

High performance stream computing for particle beam transport simulations

R. Appleby, D. Bailey, J. Higham, M. Salt

School of Physics and Astronomy, University of Manchester, Oxford Road, Manchester, M13 9PL, UK

Robert.Appleby@manchester.ac.uk, David.Bailey-2@manchester.ac.uk

Abstract. Understanding modern particle accelerators requires simulating charged particle transport through the machine elements. These simulations can be very time consuming due to the large number of particles and the need to consider many turns of a circular machine. Stream computing offers an attractive way to dramatically improve the performance of such simulations by calculating the simultaneous transport of many particles using dedicated hardware. Modern Graphics Processing Units (GPUs) are powerful and affordable stream computing devices. The results of simulations of particle transport through the booster-to-storage-ring transfer line of the DIAMOND synchrotron light source using an NVidia GeForce 7900 GPU are compared to the standard transport code MAD. It is found that particle transport calculations are suitable for stream processing and large performance increases are possible. The accuracy and potential speed gains are compared and the prospects for future work in the area are discussed.

1. Introduction

The recent demand for realistic computer graphics, motion and gaming physics models has led to a significant steps forward in the hardware and programming techniques available for graphics processing. This progress has led to the development of dedicated and affordable computer graphics processing units (GPUs), which are designed to work in tandem with conventional CPUs to handle the numerous and repetitive operations required for graphical calculations. The GPU works performing high speed arithmetical operations on multiple input streams of data, known as stream processing [1], with the repetitive operation being encoded in the GPU kernel acting on the data streams. Stream processing has resulted in massive computational power being readily available, provided the computational problem is amenable to the stream processing approach.

A suitable source of computational problems appropriate for stream processing comes from the field of accelerator physics. This subject is concerned with the simulation, design and construction of machines to accelerate and manipulate beams of sub-atomic particles, typically containing 10^{11} particles at any given time, and their subsequent exploitation. The applications of this field are dominated by the scientific, for example high-energy particle physics, and the medical, for example the diagnostic and therapeutic devices found in most hospitals. The fundamental layout of these devices is a suitable source of sub-atomic particles, followed by structures to accelerate and manipulate the particle beams, and the successful design and operation requires a detailed understanding of the motion of the particles through the accelerator. These calculations typically require tracking of many particles trajectories through the accelerating, bending and focusing elements of an accelerator to understand the spatial extent of a particle beam and the stability of motion. The required accuracy and extremely high number of particles in an accelerator, coupled with the frequent need to study many revolutions of a circular accelerator, result in an extremely computationally

intensive problem. However, the stream-like form of the particle data means this problem is well suited to the techniques of stream processing.

In this paper, the techniques of stream processing using the NVidia GeForce 7900 GPU [2] are applied to the problem of beam transport in a section of the DIAMOND light source accelerator [3]. This accelerator, known henceforth as DIAMOND, is a circular particle accelerator and storage ring located close to Oxford, UK. Stream processing is used to calculate the motion of very large numbers of particles in a linear section of the machine known as a transfer line, where the spatial extent of the particle beam and the conditions at the end of the transfer line are required to be known to very high accuracy for the subsequent particle beam storage. This paper compares the accuracy and speed of the GPU particle trajectory calculation to a conventional beam dynamics calculation using the research grade simulation code MAD [4] to demonstrate the applicability of the stream processing computational power to this class of problem.

This paper is organized as follows. In section 2 the concepts of stream processing and the realisation using GPUs are described. In section 3 the formulation and calculation techniques of particle beam dynamics are discussed, along with a brief description of DIAMOND and the relevant transfer line. The implementation of the particle tracking on the GPUs is discussed in section 4, and the results are presented in section 5. Finally the conclusions are drawn in section 6.

It is found that the calculation of particle beam dynamics in an accelerator is suitable for stream processing, and considerable increases in particle tracking speed are possible. The impact is the possibility of greater numbers of tracked particles and a resulting increase in accuracy and a better understanding of beam dynamics in a particle accelerator.

2. Stream computing and graphics processing units (GPUs)

Stream computing is a relatively new concept that allows the parallel processing of potentially large sets of data (streams). Processing is achieved by applying a compute-intensive operation (kernel function) to each element of the data stream in turn [1].

