

Imperial College London
Department of Computing

Enabling Real-time Adaptive Radiotherapy

Marco Barbone

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the Imperial College London and
the Diploma of Imperial College, April 2024

Statement of Originality

I, Marco Barbone of Imperial College London, hereby confirm that this thesis is a presentation of my independent research. It does not include content that has been presented for a degree at any other institution. In instances where contributions from others are incorporated, every effort is made to explicitly acknowledge this in the references and highlight collaborative research.

Copyright

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Abstract

The success of radiotherapy relies on precisely targeting cancerous cells while minimising harm to healthy tissue. Real-time adaptive radiotherapy addresses the challenge of rapidly changing cancer positions, dynamically adjusting the radiation beam during treatment with the help of MR-linacs—integrating Magnetic Resonance Imaging (MRI) scanners with linear accelerators. However, this approach faces significant computing challenges. The process involves converting raw scanner signals into MRI images, transforming them into a patient model, reconstructing the delivered dose, and optimising treatment parameters. These computationally intensive steps must be executed quickly to ensure the effectiveness of real-time adaptive radiotherapy.

The first contribution of the thesis introduces a novel parallelisation scheme tailored for Four Dimensional Magnetic Resonance Imaging (4D-MRI) data, effectively minimising multithreading overhead and concealing data transfer latency. This results in an efficient architecture for algorithms based on the golden angle stack-of-stars MRI reconstruction. One such algorithm has been implemented and accelerated using Graphics Processing Units (GPUs) to facilitate clinical applications.

The second contribution aims to provide a methodology for predicting the performance of accelerators in the context of Monte Carlo (MC) simulations. The methodology is derived from GPU and Field Programmable Gate Array (FPGA) computational capabilities through benchmarks. It is then validated with Coulomb Scattering, assessing the accuracy of the forecasts. The results demonstrate that the model’s forecasted speedup assists in deciding the optimal accelerator, if any, to achieve the best performance.

The third contribution of this thesis is Optimised Dose Planning Method (ODPM), an MC simulation for radiotherapy dose reconstruction designed to meet real-time adaptive radiotherapy time requirements. The use of accelerators for ODPM is analysed using performance modelling. The forecasted speedup is then used to design an architecture capable of meeting real-time adaptive radiotherapy time requirements. This contribution also introduces multiple mathematical, algorithmic, and architectural optimisations applicable to MC simulations and integrates them into ODPM.

Acknowledgements

I would like to express my gratitude to my supervisors, Prof. Wayne Luk and Prof. Uwe Oelfke, for their guidance, support, and encouragement throughout this research journey. Prof. Luk provided invaluable assistance and always believed in me, shaping me into a better person first and a better researcher. His advice on research methodologies, collaboration, and academic writing has been instrumental in my growth. Prof. Uwe Oelfke made all of this possible, this project is a direct result of his vision and his ability of integrating people of different background and collaborate towards a common goal. I am indebted to Dr. James Bedford and Mr. Rahul Nilawar for their introduction to radiotherapy, explanation of algorithms, and providing validation data.

I was fortunate for the opportunity to work with Prof. Georgi Gaydadjiev. He was a source of constant help and support. Prof. Gaydadjiev not only established the unique opportunity for me to merge my research at Imperial College with my work at Maxeler but was a constant source of encouragement. He always helped me facing any problem, giving advice, providing feedback on my papers and connecting me with the right people who provided invaluable help over the years.

I will always be thankful to Prof. Alex Tapper, he always believed in the research and provided prompt help and support. His help was invaluable on overcoming all the challenges that arose and for the success of most of the paper published. He was the bridge between me and Dr. Alex Howard and Dr. Mihaly Novak who were indispensable for this research. Mihaly Novak, implemented the physics from scratch, solving the various equations providing me a solid base for this project. He also advised on the potential challenges and his experience helped me avoiding many dead ends. Alex Howard was a great support, his help in debugging the code, pointing out various issues, mapping the physics to possible implementations validating the simulation was paramount to the success of this project. I would also like to extend my gratitude to George and Konstantin Saviddy for their invaluable contribution in providing the MixMax random number generator, which played a crucial role in the Monte Carlo simulations.

I would also like to thank Dr. Andreas Wetscherek and Bastien Lecour. Andreas Wetscherek made the work on MRI possible. Without him it would have been impossible to tackle this computational challenge. He provided the necessary background knowledge, the algorithm,

the data and the validation necessary for this research endeavour. Bastien Lecour greatly contributed to the image reconstruction by adding more features to the implementation, support for multiple data preparation strategies, and executed the validation on multiple datasets to obtain the final results.

I was fortunate for the possibility to collaborate with Maxeler Technologies, their assistance was invaluable. In particular, I would like to extend my gratitude to Dr. Bas Kwaadgras, he was a colleague, a mentor and a friend. He taught me a lot on FPGA programming and performance modelling and his help was crucial for the paper on table based methods. I would also like to thank Will Wigor, my mentor at Maxeler, for introducing me to the company and teaching me the the Maxeler toolchain. I wish to give Nils Voss a special thank as his experience helped me balance my collaboration with Maxeler with my research at Imperial College and his help and advice was necessary when problems apperad to be unsolvable. I would also like to thank Maxeler's CEO and CTO Prof. Oskar Mencer and Maxeler VP Tobias Becker for the invaluable advice and discussions.

Finally, I would like to thanks all the people I met at Imperial College. Dr. Garrick Wilson and Dr. Amani El Kholy were supportive, helping with all the administrative issues assuring that the program requirements were always met. Also, my office family was always there for me: Kang, Jiuxi, Tim, Nadeen, Ng, Vincent, Bowen, you were always available to provide any help and I am glad you were part of this journey.

Last but not least, my family. I cannot quantify the help, love and support. They always believed in me and encouraged to overcome any difficulties.

Contents

Statement of Originality	1
Copyright	3
Abstract	5
Acknowledgements	7
1 Introduction	27
1.1 Challenges and Objectives	27
1.1.1 First Challenge and Objective	30
1.1.2 Second Challenge and Objective	31
1.1.3 Third Challenge and Objective	32
1.2 Research Contributions	32
1.2.1 Fast MRI Reconstruction	33
1.2.2 Performance Modelling of Monte Carlo Simulations	34
1.2.3 Fast Dose Delivery Reconstruction	35
1.3 Selected Publications	36

2	Background and Related Work	39
2.1	Introduction	39
2.1.1	Adaptive Radiotherapy	39
2.1.2	Magnetic Resonance Imaging (MRI) processing	42
2.1.3	MR-guided online Adaptive Radiotherapy (MRgoART)	44
2.1.4	Real-time dose reconstruction	45
2.1.5	XD-GRASP algorithm	51
2.1.6	Structural Similarity Index (SSIM)	53
2.2	Kolmogorov–Smirnov Test	54
2.3	Random Number Generation	55
2.4	Central Processing Unit (CPU) architecture	57
2.5	Field Programmable Gate Array (FPGA)	57
2.5.1	Configurable Logic Blocks	59
2.5.2	Input/Output	59
2.5.3	Block RAM	59
2.5.4	Digital Signal Processor	60
2.5.5	FPGA programming	60
2.6	Graphics Processing Unit (GPU)	62
2.6.1	Single Instruction Multiple Data (SIMD)	62
2.6.2	Single Instruction Multiple Threads (SIMT)	63
2.6.3	GPU architecture	63

2.6.4	GPU programming	64
2.7	Related Work	65
2.7.1	Magnetic Resonance Imaging	65
2.7.2	Performance modelling	66
2.7.3	Monte Carlo frameworks	68
2.8	Summary	70
3	Magnetic Resonance Imaging	71
3.1	Introduction	71
3.2	Architecture	73
3.2.1	Data acquisition and preparation	74
3.2.2	Data Preparation Strategies	75
3.2.3	Parallelism	76
3.2.4	Load Balancing	79
3.2.5	Non Uniform Fast Fourier Transform (NUFFT)	80
3.2.6	Oversampling Factor	82
3.2.7	Amdahl's law, Scalability and Limitations	82
3.2.8	Approximate Implementation	85
3.3	Experimental Results	86
3.3.1	System Configuration	87
3.3.2	Data acquisition	87
3.3.3	Test configuration	88

3.3.4	Validation	89
3.3.5	Reconstruction experiments	90
3.3.6	Reconstructed images and validation	90
3.3.7	Measured speed-ups	91
3.3.8	Scaling	94
3.4	Discussion	96
3.5	Summary	98
4	Accelerating Monte Carlo Simulations	100
4.1	Introduction	100
4.2	Methodology for acceleration	103
4.3	Application analysis	106
4.3.1	Derivation	109
4.4	Software Model (FPGA)	114
4.5	Operations count	116
4.6	Performance model	117
4.6.1	Predicting I/O Bandwidth Usage	118
4.6.2	Memory transfers	119
4.6.3	FPGA performance model	120
4.7	GPU performance model	122
4.7.1	Total time	124
4.8	Example and Evaluation	125

4.8.1	Coulomb Scattering	125
4.8.2	Application analysis	126
4.8.3	Data transfers	127
4.8.4	FPGA performance model	129
4.8.5	GPU performance model	131
4.8.6	FPGA architecture	132
4.8.7	GPU architecture	134
4.8.8	Methodology forecasts	135
4.8.9	Validation	136
4.8.10	FPGA Results	137
4.8.11	GPU Results	138
4.8.12	Comparison between platforms	138
4.9	Discussion	139
4.10	Summary	140
5	Dose Delivery Reconstruction	141
5.1	Introduction	141
5.2	DPM acceleration analysis	143
5.2.1	Application analysis	144
5.2.2	FPGA performance model	146
5.2.3	GPU performance model	148
5.2.4	Comparison between platforms	151

5.3	Implementation and optimisations	152
5.3.1	Branch mispredictions	153
5.3.2	Cache misses	158
5.3.3	Elementary functions	161
5.3.4	Random Number Generator (RNG)	162
5.3.5	Overall impact	162
5.4	Experimental results	163
5.4.1	System Configuration	164
5.4.2	Test Configuration	164
5.4.3	Validation	164
5.4.4	Performance	167
5.4.5	Scaling	168
5.5	Discussion	169
5.6	Summary	171
6	Conclusion	172
6.1	Summary of Thesis Achievements	172
6.2	Future Work	174
6.2.1	Real-time MRI reconstruction	174
6.2.2	Extension to performance modelling	175
6.2.3	Extension to ODPM	175
6.3	Final thoughts	176

List of Acronyms

2D Two Dimensional.

3D Three Dimensional.

4D Four Dimensional.

4D-MRI Four Dimensional Magnetic Resonance Imaging.

5D Five Dimensional.

ADERS Angle-dependent Correction of the Respiratory Signal.

AI Artificial Intelligence.

API Application Programming Interface.

ART Adaptive Radiotherapy.

ASIC Application Specific Integrated Circuit.

CG-SENSE Sensitivity Encoding solved with a Conjugate Gradient Algorithm.

CISC Complex Instruction Set Computer.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

CT Computed Tomography.

CUDA Compute Unified Device Architecture.

DMA Direct Memory Access.

DPM Dose Planning Method.

DRAM Dynamic Random-Access Memory.

FFT Fast Fourier Transform.

FPGA Field Programmable Gate Array.

GPU Graphics Processing Unit.

HBM High Bandwidth Memory.

HLS High Level Synthesis.

HPC High Performance Computing.

ISA Instruction Set Architecture.

KW Kernel Width.

MC Monte Carlo.

MR Magnetic Resonance.

MRgoART MR-guided online Adaptive Radiotherapy.

MRI Magnetic Resonance Imaging.

MRSIGMA Magnetic Resonance SIGnature MAtching.

MT Mersenne Twister.

NN Neural Network.

NUFFT Non-Uniform Fast Fourier Transform.

OAR Organ at Risk.

ODPM Optimised Dose Planning Method.

ORB Overlapping Respiratory Bins.

PCA Principal Component Analysis.

PCIe Peripheral Component Interconnect Express.

PRNG Pseudo Random Number Generator.

PTX Parallel Thread Execution.

RAM Random-Access Memory.

RNG Random Number Generator.

SFU Special Function Units.

SIMD Single Instruction Multiple Data.

SIMT Single Instruction Multiple Threads.

SM Streaming Multiprocessor.

SPIDERM Single projection driven real-time multi-contrast.

SPIRiT Iterative Self-Consistent Parallel Imaging Reconstruction.

SSIM Structural Similarity Index.

STD Standard deviation.

SW Sector Width.

TDP Thermal Design Power.

TPU Tensor Processing Unit.

TrACR Trajectory Auto-Corrected image reconstruction.

TRNG True Random Number Generator.

VHDL VHSIC Hardware Description Language.

XD-GRASP Extra-Dimensional Golden-Angle Radial Sparse Parallel.

List of Tables

2.1	The MIXMAX parameters. $\tau_0 < t < \tau$. The iteration time t is normalised to 1. The MIXMAX is a <i>genuine 61 bit generator</i> on Galois field $\text{GF}[p]$, with Mersenne prime number $p = 2^{61} - 1$	57
3.1	The NUFFT benchmarks ordered by speed-up.	86
3.2	Comparison with state-of-the-art implementations	96
4.1	Design space exploration.	132
4.2	Comparison between multiple platforms. Metrics are Time, speedup and cost-normalised speedup.	139
4.3	Comparison with state-of-the-art modelling methodology	140
5.1	Comparison between Dose Reconstruction Methods	170

List of Figures

1.1	Real-time adaptive radiotherapy diagram. The yellow boxes correspond to the machine, the red boxes correspond to computational expensive algorithms that need acceleration considered in this thesis and the white ones are the parameters and the state of the system. The data transferred from the image device consists of raw MRI signal in k-space. The treatment machine transfers the current beam parameters to the simulation.	28
2.1	From [HHO ⁺ 18], timescales for adaptation and adaptive radiotherapy solutions implemented. Adaptive Radiotherapy (ART) stands for adaptive radiotherapy, Organ at Risk (OAR) for organ at risk.	40
2.2	Adapted from [KPB19], this image shows the increased accuracy of real-time adaptive radiotherapy (b) compared to traditional image-guided radiotherapy (c).	41
2.3	General structure of the simulations.	47
2.4	Adapted from [SV13], this image shows the difference between the real and the simulated electron path	48
2.5	Adapted from [SV13], example of a particle history.	49
2.6	Modern CPU architecture	58
2.7	FPGA programming overview	61

3.1	The input and output of the XD-GRASP algorithm.	77
3.2	Extra-Dimensional Golden-Angle Radial Sparse Parallel (XD-GRASP) architecture timing diagram. Dimensions are not representative of real-time duration. .	78
3.3	Slices reconstructed at both the end-inspiration and end-respiration phases exhibit distinct respiratory states. The variations in respiratory phases are discernible through the liver's dome size in both images. Despite noticeable streaking artefacts stemming from the arms at the field-of-view periphery, they do not compromise the exceptional clarity in visualising the internal anatomy.	90
3.4	The end-inspiratory phase of the 4D-MRI obtained under abdominal compression was reconstructed using various data preparation strategies from the identical acquisition. Fine details within non-moving structures exhibit enhanced sharpness in (b), (c), and (d) in contrast to the baseline.	92
3.5	The timing and speed-up results for various data preparation strategies utilising different implementations. Both the data preparation (a) and overall reconstruction times (b) increased notably with the Overlapping Respiratory Bins (ORB) and combined strategies. Across all cases, the GPU implementation showcased more than a 17-fold improvement compared to the MATLAB reference implementation (c), achieving image reconstruction times of 60s (baseline, Angle-dependent Correction of the Respiratory Signal (ADERS)) and 77s (ORB, combined). Notably, the MATLAB implementation employing the FINUFFT bindings exhibited slower performance than the original version for ORB and combined strategies, with speed-ups falling below 1 (dashed line).	93

3.6	The straight line represents the theoretical speed-up. Analysis of the CPU scaling experiment reveals suboptimal scaling on both testing systems, with the system featuring faster Random-Access Memory (RAM) access demonstrating improved scaling. In the GPU scenario, multiple GPUs reduce CPU idle time, leading to increased CPU utilization and super-linear hyperscaling, with actual speed-up surpassing theoretical limits.	95
4.1	Proposed accelerator architecture development workflow	103
4.2	Application analysis deciding process. Arrows indicate mandatory requirements, the dashed arrow indicates an optional requirement.	107
4.3	Clock cycles required to execute: $if(\mathcal{R} < P) \text{ sum} += \mathcal{R};$	110
4.4	Read latency of the various memories in the system	110
4.5	CUDA cores per GPU generation. Original data public domain from NVIDIA. .	111
4.6	Increase in execution time due to branching on GPU	112
4.7	Read latency of the various GPU memories	112
4.8	Peripheral Component Interconnect Express (PCIe) 4.0x16 bandwidth on an NVIDIA GeForce RTX 3090 Ti	119
4.9	GDDR Bandwidth over data size.	119
4.10	FPGA architecture schematic displaying the parallel FPGA instances and the CPU based aggregation.	133
4.11	Illustration of the while loop optimisation implemented.	134
4.12	Transverse distribution.	136
4.13	Longitudinal distribution.	136

5.1	Memory storage strategy for Dose Planning Method (DPM). The geometry is divided into subgeometries. While one section is computed another can be loaded in memory.	159
5.2	Memory storage strategy for DPM. Each subgeometry is paired with a particle queue. When simulating one subgeometry, particles that escape to another subgeometry are added to the corresponding queue and simulated when that specific subgeometry is loaded in memory.	159
5.3	This figure shows the dose depth distribution computed in a water phantom, the number of histories simulated is 10^6 . The reference results are in blue, obtained using Geant4 while in orange are the results obtained with ODPM. On the left a 6MeV Electron beam is used, and on the right, a 6MeV photon beam is used. The Figure shows perfect agreement between the simulations	165
5.4	This figure shows the dose depth distribution computed in a water, bone, titanium, and water phantom. The number of histories simulated is 10^6 . The reference results are in blue, obtained using Geant4 while in orange are the results obtained with ODPM. On the left a 15MeV Electron beam is used, and on the right, a 15MeV photon beam is used. The Figure shows good agreement between the simulations	166
5.5	Simulation speedup compared to Geant4.	166
5.6	ODPM scaling.	167
5.7	ODPM scaling.	168

Chapter 1

Introduction

1.1 Challenges and Objectives

The effectiveness of radiotherapy relies on the precision with which cancerous cells are targeted while minimising damage to healthy cells. A significant challenge in traditional radiotherapy lies in the rapid obsolescence of information, particularly as cancerous tissues may change positions - such as lung cancer changing due to respiration. To address this issue and accurately target moving cancers, it is imperative to dynamically adapt the radiation beam during treatment, accommodating changes in cancer biology and patient anatomy [KNO⁺18, KPB19, HHO⁺18]. This innovative approach, known as real-time adaptive radiotherapy, emerged with the advent of MR-linacs - a technology that integrates a Magnetic Resonance Imaging (MRI) scanner with a linear accelerator [LRR⁺08, FMR⁺09, KBC14, MLC⁺16].

The current state-of-the-art radiotherapy treatment that leverages the MR-linac is called online adaptive radiotherapy. Differently, from real-time adaptive radiotherapy, this treatment adapts the radiotherapy plan before each radiotherapy session [BSM⁺19]. This differs from the architecture proposed in figure 1.1 as this proposed architecture performs the treatment adaptation not only before each radiotherapy session but in real-time, while radiation is delivered to the patient.

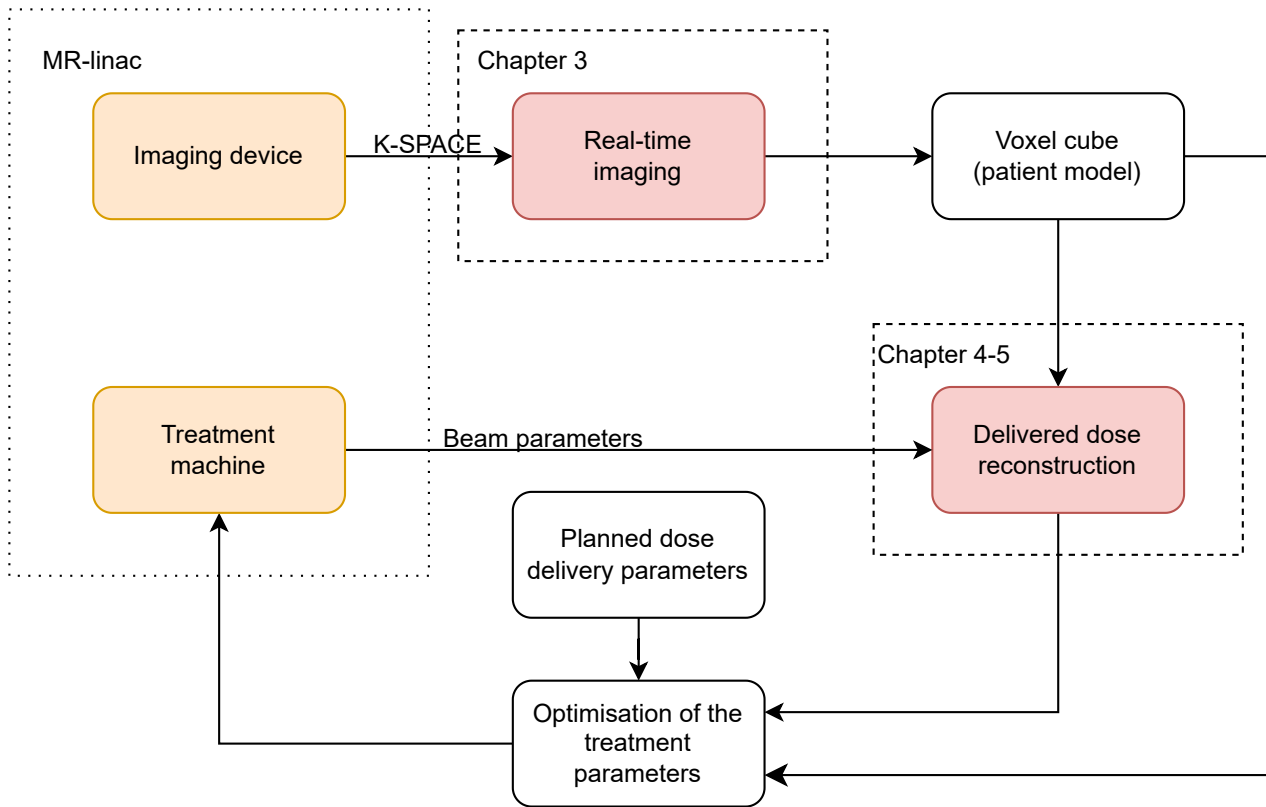


Figure 1.1: Real-time adaptive radiotherapy diagram. The yellow boxes correspond to the machine, the red boxes correspond to computational expensive algorithms that need acceleration considered in this thesis and the white ones are the parameters and the state of the system. The data transferred from the image device consists of raw MRI signal in k-space. The treatment machine transfers the current beam parameters to the simulation.

However, real-time adaptive radiotherapy suffers from significant computing challenges. The initial steps involve converting the raw scanner signal into MRI images. Subsequently, these images need to be transformed into a patient model, and the delivered dose must be reconstructed. Finally, the treatment parameters are optimised and the dose is administered. These steps are computationally intensive and must be performed as quickly as possible. Figure 1.1 shows a possible system and its main components. It is worth pointing out that this figure does not represent any real system but it offers a possibility based on [HHO⁺18, KPB19]. The yellow boxes model the medical machine needed for the treatment, the red boxes represent the computationally expensive algorithms that this thesis aims to solve. In particular, the MR-linac contains both a treatment beam and an MRI scanner. The output of the MRI scanner, consists of a raw MRI signal acquired in k-space. The k-space is an array of numbers representing spatial frequencies. The k-space data is fed to the real-time imaging algorithm. This results

in a MRI image. This image is then converted to a patient model, that is, a geometry that identifies each material (e.g. water, bone, muscle) inside each voxel. This patient model is then used by the dose delivery reconstruction algorithm together with the beam parameters from the treatment machine to simulate the radiation dose delivery, and based on these data it is possible to optimise the beam parameters, compare them with the planned treatment and send them to the machine.

The optimisation of the treatment parameters is in itself an open research question that goes beyond the scope of this thesis [MMN⁺20]. At the time of writing, the specific impact of this stage on the overall feedback loop remains unclear. Winkel et al. [WBK⁺19] propose an end-to-end concept for plan adaptation, covering the entire feedback loop. They report a time range between 17s and 485s. However, they do not provide a detailed breakdown of the time required for each stage, leaving the duration of the optimisation of treatment parameters unquantified. The inputs of the optimisation algorithm are the planned dose delivery parameters determined by radiologists and oncologists, the results of the simulation, and the patient model. It uses the information provided to optimise the treatment parameters and sends the updated parameters to the treatment machine. For example, some parameters that could be changed to improve the dose delivery are the energy, the position and the direction of the beam.

This thesis aims to contribute to real-time adaptive radiotherapy, where time requirements are more stringent than in MR-guided online Adaptive Radiotherapy (MRgoART). MRgoART is considered an intermediate step since it shares similar computational challenges, and as section 6.2.1 will present, results from MRgoART could be used to develop algorithms that meet real-time demands. However, it is important to clarify the time requirements of MRgoART and real-time adaptive radiotherapy. In MRgoART, reconstruction time must be kept within **two minutes** to minimize patient discomfort [HHO⁺18, FTO20]. The time requirements for real-time adaptive radiotherapy are significantly more stringent, with only **one second** available to execute the steps illustrated in figure 1.1, including imaging, reconstructing the delivered dose, optimizing treatment parameters, and adjusting the beam [VZV⁺20].

Chapter 3 analyses the challenges of real-time imaging and proposes some solutions to accel-

erate MRI reconstruction. Chapter 4 and 5 focus on accelerating dose delivery reconstruction, specifically addressing the challenges posed by Monte Carlo (MC) simulations underlying this reconstruction process.

1.1.1 First Challenge and Objective

The first challenge arises from the complexity of MRI image reconstruction. MRI imaging is divided in two phases, *acquisition* and *reconstruction*. *Acquisition time* refers to the time needed to acquire raw data from the scanner, *reconstruction time* instead, refers to the time needed to convert raw data into a Four Dimensional (4D) volumetric image. MRI image acquisition can last several minutes, a duration deemed too slow for MRgoART. Consequently, techniques have been developed to shorten the *acquisition time*. However, these techniques greatly increase the *reconstruction time* as they rely on expensive conjugate-gradient methods [MP17]. The reconstruction time can be greatly shortened with parallel computing, vectorization and acceleration, hence the first challenge (C1) can be formulated as follows:

C1: To accelerate an MRI reconstruction method that meets MRgoART timing requirements.

Objective O1 of this research is to address C1. The related contribution is summarised below; more information can be found in Chapter 3.

MRI reconstruction algorithms are usually implemented in high-level languages such as MATLAB which do not allow for use of Single Instruction Multiple Data (SIMD). This greatly limits their performance as most of the algorithms are conjugate-gradient based and heavily rely on linear-algebra - which is widely known to benefit from SIMD paradigms. Furthermore, they are bottle-necked by memory bandwidth. The k-space data usually consist of several gigabytes of data. Conjugate-gradient methods, being iterative, require iterating over the data multiple times until they converge. To address these issues Chapter 3 presents several techniques, including a general parallelisation scheme and the use of Graphics Processing Unit (GPU) acceleration to greatly reduce MRI reconstruction time.

1.1.2 Second Challenge and Objective

The second challenge stems from the complexity of MC simulations and the engineering effort required to program accelerators. Programming accelerators presents many different challenges that developers need to overcome to successfully accelerate any application. In addition to choosing the correct algorithm, other challenges include data transfers, utilising the various memory resources available and exploiting the accelerator specific characteristics, for example the parallelism in the context of GPUs. The successful acceleration of MC simulations is even more difficult as MC methods are known to be inefficient, hence they require significant engineering effort and extensive High Performance Computing (HPC) expertise [AAB⁺23]. To avoid investing needless development time and resources into accelerating applications that do not benefit from acceleration, it is important to model and forecast the performance of Field Programmable Gate Array (FPGA) and GPU. This is particularly important in the case of FPGAs where compilation time can take up to several days. Hence the second challenge (C2) can be formulated as follows:

C2: To forecast accelerator performance so that the architecture with the highest performance for MC simulations can be chosen.

Objective O2 of this research is to address C2. The related contribution is summarised below; more information can be found in Chapter 4.

Accelerating MC simulations is a lengthy process that requires significant developer effort. However, MC simulations are stochastic in nature and might not be suited for a particular accelerator or a particular architecture. Identifying engineering challenges and forecasting accelerator performance are crucial to implementing fast MC simulations. Chapter 4 presents how to identify MC bottlenecks, possible solutions, how to devise a heterogeneous architecture and how to forecast its performance.

1.1.3 Third Challenge and Objective

Dose delivery reconstruction relies on MC simulations as MC is the most accurate method to estimate dose depositions. In particular DPM [SWB00] is considered the MC simulator that has the best tradeoff between speed and accuracy in the context of real-time adaptive radiotherapy. However, DPM being stochastic in nature, does not meet real-time adaptive radiotherapy time requirements [VZV⁺20, ZPPKO15]. This limitation poses a considerable challenge for the widespread adoption of real-time adaptive radiotherapy, and thus, the third challenge (C3) is defined as follows:

C3: To devise a Dose Planning Method (DPM) that meets real-time adaptive radiotherapy timing requirements.

Objective O3 of this research is to address C3. The related contribution is summarised below; more information can be found in Chapter 5.

DPM is a complex MC simulation that computes particles (electrons, photons) trajectories in matter. Its complexity combined with its stochastic nature makes it fail to meet real-time adaptive radiotherapy time requirements. DPM suffers from branch mispredictions as many of the computed processes rely on random numbers sampled according to specific distributions. Additionally, the highly irregular trajectories of particles in matter, especially photons, result in numerous cache faults when accessing the geometry. Chapter 5 presents a complete analysis of DPM with forecasts of the performance of GPU and FPGA based accelerators. It also introduces several optimisation strategies, including branchless programming, geometry partitioning, and the use of approximation to enhance DPM performance and meet time requirements.

1.2 Research Contributions

This thesis focuses on solving real-time adaptive radiotherapy computational bottlenecks. The aim is to improve the performance of MRI image reconstruction and dose delivery reconstruction

by means of both optimisations and accelerators. However, there are many challenges that limit the performance of these algorithms, the three main ones are described in section 1.1. To address these challenges this thesis covers the following research contributions: (a) a new parallelisation scheme for 4D-MRI data and the resulting optimised heterogeneous implementation of the XD-GRASP algorithm [FAC⁺16] capable of meeting MRgoART [MHBB20] requirements, (b) a methodology to forecast MC performance when using GPU and FPGA acceleration and (c) Optimised Dose Planning Method (ODPM), an optimised implementation of DPM [SWB00] dose reconstruction algorithm. These contributions are explained in more detail below.

1.2.1 Fast MRI Reconstruction

The first contribution of this thesis is a heterogeneous implementation of the XD-GRASP algorithm that meets MRgoART time requirements to address Challenge C1 and Objective O1.

This contribution (chapter 3) presents a novel parallelisation scheme optimised for 4D-MRI data that minimises multi-threading overhead and hides data transfer latency. This results in an efficient architecture for MRI reconstruction algorithms based on golden angle and stack-of-stars [MP17] MRI reconstruction based algorithms. Then, one of such algorithms XD-GRASP [FAC⁺16] was implemented and accelerated with GPUs to facilitate clinical use. Compared to state-of-the-art MRI reconstruction algorithms, our implementation is up to 17 times faster while maintaining 99% Structural Similarity Index (SSIM). This work has been published in papers [BWY⁺21, LBG⁺23].

Many state-of-the-art solutions present in the literature fall significantly short of MRgoART time throughput requirements [MP17, FTO20, RHB⁺17], moreover, their approaches often focus on a single algorithm only and do not offer general solutions applicable to multiple algorithms. In this study we analyse the XD-GRASP algorithm but our findings are applicable to any reconstruction algorithm that leverages a golden angle, stack-of-stars acquisition strategy and involves the Non-Uniform Fast Fourier Transform (NUFFT) algorithm. This includes an entire class of MRI reconstruction algorithms. Compared to state-of-the-art solutions [MP17, FTO20]

our approach is more general and offers more than an order of magnitude speedup and to the best of our knowledge is the only implementation of XD-GRASP that meets MRgoART time requirements.

1.2.2 Performance Modelling of Monte Carlo Simulations

The second contribution of this thesis is a methodology to forecast the performance of MC simulations when using accelerators to address Challenge C2 and Objective O2.

MC simulations are inherently hard to accelerate, due to their stochastic nature. They do not allow sequential access to the multidimensional geometry, and their irregular behaviour requires a large number of branches in the code. These two characteristics make accelerating MC difficult demanding changes to the simulation to overcome or at least mitigate these challenges. Moreover the use of accelerators requires significant engineering effort. Especially in the context of FPGA when the compilation might take several days. Chapter 4 shows a methodology for forecasting MC performance when using accelerators. The performance model allows evaluation of the potential speedup of FPGA or GPU accelerators. The forecasted speedup can be used to decide the best (if any) accelerator to use to achieve the required performance. The results show that it provides a reasonable range for the execution time in the GPU case. In the FPGA case, since FPGA's performance is more predictable, the predictions are within 2% of the measured execution time. This work has been published in [BHT⁺23], and a revised version has been accepted for publication in the Journal of Physics: Conference Series.

Compared to state-of-the-art solutions our approach has several advantages. First, it is *a priori*, the forecasts are available before any extensive development effort. Second, it supports any MC simulation and it can target CPUs, GPUs and FPGAs. To the best of our knowledge, there is no alternative in the literature that offers similar advantages. In contrast, PPT-GPU [ABC⁺19] lacks integration of a priori forecasts and provides limited support for data transfers, focussing primarily on GPU acceleration across diverse domains. Similarly, GCoM [LHL⁺22] offers GPU support with data transfer capabilities but does not integrate a priori knowledge. Choi et al. [CRKB20] introduce an end-to-end approach with GPU support, yet they do not incorporate

a priori predictions or data transfers. Goswami et al. [GSB22] provide a priori predictions and FPGA support but do not include data transfers, and their support is limited to Convolutional Neural Networks (CNNs). Da Silva et al. [DSBDT13] support FPGA accelerators with data transfers, but have a restricted application scope, particularly lacking support for complex applications like MC simulations. Finally, Voss et al. [VKM⁺21] integrate a priori predictions, data transfers, and FPGA support, suitable for various domains, although they do not provide GPU support in their approach.

1.2.3 Fast Dose Delivery Reconstruction

The third contribution of this thesis is ODPM, a fast MC simulation for dose delivery reconstruction that meets real-time adaptive radiotherapy requirements to address Challenge C3 and Objective O3.

DPM is an MC simulation that offers a good tradeoff between speed and accuracy. However, it falls significant short of meeting real-time adaptive radiotherapy requirements [ZPPKO15]. Chapter 5 first analyses the use of accelerators for DPM. The results of this analysis show that DPM will not benefit from GPU or FPGA acceleration, hence it proposes mathematical, algorithmic and architectural optimisations that allow DPM to meet real-time adaptive radiotherapy requirements. The results show that it is possible to meet real-time requirements, 1s or less for a 66MeV photon beam [VZV⁺20], using large multi-core machines combined with these optimisations. This work is accepted for publication in “EPJ Web of Conf” (To be published soon). A paper further detailing this contribution is also completed and has been submitted for publication.

There are several state-of-the-art solutions for dose delivery reconstruction. ODPM differs from these solutions in two main ways. Some solutions fall short of meeting real-time adaptive radiotherapy requirements, others although meeting time requirements do not provide strong guarantee or proper validation of their results. Amadio et al. [AAB⁺23] use GPUs to accelerate MC electromagnetic showers for the calorimeter at CERN; however, their study highlights the challenges in handling memory accesses but does not provide reconstruction times. Sharma

et al. [SKF⁺18] developed a fast MC simulation tool for dose estimation that leverages GPU acceleration. However, they achieve a performance of 48s which fails to meet time requirements. Li et al. [LDW⁺21] adopted a similar approach for gDPM, where they use GPU acceleration to speedup radiotherapy dose reconstruction. gDPM can reconstruct the radiotherapy dose distribution in 40s. Although they do not meet real-time requirements, they support simulating magnetic fields, which is a necessary feature for the MR-linac, as the MRI scanner requires a 1.5T magnetic field to operate. Pastor-Serrano and Perkò [PSDH⁺23] provide DOTA, a CNN based framework that can predict MC dose distribution. They are among the fastest in the literature requiring between 15s to compute a full dose distribution. While DOTA does not meet real-time requirements it offers a compelling alternative to MC based algorithms. Voss et al. [VZV⁺20] implement dose reconstruction on multiple FPGAs, their results are promising as they are able to meet real-time requirements. However, their study does not mention the validation required to assess the validity of their results.

1.3 Selected Publications

The following publications contribute to chapter 3:

- **M. Barbone**, A. Wetscherek, T. Yung, U. Oelfke, W. Luk and G. Gaydadjiev, “Efficient Online 4D Magnetic Resonance Imaging”, 2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Belo Horizonte, Brazil, 2021, pp. 177-187, doi: 10.1109/SBAC-PAD53543.2021.00029
- B. Lecoer, **M. Barbone**, J. Gough, Uwe Oelfke, W. Luk, G. Gaydadjiev, A. Wetscherek, “Accelerating 4D image reconstruction for magnetic resonance-guided radiotherapy”, Physics and Imaging in Radiation Oncology, Volume 27, 2023, doi: 10.1016/j.phro.2023.100484

The following publications contribute to chapter 4:

- **M. Barbone**, A. Howard, A. Tapper, D. Chen, M. Novak and W. Luk, “Demonstration of FPGA Acceleration of Monte Carlo Simulation”, *Journal of Physics: Conference Series*, Volume 2438, 2023, doi: 10.1088/1742-6596/2438/1/012023
- **M. Barbone**, R. Brandt, A. Howard, M. Novak, W. Luk, G. Gaydadjiev and A. Tapper, “GPU acceleration of Monte Carlo simulations: particle physics methods applied to medicine”, *Journal of Physics: Conference Series*, Volume N/A, 2024, doi: N/A (Accepted)

The following publication contribute to chapter 5:

- **M. Barbone**, G. Gaydadjiev, A. Howard, W. Luk, G. Savvidy, K. Savvidy, A. Rose and A. Tapper, “Fast, high-quality pseudo random number generators for heterogeneous computing”, *EPJ Web of Conf*, Volume N/A, 2024, DOI: N/A (Accepted)

The following publications contribute to chapter 4 and 5:

- **M. Barbone**, A. Howard, M. Novak, W. Luk, U. Oelfke, G. Gaydadjiev and A. Tapper, “Application of computational modelling to particle physics”, *Journal of Computational Physics*, Volume N/A, DOI: N/A (under submission)

The following publications do not directly contribute to any of the chapters:

- **M. Barbone**, C. Brown, G. Gaydadjiev, T. Maguire, M. Mieskolainen, B. Radburn-Smith, W. Luk, A. Tapper, “Embedded Continual Learning for High-Energy Physics”, *EPJ Web of Conf*, Volume N/A, 2024, DOI: N/A (Accepted)
- **M. Barbone**, B. W. Kwaadgras, U. Oelfke, W. Luk and G. Gaydadjiev, “Efficient Table-Based Polynomial on FPGA”, 2021 IEEE 39th International Conference on Computer Design (ICCD), Storrs, CT, USA, 2021, pp. 374-382, doi: 10.1109/ICCD53106.2021.00066
- A. Sahebi, **M. Barbone**, M. Procaccini, W. Luk, G. Gaydadjiev, R. Giorgi, “Distributed large-scale graph processing on FPGAs”, *Journal of Big Data*, Volume 10, 2023, doi: 10.1186/s40537-023-00756-x

- L. Borgna, **M. Barbone**, J. Cao, A. Rose, A. Tapper, R. Bainbridge, W. Luk, “Accelerating the DBSCAN clustering algorithm for low-latency primary vertex reconstruction at the HL-LHC”, EPJ Web of Conf, Volume N/A, 2024, DOI: N/A (Accepted)
- M. Procaccini, A. Sahebi, **M. Barbone**, W. Luk, G. Gaydadjiev and R. Giorgi, “Accelerating Large-Scale Graph Processing with 2 FPGAs: Lesson Learned and Future Directions”, In Proceedings of PARMA: 15th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures & DITAM: 13th Workshop on Design Tools and Architectures for Multi-Core, Volume 116, pp. 6:1-6:12, 2024, doi:10.4230/OASICS.PARMA-DITAM.2024.6

Chapter 2

Background and Related Work

2.1 Introduction

2.1.1 Adaptive Radiotherapy

The efficacy of radiotherapy depends on the accuracy at which cancerous cells can be targeted while sparing as many healthy cells as possible. In image-guided radiotherapy, patients are imaged before the treatment, then based on these images a treatment plan is derived and the patient is treated accordingly. The main problem of this approach is that the information is quickly outdated since the cancer moves, for example lung cancer changes position due to respiration. Figure 2.1 shows the timescale at which different types of cancer change position and the different implementations of adaptive radiotherapy. To correctly target moving cancers, the radiation beam should be adapted during the treatment, accounting for the changes in cancer biology and patient anatomy [KNO⁺18, KPB19, HHO⁺18]. Figure 2.2 shows how more effective real-time adaptive radiotherapy is, compared to traditional image-guided radiotherapy.

Keall et al. [KPB19] observe that real-time adaptive radiotherapy is rapidly progressing since it, has several advantages compared to modern image-based radiotherapy. Given the ability to monitor the patient anatomy and correcting the radiation delivery, real-time adaptive radiotherapy can lead to highly accurate tumour targeting which results in improved patient safety

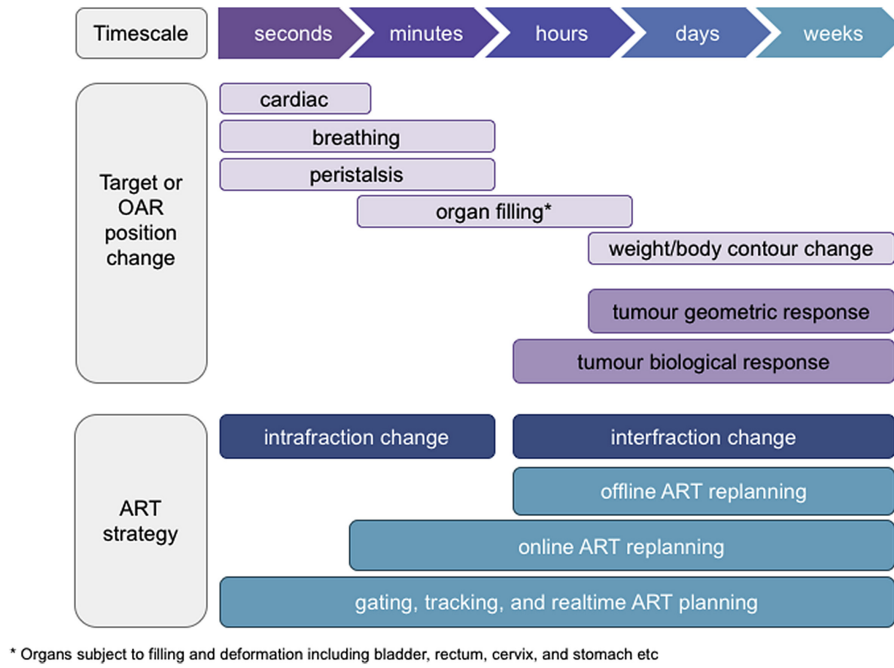


Figure 2.1: From [HHO⁺18], timescales for adaptation and adaptive radiotherapy solutions implemented. ART stands for adaptive radiotherapy, OAR for organ at risk.

and more effective treatment. A recent study [CBN⁺16], compared the radiation dose delivered to a moving dosimeter with and without real-time adaptation. The results show that, according to the gamma analysis, in case of lung cancer the fail-rate¹ was 1.6% with real-time adaptation and 15.2% with no real-time adaptation. Additionally, real-time adaptive radiotherapy is expected to speed up the treatment, allowing treating more patients in the same amount of time, improving the patient experience since the time spent to position the patient under the beam, often in uncomfortable positions, is reduced. Lastly, by removing the human element with automation, it removes the human error from the radiation delivery. The sum of the reduction in staff, in the time needed to perform one treatment, in the number of radiotherapy sessions due to the increase in efficacy, ultimately leads to minor economic costs.

