

GPU acceleration of Monte Carlo simulations: particle physics methods applied to medicine

Marco Barbone¹, Rafaël Brandt³, Alexander Howard¹,
Mihaly Novak², Wayne Luk¹, Georgi Gaydadjiev^{1,3,4}, Alex Tapper¹

¹ Imperial College London

² European Laboratory for Particle Physics (CERN)

³ University of Groningen

⁴ Delft University of Technology

E-mail: m.barbone19@imperial.ac.uk, a.tapper@imperial.ac.uk

Abstract. GPU acceleration has been successfully utilised in particle physics for real time analysis and simulation. This study investigates the potential benefits of GPU acceleration for medical physics applications by analysing performance, development effort, and availability. The application covers a stand-alone Monte Carlo simulation of single Coulomb scattering of electrons, with GPU acceleration developed by a software developer with no high performance computing experience. Such simulations contribute to real-time dose estimation for real-time adaptive radiotherapy, a new and emerging cancer treatment that heavily relies on high performance computing. As a proof of principle, we implement a single scattering process for electrons in a homogeneous material with a pencil beam at constant initial energy. We compare the performance gain offered by GPU acceleration against an optimised CPU implementation and evaluate it by computing 100M histories of a 128 keV electron interacting in water. The results show that when comparing the GPU with a multi-core CPU implementation running with 64 cores, a speedup of 10x was achieved which corresponds to a 5.5x cost-equivalent speedup. The results on both architectures were statistically equivalent. The successful implementation and measured acceleration combined with the low level of expertise needed for obtaining such speedup is a promising first step for the use of GPU acceleration in a context such as real-time adaptive radiotherapy where there are strict performance and time requirements.

1. Introduction

Monte Carlo (MC) simulations are widely used in the context of HEP and account for a large fraction of the total computational resources needed due to the required detail and precision of the experimental environments. Although MC has high computational complexity, it is embarrassingly parallel, and can thus greatly benefit from parallel and heterogeneous computing. In fact, *Geant4*, the most widely used MC software simulation toolkit in HEP, supports parallelism from version 10 [1]. However, the general trend of recent years shows that the complexity of MC simulation is growing faster than the computational power offered by new CPUs. Thus, recent studies have analysed the use of GPUs and FPGAs to accelerate MC simulations [2–4] and demonstrate compelling performance gains. *MadFlow* [3] accelerated Leading Order calculations for hadronic processes at $\sqrt{s} = 13$ TeV on GPU. Their results show that GPUs can achieve over 7x speedup compared to CPUs. Voss et al. [4] implemented a simplified electromagnetic shower simulation on an FPGA and demonstrated a speedup of over

4x compared to CPU. However, their results should be analysed with care and in the right context: they use an incomplete physics model and they do not mention a proper validation in their study. Barbone et al. [2] accelerated Coulomb scattering with FPGAs showing that MC can greatly benefit from it. A speedup of 270x and a cost equivalent speedup of over 10x, based on market prices at the time of writing, were reported.

While parallel and heterogeneous computing are widely studied in HEP, the medical community only recently started to analyse the potential of these types of computations. Parallel computing has been used for many years, for example, Ziegenhein et al. [5] proposed a parallel MC implementation for radiotherapy dose reconstruction. More recently, Sharma et al. [6] proposed a GPU-based MC simulation for individualised dose estimations. Applying parallel computing and heterogeneous computing is a challenging task that requires significant engineering effort. Parallelising and accelerating MC is even more difficult due to the stochastic and irregular nature of MC simulations. Successful parallelisation or acceleration often requires significant high-performance computing (HPC) expertise. Amadio et al. [7] show that successful GPU acceleration of electromagnetic showers relies on proper memory management and without low-level optimisations the performance achieved is far from the theoretical maximum.

HPC expertise is becoming increasingly wide-spread in HEP and most researchers have some training and experience with parallel and heterogeneous computing. On the other hand, these types of computations are only recently making their way into the medical community, hence, medical researchers often lack HPC experience. In this study, we use a simple, but representative, physics model that has all the characteristics of more involved particle physics processes such as electromagnetic shower (EM) simulation or multiple scattering used in radiotherapy. This MC simulation is used to analyse the challenges and potential pitfalls of both parallel and heterogeneous computing. We also compare the performance achieved by both researchers with limited computing experience against HPC experts in the context of MC.

