EUROTeV-Report-2008-022

Beam Dynamics using the Stream Processing Code GPMAD

v0.1

R.B. Appleby[*], D.S. Bailey[†], M.D. Salt[‡]

August 22, 2008

Abstract

GPU-Processed Methodical Accelerator Design (GPMAD) is a high-performance tool for studying beam dynamics within accelerators, using a parallelized stream processing approach. It is most suitable for large numbers of particles, with significant numerical intensity, providing optical calculations and beam tracking in second order in the transfer maps. GPMAD makes use of the huge computational power of modern commodity graphics cards to significantly increase performance over conventional CPU-based tracking codes. This report discusses the use of stream processing in accelerator physics, outlining the features and requirements of GPMAD as well as a guide to its use.

[*]Robert.Appleby@manchester.ac.uk The Cockcroft Institute and the University of Manchester, Oxford Road, Manchester, M13 9PL, UK

[†]David.Bailey@hep.manchester.ac.uk The University of Manchester, Oxford Road, Manchester, M13 9PL, UK

[‡]Michael.Salt@hep.manchester.ac.uk, The Cockcroft Institute and the University of Manchester, Oxford Road, Manchester, M13 9PL, UK

Contents

1 Introduction to Stream Processing

Charged particle tracking within accelerators can be computationally expensive, and the desire to understand a machine in simulation as fully as possible requires high-accuracy particle tracking. Whilst optical studies determine the overall beam behavior, particle tracking is necessary to determine the beam shape and extent within the beam-pipe, higher order beam behavior and resonances. Furthermore, high accuracy (low statistical errors) is particularly important for determining such phenomena as particle loss. Ideally, such simulations will require many millions a particles to be tracked through the lattice, often over many turns of a circular machine, and on a single computer these simulations can be particularly time consuming to perform. In an attempt to reduce processing time, it becomes necessary to purchase further computational hardware at great monetary expense. Particle tracking is ideally suited to a parallel architecture because it involves the same instruction being applied to every particle in simulation. In the relativistic approximation, there are no intra-beam forces and the motion of each particle in a beam is determined by the external forces; thus each particle is an independent unit which may be processed by parallel techniques.
However, a lot of current computational hardware already possesses a highly parallelized processor, the Graphics Processing Unit (GPU). Driven by a well-funded, but highly competitive consumer market, modern commodity graphics processors have evolved into very powerful computational engines. The demand for rendering on-screen textures at higher resolution and yet producing higher refresh rates has resulted in a highly parallel processor, with many computational cores. Whilst early GPUs were fixed-purpose devices, modern GPUs are flexible, multi purpose tools and this flexibility affords exploitation with the use of recently developed programming techniques. The GPU may be described as a large array of 'stream processors', which are the components of the die that are dedicated to computation. The rest of this section provides an introduction to stream processing techniques.

1.1 GPU as a Parallel-Execution Processor

The GPU is a highly parallel processor, with the latest examples having up to 256 stream processors [1]. The stream processors are Single-Instruction, Multiple Data (SIMD) devices which restricts their applications, but allows rapid manipulation of such highly parallel data. With GPUs specifically engineered for the purpose of rendering images, this potential was previously unaccesible. However, with the demand from the entertainment market for realistic in-game physics simulation, GPU manufacturers began to open up the architecture, providing the flexibility required to extract the computational power of the multiple stream processors for other applications. An organisation known as General Purpose GPU (GPGPU) [2] utilized this opportunity to apply the GPU to a much wider field, including finance [3], weather forecasting [4], monte-carlo simulation, cryptography [5] and scientific problem solving [6]. Such products of this research are Sh, Cg and Brook [7]. Brook was used to produce the proof of concept for GPMAD [8]. This made use of shader language techniques to define data streams that may be processed by the GPU. [1]

1.2 Performance

The performance gap between typical CPU and GPU processors is large, and future development seeks to widen this gap even further. The unit of Floating-Point Operations per Second (FLOPS) is used to measure this performance, with a typical dual-core CPU possessing a peak floating-point performance of approximately 50 Giga-FLOPS [10]. The latest generation of GPUs have peak performance in excess of 500 GFLOPS, with sustained real-world performance of up to 290 GFLOPS [10]. The key advantages of GPUs is the performance to price ratio compared to conventional systems, and that the hardware can be easily replaced should future solutions become available.

