Imperial College London

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

Marco Barbone, Rafael Brandt, Georgi Gaydadjiev, Alexander Howard, Wayne Luk, Mihaly Novak, Alex Tapper
m.barbone19@imperial.ac.uk

Imperial College London

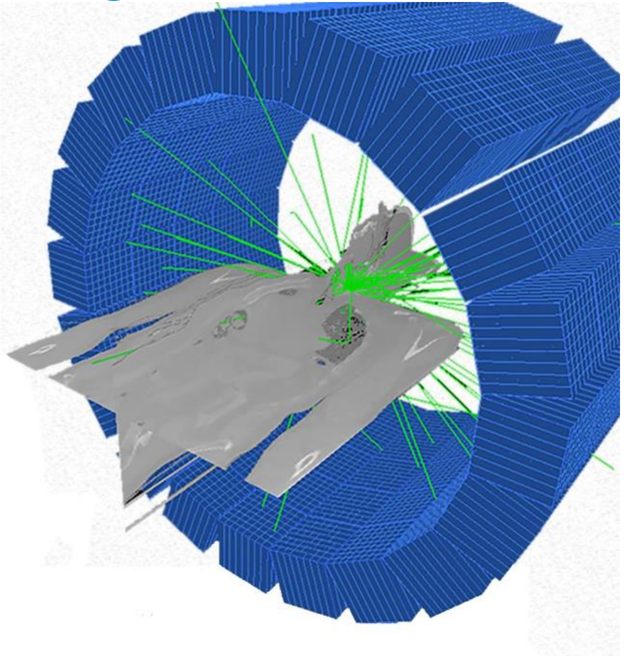
Motivation

- The medical community is starting to use GPUs to accelerate their workloads
- HEP physics shows good results with GPUs
- Analyze the potential of GPU acceleration in the context of Monte Carlo simulations
- Apply HEP-HPC concepts to medical workloads
- Considering both performance and engineering effort
- To improve the workflow for radiotherapy planning and feedback leading also to real time adaptive radiotherapy

Imperial College London

Problem statement: elastic scattering

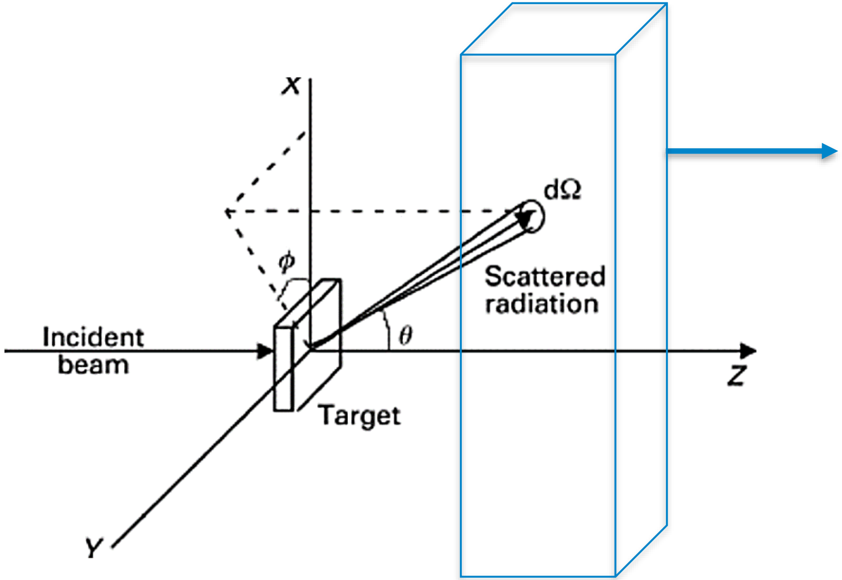
A simulation of e-/e+ transport considering only elastic scattering as possible interactions described by scattering of spin-less e-/e+ on an exponentially screened Coulomb potential



Courtesy of the Geant4 collaboration

Imperial College London

Monte Carlo Simulation



Scattered radiation final position

Many particles are simulated to achieve statistical significance

Source: Kotlarchyk M. Scattering theory. Encyclopedia of Spectroscopy and Spectrometry, 1999.

