GPGPU IMPLEMENTATION OF MATRIX FORMALISM FOR BEAM DYNAMICS SIMULATION

N. Kulabukhova*, Saint-Petersburg State University, Russia

Abstract

Matrix formalism is a map integration method for ODE solving. It allows to present solution of the system as sums and multiplications of 2-indexes numeric matrix. This approach can be easy implement in parallel codes. As the most natural for matrix operation GPU architecture has been chosen. The set of the methods for beam dynamics has been implemented. Particles and envelope dynamics are supported. The computing facilities are located in St. Petersburg State University and presented by the NVIDIA Tesla-based clusters.

INTRODUCTION

The performance available on graphics processing units (GPUs) has led to interest in using GPUs for general-purpose programming [1]. It is difficult, however, for most programmers to program GPUs for general-purpose uses.

The raw computational power of a GPU dwarfs that of the most powerful CPU, and the gap is steadily widening. Furthermore, GPUs have moved away from the traditional fixed-function 3D graphics pipeline toward a flexible general-purpose computational engine. Today, GPUs can implement many parallel algorithms directly using graphics hardware. Well-suited algorithms that leverage all the underlying computational horsepower often achieve tremendous speedups[2]. Truly, the GPU is the first widely deployed commodity desktop parallel computer.

Here, we provide a comparison of different sort of parallel technologies of the classic graphics pipeline; our goal is to highlight those aspects of the real-time rendering calculation that allow graphics application developers to exploit modern GPUs as general-purpose parallel computation engines.

COMPARISON OF GPU AND CPU

The highly parallel workload of real-time computer graphics demands extremely high arithmetic throughput and streaming memory bandwidth but tolerates considerable latency in an individual computation since final images are only displayed every 16 milliseconds. These workload characteristics have shaped the underlying GPU architecture: Whereas CPUs are optimized for low latency, GPUs are optimized for high throughput. The GPUs specialized architecture is not well suited to every algorithm. Many applications are inherently serial and are characterized by incoherent and unpredictable memory access. Nonetheless, many important problems require significant computational resources, mapping well to the GPUs many-core arithmetic intensity, or they require streaming through large quantities of data, mapping well to the GPUs streaming memory subsystem. Porting a judiciously chosen algorithm to the GPU often produces speedups of five to 20 times over mature, optimized CPU codes running on state-of-the-art CPUs, and speedups of more than 100 times have been reported for some algorithms that map especially well. Some successful examples of using GPUs are described in [2].

In contrast, most CPU programs use a sequential programming model and are not benefiting from the continued increase in transistors due to Moores law. They will need to be rewritten or modified substantially to obtain increased performance from new CPUs with multiple cores on a chip. Furthermore, CPU clock speeds have plateaued due to power concerns. On the other hand, GPU architectures have a number of disadvantages for parallel programming. First, GPUs have a SIMD programming model. GPUs are moving toward a SPMD model, although use of loops and control-flow instructions in GPU pixel shaders may currently degrade performance, not improve it. Second, the actual architectures of GPUs are hidden behind device drivers that support APIs that implement virtual machines. This abstraction and lack of detail can hamper obtaining the most performance out of a graphics processor. For example, the virtual machines have no model of caches and no easy way for programmers to indicate how to traverse memory to facilitate memory reuse. This can make it difficult to write parallel programs where memory locality is crucial to performance. In addition, we must target APIs designed to support graphics that introduce extra complexity and overhead. Third, the programmable parts of GPUs have had limited support for primitive types typically found on CPUs.

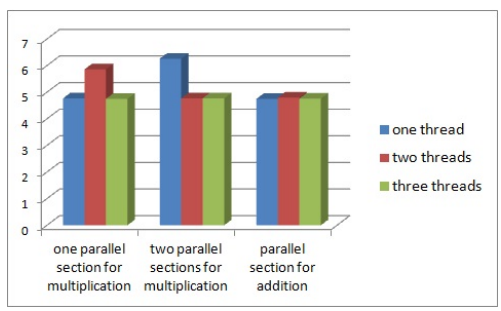

Figure 1: Diagram of comparative performance.

