of LPs

Over the last forty years considerable progress has been made in respect of theory and algorithms for the solution of large scale linear programs. Until 1984 exterior point or the so-called (pivotal) sparse simplex (SSX) dominated the computational scene. Karmarkar's projective method [17], after much controversy, was finally acknowledged both for its complexity results (low order polynomial bounded) and for its computational efficiency.

Considerable follow up research and widespread adoption of this approach have lead to the development of a family of algorithms which compute non-linear trajectory connecting analytic centers. As these algorithms search the constraint polyhedron internally these are generally known as interior point methods (IPMs). In spite of the success of IPMs, research into SSX continue to flourish side-by-side with IPM. This is mainly because

(i) broader application areas of LP, and

(ii) LP based other optimization problems, for instance, integer, non-linear, stochastic programs

continue to be discovered and adopted in main stream industrial and commercial applications. Many of these applications are either best processed by the SSX or the IPM or by a hybrid approach involving both these techniques. For a discussion of recent developments in these areas the readers are referred to Maros and Mitra, [22] and Gondzio and Terlaky [13], both appearing in the same volume edited by Beasley [2]. It is well known to researchers in the field of computational LP that three factors, namely, algorithmic developments, software techniques and hardware platforms contribute towards progressively better computational solution of these growing applications of LPs. In this paper, however, our main interest is to investigate how parallel computing can be harnessed to provide relatively faster (than a single processor) as well as robustly computed solutions of LPs using the SSX approach.

1.2 Parallel SSX algorithm

IPM can be adapted to parallel architectures with relative ease [18] and many successful implementations have been reported in the literature. In contrast, the reorganization of SSX computing steps for parallel computers have proven to be difficult. In a review article [24] we have discussed the issues which make it difficult to adapt SSX readily onto parallel machines. Following Amdahl's analysis of processing time [1], we break up an algorithm into fixed serial fraction s and parallelizable fraction p $(s + p = 1)$. We then find that for SSX the portion p can be very different from model to model and the relative portion is not large enough to offer a suitable payback in terms of speed up. Stunkel and Reed [28] and Chen et al [5] report parallel versions of simplex but their results do not take into account challenging industrial models. Forrest and Tomlin [9], using considerable ingenuity in data reorganization, speeded up the PRICE step (see Section 2) of the SSX. This was implemented in MPSX for IBM's family of vector machines and is one of the earliest and robust implementations of parallel SSX. In more recent times researchers at Edinburgh University, Hall and McKinnon, have continued to study parallel SSX and have provided interesting results for a limited range of problems [15]. These implementations are essentially based on shared memory or virtual shared memory multiple instruction multiple data (MIMD) computers. Bixby and Martin in a recent report [4] consider the scope of parallelizing dual SSX on distributed memory MIMD and shared memory MIMD computers. They report results based on their CPLEX solver: in summary the solutions slow down on distributed memory machines and show good speed up for a shared memory machine.

1.3 Choice of parallel platform

As research into parallel algorithms and software techniques for parallel architectures continue to grow it is important to analyze and establish the advantages which are to be gained by adopting these architectures. In general the main motivations for exploiting parallelism have been

(i) to speed up wall clock (elapsed) time for processing an application,

(ii) to scale up the size of problems which can be solved,

(iii) to compute the solutions in a reliable fashion,

for a given computational application. Latency (for message exchange) and communication speed play important roles in determining the performance of parallel computers. A number of investigators have found shared memory MIMD machines are good platforms for achieving speed up.

For a number of complex but entirely justifiable reasons set out here we have chosen to work with distributed memory MIMD machines. Moor's Law and the weight of empirical evidence show that single processor speed continues to improve by an order of magnitude every 12-18 months. Thus serial implementations of application software

(a) are by and large very stable, and

(b) set the most challenging performance standards for parallel algorithms.

Because single processor speeds continue to improve more development and maintenance effort are recurrently deployed for these "serial software systems" which are by default used for many challenging applications. This in turn makes these systems both mature in their features and robust in respect of bugs and fault tolerance. For parallel computing to really gain acceptance, it is also necessary to compare the performance of a parallel software system against the serial software run on the fastest yet readily available serial machine. The experience of last few decades have shown fastest serial processors released to the market outperform substantially the individual processors used in parallel machines. All these considerations lead to an implied imperative that we need to develop parallel software which is a natural extension of the serial version. This immediately rules out the use of single instruction multiple data (SIMD) machine. Virtual shared memory machines also require embedding nonstandard language constructs (such as pragma extensions of Power C, see [27]) whereas we wish to develop a system which can be readily ported.

We have therefore adopted a distributed memory multiprocessor as our hardware platform and the industry de-facto standard parallel virtual machine (PVM) as the message passing standard. Initial implementation has been on a Parsytec CC12 computer with 12 processors, 64 Mbyte RAM per processor node. The nodes are Power PC 604 processors running at 133 MHz. The software system can be also run on a cluster of PCs.

Our experimental results vindicate this choice of "cluster" MIMD architecture as we are able to show speed up and robust performance (see Section 5). Our aim is twofold. We want to create (i) a scalable blackbox software system which can be simply used as a robust LP optimization server and (ii) a tool to investigate the scope of enhancing the simplex method.

1.4 Outline of the paper

The rest of this paper is organised in the following way. In Section 2, we give computational results and profiling information for the SSX algorithm, analyse the results and discuss a few issues covering sparsity and parallel algorithms. Section 3 contains a discussion of important and well-established strategies used in SSX. The results of applying these strategies in a serial algorithm for a chosen collection of test problems are also set out in this section.