GPUs have been optimized to carry out huge numbers of simple calculations on large datasets in order to render realistic images in real time for the gaming industry. It has recently been realized that these processors are simply running streaming computations in a massively parallel environment – in principle, each pixel on the screen can be regarded as a stream computation. Although GPUs are not as flexible as conventional CPUs, the potential performance gain using them to solve specific problems makes them an attractive platform to investigate. Indeed, all of the major GPU manufacturers are now targeting the scientific market with toolkits that allow optimized implementation of algorithms on their GPUs [5]. There is also an older project started by Stanford University to exploit the Brook language on a GPU [6].

The study presented in this paper is a proof-of-concept use of the Brook language on GPUs to improve the performance of beam-dynamics simulations for accelerator design. This problem lends itself naturally to the GPU environment as potentially huge data sets (the particles being transported though the machine) are operated on sequentially by representations of the magnetic machine elements. A conventional CPU would have to process each particle sequentially, but the GPU can handle many in parallel – the number being, in principle, only limited by the memory accessible to the GPU itself.

3. Particle beam dynamics

3.1. Transport of particles in a particle accelerator

The motion of a quasi-mono-energetic collinear bunch of particles through an accelerator system is described through a formalism based on linear and non-linear maps acting on particle phase space vectors [7]. The vectors denote the position of a given particle in 6-dimensional phase space (position and momentum in each dimension and three dimensions), and a bunch of particles are typically Gaussian distributed in this space. The particle bunch moves through the various accelerator components, each of which acts on the bunch according to a transfer map. A typical accelerator is composed of accelerating elements (RF cavities), which use a longitudinal electric field to accelerate the particles, and magnetic elements, which use magnetic fields to bend and focus the particle beam. The transfer maps of a particular accelerator element are computed by solving the equations of motion in the appropriate electromagnetic field.

In this work, we use the canonical phase space description for the beam particles, and a particle vector is written in terms of canonical variables as

$$\underline{x} = \begin{pmatrix} x \\ p_x \\ y \\ p_y \\ t \\ p_t \end{pmatrix},$$

where x and y denote the horizontal and vertical transverse positions, p_x and p_y denote the transverse canonical momenta and (t, p_t) describe the longitudinal phase space. The choice of canonical variables follows from the Hamiltonian description of particle motion and is necessary for long-term particle motion stability of many turns of an accelerator, a condition known as symplecticity. The action of an accelerator element is given by the Taylor mapping of an initial vector x to a final vector z

$$z_j = \Delta x_j + \sum_{k=1}^6 R_{jk} x_k + \sum_{k=1}^6 \sum_{l=1}^6 T_{ijk} x_k x_l,$$

where R_{ij} denotes the linear map, T_{ijk} denotes second order corrections and Δx represents constant additive terms. Note the truncation of the Taylor series means the mapping is no longer symplectic. As an example, the linear map for a drift space (a field-free region of an accelerator) is given by

$$R = \begin{pmatrix} 1 & L & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{L}{\beta^2 \gamma^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

where L is the length of the region, $\beta=v/c$ and γ denotes the relativistic gamma function. Note the three planes of motion are uncoupled in this example. The optical elements relevant to this work are the drift region, the dipole bending magnet, the focusing quadrupole magnet and the particle trajectory

correcting kicker magnet. In this paper, the standard accelerator design code MAD [4], which is an implementation of the transfer maps, was used for benchmarking and accuracy comparison.

3.2. The DIAMOND light source

The DIAMOND light source [3] is a 3 GeV electron storage ring, designed to exploit the synchrotron light emitted by accelerating charged particles. The accelerator complex consists of an electron source to produce bunches of electrons, a linear accelerator structure, a booster ring to increase the electron energy and a main storage ring. The electrons are stored in the main storage ring, which has a circumference of 561.6m and consists of 24 repeating magnetic structures, for many hours at a beam current of 300 mA, and the light they produce is directed towards a large variety of experiments situated around the ring.

