Demonstration of FPGA Acceleration of Monte Carlo Simulation

M. Barbone[1], **A. Howard**[1], **A. Tapper**[1], **D. Chen**[1], **M. Novak**[2], **W. Luk**[1]

[1] Imperial College London

[2] European Laboratory for Particle Physics (CERN)

E-mail: `m.barbone19@imperial.ac.uk, w.luk@imperial.ac.uk`

Abstract. We present results from a stand-alone simulation of electron single Coulomb scattering as implemented completely on an Field Programmable Gate Array (FPGA) architecture and compared with an identical simulation on a standard CPU. FPGA architectures offer unprecedented speed-up capability for Monte Carlo simulations, however with the caveats of lengthy development cycles and resource limitation, particularly in terms of on-chip memory and DSP blocks. As a proof of principle of acceleration on an FPGA, we chose a single scattering process of electrons in water at an energy of 6 MeV. The initial code-base was implemented in C++ and optimised for CPU processing. To measure the potential performance gains of FPGAs compared to modern multi-core CPUs we computed 100M histories of a 6 MeV electron interacting in water. Without performing any hardware-specific optimisation, the results show that the FPGA implementation is over 110 times faster than an optimised parallel implementation running on 12 CPU-cores, and over 270 times faster than a sequential single-core CPU implementation. The results on both architectures were statistically equivalent. The successful implementation and acceleration results are very encouraging for the future exploitation of more sophisticated Monte Carlo simulation on FPGAs for High Energy Physics applications.

1. Introduction

Field Programmable Gate Arrays (FPGAs) are becoming increasingly popular, and thus of greatly increasing relevance in the context of High-Performance Computing (HPC). Recent advances in High-Level Synthesis (HLS) toolchains for customised hardware implementations greatly reduce the engineering effort needed to program FPGAs. As a consequence, the use of FPGA accelerators is quickly spreading from the HPC context to other disciplines that require processing large volumes of data or ultra-low latency response. In the context of High Energy Physics (HEP), the use of FPGA for track reconstruction at the CMS experiment has been investigated [1].

Monte Carlo (MC) simulations are widely utilised in the context of HEP, combined with their computational complexity they account for a large fraction of the total computational resources needed in HEP. In recent years, the general trend shows that the complexity of MC simulations is growing faster than the compute capabilities offered by newer CPUs, resulting in an increase of the computational time required by the simulations, especially for particle physics processes [2].

In the last years, we observed a proliferation of frameworks and codebases exploiting hardware accelerators to improve the performance of both scientific and industrial problems. In the context of HEP, there are many instances of both new systematic implementation or conversion of existing codebases and algorithms to accelerators. *MadFlow* is a framework that automates MC simulation on GPU for particle physics processes. According to their evaluation, they claim that *MadFlow* shows a great performance improvement while running on GPU [3]. On the other hand, Voss et al. [4] shows that the use of FPGA acceleration by achieving an 8x speedup compared to GPU implementations is able to meet real-time radiotherapy requirements. However, their results should be analysed with care and in the right context: they use a simplified model that does not implement all the physics and they do not mention a proper validation in their study.

In this study, we analyse the use of FPGAs to accelerate MC simulation. We use a simple, but representative, physics model that has all the characteristics of a more involved particle physics process such as electromagnetic shower (EM) simulation or multiple scattering. We also analyse FPGA ease of use, accounting for both expertise and engineering effort. For the sake of generality, we built a worst-case analysis: most of the engineering was carried out by junior software engineer with no previous FPGA programming experience, in addition, we did not rely on any hardware or toolchain specific optimisation, hence any software engineer using any toolchain should be able to reproduce the results presented here.

The remainder of this paper is structured as follows: A short description of the selected example algorithm is followed by a general discussion of the methodology followed for FPGA implementation. These are followed by sections presenting the results obtained and drawing conclusions.

2. Single Coulomb scatter algorithm

As a simple but physically meaningful MC test case, single Coulomb scattering has been chosen to be implemented on an FPGA. 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 [5]

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

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

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

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

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

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

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

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

$$\mu = 1 - \frac{2A\xi}{1-\xi+A} \tag{5}$$

with $\xi \in \mathcal{U}(0,1)$. The corresponding single elastic scattering model has been implemented in C++ then translated to the FPGA architecture.