Real-time adaptive radiotherapy can be defined as a real-time system that continuously scans the patient and, by integrating sensing information with prior knowledge from the treatment plan, adjusts the plan to deliver the best suitable dose [KPB19]. This definition leads to new computing challenges: **real-time sensor data processing** and **real-time dose plan update**. Figure 1.1 shows the main components of the system: the yellow boxes model the

¹difference between the dose plan and radiation dose delivered

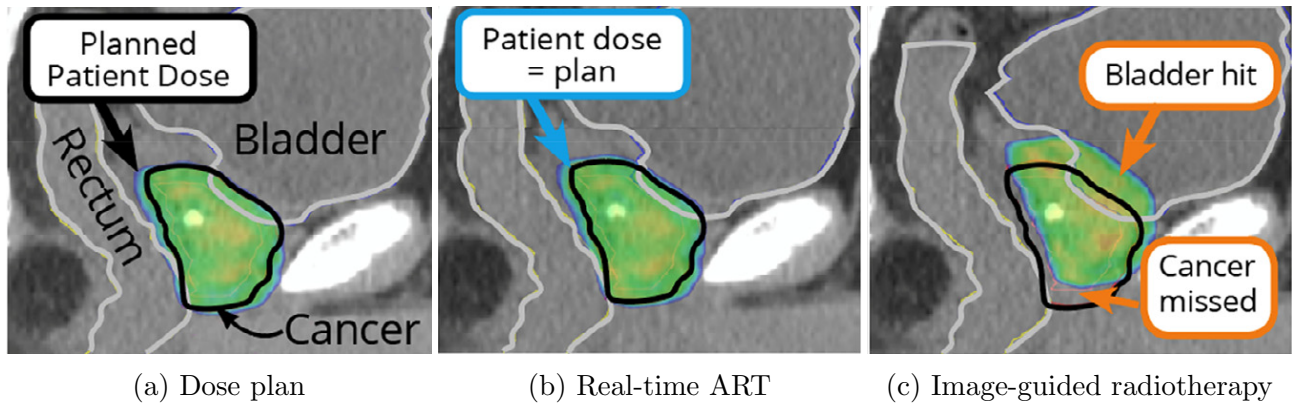


Figure 2.2: Adapted from [KPB19], this image shows the increased accuracy of real-time adaptive radiotherapy (b) compared to traditional image-guided radiotherapy (c).

medical machine needed for the treatment, the red boxes represent the computationally expensive algorithms and the white boxes are the state of the system that gets updated real-time with the new data coming from the sensors and the new parameters determined by the dose planning algorithms. Optimisation of the treatment parameters can also be computationally challenging however, it is highlighted in white as this particular step of the pipeline is outside the scope of this thesis.

In the case of the MR-linac, the sensor is an MRI scanner which produces patient image data in the frequency domain, known as k-space. Due to the limited acquisition speed in conventional MRI, the physical movement of the patient causes ghosting artifacts and blur in the resulting image [FAC⁺16, SAN⁺84, ASKC86]. To reconstruct high-quality 4D-MRI scans from this noisy and undersampled data, reconstruction algorithms are utilized. These techniques can take from 200 up to over 600 seconds to produce a good quality image, hence far too slow to be used in real-time scenarios [MP17].

The dose plan is updated in real-time by running a *dose planning method* on the new data coming from the MRI scanner to compute the *dose distribution*, where:

- The dose distribution is the amount of charge that the particles deposit in each *voxel*.
- The voxel is a discretization of the human body, usually consisting of a cube of $2mm^3$.
- The *dose planning method* is usually based on **MC** simulation of the pathway of high energy particles in biological tissue [ZPPKO15].

2.1.2 Magnetic Resonance Imaging (MRI) processing

MRI is a medical imaging technique that uses radio waves and a strong magnetic field to produce images of the organs inside the human body. Predominantly, the human body is made of water, which is a molecule made of two hydrogen atoms and one of oxygen. The hydrogen nuclei consist of only one proton and since protons are very sensitive to magnetic fields, can be controlled using radio waves. Protons in different tissues change state at a different speed, generating distinct signals. These signals are captured by radiofrequency receivers, which typically consist of two electromagnetic coils [Sin09]. The data from the coils are then combined to generate the final image [MMGP06].

Unlike radiography, MRI data acquisition takes several seconds. For instance, Mickevicius and Paulson report that the acquisition time for MRgoART is between 120s and 140s [MP17]. It is important to note that shorter acquisition times can increase reconstruction time, as more data must be processed to exploit correlations and retrieve unsampled points for accurate image reconstruction. Due to this long acquisition time, patient movements blur and generate ghosting artifacts in the resulting image. To obtain a high-quality image, a process known as image reconstruction filters these artifacts and reduces blur. There are many strategies to minimise artifacts and blur in data acquisition that ease the reconstruction so that it requires less time to perform the post-processing.

In case of respiratory motion, the simplest approach is to either suspend respiration during the scan [PB86] or acquire the data always in a specific respiratory state [FAC⁺16]. The first technique, while largely used in clinical MRI exams, is not suitable for real-time radiotherapy since it would require the patient to hold the breath for the entire duration of the radiotherapy session, which may last several minutes. In the second case, sampling the tumour twice each respiratory cycle greatly increases the acquisition time, hence it does not allow precise tracking and dose plan updates. Another major limitation is that these techniques cannot be used in all cases, for example, it is not advisable to stop heartbeat during a radiotherapy session.

In 4D-MRI there is a trade-off between data acquisition time and processing time. Techniques

that reduce the acquisition time, usually, increase the complexity, of the reconstruction algorithms needed to reconstruct the final volumetric image.

Parallel imaging techniques [PWBB01] reduce the acquisition time by combining under-sampled data from multiple receiver coils to reconstruct the original image [FGB⁺14]. The idea behind parallel imaging is similar to the techniques utilised to generate a Three Dimensional (3D) stereoscopic photograph. Either the same camera is used twice from two different angles, or two cameras are used at the same time, thus, at least halving the time needed to take the picture.

Temporal parallel imaging [KEM01] and k-t acceleration [TBP03] methods reduce the total amount of data that needs to be acquired. The first exploits the temporal correlation between different coils so that data can be reconstructed instead of being sampled. The second method avoids sampling multiples times the same data point by taking advantage of the sparsity of the coil sensor data.

Compressed sensing [LDP07] is another method to reduce data acquisition times by deploying irregular under-sampling according to both spatial and temporal correlations present in coil sensor data.

All of the above techniques, reduce the data acquisition time at the expense of the reconstruction time since data from different coils needs to be recombined and undersampled points need to be interpolated from sampled data according to their correlation.

Recently, different undersampling and reconstruction algorithms have been proposed for 4D-MRI [MP17]. Despite the long acquisition and reconstruction time measured with some of these algorithms which ranges from few minutes to several hours, the 3D radial stack-of-stars method [STDdS⁺15, FAC⁺16], is a promising approach due to its ability to reduce acquisition and reconstruction time while achieving good image quality. In particular, the XD-GRASP algorithm best overcomes the undersampling artifacts while reducing the total time to less than five minutes [MP17].

2.1.3 MR-guided online Adaptive Radiotherapy (MRgoART)

MRI has been used for many years in cancer diagnosis for its excellent soft-tissue delineation, its ability to acquire images in any orientation, and its variety of available contrasts. More recently, MRI has been adopted for radiotherapy treatment planning Hybrid MR-linacs, which integrate an MRI scanner with a linear accelerator for radiation treatment delivery, were clinically introduced to perform MR-guided radiotherapy. [LRR⁺08, FMR⁺09, KBC14, MLC⁺16]. MR-guided radiotherapy aims to deliver high-energy beams to the tumour while sparing healthy tissue based on the acquired images. Combining the two systems therefore enables treatment delivery adaptation based on the images acquired just before (online) or during (real-time) the treatment session, leading to what is called MR-guided online Adaptive Radiotherapy (MRgoART). As the images are used directly after acquisition to inform the positions of the organs for radiation treatment, the image reconstruction time becomes critical and should be kept within a couple of minutes [HHO⁺18].

In the case of thoracic cancers, respiratory-motion resolved 4D-MRIs are of particular interest to spare vital organs such as the heart and lungs by taking into account the respiratory cycle. Producing such images is however challenging, due to the need for undersampled image acquisition, which greatly shortens the acquisition time at the expense of increasing the complexity of the image reconstruction [SPT18]. To maintain short acquisition times, many solutions have been designed to recover high-quality pictures from undersampled data using computationally-expensive iterative techniques [MP17]. These techniques, however, do not meet the necessary time constraints for MRgoART use.

While the acquisition constraints are inherent in the underlying physical laws, the reconstruction throughput is decided by the algorithm employed to recover the image and the efficiency of its software implementation. In particular, parallelising and accelerating the reconstruction algorithms with the help of accelerators can result in significant increase in throughput, enabling them to meet the requirements of MRgoART. While MRgoART constraints are not defined in the literature, in this thesis we consider an image of size 256x256x48 pixels and 10 respiratory phases as a high-quality 4D image [FTO20]. Moreover, we regard reconstruction

time of at most two minutes as suitable for MRgoART use. Based on the above assumptions, the throughput that a reconstruction algorithm should achieve is 262,144 pixels/second. Additionally, the reconstructed image should be completely equivalent to an artefact-free reference image. Hence, the image should score at least 99% SSIM [WBSS04].

2.1.4 Real-time dose reconstruction

Since MC simulation is considered to be the most accurate method to calculate dose distribution in radiotherapy [SWB00], the reconstruction is implemented using computationally expensive MC methods. These methods are based on the idea of repeating an experiment many times to obtain many quantities of interest using the Law of Large Numbers and other methods of statistical inference [KBTB14].

The experiment performed many times is the simulation of the behaviour of individual particles. In particular, it simulates the trajectories and dose depositions of many individual particles traversing biologic tissue by sampling particle interactions according to the fundamental laws of physics. Due to the stochastic nature of MC algorithms, a large number of particles has to be simulated. The simulation of each particle is computationally expensive due to the high complexity of the interaction sampling [ZPPKO15]. Hence, MC algorithms, even if accurate are not directly usable for real-time use.

Voss et al. [VZV⁺19] ported a simplified version of DPM [SWB00] on FPGA and showed that by exploiting the intrinsic parallelism of MC methods, and parallelizing on multiple FPGAs it is possible to meet the real-time requirements. However, their implementation lacks essential features, such as the support of photons and other materials than water, and proper validation against clinically accepted simulators or experimental data. So we will do better here.

Structure

Figure 2.3 illustrates the general structure of the simulation: the yellow boxes show the parts related to the photons and the blue boxes are related to electrons. It can be noticed that some

Algorithm 1 MC main loop**Input:**

VoxelCube 3D matrix representing the patient data
Size 3D vector defining the size of the phantom
Resolution 3D vector defining the resolution of the Voxels
N number of particles to be simulated
Beam set of parameters that defines the radiation beam

Output:

VoxelCube 3D matrix representing dose distribution

```

1: function MC(VoxelCube, Size, Resolution, N, Beam)
2:   Particle[N]  $\leftarrow$  empty
3:   for i=0 to N do Particle.push(Beam.generateParticle())
4:   for Voxel in VoxelCube do Voxel.dose  $\leftarrow$  0
5:   while not Particle.empty() do
6:     p  $\leftarrow$  Particle.pop()
7:     v  $\leftarrow$  VoxelCube[p.position()]
8:     p.initMFP() ▷ Initialise Mean Free path
9:     while True do
10:      event  $\leftarrow$  p.MRF() ▷ Use Mean Free path to determine next interaction
11:      switch event do
12:        case Moller
13:          MollerInteraction(p, v)
14:        case Hinge
15:          HingeInteraction(p, v)
16:        case Bremsstrahlung
17:          BremsstrahlungInteraction(p, v)
18:        case No interaction
19:          No interaction, update the voxel and perform another simulation step
20:        case Particle absorbed
21:          Particle energy below the threshold, removed from the simulation
22:        case Particle escaped
23:          Particle escaped from the VoxelCube, removed from the simulation
24:        default
25:          runtime error, unknown event
26:      if event  $\notin$  {Absorbed, Escaped} then
27:        p.updateVoxel()
28:      else
29:        Break
30:  return VoxelCube

```

events generate *secondary particles*, these particles are the result of particular events and they should be simulated along with the *primary particles* that are generated by the linac.

Algorithm 1 shows more in detail the core of the simulation in case electrons only are considered.

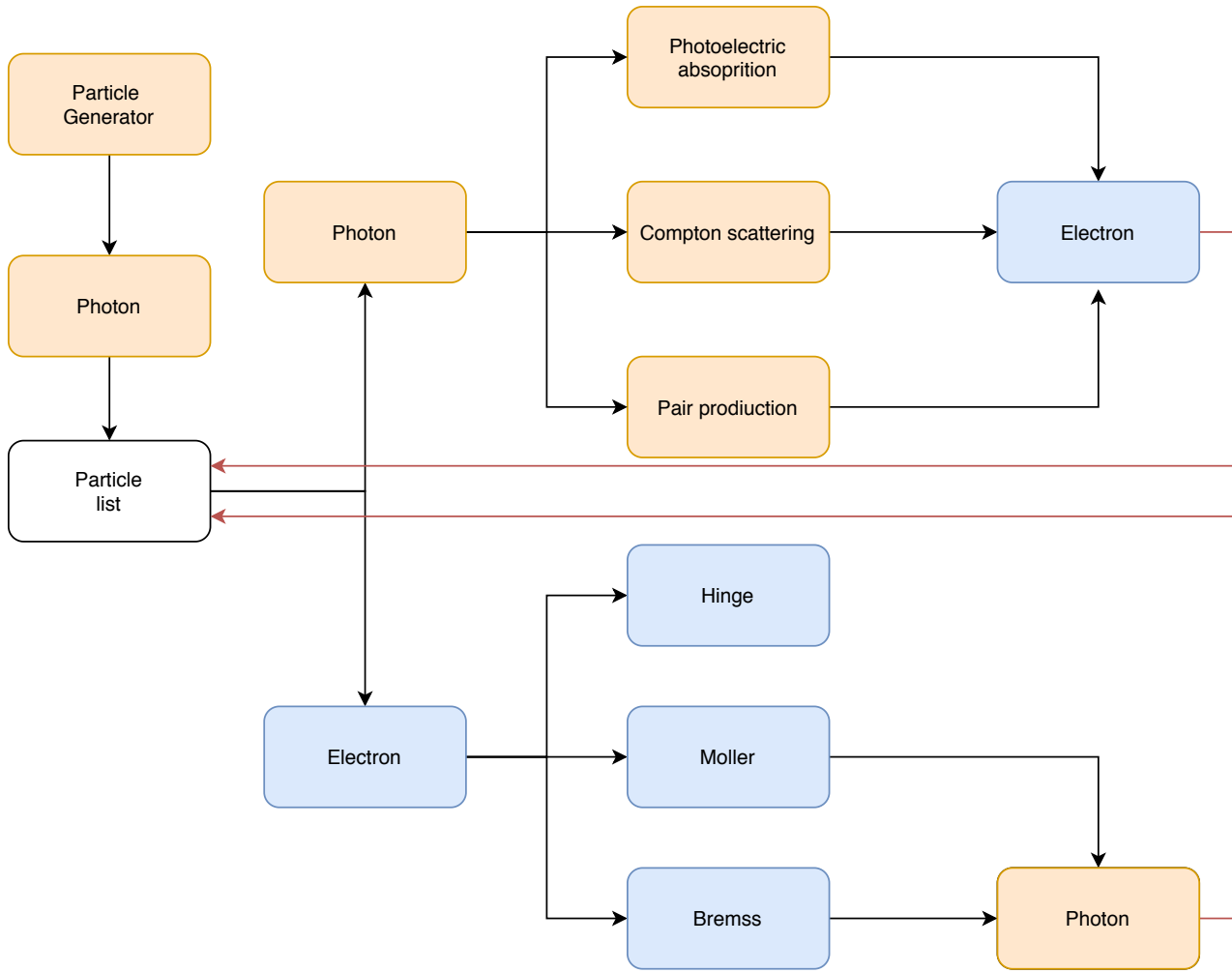


Figure 2.3: General structure of the simulations.

The inputs (line 1) are a VoxelCube, the size of the cube in terms of voxels, the resolution in millimeters, the number of particles to simulate. The VoxelCube is a model of the patient anatomy that contains information about the biological tissues of the area of interest, usually acquired by means of an MRI scan. This biological tissue is discretized in voxels which are small cubes, usually about $2mm^3$, that contain information about the tissue. More generally they are referred as material since they can contain either biological or not biologic material. The main materials supported by the simulation model are water, air, bone, muscle and brain.

The resolution is the size of the voxel in millimetres. This information is needed by the simulator to compute the energy depositions since many different physical quantities depend on the distance. Also, size and resolution combined control the level of detail of the simulation.

N is the number of particles to simulate. Ziegenhein et al. [ZPPKO15] simulate 10^7 or 10^9

particles to eliminate statistical variation.

Beam, the last parameter, represents the radiation beam configuration. There are many possible configurations covering the most common radiotherapy use-cases. It is possible to change the type of particles; currently electrons and photons are supported. The angle, distance and area irradiated and the energy of the particles.

The output is a 3D matrix in which each element is the charge that the particles deposited in that specific voxel. The amount of charge deposited in a voxel is usually referred to as the radiation dose. In Algorithm 1 the original VoxelCube is updated with the charge deposits by adding them to *voxel.dose*.

Electrons

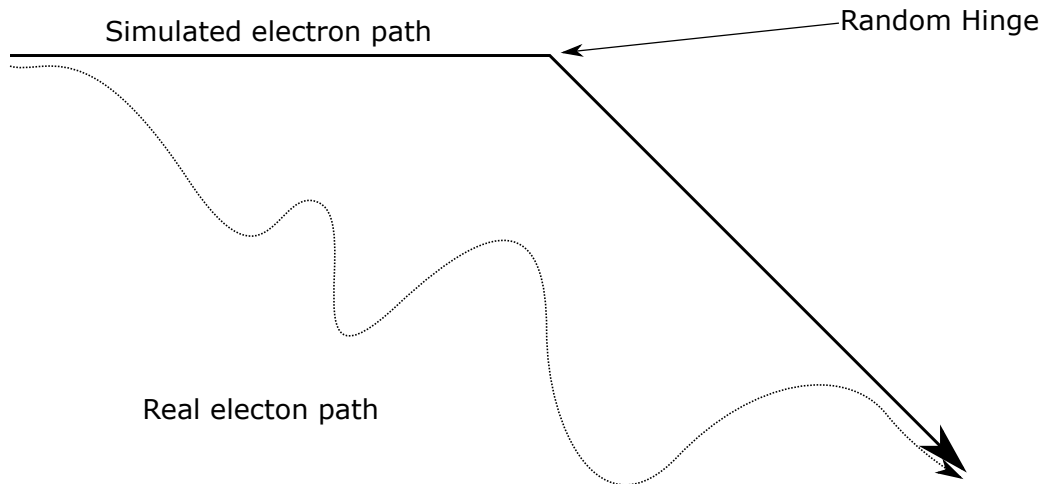


Figure 2.4: Adapted from [SV13], this image shows the difference between the real and the simulated electron path

In the case of electrons, the original DPM implementation supports three kinds of particle interactions: Hinge, Moller, Bremsstrahlung [SWB00].

Since the result of the simulation is a per voxel dose distribution, it is not necessary to simulate the exact path of the particles. Figure 2.4 shows the difference between the real electron path and the approximation performed by the simulation. In this case, an elastic scattering event is simulated using the hinge method; the electron is scattered and it changes direction. The path that the electron takes inside the voxel is irrelevant; only the type of interaction, the energy

loss, the output angle, which determines the next voxel, and the energy of the outgoing electron impact the result of the simulation [SV13].

Figure 2.5 shows a possible *history*, the list of positions and events, of an electron. The particle is scattered several times and there is one hard Moller interaction and one Bremsstrahlung event. Intuitively Moller interaction happens when the primary electron interacts with an electron present in the matter and “kicks” it out of its potential well, setting it in motion. This electron, called secondary electron, is added to the list of moving particles and is simulated until it gets absorbed or escapes from the voxel cube.

Hard Bremsstrahlung events are slightly different. They occur when a particle, usually an electron, is deflected by another charged particle, usually an atomic nucleus. In this case, the particle decelerates the kinetic energy lost is converted into electromagnetic radiation (photon) [HN04]. This photon is then simulated altogether to the other particles in the MC simulation.

Lastly, during the MC simulation, the charged particles lose energy due to soft interactions. Differently from hard Moller and Bremsstrahlung, these interactions do not change the direction of the particle but cause energy depositions in each voxel traversed by the particle.

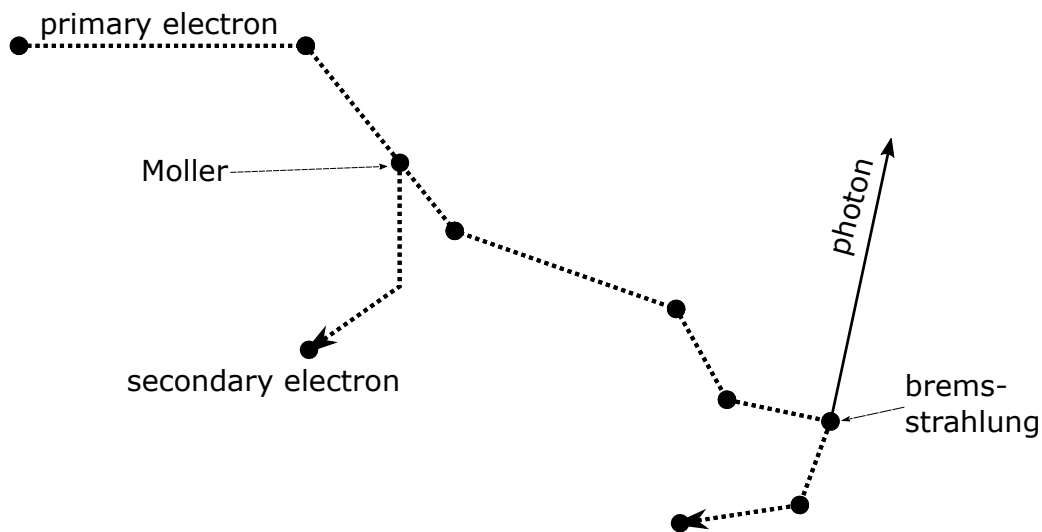


Figure 2.5: Adapted from [SV13], example of a particle history.

Photons

Due to size, the geometry of simulations, and the therapeutic energy range, differently from electrons, very few photons are simulated. As in the case of electrons, all hard interactions are explicit simulated and all soft interactions are implicitly simulated by continuous energy loss into the voxels traversed by the particle.

The code for simulating is similar to Algorithm 1. In all simulation steps, based on a stochastic process, the occurrence of the various interactions is checked. Differently from electrons, multiple interactions in one processing step are allowed. The photons are then simulated until they get absorbed or they escape from the simulation.

DPM implements three processes involving photons, *photoelectric absorption*, *Compton scattering* and *pair production*. In radiotherapy, Compton scattering is the only process that significantly impacts the dose distribution, hence DPM adopts approximations for photoelectric and pair production interactions [SWB00].

The photoelectric absorption is an interaction between a photon and an electron bound to an atom. The photon completely disappears and transfers most of its energy to the electron. If the energy is high enough, the electron escapes from the atomic bound and it is simulated with all the other moving particles [Hal36].

Compton interactions are similar to photoelectric interactions. Differently from the photoelectric absorption, in Compton interactions the photon does not disappear, it is only sharply deflected from the original trajectory. This deflection is caused by the interaction with an electron called recoil electron. A fraction of the photon energy is transferred to the electron. If the energy is high enough, or if the electron is not bound to any atom, it is set in motion and added to the simulation [Chr86].

In pair production interactions the photon disappears and it is replaced by an electron-positron pair [Hub06]. These two particles combined have the same energy as the original photon hence they need to be simulated. The positron is simulated the same way as the electrons with only one difference. Since positron is anti-matter it gets absorbed with a process known as

annihilation. In this process, the positron combines with an electron and both disappear, replaced by a pair of photons. The energy (E) of the resulting photons is as calculated from the equation formulated by Albert Einstein, $E = mc^2$, in which m is the mass of the particle and c is a constant equal to the speed of light in vacuum [SBSW61].

2.1.5 XD-GRASP algorithm

Mickevicius and Paulson [MP17] tested different reconstruction algorithms and measured the overall image quality, reconstruction time, artifacts prevalence and motion estimates. They tested four of the most promising algorithms: non-uniform fast Fourier transform (NUFFT) [HWS16], iterative self-consistent parallel imaging reconstruction (SPIRiT) [LP10], sensitivity encoding solved with a conjugate gradient algorithm (CG-SENSE) [WHG⁺14], and multi-dimensional golden-angle radial sparse parallel MRI (XD-GRASP) [FAC⁺16]. Their results show that while NUFFT is the fastest, it is also very sensitive to artifacts. On the other hand, XD-GRASP was not fast but it is the least sensitive to undersampling artifacts, hence leads to short scan duration, combined with a longer processing time of the XD-GRASP algorithm, resulted in both high-quality images and a reduction of the overall time required to acquire and process the image data.

Feng et al. claim [FAC⁺16] that “*the XD-GRASP algorithm extends the iGRASP algorithm with the self-navigation property of radial imaging and uses motion detection schemes that instead of removing or correcting the motion in question, XD-GRASP reconstructs extra motion dimensions which improves image quality and also offers new physiological information of potential clinical value*”. The description means that this algorithm exploits the correlation of temporal frames instead of considering them individually. This idea is based on the assumption that breathing is a symmetric motion so that any frame of the inhalation phase as an exact equal frame in the exhalation phase.

The idea behind iGRASP is to combine k-t acceleration methods implemented in SPARSE-SENSE [TBP03] with golden-angle radial sampling [FGB⁺14]. In MRI, the most simple sampling technique is *Cartesian sampling*. In the case of Cartesian sampling, all the sampling

points are equidistant between each-other. This sampling technique is very efficient when the input data uniformly distributed. In the case of MRI, it is inefficient since the coil sensor data is not uniformly distributed. To overcome this limitation many density-based sampling techniques, that sample coils sensor data taking the distribution of the values into account have been developed [FGB⁺14, RGJ⁺20].

The iGRASP is based on the golden-angle radial sampling combined with compressed sensing and parallel imaging [PWBB01] to process MRI signals into volumetric images. This sampling technique is used to sample complex signal in the frequency domain such as the ones received by MRI coils. Complex data is represented by amplitude and phase. These two parameters identify a point in the complex space. In MRI, points closer to the origin carry the most image-space information. Efficient sampling techniques should sample these points more than others to obtain high-quality volumetric images with low sampling rates. Golden-angle radial sampling ensures approximately uniform coverage of k-space (frequency domain), for any arbitrary number of consecutive spokes (lines that intercept the origin), while under-sampling points further from the origin [FGB⁺14].

The iGRASP reconstruction is an optimisation problem formulated as follows [FGB⁺14]:

$$d = \underset{d}{\operatorname{argmin}} \{ \|F \cdot S \cdot d - m\|_2^2 + \lambda \|T \cdot d\|_1 \} \quad (2.1)$$

where:

- $\mathbf{d}$ are image series to be reconstructed in the x-y-t space
- $\mathbf{T}$ is the temporal total-variation (TV) operator (sparsifying transform)
- $\mathbf{m} = \begin{bmatrix} m_1 \dots m_c \end{bmatrix}^T$ are the acquired multi-coil radial k-space data with c coils
- $\mathbf{F}$ is the NUFFT operator defined on the radial acquisition pattern
- $\mathbf{S} = \begin{bmatrix} S_1 \dots S_c \end{bmatrix}^T$ are coil sensitivity maps in x-y space

- λ is the regularisation weight that controls the trade-off between data consistency and the regularisation enforcing sparsity

Intuitively, this optimisation problem, solved using the conjugate-gradient, aims to minimise the distance between the true image and the acquired signal.

The XD-GRASP algorithm customised this model using the characteristics of the motion taken into account leading to a reduction in the number of operations required to solve this problem [FAC⁺16]. The main difference between the two algorithms is how the data is divided. XD-GRASP divides it by phases, e.g. respiratory and cardiac phase while iGRASP divides it in uncorrelated time frames in the order in which they were acquired.

While XD-GRASP reduces the data acquisition time by combining many different techniques, it is computationally expensive and the total acquisition and reconstruction time is by orders of magnitude higher than the real-time requirements. Given the importance of this algorithm in real-time radiotherapy, different optimisations, both run-time and compile-time, as well as the use of FPGAs and GPU accelerators should be considered.

2.1.6 Structural Similarity Index (SSIM)

SSIM is a metric to measure the similarity between images based on the degradation of structural information. It replicates the human visual perception system capability of identifying structural information and used it to measure the difference between images [WBSS04]. Two images are equivalent if they achieve SSIM equal to 1 or 100%. The SSIM is calculated using a formula that considers luminance, contrast, and structure between two images, x and y . The SSIM index is defined by the following formula:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

where:

- μ_x and μ_y are the average luminance of images x and y respectively.
- σ_x^2 and σ_y^2 are the variance of images x and y respectively.
- σ_{xy} is the covariance between images x and y .
- C_1 and C_2 are constants added to avoid instability when the denominator approaches zero.

This formula represents a comparison between local windows of the images, assessing their similarity based on luminance, contrast, and structure. The resulting SSIM value ranges between -1 (indicating no similarity) to 1 (indicating perfect similarity).

2.2 Kolmogorov–Smirnov Test

The Kolmogorov–Smirnov test [BZ14], often abbreviated as the K-S test, is a nonparametric hypothesis test used to determine whether two samples are drawn from the same probability distribution. It evaluates the equality of continuous, one-dimensional probability distributions or empirical cumulative distribution functions (CDFs) of two samples. The null hypothesis of the test assumes that the samples are drawn from the same distribution, while the alternative hypothesis suggests that the samples are drawn from different distributions. The test statistic for the K-S test is based on the maximum absolute difference between the two empirical CDFs. If the test statistic exceeds a critical value (α) corresponding to a chosen significance level, the null hypothesis is rejected, indicating that the two samples likely come from different distributions. In particle physics a value of $\alpha = 5 \times \sigma$ is considered “gold standard” [Lyo13] hence, MC simulations results should meet this requirement to be considered correct.

2.3 Random Number Generation

MC simulations are heavily reliant on random numbers. These random numbers are generated using Pseudo Random Number Generators (PRNGs) ². The preference for PRNGs over True Random Number Generators (TRNGs) stems from practical considerations. To reproduce a specific run using PRNGs, it is sufficient to utilise the same parameters and seed value. In contrast, when employing TRNGs, all generated random numbers must be accurately stored and reintroduced into the simulation, a process that can be both cumbersome and resource-intensive.

RNGs in MC simulations must satisfy two primary criteria: **quality** and **speed**. In modern MC simulations, which model complex physical systems with numerous degrees of freedom, the *quality* of RNGs holds significant importance. RNGs that fail to fulfil quality requirements can lead to subtle yet *significant* systematic errors [FLW92]. Furthermore, the quality of the RNG can impact the rate of convergence [STAS91]. Although lower quality generators might not directly compromise the simulation results, in the most adverse cases, they can escalate the number of simulated histories required for convergence by an order of magnitude. Hence, the selection of a high-quality, fast generator is crucial for the efficient implementation of MC simulations.

Knuth summarises the lessons learned by using and finding flaws in many RNGs [Knu98]:

- The period of an RNG should exceed the length of any sequence that will be used in a single computation, but this alone is not enough to guarantee the absence of flaws;
- Empirical testing can reveal flaws in an RNG (if it fails a test), but no amount of empirical testing can ever confirm the flawlessness of an RNG;
- Increasing the complexity of an algorithm, especially by combining two or more methods in the same algorithm, may enhance an RNG, but it can also significantly degrade it if the component methods are not statistically independent;

²Throughout this chapter, all Random Number Generators (RNGs) are assumed to be PRNGs unless specified.

- It is preferable to use an RNG that has been studied extensively, whose flaws are known and understood, rather than one that appears good but whose flaws are unknown;
- There is no universal method to determine the level of goodness required for a specific MC application. The most effective way to ensure that an RNG is suitable for a given application is to use one designed to be adequate for all applications.

The majority of MC simulations are computationally intensive and necessitate parallelism to achieve completion within a reasonable timeframe. RNGs that are incompatible with parallel computing cannot be effectively employed. Even when each thread selects a different seed, it **does not guarantee** the absence of correlations between streams that may lead to systematic errors [Hel98]. Hence, it is crucial for the RNGs to support parallel streams. Considering the outlined constraints, there are very few RNGs options suitable for MC simulations. One such option is the Mersenne Twister (MT) [MN98], which has served as the default choice for over two decades.

In a series of publications [STAS91, Sav15, SS16], the authors developed new principles for the construction of high-quality random number generators (RNGs). This alternative approach is based on Ergodic theory, and the authors suggested constructing RNGs using maximally chaotic dynamical systems (MCDS) developed by Anosov and Kolmogorov. These systems T are characterised by the *Kolmogorov-Sinai entropy* $h(T)$, which allows for a quantitative comparison, construction, and selection of high-quality RNGs. The analysis performed in [Sav15] made it clear that Mersenne Twister (MT) [MN98] and RANLUX [JM20] generators have low entropies of order $h(MT) \approx 4.8$ and $h(RANLUX) \approx 7.81$. The MIXMAX generator of dimension N developed in [STAS91, Sav15, SS16] has the entropy $h(MIXMAX) \propto \ln(m)N$, where $m = 2^{36}$ and $N = 8, 17, 240$. Thus, the MIXMAX generators have very high entropies $h(T)$ compared with the other RNGs, and therefore, they have very short decorrelation time $\tau_0 = \frac{1}{2Nh(T)}$ and relaxation time τ , as well as comfortably large periods q . These basic parameters are presented in Table 2.1.

Dimension N	Entropy $h(T)$	Decorrelation $\tau_0 = \frac{1}{h(T)2N}$	Iteration t	Relaxation $\tau = \frac{1}{h(T) \ln \frac{1}{\delta v_0}}$	Period q $\log_{10}(q)$
8	220	0,00028	1	1,54	129
17	374	0,000079	1	1,92	294
240	8679	0,00000024	1	1,17	4389

Table 2.1: The MIXMAX parameters. $\tau_0 < t < \tau$. The iteration time t is normalised to 1. The MIXMAX is a *genuine 61 bit generator* on Galois field $\text{GF}[p]$, with Mersenne prime number $p = 2^{61} - 1$.

2.4 CPU architecture

Figure 2.6 shows modern a modern CPU architecture where multiple dies are interconnected dies instead of using a monolithic die containing all the cores. It can be noted that on a single socket there are multiple CPU dies (called CCX using AMD terminology) with no direct connection between each other. Any data transfer goes through an I/O die that forward communications between CCXs and main memory. Moreover, modern servers can have multiple sockets hence multiple CPUs. Cache to cache communications between CPUs on the same socket are more costly than the ones than communications between cores on the same die. Communications between different CPUs are extremely expensive. Requiring the similar access latency than memory access latency. Our experiments show that when these latencies are not accounted properly parallelism causes a slowdown of multiple order of magnitude (over 100 times slower) instead of achieving a speedup.

2.5 Field Programmable Gate Array (FPGA)

FPGAs are integrated circuits designed to be reconfigurable after manufacturing. Compared to Application Specific Integrated Circuits (ASICs) that have only one functionally and cannot be re-programmed to perform any task other than the one specified by the manufacturer, FPGA can be used in many different applications. Recent advances in High Level Synthesis (HLS) toolchains for customised hardware implementations [COL⁺14, PMTL13, SKA13, XLS], combined with the continuous scaling of resource capacities and speeds of modern FPGA devices,

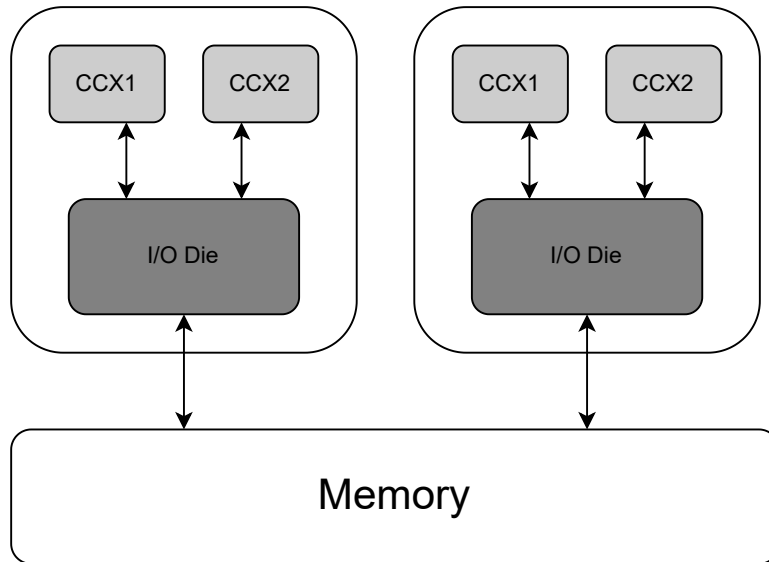


Figure 2.6: Modern CPU architecture

have enabled the development of large-scale applications on systems incorporating FPGA accelerators. Notably, FPGA are used to accelerate CNN [Mit20,VBM⁺17,Sum18], Gzip [VBMG17], MC [VZV⁺19,BHT⁺23] and function evaluation [DdD05,BKO⁺21]. CERN is also exploring the potential of porting its HEP workflows to FPGAs [Sum18], extending the traditional DAQ application.

Each FPGA has a finite number of resources, including programmable logic blocks, routing resources, input/output (I/O) interfaces, and memory components. The compiler utilises these resources to efficiently implement code on the FPGA, ensuring that the FPGA is optimally configured for the desired task. Therefore, it is important to carefully manage and allocate these resources to ensure that the FPGA is utilised to its full potential. Failure to do so can result in sub-optimal performance or even program failure, making careful resource management critical in FPGA programming [Int]. The availability of resources in FPGAs varies across different models and vendors. Nonetheless, the following resources are commonly found in most FPGAs, while some may offer additional resources beyond the ones outlined below. In the upcoming section, an overview of the various resources available will be presented; however, the precise details are beyond the scope of this paper. Additionally, since most HLS compilers handle the low-level design tasks, developers seeking to accelerate their application rarely require further information. Typically, this information is necessary only when conducting specific low-level

optimisations that the compiler cannot automate.

2.5.1 Configurable Logic Blocks

A Configurable Logic Block (CLB) is an essential repeating logic component found on FPGAs. When interconnected through routing resources, the constituent elements within CLBs execute intricate logic functions, enable memory functions, and facilitate code synchronisation on the FPGA. CLBs are constituted of mainly of four components:

- **Flip-Flops (FFs)**: binary register that is capable of storing a single bit. It is the smallest storage unit on the FPGA;
- **Look-up Tables (LUTs)**: they are a collection of gates that allows mapping output bits to input bits. It is usually used to implement truth tables.
- **Multiplexers**: a circuit that selects one of several inputs. For example, if there are two or more possible inputs, based on the control signal only one will be returned in output.

2.5.2 Input/Output

Input and output (I/O) resources of an FPGA target are physical structures that enable an FPGA target to communicate other devices. They can be used to transfer both analog or digital data however the corresponding processing logic used to interpret such data needs to be synthesised onto the FPGA. Examples of I/O resources are PCIe, QSFP, Firefly optic transceivers, and Ethernet ports. The board manufacturer can customise and include any port depending on the use case.

2.5.3 Block RAM

Block RAM (BRAM) resources are a type of random access memory that are available throughout an FPGA for the purpose of data storage. In addition to the storage purpose they can

be used to implement First In, First Out (FIFO) buffers. FIFOs can be used to transfer data between multiple clock domains, between external devices and schedule the design. In some cases, FPGAs may contain supplementary memory resources such as the Ultra RAM (URAM) in AMD Ultrascale+ or the M9k, M144k, M10k and M20K in most Intel FPGAs. For the purposes of this thesis, it is deemed appropriate to treat these auxiliary resources as equivalent to Block RAM (BRAM) and hence, all available on-chip memory resources will be referred to as BRAM.

2.5.4 Digital Signal Processor

Digital Signal Processors (DSPs) are a common resource found on most FPGAs. They are specialised units that perform multiplications, with many Intel and AMD FPGAs offering DPS with asymmetric 18 and 27 bit inputs. That means that any operation that requires a maximum of 18×27 bits will be implemented using exactly one DSP, with additional logic (LUTs, FFs) or DSPs utilised for higher bitwidth. Moreover, DSPs often include pre-post adders and other functionality such as multiplexers. For instance, operations like $a \times b + c$ may not require additional logic and can be implemented efficiently using DSPs. While the extra features of DSPs can reduce resource utilisation, they do not typically result in drastic reductions. Therefore, these features are not discussed in detail in this paper, but readers can refer to the official documentation for further information as needed.

2.5.5 FPGA programming

Figure 2.7 shows the FPGA programming overview. The first step is optional and consists of implementing the FPGA architecture using HLS. But it is recommended to use High Level Synthesis (HLS) for accelerating their applications to meet performance requirements because it can save engineering time. This is the only step that the developer needs to perform, as the following steps are automated by the compiler.

The second step involves the generation of Hardware Description Language (HDL) code to

model the application, which is typically performed by the HLS compiler. However, experienced FPGA developers may choose to undertake this task themselves. It is important to note that while HLS is a powerful tool, it has some limitations, and in certain cases where precise control over the hardware is required, it may be impossible to implement the desired architecture using HLS.



Figure 2.7: FPGA programming overview

The third step involves transforming the HDL description from an abstract human-readable format to a post-synthesis netlist, which specifies the components and their interconnectivity that need to be instantiated using FPGA resources. Depending on the size of the architecture this step might take from a few minutes to a few hours.

The fourth step involves placing and routing the design. The initial step known as placement, involves determining the optimal location of components, circuitry, and logic elements within the FPGA. Following placement, routing is performed, which determines the specific layout of wires required to connect the placed components. This process must ensure that all desired connections are established while adhering to the manufacturing process rules and restrictions. One such constraint is known as the timing requirement, which is derived from the target clock frequency. Because electromagnetic fields have a finite propagation speed, the time it takes for signals to travel from their source to their destination must be shorter than the clock interval. If this requirement is not met, the computed results will be incorrect. The routing algorithm adjusts the positions of components to increase or decrease wire lengths as necessary to meet the timing requirement. Once a valid permutation that meets the timing constraint is found, the algorithm terminates. The algorithm employed in this step has an exponential complexity. In worst-case scenarios, all possible permutations of components need to be evaluated to find one that meets timing requirements, if one exists. Therefore, depending on the size of the design, this step may take several days.