Table 1: Typical system performances relative to cost (in 2008)

System	GFLOPS	Price/£	GFLOPS per £
Intel Core 2 Duo workstation	50	$\approx$ 600	0.0833
+ nVidia 9800 GX2	768	600 + 370	0.7918
Cray XT4 HECToR	1000000	42500000	0.0235

Table 1 indicates that GPU-based computing solutions offer an attractive performance to price ratio compared to conventional workstations. Typically, sustained computational performance achievable in GPUs is half that of the peak performance, still being more than five times as cost effective as a CPU based solution. This is largely due to the competitive nature of the commodity hardware market, where both nVidia [11] and ATi [12] have to remain at the forefront of visual computing performance whilst undercutting the prices of each other to attract customers. This situation is to the benefit of the customers, whether it be for gaming or for computational purposes. Another advantage of GPU technology is high-speed memory. Currently, the fastest commercially available conventional memory is DDR3, with speeds of 800 - 1600 MHz. GPU memory is available up to 2000MHz, with high-bandwidth pipelines. GPUs are also scalable to a certain extent using twinning techniques such as CrossFire [13] from ATi, or SLi [14] from nVidia. This practically doubles the computational potential of the workstation.

The uptake in industry has been slow, despite the computational benefits. The first problem is that this only works for selected problems. These problems must be highly parallel, and not require huge amounts of memory. A 768 GFLOPS GPU solution will possess only one gigabyte of video RAM, whereas a CPU-based solution is likely to contain two gigabytes per processor, which would be thirty gigabytes in a 750 GFLOPS solution. The other reason is that programming on the GPU architecture is still very much in its infancy. There is not yet an established standard, which many software developers would prefer to wait for before adopting new techniques.

[1] Brook is currently under redevelopment to become Brook+ [9].

1.3 nVidia Compute-Unified Device Architecture

CUDA [15] from nVidia is currently the world's only C-language development environment for GPUs. It is the product of nVidia's realisation that GPUs could be used for High-Performance Computing. The architecture is designed to work with any of nVidia's GPUs upwards of the GeForce 8400, including mobile solutions and TESLA [16] (TESLA is NVidia's dedicated HPC hardware solution). Due to the current dominance of nVidia in the GPU field, it is likely that CUDA will become an unofficial standard in GPU programming. The major restriction is that it will only work with nVidia products, thus preventing it from being an offical standard. A minor point to note is that GPUs only support single-precision floating-point arithmetic, which may not be suitable for some tasks. Typically, these tasks are those with a low pre-defined computational intensity. Such examples are those where the code is multiply-branched, thus multiple-instruction, multiple-data (MIMD) data paths emerge. With a large proportion of the GPU die dedicated to computation, rather than control, MIMD execution would be very inefficient. The key limitation with GPUs is the 'kernel overhead' which arises from an initialisation delay when the kernel is called and is not insignificant. If the computation density is low per kernel call, the overhead can mask the performance benefits completely.
At the time of GPMAD's conception, CUDA support existed for only a selection of platforms. Windows XP with Microsoft Visual Studio appeared to offer the most direct route to CUDA programming. GPMAD was compiled using Microsoft Visual Express C++ 2008 and CUDA version 1.1. Both of which were available free of charge. The release version does not require Visial Studio to be used, but will require the CUDA toolkit to be installed. This is available from the nVidia CUDA Zone[15]

1.4 The Future of Stream Processing

nVidia provided CUDA with the assurance that it will be supported by future GPUs, under the 'Unified Device Architecture' driver system. CUDA will also support TESLA from nVidia, which is their High-Performance computing solution. This is available in three different forms, a single-GPU PCi expansion card, a deskside twin-GPU co-computer and a quad-GPU scalable rackmount server. Each GPU has been optimised for computation by removing VGA-specific components, and doubling the amount of RAM available to each. Being a dedicated solution, this will be considerably more expensive than commodity GPUs, but it will come with enhanced support, and will still yield a good GFLOPS/ ratio. Double-precision support has been promised for the near future.
In addition to CUDA, Brook+ is currently under development from AMD for ATi graphical products. This will re-ignite competition between nVidia and ATi, thus leading to superior hardware, available at yet lower cost to the consumer. AMD are in the process of releasing their high-performance computing, FireStream. This will also use Brook+, and double precision computation has been promised from the offset. Again, by challenging TESLA from nVidia, this can only serve to improve the GFLOPS/ ratio to the consumer.

2 Requirements, Obtaining and Installation

In this section, the requirements, and installation of GPMAD are discussed.