2. Single Coulomb Scatter Algorithm

As a simple but physically meaningful MC test case, single Coulomb scattering has been chosen to be implemented on a GPU. 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 [8]

$$V(r) = \frac{zZe^2}{r} \exp(-r/R), \quad (1)$$

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

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

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

$$\sigma^{(SR)} = \left(\frac{zZe^2}{p\beta c} \right)^2 \frac{\pi}{A(1+A)}, \quad (3)$$

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

$$f_1(\theta)^{(SR)} = \frac{1}{\pi} \frac{A(1+A)}{(1 - \cos\theta + 2A)^2}. \quad (4)$$

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

$$\mu = 1 - \frac{2A\xi}{1 - \xi + A} \quad (5)$$

with $\xi \in \mathcal{U}(0, 1)$. The corresponding single elastic scattering model has been implemented in C++ then accelerated using a GPU.

3. Monte Carlo and GPU Acceleration

GPU acceleration often requires significant engineering effort. Moreover, not all workloads benefit from GPU acceleration. Hence, in this study we deploy a methodology that allows one to analyse and determine “a-priori”, if the workload can benefit from GPU acceleration. The developer is able to determine that the application is not suited for GPU acceleration before writing any line of code, thus, greatly reducing needless development time. While a comprehensive description of the methodology is outside the scope of this study, a brief summary and its application to the MC simulation follows.

3.1. Performance model

Algorithm 1 Methodology for GPU acceleration

```

1: available parallelism analysis
2: if not enough parallelism then
3:   early termination; workload not suited for GPU acceleration
4: end if
5: problem cache and memory requirement analysis
6: if working-set does not fit in cache then
7:   early termination; not enough cache for GPU acceleration
8: end if
9: do
10:  parallel solution proposal
11:  GPU performance and data transfer modelling of the solution
12: while solution does not meet performance and engineering effort requirements
13: return solution

```

Algorithm 1 summarises the methodology used in this study. First, the potential parallelism of the algorithm is analysed (line 1). GPUs require particularly high parallelism to offer high performance since present era GPUs can execute over 10,000 threads. Less parallelism with GPU acceleration is unlikely to offer significant speedup over modern CPUs (line 2). The second step consists of analysing the cache and memory requirements of the algorithm (line 3). In contrast to CPUs, GPUs do not have prefetchers, hence, accessing data not present in the cache greatly reduces performance. Moreover, GPUs have smaller cache size compared to CPUs so the algorithm working set might fit inside a CPU cache but the GPU cache might not be able to accommodate it. In these cases, the overhead introduced by frequent data transfers from global GDDR memory to cache limits the performance achievable by GPUs. Thus, it is necessary to profile and analyse the algorithm to account for the memory and cache requirements so that either the algorithm is refactored to meet GPU computing requirements (line 5) or no programming effort is invested as GPU performance is likely worse than CPU (line 6).

If the algorithm meets the minimum requirements for GPU acceleration a design exploration phase is needed (line 10-12). This phase consists on drafting different solutions and then forecasting the expected performance of GPU acceleration. Usual parameters considered in

this phase are branches and data transfers. GPUs do not have branch predictors and the SIMT programming model requires all threads to execute the same instruction at the same time. When branch divergence occurs the diverging threads are de-scheduled and re-scheduled at a later time. This way of handling branches limits performance as some of the compute units of the GPU will be executing no-ops as threads they were supposed to execute are suspended.

Another factor that limits GPU performance is data transfer. Data can be transferred from CPU main memory (DDR) through the PCIe bus to the GPU (and vice versa). The PCIe bus has limited bandwidth, 15.75 GB/s in case of PCIe 3.0x16 lanes or 31.5 GB/s for PCIe 4.0x16 lanes. Transferring data over the PCIe bus might require more time than executing the computation entirely on the CPU, hence, a thorough analysis of where data is stored and what data needs to be transferred is necessary to achieve high performance using GPU acceleration.