The last two steps involve generating the bitstream, which is a file containing the configura-

tion information required for programming an FPGA. The final step involves streaming the bitstream onto the FPGA to reconfigure it, thereby programming the FPGA to execute the application.

2.6 Graphics Processing Unit (GPU)

Graphics Processing Unit (GPU) is a specialised electronic circuit primarily initially intended to accelerate computer graphics and image processing tasks. Subsequently, their utility expanded beyond graphic processing to include non-graphic computations, particularly those involving embarrassingly parallel problems due to their inherent parallel structure.

2.6.1 Single Instruction Multiple Data (SIMD)

Single Instruction Multiple Data (SIMD) is a fundamental aspect of GPU architecture, characterised by its wide SIMD width and innovative hardware support for handling SIMD divergence.

The GPU programming model abstracts SIMD operations by presenting each physical thread as a multitude of logical threads, equal to the SIMD width. Consequently, GPU programs are written to operate on scalars rather than vectors, with numerous logical threads generated to populate the SIMD width and execute multiple physical threads.

To distinguish between the two thread types, physical threads are termed *warps*, while the term *thread* specifically refers to logical threads. In NVIDIA GPUs, there are 32 threads in 1 warp.

NVIDIA GPUs supported Instruction Set Architecture (ISA) is called Parallel Thread Execution (PTX). All PTX instructions are SIMD instructions operating on a single warp. One of the challenges of such approach is the handling of branches.

2.6.2 Single Instruction Multiple Threads (SIMT)

The GPU programming model allows threads within the same warp to have the flexibility of taking different paths in the program. However, owing to the underlying SIMD architecture, these divergent paths must be consolidated into a single path for the warp's execution. A distinctive attribute of GPUs is that this consolidation occurs dynamically during execution, performed by the hardware itself. Processors that implement this innovative capability are referred to as having Single Instruction Multiple Threads (SIMT) architecture.

When the processor detects divergence among thread paths, it selectively disables certain threads and executes instructions along one path. Subsequently, it deactivates the remaining threads and executes instructions along the alternative path. While similar execution can be achieved on SIMD processors lacking SIMT architecture, it necessitates explicit implementation within the assembly code.

2.6.3 GPU architecture

GPUs include various functional units integral to their execution processes. CUDA cores, for instance, are responsible for executing fundamental integer and single-precision floating-point operations. Each core possesses the capability to complete one such operation per cycle, with 32 operations required to fully execute an instruction.

Special Function Units (SFU) handle more complex arithmetic operations, including reciprocations, reciprocal square roots, and transcendental functions. Generally, each SFU unit is equipped to conclude one such operation per cycle in most scenarios.

Warp schedulers play a pivotal role in instruction issue. During each issue cycle, each scheduler has the capability to issue one or two instructions, choosing from a pool of warps assigned to it. Notably, a different warp can be selected for issue each time, highlighting the flexibility of this mechanism.

Each architecture within a GPU comprises multiple Streaming Multiprocessors (SMs), general-

purpose processors with a targeted clock rate and a modest cache size. The primary function of an SM is to execute multiple thread blocks in parallel. Upon completion of one thread block's execution, the SM proceeds to process the next thread block in sequence. Generally, SMs support instruction-level parallelism but do not incorporate branch prediction.

To fulfil its purpose, an SM incorporates the following components:

- Execution cores, including single-precision floating-point units, double-precision floating-point units, and SFUs. Caches:
- L1 cache to reduce memory access latency;
- Shared memory to allow shared data access among threads;
- Constant cache responsible for broadcasting reads from read-only memory;
- Schedulers for warps, responsible for issuing instructions to warps;
- A number of registers, essential for handling multiple active threads simultaneously.

2.6.4 GPU programming

GPU programming requires either a domain specific language or a set of Application Programming Interfaces (APIs). One of such languages is Compute Unified Device Architecture (CUDA). CUDA is a proprietary and closed-source parallel computing platform and API. It enables software to utilise specific types of graphics processing units GPUs for general-purpose processing. Operating as a software layer, CUDA offers direct access to the GPU's virtual instruction set and parallel computational elements to execute compute kernels. Differently from FPGA programming, CUDA does not require multiple days to compile.

2.7 Related Work

In the context of real-time adaptive radiotherapy, there are two main reasons why state-of-the-art fails to meet real time requirements. In the context of MRgoART, algorithms like those presented in [MP17], primarily focus on algorithmic complexity, often implemented in high-level languages, and without sufficient consideration of the hardware on which they are executed. This results in algorithms that do not fully exploit the potential of CPUs, GPUs, or other accelerators. This thesis addresses this gap by bridging the gap between HPC and algorithmic development, offering an alternative to meet MRgoART time requirements. The following sections discuss this in greater depth.

Similarly, for MC simulations, Ziegenhein et al. [ZPPKO15] analyse the limitations of current state-of-the-art approaches. In their analysis, while significant efforts have been made to accelerate MC simulations using GPUs, in recent years, the reported speedups often need to be interpreted cautiously. Most of these improvements result from comparisons between non-optimized, single-threaded implementations on outdated CPU-systems and multi-threaded implementations on modern professional GPUs. This thesis aims to surpass state-of-the-art methods by first determining the best platform for MC simulations, followed by developing a customized implementation capable of meeting the time requirements for real-time adaptive radiotherapy. Further details on this are provided in the subsequent sections.

2.7.1 Magnetic Resonance Imaging

In the context of MRgoART, both acquisition and reconstruction times are limited therefore scans are inherently undersampled [LRR⁺08]. However, to deliver precision radiotherapy, it is required to minimise the amount of artefacts in the images. For example, incorporating motion estimation into image reconstruction can achieve artefact-free high quality reconstruction from more highly undersampled data. The MoCo-HDTV algorithm, was used to reconstruct 20 respiratory phases from only 240 radial spokes for 60 slices with an in-plane resolution of 1.5mm and a slice thickness of 5mm, with an average reconstruction time of 6.5 hours [RHB⁺17]. As

this reconstruction time would not be suitable for MRgoART, a deep-learning model called DRACULA [FGCN⁺21] has been trained to reproduce the results of the MoCo-HDTV algorithm, producing 4D-MRI of dimensions 256 x 256 x 16 in 28s. The use of XD-GRASP in MRgoART was investigated by Mickevicius and Paulson [MP17] where they reconstructed six respiratory phases for 40 slices with an in-plane resolution of 1.77mm and a slice thickness of 6mm in over 10 minutes. Feng et al. [FTO20] proposed the use of XD-GRASP for MR signature matching, where they reconstructed 10 respiratory phases for 48 slices with an in-plane resolution of 1.25mm and a slice thickness of 5mm in 74 minutes. The reconstruction times for these methods are too slow for MRgoART, leading to Challenge 1: “To accelerate an MRI reconstruction method that meets MRgoART timing requirements.”

2.7.2 Performance modelling

PPT-GPU [ABC⁺19] is a scalable and accurate simulation framework that enables GPU code developers and architects to predict the performance of applications in a fast and accurate manner on different GPU architectures. The framework supports models for different memory hierarchies found in GPUs and latencies for different instructions. The results indicate that the predicted performance of PPT-GPU is within a 10 percent error compared to real devices.

GCoM [LHL⁺22] is an accurate GPU analytical model which faithfully captures the key core-side stall events of modern GPUs. Through detailed microarchitecture-driven GPU core modelling, GCoM accurately models modern GPUs by revealing the following key core-side stalls overlooked by the existing GPU analytical models. First, it identifies the compute structural stall events caused by the limited per-sub-core functional units. Second, it exposes the memory structural stalls due to the limited banks and shared nature of per-core L1 data caches. Third, it correctly predicts the memory data stalls induced by the sectorized L1 data caches which split a cache line into a set of sectors sharing the same tag. Fourth, it captures the idle stalls incurred by the inter- and intra-core load imbalances. The authors evaluate GCoM using an NVIDIA RTX 2060 configuration and show that it greatly improves modelling accuracy by achieving a mean absolute error of 10.0% against Accel-Sim cycle-level simulator [KSAR20], whereas the

state-of-the-art GPU analytical model achieves a mean absolute error of 44.9%.

Choi et al. [CRKB20] propose a methodology for end-to-end performance modelling of distributed GPU applications. Their work strives to create performance models that are both accurate and easily applicable to any distributed GPU application. They combine trace-driven simulation of MPI communication using the TraceR-CODES framework with a profiling-based roofline model for GPU kernels. They make substantial modifications to these models to capture the complex effects of both on-node and off-node networks in today's multi-GPU supercomputers. They validate their model against empirical data from GPU platforms and also vary tunable parameters of their model to observe how they might affect application performance.

Goswami et al. [GSB22] propose a machine learning-based two-stage estimator for assisting in designing an FPGA-based CNN accelerator. The Global Predictor assists in making accurate estimates of FPGA resource requirements and design latency from CNN architecture and hyperparameters expressed in Python. This assists in choosing a subset of high validation accuracy and feasible designs for mapping onto FPGAs. The Detailed Predictor assists in making accurate estimates of FPGA post-route resource requirements, design latency, and final clock period for the chosen subset of designs after applying HLS level (pragma and frequency) optimisations. This proposed estimation methodology enables a software engineer to obtain rapid and accurate estimates of the final implementation Quality of Results without executing FPGA design flows. The model was trained and tested for Xilinx Zynq Ultrascale + and Kintex-7 devices with an average prediction error of less than 9% and 4%, respectively

Da Silva et al. [DSBDT13] propose a performance model for FPGAs that is able to visually condense all the helpful information for the designer. Their proposed model extends the roofline model by considering the resource consumption and the parameters used in the HLS tools to maximise the performance and resource utilisation within the area of the FPGA. The proposed model is applied to optimise the design exploration of a class of window-based image processing applications using two different HLS tools. The results show the accuracy of the model as well as its flexibility to be combined with any HLS tool.

Voss et al. [VKM⁺21] proposed a design methodology for developing complex applications tar-

getting reconfigurable hardware, relying on an analytical estimation of system performance and area utilisation. The methodology involves analysing the application, identifying the parts suitable for hardware acceleration, and implementing them as representative software models. With the results of the application analysis, a suitable system architecture is devised, iteratively refined, and evaluated until the final version satisfying the specification requirements is obtained. The methodology was validated using a convolution neural network (VGG-16) and a high-performance computing application (BQCD), achieving state-of-the-art performance within 15% of modelling estimations.

The methods highlighted above have limitations that prevent the forecasting of MC performance, leading to Challenge 2: “To forecast accelerator performance so that the architecture with the highest performance for MC simulations can be chosen”, which aims to address this research gap.

2.7.3 Monte Carlo frameworks

Amadio et al. [AAB⁺23] implemented an Electromagnetic Shower MC simulation that is similar to DPM. They also accelerated the simulation on a GPU. Their study does not provide benchmarks and runtime measurements but they show that the geometry is one of the main bottlenecks of the simulation, which limits the device utilisation to a small fraction of the theoretical maximum. In their study, this inefficiency is inherent to the geometry model and is irreducible, even after performing a number of optimisations;

Sharma et al. [SKF⁺18] developed a fast MC simulation tool using the MC-GPU framework for personalised dose estimation in Computed Tomography (CT). Leveraging multi-threaded x-ray transport capabilities, scanner-specific geometry, and vocalised patient models, the tool generates realistic radiation dose estimates. In a demonstration of its effectiveness, they applied the tool to derive scanner-specific organ dose estimates for abdominopelvic CT scans using 58 adult XCAT patient models. Comparisons with data from a previously published tool based on the PENELOPE framework revealed that their simulation tool produced dose estimates within 5% of published values for organs within the primary field, achieving speedups of up to

70x compared to CPU cluster-based execution models. The combination of high accuracy and substantial speedups make their model a promising approach for rapid and personalised dose estimation;

Li et al. [LDW⁺21] extended and validated the GPU-MC package, gDPM, for Elekta Unity on-line plan verification, demonstrating its capability for accurate and efficient online independent dose verification for MR-LINAC systems. The implementation of electron/positron propagation physics within a uniform magnetic field was integrated into the existing GPU-MC dose engine, gDPM. Initially, the accuracy of dose calculations within the magnetic field was assessed using a heterogeneous phantom with EGSnrc. Subsequently, the dose engine underwent commissioning on a Unity machine with a virtual two-photon-source model, followed by comparison with the Monaco treatment planning system. Fifteen patient plans across five tumour sites were incorporated to quantify the accuracy and efficiency of online dose verification.

Perko et al. [PSDH⁺23] proposed DoTA, a framework that integrates CNNs for extracting spatial features, such as tissue and density contrasts, with a transformer self-attention backbone facilitating information routing between the sequence of geometry slices and a vector representing the beam's energy. The model is trained using 80,000 different head and neck, lung, and prostate geometries to predict low-noise MC simulations of proton beamlets. DoTA achieves remarkable performance, predicting beamlet doses in 5 ± 4.9 milliseconds. It surpasses analytical pencil beam algorithms in both precision and speed, offering MC-level accuracy 100 times faster for pencil beams. The model computes full treatment plan doses in 10–15 seconds.

Voss et al. introduce a reconfigurable hardware architecture designed for MC-based simulation of physical dose accumulation in intensity-modulated adaptive radiotherapy. Their ultimate objective is to enable real-time, accurate dose calculation during patient treatment, facilitating the broader adoption of personalised patient therapies. Their proposed architecture harnesses the inherent parallelism of MC simulations to execute domain decomposition and achieve high-resolution simulations without constraints imposed by on-chip memory capacity. Using a Xilinx VU9P platform and the Xilinx Alveo U250, they demonstrated that three VU9P-based cards or two Alveo U250s suffice to meet the real-time target of one second.

The current state-of-the-art dose delivery reconstruction methods fall short of meeting real-time adaptive radiotherapy requirements. Therefore, Challenge 3: “To devise a DPM that meets real-time adaptive radiotherapy timing requirements,” aims to address this issue.

2.8 Summary

This chapter provides an overview of the algorithms employed in real-time radiotherapy along with a description of the underlying physics. Specifically, it introduces the DPM and XD-GRASP algorithms, highlighting their significance in the field. Additionally, the challenges associated with FPGA and GPU implementations are discussed. The chapter also examines related works that have demonstrated promising results in this domain.

Chapter 3

Magnetic Resonance Imaging

3.1 Introduction

This Chapter presents an exploration of accelerating image reconstruction for MRgoART. It delves into the criticality of rapid and accurate image reconstruction for real-time treatment adaptation during radiotherapy sessions, emphasising the computational demand challenges. The work showcases novel strategies devised to address these challenges, including efficient parallelization and acceleration schemes tailored to significantly enhance the throughput of state-of-the-art iterative methods.

MRI has long been used in cancer diagnosis for its remarkable soft-tissue delineation and its ability to capture images from diverse orientations, offering a wide range of contrasts [SP15]. Recent advancements have extended its application to radiotherapy treatment planning, particularly through the emergence of Hybrid MR-linacs. These integrated systems, pairing an MRI scanner with a linear accelerator for radiation treatment delivery [LRR⁺08, FMR⁺09, KBC14, MLC⁺16].

MR-guided radiotherapy fuses MRI imaging with radiation delivery, enabling precise tumour targeting while preserving healthy tissue in what is known as MRgoART. This adaptive technique allows immediate treatment adjustments during sessions, relying on acquired images for organ positioning. For thoracic cancers, emphasis lies on respiratory-motion resolved 4D-MRI,

vital for safeguarding organs like the heart and lungs during the respiratory cycle. However, obtaining these images presents challenges due to necessary undersampled image acquisition.

State of the art iterative techniques attempt to recover high-quality images from undersampled data, however they fall short in meeting the crucial time constraints for MRgoART utilisation [MP17, SPT18]. *MRgoART requires rapid image reconstruction within a **couple of minutes** remains a critical challenge despite its advantages [HHO⁺18]*. section 3.3.7 addresses this challenge proposing an architecture capable of meeting MRgoART time requirements.

Cutting-edge iterative methods are hindered by their substantial computational demands, leading to slow performance. *State-of-the-art implementations available in the literature **fall significantly short** of meeting the minimum throughput prerequisites for MRgoART*, compared to the minimum requirements the throughput that they achieve is more than one order of magnitude lower. Consequently, this chapter aims to solve the constraints associated with conjugate gradient methods by introducing parallelisation and acceleration schemes applicable to any 4D-MRI data. These enhancements are applicable to other MRI reconstruction methods like stack-of-stars or stack-of-spirals patterns hence they are not limited to the XD-GRASP algorithm.

Our results show that these proposed enhancements achieve a order of magnitude improvement over the current state-of-the-art methods, thereby enabling the integration and adoption of MRgoART. This approach could also be used in real-time adaptive radiotherapy as it can be extended to time-resolved 4D-MRI methods based on XD-GRASP, such as Magnetic Resonance SIGnature MATching (MRSIGMA) or Single projection driven real-time multi-contrast (SPIDERM) [FTO20, HCX⁺22]. Given the fast reconstruction times showed and the possibility to tune the accuracy to further increase the performance. Integration in Artificial Intelligence (AI)-based reconstruction schemes is also possible. AI-based methods are promising for real time adaptive radiotherapy due to their excellent performance and image accuracy [FGCN⁺21, TMVvdB23].

The main contributions of this chapter are:

- A parallelisation scheme optimised for 4D-MRI data, to reduce multi-threading overhead and hide data transfer latency;
- An efficient architecture capable of meeting online time requirements of MRgoART to facilitate clinical use;
- An heterogeneous implementation faster than the state of the art capable of exploiting multiple GPUs

The remainder of this chapter is as follows: section 3.1 provides an introduction to the MRI reconstruction and MRgoART. Section 3.2 delves into the architecture’s intricacies, including data acquisition and preparation (section 3.2.1), data preparation strategies (section 3.2.2), parallelism (section 3.2.3), load balancing (section 3.2.4), non-uniform fast Fourier transform (NUFFT) (section 3.2.5), oversampling factor (section 3.2.6), Amdahl’s law, scalability, and limitations (section 3.2.7), and approximate implementation (section 3.2.8). Section 3.3 presents the experimental results, covering system configuration (section 3.3.1), data acquisition (section 3.3.2), test configuration (section 3.3.3), validation (section 3.3.4), reconstruction experiments (section 3.3.5), reconstructed images and validation (section 3.3.6), measured speed-ups (section 3.3.7), and scaling (section 3.3.8). Section 3.4 discusses the findings, while section 3.5 provides a summary of the chapter.

3.2 Architecture

MRgoART imposes both time and quality constraints. Reconstruction algorithms should not be sensitive to undersampling artefacts while minimising the reconstruction time. These characteristics allow for reducing the acquisition time as fewer samples are required to reconstruct high-quality images. While several conjugate-gradient iterative algorithms can meet the quality requirements, such as Iterative Self-Consistent Parallel Imaging Reconstruction (SPIRiT), Sensitivity Encoding solved with a Conjugate Gradient Algorithm (CG-SENSE) and XD-GRASP [MP17], some of them are not capable of fully leveraging compute resources offered by

present-era computer architectures and accelerators. Modern CPUs incorporate multiple multi-threaded cores, furthermore, GPUs are massively parallel processors hence suitable algorithms should benefit the most from the parallelisation opportunity.

Mickevicius and Paulson conducted a comparative analysis of 4D-MRI reconstruction methods [MP17] with a focus on parameters such as overall image quality, reconstruction time, artifact occurrence, and the accuracy of motion estimations. They observed that the XD-GRASP algorithm [FAC⁺16] was least sensitive to undersampling artefacts. Consequently, the XD-GRASP algorithm was chosen to be parallelized and expedited. This choice was informed by its ability to yield clinically acceptable images with relatively brief acquisition times, thanks to its robustness against undersampling artifacts. From an HPC perspective, the XD-GRASP reconstruction is one of the best candidates for parallelisation as it can fully leverage the 4D-MRI data format: it can independently reconstruct different slices (multiple Two Dimensional (2D) images with a temporal dimension which, when stacked, compose a 4D image).

3.2.1 Data acquisition and preparation

To execute the reconstruction utilising the XD-GRASP algorithm, the data were obtained through a 3D stack-of-stars acquisition on an MR-linac, with relevant MRI acquisition parameters detailed in section 3.3.2. In this acquisition pattern, k-space data is sampled using a 3D radial pattern, where k-space is sampled at equidistant k_z -locations before transitioning to the next angle in the k_x - k_y plane. At each k_z location, a line of k-space (spoke) is acquired, traversing the centre of the k_x - k_y plane. Subsequently, the angle is incremented by the golden angle, and the process is reiterated [BCM⁺14, WSK⁺07]. Before applying the XD-GRASP algorithm to the data, a gradient delay correction is implemented using the Trajectory Auto-Corrected image reconstruction (TrACR) algorithm [IG16], and the requisite coil sensitivities are estimated.

3.2.2 Data Preparation Strategies

While the XD-GRASP reconstruction is computationally expensive, the strategies employed for binning the k-space data into respiratory phases [SPT18] can significantly impact image quality and execution time. The term “*data binning*” refers to the technique of subdividing the respiratory signal into various respiratory phases or bins. In this process, the acquired k-space data in free-breathing MRI is partitioned into different motion states or respiratory phases, typically denoted as $b = 1, 2, 3, \dots, n$. This segmentation enables the resolution of respiratory motion to an acceptable extent. Two distinct techniques for extracting the respiratory signal from raw data and two binning methods were implemented, resulting in four data preparation strategies.

Baseline The strategy, as described in the XD-GRASP implementation, served as a baseline for comparisons. In this approach, the central k_z -projection of the acquired k-space data was extracted, and a Fast Fourier Transform (FFT) was applied along the z-axis. Principal Component Analysis (PCA) was then applied across different receive coil channels, and the resulting values were used as the respiratory signal. The original k-space data was sorted based on the signal and divided into independent respiratory bins, ensuring each k-space spoke was present in only one bin [FAC⁺16].

Angle-dependent Correction of the Respiratory Signal (ADERS) ADERS is a method for extracting the respiratory signal from k-space data. The process starts with sorting the magnitude signal of the k-space centre based on the acquisition angle. Subsequently, a background subtraction is executed, relying on the moving average of the data sorted by the radial acquisition angle. Following this angle-dependent correction, an estimation of the data quality acquired by each receive coil is conducted [GBK⁺13]. The best coils were then used to perform PCA and extract the respiratory signal. In the experiments of section 3.3, 50% of the coils were retained. The k-space data was sorted, and each angle was assigned to a single bin, preserving the number of spokes allocated per respiratory bin.

Overlapping Respiratory Bins (ORB) ORB uses the baseline respiratory signal extraction (direct PCA) to obtain respiratory motion. The k-space data was sorted as before, but each spoke contributed to two adjacent respiratory bins. Unlike the data preparation of the Moco-HDTV algorithm [RHB⁺17], spokes corresponding to the most extreme points of the respiratory signal (max. inhalation and max. exhalation) contributed to only one respiratory bin. While this approach did not alter the number of reconstructed respiratory phases, each phase was based on more data.

Combined This strategy uses both the combined angle-dependent respiratory signal correction and overlapping respiratory bins.

3.2.3 Parallelism

Parallelisation strategies that minimise multi-threading overhead on CPUs and GPUs have been studied for many years [Vol16, SAL, BR92]. However, they focus only on one aspect of the problem, they either minimise shared memory latency, communication latency or GPU memory access. The parallelisation strategy proposed here slightly modifies the mathematical formulation of XD-GRASP. By doing so, it is possible to minimise multi-threading overhead, PCIe data transfers, GPU memory access and hide data transfers with computation. The mathematical details of the XD-GRASP algorithms are illustrated in section 2.

After the data preparation, the XD-GRASP algorithm intuitively performs a 3D transformation from a pre-sorted Five Dimensional (5D) signal to a 4D-MRI:

$$XD-GRASP-4D : \{s, p, nx, a, c\} \rightarrow \{s, p, y, x\} \quad (3.1)$$

where:

- s: slices acquired along the z-axis
- p: number of reconstructed respiratory phases

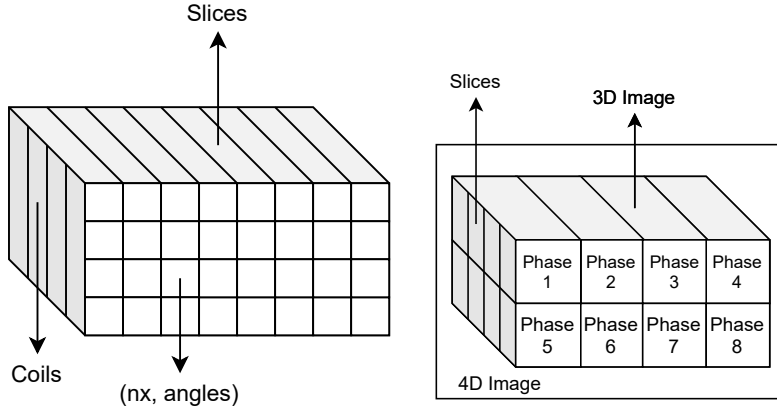


Figure 3.1: The input and output of the XD-GRASP algorithm.

- nx : number of points acquired per acquisition trajectory in a spoke
- a : the different angles of spokes constituting each slice
- c : the number of receiving coils
- x, y are the coordinates of the pixels in the reconstructed image.

Figure 3.1 illustrates the data format. The left structure is replicated multiple times per respiratory phase, creating a 5D input, while the second structure corresponds to the full output. Due to the nature of this 4D input data, each input slice contributes solely to one output slice. Thus, it's viable to divide the 4D transformation into a sequence of 3D transformations along the z -axis:

$$\{s, XD-GRASP-3D : \{p, nx, a, c\} \rightarrow \{p, y, x\}\} \quad (3.2)$$

Here, s , p , nx , a , c , x , and y retain their definitions from Equation 3.1. Implementing the reconstruction as a series of independent 3D reconstructions yields superior performance compared to a single, unified 4D reconstruction. This approach offers several advantages:

- there are no synchronisation barriers between threads;
- local cost function and early termination;
- independent use of external accelerators;
- threads are created once at the beginning of the program and used until their termination.

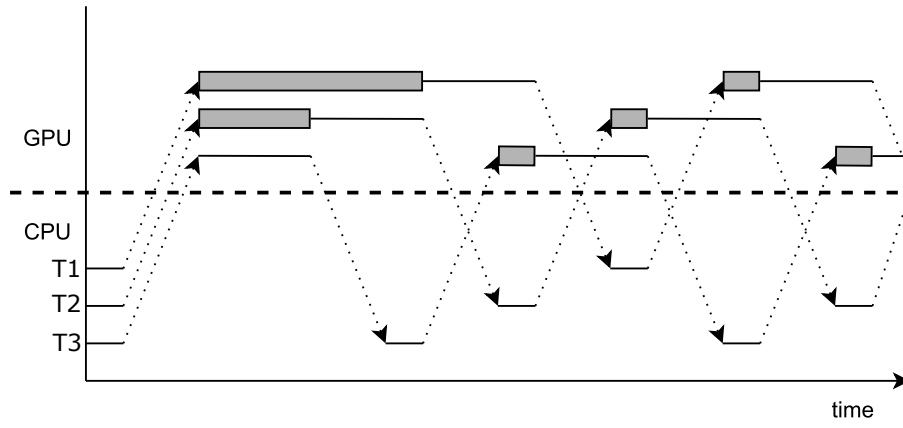


Figure 3.2: XD-GRASP architecture timing diagram. Dimensions are not representative of real-time duration.

Alternate 3D formulations where the outermost loop iterates over phases are less efficient as they necessitate either sharing NUFFT operators or communication between threads, leading to reduced performance.

In the case of a CPU-only implementation, this multi-threading model requires support from the underlying FFT libraries. Fortunately, both FINUFFT and FFTW are thread-safe, with the planning phase of FFTW being the only potential concern. This issue can be addressed by executing the planning within a critical section. The planning phase of FFTW, involving cost-model evaluations and micro-benchmark execution, marginally impacts the scalability of this parallel implementation. To enhance scalability, pre-computing and saving on permanent storage the planning phase, which is executed once per system, could be considered, thereby increasing the scalability of this parallelization strategy.

In the heterogeneous implementation, where GPU accelerators are present in the system, the thread-safe nature of both cuFINUFFT and the CUDA runtime library eliminates the necessity for explicit critical conditions. Consequently, there is no limiting factor in scalability due to any unsafe planning phase. This parallelization approach allows simultaneous access to the PCIe bus, facilitating the concurrent scheduling of read and write operations to and from the accelerator while computing another slice.

Figure 3.2 is a timing diagram that shows the overlap strategy when three CPU threads, T1, T2 and T3, are created. The GPU is called multiple times during each iteration of the conjugate

gradient. The lines represent the computation time, the rectangles represent the waiting time for the data to be processed, and the arrows show the data transfers. When only one thread is used, the communications between CPU and GPU are not overlapped meaning there are times when the CPU is idle waiting for the GPU to finish and vice versa. When more than one thread is created, communications can be overlapped and, in cases where a sufficient number of threads are created, the full overlap is achieved. While the CPU performs other steps of the XD-GRASP algorithm, the GPU performs NUFFT, ideally achieving full utilisation without idling time.

In our tests, the execution of cuFINUFFT is dominated by the compute time with the transfer time accounting for less than 1% of the total time. Section 3.3 shows that this is particularly noticeable when multiple GPUs are used as the PCIe bandwidth is constant and shared between multiple GPUs while the compute capabilities multiply.

3.2.4 Load Balancing

Earlier implementations employed a parallel XD-GRASP version, utilising a parallelization scheme akin to the one employed in this research. They resorted to multi-processing to ensure thread safety, which was lacking in the NUFFT libraries utilised. Consequently, they delved into MRI data analysis to devise a static load-balancing technique. However, subsequent updates in FINUFFT and FFTW introduced support for thread safety, eliminating the necessity to distribute the workload across distinct processes. Moreover, the transition from gpuNUFFT to the more recent cuFINUFFT introduced thread-safe capabilities. In this architecture, the entire computational workload is parallelized using OpenMP [DM98].

Nonetheless, prior load-balancing techniques maintain relevance within the domain of multi-threading. Specifically, these strategies can be leveraged by the OpenMP *static* scheduler, resulting in slightly reduced runtime multi-threading overhead. The reconstruction times for slices within our datasets exhibit differences of less than 1% of the total execution time. Therefore, each thread reconstructs a contiguous partition comprising multiple statically assigned slices. If the number of slices is not a perfect multiple of the number of threads, the remaining

slices are equally divided among a subset of threads, causing a minor workload imbalance that cannot be avoided.

Resolution of this imbalance could be feasible through internal parallelization of the 3D reconstructions. However, findings from prior research discourage this approach, as it leads to a significant surge in multi-threading overhead, to the extent where thread creation and destruction overshadow the reconstruction time.

3.2.5 Non Uniform Fast Fourier Transform (NUFFT)

MRI data acquisition typically follows a stack-of-stars pattern in k-space, utilising a golden angle spacing method [BCM⁺14, WSK⁺07]. This technique results in non-uniformly sampled data, necessitating reconstruction algorithms to rely on the NUFFT to transform the acquired k-space data into images. For the proposed architecture, the chosen libraries are *FINUFFT* [BMaK19] and *cuFINUFFT* [SWA⁺21], providing the necessary functionalities required by the XD-GRASP algorithm.

While these libraries are well-suited for general 2D and 3D cases, the challenge arises with 4D-MRI data, which introduces a time dimension. Moreover, XD-GRASP, similarly to other reconstruction algorithms, operates as an iterative conjugate-gradient algorithm. Consequently, with most NUFFT implementations handling parallelism internally, this setup is not optimal for iterative conjugate-gradient algorithms as each invocation of the NUFFT leads to thread-creation and destruction. This greatly increases the multi-threading overhead and results in a severe performance penalty: most of the time is spent managing threads instead of performing useful computations. In the GPU case, the library allocates the GPU memory, transfers to the GPU, executes the NUFFT, transfers the data back to the CPU and frees the GPU memory. However, this causes several issues, as the reconstruction requires multiple 2D NUFFTs, these cannot fully leverage the parallelism offered by modern GPU as 2D data is not enough to fill all the compute cores available. Additionally, leaving control to the library does not allow for scheduling data transfers to hide them during the computation. Finally, the memory required at each execution is the same and some data is constant across different iterations. Hence,

allocations can be done once at the beginning of the execution and frees can be done once at the end.

To increase the performance of the NUFFT, it is possible to pre-compute part of the transform and reuse these values in multiple invocations. FINUFFT creates a plan by executing FFTW planning [FJ98], pre-computes the kernel Fourier transform, bin-sorts and stores the nonuniform points. cuFINUFFT on the other hand, allocates GPU memory, pre-computes the kernel Fourier transform, bin-sorts the nonuniform points and transfers them to the GPU. These pre-computation steps, especially in the GPU case, require a non-negligible amount of time. Redoing the pre-computation multiple times severely impacts performance. Fortunately, both *FINUFFT* and *cuFINUFFT* provide an advanced interface that allows low-level control over pre-computations, data transfers and parallelism. To achieve high performance, our proposed architecture disables any sort of intrinsic parallelism offered by FFTW, FINUFFT, and cuFINUFFT fully taking control of multi-threading and data transfers to minimise overhead and hide data transfers. To do so is possible to compile FINUFFT without OpenMP [DM98]. This disables parallelism at the library level entirely, this allows user control over parallelism. In the case of cuFINUFFT, the advanced interface available can be used with data directly allocated on the GPU; leaving the data-management and transfers entirely to the user. This allows to schedule and interleave multiple NUFFT with data transfers. Different threads can perform transfers and queue GPU NUFFT executions concurrently.

A possible alternative to executing NUFFT on CPUs and GPUs is FPGA acceleration. FPGA acceleration does not offer compelling advantage compared to GPU and CPU and was not considered; both FINUFFT and cuFINUFFT offer high throughput capable of meeting MR-goART time requirements when combined with present-era CPUs and GPUs. FPGAs require significant more engineering effort to be programmed, the lack of open source, ready to use NUFFT implementations lead to the exclusion of FPGA acceleration for this problem.

3.2.6 Oversampling Factor

The NUFFT is the most computationally expensive operation with a complexity of $O(MJ^2 + \mu^2 N \ln(\mu^2 N))$ [SLG⁺09], where M is the number of k-space samples acquired, N the pixel-size of the reconstructed image, J and μ the oversampling factor and the regridding kernel size used in the NUFFT implementation. Some NUFFT implementations allow to specify the desired relative precision. When the precision is fixed the choice of μ determines the tradeoff between the time spent by regridding kernel compared to the time spent performing the FFT. Depending on the precision requested, the choice of regridding kernel, and the FFT implementation used, the choice of μ significantly impacts performance. For more details on μ , Barnett et al. [BMaK19] provide an in-depth explanation of the NUFFT algorithm, its implementation and the use of μ . Larger values increase both memory and computation requirements. For instance, reconstructing a two-dimensional dataset oversampled twice requires four times more memory than the original dataset as the size of inner FFT is four times as the original dataset. Beatty et al. [BNP05] experimented with different μ values and indicated that $\mu = 1.25$ is a sufficient oversampling factor in MRI contexts.

In our evaluations, we assessed performance and accuracy using both the commonly used $\mu = 2$ and, where supported, $\mu = 1.25$. However, the cuFINUFFT library, being relatively recent, does not yet support $\mu = 1.25$. It is worth noting that this limitation only impacts GPU results, FINUFFT who is CPU-only supports $\mu = 2$ and $\mu = 1.25$. Our results did not indicate a measurable difference in image quality between $\mu = 1.25$ and $\mu = 2$, but we observed a significant increase in performance when using $\mu = 1.25$ compared to $\mu = 2$.

3.2.7 Amdahl's law, Scalability and Limitations

To measure the efficiency of the proposed implementation, the theoretically achievable times were estimated using Amdahl's law [Rod85] for our CPU and GPU models in Equations (3.3) and (3.4), assuming that the number of CPU cores was superior or equal to the number of slices to reconstruct.

Based on the previous observations, the theoretical speedup achievable was evaluated, according to *Amdahl's law*. The calculation assumed that data are already loaded in memory. This is a fair assumption since data can be acquired from the scanner directly to Dynamic Random-Access Memory (DRAM). In the CPU case:

$$\begin{aligned}
 S_{latency}(s) &= T(Preparation) + T(XD-GRASP-4D) \\
 &= T(Preparation) + T(threading) + \frac{T(XD-GRASP-4D)}{n} \\
 &= T(Preparation) + T(threading) + T(plan) + \\
 &\quad + \max(T(XD-GRASP-3D))
 \end{aligned} \tag{3.3}$$

where:

- $T(Preparation)$ is the time needed to extract the respiratory signal from the k-space and sort it accordingly;
- $T(threading)$ is the time needed to create and destroy the threads;
- $\max(T(XD-GRASP-3D))$ is the time of the slowest 3D reconstruction;
- $T(plan)$ is the time needed to perform NUFFT planning.

From section 3.3, measuring the time using *CPU 1.25* and baseline data preparation method leads to $S_{latency}(s) = 9.42s + T(Preparation)$. 9.42s is the measured time needed to process the single slowest slice. The architecture proposed here does not support parallelism inside the slice processing. This means that if future architectures offer a number of cores greater than the number of slices acquired, this architecture will stop scaling. $T(Preparation)$ is left as a variable as it is not part of the XD-GRASP algorithm. The user can choose any suitable data preparation strategy based on input format, time and quality requirements. Section 3.3 provides performance and quality measures for four different possible strategies.

The quantity of slices necessary for reconstruction depends on the dimensions of the field of view along the superior-inferior axis and the slice thickness. In our dataset, the field of view measured 200mm, aligning with the maximum treatment field of 220mm established by the

MR-linac. Subsequent applications might necessitate a larger field of view to include extensive organs at risk, like the lungs. In such scenarios, optimising performance involves ensuring that the number of slices aligns as a multiple of the available CPU cores, enabling equitable distribution of workload across these cores.

When using accelerators, Equation 3.3 is different as it needs to take data transfers into account:

$$T(XD-GRASP-3D) = T(CPU) + T(transfers) + T(Accelerator) \quad (3.4)$$

where:

- $T(CPU)$ is the time to execute the not accelerated portions of the reconstruction;
- $T(Accelerator)$ is the time needed to perform the NUFFT on the accelerator (e.g. GPU);
- $T(transfers) = T(send) + T(receive)$ is the time needed to transfer the data back from the accelerator

$T(send)$ and $T(receive)$ can be further expanded since the amount of data transferred for each invocation is known:

$$T(send) = \frac{|\{iterations\}| * sizeof(type) * coils * slices * nx * angles}{bandwidth}$$

$$T(receive) = \frac{|\{iterations\}| * sizeof(type) * coils * phases * Y * X}{bandwidth}$$

where:

- $|\{iterations\}|$ is the estimated number of iterations needed for the conjugate-gradient method to converge;
- $sizeof(type)$ is the size in bytes of the data type used. In our case, 4 bytes single precision floating point;
- The remaining parameters are dataset dependent.

As stated above, *coils, slices, nx, angles* are dataset dependent and vary depending on the MRI acquisition parameters. The number of iterations is also dependent from the dataset, as XD-GRASP is a conjugate-gradient iterative algorithm, it terminates once the cost function falls below a threshold. Although, reconstructions on the same dataset, using the same parameters are deterministic, they require the same amount of iterations, different datasets will require different number of iterations. By substituting the values taken from the dataset as *coils, slices, nx, angles* and measuring the number of iterations until convergence on the dataset used in section 3.3 and using PCIe 3x16 lanes with 15.75 GB/s theoretical bandwidth for the data transfers we obtain $T(send) + T(receive) = 1.14s$.

Under Amdal’s assumption of infinite parallelism $T(Accelerator)$ is equal to 0 and the CPU is a bottleneck to the computation. In our architecture $T(CPU)$ can be executed in parallel in a heterogeneous way and performs the remaining operations that do not involve the NUFFT and $T(Preparation)$. This time was measured using the architecture proposed on the dataset used in section 3.3. This results in $T(CPU) = 0.81s$

$$S_{latency}(s) = 1.14s + 0.81s + T(Preparation) = 1.95 + T(Preparation)$$

Under these assumptions, our proposed architecture is PCIe-bound as the data transfers cannot be overlapped and hidden by the chosen parallelisation scheme.

3.2.8 Approximate Implementation

We propose an alternative implementation that adjusts specific algorithm parameters to enhance throughput. Employing a parallelization scheme akin to the presented architecture, this variant diverges in several key aspects. It harnesses gpuNUFFT for GPU computation, employs multi-threading in favour of multiprocessing, and utilises socket-based data transfer to the original process. The decision to utilise for processes over threads stems from gpuNUFFT’s absence of thread safety, resulting in marginally increased overhead during process creation. The diminished accuracy observed in GPU-generated results arises from the selection of parameters,

mainly μ , Sector Width (SW), and Kernel Width (KW), for the gpuNUFFT computation, which could be fine-tuned to bolster image precision or throughput. Table 3.1 shows how various combinations of KW, SW, and μ impact gpuNUFFT’s performance and accuracy, gauged by normalising execution times against the chosen configuration (highlighted in bold). Despite their omission from section 3.3 due to falling short of the requisite accuracy threshold (99% or higher SSIM), these results are discussed upon in section 3.4.

Table 3.1: The NUFFT benchmarks ordered by speed-up.

μ	KW	SW	SSIM	Speed-up
1.25	1	16	93%	1.14
2.00	1	16	91%	1.10
1.25	1	8	93%	1.09
2.00	1	8	91%	1.04
1.25	3	16	97%	1.00
2.00	3	16	97%	0.99
1.25	3	8	97%	0.94
2.00	3	8	99%	0.93

This approximation could be applied on CPU and FPGA too. However, FINUFFT and cuFINUFFT do not allow direct control over SW and KW, hence this could not be directly tested. As section 3.3 will show, it is possible to meet time requirements without leveraging these approximations.

3.3 Experimental Results

The proposed algorithm underwent evaluation on a high-end consumer system, featuring an Intel i9-11900K 8-core CPU paired with an NVIDIA RTX A5000 GPU. Evaluations were conducted both with and without GPU acceleration. Notably, input data reading times were excluded from the evaluation metrics as, in real-world scenarios, data are typically loaded directly into DRAM during the acquisition phase. This is a valid assumption as data can be transferred directly to DRAM during acquisition.

3.3.1 System Configuration

The Intel i9-11900K is a 64-bit 8-core x86 microprocessor operating on 2 contexts and based on the 14nm Rocket Lake micro-architecture. This processor features a 16 MB 16-way set associative inclusive L3 shared cache and operates with a Thermal Design Power (TDP) of 125W. It functions at a base clock speed of 3.5 GHz, with an all-core boost clock of up to 4.80 GHz and can reach up to 5.3 GHz for fewer active cores. The NVIDIA RTX A5000 is an enterprise-grade GPU built on the 8nm Ampere micro-architecture. It boasts 24GB GDDR6X memory and operates with a TDP of 230W. This GPU comprises 8,192 CUDA cores running at a base clock speed of 1.170 GHz, with a boost clock frequency of up to 1.695 GHz.

Compilers and compiler flags

The benchmarks were conducted on the described system configuration. MATLAB R2021b executed the MATLAB code, while the CPU code was compiled using GCC 11.1.0, and the GPU code was compiled with CUDA compilation tools 11.4. The NVCC flags for the GPU code were the defaults provided by *cuFINUFFT*, specifying architectural (SM_86) and code-generation flags for different GPU micro-architectures. GCC flags utilised were **-Ofast -march=native**, ensuring the generated binary maximises CPU usage of vector instructions, leveraging the widest vector units available in the system. Specifically, the AVX512 long vector instructions available in recent Intel architectures, including the one employed in this study, were utilised. The implementation is open-source, allowing anyone to compile it with optimised flags.