2.1 Requirements

GPMAD has been designed with both GPU and CPU implementation of the computational steps and, as a result, it should run on most modern machines, and can run in CPU-only mode for machines with no GPU. To make use of the GPMAD in GPU-mode, an nVidia CUDA-compatible processor is required. At the present time, the following processors are supported, as given in Table 2. Quadro is the range of nVidia professional GPUs, whereas GeForce is the commodity selection primarily used for entertainment purposes.

Table 2: NVidia Solutions Supported by GPMAD (GPU mode)

GeForce	TESLA	Quadro
9800 GX2	C870	FX 5600
9800 GTX	D870	FX 4600
9600 GT	S870	FX 3700
8800 Ultra		FX 1700
8800 GTX		FX 570
8800 GTS		FX370
8800 GT		NVS 290
8800GS		FX 3600M
8600GTS		FX 1600M
8600GT		FX 570M
8500GT		FX 360M
8400GS		Quadro Plex 1000 Model IV
8800M GTX		Quadro Plex 1000 Model S4
8800M GTS		NVS 320M
8700M GT		NVS 140M
8600M GT		NVS 135M
8600M GS		NVS 130M
8400M GT		
8400M GS		
8400M G		

2.2 Obtaining

GPMAD is intended for use on a variety of platforms, and can be obtained from www.hep.manchester.ac.uk/GPMAD , where there are appropriate installation packages for several systems. In addition, the CUDA Driver, Toolkit and Software Development kit are required and are available from www.nvidia.com/object/cuda_get.html.

2.3 Installation

The first step is to install the CUDA driver, toolkit and SDK. Under Windows XP or Vista, this requires downloading the executable files, running, and following the instructions. For all other installations, the nVidia instructions should be followed. Following successful installation, the GPMAD installation instructions should be followed for the relevant operating system.

3 Using GPMAD

The input file parser is based on a slightly modified implementation of the gmad parser used in BDSIM [17]. It is an XSIF standard parser which is also used within MAD 8 and MAD X [18]. However, minor modifications may be necessary to allow current input lattice descriptions to be used with GPMAD, due to small differences in grammer. GPMAD is case-sensitive, therefore it is necessary to define parameters in lower case, otherwise, they shall be assumed to take a value of zero. Being C++, each line ending should be terminated by a semicolon.

3.1 Input File Overview

3.1.1 Input particle file

The particles to be tracked through the lattice must be defined in a seperate file. This is a basic text file, with each row representing a single particle. Each column may be delimited by either a space or a tab. From left to right, the columns are ordered by the canonical variable set;

x	p_x	y	p_y	τ	p_t

where:
x = transverse horizontal position
p_x = transverse horizontal momentum
y = transverse vertical position
p_y = transverse vertical momentum
τ = time of flight relative to ideal reference particle
$p_t = \triangle E / p_s c$,

where $\triangle E$ is energy relative to reference particle, p_s is the nominal momentum of an on-energy particle and c is the velocity of light.

For example,:

```
0.00556 -0.756e-4 -0.00345 -0.00234 0.00465 0.00345
0.00546 -0.00765 0.00543 0.01567 -0.00657 0.00134
-0.00456 0.00876 -0.00564 0.556e-4 0.00765 0.00876
```

All particle parameters must be written in SI units, which may include base-10 exponentials in the form `0.00e0`. GPMAD will automatically count the number of particles in the file, and display this at the start of execution.

3.1.2 GPMAD Configuration File

Included with the distribution of GPMAD is a configuration file, *GPMADconf.txt*. The only functionally important part of this file is the first four integers. GPMAD was designed with certain flexibilities that are controlled by these variables and should this file be missing or otherwise damaged, GPMAD will assume the high-performance defaults, printing an appropriate warning. If the configuration file requires restoration, a replacement text file called *GPMADconf.txt* should be created, with four integer values, separated by a space according to Table 3.

Table 3: Parameters to set in GPMAD configuration file

a)Output to screen	set to 0 for off, 1 for on
b)Output every step	set to 0 for off, 1 for on
c)Order of precision	set to 1 for 1st order, 2 for 2nd order
d)GPU or CPU	set to 0 to use the GPU, 1 to use the CPU

Example: High performance defaults for `"GPMADconf.txt"`

```
0 0 2 0
```

3.1.3 Lattice File Structure

The GMAD parser is defined using the XSIF standard input. For example, to track a 3.0GeV electron through a F0D0 cell will require:

```
focusquad:quadrupole, l=1.5*m, k1=0.5;
defocusquad:quadrupole, l=1.5*m, k1=-0.5;
drift04:drift, l=0.4*m;
fodo:line=(focusquad,drift04,defocusquad,drift04);
use, period=fodo;
beam, energy = 3.0*GeV;
```