The parameters used to estimate the GPU performance of a parallel solution are:

- **Input size:** the number of items that the GPU needs to process;
- **Clock frequency:** GPU clock frequency, used to estimate the processing time;
- **CUDA cores:** estimates the parallelism achievable on the GPU, each CUDA core processes one element;
- **Efficiency:** estimates the GPU utilisation. The parameter is set to 1 and halved each time a branch is encountered, assuming the worst-case scenario analysis that half of the threads are de-scheduled;
- **ASM instructions:** number of x86 assembly instructions needed to process the input data obtained from GCC. NVIDIA GPU PTX assembly differs from x86 hence a range from best-case to worst-case is considered;
- **GDDR data transfers:** the total amount of data transferred to and from GDDR, divided by the bandwidth. It can be refined with the data access pattern, in case of irregular accesses latency should be considered;
- **PCIe data transfers:** the total amount of data transferred to and from the CPU, divided by the bandwidth of the bus.

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

$$Compute\ Time = \frac{Input\ Size * efficiency * ASM\ instructions}{Clock\ Frequency * CUDA\ cores} \quad (6)$$

To estimate the total time then, the data transfers should be added:

$$Total\ Time = Compute\ Time + \frac{GDDR\ data}{GDDR\ bandwidth} + \frac{PCIe\ data}{PCIe\ bandwidth} \quad (7)$$

Equation 7 does not account for irregular memory accesses, DMA efficiency, data transfers that happen in parallel to the computation. These details require an in depth discussion that is outside the scope of this paper since the chosen MC simulation is not memory bound.

Particles	Clock (GHz)	CUDA cores	Efficiency (0,1]	ASM instructions	Time (ms)
10 ⁸	1.455	5,120	0.0625	2,500	537
10 ⁸	1.455	5,120	0.0625	5,000	1,074
10 ⁸	1.455	5,120	0.0625	10,000	2,148

Table 1. Design space exploration.

Table 1 shows the design space exploration and the parameters used. Efficiency is computed by profiling the original CPU implementation and counting the average number of branches

a particle encounters during the simulation. The number of ASM instructions is derived by compiling the CPU code with no vectorization, no unrolling and with forced in-lining. Since the simulation consists of a main loop that evaluates the particle until it runs out of energy, this approach allows the estimation of the number of instructions in the main loop. This number is then multiplied by the average number of iterations. These values are then substituted in Equation 7 while zeroing data transfers as they are negligible in this case. The computation happens mostly in 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 [.5s, 2s]. On a 64-core machine, the MC computation time is forecasted to be ~ 11 ms, by dividing sequential time with the number of cores. We opt for a GPU implementation as it offers one order of magnitude improvement over CPUs.

3.2. Testing methodology

Since parallel and heterogeneous computing is only recently making its way into medical applications, most researchers in the field do not have HPC experience. Hence, to evaluate potential challenges and pitfalls that might be encountered and to quantify the endeavour of adapting existent simulations to these programming environment both junior and senior software engineers were tasked to parallelise first and accelerate the MC simulation. In particular:

- 21 AI and computing master students were given two weeks;
- A senior software engineer was given four hours to optimise the best solution.

The computing master students were chosen as a representation of a researcher that has coding knowledge but not explicit HPC experience. The results are obtained using the best solution proposed by the students.

4. Results and Evaluation

The final results of this MC simulation consist of a longitudinal and a transverse distribution. To evaluate the accuracy of the results we executed a Kolmogorov–Smirnov test against reference distributions which concluded that the distributions are equivalent using $\alpha = 10^{-5}$, achieving equivalence with smaller α requires a further increase in the number of histories. To evaluate the performance we computed 100M histories of a 128 KeV electron interacting in water. We compared the performance of 2x AMD EPYC 7551 32-Core Processor running at 2.0 GHz, boosting up to 3.0 GHz, and a 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.

	$\frac{T(\text{sequential})}{T(\text{parallel})}$	$\frac{T(\text{sequential})}{T(\text{GPU})}$
Expected	64	670
Junior	0.08	824
Senior	84	843

Table 2. Expected and achieved speedup

	$\frac{T(\text{Senior})}{T(\text{Junior})}$	$\frac{T(\text{Forecast})}{T(\text{Junior})}$	$\frac{T(\text{Forecast})}{T(\text{Senior})}$
Parallel	1,091	800	0.76
GPU	1.02	0.81	0.79

Table 3. Comparison between junior, senior and expected speedup.