3.3.2 Data acquisition

Four patients diagnosed with pancreatic adenocarcinoma, aged between 56 and 76, participated in the PRIMER trial (NCT02973828), undergoing imaging procedures using a 1.5T MR-linac system (Elekta AB, Stockholm, Sweden) at various stages of radiotherapy treatment. The current imaging protocol similar to [PAC⁺20] involved a volumetric radial stack-of-stars gradient echo sequence with golden angle spacing and balanced MRI contrast, acquiring 831 radial

spokes in 287 seconds. Additional parameters included:

- Sequences: Volumetric radial stack-of-stars gradient echo sequence with golden angle spacing
- Acquisition time: 287 seconds for 831 radial spokes
- Coil configuration: Vendor-provided coils, four anterior and four posterior channels
- Treatment conditions: Three patients received 36-40 Gy over 15 fractions, and one patient received 50 Gy over 28 fractions
- Compression belt: Used in 9 out of 17 datasets
- Imaging Parameters: SPAIR fat-suppression [GSH⁺14], partial Fourier factor of 0.7, over-sampling factor of 1.61 along the Cartesian k_z dimension
- Echo time: Reduced to 1.34 ms using partial echo sampling
- Other MRI parameters: Repetition time 3.5 ms, field of view $500 \times 500 \times 200$ mm³, flip angle: 40°, bandwidth 866 Hz/px, acquired voxel size: $1.5 \times 1.5 \times 3.0$ mm³
- Corrections and estimations: Gradient delay correction applied [IG16], coil sensitivities estimated [WGM00]

The acquisition parameters were identical to the clinical protocol used for daily volumetric re-contouring and re-planning of MR-guided pancreatic cancer treatments at the Institute of Cancer Research [GWL⁺23].

3.3.3 Test configuration

All experiments were conducted with 5 runs, and the average results were computed across these runs. Standard deviation (STD) was calculated for the measured values. As discussed in section 3.2, the implementation proposed here integrates a complete end-to-end process consisting of

data preparation and subsequent reconstruction. Given the diversity in preparation strategies outlined in section 3.2, each strategy is explicitly specified.

Henceforth, whenever data is presented regarding the data preparation strategy, preparation time, processing time, total reconstruction time, or speedup, it will always be specified. In instances where these specifics are not explicitly stated, the reference is consistently made to the baseline version.

The reconstruction time evaluation involves reconstructing the dataset with each preparation strategy five times and averaging the timing results across the runs. These timings are assessed on the high-end consumer system illustrated in section 3.3.1. The reference MATLAB implementation, based on Feng et al. [FAC⁺16], is used for comparison. Both the C++ and CUDA codebases are compiled using GCC 11.1.0 and NVCC 11.4 using the flags illustrated in section 3.3.1.

To investigate the bottlenecks and scalability of the accelerated implementations, a single dataset is reconstructed using the baseline strategy on two other systems. This includes an AMD Ryzen 3700X CPU paired with DDR4-3600MHz RAM for the CPU version and a multi-GPU system housing three NVIDIA 1080Ti GPUs for the GPU version.

3.3.4 Validation

The validation process involves assessing the SSIM [WBSS04] between the reference images and our reconstructed images. SSIM serves as a metric to measure image similarity by evaluating structural information degradation. It emulates the perceptual capabilities of the human visual system in recognizing and measuring differences between images [WBSS04]. Images are deemed equivalent if they achieve an SSIM of 1 or 100%. We consider images that achieve 99% SSIM or above as equivalent, acknowledging that slight disparities may arise due to the inherent non-associative and non-commutative nature of floating-point arithmetic.

3.3.5 Reconstruction experiments

Different strategies employed to bin the data have a direct influence on the resulting reconstructed images, primarily determined by the translation from data to respiratory phases [SPT18]. To explore this impact the following experiments use the conventional *Baseline* and *ADERS* strategy presented in section 3.2.2. The acquired data are categorised into respiratory bins or alternatively assigned to overlapping respiratory bins leading to the ORB and *Combined* strategies explained in section 3.2.2.

3.3.6 Reconstructed images and validation

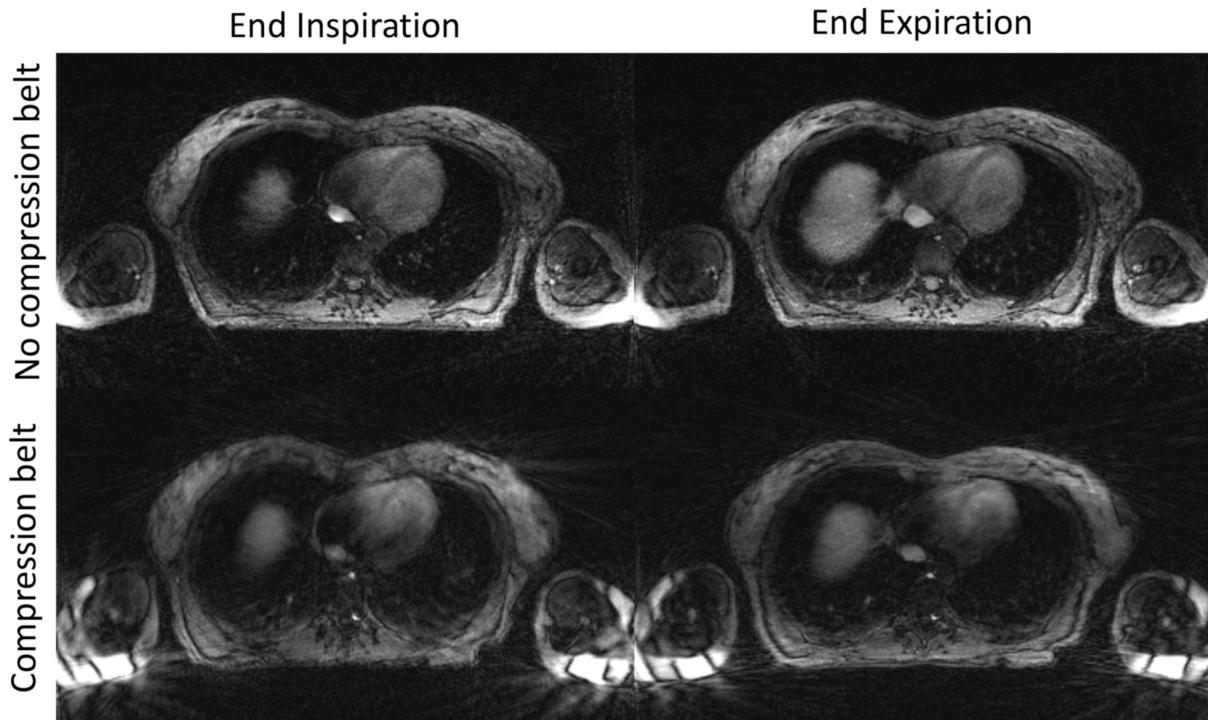


Figure 3.3: Slices reconstructed at both the end-inspiration and end-respiration phases exhibit distinct respiratory states. The variations in respiratory phases are discernible through the liver’s dome size in both images. Despite noticeable streaking artefacts stemming from the arms at the field-of-view periphery, they do not compromise the exceptional clarity in visualising the internal anatomy.

As shown in Figure 3.3, the 4D images acquired with and without the compression belt exhibit noticeable differences, particularly with the images acquired using the compression belt appearing more blurred. However, upon reconstruction using the proposed fast implementations,

the reconstructed images were indistinguishable from the reference MATLAB implementations. Each reconstruction attained an SSIM of 0.99 when compared with the image reconstructed using the same data preparation in MATLAB.

Figure 3.4 displays specific anatomical details obtained from a singular acquisition utilising abdominal compression, a routine technique employed at the Institute of Cancer Research for pancreatic cancer radiotherapy. Among the methods assessed, images acquired using the baseline binning strategy exhibited a comparatively blurrier appearance. Notably, employing a higher amount of data per bin (in ORB and combined methods) rendered non-moving structures sharper.

3.3.7 Measured speed-ups

Figure 3.5a illustrates the time required for each data preparation strategy. The original MATLAB implementation demonstrated a negligible time difference compared to the optimised CPU and GPU implementations in this part of the code, where GPU acceleration was absent. Consequently, the timings of the optimised implementations fall within the margin of error.

In Figure 3.5b, the total processing time of various implementations on the Intel system is showcased. With non-overlapping bins (baseline, ADERS), images were reconstructed on average in 1060 ± 26 seconds using MATLAB, 241 ± 3.3 seconds employing CPU with $\mu = 2.0$, two minutes using CPU with $\mu = 1.25$, and one minute with the heterogeneous CPU+GPU implementation. Reconstruction times escalated with overlapping bins (ORB) due to increased data processing. Lowering the oversampling factor from $\mu = 2.0$ to $\mu = 1.25$ halved the reconstruction time. As delineated in Figure 3.5c, the GPU version exhibited the most substantial speedup, surpassing 17 times the baseline MATLAB version's speed.

Section 3.2.7 presented the ideal speedup for the architecture proposed in this chapter. The ideal reconstruction time forecasted by Amdahl's law excludes $T(\text{Preparation})$, as although necessary, it is not technically part of the XD-GRASP algorithm. For a CPU-only implementation, the ideal reconstruction time predicted by Amdahl's law is 9.42s, whereas the fastest measured

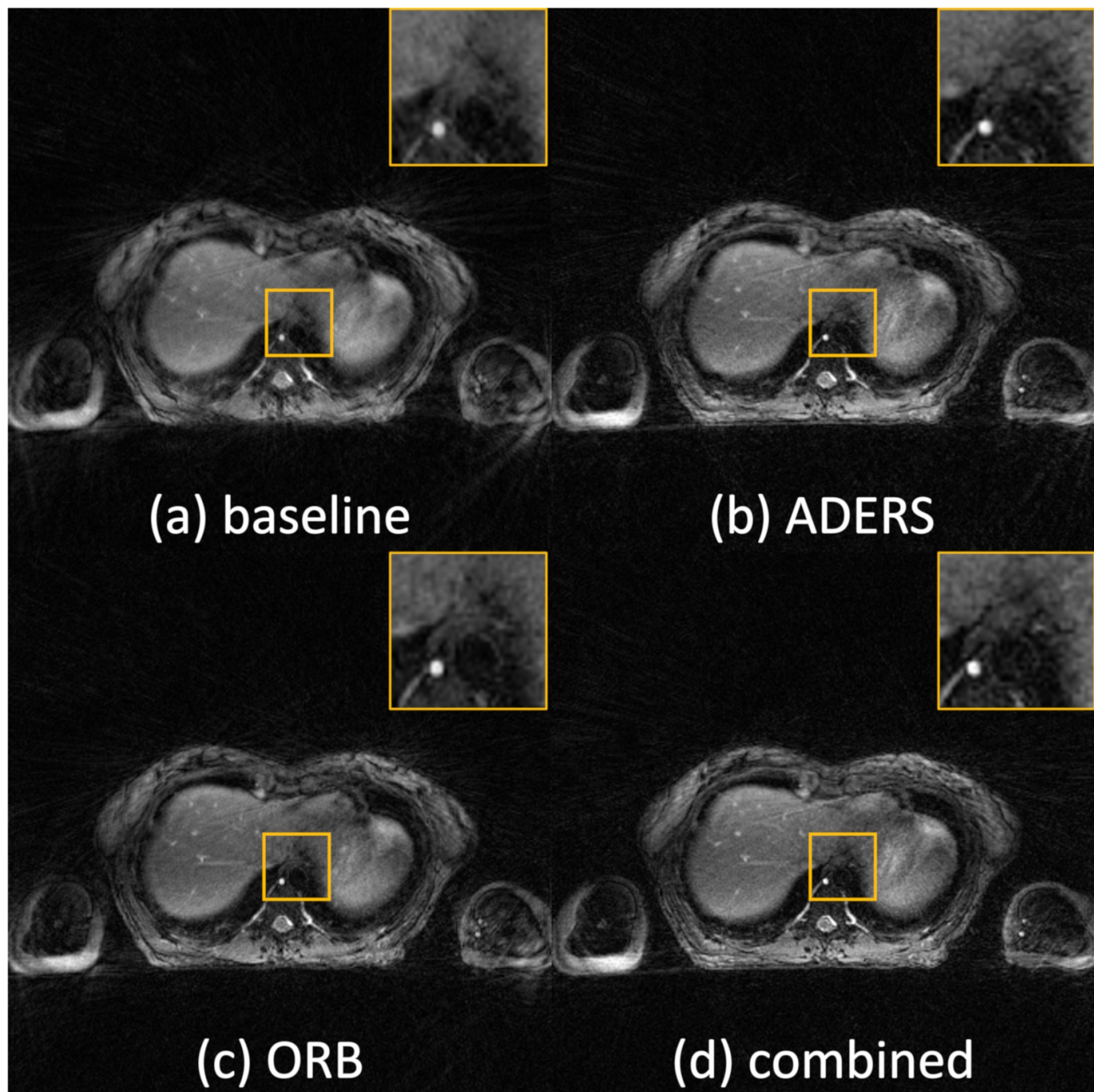


Figure 3.4: The end-inspiratory phase of the 4D-MRI obtained under abdominal compression was reconstructed using various data preparation strategies from the identical acquisition. Fine details within non-moving structures exhibit enhanced sharpness in (b), (c), and (d) in contrast to the baseline.

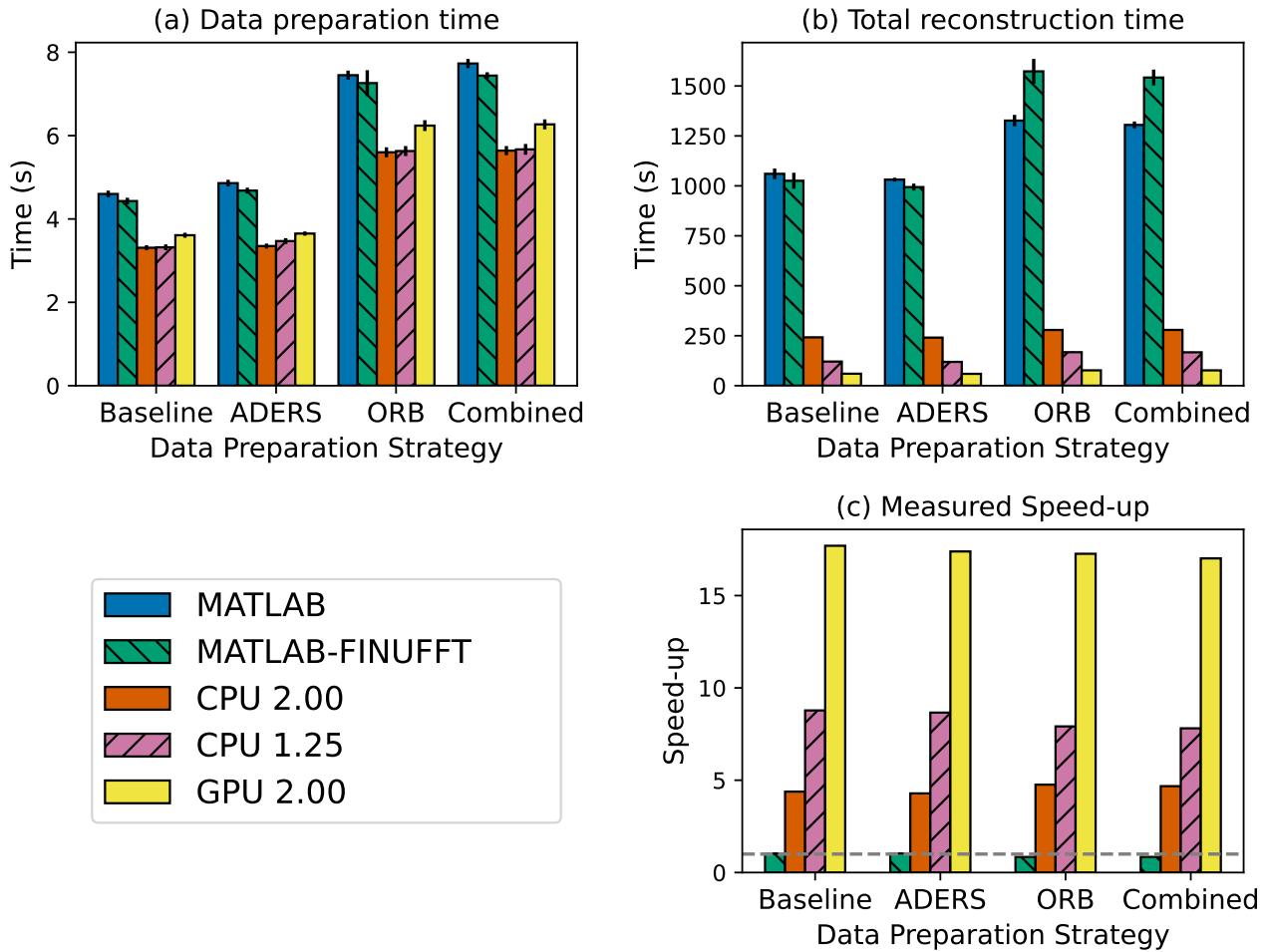


Figure 3.5: The timing and speed-up results for various data preparation strategies utilising different implementations. Both the data preparation (a) and overall reconstruction times (b) increased notably with the ORB and combined strategies. Across all cases, the GPU implementation showcased more than a 17-fold improvement compared to the MATLAB reference implementation (c), achieving image reconstruction times of 60s (baseline, ADERS) and 77s (ORB, combined). Notably, the MATLAB implementation employing the FINUFFT bindings exhibited slower performance than the original version for ORB and combined strategies, with speed-ups falling below 1 (dashed line).

time is 120s. This discrepancy does not indicate inefficiency in the current implementation but rather highlights the potential for it to be up to 12 times faster, provided a number of cores equal to or greater than the number of slices is available. In the case of GPU, the theoretical minimum reconstruction time is 1.95s, compared to the achieved reconstruction time of 60s. However, achieving the theoretical minimum would require a GPU with near-infinite parallelism capable of computing all the strengths and Fourier coefficients of the NUFFT simultaneously.

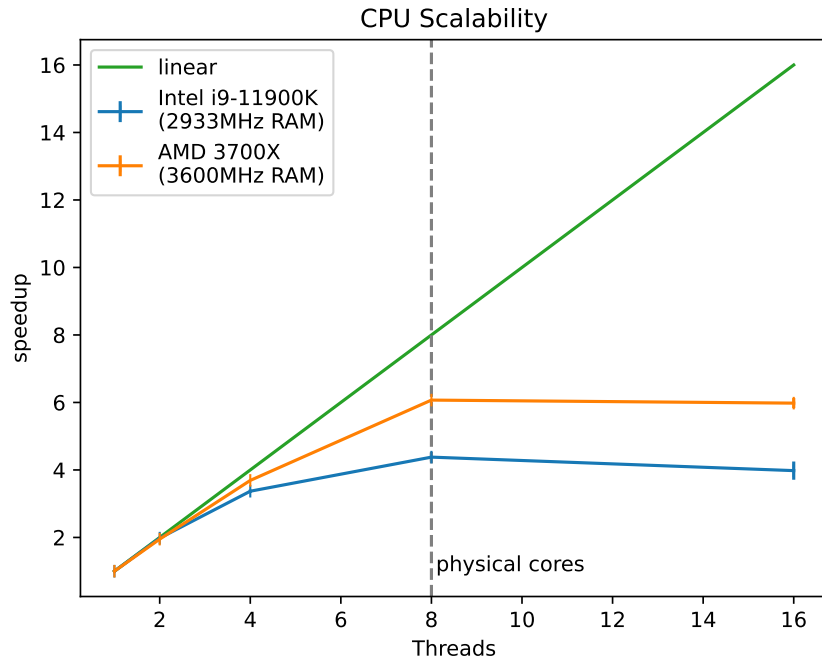
The implementations proposed here, CPU 2.00, CPU 1.25 and GPU 2.00 all meet MRgoART time requirements as they all require less than two minutes. This means that even institutions

that do not have GPUs available, can still leverage this architecture to meet time requirements. This also lowers the adoption requirements as GPU might be expensive, or violate some power or cooling requirements. CPU 2.00 is included for completeness as 2.00 is often the default oversampling factor used in many NUFFT implementations. As section 3.2.6 showed and section 3.3.6 validated, there is no advantage in using this architecture. CPU 1.25 is the architecture that is recommended when GPUs are not available. GPU 2.00 shortens the time even further, this has two main advantages, it is possible to reduce the overall treatment time, minimising patient discomfort, or it is possible to use more involved data preparation or post-processing strategies to boost image quality.

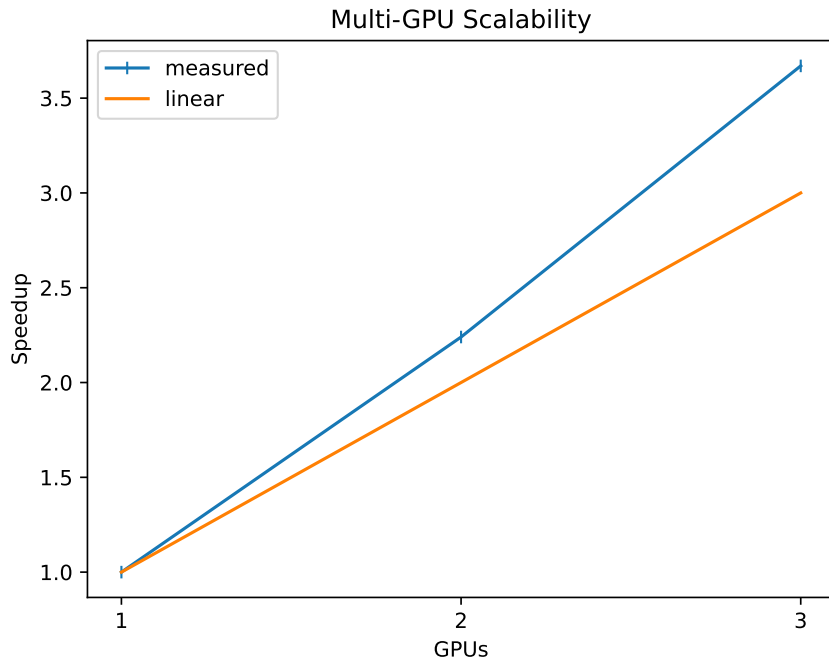
3.3.8 Scaling

Figure 3.6a illustrates both the measured and linear speed-ups in the CPU scenario. Despite both the Intel and AMD systems exhibiting similar single-core performances, the AMD system demonstrated a higher speed-up, potentially due to its superior RAM bandwidth. The observed multi-thread speed-ups align with the benchmarks conducted by the FINUFFT library developers [BMaK19]. Their study suggested that for a precision level of 10^{-3} , access time to kernel data in RAM dominates execution time compared to the theoretical Floating Point Operations Per Second of the CPU.

Furthermore, the observed speed-up in the system employing multiple GPUs, as demonstrated in Figure 3.6b, exceeded the theoretical linear speed-up. This occurrence can be attributed to increased CPU utilisation achieved by the presence of multiple GPUs. In this specific implementation, the bulk of the processing time occurs on the GPUs. Consequently, employing faster or multiple GPUs reduces CPU idle time, enhancing overall system utilisation and achieving super-linear hyperscaling.



(a) Relative CPU speed-up according to the number of CPU cores



(b) Relative GPU speed-up according to the number of GPUs used for the reconstruction.

Figure 3.6: The straight line represents the theoretical speed-up. Analysis of the CPU scaling experiment reveals suboptimal scaling on both testing systems, with the system featuring faster RAM access demonstrating improved scaling. In the GPU scenario, multiple GPUs reduce CPU idle time, leading to increased CPU utilization and super-linear hyperscaling, with actual speed-up surpassing theoretical limits.

Table 3.2: Comparison with state-of-the-art implementations

Implementation	CPU/GPU Cores	Speed (GHz)	Pixels/s
GPU approx (this chapter)	5,120	1.455	1,319,720
GPU 2.00 (this chapter)	8,192	1.170	1,029,697
CPU 1.25 (this chapter)	8	3.5	489,157
Minimum throughput	-	-	262,144
CPU 2.00 (this chapter)	8	3.5	240,109
Reference	8	3.5	52,495
Mickevicius et al. [MP17]	40	2.3	14,269
Feng et al. [FTO20]	8	NR	7,281

3.4 Discussion

Efficient integration of advanced reconstruction methods like 4D-MRI into online adaptive radiotherapy workflows necessitates short reconstruction times. Through the implementation of parallelization and high-performance computing techniques, our open-source designs significantly reduced the 4D-MRI reconstruction time while preserving image quality. Leveraging GPUs, our system achieved reconstruction times as short as one minute.

The definition of time constraints for online adaptive radiotherapy remains loosely defined in existing literature. Feng et al. indicated that for typical matrix sizes of $256 \times 256 \times 48$ voxels and 10 respiratory phases, online treatment adaptation requires a reconstruction time below two minutes [FTO20]. This criterion translates to a minimal throughput of 262,144 pixels/second. The optimised CPU and GPU implementations proposed in this chapter surpassed this threshold, positioning them as promising contenders for MRgoART. Comparative analyses with state-of-the-art XD-GRASP implementations, detailed in Table 3.2, revealed that most previous approaches fell short of meeting the throughput requirement. Notably, our approximate CPU +GPU (GPU approx) implementation was fastest, achieving a speed-up of 11x and an SSIM of 0.97 compared to the same MATLAB reference implementation. However, this was executed on a high-end system containing two AMD EPYC™ 7551 processors and an NVIDIA TESLA V100 GPU, moreover the SSIM achieved does not meet MRgoART requirements.

Alternative strategies for performance enhancement, such as calling FINUFFT directly from MATLAB or using automated conversion tools like MATLAB Coder™, were explored. However,

these methods failed to yield performance gains or caused longer reconstructions when overlapping bins were used. In all cases, employing multi-threading enhanced application performance but exhibited sub-linear scaling dependent on RAM speed. This aligns with findings by the FINUFFT library developers, indicating that RAM access times to the kernel data, rather than CPU operation speeds, primarily determine NUFFT execution times at a precision of 10^{-3} .

As enterprise-grade systems with high-core counts are integrated into clinical settings [PAC⁺20], reconstruction times for inter-slice parallelised algorithms are anticipated to decrease. This reduction would lead to speed-ups nearing those predicted by Amdahl’s law, ultimately improving workflow efficiency.

Trade-offs between reconstruction time and image quality were observed when employing various data preparation strategies. Doubling the data in each respiratory bin could lead to more temporal blurring, whereas the ORB and combined methods produced sharper images. The reduced undersampling artefacts and lower total variation score between adjacent respiratory bins likely mitigated the smoothing effects of regularisation in the latter methods.

The selection of optimal contrasts for reconstruction significantly impacts performance. While our study focused on data reconstructed with balanced contrast and fat suppression, clinical practice often involves acquiring multiple contrasts in the initial session and using the contrast offering the best tumour and organs-at-risk visibility during treatment sessions. This variability in contrasts could influence reconstruction times in our optimised implementations.

Respiratory-correlated 4D-MRI holds significant applications in MR-guided radiotherapy for measuring respiratory motion, providing input for time-resolved 4D-MRI, and computing delivered doses [SPT18, STdS⁺16, MMN⁺20]. These principles extend to image-guided proton therapy, where interplay effects and Bragg peak depth necessitate even lower image reconstruction latencies [ODM⁺17, GPL⁺22]. Our exploration focused on inter-slice parallelism for the XD-GRASP algorithm, applicable to other MRI reconstruction methods like stack-of-stars or stack-of-spirals patterns. This approach could also be extended to time-resolved 4D-MRI methods based on XD-GRASP, such as MRSIGMA or SPIDERM [FTO20, HCX⁺22]. AI-based reconstruction schemes were compared, indicating excellent agreement between our implemen-

tations and the reference implementation [FGCN⁺21, TMVvdB23].

The rapid reconstruction times achieved by our open-source implementation, even without GPU acceleration, make 4D-MRI suitable for online adaptive radiotherapy. This advancement renders 4D-MRI an attractive solution for monitoring intra-fraction motion in MR-guided radiotherapy, reducing uncertainties in tumour position and enabling online adaptive gated treatments and real-time tracking without fiducial markers [BSS⁺07]. Notably, this could benefit lung, liver, pancreas, and oligometastatic disease treatments [PWO⁺22].

3.5 Summary

In this chapter, we explored the integration between HPC and the challenges faced in MRI, particularly in the context of radiotherapy. The pivotal role of MRI in radiotherapy emerged, highlighting its integration into MR-linacs and hybrid MR-integrated proton therapy, driven by the need for superior soft-tissue contrast for MR-guided online Adaptive Radiotherapy (MRgoART) [SP15, LRR⁺08, FMR⁺09, KBC14, MLC⁺16, HOM⁺20, ODM⁺17]. We delved into the challenges of 4D-MRI reconstruction, tackling issues like undersampling and artefacts, and explored solutions such as compressed sensing and comparisons between reconstruction methods [SPT18, DR93, LDP07, MP17, FAC⁺16, PAC⁺20].

Then we presented optimisations for 4D-MRI reconstruction, delving into strategies like multithreading, CPU +GPU implementations, and their potential impact on clinical settings [FTO20, HHO⁺18, PAC⁺20]. We discussed trade-offs between reconstruction time and image quality, emphasising the influence of performance and the promising applications of fast 4D-MRI in MR-guided radiotherapy, opening doors for adaptive treatments [PWO⁺22, BSS⁺07]. In particular, we proposed CPU 2.00, CPU 1.25 and GPU 2.00. All of these implementations meet MRgoART time requirements, however there are some differences between them. The first two architectures only leverage CPU. CPU 2.00 is included for completeness as the oversampling factor default value two in most NUFFT implementations. CPU 1.25 is the architecture that is recommended for general use, when GPUs are not available. This is also cheaper to

adopt as it does require only cheaper consumer grade CPU to meet time requirements. In case GPUs are available, GPU 2.00 is the best option as it offers the best performance. This lowers the reconstruction time even further, enabling the use of more involved data preparation or post-processing strategies to enhance image quality.

Looking ahead, we analysed future enhancements in MRI reconstruction, the role of AI-based schemes, and potential applications beyond radiotherapy, highlighting ongoing advancements and room for further exploration [FGCN⁺21, TMVvdB23, BCM⁺14, LHHN03, SPT18]. This highlighted the critical need for optimised computing techniques in addressing MRI challenges, ultimately paving the way for real-time adaptive treatments and enhanced disease management.

This work has already been adopted by the Magnetic Resonance Imaging in Radiotherapy Group at the Institute of Cancer Research for research purposes, where it is utilised to explore and develop new reconstruction algorithms. Recently, the Center for Computational Mathematics at Flatiron Institute, Simons Foundation in New York expressed interest towards this implementation of the XD-GRASP algorithm as it leverages FINUFFT [BMaK19] that they developed.

Chapter 4

Accelerating Monte Carlo Simulations

4.1 Introduction

This chapter presents a comprehensive study about modelling the performance of workloads when executed on accelerators such as GPUs and FPGAs. The methodology presented here is general-purpose in the sense that there is no limitation on the workload that it can analyse. However, as the scope of this thesis is MC simulations the focus of this methodology and its evaluation is on these types of simulations and physics related workloads. MC simulations are known to be computationally expensive and hard to speed up due to their stochastic behaviour. Moreover, programming accelerators presents many different challenges that developers need to overcome to successfully accelerate any application. To avoid investing needless development time and resources into accelerating applications that do not benefit from acceleration, it is important to model and forecast the performance of FPGAs and GPUs.

Since MC simulation is considered to be the most accurate method to calculate dose distribution in radiotherapy [SWB00] the reconstruction is implemented using computationally expensive MC methods. In the medical physics context, the simulation computes the trajectories and dose depositions of many individual particles traversing biological tissue by sampling particle interactions according to the fundamental laws of physics. Due to the stochastic nature of MC algorithms, a large number of particles have to be simulated. The simulation of each particle is

computationally expensive due to the high complexity of the interaction sampling [ZPPKO15]. Hence, MC algorithms, even if accurate are not directly usable for real-time situations.

To achieve real-time performance the use of accelerators needs to be considered since recent results show that both GPUs and FPGAs can achieve significant speedup compared to CPUs. GPU shows promising results, achieving a speedup of over 7x for Leading Order calculations on GPU compared to CPU using the MadFlow framework [CCMRZ21]. Opticks is an open-source initiative aimed at enhancing optical photon simulations by seamlessly incorporating NVIDIA GPU ray tracing capabilities through NVIDIA Opticks into simulations based on the Geant4 toolkit. Opticks shows impressive performance gains: a single NVIDIA Turing architecture GPU can achieve optical photon simulation speedup factors exceeding 1500 times the performance of a single-threaded Geant4 simulation [Bly21]. FPGAs also show promising results. Voss et al. implemented a simplified Electromagnetic Scattering simulation on an FPGA that compared to CPU achieves a speedup of over 4x [VZV⁺19].

Since accelerating MC simulations demands considerable engineering effort, our study introduces a methodology for MC acceleration. This approach involves an initial analysis of the simulation to determine whether there exists significant potential for performance enhancement before proceeding with the acceleration. Additionally, this methodology aims to minimise the frequency of executing place and route, a lengthy FPGA compilation process that can take several days to complete. Our findings demonstrate that our model effectively predicts the performance of MC on both a Xilinx VU9P FPGA and NVIDIA GeForce RTX 3090 Ti GPU, offering a reliable assessment of whether the simulation stands to benefit from acceleration. In contrast to blindly prototyping an accelerated simulation, this approach has the potential to save substantial engineering effort by allowing early termination in cases of limited achievable performance and by guiding developers in designing the accelerated architecture.

The primary contributions of this chapter include:

- Introducing a methodology for accelerating MC simulations applicable to both, GPU and FPGA accelerators;

- Providing a comprehensive explanation of the methodology derivation process, enabling its potential extension to other accelerators like the Tensor Processing Unit (TPU);
- Conducting an evaluation of the proposed methodology using MC simulation involving Coulomb Scattering on a Xilinx VU9P FPGA and NVIDIA GeForce RTX 3090 Ti GPU.

Our approach presents a unique combination of features, incorporating a priori forecasts (before implementation), data transfers, and support for both FPGA and GPU accelerators, validated on MC simulations. In contrast, PPT-GPU [ABC⁺19] lacks integration of a priori knowledge and offers limited support for data transfers, focusing solely on GPU acceleration. Similarly, GCoM [LHL⁺22] provides GPU support with data transfer capabilities but lacks integration of a priori forecasts. Choi et al. [CRKB20] introduce an end-to-end approach with GPU support but without a priori forecasts or data transfers. Goswami et al. [GSB22] provide a priori forecasts and FPGA support but omit data transfers, supporting only CNN architectures. Da Silva et al. [DSBDT13] support FPGA accelerators with data transfers but have limited application scope, particularly lacking support for complex applications like MC simulations. Finally, Voss et al. [VKM⁺21] incorporate a priori forecasts, data transfers, and FPGA support, suitable across various domains; however, their approach lacks GPU support.

The remainder of this chapter is as follows: section 4.1 provides an introduction to the methodology. In section 4.2, the methodology for acceleration is discussed, detailing its components and functionalities. Following this, section 4.3 explains the application analysis process, examining its various aspects and requirements. Section 4.4 focuses on the software model for FPGA, outlining its structure and implementation. Section 4.5 explores operations count and usage in the context of the methodology. In section 4.6, highlights the performance model, with subsections dedicated to predicting I/O bandwidth usage, memory transfers, and FPGA performance modelling. Similarly, section 4.7 delves into GPU performance modelling, providing insights into the total time required for execution. Evaluation is presented in section 4.8, including by a specific examination of Coulomb Scattering, which includes application analysis, the FPGA performance modelling, the GPU performance modelling, with the respective implementations, validation, results and comparison between platforms. Discussion and summary of the findings

are presented in sections 4.9 and 4.10, respectively.

4.2 Methodology for acceleration

Incorporating accelerators into a system can be a challenging undertaking, as it presents various obstacles. As a result, it is essential to ascertain the potential advantages of acceleration in advance, determine whether the intended application can benefit from acceleration, and identify which accelerator can offer the optimal balance of performance, cost, and engineering efforts. Doing so is vital in meeting the performance, budget, and time requirements of a project. This section outlines a methodology based on the one originally proposed by Voss et al. [VKM⁺21] for FPGA and extends it to include both GPUs and FPGAs.

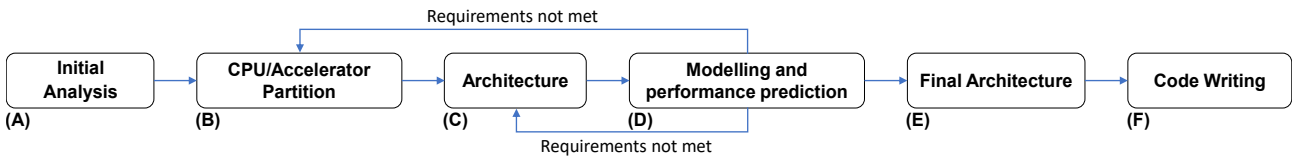


Figure 4.1: Proposed accelerator architecture development workflow

Figure 4.1 shows the proposed steps for accelerator architecture development. The proposed methodology is applied before writing a single line of accelerator code and aims to:

- determine whether the application is suitable for acceleration;
- the achievable speedup;
- which parts of the application need to be accelerated;
- estimate the engineering effort.

The resulting performance model forecasts the achievable speedup, while the architecture shows the partitioning between CPU and accelerator. The architecture provides also estimates on the engineering effort as it shows which portions of the applications need to be accelerated and which algorithms are needed.

The proposed workflow is iterative in nature and it requires incremental refinements until the requirements are met or further improvements are either impossible or not worth the engineering effort. It should be noted that the methodology provided in this section is intended for manual application by the developer and is not to be fully automated. However, certain tools, such as Intel OneAPI [One, CPL⁺95], can be employed to decrease development time, and with the availability of more advanced tools, automation of certain aspects may become possible. While the use of such tools is encouraged, it is recommended that developers verify their results to ensure the accuracy of the performance models.

Algorithm 2 Methodology for acceleration

- 1: perform initial **analysis** of the original application
 - 2: determine the target accelerator
 - 3: create initial partitioning in accelerator and software
 - 4: create first initial **performance model** and **architecture**
 - 5: create first initial **software model** ▷ Applies only to FPGA
 - 6: **while** bottleneck exists **do** ▷ Requirements are not met
 - 7: identify bottlenecks with new **architecture** or accelerator/CPU partitioning
 - 8: resolve bottlenecks with new **architecture** or accelerator/CPU partitioning
 - 9: verify new **architecture** in updated **software model** ▷ Applies only to FPGA
 - 10: perform further **analysis** as necessary
 - 11: refine **performance model** based on updated **analysis** and **architecture**
 - 12: start implementation
-

Algorithm 2 outlines the complete methodology with greater details compared to figure 4.1. The methodology begins with the analysis of the original application to identify its compute-intensive, data-intensive parts and workload characteristics such as accuracy requirements (line 1, step A). The results of this analysis aid in determining the most suitable accelerator for the workload (line 2, step B), and an initial partitioning between the CPU and accelerator is established (line 3, step B). Subsequently, the accelerator architecture is outlined, including the portion of the application executed by the accelerator, data allocation, and transfer (line 4, step C). A preliminary performance model is then constructed, which is platform-specific, and the specific details for each platform will be presented later in section 4.6 and 4.7.

The fifth step regards solely FPGA acceleration and involves the development of a software model that accurately simulates the hardware architecture (line 5, step D). This serves three purposes: firstly, it clarifies the functionality of each component of the architecture, and sec-

ondly, it simulates the precise behaviour of the hardware, which is beneficial for debugging the FPGA implementation, it greatly reduces the number of compilations as FPGA compilation might take several days due to place and route. Thirdly, it can be used to unit test the FPGA implementation; it is possible to execute all the functionalities in the software model, except one that can be executed on a real FPGA or the resulting hardware description can be simulated. The hardware description and the simulation tools depend on the toolchain used, for example, Google XLS [XLS] can generate x86_64 ASM that can be simulated by direct CPU execution, it can also generate Verilog which can be simulated with tools such as ModelSim. This is not necessary in the GPU case as GPUs do not require lengthy compilation processes such as place and route. Hence, it is possible to unit test and integrate smaller changes directly in the accelerator code.

Then, the iterative process starts. While a bottleneck exists, meaning that the performance requirements are not met, (line 6, step D) the developer has to first identify the type of bottleneck (line 7, step D). If the bottleneck is computational, it is possible to use better algorithms, better data structures, or even distribute over multiple devices, this does not involve changing the partition between accelerator and CPU so the arrow that goes to step 3 is followed. In case the bottleneck is I/O bound then it is possible to use compression, caching or accelerating data-generating portions of the workload (line 8, step D), this can involve changing the partition between CPU and accelerator so in this case the arrow that goes to step 2 is followed. Once the application is optimised it is necessary to update the performance model to account for the changes (lines 10-11, step D-E). If the requirements are met then it is possible to start the implementation (line 13, step F), otherwise, it is necessary to proceed to identify and resolve new bottlenecks (lines 7-11, step F).

The proposed methodology however has some limitations. In the FPGA case, the results are accurate only when using toolchains that allow for low-level control of the design. For example, writing directly VHSIC Hardware Description Language (VHDL) or using HLS toolchains such as MaxCompiler [PMTL13], which allows for such control. MaxCompiler mainly automates scheduling and performs optimisation aimed at reducing resource utilisation. The user still specifies the main design components, such as adders and multiplexers and their connection in

a dataflow fashion. However, the compiler can automate scheduling and offers a less verbose syntax compared to writing directly VHDL. In contrast, when using HLS toolchains such as Intel OneAPI and AMD Vitis HLS, their compilers can perform optimisations that can drastically lower the accuracy of the proposed methodology.

In the GPU case, the number of instructions is derived using approximations. In particular, x86 assembly instructions are used to estimate NVIDIA PTX instructions. While this leads to reasonable estimates in the proposed tests, changes to both CPU and GPU compilers can lead to inaccuracies in this methodology. We also point out that estimating the GPU performance is advantageous only in case of complex applications that require significant development effort to be accelerated. In the case of small applications or small functionalities, it might be faster to explore GPU designs directly.

When accelerating MC it is important to ensure the accuracy of the results. As MC results depend on random number generation, the results may vary even if the simulation and its accelerated version are both executed with the same initial configuration. It is possible that the accelerated version of the simulation uses random numbers in a different order to cause small variations in the results. Hence, to test the accuracy of MC, it is necessary a test that accounts for these small variations that can arise and verify that they are of no statistical significance. One such test is Kolmogorov–Smirnov test [BZ14], which when used with $\alpha = 5 \times \sigma$ [Lyo13] can be used to verify the correctness of the results. Section 2.2 gives more details about this test.

Section 4.8 will show how to apply the methodology and the accuracy of the forecasts.