Notice the use of units when defining values. If the units are not given, SI units will be assumed. The user must first define the elements to be used in the beamline (using *drift*, *quadrupole* etc). From this, a beamline must be defined such as *line*. Using the command *use*, the beamline to be studied is selected. Beam parameters are set using the commands *beam* and *option*. To output the particle vectors at any point, use the *sample* command. For example, to output at the entrance to the defocussing quadrupole in the F0D0 cell, write;

```
sample, range=defocusquad;
```

If an element is used more than once in the beamline, disambiguate using square-bracket notation. For example, to sample at the second drift in the F0D0 lattice, write;

```
sample, range=drift04[2];
```

3.2 Physical Elements

GPMAD provides a basic set of physical elements to use in the beamline. Please note that SI units are assumed in element definition, unless overwritten with explicit units. Where a global variable has been set, an element with a conflicting value will override the global parameter. Such a condition would be the beam pipe radius, if an aperture is defined for an element, the beam pipe radius is ignored.

3.2.1 marker

This element has no effect on the particles or optics. It is a convenience element to identify a position along the line. It is often used with the *sampler* command.
Example:

```
pointofinterest :  marker;
```
...
sampler, range=pointofinterest;

3.2.2 drift

The drift element represents a length of accelerator in which there are no magnetic or electric forces acting.

• **l** - length [m]

• **aper** - aperture radius [m], will override the beampiperadius

Example:

```
drift04 :  drift, l=0.5*m;
drift02 :  drift, l=0.2*m, aper=0.3*m
```

3.2.3 sbend

The sbend element represents a sector dipole. GPMAD features the ability to model combined-function dipoles with a quadrupole component. It is also possible to define the pole face rotation relative to the beam direction.

• **l** - length [m]

• **angle** - bending angle [rad]

• **k1** - quadrupole field strength in combined-function dipole

• **e1** - pole face rotation at entry [rad] (default = 0)

• **e2** - pole face rotation at exit [rad] (default = 0)

• **aper** - aperture radius [m]

Example:

```
secdip003 :  sbend, l=0.5*m, k1=1.4, angle=0.03;
```

3.2.4 rbend

An rbend element represents a rectangular dipole magnet. This is modelled as a sector dipole with pole face rotations equal to half of the bending angle.

• **l** - length [m]

• **angle** - bending angle [rad]

• **k1** - quadrupole field strength in combined-function dipole

• **aper** - aperture radius [m]

Example:

```
recdip004 :  rbend, l=0.7*m, k1=0.0, angle=0.04;
```

3.2.5 quadrupole

The quadrupole element represents the region in which a quadrupole magnetic field is present.

• **l** - length [m]

• **k1** - normal quadrupole field strength (coefficient). Postive k1 means horizontally focussing

• **aper** - aperture radius [m]

Example:

```
quad0314 :  quadrupole, l=0.3*m, k1=1.4;
```

3.2.6 sextupole

The sextupole element represent a sextupole.

• **l** - length [m]

• **k2** - normal sextupole field strength

• **aper** - aperture radius [m]

Example:

```
sext0102 :  sextupole, l=0.1*m, k2=0.2;
```

3.2.7 kicker

A hkicker represents a horizontal kick, whereas a vkicker represents a vertical kick. Where a kicker is defined as having a finite length, it is modelled as a half-length drift, then a kicker, and then a second half-length drift.

• **angle** - the angle by which the particle is kicked in the appropriate direction

• **l** - length [m]

• **aper** aperture radius [m]

Example:

```
kickinx02003 :  hkick, l=0.2, angle = 0.03;
```

3.2.8 rcol

The rcol element describes a rectangular collimator. GPMAD assumes a hard-edged model in which particles that are considered lost if their spatial co-ordinates equal, or exceed the dimensions of the collimator. For tracking and optics, a collimator is treated as a drift, with collimation tested at both entrance and exit.

• **l** - length [m]

• **xsize** - horizontal width of collimator aperture [m]

• **ysize** - vertical height of collimator aperture [m]

Example:

```
col05002001 :  rcol, l=0.5*m, xsize=0.02*m, ysize=0.0*m1;
```

3.2.9 ecol

The ecol element is a eliptical collimator that utilises the same assumptions as that for the rcol.