In this work we shall study the beam dynamics of an electron bunch in the piece of accelerator carrying the particles from the booster ring to the storage ring. This is known as the booster-to-storage-ring (BTS) transfer line. This line contains no accelerating structures, so the electrons leave the line with the same energy they started with, and is composed of a series of bending (dipole) and focusing (quadrupole) magnets designed to transport the beam and prepare it for injection into the main ring. Beam collimators perform this beam preparation. The layout of the magnetic elements of the transfer line is shown in figure 1.

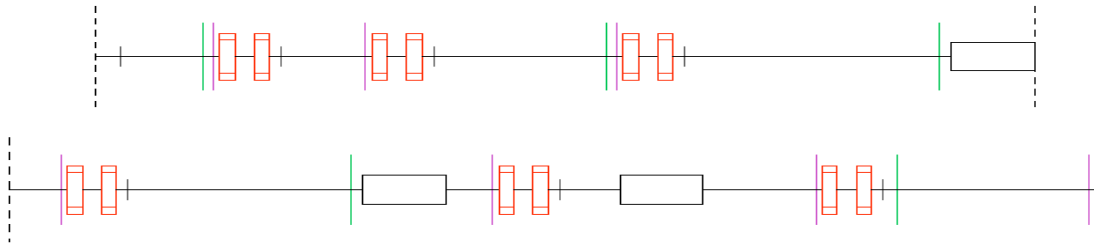


Figure 1: The magnetic layout of the BTS. Dipoles are black, quadrupoles are red and collimators are green.

The beam parameters at the start of the BTS (in the booster ring) are known [8] and a suitable Gaussian beam distribution in 6 dimensions can be created and tracked through the BTS. In this work we track the evolution of the distribution along the BTS using MAD and the GPU code to perform a cross-check of accuracy.

4. Implementation of beam dynamics on the NVidia GeForce 7900 GPU

4.1 BrookGPU and the Kernel System

The choice of dedicated GPU compiler for this work was the general-purpose compiler from Stanford University, called BrookGPU [6]. This is a compiler and runtime implementation of the Brook stream program language for modern graphics hardware. Although based on standard ANSI C, Brook has a number of differences compared to its predecessor, although the general programming syntax is the same. Brook files are initially compiled with the Brook Runtime C Compiler (BRCC), producing an output of a C code. This code can then be run as normal within a developer environment, in this case Microsoft's Visual Studio.

Programming within Brook is based on a kernel system. Firstly, the function to be evaluated using the GPU is declared as a function within the code. The input data streams are also initialised as arrays within the C code, up to a size equal to the maximum texture capacity of the GPU. The NVIDIA 7900 chip has a maximum of 2048x2048 floating-point numbers. The streams are read into the kernel using the *streamread* function, and the operation contained within the kernel is carried out once on every element of the stream. This approach to stream programming comes with its own advantages and disadvantages, as a separate instruction can not be carried out on a single elements of a stream, but the lack of instruction calls generates a massive performance gain to the program. Once passed out of the kernel as an output stream, control is returned to the main C program.

The Brook files have certain limitations. The Brook header files can interfere with normal C definitions, producing unwanted output. For this reason the Brook kernel definitions were created in headers while the main body of the code was written in C++. Using a standard Visual studio project, the Brook header files containing the kernels could be compiled using a custom build step to call the BRCC, and the .h files could be included within the main code. This prevented any errors due to Brook/C compiler interaction, and spread the code for ease of use.

4.2 Matrix Representation within the Kernel

The transfer map evolution of beam particles through an accelerator, as described in section 3, can be readily cast into a stream processing form suitable for a GPU. Within the kernel, a limited number of operations can be carried out on all 2048x2048 input streams. Although these operations on each co-ordinate cannot reference or alter any other co-ordinate in the stream, due to the parallel nature of the process, they can reference the same co-ordinate in separate streams. The transfer map formalism can be set up by associating each of the input streams as one canonical variables of the particle. For example, if a, b denotes the horizontal canonical variable input stream (x, p_x) , the drift transfer map of length L would give an horizontal position output stream of $a + L.b$ When carried out within a kernel, this operation will be carried out for the 4 million particles in the stream.