4.3 Application analysis

The initial step of the methodology involves measuring various metrics to determine the most appropriate accelerator for the application. The metrics that need to be analysed are **cache miss rate**, **branch miss rate** and whether the application is embarrassingly parallel as these are the main factors that dictate the performance of an application. Based on these three

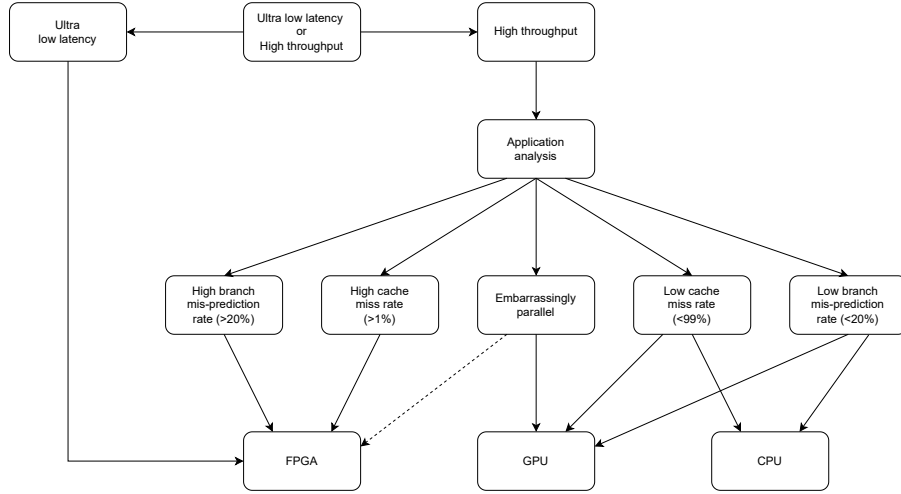


Figure 4.2: Application analysis deciding process. Arrows indicate mandatory requirements, the dashed arrow indicates an optional requirement.

factors, it is possible to determine which platform is the most suitable candidate. Figure 4.2 summarises the decision process. The starting point is the level of parallelism in the algorithm. If the algorithm is not embarrassingly parallel, i.e. the application does not have enough parallelism to saturate all the CUDA cores available on the GPU, GPUs are unlikely to offer a significant performance advantage. Then, the branch misprediction rate and cache miss rate need to be evaluated. The former refers to the frequency with which the CPU branch predictor incorrectly predicts the outcome of a branch, while the latter refers to the frequency with which the data is not found in the cache and must be retrieved from the main memory. Figure 4.2 gives some indicative values. The exact values to be considered depend on the specific device used as different micro-architectures offer different mispredictions and cache miss penalties. Section 4.3.1 explains how to measure both penalties. The value 80% assumes that branch misprediction penalties slow down the code by a factor of 5x. However, these values are not fixed and different architectures might require some adaptation. For example, figure 4.3 shows that a branch misprediction requires 5 times the clock cycles of a branch prediction, different CPUs might have a different misprediction penalty, in that case, the value 80% needs to be tuned with the correct cost of branch mispredictions. On the other hand, cache misses cause an order of magnitude slowdown in performance, hence, in this case, the reference value assumes that an application is cache-friendly if at least 99% of the accesses are cache hits. This value needs to be validated on the particular CPU. For example, AMD

3D V-cache technology [WAC⁺22] greatly increases the size of the L3 cache hence, workloads that achieve a high cache miss rate on other CPUs might achieve a low cache miss rate with this particular technology. In case the workload exhibits both a high cache miss rate and high branch misprediction rate FPGAs almost certainly are the accelerators that will perform best. This is due to the dataflow paradigm of FPGAs where both sides of the branch are present in hardware and a multiplexer picks the correct side, discarding the other computation with no overhead. High cache miss rate favours FPGAs due to their customizability. FPGA allows to design of a custom prefetcher that accounts for the memory access pattern that is able to load data in advance effectively hiding memory latency. Otherwise, if both these rates are low and the application is embarrassingly parallel GPU will offer the best performance. There is one last case to consider, when the application is embarrassingly parallel but it has only a high branch misprediction rate or only a high cache miss rate. In these cases, there is no clear winner between FPGAs and GPUs and further analysis is needed.

It is worth pointing out the assumptions and limitations of this approach. The methodology assumes GPU acceleration of embarrassingly parallel workloads able to exploit all the available CUDA cores from the beginning. Workloads that are not embarrassingly parallel but contain a lot of branches might not experience a major slowdown on GPU as the slowdown that section 4.3.1 will show, diverging branches could be executed by idle threads. However, the performance limitation of these types of workloads is that the overall parallelism is the same. For example, a workload that utilises 10% of the available GPU parallelism, it will still utilise the same 10% after a diverging branch; warps will be partially filled with threads executing useful work and threads executing no-ops to avoid injecting wrong answers in the results. Despite this, in some cases, these types of workloads might still be perform better on GPUs compared to FPGAs and a CPUs.

However, the methodology highlighted in this chapter cannot forecast performance of such workloads, as it would require a different method to account for the performance penalties of branches on GPU. In the case of FPGA acceleration, the methodology assumes that the body of the “if-else” is relatively small. When the body of these branches is a big computations encompassing thousands or tens of thousands of instructions, the analysis in Figure 4.2 will

underestimate the branching penalty on FPGA. Branching penalties on FPGA are incurred in space, as both sides of the branch needs to be implemented and executed with the wrong result discarded afterward. This space penalty reduces the parallelism available on the FPGA, thereby limiting performance and potentially making the FPGA slower than the alternatives, which would invalidate the methodology presented here. Nonetheless, in our experience, such cases are relatively rare and even previous results [VKM⁺21] support the accuracy of this methodology.

Lastly, FPGAs are the only viable choice in cases of strict ultra-low latency requirements or when jitter is not acceptable. Ultra-low latency requirements are usually in the order of nanoseconds to microseconds, and CPUs cannot meet these performance requirements as the kernel scheduling overhead is already in the order of milliseconds. Multi-threading in these cases only increases the overhead, as creating a thread requires tens of microseconds. Moreover, the system scheduler cannot be controlled, and even threads with the highest priority can be de-scheduled, causing jitter as sleeping threads need to be scheduled before performing the computation. Therefore, in systems such as the CERN L1 trigger system or ultra-low latency trading systems, FPGAs are the only viable option that meets the time requirements.

4.3.1 Derivation

This section covers the details behind the methodology. All the data presented here was gathered on a system consisting of an AMD EPYC 7543 32-core Processor and an NVIDIA GeForce RTX 3090. The algorithms and tests used to obtain such results are well known. This section formalises them and shows the exact values that are required in section 4.3 to determine misprediction penalties and cache misses percentages used to determine whether the application suffers from high branch misprediction, high cache misses or not.

Figure 4.5 shows the evolution of NVIDIA CUDA cores over the past years, revealing a remarkable increase by two orders of magnitude within a decade. Furthermore, recent years have seen an acceleration in this trend, indicating that this growth is likely to continue in the future. To achieve high performance on present-day GPUs, it is imperative that the application is

embarrassingly parallel and can exploit the increased number of CUDA cores. A few hundred parallel threads were sufficient for achieving significant speedup in the early 2010s, but to fully utilise top-of-the-line GPUs at the time of writing, the level of parallelism required is in the order of tens of thousands. To ensure future-proofing, it is advisable to aim for at least one order of magnitude higher parallelism than the current number of CUDA cores available on state-of-the-art GPUs. The following two metrics are branch miss rate and cache miss rate. It

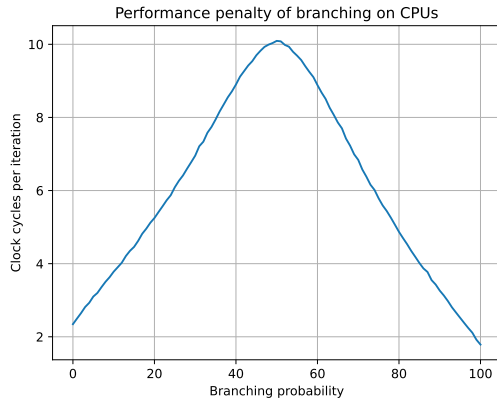


Figure 4.3: Clock cycles required to execute: $\text{if}(\mathcal{R} < P) \text{ sum} += \mathcal{R};$

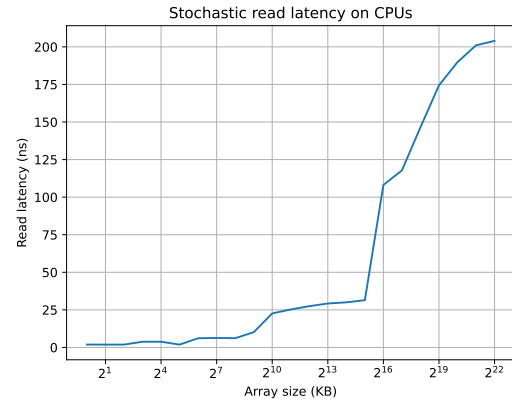


Figure 4.4: Read latency of the various memories in the system

is important to note that rate refers to the amount of branches or caches missed relative to the total computation. This aims to filter branches that have almost zero probability of being executed, as error checks from the analysis. Even on GPUs, branches that are either always or never executed do not impact performance.

Listing 4.1: CPU memory latency benchmark

```
auto vector = [i, padding for i in N]
vector.shuffle();
auto index = 0UL;
for (auto i = 0; i < num_ops; ++i) {
    //Access the next node
    index = list[index].index;
}
```

If the workload exhibits a high branch miss rate or high cache miss rate, the performance of

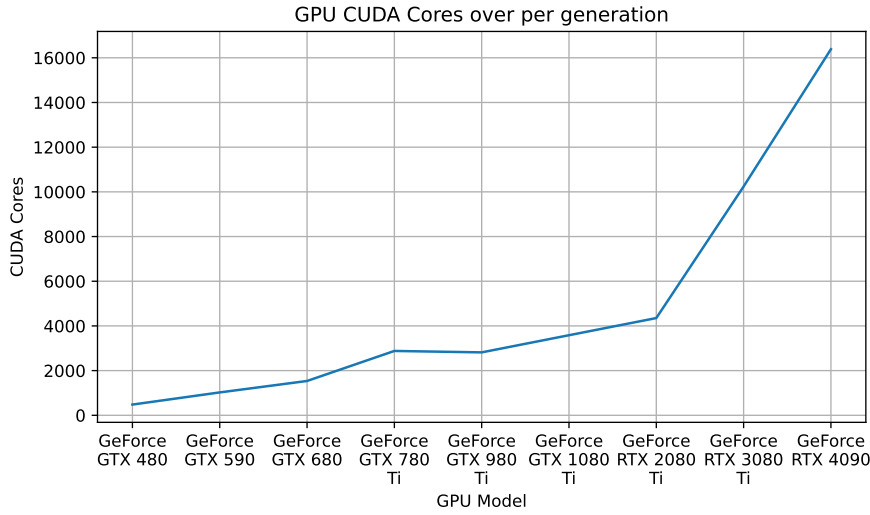


Figure 4.5: CUDA cores per GPU generation. Original data public domain from NVIDIA.

CPUs may be poor. Figure 4.3 shows the number of clock cycles required to execute an addition at varying branching probabilities. When the probabilities are close to zero or 100%, the CPU requires approximately two clock cycles to execute $if(\mathcal{R} < P) \text{ sum} += \mathcal{R}$. However, in the worst case, it requires approximately ten clock cycles, resulting in over five times slower performance. Memory reads are even more expensive. Figure 4.4 illustrates the memory latency for stochastic reads. Listing 4.1 shows the test used to measure the latency. The code begins by creating an array where $array[i] = i$ with padding and then shuffling it using MersenneTwister [MN98]. Then the array is accessed in a for loop using the index of the list. This causes stochastic uniform random accesses that the prefetcher fails to predict. It reveals that compared to the L1 cache, L2 cache, L3 cache, and DRAM are over 5, 15, and 108 times slower, respectively. Consequently, applications that execute a large number of stochastic memory reads can be over two orders of magnitude slower on CPUs. In these scenarios, the NVIDIA GeForce RTX 3090 Ti GPU is unlikely to provide significant performance improvements, as branches are even more expensive on the GPU, caches are smaller, and memory read latency is higher. Figure 4.6 illustrates the performance penalty of branching on the GPU. When threads diverge, the GPU de-schedule them, so even if there is a low branching probability, the few diverging threads will be executed later, resulting in increased execution time. This implies that the cost of executing the code is the sum of the time needed to execute both branches of the if statement. The code used for to measure branching on GPU is available in listing 4.2. The code

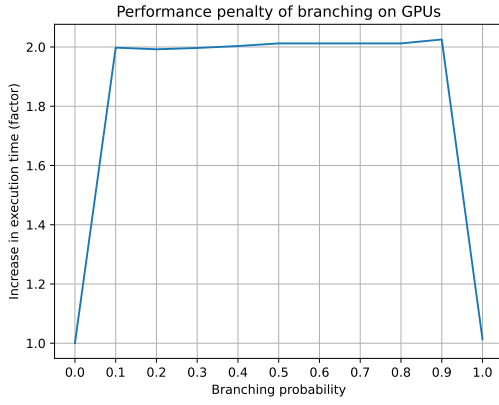


Figure 4.6: Increase in execution time due to branching on GPU

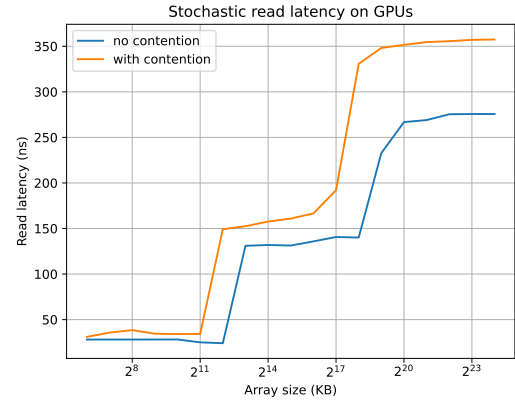


Figure 4.7: Read latency of the various GPU memories

performs the same amount of operations between the two branches if $iterations1 = iterations2$ and the probability of thread divergence can be set by the user.

Listing 4.2: GPU branch evaluation kernel

```
__global__ void branch_kernel(
    int* result,
    const int y, const int z,
    const int iterations1,
    const int iterations2,
    const bool* random) {
    int a = 0;
    int b = 0;
    int idx = (threadIdx.x + blockIdx.x * blockDim.x) % 32;

    if (random[idx]) {
        for (int i = 0; i < iterations1; i++) {
            a = a & y;
            a = a | z;
        }
    } else {
        for (int i = 0; i < iterations2; i++) {
```

```

        b = b | y;
        b = b & z;
    }
}
result[idx] = a + b;
}

```

Figure 4.7 shows that the memory access latency is even higher than on the CPU. These data are obtained using the same code presented in listing 4.1 ported to CUDA. Figure 4.7 shows data derived on the NVIDIA GeForce RTX 3090 Ti and AMD EPYC 7543. In cases of random access to the cache or on-board memory, the code slows down by 56 to 511 times, or even up to 660 times in cases of memory contention. For reference, shared memory is slightly faster, and accessing data from there slows down the computation by 32 times.

In summary, the NVIDIA GeForce RTX 3090 Ti offers a compelling advantage over AMD EPYC 7543 only for applications that have little to no branches, do not perform stochastic memory reads, and are embarrassingly parallel. In the case of sequential reads, the NVIDIA GeForce RTX 3090 Ti can provide a significant performance benefit compared to AMD EPYC 7543 as the NVIDIA GeForce RTX 3090 Ti’s Graphics DDR SDRAM (GDDR) has much higher bandwidth than AMD EPYC 7543’s Synchronous Dynamic Random-Access Memory (SDRAM). Figure 4.9 shows that NVIDIA GeForce RTX 3090 Ti’s GDDR6 can achieve over 800 GB/s in bandwidth. By contrast, DDR5 bandwidth is only 38.4 GB/s per channel, double or triple that in a dual/triple channel configuration, which is one order of magnitude slower than GDDR6 peak bandwidth. Therefore, for memory-bound applications that rely on sequential reads, the NVIDIA GeForce RTX 3090 Ti GPU can provide a significant advantage over the AMD EPYC 7543 CPU. However, in cases where there are a lot of branch or cache misses, FPGAs can be the right platform. Branches do not impact performance as both the “then” and “else” statements are executed in parallel in hardware, and the unneeded result is discarded at the end. Cache misses can be overcome by implementing custom pre-fetching and caching strategies that overlap computation and communication so that the FPGA is never idle waiting

for data. A high branch misprediction rate and high cache miss rate combined indicate that the NVIDIA GeForce RTX 3090 Ti GPU would perform quite poorly. However, a high branch misprediction rate alone is not enough to rule out the NVIDIA GeForce RTX 3090 Ti GPU. As the NVIDIA GeForce RTX 3090 Ti GPU offer multiple orders of magnitude higher parallelism compared to AMD EPYC 7543 and Xilinx VU9P FPGA exploiting even a fraction of the total available parallelism could achieve significant speedup over these other platforms. Hence, in this case is necessary to build a complete performance model, as section 4.6 and 4.7 explained, and use the total execution time forecasts to determine the best platform to use.

From our experience FPGA implementation often takes significantly more effort than GPU implementation; this should be taken into account when considering which accelerator to use.

The results from this section are likely to be valid for many other CPUs and GPUs, however, the readers should always validate their specific processors to confirm and quantify the results from such comparisons.

4.4 Software Model (FPGA)

FPGA programming presents several challenges. In particular, the long compilation time, which can span multiple days, makes design iterations in hardware extremely expensive in terms of time. Hence, reducing the number of compilations is crucial for successfully accelerating any application. The software model is a simplified software implementation of the parts of an application that are to be ported onto the FPGA. It should mirror the intended FPGA implementation and use the same API. It serves three purposes:

- insight into the selected code;
- testbench for verifying numerical and algorithmic changes;
- debugging reference for FPGA implementation.

To accomplish the initial purpose of the software model, the algorithm can be implemented as basic, unoptimised code, which aids in analysing the application and refining the performance model and architecture. Runtime profiling is not typically done in the software model, but rather metrics like memory access patterns and loop iterations executed are examined. Additionally, the software model is used to assess various algorithmic and numerical options, which inform the architecture and performance prediction. Consequently, the software model is usually developed in conjunction with both performance model and architecture.

It is important to reintegrate the software model into the original application to ensure that the planned API between the original application and the accelerated modules is adequate to recreate the original functionality. Furthermore, this integration facilitates the verification of the software model by comparing the results between the original software and the application using the software model. It should be noted that due to the non-associativity of arithmetic using floating-point numbers, minor differences in the result are expected at this stage.

Using the same API for the software model as for the FPGA implementation offers an easy-to-use debugging tool for the FPGA implementation. It is essential to verify against both the original application and the software model because the software model undergoes the same numerical and algorithmic changes as the FPGA implementation. This makes it easier to determine if an unexpected result for a given input dataset is due to an error in the FPGA design, side effects, or fundamental issues with the selected algorithm or its numerical properties. Resolving numerical and algorithmic problems in the software model is significantly easier than in an FPGA implementation. Additionally, this provides an FPGA mock-up implementation for earlier integration into the host application.

Although creating the software model and continuously improving it during the design process incurs additional design effort, it avoids the need to make these design iterations in hardware. It is widely acknowledged that software development, especially if the code is not optimised for performance, is easier than hardware development, resulting in time and cost savings compared to performing these design iterations in hardware. Moreover, the software model can be developed by software engineers and does not require hardware expertise.

4.5 Operations count

Accelerating applications requires identifying analysing and accelerating the bottlenecks. To estimate the performance it is necessary to know the number of operations executed in the bottleneck. This section explains how to count these operations in both GPUs and FPGAs.

FPGA operations count In the case of FPGA, the number of operations corresponds to the number of arithmetic operations (+, −, *, /) and elementary functions (sin, cos, log, exp) present in the program. These operations can be manually counted by analysing each line of code of the software model which is a realistic representation of the hardware design. There are some tools to aid in the process, such as the ones present in Intel OneAPI [One] or using hardware performance counters. However, some applications might require a manual count when these tools do not offer sufficient level of detail required for the model. In the case of for loops, the number of operations needs to be multiplied by the number of iterations. When encountering while loops the average number of iterations can be determined using statistics from the code executed on one or more representative inputs.

GPU operations count When it comes to estimating the number of operations on GPUs, simply counting the number of operations like in the FPGA case is not sufficient. On GPUs, the application is a sequence of instructions, PTX and AMDGCN assembly (ASM) for NVIDIA and AMD, respectively, and all of the instructions need to be executed. Therefore, to estimate the number of operations, one can compute the number of ASM instructions. While some operations require fewer clock cycles to execute than others, assigning a fixed constant cost to all operations does not significantly compromise the accuracy of the model.

To obtain the number of ASM instructions, the best approach would be to implement the bottleneck on the GPU. However, this is not always practical since the aim of the methodology is to minimise development time. As an alternative, x86 CPU instructions can be used as a substitute. While the number obtained this way may be less accurate than that obtained from the GPU Instruction Set Architecture, a range analysis from best to worst case can still be

performed to obtain reasonable estimates, as section 4.8.11 will show. The number of ASM instructions can be obtained by compiling the CPU code without vectorization and parallelism. Then the code can be executed on a representative input while using hardware performance counters to measure the number of ASM instructions executed. This value then needs to be divided by the input size (how many elements were processed) which is the source of parallelism. This is to obtain the number of instruction per element required by the model that section 4.7 will present.

When using micro-benchmarks, i.e., implementing specific algorithms and functionalities to derive the model, the forecasts were within 30% accuracy in most cases. The predictions were occasionally too optimistic and, at other times, too pessimistic. However, the optimistic and pessimistic cases remained within the same order of magnitude, resulting in an average error below 10%. Many of these micro-benchmarks are drawn from the CERN CMS high-level trigger [ABD⁺06], whose codebase is open source. Some algorithms are implemented on both CPU and GPU, allowing for direct comparison. These examples are particularly relevant to physics, making them especially suitable for the types of workloads analysed in this thesis. Nevertheless, this focus may limit the accuracy of the operation count for workloads differing from those considered here. Another limitation is given by the compiler and compiler version used here. Any change in the toolchain might invalidate the operation count methodology.

4.6 Performance model

By using the code analysis results, the operations count and the initial software model, a representative performance model can be created to capture the planned implementation on the selected platform. This involves predicting the hardware and bandwidth usage, as well as the execution time for a problem of a known size, and obtaining a preliminary speedup estimation. The performance model not only enables design space exploration but also highlights potential problems and bottlenecks of the architecture before the implementation begins. As the architecture aims to alleviate these bottlenecks, an iterative improvement of the performance

model and architecture is often required. This allows for early design space exploration in the acceleration process, facilitating cost-benefit analysis and decision-making.

4.6.1 Predicting I/O Bandwidth Usage

The bandwidth available for communication between the host and the FPGA relies on the physical interconnect employed. On most acceleration platforms, PCIe is used as the interconnect. PCIe is constructed using bidirectional links, implying that the full bandwidth can be utilised for both reading and writing concurrently. For instance, an eight-lane link of second-generation PCIe can transfer up to 4 GB/s. Consequently, the communication time ($\mathcal{T}$) between the host and accelerator can be estimated by dividing the sum of the data size transferred in either direction ($\mathcal{S}$) by the bandwidth (BW) multiplied by the relative efficiency (η), as shown in Equation 4.1. It is worth noting that in case the applications perform data transfers of different magnitudes, the lowest magnitudes can be ignored without affecting the accuracy of the result.

$$\mathcal{T}_{comm} = \sum_i \frac{\mathcal{S}_i}{BW \times \eta(\mathcal{S}_i)} \quad (4.1)$$

Efficiency is a function that captures several factors and returns a percentage. These factors include the PCIe encoding scheme, which is the strategy used to convert data into packets for transfer over the PCIe bus. For example, the 8b/10b encoding scheme used in PCIe 1.0-2.1 introduces a 20% overhead, whereas PCIe 3.0-4.0 uses a 128b/130b encoding scheme resulting in a much lower 1.5% overhead. The different Direct Memory Access (DMA) implementations can also reduce efficiency. DMA characteristics and benchmarks are well-documented in the FPGA case. In the case of GPUs, one can test PCIe bandwidth using CUDA-samples [NVI] or by implementing any custom code that transfers data to the GPU and measures the time it takes.

Figure 4.8 shows the measured bandwidth for an NVIDIA GeForce RTX 3090 Ti on an AMD system. Bandwidth is affected by the transfer size, with smaller transfers being more expensive

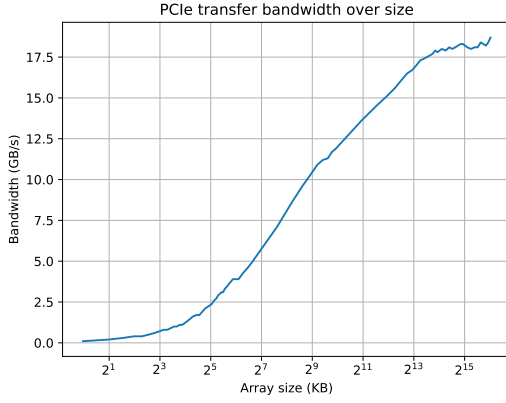


Figure 4.8: PCIe 4.0x16 bandwidth on an NVIDIA GeForce RTX 3090 Ti

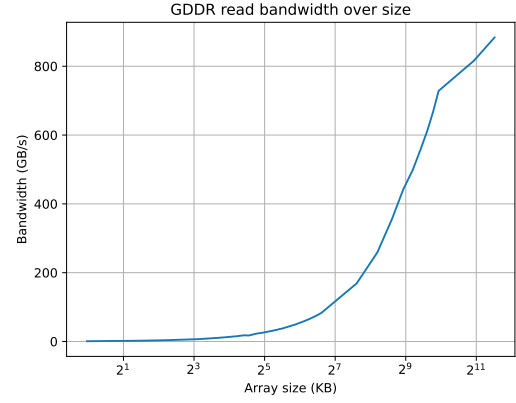


Figure 4.9: GDDR Bandwidth over data size.

due to the possibility of not fully filling PCIe packets. The measured bandwidth is 18 GB/s, lower than the theoretical maximum of 31.5 GB/s, which can be attributed to several factors, such as DMA not being designed to handle high bandwidth, CPU bottleneck or driver issues. It is important to consider these limitations in the model to allow applications to benefit from GPU acceleration even if the available bandwidth is far lower than the theoretical maximum.

4.6.2 Memory transfers

Memory transfers have a significant impact on performance, and therefore they must be carefully considered in the model. Both the GPU and the FPGA lack pre-fetchers, which means that memory accesses need to be handled with care. There is no automated mechanism that can mitigate the cost of memory access.

Bandwidths of this memory are unidirectional, e.g., bandwidths in both directions (read and write) are shared. As such, Equation 4.2 shows that the time to transfer data ($\mathcal{T}_{mem}$) between onboard memory and the accelerator can be calculated using the sum of the size of the data either read or written ($\mathcal{S}$) divided by bandwidth (BW) and efficiency (η). Efficiency is a function that maps the data transfer size to a percentage that represents the amount of the maximum bandwidth achievable by a transfer of size ($\mathcal{S}$). In most cases, lower-magnitude data transfers could be omitted. However, the user needs to be careful when doing so; small not sequential data transfers that happen multiple times during the computation, impact performance heavily

due to their large latency as shown in Figure 4.7, and cannot be omitted.

$$\mathcal{T}_{mem} = \sum_i \frac{\mathcal{S}_i}{BW \times \eta(\mathcal{S}_i)} \quad (4.2)$$

Voss et al. [VKM⁺21] measured the efficiency of DDR4 memory on a Xilinx VU9P FPGA and found that for a single word, the efficiency is below 20%. However, it increases to 80% for ten accessed words and reaches 90% for 100 accessed words. This indicates that random memory accesses could result in poor FPGA performance. Hence, it is essential to utilise on-chip memory effectively and read and buffer more data to maximise FPGA performance.

GPU performance is also significantly affected by memory transfers, and it is important to consider these factors in the model. Figure 4.9 shows the memory bandwidth over data size for GPUs, and it demonstrates that GDDR efficiency is initially worse than that of FPGAs until at least 128KB are read in one go. After this point, performance increases exponentially with the data increase and reaches a peak of over 800 GB/s.

4.6.3 FPGA performance model

FPGAs consist of predictable building blocks, making it possible to estimate the performance of a given architecture accurately using a few linear equations that will be presented later in this section. The total time it takes to process a workload on an FPGA is the sum of the time needed for initialisation and the time for actual execution. If the workload is sufficiently large, the execution time is expected to dominate over initialisation time and can be neglected. However, the exact initialisation time depends on the used platform and framework and can vary between 1 and 100 ms. In a streaming dataflow design, the execution time is determined by the longest latency, which is the maximum time needed for computations, data transfer between the host and FPGA, and data transfer between the FPGA device and the board memory. If utilisation is not constant, the execution time of these tasks is treated as an additive term to the overall runtime.

Predicting area usage To determine the computational time requirements for a given device, the achievable degree of parallelism must first be identified. The main limiting factor is the availability of FPGA hardware resources. Typically, FPGA resources are used for three purposes: implementing arithmetic operations, scheduling operations, and supporting other IP modules such as PCIe and memory controllers.

To estimate the area usage for arithmetic operations, one must first determine the amount of hardware resources required to implement a simple operation on the target FPGA device. This can be done using automated tools, vendor documentation, or by creating micro-benchmarks and running them through vendor tools. The area usage can then be estimated by multiplying the area cost of a single operation by the number of operations required on the fabric. Scheduling operations rely on FIFOs, which are typically implemented in on-chip memory or using registers.

For IP modules, the resource requirements can typically be predicted using micro benchmarks. To predict on-chip memory usage, it is important to understand the three main purposes for on-chip memory: (1) in operations and IP blocks, (2) for buffering or reordering of data, and (3) for balancing the pipeline of computational structures. The area usage for the first two cases can be estimated using micro benchmarks or Equation 4.3 which calculates the number of memory blocks required based on the width, depth, and number of read ports required.

Predicting the area usage for operation scheduling is difficult since it requires knowledge of the scheduling algorithm used by the toolchain. However, it is possible to identify the largest FIFOs needed to access data from previous cycles and treat them as normal buffers. The amount of memory resources required by smaller FIFOs is highly dependent on the application, so only a rough estimate is possible. Overall, it is recommended to assume slightly higher memory and logic usage to ensure sufficient safety margins for additional circuitry, especially for scheduling and control logic.

$$n_{mem} = \frac{w_{req}}{w_{hardware}} \times \frac{d_{req}}{d_{hardware}} \times \frac{p_{req}}{p_{hardware}} \quad (4.3)$$

Predicting the Compute Performance The next step is to estimate the achievable parallelism and the number of data items that can be processed per clock cycle, without taking into account bandwidth limitations. The time required ($\mathcal{T}_{comp}$) to process a set of data items can be calculated using Equation 4.4, where the total number of items to be processed n_{total} is divided by the product of the target frequency (f) and the number of items processed per cycle (n_{cycle}). In case the same item needs to be processed multiple times during the computation the total number of items to be processed (n_{total}) needs to be multiplied by the number of iterations ($n_{iterations}$).

$$\mathcal{T}_{comp} = \frac{n_{total}}{n_{cycle} \times f} = \frac{n_{total} \times n_{iterations}}{n_{cycle} \times f} \quad (4.4)$$

Frequency prediction is not always precise. However, if chip resources are used less than 80% and sufficiently deep pipelining is implemented, the frequency on the same FPGA can be estimated based on previous design experience or by conducting experiments using artificial designs to fill up the chip. For instance, the Xilinx VU9P FPGA can typically achieve at least a frequency of 200 MHz if the chip is filled up to 80% using any toolchain to program it. However, as new technologies are developed, the target clock frequency should be revised accordingly so that it accounts for such improvements. The methodology suggests using a value that can be achieved without the use of advanced optimisations. In case the achieved clock frequency does not meet the performance requirements, the clock frequency could be raised but the necessary optimisations needed to achieve it need to be factored inside the performance model and the architecture so as to forecast the performance and estimate engineering effort.

4.7 GPU performance model

The performance of a GPU is influenced by various factors such as branches, data transfer, and parallelism, which were discussed in section 4.3. To develop a complete performance model, these parameters, along with others, need to be taken into account.

The operations count described in section 4.5 allows for the estimation of the number of assembly instructions ($\mathcal{N}_{ASM}$). The approach involves using half of the measured number of instructions as a lower bound and double that amount as a high range. The lower bound provides an estimate of the best-case scenario, while the high bound gives an estimate of the worst-case scenario. It is important to consider both the best and worst-case scenarios to gain insights into the likelihood of successfully accelerating the application and the best achievable performance.

Another important parameter to consider in GPU acceleration is efficiency (η). Figure 4.6 illustrates the performance penalty of branching on GPUs. When branching occurs, the GPU de-schedules the diverging branches, and as a result, the execution time becomes the sum of the time to execute the “then” and the “else” branches. However, the number of ASM instructions already accounts for this, as the instructions of both sides of the branch are already considered. Nevertheless, the model does not take into account the idle CUDA cores that are not executing any useful work due to their thread being de-scheduled. The “efficiency” parameter accounts for this behaviour and its initial value is set to 1, but it is halved each time a branch is encountered.

Input size ($\mathcal{S}$) is an important parameter that is required to forecast the total execution time of an application. The input size is used in combination with the number of ASM instructions ($\mathcal{N}_{ASM}$) to estimate the total amount of instructions that the GPU will execute. The method used to estimate the number of ASM instructions, as previously illustrated in section 4.5, produces the number of instructions needed to process only one element of the input. To estimate the total number of instructions, the number of ASM instructions needs to be multiplied by the input size. In more complex workloads, the input size might be a vector instead of a scalar. In such cases, it is essential to be careful when obtaining the forecasts, the input size is kept as a separate parameter instead to allow for further analysis of the input size. For example, GPU acceleration might not be worth it for small-medium sized inputs. However, large inputs might change the outcome, and having the input size as a separate parameter enables exploring this aspect.

The remaining parameters are hardware-dependent and they are the GPU **clock frequency**

(f) which is used to estimate the processing time and **cores** ($\mathcal{N}_{CORES}$). $\mathcal{N}_{CORES}$ models the number of CUDA cores from NVIDIA or Stream processors in the AMD case. They estimate the parallelism achievable on the GPU. Each core is assumed to process one element.

Once the above parameters are computed the following equations help estimate the performance:

$$\mathcal{T}_{compute} = \frac{\mathcal{S} \times \eta \times \mathcal{N}_{ASM}}{f \times \mathcal{N}_{CORES}} \quad (4.5)$$

4.7.1 Total time

To accurately estimate the total execution time, data transfers must be taken into account, as this is a complex and extensive topic. Section 4.6.2 provides the necessary details for modelling data transfers. Using Equation 4.1 and 4.2 the total compute time becomes as shown in Equation 4.6.

$$\mathcal{T}_{total} = \mathcal{T}_{compute} + \mathcal{T}_{mem} + \mathcal{T}_{comm} \quad (4.6)$$

$$\mathcal{T}_{total} = \max\{\mathcal{T}_{compute}, \mathcal{T}_{mem}, \mathcal{T}_{comm}\} \quad (4.7)$$

However, in case data transfers can be overlapped with the computation, that is to perform the transfers while executing some of the computation then the total time is given by the maximum between the three components, as shown by Equation 4.7. In reality, the total time is the result of a combination of data transfers that can be overlapped and some that cannot be hence, in practical terms Equation 4.6 and 4.7 should be used together.

4.8 Example and Evaluation

To evaluate the methodology introduced earlier, the MC in this section was modelled using the proposed decision process, which was then used to determine whether acceleration using either FPGA or GPU would be suitable. This MC simulation computes Coulomb Scattering and section 4.8.1 will give the necessary details. This evaluation serves as an example of how to use the methodology and as a validation of the decision process and the ability of the model to accurately determine the most suitable platform for acceleration.

4.8.1 Coulomb Scattering

As a simple but physically meaningful MC test case, single Coulomb scattering has been chosen. The screened Rutherford Differential Cross-section (DCS) can be obtained by solving the scattering equation under the first Born approximation using a simple exponentially screened Coulomb potential in the form of [Wen26]

$$V(r) = \frac{zZe^2}{r} \exp(-r/R), \quad (4.8)$$

with a screening radius R , target atomic number of z and projectile charge Ze . This leads to the screened Rutherford DCS for elastic scattering

$$\frac{d\sigma^{(SR)}}{d\Omega} = \left(\frac{zZe^2}{p\beta c} \right)^2 \frac{1}{(2A + 1 - \cos\theta)^2}, \quad (4.9)$$

where p is the momentum, β is the velocity of the projectile particle and A is the screening parameter. The corresponding total elastic scattering cross-section is given by

$$\sigma^{(SR)} = \left(\frac{zZe^2}{p\beta c} \right)^2 \frac{\pi}{A(1 + A)}, \quad (4.10)$$

while the angular distribution of single elastic scattering can be given as

$$f_1(\theta)^{(SR)} = \frac{1}{\pi} \frac{A(1+A)}{(1 - \cos \theta + 2A)^2}. \quad (4.11)$$

This leads to an analytical solution of the inverse equation and sampling of $\mu = \cos(\theta)$ as

$$\mu = 1 - \frac{2A\xi}{1 - \xi + A} \quad (4.12)$$

with $\xi \in \mathcal{U}(0, 1)$. The corresponding single elastic scattering model has been implemented in C++ and then accelerated. The original code can be found on GitHub: <https://github.com/DiamonDinoia/mcss>.

4.8.2 Application analysis

The MC simulation is embarrassingly parallel as each particle can be simulated independently. However, it should be noted that this requires a parallel random number generator as well as a coordination mechanism to avoid race conditions when updating the radiation deposition in the geometry.

According to the Linux *perf* command, the MC simulation analysed in this study suffers from a low branch misprediction rate relative to the total number of instructions. It shows that 2.3% of the total branches are missed, thus indicating that this MC simulation falls in the category of workloads that are suitable for GPU acceleration. However, this information alone is not enough to conclude that the GPU would be faster as further efficiency analysis is needed.

The cache misprediction rate is low, analysis using the Linux *perf* command shows that 0.81% of the total cache reads are missed. This is expected because the MC simulation supports only electrons that do not travel far in space hence, by using proper geometry partitioning is possible to always access memory locations in the cache.

Given the characteristics of this workload, the GPU seems to be a better candidate compared to the FPGA, however as section 4.3 explained, this is one of the border case applications where a complete performance model is necessary to determine the best accelerator to use.

The accuracy requirement of Coulomb Scattering requires the resulting distribution to pass a Kolmogorov–Smirnov test with accuracy $\alpha = 5 \times \sigma$. This is important as this accuracy requirement can be met using single-precision floating-point instead of double-precision. This greatly increased performance on both the GPU and the FPGA. According to the NVIDIA C programming guide [CUD], single precision floating point achieves at least 50% more performance than double precision on NVIDIA GPUs. According to Xilinx documentation [Per], a double precision floating point addition requires up to 45 DSPs, while a single precision addition requires at most 7 DSPs. Hence, in this case, it would be possible to use the same hardware resources to perform 5 additions in parallel instead of one, potentially increasing the performance by 5 times. Another additional gain of using a smaller data type is that memory storage requirements and data transfers are halved since single precision requires only 4 bytes of storage compared to the 8 required by double precision.

The complete model that section 4.8.4 and 4.8.5 will explain shows that there is no clear winner between the FPGA and the GPU as both should achieve similar performance, the performance of the FPGA is within the range estimated for the GPU. However, in the best case scenario the GPU can be faster. In this case, the developer can decide to use its accelerator of preference as both are viable options.

4.8.3 Data transfers

According to Algorithm 2 the first step is analysing the application to perform an initial architecture draft and split between CPU and accelerator. To do so, our approach consists of first identifying the boundaries of the design exploration, in terms of data transfers and accelerator resource utilisation. It is possible to analyse resource utilisation and data transfers in any order as the final set of possible designs will be the same. This section analyses possible CPU accelerator partitions and identifies the ones (if any) where data transfer time remains within the time constraints.

Data transfers can occur from the CPU through the PCIe bus and from and to the on-board memory. Depending on the partition between CPU and accelerators the amount of data that

needs to be transferred can vary significantly. The previous section showed that the simulation is composed mainly of a single while loop with no nested loop. So any partition between CPU and accelerator needs to divide the main loop between CPU and accelerator. Dividing the loop, however, results in large data transfer overhead through the limited PCIe bandwidth. This entails transferring partial particles between CPU and accelerator multiple times. The lower bound for the volume of data transfer is given by the number of iterations multiplied by the number of particles and the size of a single particle. The particle is composed of 7 32-bit floating-point numbers, 3 for the position, 3 for the direction and 1 for the energy, totalling 224 bytes. The average number of iterations is 34.5 and the number of particles necessary to reach convergence is 100×10^6 . This results in a data transfer of at least 772.8GB. Using equation 4.1 and an efficiency of 80% over a PCIe 3x16 results in a $\mathcal{T}_{comm} = 61\text{s}$. It is worth mentioning that executing the same simulation on CPU only requires 8.4s as table 4.3 will show. Even when using PCIe 4x16 that doubles the bandwidth, the resulting $\mathcal{T}_{comm}$ would be 31s hence, dividing the main loop between accelerator and CPU does not give any performance advantage.

While the original Coulomb Scattering implementation requires full 3D geometry which does not fit in the on-chip memory, it is possible to partition the geometry and perform the simulation sequentially on each partition. This requires simulating all the particles in one partition before moving on to the next partition. A partition size can be arbitrary but it is set to less than half the available on-chip memory. This allows for double buffering and prefetching the next partition while computing the first. Assuming that the number of particles is high enough (at least 10×10^4), this allows to completely hide memory transfers with computation except for loading the first partition, this data transfer is negligible compared to the total computation time.

The input and the output of the simulation are a few parameters (seed, beam position, direction, energy and seed) while the output contains two distributions that require a few KBs of storage. These data require less than 1 ms to complete so they can be omitted from the model.

4.8.4 FPGA performance model

The second part of the initial analysis consists of analysing which parts of the application can be accelerated in terms of accelerator resources. In the case of FPGA, the entire application might not fit on-chip as it might require more DSPs or more BRAMs than the ones available. Hence, using the results shown in the previous section the design exploration space here is further limited.

The previous section showed that dividing the main simulation loop between CPU and accelerator leads to low performance due to the amount of data transfers. Hence, the only possible architecture that does not require major changes to the original application consists of accelerating the main simulation loop in its entirety. The initial (deeply pipelined) architecture proposes that the FPGA simulates one iteration of the main simulation loop per clock cycle, in case this architecture requires more resources than the one available, the possibility of sharing resources within the loop and using more clock cycles to perform one iteration can be explored. In our experience, FPGA programming requires one order of magnitude more engineering effort compared to standard software development, hence we accelerate only applications that are at least one order of magnitude faster than CPU, this cutoff can be adapted to account for the experience level of the developers. For this architecture to be at least one order of magnitude faster on FPGA compared to the CPU, the resulting FPGA design must either reach a clock frequency of 250 MHz or achieve parallelism greater than one. HLS toolchains might not guarantee reaching 250 MHz on a VU9P used in this study, hence the second option needs to be explored. According to the model's predictions, when targeting a Xilinx VU9P FPGA, simulating one iteration of one particle simulation main loop per clock cycle requires 127 DSPs and 10 Mb of on-chip memory. These values are obtained by manually analysing the code line by line and counting the required DSPs and memory requirements. For example in the following code:

```
double theLongiDistr[longiDistNumBin] = {0.0};
double theTransDistr[transDistNumBin] = {0.0};
```

...

```
aTrack.fPosition[0] += aTrack.fDirection[0] * stepLength;
aTrack.fPosition[1] += aTrack.fDirection[1] * stepLength;
```

The memory requirement for lines 1-2 is:

```
sizeof(double)*longiDistNumBin + sizeof(double)*transDistNumBin
```

The DSP requirement instead, according to Xilinx documentation [Per], is two single-precision floating-point additions and two single-precision floating-point multiplications. Hence, lines 6-7 require 2 DSPs for addition + 3 DSPs for multiplication each = 10 DSPs. The counting operation illustrated above needs to be repeated for the entire software model. In this case, modelling LUTs and FFs have been omitted as the DSP utilisation is one order of magnitude higher than the rest as most of the simulation is composed of floating-point arithmetic. From [Per] it is possible to notice that floating point arithmetic mostly requires DSPs instead of LUTs and FFs. While other operations (multiplexers, de-multiplexers) require LUTs and FFs, these operations are one order of magnitude less common in this MC simulation and can be neglected.