Imperial College London

GPU acceleration

Methodology

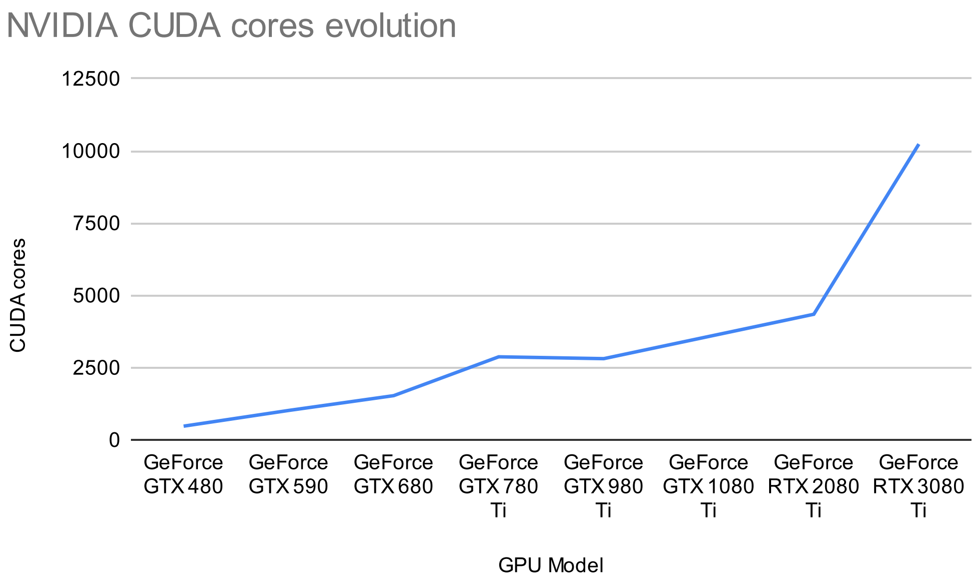
1. Analyze the parallelism available
2. Analyze problem cache and memory requirements
    - GPUs have small cache, limited memory and no prefetcher
3. Model GPU performance and data transfer overhead
4. Draft the parallel solution
5. Analyze the engineering effort of the proposed solution
6. Implement the solution

Imperial College London

Problem selection

The goal is to achieve high throughput

The problem must be embarrassingly parallel



Methodology

1. Analyze the parallelism available ✓
2. Analyze problem cache and memory requirements
    - GPUs have small cache, limited memory and no prefetcher
3. Model GPU performance and data transfer overhead
4. Draft the parallel solution
5. Analyze the engineering effort of the proposed solution
6. Implement the solution

Imperial College London

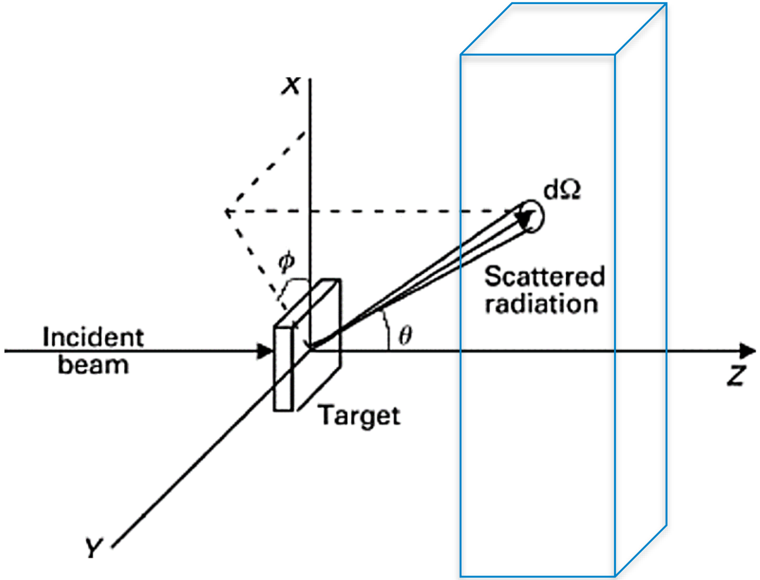
Design space exploration

Simulate the particles in parallel (independent)

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**

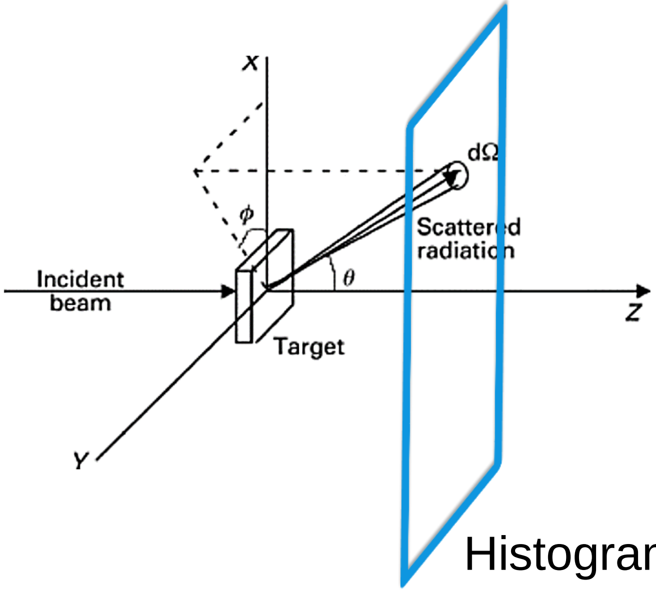
Predicted 64-Cores CPU performance: 11,252 ms