* kulabukhova.nv@gmail.com

ISBN 978-3-95450-116-8

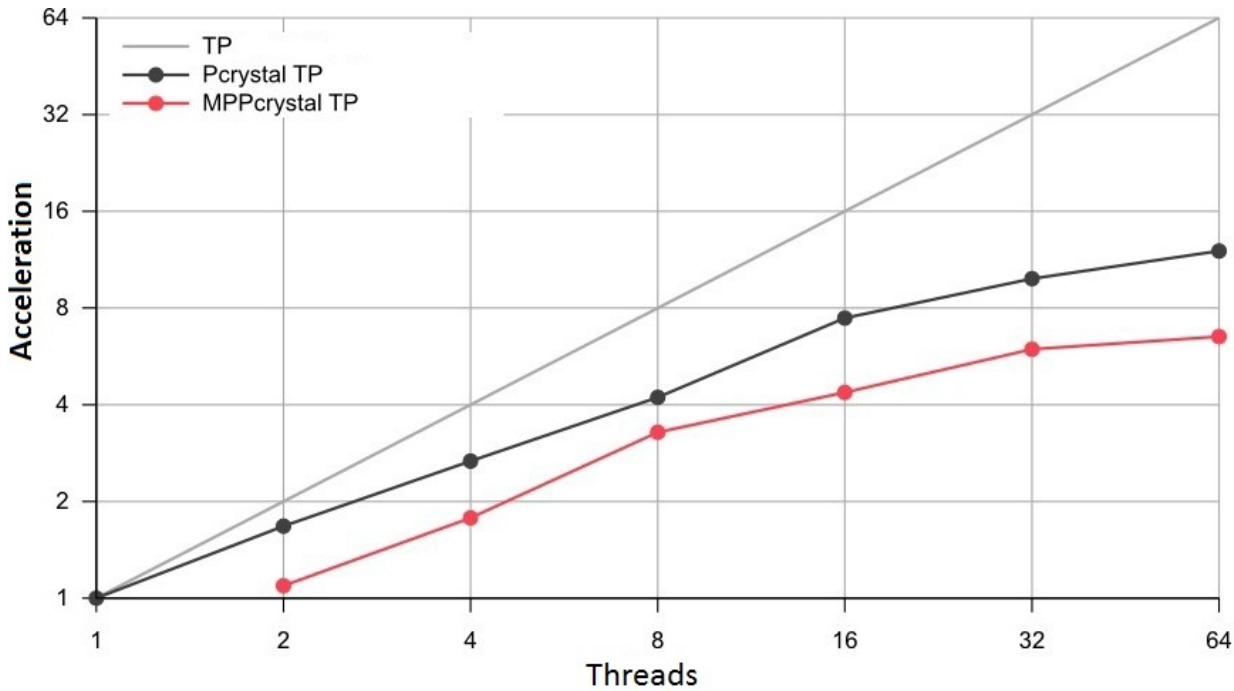

Figure 2: Comparison of SPbSU T-platform.

PARALLEL TECHNOLOGIES

Methods, libraries, interfaces to parallel a program are a lot to choose. There are different groups of technologies, those who use:

- a high-level communication libraries and interfaces (API) (MPI, MPL, OOMPI, OpenMP);
- special "parallelizing" structures in the programming language (MPC++, mpC, Ada,MC#, Cray MPP Fortran);
- automatic parallelization of sequential programs (FORSE, KAP, PIPS, VAST, V-Ray);
- parallelized procedures of the specialized libraries (ATLAS, PLAPACK, PIM, PARPACK);
- specialized software packages (ANSYS, ABAQUS, CFX, FLOWVISION, LMS Virtual Lab., GDT).

We evaluate the effectiveness of using data parallelism to program GPUs by providing results for a set of compute-intensive benchmarks. All calculations were performed on a hybrid cluster of SPbSU (see Fig. 2) computing center. Its nodes contain a NVIDIA Tesla S2050 system that was developed specifically as a GPGPU unit. For our goal we choose OpenMP technology.

OPENMP

When we speak about parallel computations, some problems must be solved. First, the time is spent on writing the program. Code must be effective and removable. It is important to view how this program will be used in future. The OpenMP API using the fork-join model of parallel execution is suitable for these aims.

OpenMP parallel program is constructed on sequential code by adding directive, procedures, process variables. This technology bases on the concept of shared memory, thats why Symmetric Multiprocessing is used (SMP computers). For this architecture threads (flows), running on different processors, easy to support. For classic UNIX-processes is more expensive to do this[4].

Scheme FORK/JOIN is used to support the parallel code. Entering the parallel environment thread-master generates complementary threads (operation FORK is running). After that each thread has its own number, thread-master has zero number. All threads run the same code of the parallel environment. After work thread-master waits ending of all other threads and continues to do next step (operation JOIN is running).