3. Monte Carlo on FPGA

The availability of HLS toolchains greatly simplifies the FPGA programming process as it provides a high-level interface capable of logic design greatly reducing the engineering effort required. However, FPGA compile time can take up to several days and in recent times, due to the resource-increase of modern FPGAs the compile time is still increasing, even when accounting for the performance increase of modern CPUs. Thus, when targeting FPGAs Agile development is inefficient, causing many unnecessary compilations to test the design. Moreover, not all workloads benefit from FPGA acceleration, GPUs offer higher parallelism and they achieve one order magnitude higher clock frequency. Depending on the workload characteristics GPUs might achieve higher performance than FPGA or vice versa. In this study, we utilised the methodology proposed by Voss et al. [6]. This methodology can be summarised in five main steps:

(i) Workload analysis;
(ii) Performance and resource modelling;
(iii) Acceleration target selection;
(iv) Software model of the target;
(v) Hardware implementation.

We selected a junior software engineer with no previous FPGA programming experience and tasked them with following the methodology in the context of MC, to implement a representative but a simple component of the Dose Planning Method (DPM) [7] electromagnetic shower simulation and measure its performance on FPGA.

3.1. Workload analysis

Depending on the characteristics of the workload it is possible to use either FPGAs, GPUs or both to improve performance. Intuitively, CPUs and GPUs achieve an order of magnitude higher clock frequency, hence FPGAs can be faster only on workloads that are not able to exploit all the computing resources available on these architectures. Given the engineering effort and the compile-time required to program FPGA, choosing the wrong accelerator might cause a massive waste of resources and personpower that can result in performance loss instead of performance gain, hence we focus on accelerating only workloads that we deem likely to benefit from FPGA acceleration.

Workloads that are inefficient on CPUs share at least one of these two main characteristics:

- High branch misprediction rate;
- High cache miss rate.

In the case of GPUs, that lacks both branch predictors and prefetchers the performance penalty of these two characteristics is higher than CPUs, hence workloads that waste CPU compute resources will result even less efficient on GPUs. These workloads can greatly benefit from FPGA acceleration; FPGA accelerators are not affected by mispredictions penalties, different branches are implemented in hardware, and can overcome cache misses with a custom loading logic.

Due to their stochastic nature, MC simulations are prone to cache misses and branch-mispredictions, through the massive use of random numbers to drive the logic of the simulation. Thus, we conclude that while MC simulations are inefficient on CPUs and GPUs they can benefit from FPGA acceleration. In addition to the characteristics listed above there are other two optional other ones that can simplify the design and improve performance: **parallelism** and **independence**. Section 3.3 will discuss how these can be exploited in the context of MC simulations and FPGAs.

3.2. Performance and Resource model

Subsection 3.1 described the characteristics for a suitable FPGA workload. According to the Linux *perf* command, the MC simulation analysed in this study suffers from a high branch misprediction rate relative to the total number of instructions, thus this MC simulation falls in the category of workloads that is suitable for FPGA acceleration. However, this information alone is not enough to start programming the FPGA as, even if the workload is suitable, there might not be enough resources on an FPGA to achieve a meaningful increment in performance.

The original methodology assumes that FPGAs will always be used, hence, modelling is used to forecast the performance of the design, and in the case that the performance improvement is not enough, the developer has to reiterate this process multiple times until the performance goal is met. In our case, in addition to forecasting the performance gains, we also consider if FPGA accelerators can meet the project requirements: we evaluate engineering effort and potential speedup to decide if we can use FPGAs to solve our performance issue. We assume the trade-off, between engineering time and performance, is positive if the speedup is at least one order of magnitude higher than CPUs. If the speedup is less than one order of magnitude, we do not proceed with FPGA acceleration. In the MC case, the model predicted that to be at least one order of magnitude faster than CPUs, the 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 hence we further explored the second option. According to the model's predictions, when targeting a Xilinx VU9P FPGA, simulating one particle per clock cycle requires 127 DSPs and 10 Mb of on-chip memory. The 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. Trivially, 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. However, even in this worst-case scenario, the theoretical maximum achievable parallelism is 15. Hence, the parallelism of at least two, needed to meet the goal of one order of magnitude speedup should be easily achievable.

3.3. FPGA implementation

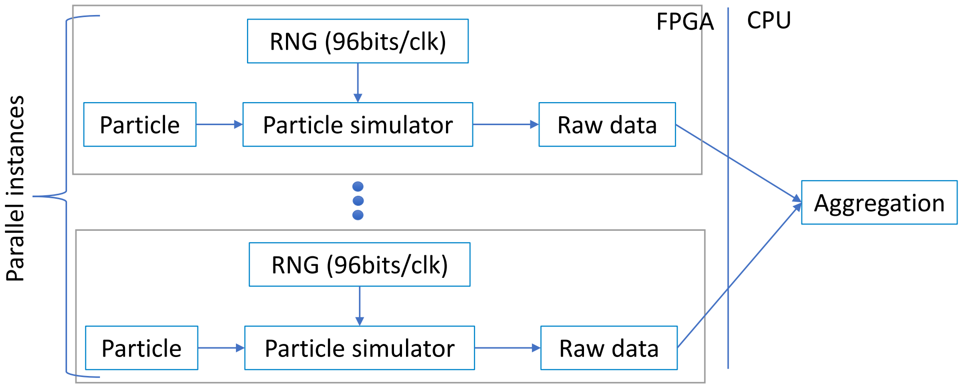

Figure 1. FPGA architecture schematic displaying the parallel FPGA instances and the CPU based aggregation.