Minimize data transfers



3D

Reduction



Histograms

Methodology

1. Analyze the parallelism available √
2. Analyze problem cache and memory requirements √
    - GPUs have small cache, limited memory and no prefetcher
3. Model GPU performance and data transfer overhead √
4. Draft the parallel solution
5. Analyze the engineering effort of the proposed solution
6. Implement the solution

Experimental set-up

- Junior software engineers (computing master students' group 2 weeks)
  - Parallel implementation
  - GPU acceleration
- Senior software engineer (4 Hours)
  - Optimizing the previous solutions

Imperial College London

Test configuration

Hardware:

- 2x AMD EPYC 7551 32-Core Processor (2.0GHz)
- NVIDIA Tesla V100 PCIe 32GB (1.455 GHz)

Toolchain:

- GCC 9.3.1
- NVCC 11.6
- OpenMP

Imperial College London

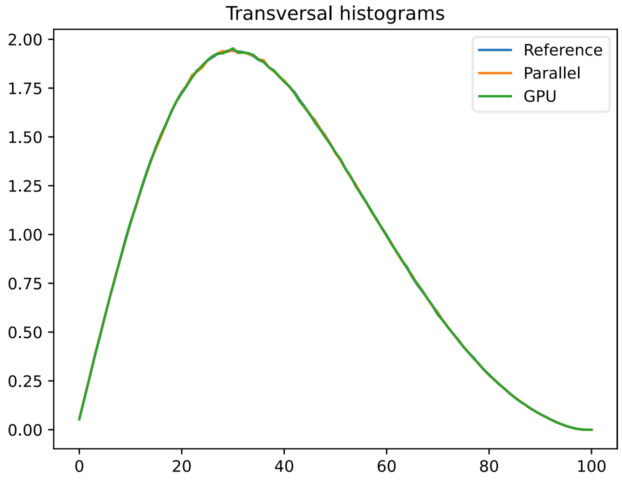
Test configurations (continued)

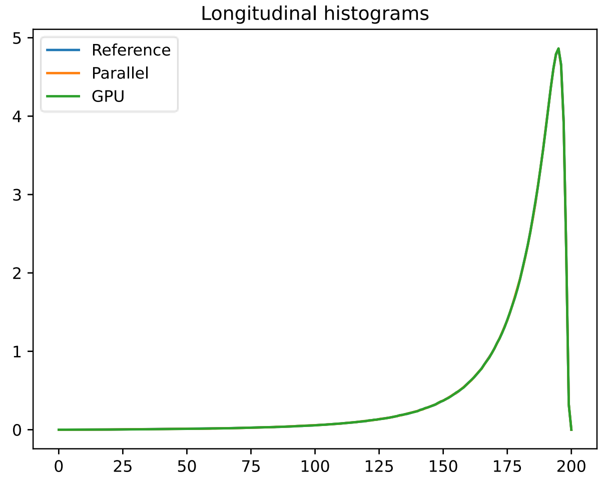
Simulation configuration:

- 100M histories
- 128 KeV beam
- Water

Imperial College London

Validation histograms





It passes a KS-test

Imperial College London

Junior developers results

- $\text{Parallel} = \frac{sequential\ time}{parallel\ time} = 0.08$

- $\text{GPU} = \frac{Sequential\ time}{GPU\ time} = 824$

The parallel version achieves 2.5x speedup on an AMD Ryzen 5900x 12-cores 3.7GHz (4.8GHz) CPU

The student failed to identify false sharing

Problem

```
std::vector<Histograms> histograms(numThreads);
#pragma omp parallel for num_threads(numThreads)
        for (int i = 0; i < n_particles; i++) {
            auto& thread_histogram = histograms[tid];
            ...
        }

```

Solution

```
#pragma omp parallel num_threads(numThreads)
        Histograms thread_histogram;
        # pragma omp for
        for (int i = 0; i < n_particles; i++) {
            ...
        }

```

Imperial College London

Senior developer results

- $\text{Parallel} = \frac{sequential\ time}{parallel\ time} = 84.78$

- $\text{GPU} = \frac{Sequential\ time}{GPU\ time} = 843.36$

The more experienced developer achieved 1091x and 1.02x performance increase compared to the junior developer

Imperial College London

Optimizations

More involved optimizations (e.g. explicit shared memory use) did not further improve GPU performance whilst greatly increasing engineering effort

Imperial College London

CPU vs GPU results

- $\mathrm{GPU} = \frac{parallel\ time}{GPU\ time} = 9.95$

100 M Particles

- $\mathrm{GPU} = \frac{parallel\ time}{GPU\ time} = 47.8$

1B Particles

Imperial College London

Conclusions

- GPUs can be orders of magnitude faster than CPUs in the context of Monte Carlo simulations
- Parallel implementations can be harder to implement compared to GPU implementations
- The usage of GPUs in the medical context can lead to real-time adaptive radiotherapy
- Code available: https://github.com/DiamonDinoia/mcss/