SIMULATION

Matrix formalism [5, 6] is a high-performance mapping approach for ODE solving. It allows to present solution of the system in following form

$$X = \sum_{i=0}^{k} R^{1i}(t) X_0^{[i]}, \qquad (1)$$

where R^{1i} are numerical matrices. So this approach can be easy implement in parallel code. Due to the fact that only matrix multiplication and addition are used, GPU programming is especially suitable for this purpose.

There are exist two way for beam dynamics simulation:

ISBN 978-3-95450-116-8

- based on particle simulation;
- based on envelope description.

Particles Simulation

The research have shown that there is no great benefits via parallelization of computational code for one particle by using GPU (see Fig. 1) In Table 1 performance of different parallelization modes are presented, where

- **1 mult** means one parallel section for multiplication;
- **2 mult** means two parallel section for multiplication;
- **add** means one parallel section for addition.

In this case overhead on data sending is significant. On the other hand matrix formalism allows to process a set of the initial points, where parallelization is more preferably.

Let's introduce a set of initial particle

$$M = (X_0^1 X_0^2 \dots X_0^p).$$

In according to the equation (1) the resulting points can be calculated

$$M = \sum_{i=0}^{k} R^{1i}(t)((X_0^1)^{[i]}(X_0^2)^{[i]} \dots (X_0^p)^{[i]}). \qquad (2)$$

Note that the sizes of matrices in the equation (2) is much greater than in (1) when a set of initial particles is quite large.

Table 1: Performance of Different Parallelization [sec]

	1 mult	2 mult	add
1	4.77	6.26	4.75
2	5.86	4.77	4.8
3	4.75	4.77	4.76

Envelope Simulation

Beam dynamics description based on envelope provides an efficient approach to modeling. Let's consider the envelope simulation is linear case (Fig. 3). In nonlinear case the equations are quit difficult, but the concept is based on linearization by introducing an extended space.

In linear case equation (1) is wrote in following form

$$X = R \cdot X_0.$$

The elliptical envelope can be described by a quadratic form

$$X^* A X < 1,$$

where X^* means transpose of vector X.

By these equations a new envelope can be obtained:

$$X^*(R^* A R)X < 1,$$

where $R^* A R$ is a matrix of new quadratic form.

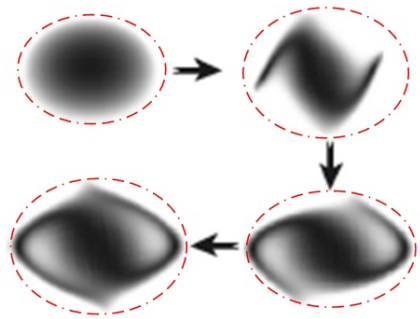

Figure 3: Envelope simulation.

CONCLUSION

Matrix formalism is a high-performance approach for beam dynamic modeling. The method can be implemented in parallel codes on GPU. It allows simulate both long-term evolution of a set of particles, and evaluating based on envelope description.

The future development of the research can be based on other parallel techniques investigation and complete implementation of the described approaches.

ACKNOWLEDGMENT

Computations were partly carried out on cluster HPC-0011654-001 of Saint-Petersburg State University, Faculty of Applied Mathematics and Control Processes. Special thanks for my scientific supervisor S. Andrianov.

REFERENCES

[1] D. Tarditi, S. Puri, J. Oglesby, "Accelerator: Using Data Parallelism to Program GPUs for General-Purpose Uses," proceedings of ASPLOS, San Jose, California, USA, 2006, pp. 325–335.

[2] D. Luebke, G. Humphreys, "How GPUs Work," How Things Work, Computer, February, 2007, pp. 126-130.

[3] A. Degtyarev, I. Gankevich, "Efficiency Comparison of Wave Surface Generation Using OpenCL, OpenMP and MPI," Proceedings of 8th International Conf. Computer Science and Information Tech., Yerevan, Armenia, 2011, p. 248–251.

[4] V. V. Voevodin, "Mathematical foundations of parallel computing," World Scientific Publishing Co., Series in computer science, 1992, pp. 33–343.

[5] S. Andrianov, "The Convergence and Accuracy of the Matrix Formalism Approximation," TUSCC2, ICAP2012.

[6] A. Ivanov, S. Andrianov, "Matrix formalism for long-term evolution of charged particle and spin dynamics in electrostatic fields," WEACC3, ICAP2012.

ISBN 978-3-95450-116-8