A description of our new cooperating SSX algorithm is given in Section 4 and computational results are presented and analysed in Section 5. The paper is concluded with a short discussion set out in Section 6.

2 Computational Analysis of SSX Algorithm

2.1 Structure of the SSX

2.1.1 LP problem statement

We consider the primal linear programming problem in the following general form:

$$\begin{aligned} &\text{minimize} && c^Tx, \\ &\text{subject to} && Ax = b, && (1) \\ & && l \le x \le u, && (2) \end{aligned}$$

where $A \in I\!R^{m\times n}$, c, x, l, $u \in I\!R^n$, and $b \in I\!R^m$. Some or all components of l and u can be $-\infty$ or $+\infty$, respectively.

A is assumed to contain a unit matrix I, $A = [I, \bar{A}]$, so that each row has a *logical variable* and, therefore, A is of full row rank.

Let B denote a basis of A; the matrix and the vectors are partitioned as $A = [B, N]$, $x = [x_B, x_N]$, $c = [c_B, c_N]$. We rewrite system (1 as $Bx_B + Nx_N = b$, or equivalently, $x_B = B^{-1}(b - Nx_N)$. Variables in N are either at zero or upper bound or some 'superbasic' value.

2.1.2 The primal simplex iteration

In order to analyse the structure of the algorithm, we need to identify the main algorithmic components of the computational version of the primal revised SSX method. The iteration cycle consists of the following steps.

Step 1. Compute **simplex multiplier** $\pi^T = v^TB^{-1}$ using a form vector v (different for Phase-1 and Phase-2).

Step 2. Check optimality (also known as **Choose Column** or PRICE): compute $d_j = c_j - \pi^Ta_j$ for $j \in N$. If all of them satisfy the optimality condition, stop. Otherwise, there is at least one variable that can enter the basis. Its subscript is denoted by k.

Step 3. **Transform** the vector of the improving candidate a_k: $\alpha_k = B^{-1}a_k$.

Step 4. **Choose row (pivot)**: determine the outgoing variable or bound swap (ratio test).

Step 5. **Update** solution, basis inverse; do reinversion (refactorization) if necessary.

The (sparse) simplex method as described above, is inherently a serial algorithm.

2.2 Profiling the primal SSX components

In order to decide which components of the SSX can benefit from parallel execution it is necessary to find out the relative time spent in the main algorithmic components. We have provided some preliminary information in earlier papers [22], [24]; Bixby and Martin also give comparable results [4]. In Table 1 we summarize and explain these components.

Table 1: Main algorithmic components of SSX

Component	Brief explanation
CHKFEZ	Checking feasibility of the current solution and, accordingly, constructing Phase-1 or Phase-2 from vector v. It appears in Step 1 of the SSX.
BTRAN	Backward transformation: operation $\pi^T = v^T B^{-1}$ leading to the price vector π in Step 1.
PRICE	Computation of the d_j reduced cost for a designated set of nonbasic variables as given in Step 2.
FTRAN	Forward transformation: updating the column(s) selected by PRICE. Step 3 on one or more vectors.
CHUZRO	Choosing the pivot row (determining the outgoing variable) by applying the ratio test. Step 4.
HUZKEP	"House-keeping" step after each iteration whereby basis or bound changes and other related information are updated. Step 5.
INVERT	Refactorization (reinversion) of the current basis to refresh the LU representation of B^{-1}. This appears in Step 5 and is carried out to limit the growth of nonzeros and restore computational accuracy.
OTHER	Remaining computational work not otherwise accounted above.

In Table 2 the statistics of the problems under consideration are displayed and in Table 3 we set out the profiling information of applying the default PRICE strategy. These strategies are well known (c.f., [22]) and are also explained briefly in section 3.1.

We observe that the proportion of time spent in different components can be extremely variable from problem instance to problem instance. This, in turn, makes it difficult to adopt a uniform parallelization strategy.

2.3 Sparsity and Parallelization

The sparse matrix methods both in terms of data representation, and in terms of reordering and processing algorithms are applied in the serial SSX algorithm. SSX requires frequent operations using packed and unpacked vectors simultaneously and in this respect it is a "tightly data coupled" algorithm. In general, it has been well known that PRICE step does take a good proportion of processing time hence Forrest and Tomlin parallelized this procedure in their implementation [9]. Fine grain parallelization of the other SSX steps is possible. For instance, Hall and McKinnon have parallelized both PRICE and INVERT [15]. Analyzing the profiles of different models set out in Table 3, it is easily seen that uniform speed-up is hard to achieve.

Adopting a fine grain parallel approach invariably limits the choice to shared (virtual) memory platforms only. On distributed memory computers a coarser grain parallelism is more meaningful than fine grain parallelization of SSX. Fine grain parallel steps on distributed memory machines can only lead to slow down due to latency and communication bottleneck. This is vindicated by the results presented in [4]. An analysis of the computational profile of the SSX, however, allows us to decide how to structure a coarse grain parallel algorithm which we discuss in the next section.

