Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Hes·so

Student-s:
Barras Simon

Bachelor Abstract 2023
Computer science and communication systems

SUPERVISOR: Frederic Bapst, Jean Hennebert	
COMMISSIONED BY: Lawrence Berkeley National Laboratory, Paolo Calafiura, Julien Esseiva	
INTERNAL CONTACT: *iCoSys*	
PROJECT ABBREVIATION: GPU Optimization Celeritas	INTERNAL PROJECT NR.: B23ISC27
MAJOR :Computer scientist	EXPERT: Dr. Baptiste Wicht
SUSTAINABILITY GOALS:	

GPU Optimization in Celeritas, a High Energy Physics Detector Simulation Library.

As the detectors at CERN's LHC produce more and more data, simulations for the ATLAS and CMS experiments can no longer make with CPU-only tools. The arrival of GPUs in the supercomputers opens up new development perspectives. This is what the Celeritas team is exploiting, to provide a library of GPU-accelerated particle physics simulations.
For this bachelor thesis, Simon Barras went to Berkeley in the USA to work at the Lawrence Berkeley National Laboratory. There, he collaborated Celeritas team, which is located to several laboratories all over the USA, notably Oak Ridge.

Celeritas

This project is a Monte Carlo particle transport library for simulating High Energy Physics detectors. It is motivated by the massive computational requirements of the ATLAS and CMS experiments in the LHC at CERN. The goal is to replace Geant4 which is the legacy tool for the simulation of particles. Celeritas is developed to be launched on supercomputers as Perlmutter, the one of the NERSC. The new supercomputing emerging have more and more part of their computational resources delivered by GPUs.

CELERITAS

GPU

Graphical Processor Unit (GPU) are some chips that are defined as Single Instruction, Multiple Data (SIMD) or Signe Instruction, Multiple Threads (SIMT). These cards are efficient when a small set of operations have to be made on a large scale of data.
To develop a program using GPU, the platform of the GPU is needed to define which part of the code has to be run on the device. For instance, the platform for Nvidia is called CUDA. When we launch a kernel, we define the number of blocks and number of threads per block.

Dormand Prince

Runge Kutta Dormand Prince (RKDP) is an ordinary differential equation solver of the fourth and fifth order. It is defined by the coefficient from the Butcher Tableau and to get the fifth order, it is necessary to do six iterations but the implementation is made of nine steps. The first six are computing the intermediate step using the equation and updating the current state. The three last steps are mostly computing the error and the resulting point state.

Design

It is important to have a good understanding of the code to find a way to parallelize that even if RKDP is iterative and it seems impossible.
The implementation of the method in the project has three types of code. The first category is the compute the intermediate state with the equation. The second part is the code that has no

Student-s:
Barras Simon

dependencies and it is the multiplication of the coefficients by the step size. The last category is the update of the state which is a two 3-Dim vectors. This part can be split into subtasks that can be parallelized. Figure 1 shows a profile of the execution time for each step and each type.
According to these results and assuming the new implementation will be distributed to four threads, it is possible to distribute the tasks to reduce the execution time of the method. Figure 2 shows an example of optimal workload.

Implementation

Two implementations have been made. Both have four threads, the first one is designed as a controller and it computes the sequential part when the three workers compute the vector multiplication. The coefficients are only computed on the fly because storing into variables slow the process. The difference between the two implementations is the usage of the shared memory. This memory space is faster than the global but it size is limited as few kilobytes per block. Figure 3 is two diagrams that illustrates the communication within the threads.

Results

The result of the two implementations are pretty good if only one block is used. It speedup the process with an average of 150% but they can track less particles than the old version. The global memory version is limited by the number of threads per track and the version with the shared memory is limited by itself size. Figure 4 shows the speedup and the limitations.
As Celeritas want to track a large number of particles, it runs on more than one block.
Figure 5 shows the evolution of the time on the number of tracks. The results are pretty bad because the new version have some difficulties to scale up. Every time 84 blocks are requested (approximately 15'000 tracks for the new implementations) the amount of resources requested to GPU exceeds what it runs at a time. These exceed are illustrated as the jump shows on the graph.

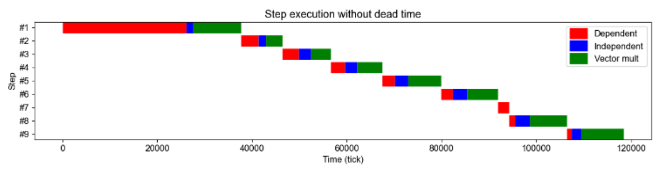

Figure 1 Results of the RKDP profiling

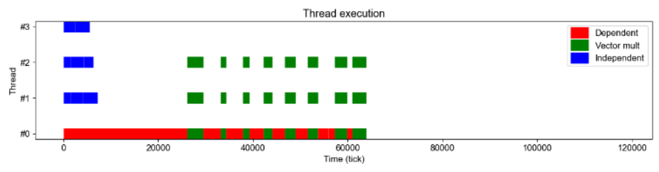

Figure 2 Optimal task scheduling

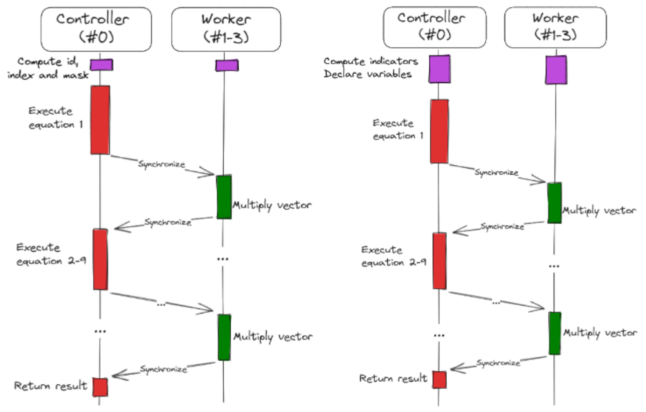

Figure 3 Implementation with respectively global and shared memory

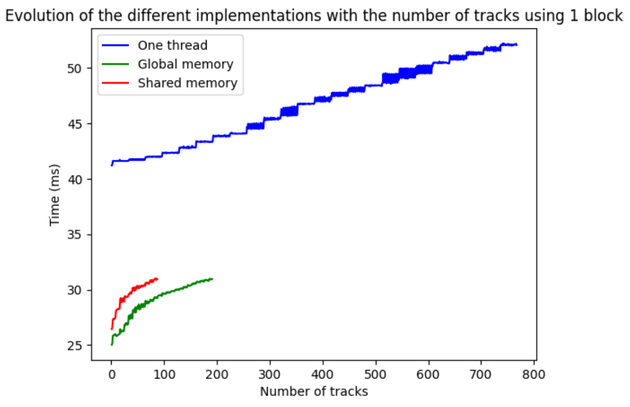

Figure 4 Performance with one block

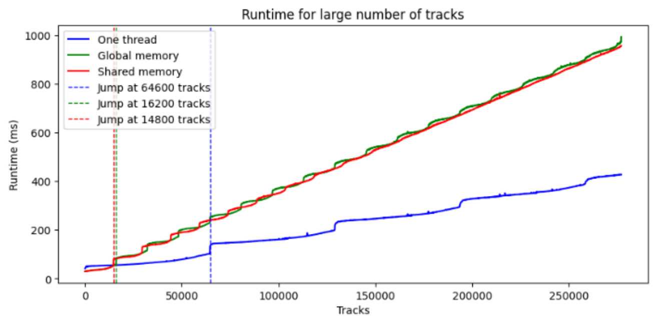

Figure 5 Performance with multi-block