FaMAF
Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba

GPGPU
General Programming on Graphic Processing Units

CSCAMM

NVIDIA

Integration of Post-Newtonian Equations on GPUs

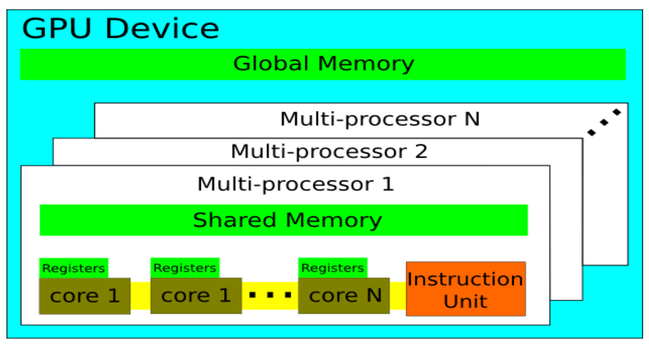
Matías Bellone[1], *Manuel Tiglio*[2], *Frank Herrmann*[2], *John Silberholz*[2]
[1] Universidad Nacional de Córdoba, Argentina
[2] CSCAMM, University of Maryland, College Park, MD

ABSTRACT

Given a variable-timestep integrator, the most simple parallelization would be to run many of them at the same time with different starting conditions. As the integrators are completely independent from each other, this problem can be labeled under the “embarrassingly parallel” category.

The original C code performed the integration of a 10-variable system and was ported to run on Nvidia's CUDA framework for parallelization on CUDA-capable video cards. Speed ups were really good; nonetheless they could be improved.

GPU Architecture

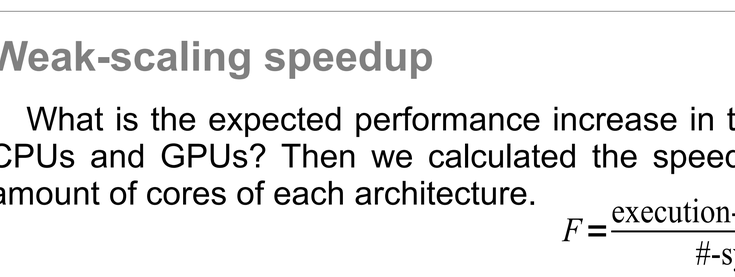


GPUs are very different from regular multi-core CPUs. It's cores are grouped into *multi-processors* and its instruction unit can have each core run more than one thread with no overhead making sure that threads locked waiting for memory data leave place for other threads that can make better use of the resources.

This allows for the execution of massive amount of threads. To facilitate mapping of those threads to the application's data, CUDA provides a logical organization of threads in 3-dimensional arrays (*blocks*) where each block can contain a maximum of 512 threads. And blocks themselves can be arranged on a 2-dimensional array (*grids*) and the indexes of the current block and thread are available as a variable for use an one sees fit.

GPU memory layout improvements

GPU's architecture include a specialized memory manager that favors certain access patterns over others. In particular, accessing contiguous memory blocks is better than accessing scattered ones.



Modifying the layout of the variables' array of a system so that memory access favoured such pattern was the one improvement made to the system.

2.5x

Further parallelization

Even when the application is completely parallelized by integrating several systems simultaneously, the code that update estimates (see code structure) are exactly the same for all 10 variables of one system. It made sense, then, to change the code so that it would process just one variable instead of the whole system and parallelizing at that level.

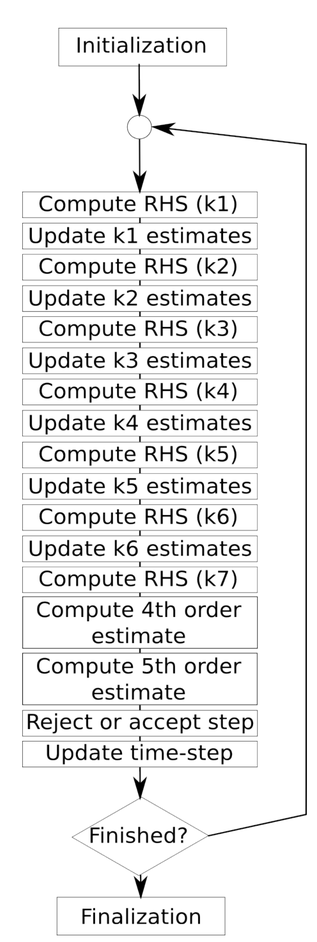
The result was that there are portions of the code that run one thread per system, and others that run one thread per variable per system. Which CUDA allows and even encourages running as many threads as possible.