• **l** - length [m]

• **xsize** - horizontal width of ellipse [m]

• **ysize** - vertical height of ellipse [m]

Example:

```
col03003004 :  ecol, l=0.3*m, xsize=0.03*m, ysize=0.04*m;
```

3.2.10 line

To define a beamline, an ordered list of the elements is required. The *line* element provides this functionality.

```
beamline01 :  line=(element1, element2,...);
```

Example:

```
focusquad:quadrupole, l=1.5*m, k1=0.5;
defocusquad:quadrupole, l=1.5*m, k1=-0.5;
drift04:drift, l=0.4*m;
fodo:line=(focusquad,drift04,defocusquad,drift04);
```

As an element, the line can be part of the beamline of another line:

```
beamline09 :  line=(fodo, fodo, fodo);
```

By using the BDSIM parser, other manipulations are available such as reversing the beamline:

```
beamlinebackwards :  line=(-beamlineforwards);
```

3.3 Beam and Option Parameters

Certain parameters within GPMAD are controlled via *option* and *beam* commands.

3.3.1 beam

It is necessary to define certain beam parameters for runtime. GPMAD requires the ratio of beam energy to charged particle mass (γ_{rel}). GPMAD will assume an electron at 1.0GeV, warning the user of this assumption.

beam, name=value,...;

Example:

```
beam, particle="e-", energy=3.0*GeV;
```

The particle may be either ”e-”, ”e+” or ”proton”. Any undefined values will assume the particle is an electron.

3.3.2 option

The *option* command may be used to set various parameters.

option, name=value,...;

Example:

```
option, beampipeRadius=0.03*m
```

This would set the beampipe radius to be 30mm. At the end of each element in the beamline, GPMAD uses collimation testing to record particles lost to the beampipe wall.

Example:

```
option, betx=12.1345, alfx=-2.9196, bety=2.94009, alfy=0.748352;
```

This sets the initial conditions of the optical functions at the start of the beamline. These are used to produce the values of the optical functions at each point in the beamline.

3.4 Running GPMAD

To run, the GPMAD executable, lattice file and particle input file should be in the same directory. GPMAD uses command-line arguements to select the particle and lattice files. At the command line, type:

```
→ GPMAD.exe <XSIF input> <Particle Data Input>
```

Should the command arguements not be present, GPMAD will display an error, instructing the proper arguement layout. Once these arguements have been satisfied, GPMAD will run. GPMAD displays the bare minimum of screen output for perfomance reasons. Should this information be required for debugging processes, the 'Output to screen' variable in the GPMAD configuration file should be set to '1'.

3.5 Output File Overview

GPMAD will produce four output files. GPMAD has been designed such that the output files will contain the name of the input lattice file used. This makes it easier for the user to identify which output belongs to each input.

3.5.1 OUTPUT_OF_<XSIF input>

The output of this file depends on the value 'Output every step' in the GPMADconfig.txt file. If this value is set to '0', this file will only contain particle data at the start, end and sampler positions. If the 'Output Every Step' integer is set to '1', this file will contain particle data at every element in the beamline. The former is the high-performance default, the latter is useful if the particle distribution is required to be known at every point. The output contains the name of the element, and then lists the all of the particles using the same format as the particle input file. The particle data is based upon the values at the exit of the element.

Example:

```
BFQUD:
0.00556 -0.756e-4 -0.00345 -0.00234 0.00465 0.00345
0.00546 -0.00765 0.00543 0.01567 -0.00657 0.00134
-0.00456 0.00876 -0.00564 0.556e-4 0.00765 0.00876

DFT180:
0.00556 -0.00665 0.00843 0.00527 -0.01647 0.00134
...
```

3.5.2 OUTPUTDATA_OF_<XSIF input>

For every run, GPMAD will output this file. It contains particle data for the first particle in the particle input data. Each column represents s, x, p_x, y, p_y, τ and p_t respectively, where s is the longitudinal distance along the latttice. Each row represents the particle position at the exit of each element in beamline. The purpose of this is mainly diagnostic, to plot the trajectory along the beamline.

Example:

```
1.06 0.00556 -0.756e-4 -0.00345 -0.00234 0.00465 0.00345
1.67 0.00546 -0.00765 0.00543 0.01567 -0.00657 0.00134
2.35 -0.00456 0.00876 -0.00564 0.556e-4 0.00765 0.00876
```

3.5.3 OPTICS_OF_<XSIF input>

To be able to obtain plots of the optical functions in GPMAD, the user must first define α_x, α_y, β_x and β_y. If these are not defined, they are defaulted to zero, and therefore the results will become non-physical. Defining the optical functions at the beginning of the lattice is detailed in 3.3.2, γ_x and γ_y are calculated from these values.
The output file will contain the values of α_x, α_y, β_x, β_y, γ_x and γ_y. In addition, the first-order transfer matrix is displayed laid out as in equation 1. This is the overall matrix between the start of the beamline, and the exit at that particular element. This is helpful in diagnosing problems within the lattice.