The Xilinx VU9P FPGA is equipped with 6840 DSPs and 345 Mb of on-chip memory, thus the simulation of one particle requires 1.86% and 3.35% of the total available DSPs and on-chip memory respectively. Since $3.35 > 1.86$, the limiting factor is the on-chip memory. Moreover, the compiler requires part of the on-chip memory to schedule the design, in our worst-case analysis we consider only 50% of the total memory available to store data.

Equation 4.4 shows that $\mathcal{T}_{compute}$ is 1,150 ms where:

$$n_{total} = 100 \times 10^6$$

$$n_{iterations} = 34.5$$

$$f = 200 \times 10^6 s^{-1}$$

$$n_{cycle} = 15$$

As section 4.8.2 previously stated, memory accesses can also be omitted as the vast majority of the memory accesses happen in cache due to the specific geometry partitioning scheme adopted. As mentioned in section 4.8.3, the data transfers account for less than 1 ms on a PCIe 3.0x16 bus and hence are neglected from the computation.

For the sake of simplicity, aggregation of the raw data to the final histograms was not implemented on the FPGA. According to profiling results the CPU requires 198 ms to perform the final aggregation hence the $\mathcal{T}_{compute}$ is forecasted 1,348 ms.

4.8.5 GPU performance model

Similarly, to the previous section, the results of section 4.8.3 are used to limit the design space exploration.

This section uses NVIDIA terminology as the GPU available for this study is an NVIDIA V100. However, the concepts illustrated here apply to AMD GPUs as well. Similarly to the FPGA performance model, the first architecture was drafted by applying Algorithm 2. The initial architecture exploits the embarrassingly parallel characteristics of the Coulomb scattering simulation by parallelising over particles – each CUDA core simulates one particle at a time. This is possible as modern GPUs (from CUDA 1.3 onwards) support atomics for updating the shared geometry. The efficiency is 0.0625 as each particle encounters an average of 16 diverging branches during the simulation. The number of ASM instructions counted using performance counters on CPU and the previously mentioned methodology is rounded to approximately 5000. Rounding the value by a small margin does not impact the accuracy of the model. The parallelism depends on the physical hardware capabilities as the simulation is embarrassingly parallel and exploits all available resources. In this case, an NVIDIA V100 is used for this study, which offers 5,120 CUDA cores running at 1.455 GHz. Finally, the input size is 100×10^6 particles.

As section 4.8.3 showed data transfers can be omitted. The geometry is assumed to be stored in a format that does not cause costly cache misses. This is possible thanks to electron position

locality as they do not travel far at each iteration step. Hence, memory access overhead is negligible in this case. Please note that any architecture that is designed not to be memory-bound can neglect data transfers from the model. PCIe data transfers are also negligible as the geometry can be populated directly on the GPU, and the results comprise only longitudinal and transverse distributions instead of the full geometry.

Particles	Clock (GHz)	CUDA cores	Efficiency (0,1]	ASM instructions	Time (ms)
100×10^6	1.455	5,120	0.0625	2,500	537
100×10^6	1.455	5,120	0.0625	5,000	1,074
100×10^6	1.455	5,120	0.0625	10,000	2,148

Table 4.1: Design space exploration.

Table 4.1 shows the design space exploration and the parameters used. These values are substituted in Equation 4.6 while zeroing data transfers as they are negligible in this case. The computation happens mostly in the L1 cache, and results are written to GDDR asynchronously. Data transfers over the PCIe account for less than one millisecond. The expected computation time ranges between 0.5 and 2 seconds.

4.8.6 FPGA architecture

This section shows how to use the previous design space exploration constraints to derive an architecture and forecast its performance. For the sake of clarity, we omit the intermediate architectures, only showing the final architecture that meets time requirements. This corresponds to lines (4-5) of Algorithm 2. It is worth noting that lines (7-11) require doing the same tasks performed here but with a design space exploration further limited by the new constraints given by the bottleneck.

The model's predictions show that to meet the speedup requirement, the resulting architecture should be parallel. Figure 4.10 shows the resulting architecture. In total, there are 15 instances, as predicted by the model: each instance is composed of a particle generator, a RNG capable of generating 96 bits per clock cycle, and a particle simulator. The particle generator models a pencil beam; hence, all particles generated share the same initial energy and direction. The

RNG is implemented using three separate instances of the Mersenne Twister 32-bits (MT32) RNG [TN98]. This generator meets the requirements illustrated in section 2.3. It is also the most popular RNG and is most likely to be available in any programming language; hence, for the sake of generality, it was used in this study.

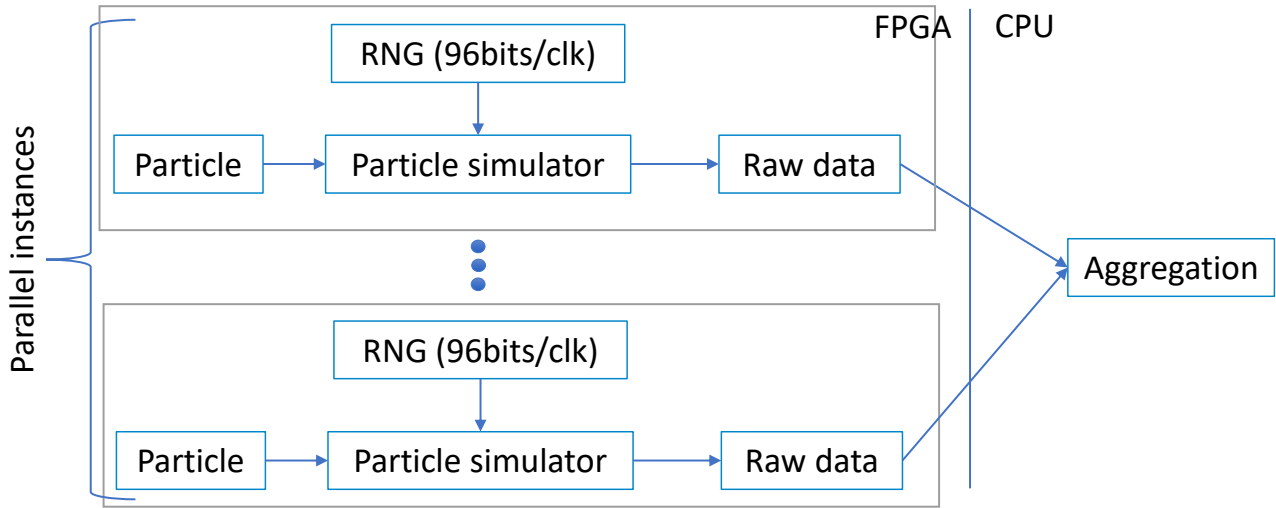


Figure 4.10: FPGA architecture schematic displaying the parallel FPGA instances and the CPU based aggregation.

The particle simulator is the core of the simulation as it implements the MC logic. It is composed of a while loop that iterates until the particle runs out of energy. The moment it runs out of energy, a sample point is taken and sent back to the CPU, which aggregates all of them at the end of the computation. It is worth mentioning that hardware implementations of while-loops cause backward edges in the dataflow graph. As shown in Figure 4.11, this backward edge feeds the result back and is processed multiple times until the guard is false; while loops can produce a result every n cycles where n is the depth of the pipeline. However, one of the characteristics of MC is independence: multiple particles can be fed to the while-loop since the order of the result does not matter. If the guard is true, a particle is sent back; if it is false, a new particle is fed in and a result is produced. Exploiting this characteristic can increase MC performance by a huge margin. For example, if n corresponds to 68, without independence, the performance would be 68 times slower.

This optimised particle simulator computes the final position of the particle, the moment it runs out of energy, and sends the result back to the CPU. For the sake of simplicity, aggregation

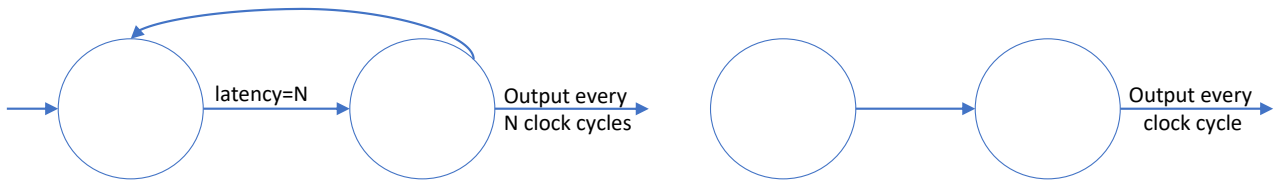


Figure 4.11: Illustration of the while loop optimisation implemented.

of the results is performed by the CPU. According to the model, moving the aggregation onto the FPGA would improve the performance by 15%.

4.8.7 GPU architecture

As previously stated, this section presents the final architecture obtained by applying the methodology and its performance forecasts. Intermediate architectures were omitted for the sake of clarity. The GPU architecture exploits the embarrassingly parallel characteristics of the simulation by processing particles in parallel. As the number of particles is on the order of 10^8 , this is likely to exceed the number of CUDA cores/Stream Processors for many generations. The MC simulation does not support any secondary generation, so each core integrates a pencil beam and a particle simulator. Replicating the pencil beam allows us to avoid using memory to store stacks of pre-generated particles, which reduces overhead and simplifies load-balancing. Differently from the FPGA, the GPU offer native support to atomics hence, to reduce data transfers raw data aggregation to the histograms is performed on the GPU itself.

The RNG used is Philox_4x32_10 [SMDS11], which is a fast and parallel RNG with reasonable quality and meets the criteria presented in 2.3. The implementation available in cuRAND, the free public NVIDIA RNG library for GPUs, allows each core to generate numbers independently in parallel. Other generators like Mersenne Twister for Graphics Processors [SM13] were tested, as they offer higher quality numbers, but they severely limit GPU performance.

Explicit use of shared memory is also investigated. Shared memory could be used to store the transverse and longitudinal distributions, RNG state, and a portion of the geometry. However, tests showed that manually managing shared memory results in lower performance.

Modern GPU micro-architectures have a monolithic on-chip memory, which is partitioned between shared memory and cache. The developer can explicitly use a portion of this memory to allocate frequently allocated data or to manually pre-fetch data from memory. However, doing so reduces the amount of memory available to the GPU to use as a cache. Since the Coulomb scattering algorithm causes electrons to not travel long distances, leaving the cache management to the GPU results in higher performance as this is done by specialised hardware. Any user-developed software, on the other hand, is executed in software by the CUDA cores, resulting in higher overhead, lower parallelism, and lower general performance.

4.8.8 Methodology forecasts

Using the results derived in previous sections it is possible to answer the question presented in section 4.2.

Question 1 The application is suitable for acceleration as at least one of the accelerators is forecasted to achieve higher performance than CPU and as will be explained extensive engineering effort is not expected.

Question 2 The achievable speedup is 6.4x on the FPGA and between 3.9x and 15x on the GPU.

Question 3 The architecture for the chosen accelerator requires accelerating the main simulation loop, data initialisation and aggregation will be performed on CPU.

Question 4 Estimating the engineering effort in terms of hours required is outside the scope of this methodology. However, it is possible to estimate the difficulty of accelerating the application in terms of:

- type of accelerator used;

- Major changes to the original application;
- Using complex algorithms or variations not available in libraries or already implemented for the specific accelerator;
- Developing complex data transfer strategies to overlap communications and computations.

In this case, both the FPGA and the GPU are forecasted to achieve performance in the same region. As previously stated, in this case the developer can pick the platform he is most familiar with. As for the other factors that impact engineering effort listed above, the architecture requires none of them. Hence, the methodology estimates that accelerating the application will not require extensive engineering effort.

4.8.9 Validation

The final results of this MC simulation consist of a longitudinal and a transverse distribution. To evaluate the accuracy of the results, a Kolmogorov–Smirnov test was executed against reference distributions, which concluded that the distributions are equivalent using $\alpha = 5 \times \sigma$. Section 2.2 gives more details on the Kolmogorov–Smirnov and the value used for α . Achieving equivalence with smaller α requires a further increase in the number of histories.

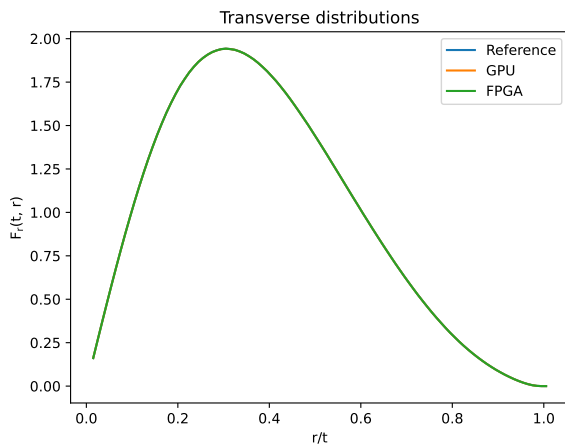


Figure 4.12: Transverse distribution.

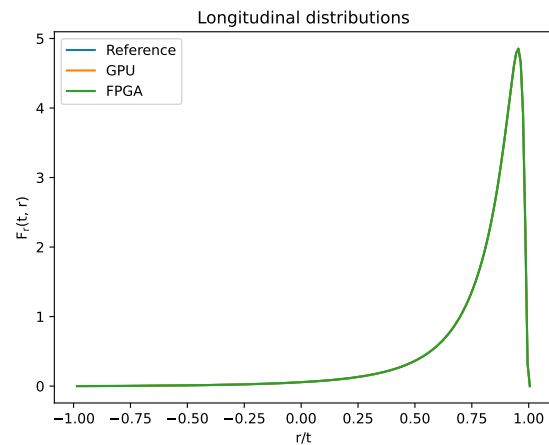


Figure 4.13: Longitudinal distribution.

Figures 4.12 and 4.13 show the longitudinal and transverse distributions obtained by simulating 100×10^6 particles in gold using a pencil beam of 128 keV. It can be noted that the lines of the reference implementation, the GPU, and the FPGA-accelerated versions overlap perfectly.

4.8.10 FPGA Results

The methodology advises against using FPGAs to accelerate this MC simulation. However, to show the accuracy of the forecasts, the FPGA architecture derived using the methodology was implemented and the performance is shown below.

To evaluate the performance we computed 100M histories of a 128 KeV electron interacting in water. The tests are executed on a Xilinx Alveo U200 that incorporates a VU9P FPGA device. The FPGA implementation was limited at 200 MHz, using single-precision floating-point. The CPU implementation was parallelised using OpenMP while the FPGA implementation was developed in MaxJ and compiled using MaxCompiler 2021.1 and Vivado 2019.2. It is worth mentioning that we did not rely on any compiler-specific optimisation; these results are therefore representative of any HLS toolchain. Relying on MaxCompiler-specific optimisations increases the achievable clock frequency up to 300 MHz obtained with no changes to the codebase. While the methodology takes the clock frequency as a parameter and can account for the differences between compilers, we used a value achievable by any HLS toolchains without having to implement complex buffering and fan-out reduction techniques to boost the clock frequency. In case higher clock frequencies are expected to be achievable, the user can change the value accordingly. Moreover, conservative parameters are encouraged as if the acceleration is worth with conservative parameters, any extra performance is welcome. However, a final design that fails to meet timing at the chosen frequency, thus does not meet the performance requirements, which might result in a needless investment of engineering effort. Hence, the final results do not include higher performance achievable by leveraging toolchain-specific optimisation and the final execution time is 1321 ms which is within the margin of error compared to the 1,348 ms predicted by the model. This confirms the accuracy of the modelling methodology.

4.8.11 GPU Results

This section shows the performance achieved by implementing and benchmarking the architecture derived using the methodology. To evaluate the performance we computed 100M histories of a 128 KeV electron interacting in water. The benchmarks were executed on an NVIDIA Tesla V100 PCIe 32GB equipped with 32GB GDDR5X and 5,120 CUDA cores running at 1,455 MHz. The code was compiled using GCC 9.3.1, NVCC 11.6 and OpenMP for parallelism. The time was obtained by running the tests five times and averaging the results. The GPU implementation achieves a performance of 873 ms which falls in the range predicted from the model and is close to the best achievable performance. Explicit use of shared memory is also tested, however, it fails to improve performance in any meaningful way hence the results are omitted.

4.8.12 Comparison between platforms

We compared the performance of 2x AMD EPYC 7551 32-Core Processor based on the 14nm Zen microarchitecture running at 2.0 GHz, boosting up to 3.0 GHz with an NVIDIA Tesla V100 based on the 12nm Volta microarchitecture and a Xilinx VU9P based on 20nm FinFet productive process. The price of the NVIDIA Tesla V100 is 11,458 USD and the price of a Xilinx U200 that uses a VU9P FPGA is 4,945 USD, the EPYC 7551 original price is 6,998 USD for the pair. The code was compiled using GCC 9.3.1 and OpenMP for parallelism. Table 4.2 shows the comparison between the platforms. This comparison is valid since both applications implement an accelerator-optimised architecture derived using the methodology, as shown in previous sections. In this case, the FPGA is 6.42 times faster and 9 times more cost-effective than a 64-core CPU. The GPU, on the other hand, is almost 10 times faster and 6 times more cost-effective according to the original manufacturer recommended prices. If the amount of engineering time is included in the equation, then the FPGA might not be as cost-effective compared to the other platforms. While the proposed model cannot capture these details, users are encouraged to consider them and make informed decisions. It is worth pointing out that these results are not intended to be a general statement about which accelerator technology is

better for this application. They provide a specific comparison between a given CPU, a given GPU and a given FPGA using particular code, data and compilers.

	CPU	FPGA	GPU
Time (ms)	8,494	1,321	853
Speedup	1	6.42	9.95
Speedup/Price	1	9.09	6.08

Table 4.2: Comparison between multiple platforms. Metrics are Time, speedup and cost-normalised speedup.

4.9 Discussion

Accelerating MC simulations requires significant engineering time. Not all MC behave the same when accelerated. Some simulations are faster on GPU while others are faster on FPGA, and others do not benefit from acceleration. Choosing the wrong accelerator might result in needless engineering effort. Achieving the best performance while minimising engineering effort requires a methodology that allows for “a priori” performance forecasts and steers the development in the most efficient direction.

The methodology proposed in this chapter aims to achieve all the requirements listed above, and validation on Coulomb Scattering shows that it is possible to reliably forecast performance while at least halving engineering effort. The methodology suggested that the GPU would achieve similar performance of the FPGA, hence instead of implementing the algorithm on both accelerators, only one implementation would have been required. Choosing the right accelerator at the early stages of development could have at least halved the time required for a successful implementation.

Table 4.3 provides a comprehensive comparison of various modelling methodologies, including the approach outlined in this chapter alongside state-of-the-art methods. Our method offers a unique blend of features, including a priori forecasts i.e. prior to any implementation, data transfers, and support for both FPGA and GPU accelerators, validated on MC simulations. In contrast, PPT-GPU [ABC⁺19] lacks a priori knowledge integration. Similarly,

Table 4.3: Comparison with state-of-the-art modelling methodology

Method	A priori	Data Transfers	Accelerator Support	Application Support
This chapter	Yes	Yes	FPGA & GPU	Any
PPT-GPU [ABC ⁺ 19]	No	Yes	GPU	Any
GCoM [LHL ⁺ 22]	No	Yes	GPU	Any
Choi et al. [CRKB20]	No	No	GPU	Any
Goswami et al. [GSB22]	Yes	No	FPGA	CNN
Da Silva et al. [DSBDT13]	No	Yes	FPGA	Limited
Voss et al. [VKM ⁺ 21]	Yes	Yes	FPGA	Any

GCoM [LHL⁺22] offers GPU support with data transfer capabilities but lacks a priori forecasts. Choi et al. [CRKB20] present an end-to-end approach with GPU support but without a priori forecasts or data transfers. Goswami et al. [GSB22] offer a priori forecasts and FPGA support but omit data transfers, and support only CNNs. Da Silva et al. [DSBDT13] support FPGA accelerators with data transfers but have limited application scope, in particular, they do not support complex applications such as MC simulations. Finally, Voss et al. [VKM⁺21] encompass a priori forecasts, data transfers, and FPGA support, applicable across various domains, however, their approach lacks the GPU support considered here.

4.10 Summary

In this chapter, a methodology for accelerating MC and its derivation are presented. Firstly, MC acceleration challenges are presented and the methodology explains how to overcome them. In particular, it shows how to predict the accelerator’s performance considering the accelerator’s compute characteristics depending on how the application leverages them. Then an MC simulation for calculating Coulomb Scattering, a real physics process, was used to evaluate the correctness of the methodology. The results show that the predictions closely match the measurements in all cases and provide sufficient precision to select the accelerator that offers the most favourable tradeoffs between performance and engineering effort.

Chapter 5

Dose Delivery Reconstruction

5.1 Introduction

The DPM is a fast and precise MC simulation. It is optimised specifically for photon and electron radiotherapy treatment planning for dose calculations. Utilising MC methods, the DPM code simulates the transportation of photons and electrons through various materials to derive resulting dose distributions. Key components of the DPM code encompass mathematical models for photon and electron interactions, particle transport, energy deposition, and subsequent dose computations, as explained in the study by Sempau et al. [SWB00].

In the domain of radiotherapy, MC simulation is widely acknowledged as the most precise technique for computing dose distribution [SWB00]. However, MC is computationally intensive as it computes the trajectories and dose depositions of numerous individual particles traversing biological tissue through sampled particle interactions, adhering strictly to fundamental laws of physics. Yet, owing to the stochastic nature of MC algorithms, simulating a substantial volume of particles becomes necessary to achieve statistical significance. Each particle simulation incurs considerable computational overhead due to the complexity involved in interaction sampling, as discussed in studies like Ziegenhein et al. [ZPPKO15]. Consequently, despite their accuracy, the direct real-time application of MC algorithms remains a significant challenge.

Recent findings in particle physics illustrate the potential for achieving notable performance enhancements through the utilisation of GPU and FPGA accelerators, as evidenced by studies such as Carrazza et al. [CCMRZ21], Blyth et al. [Bly21], and Voss et al. [VZV⁺19]. Consequently, the integration of accelerators necessitates careful consideration within the framework of DPM. Nevertheless, it is essential to acknowledge that programming accelerators pose inherent challenges, and they might not universally guarantee substantial performance improvements, as highlighted by Voss et al. [VKM⁺21].

Therefore, it becomes imperative to initially analyse and model DPM within the context of accelerators. If an optimised architecture tailored for accelerators exists, its implementation becomes viable. However, in instances where such an optimised architecture is not feasible, exploring alternative solutions becomes necessary.

In this chapter, DPM is firstly analysed and modelled. The model forecasted that DPM would not benefit from FPGA or GPU acceleration, hence various optimisations were proposed in order to meet real-time requirements on CPU. The results show that using multi-core machines enables the simulation to meet real-time requirements.

The main contributions of this chapter are:

- A performance model for DPM that allows to forecast accelerator's performance;
- Several optimisation methods that can be applied to DPM and any other MC simulations;
- An Optimised-DPM (ODPM) implementation capable of meeting real-time requirements

There are several state-of-the-art solutions available for delivering dose reconstruction. ODPM differs from these solutions in two primary aspects. Some solutions fail to meet the demands of real-time adaptive radiotherapy, while others, although meeting time requirements lack robust validation or strong guarantees for their results. Amadio et al. [AAB⁺23] utilise GPU acceleration to speed up MC electromagnetic showers for the CERN calorimeter. However, their study highlights challenges in managing memory access without providing reconstruction times. Sharma et al. [SKF⁺18] developed rapid MC simulation tools for dose estimation

that utilise GPU acceleration. Nevertheless, they achieve a performance of 48 seconds, falling short of meeting time requirements. Li et al. [LDW⁺21] adopt a similar approach for gDPM, employing GPU acceleration to expedite radiotherapy dose reconstruction. gDPM can reconstruct the radiotherapy dose distribution in 40 seconds. Although they do not meet real-time requirements, they support the simulation of magnetic fields—a necessary feature for the MR-linac, as the MRI scanner necessitates a 1.5T magnetic field to operate. Pastor-Serrano and Perkò [PSDH⁺23] present DOTA, a CNN-based framework capable of predicting MC dose distribution. They are among the fastest in the literature, requiring between 10 – 15 seconds to compute a full dose distribution. Although DOTA does not meet real-time requirements, it offers a compelling alternative to MC-based algorithms. Voss et al. [VZV⁺20] implement dose reconstruction on multiple FPGA platforms, showing promising results as they meet real-time requirements. However, their study does not mention the validation performed to assess the validity of their results.

The remainder of this chapter is as follows: section 5.1 introduces the subsequent topics discussed in this chapter. Following this, section 5.2 conducts an in-depth analysis of the application, with subsections dedicated to FPGA performance modelling, GPU performance modelling, and comparison between platforms. Section 5.3 delves into the implementation and optimisations, covering aspects such as branch mispredictions, cache misses, elementary functions, and Random Number Generator (RNG). In section 5.4, experimental results are presented, including system configuration, test configuration, validation, comparison with Geant4, benchmarks, and scaling analysis. A discussion of the findings is provided in section 5.5, followed by a summary of the chapter’s key points in section 5.6.

5.2 DPM acceleration analysis

This section shows the application of the methodology for acceleration presented in section 4.2 to DPM.

5.2.1 Application analysis

The first step of the methodology requires analysing the application in order to determine which accelerator can achieve the performance requirements (if any), determine the acceleration target and create an initial partitioning between hardware and software. The first step analyses also the features that the application requires. These features need to be available on the target accelerator to implement the application. In the GPU case, NVIDIA GPUs with compute capability 1.3 or more recent support double precision. AMD GPUs also support double precision. However, not all Intel GPUs support double precision as the OpenCL standard does not require double precision arithmetic support.

ODPM should guarantee that the resolution of the results achieves at least 0.5mm accuracy while passing a Kolmogorov–Smirnov test with $\alpha = 5 \times \sigma$ on reference data. This is a more relaxed accuracy requirement compared to Geant4 [AAA⁺03] which guarantees accuracy in the order of nanometres. Hence, as section 5.4.3 will show, DPM can use different models for the physics and achieve multiple orders of magnitude speedup. DPM, however, still requires the use of double-precision floating points to calculate the mean free path. Differently from Coulomb Scattering where the entire computation can be carried out in single precision floating points while meeting the accuracy requirements DPM still requires double precision. Hence, any accelerator requires double-precision floating points to accelerate ODPM. FPGAs support double-precision while most vendors offer APIs for floating-point arithmetic, certain devices might require manual implementation if no open-source library is available. However, it is worth pointing out that there are some components in DPM where the accuracy requirements are less stringent. In these cases, single-precision floating points are used and as section 5.3.3 will show, approximations can be deployed to achieve higher performance.

To achieve optimal performance on accelerators such as FPGAs and GPUs, the simulation's ability to run in parallel plays a crucial role. These platforms heavily rely on parallelism to achieve high throughput. As a result, the initial step of the analysis involves assessing the potential for parallelisation within DPM.

Similar to Coulomb Scattering, DPM demonstrates an embarrassingly parallel nature, allowing each particle to be simulated independently. However, this requires a parallel random number generator and a coordination mechanism to prevent race conditions when updating radiation deposition within the geometry. Notably, dedicated parallel random number generators exist for this purpose and will be discussed further in section 5.3.4.

On both FPGA and GPU platforms, executing parallel updates to the geometry is feasible. For FPGA, employing a combination of adders and multiplexers facilitates parallel addition, enabling each clock cycle to incorporate either zero or the resulting value from the kernel computation. This tree-shaped structure, connecting the leaves to the kernel and the root to the geometry, enables parallel addition. However, such an architecture requires consideration of increased resource requirements and latency within the model. In the case of GPU, NVIDIA GPUs support *atomics*. When targeting other vendor GPUs, it is necessary to verify that the atomics are supported, as this support is only available in OpenCL 3.0 onwards,. Atomics are necessary for parallel geometry updates. However, it's important to note that *atomics* introduce computational overhead on GPU, prompting the performance model to analyse their impact on performance.

In conclusion, both GPUs and FPGAs can be used to accelerate ODPM. In the GPUs case, it is necessary to verify that the target GPU offers double-precision arithmetic and atomics support. The NVIDIA GPU used in this study offer both capabilities.

Figure 4.2 shows the application analysis deciding process. Firstly, DPM falls in the high-throughput category as it requires simulating a large number of particles in a millisecond timescale. Then the analysis requires verifying the cache miss rate and the branch misprediction rate.

By profiling the code using *perf*, it is shown that DPM suffers from a small branch misdirection rate relative to the total instruction executed with 3% of the total branches being missed. However, the cache misprediction rate is high, as 19% of all cache accesses are cache missuses. The reason for this behaviour is that DPM supports both electrons and photons. While electrons are relatively cache-friendly, photons manifest the opposite behaviour. Photons might be

scattered quite far hence any geometry storage layout would be inefficient.

Given the characteristics of the workload, it is likely that the simulation can benefit from FPGA acceleration, according to the process shown in figure 4.2. As applications need to have both low branching and low cache miss rate to be efficient on GPU. However, further analysis is required to determine whether FPGA offers a significant performance improvement.

5.2.2 FPGA performance model

Section 5.2 outlined the characteristics of DPM. The most promising platform to use is forecasted to be an FPGA. Since, in our experience, FPGAs are hard to design and program. This section describes a starting architecture designed with optimistic assumptions and ease of implementation. Then the architecture is refined with more realistic assumptions. If an optimistic architecture does not provide sufficient speedup, any further engineering effort would be pointless.

The starting architecture assumes that it is possible to design a prefetcher capable of minimising and hiding memory data transfers. Hence, it would be possible to fully exploit an FPGA to execute the simulation. This greatly simplifies the model as only $\mathcal{T}_{compute}$ needs to be calculated. Another important assumption made here is that the architecture is assumed to be fully pipelined. This means that maximum throughput is achieved, which corresponds to a particle requiring only a clock cycle to do a simulation step. The final assumption is not accelerated

portions complete instantaneously. The best case can be summarised as follows:

$$\mathcal{S} = 10 \times 10^6$$

$$n_{photons} = 100,009,805$$

$$n_{electrons} = 37,895$$

$$n_{iteration\ photons} = 140$$

$$n_{iteration\ electrons} = 12$$

$$f = 300MHz$$

$$\mathcal{T}_{mem} = 0$$

$$\mathcal{T}_{comm} = 0$$

The number of particles to be simulated $\mathcal{S}$ is an input to DPM. However, as explained in section 2.1.4, some of the physical processes simulated create secondary particles. Hence, even if the number of photons and electrons specified in input is fixed the actual number of particles simulated is larger. $n_{photons}$ and $n_{electrons}$ show the average number of particles simulated given the input $\mathcal{S}$.

The next step then is to forecast $\mathcal{T}_{compute}$, the amount of time the FPGA employs to compute the entire simulation under these assumptions. To compute $\mathcal{T}_{compute}$ we first need to determine the parallelism achievable on FPGA. To achieve this there are only two possible strategies, one is manually counting the number of operations required by the simulation, and the other is to rely on previous research results where similar simulations were accelerated to gather realistic estimates. As a starting point, previous results can be used and the model can be refined as necessary, however, these results should be either precise or optimistic compared to the actual simulation, latter refinement will require manual analysis of the simulation to obtain more precise measurements.

Voss et al. [VZV⁺20] analyse and accelerate a simplified version of DPM. Their implementation only simulated electrons, in particular, they consider only Moller and Hinge scattering which are only two of the many components of DPM. They achieve a parallelism of 6 when targeting

a Xilinx VU9P FPGA and a parallelism of 8 when targeting Xilinx VU13P FPGA. Hence, for the purpose of the model, we could assume a parallelism n_{cycle} of 8 as a best-case scenario as DPM requires additional area-expensive features that will likely reduce the maximum achievable parallelism.

Setting $n_{cycle} = 8$ and substituting the hypotheses shown above in Equation 4.4 we obtain $\mathcal{T}_{compute} = 5.85s$. In particular, the number of clock cycles required is given by:

$$n_{electrons} \times n_{iteration\ electrons} + n_{photons} \times n_{iteration\ photons}$$

this value is then divided by $n_{cycle} \times f$ to obtain the computational time.

This is an optimistic estimate that does not account for the data transfer from and to the FPGA. This is slower than the 1s time limit for real-time use. Hence, implementing this architecture on a Xilinx VU13P FPGA will not solve the performance problem. It is worth mentioning that the Xilinx VU13P FPGA is one generation behind the latest Versal FPGAs. The Versal VE1752 and VE2802 have an array of over 300 VLIW vector processors called AI Engines, which may potentially boost performance.

5.2.3 GPU performance model

Section 5.2 revealed that the NVIDIA GeForce RTX 3090 Ti GPU may not outperform CPUs in accelerating DPM due to limited cache size and the absence of branch predictors. Figure 4.5 shows that the number of CUDA cores has greatly increased in recent years and building a performance model does not require a high amount of engineering effort. Hence, building a model is a useful analysis while the findings might not solve the issue at present, they offer useful guidance and in the future, when GPUs will have bigger caches and more CUDA cores, they might be the best platform to execute DPM. One of the bottlenecks of DPM is the amount of cache misses. The NVIDIA GeForce RTX 3090 Ti used in this study differently from the AMD EPYC 9654 has much smaller caches and does not have prefetchers. Hence, reducing the amount of cache misses is crucial to achieve high performance on the GPU. It may be possible

to reduce the amount of cache misses by:

1. Partition the geometry;
2. Simulate all the particles in one partition while loading another partition;
3. Repeat the previous step until completion.

If this approach could be adapted to GPUs, they could offer significantly higher performance due to their greater parallelism. Despite its promise, this approach faces several challenges. First, for the NVIDIA GeForce RTX 3090 Ti used in this study, 6 MB of on-chip memory might not be sufficient. Given that geometries contain millions of voxels, which correspond to several gigabytes, the number of partitions smaller than the cache size would be quite large, resulting in high overhead as each partition has a particle queue to maintain. Furthermore, since partitions contain too few particles, the time needed to simulate them is less than the time required to load the next partition into memory, resulting in low performance. Moreover, the small partition size leads to multiple threads updating the same voxel, this generates a great synchronisation overhead. Updating voxels requires the use of atomics. Atomic operations on GPUs can be quite expensive in case of high contention. For example, CUDA official documentation states that atomics can be implemented using the following code:

```
--device-- double atomicAdd(double* address , double val)
{
unsigned long long int* address_as_ull =
    (unsigned long long int*)address;
unsigned long long int old = *address_as_ull , assumed;
do {
    assumed = old;
    old = atomicCAS(address_as_ull , assumed ,
        __double_as_longlong(val +
            __longlong_as_double(assumed)));

// Note: uses integer comparison to avoid hang
```

```

// in case of NaN (since NaN != NaN)
} while (assumed != old);

return _longlong_as_double(old);
}

```

In case of contention, the code above can achieve low performance. It contains a *while loop*. This loop can be classified as a branch, hence at each point in time only one thread will successfully update the value, while the others have to retry. Differently, from CPUs the thread that updated the atomic variable cannot proceed to perform useful work, instead, it is de-scheduled until all the other threads composing the warp successfully update the value. This results in a serialisation of the update that limits the parallelism of NVIDIA GeForce RTX 3090 Ti and in low performance.

In conclusion, an architecture that minimises cache misses such as the one proposed here is unlikely to achieve high performance on the NVIDIA GeForce RTX 3090 Ti. To further evaluate this claim the proposed architecture was implemented in software on CPU. When subgeometries of a size compatible with the NVIDIA GeForce RTX 3090 Ti cache size were used, the core results over one order of magnitude slower, sometimes even failing to terminate in a reasonable amount of time, hence strengthening the hypothesis that this architecture is inefficient on the NVIDIA GeForce RTX 3090 Ti.

Amadio et al. [AAB⁺23] implemented an EM MC simulation that is similar to DPM. They also accelerated the simulation on a GPU. Their results validate our forecasts, as they show that the geometry is one of the main bottlenecks of the simulation, which limits the device utilisation to a small fraction of the theoretical maximum. In their study, this inefficiency is inherent to the geometry model and is irreducible, even after performing a number of optimisations.

5.2.4 Comparison between platforms

Section 5.2.2 and 5.2.3 show the performance model and forecasts for both Xilinx VU13P FPGA and NVIDIA GeForce RTX 3090 Ti GPU. DPM is forecasted to be inefficient on both platforms, the Xilinx VU13P does not have enough resources to achieve higher performance than the AMD EPYC 9654 CPU and NVIDIA GeForce RTX 3090 Ti GPU due to their programming model and small cache size. Thus, both devices fail to offer a compelling advantage compared to AMD EPYC 9654. Previous studies from Voss et al. [VZV⁺20] and Amadio et al. [AAB⁺23] confirm this forecast by showing results similar to the ones predicted by the model. Voss et al. achieve a 7s simulation time on a Xilinx VU13P FPGA with a parallelism of 16 when using the same pencil beam as the one assumed in section 5.2.2. They also implement only a third of the physics present in DPM hence, when compared with the results shown by Voss et al., the forecasts presented in section 5.2.2, although optimistic do not meet real-time requirements and further refinements to the architecture would be pointless as the real performance will likely to be even lower than the forecasts. Amadio et al. do not provide exact performance measurements for their GPU acceleration of DPM but they claim that “*the geometry was identified as being the main bottleneck in the current project phase, limiting the device usage to a small fraction of the theoretical maximum*”. Hence represents an MC simulation that does not benefit from acceleration when targeting the NVIDIA GeForce RTX 3090 Ti GPU or the Xilinx VU13P FPGAs. In this case, the performance model saved hundreds if not thousands of hours of needless engineering effort.

The main difference between Coulomb Scattering presented in chapter 4 and DPM is the presence of photons. Coulomb scattering is computed only using electrons. Electrons have a high locality. Each iteration they move to neighbouring voxels. This means that at the beginning there is some performance degradation due to the atomic addition of radiation doses in the same voxel. However, after the first few iterations of the simulation, they move away from each other but they still maintain some locality. This greatly reduces the amount of atomic operation required but it is relatively cache-friendly. Most of the voxel accessed are present in the cache. DPM shows a completely different behaviour. Photons start near each other and

then travel quite far at each iteration, this results in low performance for two reasons. When the photons are close to each other, atomic addition serialises the computation. When they move away from each other the cause cache misses due to accessing voxels in a stochastic not sequential manner. This limits the performance achievable by GPU in this type of simulation. In the FPGA case, Coulomb Scattering requires far fewer resources to be implemented, this leads to a higher parallelism. DPM however being a more complex simulation that involves multiple processes requires more resources. The resources available on a Xilinx VU13P or VU9P are not enough to achieve throughput and parallelism required. Future FPGAs might offer more resources that can potentially meet the real-time performance requirements.

5.3 Implementation and optimisations

Given the analysis shown in section 5.2, an optimised CPU implementation of DPM is the best solution to meet real-time radiotherapy requirements. To this end, from this section onwards we present ODPM, our custom DPM implementation. We refer to DPM to the original implementation by Sempau et al. [SWB00] and to ODPM as our implementation.

The structure of ODPM proposed here consists of three components: Beam, Geometry and Physics. Beam is an interface where the user can implement any source of particle, the only requirement is to provide a function that generates a particle. Geometry is also an interface that offers a 3D addressable structure that returns an *std::optional* containing either a Voxel or None in case the particle is out of bounds. Geometry also needs to provide a method to compute the distance between the particle and the boundary. This is required to compute the mean free path. Physics contains the implementation of Hinge, Moller and Bremsstrahlung in the case of electrons, annihilation in the case of protons, and Compton scattering, pair production in the case of photons. Moreover, Physics offers the tracking infrastructure that uses the mean free path to determine the events.

The performance of this MC simulation is limited by four main factors, branch mispredictions, cache misses, elementary functions, and random number generation. The following sections

analyse these factors in detail and show the techniques used to meet real-time requirements. The DPM implementation by Sempau et al. [SWB00] is single-threaded hence its performance is limited. Ziegenhein et al. proposed PhiMC which is 37 times faster than the original DPM code. However, their results show that their optimised implementation is still one order of magnitude slower than real-time radiotherapy requirements.

5.3.1 Branch mispredictions

MC simulations are stochastic in nature, hence they often suffer from branch misprediction, that is when the CPU branch predictor fails to correctly predict the outcome of a branch. This causes a misprediction penalty. The misprediction penalty consists of the CPU unravelling the work done and resuming the work from the correct branch resulting in severe performance degradation. However, there are several techniques to mitigate this issue: using algorithms with fewer branches, algorithms that have predictable branch patterns or applying “*Branchless Programming*”. Branchless Programming consists of a set of methods that can be used instead of branches, such as bit-wise operations that zeros one side of the branch, or storing the results in an array using the result of the comparison to draw the correct value. However, it is important to mention that these changes must be validated with proper benchmarks. Most commercial CPUs like Intel and AMD support x86 and x86-64 assembly, which belongs to the Complex Instruction Set Computer (CISC) family. CISC assembly might already offer an instruction to remove the branch without changing the semantics of the program. This particular instruction can be manually forced by using intrinsics, inline assembly or compilers could optimise the branch out with an equivalent operation. Hence, it is crucial to analyse the assembly generated and to measure the performance of both the branch and the branchless version using proper benchmarks. Pikus provides an introduction to branchless programming and the most common techniques in [Pik21], including some of the techniques applied below.

The following listing shows a portion of the tracking code in DPM. It can be noted that the code has several branches. The actual event is determined stochastically using the mean free path which makes it hard for the CPU branch predictor to determine the correct result.

```

1  if (event == BREMSSTRAHLUNG) {
2      state.numBremMFP = 0;
3  } else {
4      state.numBremMFP -= stepLength * delNumBremMFP;
5  }
6
7  if (event == MOLLER) {
8      state.numMollerMFP = 0;
9  } else {
10     state.numMollerMFP -= stepLength * delNumMollerMFP;
11 }
12
13 if (event == HINGE) {
14     state.numTr1MFP = 0;
15 } else {
16     state.numTr1MFP -= stepLength * delNumTr1MFP;
17 }

```

The following listing shows an alternative that uses bitwise operations to avoid the branches:

```

1  const auto delBrem = std::bit_cast<uint64_t>(state.numBremMFP -
        stepLength * delNumBremMFP);
2  const auto delMoller = std::bit_cast<uint64_t>(state.numMollerMFP -
        stepLength * delNumMollerMFP);
3  const auto delTr1 = std::bit_cast<uint64_t>(state.numMollerMFP -
        stepLength * delNumTr1MFP);
4
5  const auto isBrem =
6      static_cast<std::int64_t>(-(event != BREMSSTRAHLUNG));
7  const auto isMoller =

```

```

8      static_cast<std::int64_t>(-(event != MOLLER));
9  const auto isHinge =
10     static_cast<std::int64_t>(-(event != HINGE));
11
12  state.numBremMFP = std::bit_cast<double>(delBrem & isBrem);
13  state.numMollerMFP = std::bit_cast<double>(delMoller & isMoller);
14  state.numTr1MFP = std::bit_cast<double>(delTr1 & isHinge);

```

The second version calculates all three values in one go and then uses bitwise operations to update the Mean Free Path (MFP) either with a 0 or with the reduction value given by the step length. The variables *isBrem*, *isMoller*, *isHinge* are set to all ones or all zeros by casting -1 or 0 to an unsigned number. Then this value is used as to zero or leave the *delBrem*, *delMoller*, *delTr1* unchanged.

However, it is important to notice that the branchless version executes more operations than the first one. In the first version, when the branch is true only an assignment is executed instead of a fused multiply-add. Moreover, the bitwise version has to convert the double floating points to integers, perform a bitwise operation and convert it back. Thus, it is important to test the two versions to measure which one of those is faster. The results show that the second version is faster than the first one. This is likely to be the case because the extra operations added, except for the fused multiply-add are relatively fast. The conversion is done by a register swap, while the bitwise operations are known to be fast on CPU.