A very common accelerator structure used in the design of particle accelerators is the FODO cell, made up of a focusing quadrupole magnet (F), followed by a drift (O), then a defocusing quadrupole (D), and finally another drift (O). The FODO cell was used in an initial study to assess the accuracy and speed of beam dynamics calculations on the GPU. Three Brook header files were set up, one for each of the elements in the FODO cell. The functions, containing the respective kernels, could then be accessed from the main code, and when called the particle streams would be read, the transfer map operations carried on the particles, and the result read back into the main code, in order to be used in by the following element's kernel.

4.3 Parsing the Input File and Creating the Beamline

The accelerator structure is specified using the XSIF standard [4], which is understood by the beam transport code implemented on the GPU, called GPMAD, and the comparison code MAD. The parser was based on the BDSIM [9] parser.

The parser scans through the XSIF accelerator definition file and populates a list of type element, a defined type containing the name, type and parameters of the element. The GPU kernel calls were implemented within an iterator over this list, with calls to the appropriate kernel, with the required variables depending on the type of the element.

5. Results

5.1 Overview of particle tracking results in DIAMOND

The GPMAD particle transport code is able to evolve particle phase space vectors along the DIAMOND beamline input file. The evolution of a single particle phase space point can now be computed using GPMAD and compared to the calculation of MAD, to ensure single particle trajectory benchmarking of GPMAD. For example, the horizontal displacement x as a function of distance s along the beamline demonstrates the accurate tracking of GPMAD. The analysis with higher numbers of particles can be run with increasing numbers of particles to obtain a speed comparison and show how the GPUs speed increases with higher arithmetic intensities. It was found that only the first order tracking was sufficient to give agreement between MAD and GPMAD.

5.1. Single Particle Tracking

Single particle analysis provides a crucial benchmark to confirm the accuracy and precision of GPU processing for particle beam dynamics. The GPU on the NVIDIA 7900 used in this work operates on the fp16 floating-point standard, whereas a CPU is fp32, IEEE 754 compliant. Rounding numbers on a GPU is also not compliant with the standard. This increases the need to perform a single particle benchmark.

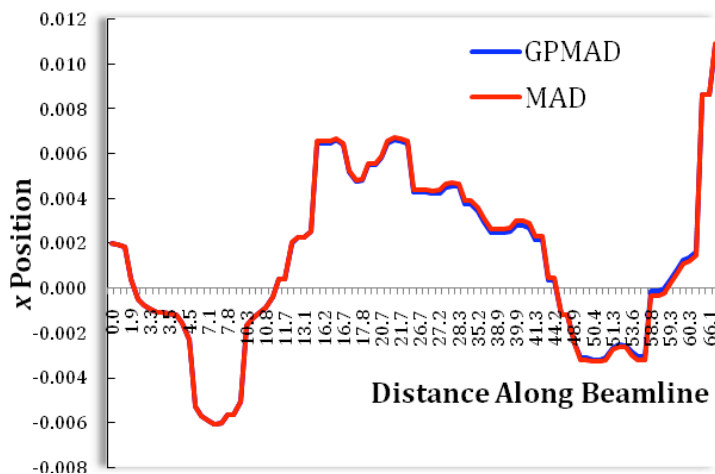


Figure 1: The evolution of the single particle phase space variable x along the DIAMOND transfer line with GPMAD and MAD.

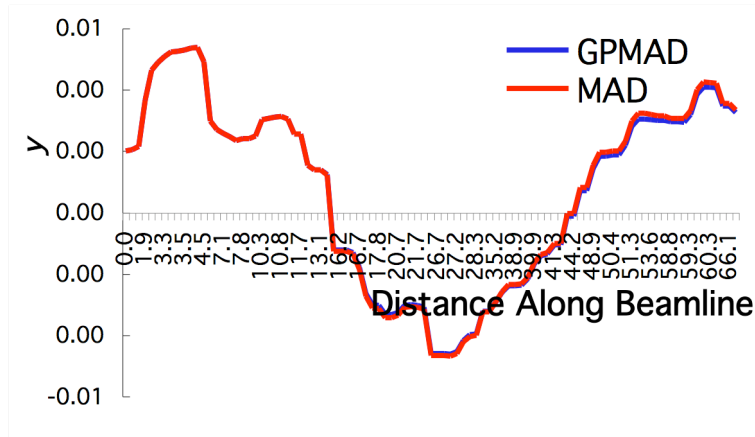


Figure 2: The evolution of the single particle phase space variable y along the DIAMOND transfer line with GPMAD and MAD.