3 Sparse Primal Simplex Solution Strategies

SSX iterations do not uniquely define the solution path. The alternative parameter and control choices in several SSX steps allow us to create a number of alternative versions of the SSX. Most of the current

Problem	Number of rows	Number of columns	Number of nonzeros
80bau3b	2263	9799	29063
bnl2	2325	3489	16124
d2q06c	2172	5167	35674
d6cube	416	6184	43888
degen3	1504	1818	26230
fit2p	3001	13525	60784
greenbea	2393	5405	31499
greenbeb	2393	5405	31499
maros-r7	3137	9408	151120
pilot	1442	3652	43220
pilot87	2031	4883	73804
qap8	913	1632	8304
stocfor3	16676	15695	74004
truss	1001	8806	36642
woodw	1099	8405	37478

Table 2: Problem statistics of our test set. Models are from `netlib/lp/data`.

implementations are equipped with a variety of strategies which by and large enhance the reliability and efficiency of the solvers. The important parameter and control choices are summarized below.

(i) *main algorithm*: primal or dual method,

(ii) *pricing*: a large number of variants, including partial and sectional pricing, multiple pricing, different versions of normalized pricing,

(iii) *determining the outgoing variable*: several strategies for row choice which are different in Phase-1 and, Phase-2,

(iv) *anti-degeneracy strategies*,

(v) *inverse update*,

(vi) *presolve, scaling*.

Most of the industrial strength solvers provide a default setting for the above choices and, in many cases, a strategy is set and it is used throughout the solution. It is generally accepted that there is no single strategy which is the best for all problems. At different stages of the solution, different strategies would perform better. However, the investigation of this conjecture requires a demanding experimental simplex framework and very little results have been reported in the open literature.

Some 'smart' solvers can detect certain simple situations and change strategy for the rest of the solution (e.g., changing from *standard pricing* to *Devex*), others can notice the size of the available memory and change for a strategy that exploits if 'more than enough' memory is available, e.g., they create additional data structures like rowwise storage of A.

In this investigation, we have set out to find a good mix of strategies. We have found that the cooperating (parallel) version of SSX throws considerable light on the relative performance of individual as well as mixed strategies.

Problem	CHKFEZ	BTRAN	PRICE	FTRAN	CHUZRO	HUZKEP	INVERT	OTHERS
80bau3b	2.9	16.4	25.2	22.4	12.8	9.7	2.3	8.3
bnl2	3.1	16.9	16.0	28.2	18.6	8.1	2.5	6.6
d2q06c	0.3	20.1	14.1	31.8	15.4	5.3	6.7	6.3
d6cube	0.2	17.6	31.4	25.8	9.3	2.0	6.5	7.2
degen3	1.3	26.3	13.2	32.9	13.2	4.2	8.4	0.5
fit2p	0.1	17.8	17.3	44.4	14.6	1.7	1.1	3.0
greenbea	2.1	21.2	11.9	28.4	17.0	7.5	4.7	7.2
greenbeb	2.8	20.8	12.3	27.6	16.8	8.4	4.6	6.7
maros-r7	0.2	29.0	10.8	39.9	6.9	2.2	6.5	4.5
pilot	1.0	17.2	5.6	34.1	7.3	1.7	29.7	3.4
pilot87	0.3	14.4	3.1	30.2	3.5	1.0	40.3	7.2
qap8	0.4	19.1	3.7	39.2	12.7	1.7	19.4	5.8
stocfor3	2.5	21.1	8.1	34.1	19.1	7.7	2.5	4.9
truss	1.4	20.1	21.3	28.9	12.2	3.5	5.9	6.7
woodw	1.6	15.1	37.6	16.1	12.1	5.0	1.9	10.6

Table 3: Time percentages of algorithmic components of primal SSX using default parameter settings.

3.1 Column selection (PRICE) strategies

Column selection is often referred to as the PRICE step (see section 2.1.2). During pricing, d_j of the nonbasic variables is computed. If there is one that violates the optimality condition then it is a candidate to enter the basis.

PRICE is computationally an expensive step because of the many dot products that have to be calculated for some or all the columns of the A matrix. We set out below the important and most frequently adopted PRICE strategies for SSX.

1. *First improving candidate.* The first a_j, $j \in N$ with nonoptimal d_j is selected. This is the cheapest criterion (part of Bland's rule) but usually leads to very large number of iterations and, therefore, is not used in practice.

2. *Dantzig rule [6].* In this option all nonbasic variables (full pricing) and select one which violates optimality condition the most. This rule is quite expensive (dot products with all nonbasic variables) but, overall, is considerably better than the previous method. One disadvantage is that it is scale dependent.

3. *Partial pricing.* To alleviate the computational burden, only a part of the nonbasic variables is scanned and the best candidate from this part is selected [26]. In the next step, the next part is scanned, and so on. If the partition is chosen initially and kept fixed then this is known as static partial pricing. In contrast, the parts of A matrix may be dynamically redefined during the SSX iterations; this is known as dynamic partial pricing.

4. *Multiple pricing.* Some of the most profitable (in terms of the magnitude of d_j) candidates are selected during one scanning pass (*major iteration*, [26]). They are updated and a *suboptimization* is performed involving the current basis and the selected candidates using the criterion of greatest improvement. During these *minor iterations* the update of the selected candidates is inexpensive but some updated columns may become nonprofitable even before using them. Still, this technique is successful and is included in many solvers.

5. *Steepest edge.* The magnitude d_j shows the rate of change (scale dependent) but not how far the selected variable can go. Hence, it is not a good indicator of the achievable progress in the objective

function. Normalized pricings attempt to estimate the relative merit of the improving candidates by evaluating d_j in a fixed framework. This is achieved by scaling the computed d_j by a scale factor. Scale factors are updated for each basis change.