The update shown in lines 1-3 can also be done using SIMD instructions and by filling the last 64 bits of the 256bit vector register with zeros. It is important to test if SIMD improves the performance as using SIMD requires moving data from scalar to vector registers and vice versa. The compiler has internal cost models that forecast whether SIMD will result in faster execution or not and emit the best assembly. However, these cost models sometimes can be wrong or not tailored to the particular architecture used, hence, when seeking the highest performance it is possible to use intrinsics to force or disable SIMD. Then a benchmark can be executed to find the fastest version.

The following listing shows another example code with several branches:

```

if (m_x[ilow + 1] <= xval) {
    ilow += 1;
}

if (m_x[ilow] > xval) {
    ilow -= 1;
}

```

This code performs a boundary condition check and depending on the value of `ilow` and `xval` adjusts `ilow` to the correct element. By analysing the code using `perf` the CPU branch predictor fails to determine the correct branch in most cases causing a performance degradation. The following listing shows a different implementation that does not use branches:

```

ilow += m_x[ilow + 1] <= xval;
ilow -= m_x[ilow] > xval;

```

In this case, it is possible to directly sum the comparison result to the index. This has the same semantics as the above code as it sums either 0 or 1 depending on the comparison. However, the resulting code, although might perform two integer sums instead of zero, is often faster.

The last example of branchless programming is found in the following transformation. The following listing shows the code for a geometrical boundary condition check:

```

const ThreeVector boundary{
    (particle.direction.x > 0) ? maxBoundary.x : minBoundary.x,
    (particle.direction.y > 0) ? maxBoundary.y : minBoundary.y,
    (particle.direction.z > 0) ? maxBoundary.z : minBoundary.z
};

```

Depending on the particle direction one Voxel boundary or another is picked. As in the previous examples, the code is stochastic and the CPU branch predictor fails to determine the correct

branch incurring a performance penalty. However, the code can be transformed using one of the most common techniques for branchless programming:

```
const real_type boundary_x[2] = {maxBoundary.x, minBoundary.x};
const real_type boundary_y[2] = {maxBoundary.y, minBoundary.y};
const real_type boundary_z[2] = {maxBoundary.z, minBoundary.z};
const ThreeVector boundary{
    boundary_x[particle.direction.x < 0],
    boundary_y[particle.direction.y < 0],
    boundary_z[particle.direction.z < 0]
};
```

This version stores the results in an array and uses the result of the comparison to address the array. This transformation stores the possible results in an array and then uses the result of the comparison to determine the correct result. However, this can be optimised further:

```
const real_type boundary_x[2] = {maxBoundary.x, minBoundary.x};
const real_type boundary_y[2] = {maxBoundary.y, minBoundary.y};
const real_type boundary_z[2] = {maxBoundary.z, minBoundary.z};
const ThreeVector boundary{
    boundary_x[std::signbit(particle.direction.x)],
    boundary_y[std::signbit(particle.direction.y)],
    boundary_z[std::signbit(particle.direction.z)]
};
```

Floating point numbers store the sign bit explicitly. Hence, instead of doing a comparison, it is possible to retrieve the sign bit directly and use it to address the array. This results in even faster performance.

The main issue of branchless programming is its inability to eliminate all hard-to-predict branches. In the case of the hot loop within ODPM, where the majority of processing time is consumed, the task of “Electron Tracking” is particularly affected. This section of the code

is responsible for advancing electrons through space and determining whether to execute the Moller, Hinge, or Bremsstrahlung processes. Within this loop, three branches are essential and cannot be bypassed. Attempting to execute all three processes and then undoing the two unnecessary ones would be significantly less efficient, with performance being orders of magnitude slower than executing the correct process directly. Nevertheless, the remaining branches within this loop follow the optimised approach discussed earlier in this section, with the provided examples being drawn directly from this segment of the code.

By combining the “branchless programming” techniques shown above and applying them where possible, It is possible to greatly speed up any MC simulation as most MC code requires several branches.

5.3.2 Cache misses

Multidimensional data are commonly stored using either row-major or column-major storage orders, depending on the use case and device. Algorithms that traverse a matrix row by row will go faster when the matrix is stored in row-major order because of better data locality. Similarly, column-by-column traversal is faster for column-major matrices. This also applies to external accelerators such as GPUs. These storage orders can be further generalised to the multi-dimensional case, i.e. in the 3D case, it is possible to store a slice row-by-row and then each slice along the z -axes.

DPM simulation suffers from severe cache misses. The issue is mainly caused by photons that can travel quite far in this simulation hence, any column-major or row-major storage orders are not efficient as particles are unlikely to travel in a straight line parallel to the chosen storage order.

Storage orders that leverage space-filling curves, functions that map multidimensional data to one dimension, such as Hilbert [Sag94] and Morton [NR10] are an alternative to column and row major orders described above. Depending on the application these storage orders can be cache friendly and achieve significant speedup. However, these curves are not guaranteed to reduce the

cache misses as they are application-dependent, moreover computing those curves introduces additional overhead. Hence, their impact on performance needs to be validated using proper benchmarks. In DPM's case, Hilbert curve indexing introduces too much overhead as even the fastest algorithms require a significant amount of operations to compute the indices [Ski04]. Morton transform is more promising. Morton transform can be implemented efficiently using bitwise operations and can be also vectorized for additional speedup. Thus it is very efficient and introduces a negligible amount of overhead to the simulation. The benchmarks, however, show that in this case this transform increases cache misses instead of reducing them and results in a 20% increase in computational time. Given the results achieved using out-of-the-box storage orders, to mitigate cache misses is necessary to develop a custom storage strategy for this MC simulation.

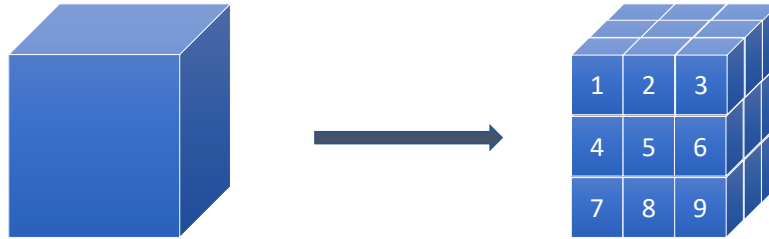


Figure 5.1: Memory storage strategy for DPM. The geometry is divided into subgeometries. While one section is computed another can be loaded in memory.

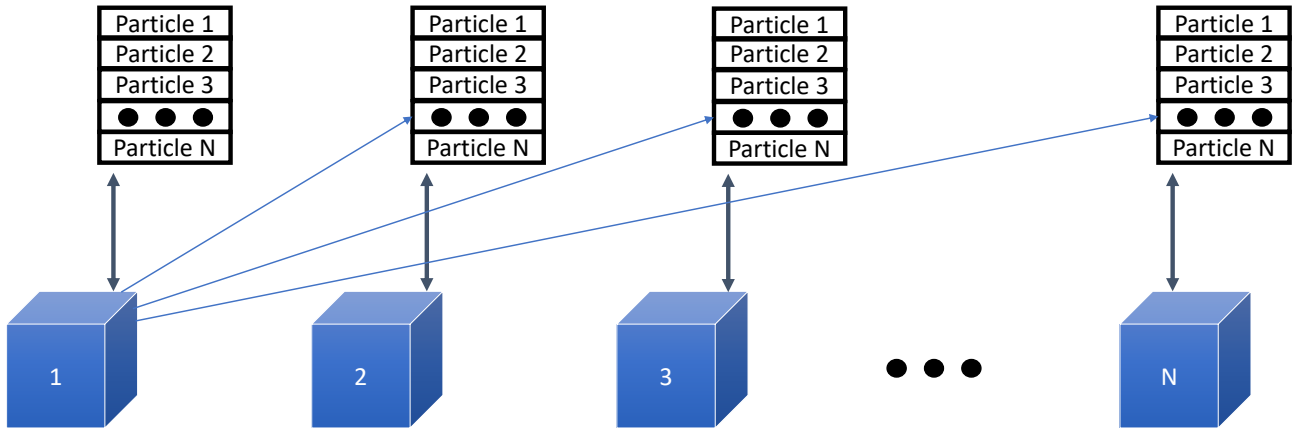


Figure 5.2: Memory storage strategy for DPM. Each subgeometry is paired with a particle queue. When simulating one subgeometry, particles that escape to another subgeometry are added to the corresponding queue and simulated when that specific subgeometry is loaded in memory.

Figure 5.1 shows a possible storage strategy. In this memory layout, the geometry is divided into subgeometries (numbered 1,2 ...9 in the right figure). Figure 5.2 shows that each of

these subgeometries has its own particle queue. At the beginning of the simulation, the beam generates a certain $\mathcal{M}$ particles and fills $\mathcal{K}\%$ of the queues. Then the simulation starts. The subgeometries are ordered by the filling percentages of the queue, and the subgeometry where the corresponding queue has the highest number of particles is simulated. While one subgeometry is simulated, the subgeometry with the second fullest queue is loaded in memory so that once the queue is empty the next subgeometry is already loaded. Particles are loaded from the queue and eventual particles generated are stored into the queue. Particles are simulated until they run out of energy or escape from the subgeometry. If they escape from the medium entirely they are removed from the simulation, if they escape into another subgeometry they are stored in the corresponding queue. Once the fullest queue has less than $\mathcal{P}$ particles, the beam generates another $\mathcal{M}$ particles. This process is repeated until all the particles are computed.

According to the tests, this storage order greatly reduces the number of cache misses. However, the actual performance is actually lower compared to a standard column-major layout. This is mainly due to the overhead required to maintain different queues and subgeometries. Efficient spacial-addressable data structures reduced overhead but the total performance was still lower. This underlines the importance of benchmarks to carefully evaluate the performance achieved by any optimisation.

After realising that partitioning was not improving performance the next step was to measure and reduce cache-to-cache communication overhead. The result of the MC simulation is a 3D dose distribution, that is the amount of radiation contained in each voxel. Since the doses are additive and commutative, multiple threads could work separately on different copies of the geometry with no synchronisation and the results can be aggregated in the end. Having each thread working on a copy of the geometry might not be a good idea as the memory occupancy might explode. For example, a server with 128 cores and a geometry that requires 20 GB would consume $128 \times 20 = 2,560$ GB of memory. Also, copying and merging this many replicas requires at least 85s using recent DDR4 memory as read/write bandwidth is limited. A possible approach is to have a copy of the geometry for each CPU or for each CCX (section 2.4 provides a definition for CCX), however depending on the number of CCX, this copy might still take several seconds. As with any other approach this solution must be properly tested

using benchmarks. Given the hard limit of 1s of real-time adaptive radiotherapy and the size of geometries commonly used in radiotherapy, ODPM replicates one geometry per CPU. Results in section 5.4.5 will provide more details.

5.3.3 Elementary functions

Functions such as trigonometric functions, logarithms and square roots are usually called “*elementary functions*”. Since their implementation is quite delicate and any minor mistake or poor handling of corner cases can cause major accuracy problems they are usually provided by a library and is unlikely for developers to implement them. However, most library implementations focus on providing the most accurate answer [DDDM03, Mar00]. This limits the performance achieved by these libraries. Alternatively, it is possible to use polynomial interpolation or other techniques to increase the performance of such implementations by specifying the accuracy requirements [BKO⁺21]. However, when using less accurate implementations careful consideration of the use of such functions is required. Independently of the particular application, they can only be used where the accuracy requirements are clearly defined and the application must be thoroughly tested, including edge cases, to ensure reliability.

The application analysis shows that 13% of the total time was spent computing logarithms. The MC simulation heavily relies on function interpolation to avoid computing expensive complex functions, distributions and integrals at runtime. These tables are mostly logarithmic, that is to say that they are addressed using the logarithm of the value ($\log(v)$) instead of the value (v) itself. Thus, every time the tables are accessed, a logarithm is computed. However, the accuracy requirements for such a table are quite low, since the result is rounded to an integer three decimal places suffice, in the worst case instead of the correct element a neighbouring one could end up being accessed. This error can be easily fixed by storing the boundaries in terms of the original value (v) and by checking if the value falls within the margin of the accessed element, i.e $i = \text{int}(\log(v)); V[i] \leq v < V[i + 1]$. Then i can be decreased or increased by one to make sure that the correct element is accessed.

The logarithm is the inverse of the exponential function. A fast logarithm that meets the

accuracy requirements can be obtained by inverting Schraudolph’s algorithm for the exponential function [Sch99]. The resulting implementation developed using the constants provided in the open source approximation library available on GitHub <https://github.com/ekmett/approximate/>, when tested on an AMD EPYC 9654 and compiled using GCC 12, is 57 times faster than `std::log` while meeting the accuracy requirements. This reduces the time spent on computing logarithms from a significant fraction to $< 1\%$ of the total time.

It is worth noting that not all logarithms can be approximated in ODPM. For example, ODPM uses logarithms in determining the mean free path, in this case, the accuracy requirements are much higher and approximations lead to numerical inaccuracies. Hence, as shown above, it is necessary to analyse the accuracy requirements of each function, given the context where it is utilised. In general, developers can analyse which function consumes most of the runtime, where it is used and what are the specific accuracy requirements. If the requirements are more relaxed compared to the accuracy guarantees of standard implementation. Then it is possible to replace these functions with carefully selected approximations.

5.3.4 Random Number Generator (RNG)

In this chapter, ODPM is implemented using MIXMAX. MIXMAX meets all the criteria illustrated in section 2.3 and it achieves higher performance than RANLUX [Sav15] and Mersenne Twister (MT) [MN98], while providing high-quality numbers and parallel streams. Moreover, for a long time, Geant4 (via CLHEP) has employed MIXMAX as its default generator.

5.3.5 Overall impact

The techniques outlined earlier in this section enabled ODPM to meet its real-time performance requirements. It is, however, challenging to precisely quantify the contribution of each individual optimisation, as some were integrated into the development of new algorithms, while others only affect specific CPUs. For instance, the implementation of the log functions utilises branchless programming. Nonetheless, approximate estimates can be provided.

The results below, although approximations, reflect compounded effects. These optimisations were applied sequentially, in the order presented. For example, the custom log function was the first optimisation implemented, followed by the application of branchless programming, which was layered on top. Consequently, the performance improvement gained from branchless programming builds upon the prior speedup achieved by the custom log function. These estimates are intended to guide the reader towards most impactful optimizations as not all of the might be needed to meet the time requirements.

The first optimisation applied to ODPM was an approximate log function, which initially consumed 13% of the total runtime. The new approximation, 57 times faster than the original, led to a 13% speedup, as the time spent on computing logarithms became negligible. Branchless programming further accelerated the code by $\approx 40\%$, not only due to better pipelining but also because the compiler could vectorise instructions, given that many computations were performed on three-element vectors (representing particle positions and directions). Additionally, replacing Mersenne Twister with MixMax increased the simulation's speed by $\approx 16\%$, reducing the time spent on random number generation from 22% to 6% of the total runtime. Finally, the partitioning scheme outlined in section 5.3.2 had no effect on systems with a single CPU, but reduced the total runtime by $\approx 7\%$ in the system described in section 5.4, with the minor impact mainly due to the additional overhead of copying geometry and merging results at the end.

5.4 Experimental results

The proposed implementation of DPM was evaluated on a high-end server system featuring two AMD EPYC 9654 96-core processors. Notably, input reading times and initialisation times were excluded from the evaluation metrics, as in real-world scenarios these steps are executed before starting the radiotherapy treatment.

5.4.1 System Configuration

The AMD EPYC 9654 is a 64-bit 96-code x86 microprocessor operating on 2 contexts and based on 5 nm Zen 4 “Genoa” micro-architecture. This processor features a 6 MB 8-way set associative L1 data and L1 instruction cache. It also features a 96 MB 8-way set associative L2 cache and a 384 MB 16-way set associative L3 cache. It operates with a TDP of 360W. It functions at a base clock speed of 2.4 GHz, with an all-core boost up to 3.558 GHz and can reach up to 3.7 GHz for one active core.

Compiler and Compiler Flags The benchmarks that the following sections will present are conducted on the described system configuration. The compiler used is *gcc 12.3.0*. The flags used are “-O3 -ffp-contract=fast -freciprocal-math -fmerge-all-constants -fno-math-errno -march=native -funroll-loops -ftree-vectorize -fno-trapping-math -fassociative-math -ffinite-math-only -fno-signed-zeros” with *Link Time Optimization* enabled. Flags such as “-Ofast” were discarded as they cause floating point rounding errors.

5.4.2 Test Configuration

A radiotherapy-like experiment is executed consisting of a pencil beam with fixed energy and a water phantom of size $333 \times 333 \times 128$ mm. The pencil beam is used to have a “fair” comparison with Geant4 as pencil beams do not introduce any variance in the simulation. Moreover, pencil beams are a worst-case scenario for ODPM as section 5.4.5 will show. The energy ranges between 0 and 20 MeV, the primary histories are 10^6 or 10^7 depending on the experiment. The tests are executed for both electron and photon beams. The benchmarks were repeated 5 times and the average time measurements were computed across these runs.

5.4.3 Validation

The final results of ODPM consist of a 3D dose distribution. To evaluate the accuracy of the results, these results are integrated over the z axes and we executed a Kolmogorov–Smirnov

test against the reference distributions, which concluded that the distributions are equivalent using $\alpha = 5 \times \sigma$. Section 2.2 gives more details on the Kolmogorov–Smirnov and the value used for α . The reference distributions are obtained from Geant4 [AAA⁺03] the state-of-the-art MC simulator for High Energy Physics. The physics used in Geant4 is “FTFP_BERTF” with “G4EmStandardPhysics_option4”. Figure 5.3a and 5.3b show the electron and photon distributions obtained by simulating 10^6 particles in water using a pencil beam of 6MeV. It can be noted that the lines of both simulations overlap almost perfectly. Figure 5.4a and 5.4a show the results obtained with a 15MeV pencil beam in a water, bone, titanium, water phantom. In particular, the first 10mm are water, between 10mm and 15mm is bone, between 15mm and 25mm is titanium and the rest is water. In this case, it is possible to observe an almost perfect agreement, with small differences being within the margin of error.

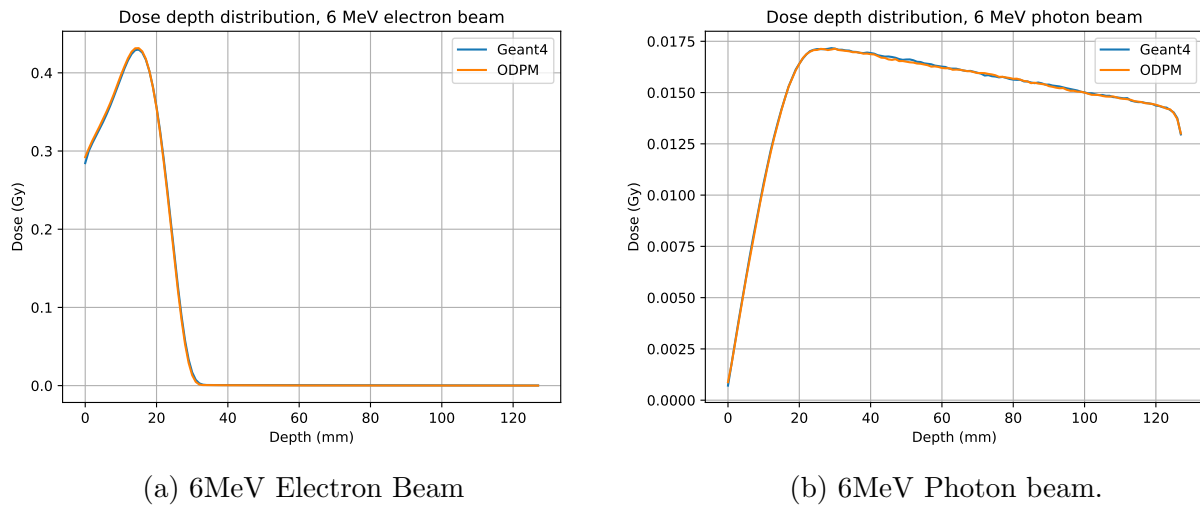


Figure 5.3: This figure shows the dose depth distribution computed in a water phantom, the number of histories simulated is 10^6 . The reference results are in blue, obtained using Geant4 while in orange are the results obtained with ODPM. On the left a 6MeV Electron beam is used, and on the right, a 6MeV photon beam is used. The Figure shows perfect agreement between the simulations

Geant4 is a more complete MC simulator compared to ODPM. It supports more particles and processes. The accuracy is also much higher, in the order of nanometres. Hence, the performance of the two simulators cannot be directly compared. The results here are meant to contextualise the performance of ODPM as Geant4 is a widely known and adopted simulator.

In the context of real-time adaptive radiotherapy, this much precision is not necessary. The

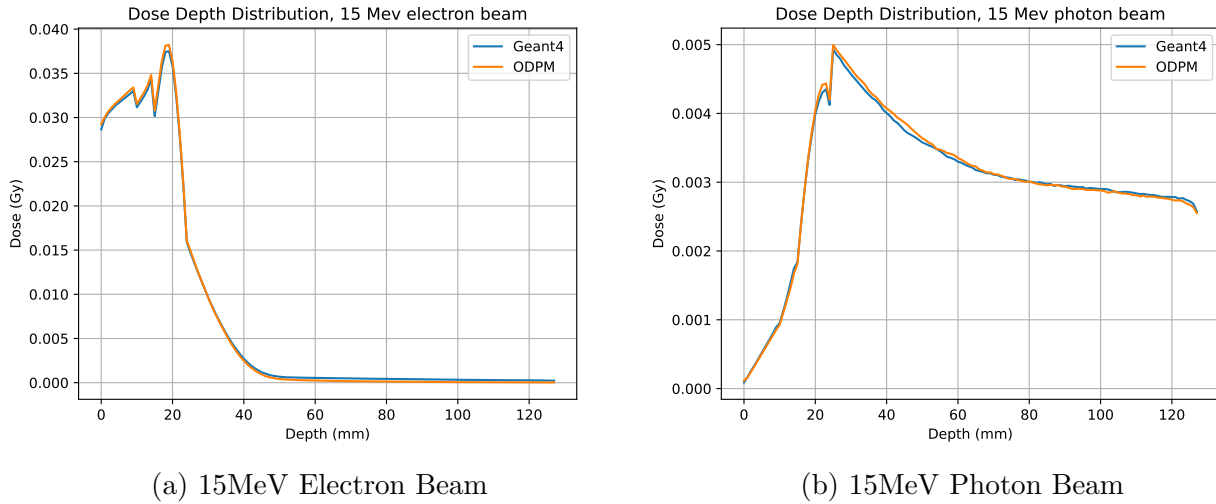


Figure 5.4: This figure shows the dose depth distribution computed in a water, bone, titanium, and water phantom. The number of histories simulated is 10^6 . The reference results are in blue, obtained using Geant4 while in orange are the results obtained with ODPM. On the left a 15MeV Electron beam is used, and on the right, a 15MeV photon beam is used. The Figure shows good agreement between the simulations

margin of error in radiotherapy is in the order of millimetres [ZPPKO15] hence, it is possible to use faster approximations. Figure 5.5 gives an idea of the possible speedup that is achievable with these approximations. The speedup ranges between two or three orders of magnitude depending on the energy level used.

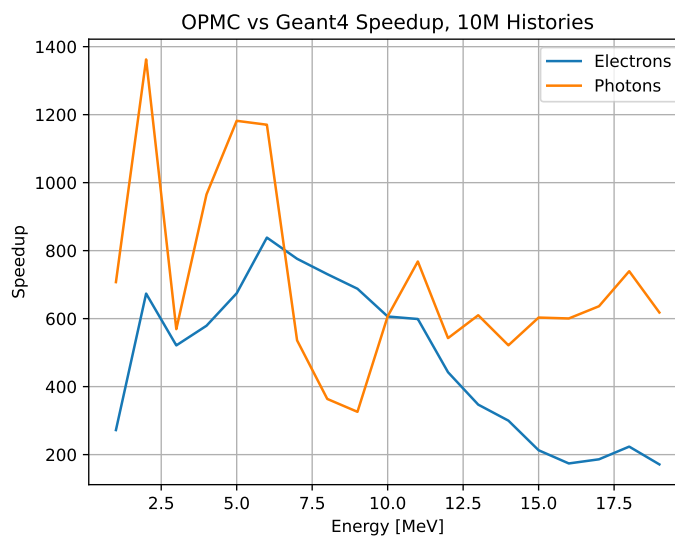


Figure 5.5: Simulation speedup compared to Geant4.

5.4.4 Performance

Figure 5.6 shows the time needed to simulate 10^6 histories at various energy levels. 10^6 histories are needed for the MC simulation to converge [ZPPKO15]. The simulation time increases with the energy level. This is mainly due to higher energy particles travelling further before being absorbed hence they require time to be simulated. Photons require roughly half the time as electrons to be simulated. This is mainly caused by the fact that Photons travel much further than Electrons. Electrons tend to travel short distances at each iteration, i.e. few voxels contain a huge number of electrons especially at the first stages of the simulation. This degrades the performance because updating the dose deposition inside the geometry is implemented using “*atomics*”. When many threads update the same voxel, one of the threads performs the update while others will have to retry, causing the performance difference between electrons and photons.

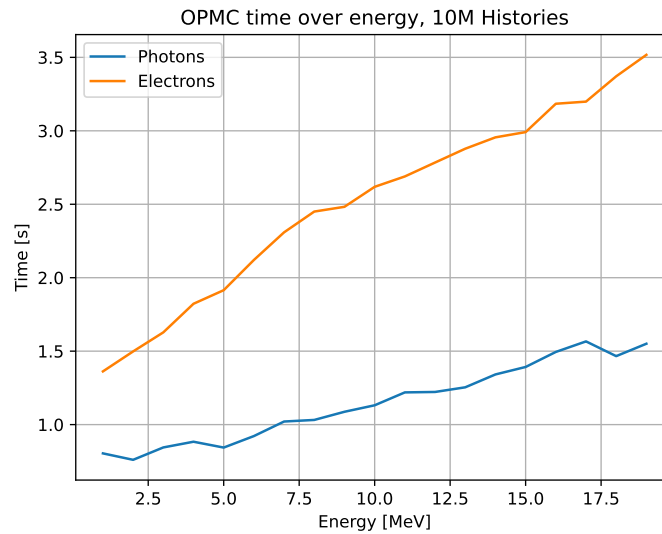


Figure 5.6: ODPM scaling.

The simulation time requirements for real-time adaptive radiotherapy are set to 1s [VZV⁺20]. While Figure 5.6 shows the entire energy range supported by ODPM, 6MeV photon beams are commonly used in real-time adaptive radiotherapy [ZPPKO15]. Hence, it is crucial that ODPM achieves sub 1s simulation time at 6MeV photon beams. The results show that the simulation meets the time requirement until 7.5MeV beams are used. In particular 6MeV beams require only ≈ 600 ms. It is worth noting that pencil beams, for the reason discussed

above are a worst-case scenario for ODPM. Clinical beams have a surface area of $10 \times 10\text{cm}$ to even $40 \times 40\text{cm}$. The usual Voxel size is $1 - 2\text{mm}$. Hence, these field sizes cause very little dose update contention, reducing the total simulation time.

5.4.5 Scaling

Figure 5.7 shows the scaling characteristics of ODPM. It is possible to notice that until 96 threads are used photon beams scale almost linearly with the number of threads. The slightly lower scaling can be attributed to a lower clock frequency achieved by the CPU as more cores are utilised and to the memory contention caused by parallel updates to the radiation dose in the geometry. After 96 threads are utilised, the behaviour changes, with a worse scaling. The

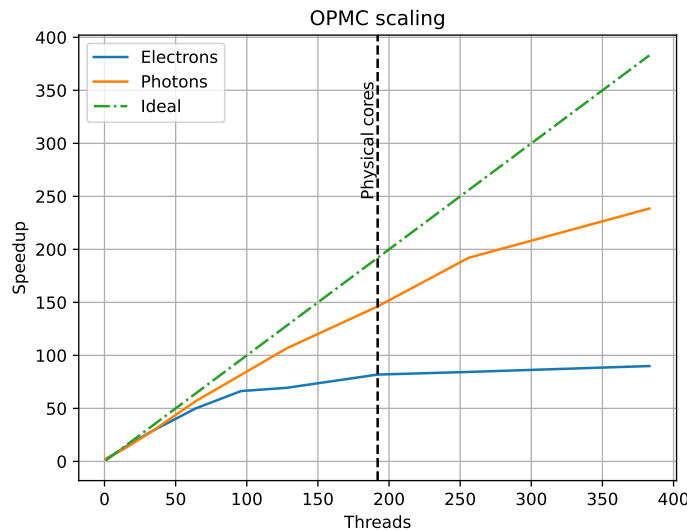


Figure 5.7: ODPM scaling.

main reason for this lies in the system used, as section 5.4.1 explained the System contains two 96-core CPUs. In these systems, cache-to-cache communication becomes more expensive, hence the geometry is partitioned so as to minimise inter CPU communications. The main cause of lower scaling in these cases is due to the replication and merging of the geometries at the end of the simulation. The geometries require several gigabytes of memory and the additional time required for the replication and merging process increases the overall execution time and lowers the scaling. The alternative to replication achieves even worse performance as

now, ODPM performs expensive cache-to-cache communications to update dose distributions atomically between the two CPUs.

5.5 Discussion

Meeting real-time dose reconstruction requirements is a challenging endeavour and to the best of our knowledge, most of the state-of-the-art implementations fall short of the minimum requirements. Not all MC simulations benefit from either GPU or FPGA acceleration. ODPM requires significantly more parallelism than the one that FPGA can provide. In the GPU case, the combination of high branching rate and stochastic memory access severely limits the performance achievable. Looking ahead, it is unlikely that these platforms will offer a compelling advantage compared to CPU for ODPM. ODPM mainly leverages DSPs for parallelism and the rate of increment of DSPs on FPGA slowed down in recent years. For example, Xilinx XC7VX690T FPGA offers 3,600 DSPs. The previous generation Xilinx VU13P greatly increased the number of DSPs available to 12,288 DSPs. On the other hand, after the Xilinx VU13P, the generation of the Xilinx VP1802 offers a small increment to 14,352 DSPs. Current era FPGAs often integrate application-specific accelerators or faster memories, such as AI-accelerators or High Bandwidth Memory (HBM). Unfortunately, these technologies do not help when accelerating ODPM. In the GPU case, performance is limited by accesses to the geometry, as Amadio et al. [AAB⁺23] state, this is an inherent characterising of some MC simulation and cannot be avoided. Unfortunately, these accesses require the use of atomics which results in severe overhead due to the serialisation of the memory updates. In recent years, GPUs improved performance by increasing the number of CUDA cores, clock frequency and memory bandwidth. However, these improvements do not affect ODPM performance in any meaningful way. Given the recent advances in both GPU and FPGA micro-architectures we do not expect these platforms to provide any meaningful advantage. However, it is important to highlight that this is not a general statement about all MC simulations but is restricted to ODPM. Future physics models could be developed with these types of accelerators in mind and could leverage AI-accelerators, HBM or minimise thread divergence.

Table 5.1: Comparison between Dose Reconstruction Methods

Framework	Method	Validation	Magnetic Field	Time (s)
This chapter	MC	Yes	No	0.925
Minimum requirements	N/A	Yes	No	1.000
Voss et al. [VZV ⁺ 20]	MC	No	No	3.575
DOTA [PSDH ⁺ 23]	CNN	Yes	No	15.000
gDPM [LDW ⁺ 21]	MC	Yes	Yes	40.000
Sharma et al. [SKF ⁺ 18]	MC	Yes	No	48.000
Amadio et al. [AAB ⁺ 23]	MC	Yes	No	N/P

Table 5.1 shows several state-of-the-art solutions exist for delivering dose reconstruction. ODPM stands out from these solutions in two key aspects. Some approaches do not meet the demands of real-time adaptive radiotherapy, while others, although meeting time requirements lack robust validation of their results. Amadio et al. [AAB⁺23] utilise GPU acceleration to accelerate MC electromagnetic showers for the CERN calorimeter, yet their investigation highlights the challenges associated with memory access without specifying reconstruction times. Sharma et al. [SKF⁺18] have devised fast MC simulation tools for dose estimation leveraging GPU acceleration, achieving a runtime of 48 seconds, falling short of the necessary time constraints. Li et al. [LDW⁺21] employs a similar approach with gDPM, utilising GPU acceleration for radiotherapy dose reconstruction, completing the task in 40 seconds. Despite supporting magnetic field simulation—an essential feature for the MR-linac as the MRI scanner leverages a magnetic field—they fail to meet real-time requirements. Pastor-Serrano and Perkò [PSDH⁺23] introduce DOTA, a CNN-based framework for predicting MC dose distribution. They boast one of the fastest computational times in the literature, between 15 seconds for a full dose distribution. Although DOTA does not meet real-time demands, it presents a compelling alternative to MC algorithms. Voss et al. [VZV⁺20] implement dose reconstruction across multiple FPGA platforms, demonstrating promising results falling slightly short of meeting real-time requirements, albeit without explicit validation procedures mentioned in their study. It is worth pointing out, that Voss et al. [VZV⁺20] provide results that meet real-time adaptive radiotherapy time requirements but in doing so they do not use a pencil beam configuration. In this study, the performance measurements are always done using a pencil beam for reproducibility reasons and because pencil beams are a worst-case scenario for this simulation. Any other beam, including

a model for the MR-linac is expected to achieve same or higher performance but not lower.

5.6 Summary

In this chapter, DPM has been analysed and, ODPM, an optimised implementation capable of meeting real-time adaptive radiotherapy constraints has been provided. Firstly, a performance model for DPM was built. The performance model shows that DPM does not benefit from GPU or FPGA acceleration.

The primary distinction between Coulomb Scattering discussed in Chapter 4, and DPM lies in the inclusion of photons. Coulomb scattering exclusively involves electrons, which exhibit a high degree of locality by moving to adjacent voxels in each iteration. Initially, this leads to some performance degradation due to the atomic addition of radiation doses within the same voxel. However, as the simulation progresses, electrons disperse while still retaining some degree of locality, thereby reducing the number of atomic operations required and maintaining cache friendliness as most accessed voxels remain in the cache. In contrast, DPM exhibits markedly different behaviour. Photons initially start in close proximity but travel significant distances in each iteration, resulting in diminished performance for two reasons. Firstly, when photons are near each other, the computation is serialised due to atomic addition. Secondly, as they move apart, cache misses occur due to accessing voxels in a stochastic rather than sequential manner. This inherent stochasticity limits the performance achievable by GPU in such simulations.

Given the performance modelling forecasts, various MC optimisations have been analysed and integrated into ODPM. Mainly, the use of approximations, fast RNGs and branchless programming. The results show that ODPM is capable of meeting real-time adaptive radiotherapy time constraints for commonly 6MeV beams used with the MR-linac [ZPPKO15]. These results could facilitate the integration of ODPM in the context of real-time adaptive radiotherapy.

Chapter 6

Conclusion

6.1 Summary of Thesis Achievements

This thesis focuses on solving real-time adaptive radiotherapy computational bottlenecks. The aim is to improve the performance of MRI image reconstruction and delivery dose reconstruction by means of both optimisations and accelerators. This thesis presents three main research contributions: (a) a new parallelisation scheme for 4D-MRI data and the resulting optimised heterogeneous implementation of the XD-GRASP algorithm [FAC⁺16], capable of meeting MRgoART [MHBB20] requirements, (b) a methodology to forecast MC performance when using GPU and FPGA acceleration and (c) Optimised Dose Planning Method (ODPM), an optimised implementation of DPM [SWB00] dose reconstruction algorithm. The following describes these contributions in more detail.

Chapter 3 presents a novel parallelisation scheme optimised for 4D-MRI data that minimises multi-threading overhead and hides data transfer latency. This results in an efficient architecture for golden angle, stack-of-stars [MP17] MRI reconstruction-based algorithms. One of such algorithms XD-GRASP [FAC⁺16] was then implemented and accelerated with the GPUs to facilitate clinical use. Compared to state-of-the-art MRI reconstruction algorithms, our implementation is up to 17 times faster while maintaining 99% SSIM. This work has been published in papers [BWY⁺21, LBG⁺23]. These results allow us to solve research challenge C1, as the

time requirements of two minutes are met while producing high-quality 4D-MRI images.

Chapter 4 shows a methodology for forecasting MC performance performance when using accelerators. The performance model allows evaluating the potential speedup of the FPGA or the GPU accelerator. The forecasted speedup can be used to decide the best (if any) accelerator to use to achieve the required performance. The results show that it provides a reasonable range for the execution time in the GPU case. While, in the FPGA case, as the FPGA's performance is more predictable the predictions are within 2% of the measured execution time. This work has been published in paper [BHT⁺23] and is currently been accepted in the Journal of Physics: Conference Series. The accuracy of the methodology is enough to provide the necessary indication of the performance that MC would achieve when using specific accelerators. This solves research challenge C2, however, extensions to the modelling methodology would be necessary when using accelerators with different architectural characteristics from the ones considered. For example, future GPUs that combine AMD's 3D V-Cache technology [WAC⁺22] might change the decision process when forecasting the impact of cache misses.

Chapter 5 analyses the use of the accelerators for DPM and provides an implementation that meets real-time adaptive radiotherapy constraints. DPM is an MC simulation that offers a good tradeoff between speed and accuracy. However, it falls significantly short of meeting real-time adaptive radiotherapy requirements [ZPPKO15]. Hence, a performance model for DPM was built. The forecasts of this analysis show that DPM will not benefit from the GPU or the FPGA acceleration, hence it proposes mathematical, algorithmic and architectural optimisations resulting in ODPM. DPM to meets real-time adaptive radiotherapy requirements consisting of, 1s or less for a 6MeV photon beam [VZV⁺20]. These results address research challenge C3 as both real-time adaptive radiotherapy accuracy and time requirements are met. However, a mechanism that converts 4D-MRI images to a patient model [BNC⁺21] will be required to use ODPM with real data and using real beams.

6.2 Future Work

There are several areas where research can be conducted to enable real-time adaptive radiotherapy. Some possible directions for future work will be presented in the following sections. Some are short-term improvements while others are part of a long-term plan that we expect to be possible further ahead in the future.

6.2.1 Real-time MRI reconstruction

While the proposed implementation of XD-GRASP meets MRgoART time requirements, real-time adaptive radiotherapy where the time requirements are more stringent requires real-time motion tracking. As section 2.1.1 explained, the patient’s anatomy might vary during the treatment and this can cause the cancer to move. For example, in the case of lung cancer, the breathing motion might cause the cancer to move several centimetres.

In the short term XD-GRASP could be integrated into other existing frameworks to meet real-time adaptive radiotherapy time requirements. Algorithms such as MRSIGMA [FTO20] and SPIDERM [HCX⁺22] based on XD-GRASP can be used for real-time imaging. In this context, our implementation of XD-GRASP could be integrated into those frameworks to provide additional speedup or our architecture might be directly applied, enabling real-time imaging. AI-based reconstruction schemes are also a promising way to provide real-time imaging [FGCN⁺21, TMVvdB23]. In this context, our XD-GRASP implementation could be used to generate reference training datasets or could be integrated as a feature of Neural Networks (NNs).

In the long term, the approach used to accelerate XD-GRASP could influence the future development of 4D-MRI reconstruction algorithms. The possibility of leveraging accelerators while hiding and overlapping data transfers could lead to the development of accelerator-specific algorithms that can meet the real-time adaptive radiotherapy time and accuracy requirements.

6.2.2 Extension to performance modelling

The methodology proposed supports only MC simulations and FPGA, GPU accelerators. In the short term, one possible future direction would be extending the modelling methodology to forecast the performance of NNs on custom AI accelerators, such as TPU and Groq [Gwe]. This is of interest in the context of real-time adaptive radiotherapy as dose delivery reconstruction can be performed using NNs [EFK⁺22, PSDH⁺23]. This approach combined with the use of custom AI accelerators could potentially allow NNs to meet real-time adaptive radiotherapy time requirements. Hence, one possible future direction would be extending the modelling methodology to forecast the performance of NNs on these accelerators.

In the long term, another possible extension to performance modelling would be automating the approach. Initially, it would be possible to automatically derive the operation count and the volume of data transfers by developing tools that can instrument and monitor the execution of the MC simulation. Building on this initial tool, the automation could be further extended where given functional and performance requirements for MC simulation, the tool could automatically explore the design space and produce optimised CPU, GPU and FPGA implementation that meet the requirements.

6.2.3 Extension to ODPM

ODPM is an MC simulator that is capable of computing radiation doses in the context of radiotherapy and meets real-time adaptive radiotherapy time requirements. However, to use ODPM on real data additional features are required.

The shorter term ODPM has basic support for converting CT images into patient models where Hounsfield Units [BG19] are mapped to electron densities and then discretized over a set of NIST materials [CDS20]. However, ODPM is not evaluated on real CT images. The next step for ODPM is validating it on commonly used radiotherapy treatment plans using real CT images. This would require implementing a more rigorous conversion from Hounsfield Units to electron densities [Sud20]. Another possible, shorter-term future direction for ODPM is integrating it

in NN based approaches. NNs [EFK⁺22, PSDH⁺23] offer compelling performance but there are limitations on their accuracy. Given the performance of ODPM, it could be integrated in three possible ways. Firstly, it is possible to use ODPM to generate large datasets in real time. Evaluating all possible beam parameters might result in huge datasets that are impossible to store, on volumetric CT images can result in hundreds of resulting dose distributions depending on the parameters used. ODPM can generate these datasets on the fly, avoiding the need to store the entire dataset. Only beam parameters and original images need to be on permanent storage. Secondly, it is possible to use ODPM to generate a low-accuracy dose distribution and use NN to enhance it to the required accuracy. Finally, it is possible to use ODPM as a feature of the NN, where a lower accuracy distribution is used internally by the NN to perform a more accurate reconstruction.

In the long term, deploying selected approaches to convert MRI data to the patient model will be necessary to use ODPM in the context of the MR-linac. These approaches are based on NNs and convert MRI images to synthetic CTs [BNC⁺21]. Then these CTs could then be transformed into patient models. However, achieving this objective requires several intermediate steps first. For example, several of the MRI challenges listed in section 6.2.1 need to be addressed, as well as some of the ODPM challenges listed previously in this section.

6.3 Final thoughts

Predicting the future is always a challenging task, and it falls beyond the scope of this thesis. However, the research conducted herein could open up several potential avenues. This work has implications across various domains, ranging from MRI and radiotherapy to experimental physics.

The investigation into accelerating XD-GRASP introduces a novel approach to meeting MR-goART time requirements. While recent research in this field primarily concentrates on discovering new algorithms with faster convergence or lower computational complexity, the exploration of applying HPC to MRI represents a new approach explored in this thesis. This

exploration opens new research possibilities; future algorithms may emerge not solely based on mathematical properties or computational complexity, but also tailored to particular accelerators, as shown by the performance demonstrated by our XD-GRASP implementation. These new algorithms could potentially find applications in real-time adaptive radiotherapy, enhancing cancer treatment for future generations.

The proposed performance modelling could influence future research in both MC simulations and chip design. This modelling methodology enables the forecasting of performance given a specific accelerator and application. While in this thesis, the results are utilised to evaluate the performance of an application using a given accelerator with the goal of selecting an existing accelerator, it would be possible to go beyond existing accelerators and algorithms. The modelling result could be used to conceive accelerator-specific MC simulations or design specific accelerators for MC in the future. Particularly in real-time adaptive radiotherapy scenarios, where guaranteed latency is crucial, potential future research might explore the feasibility of designing custom chips for this specific purpose.