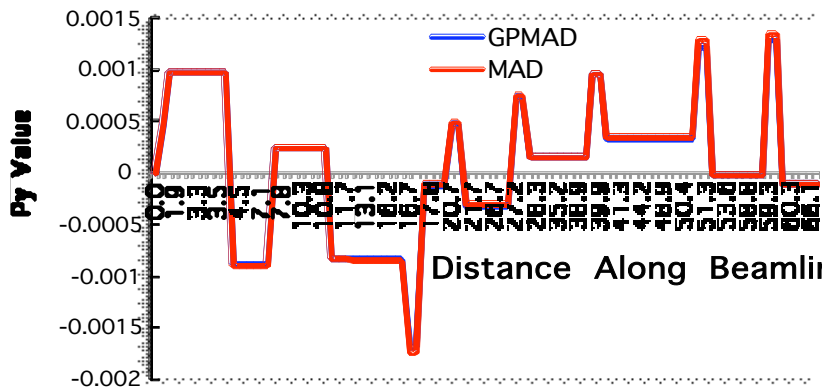


Figure 3: The evolution of the single particle phase space variable p_y along the DIAMOND transfer line with GPMAD and MAD.

Figures 1, 2 and 3 show the evolution of the particle phase space variables x , y and p_y respectively along the DIAMOND transfer line, for a single particle trajectory computed with the GPU (GPMAD) and with the CPU (MAD). The figures show the agreement between MAD and GPMAD is very good in the early stages of the beamline. This agreement is preserved for x and y until approximately 16.2 metres into the beamline. At this point, it appears that there is a very small, cumulatively propagating, error of position. At the end of the beamline, this results 0.39% disagreement between MAD and GPMAD in the position of x . This can be attributed to the absence of small second order terms in the GPMAD transfer map. The implementation of these terms will have a negligible effect on the GPU computation times. The evolution of p_y as calculated by GPMAD is identical to that of MAD, demonstrating that GPMAD retains satisfactory precision for beam dynamics. However, for a longer beamline, it may be necessary to have additional precision improvements to prevent any major cumulative effects.

5.2. Bunch Tracking and Tracking Accuracy

The same phase space distributions of 40,000 particles were tracked through the complete DIAMOND lattice using both GPMAD and MAD, to benchmark tracking accuracy for multiple particles. The GPMAD code can track up to 4,194,304 (2048 x 2048) particles, with the initial particle phase space distributions, shown on the left-hand side of figure 4, were drawn from a Gaussian function, with a zero first moment and a second moment related to the particle beam size at the entry of the transfer line. At the end of the beamline, the distribution of the data is Gaussian, as expected from elementary beam dynamics, and the MAD and GPMAD distributions are virtually identical, as shown in the right-hand side of figure 4. The mean of both distributions is 0.00815, and non-zero is due to the beam kicker in the beamline. The RMS value is slightly different, with an error of 0.32%. This difference arises for the same reasons as the single particle trajectory differences. Figure 5 compares GPMAD and MAD at a collimating positron HCOLL1 in the beamline, again showing agreement.