Steepest edge [10] is the most powerful normalized pricing algorithm but this is computationally the most expensive. It is a full pricing and does not adapt to multiple pricing. It can dramatically reduce the total number of iterations but the work per iteration can be so large that there can be no computational gain. However, in certain problem instances it is superior to any other method.

6. *Devex.* Introduced by Paula Harris [16], this was the first effective normalized pricing; today we see it as an approximation to steepest edge. It requires less work per iteration but still can reduce the number of iterations quite considerably. In a number of cases it results in the reduction of the overall computational effort and is considered a useful tool for the primal SSX. It is a full pricing and is not suitable for multiple pricing.

7. *Dynamic scaling [3].* This is an inexpensive approximation to the steepest edge pricing. Its advantage is its suitability for partial and multiple pricing. Its effectiveness is, however, questionable for many problem instances.

8. *Sectional pricing [14].* LP models often exhibit some structure which is characterized by the presence of sections. In this case pricing can operate more effectively by choosing candidates from different sections during a multiple pricing step. This increases the possibility that the selected candidates are not immediately related and most of them will remain candidates during the minor iterations.

9. *Partial sectional partial pricing [21].* In identified sections we perform partial pricing. But not in all of them at a time. The sections themselves are taken in a rotating order. The procedure is controlled by three parameters:

ns	Number of sections
ks	Number of sections scanned in one major pricing
kv	Number of improving vectors evaluated in a scanned section (uniform value for each section)

   Logical details of this procedure are described in [21] for network optimization that equally hold for the general simplex method.

10. *Composite pricing.* In Phase-1 the true objective function is usually not considered. This may entail that the first feasible solution is far from optimal. Some algorithms create a composite Phase-1 objective function (which is a weighted sum of the true objective function z and the sum of the infeasibilities w: $s = \lambda z + w$) and use it to find a better first feasible solution. In this approach the determination of the weight is, however, critical as well as and non-trivial. Some favorable results have been reported with a version of this technique which adaptively adjusts the weight [20]. It requires some extra work per major iteration.

3.2 Row selection strategies

Row selection determines the outgoing basic variable which by performing a *ratio test.* If the minimum ratio is unique the corresponding choice of the leaving variable is also unique. However, from numerical point of view it is always advantageous to have control on the magnitude of the pivot element. Therefore, by introducing suitable tolerances alternative non-unique choices are constructively created.

In Phase-1, one of the favorable "by-products" of the algorithm described in [19] is its flexibility in pivot row selection enabling increased numerical stability. In Phase-2, the seminal work of Harris [16] and a follow-up in [12] give a framework for variations in determining the pivot row.

3.3 Strategies considered

In order to identify a good mix of strategies, we first undertook a stand-alone analysis of performance of the selected eight strategies. The motivation was to pick relatively good strategies based on our experience. For our investigations we used FortMP system developed at Brunel University [7]. FortMP is equipped with a number of solution strategies. In our investigations we used the following.

Standard (STD): Dynamic cyclic partial multiple pricing with 4 candidates. This is used as the default strategy.

Devex (DVX): single, full pricing.

Full Dantzig Pricing (FDP): the whole matrix is scanned for the 'best' candidate (with largest $|d_j|$).

Adaptive Composite Pricing (ACP): True objective appears in Phase-1 with an adaptively adjusted relative weight λ (value used: 0.1), with multiple pricing on 4 candidates.

Dynamic Scaling (DSC) of d_j with multiple pricing on 4 candidates, in the framework of STD.

Partial Sectional Partial Pricing-1 (PSPP-1): as described above with $ns = 5,\ ks = 2,\ kv = 10$.

Partial Sectional Partial Pricing-2, (PSPP-2): as above, with $ns = 12,\ ks = 3,\ kv = 10$.

Partial Sectional Partial Pricing-2, (PSPP-3): as above, with $ns = 12,\ ks = 7,\ kv = 8$.

3.4 Performance analysis

The models introduced in section 2, Table 2, and used for SSX profiling were also used to evaluate the performance of the eight strategies described in section 3.3. A number of default yet standard algorithm controls such as (i) scaling, (ii) crash basis, (iii) Refactorization every 100 iterations and (iv) Forrest Tomlin sparse update were set. We specifically switched off presolve to increase solution time and thus make timing more accurate. The runs were carried out on a single processor of the Parsytec CC-12 computer and the solution times and the iteration numbers were recorded. These results are displayed in Table 4.

A brief analysis of the result leads to a number of observations and inferences set out below.

(i) There is no single solution strategy that outperforms all the others on all problems. This is another confirmation of the belief held by SSX code developers.

(ii) The ratios of the worst and best solution times vary between 5.55 and 1.64 and the average is 2.92. This can be interpreted as that the single best strategy (which is a priori not known at the time of processing a model and which is different from model to model) can be expected to be three times faster than the worst strategy. This represents quite a large relative time gain.

(iii) DSC strategy seems to be quite inferior to others in many but not all problem instances. We did not make an immediate decision to exclude it from further consideration. The inclusion of this strategy in the parallel algorithm led to interesting results.

(iv) For a number of problem instances PSPP variants provided good performance. The equisized column partitions are defined in a simple way, but do not make use of any structure information. We believe if partitioning is done, taking into account structure, this will lead to improvement of performance.

(v) We conclude that Devex is good for most problem instances. This is a common wisdom. However, when it performs poorly the time penalty is very high.