Example:

```
BFQUD s = 0.17 length = 0.17 K1 = 1.40278

0.979798 0.168854 0 0 0 0
-0.236864 0.979798 0 0 0 0
0 0 1.02034 0.171151 0 0
0 0 0.240086 1.02034 0 0
0 0 0 0 1 0.0188889
```

```
0 0 0 0 0 1

Alpha(x) = 0.000247588 Beta(x) = 12.6376 Gamma(x) = 0.0791289
Alpha(y) = -0.00303968 Beta(y) = 2.81507 Gamma(y) = 0.355234
```

3.5.4 LOSTPARTICLEDATA_OF_<XSIF input>

GPMAD uses a hard-edged collimation model for both collimators, and the surface of the beam pipe. If the spatial co-ordinates of a particle exceed the confines of the containment vessel, it is recorded as a lost particle in this file. Contained within is a list of the names of the elements, the length along the beam pipe, the particle positions at which they were declared lost, and a tally of the loss. For beampipe losses, testing happens at the end of the element. For dedicated collimators, testing takes place at both entrance and exit.

Example:

```
Particles lost at DFT180 ENTRANCE, S = 1.06:
-0.00222907 -0.000716004 -0.000573608 0.000181213 -0.00212511 -0.00151551
Total lost at this element = 1
Particles lost at DFT180 EXIT, S = 1.30:
Total lost at this element = 0
Particles lost at DFT180, S = 1.30:
Total lost at this element = 0
```

4 GPMAD and the Second-Order TRANSPORT Maps

GPMAD uses the same TRANSPORT [20] mapping system as used in MAD. This consists of a 6 x 6 matrix to represent the first-order terms, known as the the R-matrix, and second order effects known as the T-terms.

4.1 Production of GPMAD

GPMAD is mainly written in C++, with CUDA implementations for regions of high computational density. This permits uncomplicated interface with the many C++ oriented programs available in physics, including the data analysis package, ROOT. One point of note is that CUDA is based on non-object oriented C, not C++. As a result, GPMAD is typically not object oriented. Later releases of GPMAD will feature enhanced use of object orientation.
GPMAD is comprised of two main components, the parser and the program itself. The parser is a slight modification of that used in BDSIM [17], wrapped up in as a library. The program itself extracts the appropriate information from the parser, calls the relevant functions, calculates the TRANSPORT parameters required, calls the CUDA implementation for the computation and handles any output required.

4.2 TRANSPORT maps

The motion of a charged particle is modelled by a six-dimensional vector, using the following canonical variables, as defined in Section 3.1.1;

$$X = \begin{pmatrix} x \\ p_x \\ y \\ p_y \\ \tau \\ p_t \end{pmatrix}$$

To first-order approximation, the motion may be modelled as a matrix-vector operation per magnetic element, the R-matrix.

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} & R_{14} & R_{15} & R_{16} \\ R_{21} & R_{22} & R_{23} & R_{24} & R_{25} & R_{26} \\ R_{31} & R_{32} & R_{33} & R_{34} & R_{35} & R_{36} \\ R_{41} & R_{42} & R_{43} & R_{44} & R_{45} & R_{46} \\ R_{51} & R_{52} & R_{53} & R_{54} & R_{55} & R_{56} \\ R_{61} & R_{62} & R_{63} & R_{64} & R_{65} & R_{66} \end{pmatrix} \tag{1}$$

This particular example is of a drift element (no magnetic field present). L is the length of the drift space, and β_s and γ_s are the usual relativistic factors;

$$X_f = \begin{pmatrix} 1 & L & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{L}{\beta_s^2\gamma_s^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \times X_i$$

The complete TRANSPORT map is modelled including second order T_{ijk} terms;

$$Z_j = \triangle X_j + \sum_{k=1}^{6} R_{jk} X_k + \sum_{k=1}^{6} \sum_{l=1}^{6} T_{jkl} X_k X_l$$

,
where $\triangle X_j$ gives a beam centroid shift. The TRANSPORT system assumes the charged particles to not interact with each other in the ultra-relativistic limit. Therefore, the same operation is applied to each particle independently. As the number of particles is increased, the operation becomes highly parallel, a task well suited to the parallel architecture of the GPU.