Table 2 shows the expected and measured speedup for both parallel CPU and GPU accelerations. The junior programmers produced a two orders of magnitude slower implementation than the sequential one. The problem was not evident on non dual-socket servers where core-to-core communications are less expensive. The implementation achieves 2.5x speedup on a machine equipped with an AMD Ryzen 5900x 12-cores 3.7GHz (4.8GHz) which has more cores than any of the machines used by the juniors. The cause of the slowdown has

been identified as false sharing. The junior developers allocated thread-local data sequentially in memory outside the parallel region. The senior developer needed only to move the allocation inside the parallel region to improve performance by three orders of magnitude achieving a 84x speedup, outperforming the expected factor of 64.

There is only a 2% performance difference between the GPU solution proposed by the junior programmers and the one proposed by the senior as shown in table 3. Compared to the model the performance is between 20% and 30% better than expected. Since the model considers a worst case scenario, the measurements are always expected to outperform the model. The results also show that GPUs are 10 times faster compared to the optimised parallel implementation. This results in a 5.5x higher cost normalised speedup; that is to say that in case of suitable workload GPUs can be over five times more cost-effective than CPUs.

5. Conclusion and Future Work

This study shows that GPUs have the potential to be multiple orders of magnitude faster and more cost-effective than CPUs in the context of MC simulations. The results show that junior developers can produce competitive heterogeneous implementation in a reasonable amount of time. However, producing a fast parallel implementation requires more experience. In conclusion, GPUs could be an effective solution to accelerate medical MC simulations.

In the future, we would like to extend this MC simulation to perform a complete EM for real-time adaptive radiotherapy use. This simulation requires adding similar processes like multiple scattering, Moller scattering and Bremsstrahlung, which are similar in nature to Coulomb scattering and do not require changing the structure of the codebase to be implemented. Hence, we expect that the speedup observed in this study can be transferred to other and more complex MC simulations that are not memory-bound.

Acknowledgements

This research project was supported by the CRUK Convergence Science Centre at The Institute of Cancer Research, London, and Imperial College London (A26234). We acknowledge funding from the Cancer Research UK programme grant C33589/A19727. The Institute of Cancer Research and The Royal Marsden NHS Foundation Trust are members of the Elekta MR-Linac Research Consortium.

References

- [1] Liu R, Higley K A and Cosmo G 2014 *Journal of Physics: Conference Series* **513** 022005 ISSN 1742-6596
- [2] Barbone M, Howard A, Tapper A, Chen D, Novak M and Luk W 2023 *Journal of Physics: Conference Series* **2438** 012023 ISSN 1742-6596
- [3] Carrazza S, Cruz-Martinez J, Rossi M and Zaro M 2021 *EPJ Web of Conferences* **251** 03022 ISSN 2100-014X
- [4] Voss N, Ziegenhein P, Vermond L, Hoozemans J, Mencer O, Oelfke U, Luk W and Gaydadjev G 2019 Towards real time radiotherapy simulation *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors* vol 2019-July (Institute of Electrical and Electronics Engineers Inc.) pp 173–180 ISBN 9781728116013 ISSN 10636862
- [5] Ziegenhein P, Pirner S, Ph Kamerling C and Oelfke U 2015 *Physics in Medicine and Biology* **60** 6097–6111 ISSN 13616560
- [6] Sharma S, Kapadia A J, Fu W, Segars W P, Samei E and Abadi E 2018 <https://doi.org/10.1117/12.2294965> **10573** 982–990 ISSN 16057422
- [7] Amadio G, Apostolakis J, Buncic P, Cosmo G, Dosaru D, Gheata A, Hageboeck S, Hahnfeld J, Hodgkinson M, Morgan B, Novak M, Petre A A, Pokorski W, Ribon A, Stewart G A and Vila P M 2023 *Journal of Physics: Conference Series* **2438** 012055 ISSN 1742-6596
- [8] Wentzel G 1926 *Zeitschrift für Physik* **40** 590–593 URL <https://doi.org/10.1007/BF01390457>