Problem	Strategy								W/B
	STD	DVX	FDP	ACP	DSC	PSPP-1	PSPP-2	PSPP-3	ratio
80bau3b	65.54	111.68	101.29	78.83	118.82	59.33	64.92	65.85	2.00
bnl2	70.75	52.14	67.63	85.54	77.38	64.51	67.18	72.51	1.64
d2q06c	384.64	217.40	468.65	506.22	458.11	407.79	382.43	409.94	2.33
d6cube	416.33	173.03	960.43	935.73	724.86	359.29	350.70	629.16	5.55
degen3	109.17	56.12	94.30	107.59	107.22	103.06	101.79	104.76	1.95
fit2p	744.10	490.34	635.15	549.46	1046.51	831.88	805.77	813.08	2.13
greenbea	133.31	135.81	136.07	133.69	395.75	108.55	107.26	111.69	3.69
greenbeb	88.23	95.39	110.93	126.98	349.43	96.66	90.94	88.03	3.97
maros-r7	147.14	262.05	171.15	228.87	369.11	134.87	147.78	152.56	2.75
pilot	303.85	245.39	339.91	336.66	935.66	319.33	312.28	304.85	3.82
pilot87	1229.91	674.82	1542.03	1712.55	1976.33	1435.60	1484.20	1510.67	2.93
qap8	100.65	73.78	98.67	96.60	121.86	88.07	100.92	100.60	1.65
stocfor3	1121.02	1070.09	1117.14	1062.85	2659.73	1155.79	1161.76	1196.45	2.50
truss	62.73	121.72	85.03	81.67	204.71	68.99	56.30	76.05	3.64
woodw	8.35	21.94	14.24	14.94	19.91	8.10	6.70	10.55	3.27
Sum	4985.72	3801.7	5942.62	6058.18	9565.39	5241.82	5240.93	5646.75	2.92
Average	332.38	253.45	396.17	403.88	637.69	349.45	349.40	376.45	2.92

Table 4: Solution times in seconds with different strategies. W/B is the worst to best ratio.

In Table 5 we have set out the strategies with their ranked order. This provides an alternative view of the strategies and it is easily seen that on average solution time ranking Devex is the outright winner and the standard strategy (STD) is a good second. These results provide a considerable justification of the default settings of our FortMP solver.

Ranking on the average also reveals that PSPP-2 is the surprise winner, followed by Devex and PSPP-1 in the tied second place, which is followed by STD. This somehow vindicates our earlier claim in (iv) that exploiting structure may improve the effectiveness of the PSPP strategy.

Altogether, the results are very interesting but inconclusive regarding the identification of the 'best' single strategy.

4 A Cooperating Framework of SSX

In this section we introduce the cooperating algorithmic framework for the SSX method. We first give a motivational overview then an outline of the algorithm which is followed up by a formal description and a pseudocode.

The distributed memory MIMD machine is organized in a simple tree topology with a single master and eight slave processors. The main idea is to reserve and dedicate each processor node to execute one of the eight chosen strategies. Thus each processor runs the same problem instance virtually independently of the other except when reaching a communication point (CP). From qualitative point of view, two types of CPs (*active* and *passive*) are defined for the slaves. When a slave reaches an active CP it initiates communication. Slaves are prepared to take messages at designated points only. They are the passive CPs. When such a CP is reached, the slave checks if there is a message for it. If not, normal program execution continues; if yes, it enters communication in accordance with the nature of the message.

Since nodes run independently and with different strategies, one of them reaches an active CP first. It reports to the master. Master collects information from all other nodes about their current status of progress. Master determines the best node (in terms of objective or feasibility) and requests the owner

Problem	Strategy							
	STD	DVX	FDP	ACP	DSC	PSPP-1	PSPP-2	PSPP-3
80bau3b	3	7	6	5	8	1	2	4
bnl2	5	1	3	8	7	2	4	6
d2q06c	3	1	7	8	6	4	2	5
d6cube	4	1	8	7	6	3	2	5
degen3	8	1	2	7	6	4	3	5
fit2p	4	1	3	2	8	7	5	6
greenbea	4	6	7	5	8	2	1	3
greenbeb	2	4	6	7	8	5	3	1
maros-r7	2	7	5	6	8	1	3	4
pilot	2	1	7	6	8	5	4	3
pilot87	2	1	6	7	8	3	4	5
qap8	6	1	4	3	8	2	7	5
stocfor3	4	2	3	1	8	5	6	7
truss	2	7	6	5	8	3	1	4
woodw	3	8	5	6	7	2	1	4
Sum	54	49	78	83	112	49	48	67
Average	3.6	3.27	5.20	5.53	7.47	3.27	3.20	4.47
Rank on average	4	2	6	7	8	2	1	5
Rank on sol. time	2	1	6	7	8	3	3	5

Table 5: Ranking of strategies by solution times. Identical scores are given same rank.

of the current best solution to communicate the associated basis to all other nodes. Nodes receive the basis (list of basic variables and variables at upper bound), factorize this basis and resume optimization with their designated strategy. This procedure goes on in this cooperative fashion until one of the nodes reports final solution (optimal, unbounded or infeasible). Subtleties are omitted in this verbal description.

It is clear that what we have defined is an algorithmic framework that can be filled with as many solution strategies as one wishes. Also, the evaluation criteria at CP can vary.

For a more formal description of the cooperating SSX algorithmic framework we introduce the following notations.