The MC simulation developed here targets real-time adaptive radiotherapy, offering the medical physics community opportunities to explore novel approaches to cancer treatment. While the primary aim is to optimise treatment parameters downstream, the simulator holds potential for various future applications. Firstly, it could facilitate the testing of different physics implementations; for instance, developing faster or more accurate methods to simulate specific physics processes that can be seamlessly integrated and verified for performance and accuracy. Furthermore, it could enable the evaluation of various treatment parameters and their outcomes. Additionally, the simulator could aid in designing newer radiotherapy beams by quickly assessing dose distribution across a range of test cases. Another potential application lies in utilising the simulator during the design phase of a calorimeter, although electromagnetic shower simulation inside a calorimeter is different from a radiotherapy treatment, it requires the same physics implemented in ODPM; for instance, future upgrades at the CERN Large Hadron Collider's calorimeter could be influenced by ODPM.

Bibliography

- [AAA⁺03] S Agostinelli, J Allison, K Amako, J Apostolakis, H Araujo, P Arce, M Asai, D Axen, S Banerjee, G Barrand, F Behner, L Bellagamba, J Boudreau, L Broglia, A Brunengo, H Burkhardt, S Chauvie, J Chuma, R Chytrcek, G Cooperman, G Cosmo, P Degtyarenko, A Dell’Acqua, G Depaola, D Dietrich, R Enami, A Feliciello, C Ferguson, H Fesefeldt, G Folger, F Foppiano, A Forti, S Garelli, S Giani, R Giannitrapani, D Gibin, J J Gómez Cadenas, I González, G Gracia Abril, G Greeniaus, W Greiner, V Grichine, A Grossheim, S Guatelli, P Gumplinger, R Hamatsu, K Hashimoto, H Hasui, A Heikkinen, A Howard, V Ivanchenko, A Johnson, F W Jones, J Kallenbach, N Kanaya, M Kawabata, Y Kawabata, M Kawaguti, S Kelner, P Kent, A Kimura, T Kodama, R Kokoulin, M Kossov, H Kurashige, E Lamanna, T Lampén, V Lara, V Lefebure, F Lei, M Liendl, W Lockman, F Longo, S Magni, M Maire, E Medernach, K Minamimoto, P Mora de Freitas, Y Morita, K Murakami, M Nagamatu, R Nartallo, P Nieminen, T Nishimura, K Ohtsubo, M Okamura, S O’Neale, Y Oohata, K Paech, J Perl, A Pfeiffer, M G Pia, F Ranjard, A Rybin, S Sadilov, E Di Salvo, G Santin, T Sasaki, N Savvas, Y Sawada, S Scherer, S Sei, V Sirotenko, D Smith, N Starkov, H Stoecker, J Sulkimo, M Takahata, S Tanaka, E Tcherniaev, E Safai Tehrani, M Tropeano, P Truscott, H Uno, L Urban, P Urban, M Verderi, A Walkden, W Wander, H Weber, J P Wellisch, T Wenaus, D C Williams, D Wright, T Yamada, H Yoshida, and D Zschesche. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003.

- [AAB⁺23] G Amadio, J Apostolakis, P Buncic, G Cosmo, D Dosaru, A Gheata, S Hageboeck, J Hahnfeld, M Hodgkinson, B Morgan, M Novak, A A Petre, W Pokorski, A Ribon, G A Stewart, and P M Vila. Offloading electromagnetic shower transport to GPUs. *Journal of Physics: Conference Series*, 2438(1):012055, 2 2023.
- [ABC⁺19] Yehia Arafa, Abdel Hameed A. Badawy, Gopinath Chennupati, Nandakishore Santhi, and Stephan Eidenbenz. PPT-GPU: Scalable GPU Performance Modeling. *IEEE Computer Architecture Letters*, 18(1):55–58, 1 2019.
- [ABD⁺06] W. Adam, T. Bergauer, C. Deldicque, J. Erö, R. Fruehwirth, M. Jeitler, K. Kastner, S. Kostner, N. Neumeister, M. Padrta, P. Porth, H. Rohringer, H. Sakulin, J. Strauss, A. Taurok, G. Walzel, C. E. Wulz, S. Lowette, B. van de Vyver, G. de Lentdecker, P. Vanlaer, C. Delaere, V. Lemaître, A. Ninane, O. van der Aa, J. Damgov, V. Karimäki, R. Kinnunen, T. Lampén, K. Lassila-Perini, S. Lehti, J. Nysten, J. Tuominiemi, P. Busson, T. Todorov, G. Schwering, P. Gras, G. Daskalakis, A. Sfyrla, M. Barone, T. Geralis, C. Markou, K. Zachariadou, P. Hidas, S. Banerjee, K. Mazumdar, S. Banerjee, M. Abbrescia, A. Colaleo, N. D’Amato, N. de Filippis, D. Giordano, F. Loddo, M. Maggi, L. Silvestris, G. Zito, S. Arcelli, D. Bonacorsi, P. Capiluppi, G. M. Dallavalle, A. Fanfani, C. Grandi, S. Marcellini, A. Montanari, F. Odorici, R. Travaglini, S. Costa, A. Tricomi, V. Ciulli, N. Magini, R. Ranieri, L. Berti, M. Biasotto, M. Gulmini, G. Maron, N. Toniolo, L. Zangrando, M. Bellato, U. Gasparini, S. Lacaprara, A. Parenti, M. Passaseo, P. Ronchese, S. Vanini, S. Ventura, P. L. Zotto, D. Benedetti, M. Biasini, L. Fanò, L. Servoli, G. Bagliesi, T. Boccali, S. Dutta, S. Gennai, A. Giassi, F. Palla, G. Segneri, A. Starodumov, R. Tenchini, P. Meridiani, G. Organtini, N. Amapane, F. Bertolino, R. Cirio, J. Y. Kim, I. T. Lim, M. Y. Pac, K. K. Joo, S. B. Kim, Y. I. Choi, I. T. Yu, K. Cho, J. Chung, S. W. Ham, D. H. Kim, G. N. Kim, W. Kim, J. C. Kim, S. K. Oh, H. Park, S. R. Ro, D. C. Son, J. S. Suh, Z. Aftab, H. Hoorani, A. Osman, K. Bunkowski, M. Cwiok, W. Dominik, K. Doroba, M. Kazana, J. Krolikowski, I. Kudla, M. Pietrusinski, K. Pozniak, W. Zabolotny, J. Zalipska, P. Zych, L. Goscilo, M. Górski,

G. Wrochna, P. Zalewski, R. Alemany-Fernandez, C. Almeida, N. Almeida, J. C. da Silva, M. Santos, I. Teixeira, J. P. Teixeira, J. Varela, N. Vaz Cardoso, I. Golutvin, V. Konoplyanikov, S. Shmatov, S. Shulga, A. Zarubin, S. Gninenko, N. Krasnikov, V. Matveev, A. Toropin, V. Gavrilov, V. Kolosov, A. Krokhotin, A. Oulianov, N. Stepanov, S. V. Rusakov, O. L. Kodolova, L. Sarycheva, I. Vardanyan, V. Kachanov, V. Krychkine, S. Slabospitski, N. Tyurin, J. Ilic, G. Skoro, C. Albajar, J. F. de Trocóniz, A. Calderon, M. A. Lopez Virto, R. Marco, C. Martinez Rivero, F. Matorras, I. Vila, S. Cucciarelli, M. Konecki, S. Ashby, D. Barney, P. Bartalini, R. Benetta, V. Brigljevic, G. Bruno, E. Cano, S. Cittolin, M. della Negra, A. de Roeck, P. Favre, A. Frey, W. Funk, D. Futyan, D. Gigi, F. Glege, J. Gutleber, M. Hansen, V. Innocente, C. Jacobs, W. Jank, M. Kozlovsky, H. Larsen, M. Lenzi, I. Magrans, M. Mannelli, F. Meijers, E. Meschi, L. Mirabito, S. J. Murray, A. Oh, L. Orsini, C. Palomares Espiga, L. Pollet, A. Racz, S. Reynaud, D. Samyn, P. Scharff-Hansen, C. Schwick, G. Sguazzoni, N. Sinanis, P. Sphicas, M. Spiropulu, A. Strandlie, B. G. Taylor, I. van Vulpen, J. P. Wellisch, M. Winkler, D. Kotlinski, K. Prokofiev, T. Speer, I. Dumanoglu, D. S. Bailey, J. J. Brooke, D. Cussans, G. P. Heath, D. Machin, S. J. Nash, D. M. Newbold, M. G. Probert, J. A. Coughlan, R. Halsall, W. J. Haynes, I. R. Tomalin, N. Marinelli, A. Nikitenko, S. Rutherford, C. Seez, O. Sharif, G. Antchev, E. Hazen, J. Rohlf, S. Wu, R. Breedon, P. T. Cox, P. Murray, M. Tripathi, R. Cousins, S. Erhan, J. Hauser, P. Kreuzer, M. Lindgren, J. Mumford, P. Schlein, Y. Shi, B. Tannenbaum, V. Valuev, M. von der Mey, I. Andreeva, R. Clare, S. Villa, S. Bhattacharya, J. G. Branson, I. Fisk, J. Letts, M. Mojaver, H. P. Paar, E. Trepagnier, V. Litvine, S. Shevchenko, S. Singh, R. Wilkinson, S. Aziz, M. Bowden, J. E. Elias, G. Graham, D. Green, M. Litmaath, S. Los, V. O'Dell, N. Ratnikova, I. Suzuki, H. Wenzel, D. Acosta, D. Bourilkov, A. Korytov, A. Madorsky, G. Mitselmakher, J. L. Rodriguez, B. Scurlock, H. Stoeck, S. Abdullin, D. Baden, S. C. Eno, T. Grassi, S. Kunori, S. Pavlon, K. Sumorok, S. Tether, L. M. Cremaldi, D. Sanders, D. Summers, I. Osborne, L. Taylor,

- L. Tuura, W. C. Fisher, J. Mans, D. Stickland, C. Tully, T. Wildish, S. Wynhoff, B. P. Padley, P. Chumney, S. Dasu, and W. H. Smith. The CMS high level trigger. *EUROPEAN PHYSICAL JOURNAL C*, 46(3):605–667, 6 2006.
- [ASKC86] L. Axel, R. M. Summers, H. Y. Kressel, and C. Charles. Respiratory effects in two-dimensional Fourier transform MR imaging. *Radiology*, 160(3):795–801, 1986.
- [BCM⁺14] Kai Tobias Block, Hersh Chandarana, Sarah Milla, Mary Bruno, Tom Mulholland, Girish Fatterpekar, Mari Hagiwara, Robert Grimm, Christian Geppert, Berthold Kiefer, and Daniel K. Sodickson. Towards Routine Clinical Use of Radial Stack-of-Stars 3D Gradient-Echo Sequences for Reducing Motion Sensitivity. *Journal of the Korean Society of Magnetic Resonance in Medicine*, 18(2):87–106, 6 2014.
- [BG19] Daniel Bell and Kyle Greenway. Hounsfield Unit. *Radiopaedia.org*, 10 2019.
- [BHT⁺23] M Barbone, A Howard, A Tapper, D Chen, M Novak, and W Luk. Demonstration of FPGA Acceleration of Monte Carlo Simulation. *Journal of Physics: Conference Series*, 2438(1):1–6, 2 2023.
- [BKO⁺21] Marco Barbone, Bas W. Kwaadgras, Uwe Oelfke, Wayne Luk, and Georgi Gaydadjiev. Efficient Table-Based Polynomial on FPGA. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 374–382. IEEE, 10 2021.
- [Bly21] Simon Blyth. Integration of JUNO simulation framework with Opticks: GPU accelerated optical propagation via NVIDIA® OptiX™. *EPJ Web of Conferences*, 251:03009, 2021.
- [BMaK19] Alexander H Barnett, Jeremy Magland, and Ludvig af Klinteberg. A Parallel Nonuniform Fast Fourier Transform Library Based on an “Exponential of Semi-circle” Kernel. *SIAM Journal on Scientific Computing*, 41(5):C479–C504, 2019.

- [BNC⁺21] M. Boulanger, Jean Claude Nunes, H. Chourak, A. Largent, S. Tahri, O. Acosta, R. De Crevoisier, C. Lafond, and A. Barateau. Deep learning methods to generate synthetic CT from MRI in radiotherapy: A literature review. *Physica Medica*, 89:265–281, 9 2021.
- [BNP05] Philip J. Beatty, Dwight G. Nishimura, and John M. Pauly. Rapid gridding reconstruction with a minimal oversampling ratio. *IEEE Transactions on Medical Imaging*, 24(6):799–808, 6 2005.
- [BR92] Bob Boothe and Abhiram Ranade. Improved multithreading techniques for hiding communication latency in multiprocessors. In *Proceedings of the 19th Annual International Symposium on Computer Architecture, ISCA '92*, pages 214–223, New York, NY, USA, 1992. Association for Computing Machinery.
- [BSM⁺19] Anders S. Bertelsen, Tine Schytte, Pia K. Møller, Faisal Mahmood, Hans L. Riis, Karina L. Gottlieb, Søren N. Agergaard, Lars Dysager, Olfred Hansen, Janne Gornitzka, Elisabeth Veldhuizen, Dean B. ODwyer, Rasmus L. Christiansen, Morten Nielsen, Henrik R. Jensen, Carsten Brink, and Uffe Bernchou. First clinical experiences with a high field 1.5 T MR linac. *Acta Oncologica*, 58(10):1352–1357, 10 2019.
- [BSS⁺07] Christoph Bert, Nami Saito, Alexander Schmidt, Naved Chaudhri, Dieter Schardt, and Eike Rietzel. Target motion tracking with a scanned particle beam. *Med Phys*, 34(12):4768–4771, 2007.
- [BWY⁺21] Marco Barbone, Andreas Wetscherek, Thomas Yung, Uwe Oelfke, Wayne Luk, and Georgi Gaydadjiev. Efficient Online 4D Magnetic Resonance Imaging. *Proceedings - Symposium on Computer Architecture and High Performance Computing*, 2021.
- [BZ14] Vance W. Berger and YanYan Zhou. Kolmogorov–Smirnov Test: Overview. *Wiley StatsRef: Statistics Reference Online*, 9 2014.

- [CBN⁺16] Emma Colvill, Jeremy Booth, Simeon Nill, Martin Fast, James Bedford, Uwe Oelfke, Mitsuhiro Nakamura, Per Poulsen, Esben Worm, Rune Hansen, Thomas Ravkilde, Jonas Scherman Rydhög, Tobias Pommer, Per Munck Af Rosenschold, Stephanie Lang, Matthias Guckenberger, Christian Groh, Christian Herrmann, Dirk Verellen, Kenneth Poels, Lei Wang, Michael Hadsell, Thilo Sothmann, Oliver Blanck, and Paul Keall. A dosimetric comparison of real-time adaptive and non-adaptive radiotherapy: A multi-institutional study encompassing robotic, gimbaled, multileaf collimator and couch tracking. *Radiotherapy and Oncology*, 119(1):159–165, 4 2016.
- [CCMRZ21] Stefano Carrazza, Juan Cruz-Martinez, Marco Rossi, and Marco Zaro. MadFlow: towards the automation of Monte Carlo simulation on GPU for particle physics processes. *EPJ Web of Conferences*, 251:1–03022, 2021.
- [CDS20] Steven J. Choquette, David L. Duewer, and Katherine E. Sharpless. NIST Reference Materials: Utility and Future. *Annual Review of Analytical Chemistry*, 13(Volume 13, 2020):453–474, 6 2020.
- [Chr86] P Christillin. Nuclear Compton scattering. *Journal of Physics G: Nuclear Physics*, 12(9):837, 1986.
- [COL⁺14] F. Callaly, D. O’Loughlin, D. Lyons, A. Coffey, and F. Morgan. Xilinx Vivado High Level Synthesis: Case studies. In *25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communities Technologies (ISSC 2014/CICT 2014)*, pages 352–356. Institution of Engineering and Technology, 2014.
- [CPL⁺95] Departament Computadors, Vincent Pillet, Jesús Labarta, Toni Cortes, and Sergi Girona. PARAVÉR: A tool to visualize and analyze parallel code. *WoTUG-18*, 44, 3 1995.

- [CRKB20] Jaemin Choi, David F. Richards, Laxmikant V. Kale, and Abhinav Bhatele. End-to-end performance modeling of distributed GPU applications. *Proceedings of the International Conference on Supercomputing*, pages 1–30, 6 2020.
- [CUD] CUDA C++ Programming Guide.
- [DdD05] J. Detrey and F. de Dinechin. Table-based polynomials for fast hardware function evaluation. In *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, pages 328–333, 2005.
- [DDDM03] Catherine Daramy, David Defour, Florent de Dinechin, and Jean-Michel Muller. CR-LIBM: a correctly rounded elementary function library. <https://doi.org/10.1117/12.505591>, 5205:458–464, 12 2003.
- [DM98] L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998.
- [DR93] A Dutt and V Rokhlin. Fast Fourier Transforms for Nonequispaced Data. *SIAM J Sci Comput*, 14(6):1368–1393, 11 1993.
- [DSBDT13] Bruno Da Silva, An Braeken, Erik H. D’Hollander, and Abdellah Touhafi. Performance modeling for FPGAs: Extending the roofline model with high-level synthesis tools. *International Journal of Reconfigurable Computing*, 2013:1–7, 2013.
- [EFK⁺22] Bayirta V Egorova, Olga A Fedorova, Stepan N Kalmykov, Oscar Pastor-Serrano, and Zoltán Perkó. Millisecond speed deep learning based proton dose calculation with Monte Carlo accuracy. *Physics in Medicine & Biology*, 67(10):105006, 5 2022.
- [FAC⁺16] Li Feng, Leon Axel, Hersh Chandarana, Kai Tobias Block, Daniel K. Sodickson, and Ricardo Otazo. XD-GRASP: Golden-angle radial MRI with reconstruction of extra motion-state dimensions using compressed sensing. *Magnetic Resonance in Medicine*, 75(2):775–788, 2 2016.

- [FGB⁺14] Li Feng, Robert Grimm, Kai Tobias Block, Hersh Chandarana, Sungheon Kim, Jian Xu, Leon Axel, Daniel K. Sodickson, and Ricardo Otazo. Golden-angle radial sparse parallel MRI: combination of compressed sensing, parallel imaging, and golden-angle radial sampling for fast and flexible dynamic volumetric MRI. *Magnetic Resonance in Medicine*, 72(3):707–717, 9 2014.
- [FGCN⁺21] Joshua N Freedman, Oliver J Gurney-Champion, Simeon Nill, Anna-Maria Shiarli, Hannah E Bainbridge, Henry C Mandeville, Dow-Mu Koh, Fiona McDonald, Marc Kachelrieß, Uwe Oelfke, and Andreas Wetscherek. Rapid 4D-MRI reconstruction using a deep radial convolutional neural network: Dracula. *Radiother Oncol*, 159:209–217, 2021.
- [FJ98] Matteo Frigo and Steven G. Johnson. FFTW: An adaptive software architecture for the FFT. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, volume 3, pages 1381–1384, 1998.
- [FLW92] Alan M Ferrenberg, D P Landau, and Y Joanna Wong. Monte Carlo simulations: Hidden errors from “good” random number generators. *Physical Review Letters*, 69(23):3382–3384, 12 1992.
- [FMR⁺09] B G Fallone, B Murray, S Rathee, T Stanescu, S Steciw, S Vidakovic, E Blosser, and D Tymofichuk. First MR images obtained during megavoltage photon irradiation from a prototype integrated linac-MR system. *Med Phys*, 36(6):2084–2088, 2009.
- [FTO20] Li Feng, Neelam Tyagi, and Ricardo Otazo. MRSIGMA: Magnetic Resonance SIGnature MAtching for real-time volumetric imaging. *Magnetic Resonance in Medicine*, 2 2020.
- [GBK⁺13] Robert Grimm, Simon Bauer, Berthold Kiefer, Joachim Hornegger, and Tobias Block. Optimal Channel Selection for Respiratory Self-Gating Signals (abstr). In *Proc Intl Soc Mag Reson Med 21*, page 3749, 2013.

- [GPL⁺22] Rosie J Goodburn, Marielle E P Philippens, Thierry L Lefebvre, Aly Khalifa, Tom Bruijnen, Joshua N Freedman, David E J Waddington, Eyesha Younus, Eric Aliotta, Gabriele Meliador, Teo Stanescu, Wajiha Bano, Ali Fatemi-Ardekani, Andreas Wetscherek, Uwe Oelfke, Nico van den Berg, Ralph P Mason, Petra J van Houdt, James M Balter, and Oliver J Gurney-Champion. The future of MRI in radiation therapy: Challenges and opportunities for the MR community. *Magn Reson Med*, 88(6):2592–2608, 2022.
- [GSB22] Pingakshya Goswami, Masoud Shahshahani, and Dinesh Bhatia. Robust Estimation of FPGA Resources and Performance from CNN Models. *Proceedings - 2022 35th International Conference on VLSI Design, VLSID 2022 - held concurrently with 2022 21st International Conference on Embedded Systems, ES 2022*, pages 144–149, 2022.
- [GSH⁺14] Filippo Del Grande, Francesco Santini, Daniel A. Herzka, Michael R. Aro, Cooper W. Dean, Garry E. Gold, and John A. Carrino. Fat-Suppression Techniques for 3-T MR Imaging of the Musculoskeletal System. *Radiographics : a review publication of the Radiological Society of North America, Inc*, 34(1):217, 1 2014.
- [Gwe] Linley Gwennap. GROQ ROCKS NEURAL NETWORKS Startup Creates New Architecture: One Core, 1,000 TOPS.
- [GWL⁺23] J. Gough, A. Wetscherek, B. Lecoeur, S. Alexander, R. Westley, B. Ng-Cheng-Hin, A. Hunt, and K. Aitken. PD-0658 Use of 4DMRI acquired on an MR-Linac to quantify intra-fraction motion in pancreatic cancer. *Radiotherapy and Oncology*, 182:S549–S550, 5 2023.
- [Hal36] Harvey Hall. The theory of photoelectric absorption for X-rays and γ -rays. *Reviews of Modern Physics*, 8(4):358–397, 10 1936.
- [HCX⁺22] Pei Han, Junzhou Chen, Jiayu Xiao, Fei Han, Zhehao Hu, Wensha Yang, Minsong Cao, Diane C Ling, Debiao Li, Anthony G Christodoulou, and Zhaoyang

- Fan. Single projection driven real-time multi-contrast (SPIDERM) MR imaging using pre-learned spatial subspace and linear transformation. *Phys Med Biol*, 67(13):135008, 6 2022.
- [Hel98] P Hellekalek. Don't Trust Parallel Monte Carlo! In *Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation*, PADS '98, pages 82–89, USA, 1998. IEEE Computer Society.
- [HHO⁺18] A. Hunt, V. N. Hansen, U. Oelfke, S. Nill, and S. Hafeez. Adaptive Radiotherapy Enabled by MRI Guidance. *Clinical Oncology*, 30(11):711–719, 11 2018.
- [HN04] Eberhard Haug and Werner Nakel. *The Elementary Process of Bremsstrahlung*, volume 73 of *World Scientific Lecture Notes in Physics*. WORLD SCIENTIFIC, 3 2004.
- [HOM⁺20] Aswin Hoffmann, Bradley Oborn, Maryam Moteabbed, Susu Yan, Thomas Bortfeld, Antje Knopf, Herman Fuchs, Dietmar Georg, Joao Seco, Maria Francesca Spadea, Oliver Jäkel, Christopher Kurz, and Katia Parodi. MR-guided proton therapy: a review and a preview. *Radiat Oncol*, 15(1):129, 2020.
- [Hub06] J H Hubbell. Electron–positron pair production by photons: A historical overview. *Radiation Physics and Chemistry*, 75(6):614–623, 2006.
- [HWS16] CK Hui, Z Wen, and B Stemkens. 4D MR imaging using robust internal respiratory signal. *RHN Tijssen - Physics in Medicine*, 2016.
- [IG16] Julianna D. Ianni and William A. Grissom. Trajectory Auto-Corrected image reconstruction. *Magnetic Resonance in Medicine*, 76(3):757–768, 9 2016.
- [Int] Introduction to FPGA Resources - NI.
- [JM20] Frederick James and Lorenzo Moneta. Review of High-Quality Random Number Generators. *Computing and Software for Big Science*, 4(1):1–12, 12 2020.
- [KBC14] Paul J Keall, Michael Barton, and Stuart Crozier. The Australian Magnetic Resonance Imaging-Linac Program. *Semin Radiat Oncol*, 24(3):203–206, 2014.

- [KBTB14] Dirk P Kroese, Tim Brereton, Thomas Taimre, and Zdravko I Botev. Why the Monte Carlo method is so important today. *WIREs Computational Statistics*, 6(6):386–392, 11 2014.
- [KEM01] Peter Kellman, Frederick H. Epstein, and Elliot R. McVeigh. Adaptive sensitivity encoding incorporating temporal filtering (TSENSE). *Magnetic Resonance in Medicine*, 45(5):846–852, 2001.
- [KNO⁺18] Paul J Keall, Doan Trang Nguyen, Ricky O’Brien, Pengpeng Zhang, Laura Happersett, Jenny Bertholet, and Per R Poulsen. Review of Real-Time 3-Dimensional Image Guided Radiation Therapy on Standard-Equipped Cancer Radiation Therapy Systems: Are We at the Tipping Point for the Era of Real-Time Radiation Therapy? *International Journal of Radiation Oncology*Biology*Physics*, 102(4):922–931, 2018.
- [Knu98] Donald E Knuth. *The art of computer programming. Vol. 2: Seminumerical algorithms*. Bonn: Addison-Wesley, 3rd ed. edition, 1998.
- [KPB19] Paul Keall, Per Poulsen, and Jeremy T Booth. See, Think, and Act: Real-Time Adaptive Radiotherapy. *Seminars in Radiation Oncology*, 29(3):228–235, 2019.
- [KSAR20] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. *Proceedings - International Symposium on Computer Architecture*, 2020-May:473–486, 5 2020.
- [LBG⁺23] Bastien Lecoer, Marco Barbone, Jessica Gough, Uwe Oelfke, Wayne Luk, Georgi Gaydadjiev, and Andreas Wetscherek. Accelerating 4D image reconstruction for magnetic resonance-guided radiotherapy. *Physics and Imaging in Radiation Oncology*, 27:100484, 7 2023.
- [LDP07] Michael Lustig, David Donoho, and John M. Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine*, 58(6):1182–1195, 12 2007.

- [LDW⁺21] Yongbao Li, Shouliang Ding, Bin Wang, Hongdong Liu, Xiaoyan Huang, and Ting Song. Extension and validation of a GPU-Monte Carlo dose engine gDPM for 1.5 T MR-LINAC online independent dose verification. *Medical Physics*, 48(10):6174–6183, 10 2021.
- [LHHN03] Jin Hyung Lee, Brian A Hargreaves, Bob S Hu, and Dwight G Nishimura. Fast 3D imaging using variable-density spiral trajectories with applications to limb perfusion. *Magn Reson Med*, 50(6):1276–1285, 2003.
- [LHL⁺22] Jounghoo Lee, Yeonan Ha, Suhyun Lee, Jinyoung Woo, Jinho Lee, Hanhwi Jang, and Youngsok Kim. GCoM: A Detailed GPU Core Model for Accurate Analytical Modeling of Modern GPUs. *Proceedings - International Symposium on Computer Architecture*, 22:424–436, 6 2022.
- [LP10] Michael Lustig and John M. Pauly. SPIRiT: Iterative self-consistent parallel imaging reconstruction from arbitrary k-space. *Magnetic Resonance in Medicine*, 64(2):457–471, 8 2010.
- [LRR⁺08] Jan J.W. Lagendijk, Bas W. Raaymakers, Alexander J.E. Raaijmakers, Johan Overweg, Kevin J. Brown, Ellen M. Kerkhof, Richard W. van der Put, Björn Hårdemark, Marco van Vulpen, and Uulke A. van der Heide. MRI/linac integration. *Radiotherapy and Oncology*, 86(1):25–29, 1 2008.
- [Lyo13] Louis Lyons. Discovering the Significance of 5 sigma. 10 2013.
- [Mar00] Peter. Markstein. *IA-64 and elementary functions : speed and precision*. Prentice Hall, 2000.
- [MHBB20] Kathryn E. Mittauer, Patrick M. Hill, Michael F. Bassetti, and John E. Bayouth. Validation of an MR-guided online adaptive radiotherapy (MRgoART) program: Deformation accuracy in a heterogeneous, deformable, anthropomorphic phantom. *Radiotherapy and Oncology*, 146:97–109, 5 2020.
- [Mit20] Sparsh Mittal. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications*, 32(4):1109–1139, 2 2020.

- [MLC⁺16] S Mutic, D Low, T Chmielewski, G Fought, M Hernandez, I Kawrakow, A Sharma, S Shvartsman, and J Dempsey. TU-H-BRA-08: The Design and Characteristics of a Novel Compact Linac-Based MRI Guided Radiation Therapy (MR-IGRT) System (abstr). *Med Phys*, 43(6):3770, 11 2016.
- [MMGP06] Donald W. McRobbie, Elizabeth A. Moore, Martin J. Graves, and Martin R. Prince. *MRI from picture to proton*. Cambridge University Press, 1 2006.
- [MMN⁺20] Martin J Menten, Jonathan K Mohajer, Rahul Nilawar, Jenny Bertholet, Alex Dunlop, Angela U Pathmanathan, Michel Moreau, Spencer Marshall, Andreas Wetscherek, Simeon Nill, Alison C Tree, and Uwe Oelfke. Automatic reconstruction of the delivered dose of the day using MR-linac treatment log files and online MR imaging. *Radiother and Oncol*, 145:88–94, 2020.
- [MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1 1998.
- [MP17] Nikolai J Mickevicius and Eric S Paulson. Investigation of undersampling and reconstruction algorithm dependence on respiratory correlated 4D-MRI for online MR-guided radiation therapy. *Physics in Medicine and Biology*, 62(8):2910–2921, 2017.
- [NR10] Anthony E. Nocentino and Philip J. Rhodes. Optimizing memory access on GPUs using morton order indexing. *Proceedings of the Annual Southeast Conference*, 2010.
- [NVI] NVIDIA/cuda-samples: Samples for CUDA Developers which demonstrates features in CUDA Toolkit.
- [ODM⁺17] Bradley M Oborn, Stephen Dowdell, Peter E Metcalfe, Stuart Crozier, Radhe Mohan, and Paul J Keall. Future of medical physics: Real-time MRI-guided proton therapy. *Med Phys*, 44(8):e77–e90, 2017.
- [One] oneAPI: A New Era of Heterogeneous Computing.

- [PAC⁺20] Eric S. Paulson, Ergun Ahunbay, Xinfeng Chen, Nikolai J. Mickevicius, Guang Pei Chen, Christopher Schultz, Beth Erickson, Michael Straza, William A. Hall, and X. Allen Li. 4D-MRI driven MR-guided online adaptive radiotherapy for abdominal stereotactic body radiation therapy on a high field MR-Linac: Implementation and initial clinical experience. *Clinical and translational radiation oncology*, 23:72–79, 7 2020.
- [PB86] Michael R. Paling and James R. Brookeman. Respiration artifacts in MR imaging: Reduction by breath holding. *Journal of Computer Assisted Tomography*, 10(6):1080–1082, 1986.
- [Per] Performance and Resource Utilization for Floating-point v7.1.
- [Pik21] Fedor Pikus. *The Art of Writing Efficient Programs: An advanced programmer’s guide to efficient hardware utilization and compiler optimizations using C++ examples*. Packt Publishing Ltd, 2021.
- [PMTL13] Oliver Pell, Oskar Mencer, Kuen Hung Tsoi, and Wayne Luk. Maximum Performance Computing with Dataflow Engines. In *High-Performance Computing Using FPGAs*, pages 747–774. Springer New York, New York, NY, 2013.
- [PSDH⁺23] Oscar Pastor-Serrano, Peng Dong, Charles Huang, Lei Xing, and Zoltán Perkó. Sub-second photon dose prediction via transformer neural networks. *Medical Physics*, 50(5):3159–3171, 5 2023.
- [PWBB01] Klaas P. Pruessmann, Markus Weiger, Peter Börnert, and Peter Boesiger. Advances in sensitivity encoding with arbitrary k-space trajectories. *Magnetic Resonance in Medicine*, 46(4):638–651, 2001.
- [PWO⁺22] Trang Thanh Pham, Brendan Whelan, Bradley M Oborn, Geoff P Delaney, Shalini Vinod, Caterina Brighi, Michael Barton, and Paul Keall. Magnetic resonance imaging (MRI) guided proton therapy: A review of the clinical challenges, potential benefits and pathway to implementation. *Radiother Oncol*, 170:37–47, 2022.

- [RGJ⁺20] Adam Rich, Michael Gregg, Ning Jin, Yingmin Liu, Lee Potter, Orlando Simonetti, and Rizwan Ahmad. CArtesian sampling with Variable density and Adjustable temporal resolution (CAVA). *Magnetic Resonance in Medicine*, 83(6):2015–2025, 6 2020.
- [RHB⁺17] Christopher M. Rank, Thorsten Heußer, Maria T.A. Buzan, Andreas Wetscherek, Martin T. Freitag, Julien Dinkel, and Marc Kachelrieß. 4D respiratory motion-compensated image reconstruction of free-breathing radial MR data with very high undersampling. *Magnetic resonance in medicine*, 77(3):1170–1183, 3 2017.
- [Rod85] David P Rodgers. Improvements in Multiprocessor System Design. pages 225–231, Washington, DC, USA, 1985. IEEE Computer Society Press.
- [Sag94] Hans Sagan. Hilbert’s Space-Filling Curve. pages 9–30, 1994.
- [SAL] Kyle Singer, Kunal Agrawal, and I-Ting Angelina Lee. Scheduling I/O Latency-Hiding Futures in Task-Parallel Platforms. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 147–161.
- [SAN⁺84] C. L. Schultz, R. J. Alfidi, A. D. Nelson, S. Y. Kopiwoda, and M. E. Clampitt. The effect of motion on two-dimensional Fourier transformation magnetic resonance images. *Radiology*, 152(1):117–121, 1984.
- [Sav15] Konstantin G. Savvidy. The MIXMAX random number generator. *Computer Physics Communications*, 196:161–165, 11 2015.
- [SBSW61] L. Sodickson, W. Bowman, J. Stephenson, and R. Weinstein. Single-quantum annihilation of positrons. *Physical Review*, 124(6):1851–1861, 12 1961.
- [Sch99] Nicol N. Schraudolph. A Fast, Compact Approximation of the Exponential Function. *Neural Computation*, 11(4):853–862, 5 1999.
- [Sin09] Matthew J. Singleton. Functional Magnetic Resonance Imaging. *The Yale Journal of Biology and Medicine*, 82(4):233, 2009.

- [SKA13] Kavya Shagrithaya, Krzysztof Kepa, and Peter Athanas. Enabling development of OpenCL applications on FPGA platforms. In *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, pages 26–30. IEEE, 6 2013.
- [SKF⁺18] Shobhit Sharma, Anuj J. Kapadia, Wanyi Fu, William P. Segars, Ehsan Samei, and Ehsan Abadi. A rapid GPU-based Monte Carlo simulation tool for individualized dose estimations in CT. <https://doi.org/10.1117/12.2294965>, 10573:982–990, 3 2018.
- [Ski04] John Skilling. Programming the Hilbert curve. *AIPC*, 707:381–387, 6 2004.
- [SLG⁺09] Jiayu Song, Yanhui Liu, Sally L. Gewalt, Gary Cofer, G. Allan Johnson, and Qing Huo Liu. Least-square NUFFT methods applied to 2-D and 3-D radially encoded MR image reconstruction. *IEEE Transactions on Biomedical Engineering*, 56(4):1134–1142, 4 2009.
- [SM13] Mutsuo Saito and Makoto Matsumoto. Variants of Mersenne Twister Suitable for Graphic Processors. *ACM Transactions on Mathematical Software (TOMS)*, 39(2):1–12, 2 2013.
- [SMDS11] John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw. Parallel random numbers: As easy as 1, 2, 3. *Proceedings of 2011 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2011.
- [SP15] Maria A Schmidt and Geoffrey S Payne. Radiotherapy planning using MRI. *Physics in Medicine and Biology*, 60(22):R323–R361, 2015.
- [SPT18] B Stemkens, E S Paulson, and R H N Tijssen. Nuts and bolts of 4D-MRI for radiotherapy. *Phys Med Biol*, 63(21):21TR01, 11 2018.
- [SS16] Konstantin Savvidy and George Savvidy. Spectrum and entropy of C-systems MIXMAX random number generator. *Chaos, Solitons & Fractals*, 91:33–38, 10 2016.

- [STAS91] G. K. Savvidy and N. G. Ter-Arutyunyan-Savvidy. On the Monte Carlo simulation of physical systems. *Journal of Computational Physics*, 97(2):566–572, 12 1991.
- [STDdS⁺15] Bjorn Stemkens, Rob Tijssen, Baudouin Denis de Senneville, Hanne D Heerkens, Marco van Vulpen, Jan Lagendijk, C van den Berg, Rob H Tijssen, Baudouin D de Senneville, Jan JW Lagendijk, and Cornelis AT van den Berg. Optimizing 4-dimensional magnetic resonance imaging data sampling for respiratory motion analysis of pancreatic tumors. *International Journal of Radiation Oncology-Biology-Physics*, 91(3), 2015.
- [STdS⁺16] Bjorn Stemkens, Rob H N Tijssen, Baudouin Denis de Senneville, Jan J W Lagendijk, and Cornelis A T van den Berg. Image-driven, model-based 3D abdominal motion estimation for MR-guided radiotherapy. *Physics in Medicine and Biology*, 61(14):5335–5355, 2016.
- [Sud20] Atchar Sudhyadhom. On the molecular relationship between Hounsfield Unit (HU), mass density, and electron density in computed tomography (CT). *PLOS ONE*, 15(12):e0244861, 12 2020.
- [Sum18] Sioni Paris Summers. Application of FPGAs to Triggering in High Energy Physics, 2018.
- [SV13] Joao Seco and Frank Verhaegen. *Monte Carlo Techniques in Radiation Therapy, Imaging in medical diagnosis and therapy Series*. CRC Press Taylor & Francis Group, 2013.
- [SWA⁺21] Yu-hsuan Shih, Garrett Wright, Joakim Andén, Johannes Blaschke, and Alex H. Barnett. cuFINUFFT: a load-balanced GPU library for general-purpose nonuniform FFTs. *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 1:688–697, 2 2021.
- [SWB00] Josep Sempau, Scott J. Wilderman, and Alex F. Bielajew. DPM, a fast, accurate Monte Carlo code optimized for photon and electron radiotherapy treatment

- planning dose calculations. *Physics in Medicine and Biology*, 45(8):2263–2291, 2000.
- [TBP03] Jeffrey Tsao, Peter Boesiger, and Klaas P. Pruessmann. k-t BLAST and k-t SENSE: Dynamic MRI With High Frame Rate Exploiting Spatiotemporal Correlations. *Magnetic Resonance in Medicine*, 50(5):1031–1042, 11 2003.
- [TMVvdB23] Maarten L Terpstra, Matteo Maspero, Joost J C Verhoeff, and Cornelis A T van den Berg. Accelerated respiratory-resolved 4D-MRI with separable spatiotemporal neural networks. *Med Phys*, Epub:2023 Aug. 1, 2023.
- [TN98] Matsumoto M T and Nishimura. Dynamic creation of pseudorandom number generators. *Monte Carlo and Quasi-Monte Carlo Methods*, 2000:56, 1998.
- [VBM⁺17] Nils Voss, Marco Bacis, Oskar Mencer, Georgi Gaydadjiev, and Wayne Luk. Convolutional neural networks on dataflow engines. In *Proceedings - 35th IEEE International Conference on Computer Design, ICCD 2017*, pages 435–438. Institute of Electrical and Electronics Engineers Inc., 11 2017.
- [VBMG17] Nils Voss, Tobias Becker, Oskar Mencer, and Georgi Gaydadjiev. Rapid development of Gzip with MaxJ. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10216 LNCS:60–71, 2017.
- [VKM⁺21] Nils Voss, Bastiaan Kwaadgras, Oskar Mencer, Wayne Luk, and Georgi Gaydadjiev. On Predictable Reconfigurable System Design. *ACM Transactions on Architecture and Code Optimization (TACO)*, 18(2):1–17, 2 2021.
- [Vol16] Vasily Volkov. Understanding Latency Hiding on GPUs. 2016.
- [VZV⁺19] Nils Voss, Peter Ziegenhein, Lukas Vermond, Joost Hoozemans, Oskar Mencer, Uwe Oelfke, Wayne Luk, and Georgi Gaydadjiev. Towards real time radiotherapy simulation. In *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, volume 2019-July, pages 173–180. Institute of Electrical and Electronics Engineers Inc., 7 2019.

- [VZV⁺20] Nils Voss, Peter Ziegenhein, Lukas Vermond, Joost Hoozemans, Oskar Mencer, Uwe Oelfke, Wayne Luk, and Georgi Gaydadjiev. Towards Real Time Radiotherapy Simulation. *Journal of Signal Processing Systems*, 92(9):949–963, 9 2020.
- [WAC⁺22] John Wu, Rahul Agarwal, Michael Ciraula, Carl Dietz, Brett Johnson, Dave Johnson, Russell Schreiber, Raja Swaminathan, Will Walker, and Samuel Nafziger. 3D V-Cache: the Implementation of a Hybrid-Bonded 64MB Stacked Cache for a 7nm x86-64 CPU. pages 428–429, 3 2022.
- [WBK⁺19] Dennis Winkel, Gijsbert H. Bol, Petra S. Kroon, Bram van Asselen, Sara S. Hackett, Anita M. Werensteijn-Honingh, Martijn P.W. Intven, Wietse S.C. Eppinga, Rob H.N. Tijssen, Linda G.W. Kerkmeijer, Hans C.J. de Boer, Stella Mook, Gert J. Meijer, Jochem Hes, Mirjam Willemsen-Bosman, Eline N. de Groot-van Breugel, Ina M. Jürgenliemk-Schulz, and Bas W. Raaymakers. Adaptive radiotherapy: The Elekta Unity MR-linac concept. *Clinical and Translational Radiation Oncology*, 18:54, 9 2019.
- [WBSS04] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 4 2004.
- [Wen26] G. Wentzel. Zwei Bemerkungen über die Zerstreuung korpuskularer Strahlen als Beugungserscheinung. *Zeitschrift für Physik*, 40(8):590–593, 8 1926.
- [WGM00] David O Walsh, Arthur F Gmitro, and Michael W Marcellin. Adaptive reconstruction of phased array MR imagery. *Magn Reson Med*, 43(5):682–690, 2000.
- [WHG⁺14] Katherine L. Wright, Jesse I. Hamilton, Mark A. Griswold, Vikas Gulani, and Nicole Seiberlich. Non-Cartesian parallel imaging reconstruction. *Journal of Magnetic Resonance Imaging*, 40(5):1022–1040, 11 2014.
- [WSK⁺07] Stefanie Winkelmann, Tobias Schaeffter, Thomas Koehler, Holger Eggers, and Olaf Doessel. An optimal radial profile order based on the Golden Ratio for time-resolved MRI. *IEEE transactions on medical imaging*, 26(1):68–76, 1 2007.

- [XLS] XLS: Accelerated HW Synthesis.
- [ZPPKO15] Peter Ziegenhein, Sven Pirner, Cornelis Ph Kamerling, and Uwe Oelfke. Fast CPU-based Monte Carlo simulation for radiotherapy dose calculation. *Physics in Medicine and Biology*, 60(15):6097–6111, 8 2015.