CPU parallelization

For a more fair comparison between the CPU and the GPU version of the code, the CPU code was also parallelized on the system-level by using OpenMP resulting on the application running one thread per system.

8x

Code structure



Theoretical improvement

The classification of the application as “embarrasingly parallel” means that, if one has more cores available, they should all be able to run 100% in parallel. The theoretical speedup expected would be 1x per core.

1x/core

Weak-scaling speedup

What is the expected performance increase in the future when we have more powerful CPUs and GPUs? Then we calculated the speedup factor taking into consideration the amount of cores of each architecture.

$$F = \frac{\text{execution-time} \times \#\text{-cores}}{\#\text{-systems}}$$

If one considers that quantity for both architectures and the fact that the multi-threaded CPU code averaged 0.0398 secs per system whereas the GPU code averaged 0.0000824 seconds per system. The GPU code has a 16x factor over the CPU.

16x

Effective speedup

Given the performance seen in the code where the best implementations on each architecture yielded 3.98×10^{-2} *secs/system* on the CPU and 8.24×10^{-5} *secs/system* The speedup we could see is 483x.

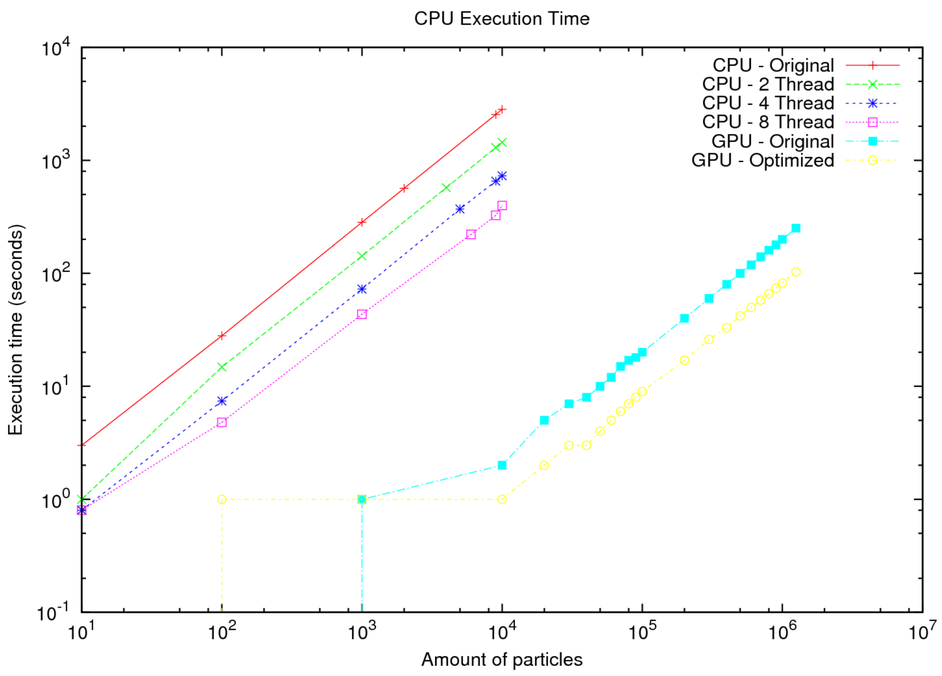
483x

Practical speedup

When running on cluster environments, resources offered are: a single core on CPU clusters, and 1 whole GPU on GPU clusters. Thus for all practical purposes, based on execution times of the single-core CPU version, the overall speedup achieved is 3400x.

3400x

Performance Comparisons



Empirical execution times on Dell Precision R5400 (Quad Core Xeon E5410 2.33GHz) CPU and 1 of the 4 240-core GPUs of Nvidia's Tesla S1070 running at a peak of 1.44Ghz.