M	Master processor
P	Number of strategies
$S(i)$	Slave processor i, $i = 1, \ldots, P$
$w(i)$	Sum of infeasibilities computed by $S(i)$
$z(i)$	Value of the objective function computed by $S(i)$

Assuming that SSX is prepared for communication, we define actually three different types of communication points:

A: Before every minor iteration: a node is ready to receive and respond to messages from master.

B: Regular message sending point of slaves (currently: before every regular refactorization): initiate communication with master.

C: Termination: one node reaches a terminal solution (optimal, infeasible, unbounded) and is ready to report it to master.

The operation of the adaptive composite strategy (ACP) is quite different from the others. Its effort is aimed at finding a good first feasible solution. In Phase-2 it loses its importance and, therefore, can follow any other strategy. In our present version it reverts to the standard (STD) by default. Slave with the adaptive composite (ACP) strategy is denoted by $S(k)$.

All strategies use the expanding feasibility tolerance technique by Gill et al [12]. Therefore, additional information also has to be included in some messages.

The algorithm

```
1. Initialization Phase
   1.1 Load master node: Control program $\longrightarrow M$.
   For $i = 1, \ldots, P$
     1.2 Load node $i$: SSX solver $\longrightarrow S(i)$,
     1.3 Load the description of solution strategy of $S(i)$: $i$-th specs file $\longrightarrow S(i)$,
     1.4 Input the same LP problem (defined in specs) into $S(i)$,
     1.5 Start solving the (same) LP problem in $S(i)$ (minimization assumed).
2. Solution: Cooperative Processing Phase
   If **C** in not reached **do**
     Assume $S(p)$ reaches **B**.
       (a) $S(p)$ sends $w(p)$ and $z(p)$ to M.
       (b) M receives message.
       (c) M requests all slaves $i = 1, \ldots, P,\ i \neq p$ to send $w(i)$ and $z(i)$.
       (d) Slaves reach **A** or **B** and respond to M.
     M evaluates situation.
       (a) If $S(p)$ is in Phase-2, i.e., $w(p) = 0$, then
           i. it finds $i$ such that $z(i) = \min_j\{z(j) : j \neq k,\ w(j) = 0\}$,, i.e., node with best objective
              value.
           ii. it requests $S(i)$ to send basis (list of basic variables and variables at bound) and
              information about expanding feasibility tolerance (current iterate, value, increment),
           iii. it broadcasts this info to all slaves except $i$ and $k$ if latter is still in Phase-1,
           iv. slaves receive new basis and restart from this basis except $S(i)$ (and perhaps $S(k)$)
              which resumes where it was.
       (b) If $S(p)$ is in Phase-1, i.e., $w(p) \neq 0$, then
           i. if $p = k$ then do not do anything, else
           ii. denote $I = \{j : w(j) = 0\}$. If $I = \emptyset$ (all slaves are in Phase-1) do the same as in
              Phase-2 replacing $z(i)$ by $w(i)$. If $I \neq \emptyset$ then
           iii. find $i$ such that $z(i) = \min_j\{z(j) : j \neq k,\ j \in I\}$,
              and do the same as in Phase-2.
   **end do**
3. Termination Phase
   Assume $S(p)$ reaches **C**. It sends the current status to M.
```

(a) M orders all nodes to terminate computations and report statistics (Phase-1 and total iterations, Phase-1 time) to M.

(b) M declares: solution reached, reports CPU times of all processors, takes the winner's time as solution time.

(c) M also terminates.

If a node terminates with an error message, like numerical troubles, then (i) make a note what happened, (ii) take it as if it had reached **B** reporting an infinitely bad objective value and let it take the currently best basis to continue with.

Handling of other exceptional cases is not described here because they do not contribute to the essence of the general algorithmic framework.

5 Computational Investigation

In this section, we first present the performance of the mixed strategy (see section 4); our main measure of performance is the wall clock time for computing the solution. Next, we analyze the contribution of different strategies.

5.1 Results with mixed strategy

In Table 6 we summarize the performance of the mixed strategy algorithm for the chosen set of test problems. For convenience, we include the solution times of the standard (STD) strategy which is the default of FortMP. Next, we give the best serial algorithm and its solution time, followed by the ratio of the worst and best solution times of the eight serial strategies. In the next section of the table there are three columns. The first one is the solution time of the mixed strategy in seconds. The next column gives the ratio of the times of the standard strategy to the mixed one. The last column shows the ratio of the best serial and the mixed strategy in a similar sense.

The results reveal some interesting points. First of all, the mixed strategy is almost always better than the best serial strategy. Here, we emphasize again, that the best serial strategy is usually not known in advance. These timing improvements we see as a real achievement because the new algorithm is applied directly without any fine tuning. This supports our main claim that the performance of SSX can be improved by a considerable margin. We are convinced that by allowing more refinements and including more strategies the performance can be enhanced still further.

We justify this claim by xonsidering the data instance of problem d6cube for which the Devex algorithm was far superior to its nearest contender (DVX=173.03 sec, PSPP-2=350.70 sec). Yet the mixed strategy was able to beat this performance very substantially (M=115.57 sec).

The fact that we have not achieved uniform improvement just underlines the importance of further research in this direction. Although our experiments are comprehensive, we have not explored the full scope of improving SSX. Our approach with the strategy mix seems to be a promising avenue for improvement.