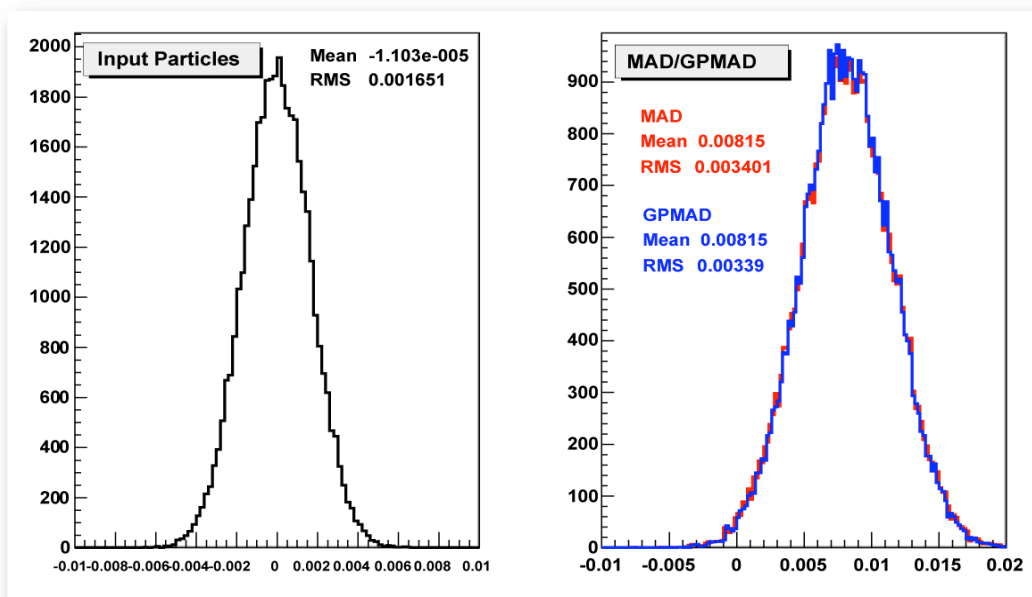


Figure 4: High statistics comparison between MAD and GPMAD for particle x-positions before (left) and after (right) transport through the complete DIAMOND lattice.

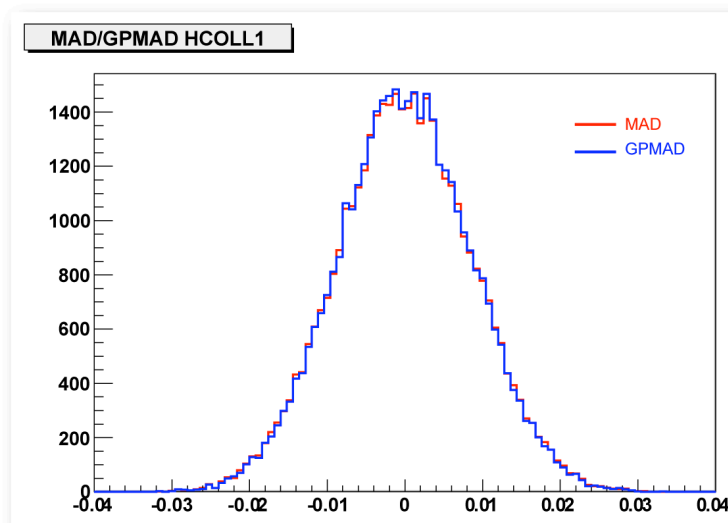


Figure 5: High statistics comparison between MAD and GPMAD, showing the Gaussian beam horizontal position distribution at the collimator HCOLL1 in the DIAMOND beamline.

5.3. High particle number beam tracking

The calculation of beam dynamics in particle accelerators has a high demand for systematic and statistical accuracy. This, coupled with the very large number of particles stored in an accelerator, results in a need to analysis the trajectories of many particles using the beam simulation tools. The computational time of the CPU-based code MAD Time is roughly linear scaling with number of particles. This is due to the serial nature of the processing. Figure 6 shows the time taken for the transport of varying number of particles along the DIAMOND transfer line, for MAD (CPU) and GPMAD (GPU). The linear scaling of MAD with particle number can be seen. The unexpected linear behaviour of GPMAD with particle number arises from non-GPU aspects of the program such as file handling, and communication between the C++ and Brook generated code, and the core GPU processing time is constant with particle number. This figure shows the large reduction in processing time gained with the GPU over the CPU when high numbers of particles are tracked. This performance gain is particularly significant at very large tracked particle number. Note that for small number of tracked particles, a ‘kernel overhead’ affects GPMAD, and this fixed kernel set-up time means the CPU tracking is faster than the GPU tracking. For example, when tracking 4 particles through the beamline, MAD takes 0.062 seconds and GPMAD takes 0.331 seconds. The transition between MAD and GPMAD occurs at approximately 10,000 particles, with GPMAD showing large performance gains above this number.