4.3 R-matrix performance enhancements

For the current set of elements available in GPMAD, many of the values in the R-matrix are always zero. Standard matrix-vector techniques would implement the multiplication by zero, even though it is redundant. As a result, the GPMAD kernel only contains the relevant matrix-vector terms, with other omitted. As GPMAD becomes more general-purpose, this specificity will be lost, and the full matrix-vector multiplication will be required. This can be provided by the CUBLAS library within CUDA, and will be available in future releases.

4.4 T-term definition different to MAD

For performance reasons, GPMAD uses symmetry to reduce the number of terms of T_{xyz}

$$T_{xyz}^{GPMAD} = T_{xyz}^{MAD} \tag{2}$$

where y = z and;

$$T_{xyz}^{GPMAD} = 2 \times T_{xyz}^{MAD} \tag{3}$$

where y $\neq$ z. As such, there are no terms T_{xyz} in GPMAD where y > z.

4.5 R-matrix element definitions

4.5.1 drift

$$\begin{pmatrix} x \\ p_x \\ y \\ p_y \\ \tau \\ p_t \end{pmatrix} = \begin{pmatrix} 1 & L & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{L}{\beta_s^2\gamma_s^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x^0 \\ p_x^0 \\ y^0 \\ p_y^0 \\ \tau^0 \\ p_t^0 \end{pmatrix}$$

where L is the length of the element.

4.5.2 sbend/rbend

In GPMAD, the dipole is modelled by three matrices, one dipole matrix sandwiched between an entrance and exit matrix to represent edge-focusing effects. Edge focusing effects at both the entrance and exit are modelled by:

$$\begin{pmatrix} x \\ p_x \\ y \\ p_y \\ \tau \\ p_t \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ +h\tan\psi_i & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -h\tan\psi_i & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x^0 \\ p_x^0 \\ y^0 \\ p_y^0 \\ \tau^0 \\ p_t^0 \end{pmatrix}$$

Where ψ is the pole face rotation. For an rbend, this is treated as half of the bending angle. For the body of the dipole. The value of k_x and k_y is defined as;

$$k_x = \sqrt{K_1 + \left(\frac{\theta}{L}\right)^2} \tag{4}$$

$$k_y = \sqrt{-K_1} \tag{5}$$

where K_1 is the quadrupole co-efficient of the combined function component of the dipole and θ is the bending angle. The dipole R-matrix is given by;

$$\begin{pmatrix}
\cos(k_x L) & \frac{\sin(k_x L)}{k_x} & 0 & 0 & 0 & \frac{h}{\beta_s} \times \frac{1-\cos(k_x L)}{k_x^2} \\
-k_x \sin(k_x L) & \cos(k_x L) & 0 & 0 & 0 & \frac{h}{\beta_s} \times \frac{\sin(k_x)}{k_x} \\
0 & 0 & \cos(k_y L) & \frac{\sin(k_y L)}{k_y} & 0 & 0 \\
0 & 0 & -k_y \sin(k_y L) & \cos(k_y L) & 0 & 0 \\
-\frac{h}{\beta_s} \times \frac{\sin(k_x)}{k_x} & -\frac{h}{\beta_s} \times \frac{1-\cos(k_x L)}{k_x^2} & 0 & 0 & 1 & \frac{L}{\beta_s^2 \gamma_s^2} - \frac{h^2}{\beta_s^2} \frac{L - \frac{\sin(k_x L)}{k_x}}{k_x^2} \\
0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

It is possible that either k_x or k_y could be imaginary, in which case the hyperbolic identities are used

$$\cos(kL) \equiv \cosh(ikl) \tag{6}$$

$$\frac{\sin(kL)}{kL} \equiv \frac{\sinh(ikL)}{ik} \tag{7}$$

4.5.3 quadrupole

$$k_x^2 = K_1 = -k_y^2 \tag{8}$$

where K1 is the quadrupole coefficient, the quadrupole is given by;

$$\begin{pmatrix}
\cos(k_x L) & \frac{\sin(k_x L)}{k_x} & 0 & 0 & 0 & 0 \\
-k_x \sin(k_x L) & \cos(k_x L) & 0 & 0 & 0 & 0 \\
0 & 0 & \cos(k_y L) & \frac{\sin(k_y L)}{k_y} & 0 & 0 \\
0 & 0 & -k_y \sin(k_y L) & \cos(k_y L) & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & \frac{L}{\beta_s^2 \gamma_s^2} \\
0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

4.5.4 sextupole

To first-order, the sextupole is identical to the R-matrix for drift. Second order terms are the important components of the sextupole.