5.2 Analysis of mixed strategy

The purpose of this section is to investigate the individual contribution of different strategies to the overall composition of the best mixed strategy. In other words, we wish to establish how many times a specific strategy was locally (between two communication points, i.e., refactorizations) chosen by comparative evaluation and thus defining the best sequence which constituted the best mixed strategy.

Table 7 shows the composition of mixed strategy in terms of number of times a given strategy was used. This table, again, reveals information which is in some ways couter intuitive.

First, it can be seen that all strategies contributed to the mix (no column contains all zeros). Of course, if a problem is solved with few iterations (and refactorizations) the chance of using all strategies

Problem	STD	Best Serial			Mixed Strategy		
	sec	Strat.	sec	W/B	sec	STD/M	BS/M
80bau3b	65.54	PSPP-1	59.33	2.00	58.70	1.12	1.02
bnl2	70.75	DVX	52.14	1.64	40.09	1.76	1.30
d2q06c	384.64	DVX	217.40	2.33	193.76	1.99	1.12
d6cube	416.33	DVX	173.03	5.55	115.57	3.60	1.50
degen3	109.17	DVX	56.12	1.95	49.57	2.20	1.13
fit2p	744.10	DVX	490.34	2.13	449.38	1.66	1.09
greenbea	133.31	PSPP-2	107.26	3.69	72.65	1.83	1.48
greenbeb	88.23	PSPP-3	88.03	3.97	65.50	1.35	1.34
maros-r7	147.14	PSPP-2	134.87	2.75	134.79	1.09	1.00
pilot	303.85	DVX	245.39	3.82	188.50	1.61	1.30
pilot87	1229.91	DVX	674.82	2.93	650.76	1.89	1.04
qap8	100.65	DVX	73.78	1.65	58.87	1.71	1.25
stocfor3	1121.02	ACP	1062.85	2.50	806.95	1.39	1.32
truss	62.73	PSPP-2	56.30	3.64	59.63	1.05	0.94
woodw	8.35	PSPP-2	6.70	3.27	6.91	1.21	0.97

Table 6: Comparison of best serial and mixed strategies. W/B = ratio of worst to best serial, STD/M = ratio of the standard serial to mixed strategy, BS/M = best serial to mixed.

is smaller. At the same time, there are really very few zeros in this table. Also to be noted that ACP can contribute to the mix in Phase-1 only, therefore the corresponding column is more likely to contain zeros but, in fact, it contains only three.

Second, we point out that the individually rather poorly performing dynamic scaling (DSC) technique took part in defining the best mix for all but two problems. Even in the case of `d6cube` (where Devex is by far the best serial strategy) contributed 11 times to the mix. In case of the largest (in terms of size) problem in the test set, `stocfor3`, DSC became the dominant technique, well ahead of Devex.

Third, the real poor performer is the full Dantzig pricing with seven zeros in its column. This further confirms that FDP, without normalizing the reduced costs (d_j), is unlikely to be efficient.

Fourth, while standard pricing (STD) is very good individually, its importance diminishes when used in potential combination with others.

In an earlier paper on the topic of parallel multiple tree search strategies [25] our computational experience showed that information sharing of multiple strategies leads to a "mixed strategy" which performs well and often better than the best individual strategy. We are encouraged to discover that in the case of SSX comparable performance behaviors take place.

Finally, we wish to point out the robustness property of our cooperating SSX approach. It is well known that the ill-behaved problems have difficult and easier stages during solution. If a difficult stage is bypassed in some way solution can proceed successfully. In the outlined algorithm if any of the nodes reports numerical difficulties or failure the others still can provide bases suitable for further computations.

6 Conclusions

We have created a cooperative parallel version of the sparse simplex method. It is a general framework that can be used to investigate the effects of changing strategies during solution. We presented the results of our first investigations with a specific selection of strategies. We identified that a considerable improvement in efficiency and reliability can be achieved in this way. On the basis of these findings, it can be expected that the provision of a larger richness of strategies could further enhance the efficiency

Problem	Strategy							
	STD	DVX	FDP	ACP	DSC	PSPP-1	PSPP-2	PSPP-3
80bau3b	15	7	0	6	7	4	16	4
bnl2	4	8	2	0	10	4	9	9
d2q06c	9	84	0	5	5	15	12	8
d6cube	38	140	6	13	11	102	107	14
degen3	3	24	5	3	8	7	3	7
fit2p	8	52	0	2	1	2	14	16
greenbea	11	12	3	3	9	5	22	14
greenbeb	5	21	2	3	8	5	16	4
maros-r7	15	6	0	10	2	3	6	5
pilot	9	27	2	2	17	10	10	4
pilot87	7	58	0	0	15	12	4	4
qap8	3	37	2	3	0	5	26	4
stocfor3	0	30	0	0	47	10	12	2
truss	7	15	3	5	3	21	22	3
woodw	3	0	0	3	0	2	6	3

Table 7: Break-down of mixed strategy. Number of times a given strategy was used.

of the solution. To this end, some more research is justified so that this statement can be verified in quantitative terms.

There are several issues to be addressed in the future. First, some more algorithmic variants should be included in the analysis, and, second, we have to identify the situations when a switch is needed. If the investigations are successful, the serial version of the SSX can ultimately be enhanced such that the scenario of algorithmic changes could be selected automatically for each problem. In the light of our findings, the expected rate of improvement seems to be promising.

Acknowledgements

We are grateful to both M. Hajian and E.F.D. Ellison who set up the profiling and cooperative processing PVM driver program for FortMP. We are particularly indebted to E.F.D. Ellison who spent considerable effort in interpreting the cooperative SSX algorithm on to the PVM based message passing distributed computing environment.