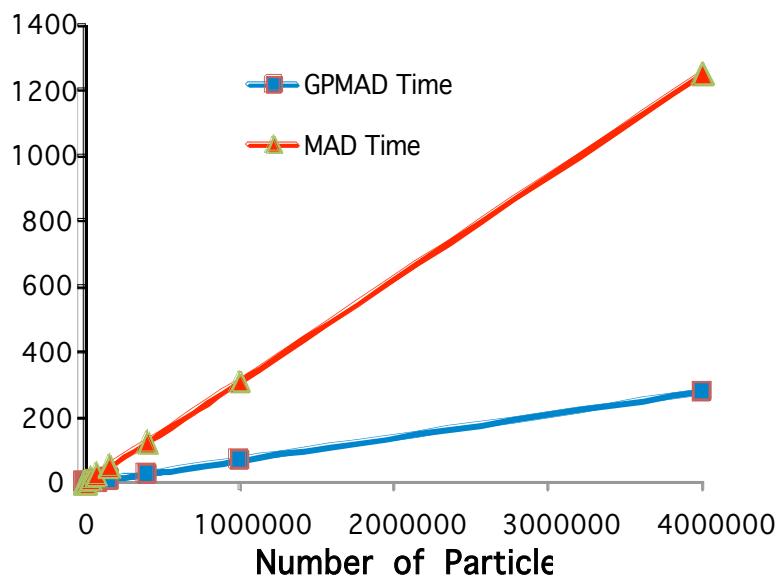


Figure 6: The processing time for tracking a variable number of particles along the DIAMOND lattice for the MAD and GPMAD codes.

6. Conclusions and further work

In this paper, the application of stream processing using GPUs was applied to the problem of particle beam transport in a particle accelerator. The particles were evolved down the DIAMOND accelerator transfer line using a research grade transport code MAD, and using a transport code designed to exploit the stream processing capabilities of a GPU, called GPMAD. Particle transport in an accelerator is particularly suited for this kind of processing. It was shown, by tracking single particles and bunches, the GPU-based code was in agreement with the CPU-based code and the performance

increases of the GPU-based code over the CPU-based code are significant. This paper therefore demonstrates the applicability of stream processing to particle tracking problems, and the possible speed, and hence accuracy, benefits.

On-going work is the completion of a second order transport code for ultra-relativistic particles, which fully exploits the current generation of GPUs, and inclusion of fully symplectic particle tracking. This can then be applied to a variety of accelerator physics problems, focusing on spin tracking and frequency map analysis in the short term. A further possible extension is to the regime of non-relativistic beams. On-going developments on the GPU hardware exploitation will be focused on the new GPU chips being developed. Of particular relevance to accelerator physics are higher numerical precision in kernels, lower kernel latencies and stream indexing. This last feature should allow application to problems inter-particle interactions. The use of multiple GPUs and custom hardware boards will also be explored.

In conclusion, this paper has shown the applicability of stream processing to particle tracking problems in accelerator physics, and the large performance gains achievable using GPU hardware. The on-going work should further develop this application and being to use GPUs to solve real world challenges.

References

- [1] See, for example, <http://www.research.att.com/~suresh/papers/mpds/mpds.pdf>
- [2] www.nvidia.com/
- [3] <http://www.diamond.ac.uk>
- [4] <http://www.cern.ch/mad>
- [5] <http://developer.nvidia.com/object/cuda.html>,
<http://ati.amd.com/technology/streamcomputing/index.html>
- [6] <http://graphics.stanford.edu/projects/brookgpu/>
- [7] K. L. Brown. A First-and Second-Order Matrix Theory for the Design of Beam Transport Systems and Charged Particle Spectrometers. SLAC 75, Revision 3, SLAC, 1972
- [8] J.K. Jones, Initial studies on Collimation in the BTS, AP-BTS-REP-008
- [9] Agapov *et al*, BDSIM - simulation toolkit based on GEANT4, EuroTeV report 2006-035