Summer 2014

GPU Acceleration of 3D lattice simulations

C. Ross Mauck

GPU Acceleration of 3D lattice simulations

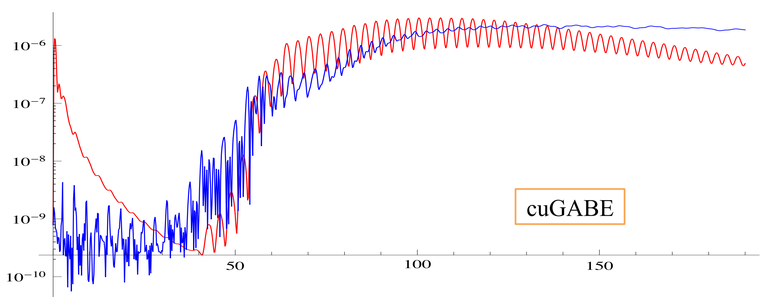
C. Ross Mauck '15 and Professor John T. Giblin,Jr.
Department of Physics, Kenyon College

Abstract

In today's realm of complex theoretical high-energy physics problems, meaningful predictive simulations are bound by computation time. We propose an accelerated Graphics Processing Unit (GPU) implementation of a finite difference 3D lattice simulation in an expanding background. Grid and Bubble Evolver (GABE), the original application, evolves interacting scalar fields over time using equations of motion. We use Nvidia's CUDA language and heterogeneous-programming platform to port the original C plus plus application to the GPU. The GPU we use in these implementations is Nvidia's Tesla K40m. We compare the serial and multi-threaded CPU implementations to the GPU accelerated implementation. Multiple methods utilizing the GPU's complex architecture are described. The computation time speed-up given by the GPU based implementation indicates the importance of such optimizations.

CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model implemented on graphics processing units. Using CUDA, we implement our GPU for general purpose processing (known as GPGPU). The CUDA platform provides CUDA-accelerated libraries to industry-standard programming languages such as C++. CUDA kernels are GPU functions that launch an organized group of threads. These threads perform computation on large sets of data simultaneously.



The plots above compare the phi (blue) and chi (red) field variances of the cuda enabled GABE and original GABE. Both plots exhibit the expected increase in mean and variances over time.

Motivation

The shear computational power, of today's GPUs, offers computational scientists the ability to accelerate numerical simulations that are inherently parallelizable.

Evolving interacting scalar fields in a 3D lattice fits our description of a parallelizable simulation. Each point of the lattice must run through the same number of computations. It is obvious to see that as we increase our box size, the number of points in our lattice grows as a function of the box size cubed. In the past, we have been limited by number of processors. Using our Tesla K40m GPU we are able to employ 1000s of processors at once.

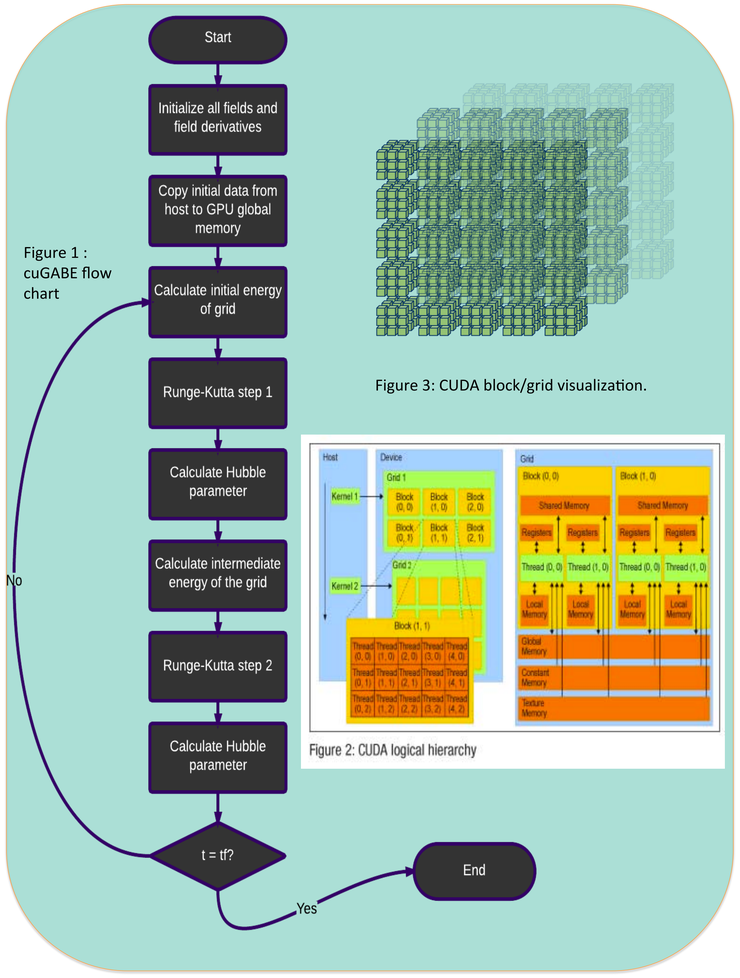


Figure 1 : cuGABE flow chart

Figure 3: CUDA block/grid visualization.

Figure 2: CUDA logical hierarchy

GABE

Numerical tools are fundamental in the understanding of modern cosmology. GABE is the original C++ program that was ported to the GPU. GABE was written by John T. Giblin Jr. and two Kenyon alumni, Tate Deskins and Hillary L. Child. GABE evolves a 3D lattice of scalar fields in an expanding background.

For both GABE and GPU accelerated GABE, we use a Runge-Kutta 2 method at each point in the lattice. GABE uses OpenMP, which allows parallelization by a mulit-threaded CPU. Depending on the resolution, meaningful GABE simulations require hours, days or weeks.

Performance

We compared three different architectures (serial CPU, multi-threaded CPU, and GPU) at varying lattice sizes (128, 256, and 512). All simulations were ran on our local computer, nicknamed Ann. The serial CPU code ran using latticeeasy (serial version of GABE). The multi-threaded CPU code ran using GABE with 16 threads. The GPU code was ran using cuGABE on 2880 CUDA cores.

Comparing computation time in seconds	N=128 (dt = 0.005)	N=256 (dt = 0.001)	N=512 (dt = 0.0005)
Serial CPU (latticeeasy)	85	2,940	N/A
Mult-threaded CPU (GABE)	58	1,995	37,056
GPU (cuGABE)	10	46	310

Acknowledgements

I would like to thank the NSF for funding this project. I would like to thank John T. Giblin, Jr. for this opportunity and his help throughout this project.
I would like to thank Elizabeth Halper for her work in the early stages of development. Also, I would like to thank Ann and Eliza for their participation as GPU translators.

Future Work

We plan to continue to optimize our code by using CUDA's dynamic parallelism. Dynamic parallelism will allow the GPU to create new work for itself. Effectively, this feature will reduce the GPU's dependence on the CPU.