References

[1] Amdahl, G. M., Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, AFIPS Conference Proceedings, 20, 1967, p. 483–485.

[2] Beasley, J., (ed.), *Advances in Linear and Integer Programming*, Oxford University Press, Oxford, 1996.

[3] Benichou, M., Gautier, Hentges, Ribiere, The efficient solution of large-scale linear programming problems, *Mathematical Programming*, (13) 1977.

[4] Bixby, R.E., Martin, A., Parallelizing the Dual Simplex Method, Konrad Zuse Zentrum, Berlin, Preprint SC-95-45, December 1995.

[5] Chen, G.H., Ho, H.F., Lin, H.F. and Sheu, J.P., Data Mapping of Linear Programming on Fixed Size Hypercubes, *Parallel Computing*, 13, 1990, p. 235–243.

[6] Dantzig, G.B., *Linear Programming and Extensions*, Princeton University Press, Princeton, N.J., 1963.

[7] Ellison E.F.D., Hajian M., Levkovitz R., Maros, I., Mitra G. and Sayers D., *FortMP Manual*, Department of Mathematics and Statistics, Brunel University, London, and Numerical Algorithms Group NAG, Oxford, May 1994, revised for version 1.04 in June 1995.

[8] Forrest, J.J.H. and Tomlin, J.A., Updated Triangular Factors of the Basis to Maintain Sparsity in the Product Form of the Simplex Method, *Mathematical Programming*, 2, 1972, p. 263–278.

[9] Forrest, J.J.H. and Tomlin, J.A., Vector Processing in Simplex and Interior Point Methods for Linear Programming, *Annals of Operations Research*, 22, 1990, p. 71–100.

[10] Forrest, J.J., Goldfarb, D., Steepest edge simplex algorithms for linear programming *Mathematical Programming*, 57, 1992, No. 3., p. 341–374.

[11] Gay, D.M., "Electronic mail distribution of linear programming test problems", *Mathematical Programming Society COAL Newsletter* 13 (1985), p. 10–12.

[12] Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H., "A Practical Anti–Cycling Procedure for Linearly Constrained Optimization", *Mathematical Programming*, 45, 1989, p. 437–474.

[13] Gondzio, J., Terlaky, T., A computational view of interior point methods, Chapter 3. in Beasley, J. (ed), *Advances in Linear and Integer Programming*, Oxford University Press, Oxford, 1996, p. 103-144.

[14] Greenberg, H.J., Pivot Selection Tactics, in Greenberg H.J., (ed.) *Design and Implementation of Optimization Software*, Sijthoff and Nordhoff, 1978, p. 143–174.

[15] Hall, J.A.J., McKinnon, K., An Asynchronous Parallel Revised Simplex Algorithm, to appear in Applied Mathematical Programming and Modelling, APMOD III, Maros, I. and Mitra, G. (eds.), *Annals of Operation Research*, Baltzer, 1998.

[16] Harris, P.M.J., Pivot Selection Method of the Devex LP Code, *Mathematical Programming*, 5, 1973, p. 1–28.

[17] Karmarkar, N., A New Polynomial Time Algorithm for Linear Programming. *Combinatorica*, 4, 1984, p. 373–394.

[18] Lustig, I.J., Rothberg, E., Gigaflops in linear programming, *Operations Research Letters*, 18 (4) 1996, p. 157–165.

[19] Maros, I., A general Phase–I method in linear programming, *European Journal of Operational Research*, 1986, (23) p. 64–77.

[20] Maros, I., A multicriteria decision problem within the simplex method, in Mitra G. (ed.) *Mathematical Models for Decision Support*, NATO ASI Series, Springer Verlag, 1988, p. 263–272.

[21] Maros, I., A structure exploiting pricing procedure for network linear programming, RUTCOR Research Report, RRR #18–91, Rutgers University, NJ, USA, May, 1991.

[22] Maros, I., Mitra, G., Simplex Algorithms, Chapter 1 in Beasley J. (ed.) *Advances in Linear and Integer Programming*, Oxford University Press, 1996, p. 1–46.

[23] Maros, I., Mitra, G., Strategies for creating advanced bases for large-scale linear programming problems, to appear in *INFORMS Journal on Computing*, Spring 1998.

[24] Mitra, G., Levkovitz, R., Solution of Large Scale Linear Programs: A Review of Hardware, Software and Algorithmic Issues, in *Optimisation in Industrial Environments*, T. Ciriani and R. Leachman, (eds.), John Wiley, 1993, p. 139–171.

[25] Mitra, G., Hajian, M., Hai, I., A Distributed Processing Algorithm for Solving Integer Programs Using a Cluster of Workstations, *Parallel Computing*, Vol. 23, 1997, p. 733–753.

[26] Orchard-Hays, W., *Advanced Linear-Programming Computing Techniques*, McGraw-Hill, 1968.

[27] *Power C User's Guide*, Silicon Graphics, Inc.

[28] Stunkel, C.B. and Reed, D.A., Hypercube Implementation of the Simplex Algorithm, *Proc. Hypercube Concurrent Computers and Applications*, 1988.

[29] Suhl, U.H., Suhl, L.M., Computing Sparse LU Factorizations for Large-Scale Linear Programming Bases, *ORSA Journal on Computing*, Vol. 2, No. 4, Fall, 1990, p. 325–335.