The model's predictions show that to meet the speedup requirement the resulting architecture should be parallel. Figure 1 shows the resulting architecture. In total there are 15 instances, as predicted by the model: each instance is composed of a particle generator, a random number generator (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) random number generator [8]. This generator is not suited for use on FPGA, as it requires large memory resources compared to FPGA optimised generators, however, it is the most popular RNG and is the most likely to be available in any programming language, hence for the sake of generality it was used in this study. 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 2, 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 over 60 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 of the results is performed by the CPU. According to the model, moving the aggregation onto the FPGA would improve the performance by 15%.

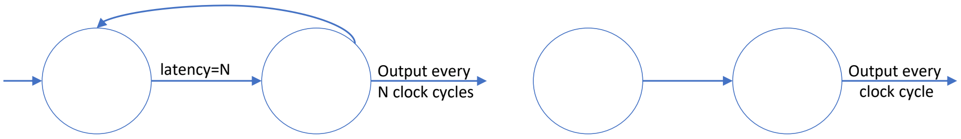

Figure 2. Illustration of the while loop optimisation impelemented.

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 to further increase the number of histories. To evaluate the performance we computed 100M histories of a 6 MeV electron interacting in water. We compared the performance of an AMD Ryzen 5900x 12-core CPU running at 3.7 GHz and boosting up to 4.8GHz and a Xilinx Alveo U200 that incorporates a VU9P FPGA device. The FPGA implementation was limited at 200 MHz, using single-precision floating-point, leaving out additional performance benefits given by the use of fixed-point data representations. 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. The results show that the FPGA implementation is 270x faster than an optimised single-core implementation and 110x times faster than a multi-core implementation. For today's market prices, this shows a cost equivalent speed-up of more than 10. It is worth mentioning that we did not perform any hardware-specific optimisation or 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, corresponding to a 160x speedup, obtained with no changes to the codebase. However, since these results are not representative of the performance obtainable with other HLS toolchains, they are mentioned but not included in our final evaluation.

5. Conclusions and Future Work

The results show that MC can greatly benefit from FPGA acceleration. In the case of Coulomb scattering, we observe a speedup of 270x and a cost equivalent speedup of over 10x according to recent market prices. These results are general since they are obtained in a worst-case study where the developer had no previous FPGA experience and did not perform any platform or compiler-specific optimisation. Hence, these results can be obtained by any programmer using any toolchain. However, due to the limitations of FPGA programming, following a clear methodology such as the one adopted in this study is crucial to developing a working implementation employing a reasonable amount of engineering time.

In the future, we would like to extend this MC simulation to perform a complete electromagnetic shower simulation by adding similar processes like multiple scattering, Moller scattering and Bremsstrahlung. These processes are similar in nature to Coulomb scattering, with the addition of a look-up table used to store the various distributions needed. Implementing such distributions on FPGA can be done with a small increase in resource utilisation [9]. Hence, we expect that the speedup observed in this study can be transferred to other and more complex MC simulations.

References

[1] Summers S and Rose A 2019 *EPJ Web of Conferences* **214** 01003 ISSN 2100-014X

[2] Niehues J and Walker D M 2019 *Physics Letters, Section B: Nuclear, Elementary Particle and High-Energy Physics* **788** 243–248 ISSN 03702693

[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 Gaydadjiev 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] Wentzel G 1926 *Zeitschrift für Physik* **40** 590–593 URL `https://doi.org/10.1007/BF01390457`

[6] Voss N, Kwaadgras B, Mencer O, Luk W and Gaydadjiev G 2021 *ACM Transactions on Architecture and Code Optimization (TACO)* **18** ISSN 15443973

[7] Sempau J W S J and F B A 2000 *Phys. Med. Biol.* **45** 2263 ISSN 0031-9155 URL `http://stacks.iop.org/0031-9155/45/i=8/a=315`

[8] T M M and Nishimura 1998 *Monte Carlo and Quasi-Monte Carlo Methods* **2000** 56

[9] Barbone M, Kwaadgras B W, Oelfke U, Luk W and Gaydadjiev G 2021 Efficient Table-Based Polynomial on FPGA *2021 IEEE 39th International Conference on Computer Design (ICCD)* (IEEE) pp 374–382 ISBN 978-1-6654-3219-1