4.6 Second Order T terms

These have been derived precisely from those given in the MAD 8 physics manual. In parts of the MAD Physics Manual, the TRANSPORT map is given as a complete equation, including both first-order and second-order terms. The second order (T) terms have been derived from expansion of these equations. This is an important diagnostic tool of GPMAD, permitting the user to run in at first-order approximation only. For example,

$$T_{211} = -\frac{1}{6}(K_2 + 2hK_1)\frac{sin(k_xL)}{k_x}(1 + 2cos(k_xL)), \tag{9}$$

gives the second order term coupling x^2 in the initial state to p_x in the final state, where K_2 is the sextupole coefficient, h the curvature, K_1 the quadrupole coefficient, L the element length and $k_x = \sqrt{K_1}$.

4.7 Zero-order elements

Both hkicker and vkicker are modelled as zeroth order terms that change the transverse momentum by a fixed amount.

$$p_{x_{or}y} = p^0_{x_{or}y} + \theta \tag{10}$$

5 Case Study - The DIAMOND BTS Lattice

The DIAMOND light source in Oxford [21], United Kingdom, provided an opportunity to study GPMAD against known results. In particular, the 68.04 metre booster to storage-ring (BTS) lattice provided a source of study to compare GPMAD against the conventional computing code, MAD. This particular section has been well studied due to the requirement for small emittance for injection into the storage ring.

5.1 Introduction to the BTS

The DIAMOND BTS lattice consists of drift, sector bend, quadrupole, kicker and collimator elements. None of these elements have an associated skew value, which is not covered in this first release of GPMAD. The lattice was already available in MAD format for study, requiring only minor modifications to be compatible with GPMAD. Of particular importance was to determine that GPMAD lost very little in terms of accuracy compared to MAD. GPUs are single-precision, but due to small deviations, not fully IEEE 754 compatible. The accuracy was determined using single-particle tracking and ensuring that particle distribution characteristics were maintained. The performance enhancements were also measured. Similar results were presented at EPAC [6].

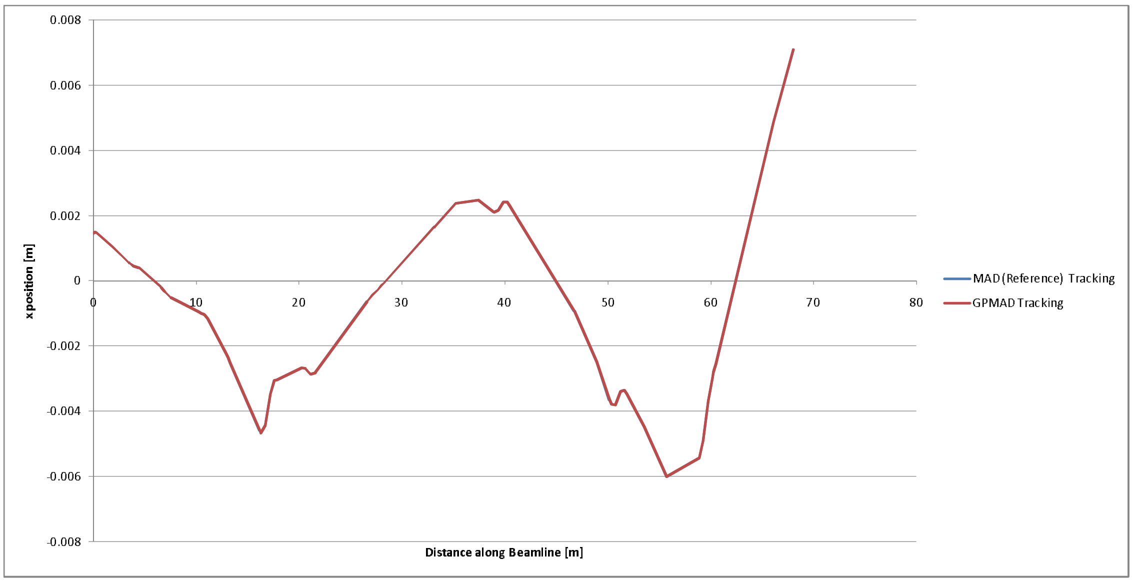

Figure 1:
Single-Particle, Accuracy-Verifying Tracking in x

5.2 Results

5.2.1 Single Particle Tracking

In Figure 1, the plot of the horizontal displacement is given for the length of the beamline for both GPMAD and MAD, each with the same starting particle for simulation. Both these plots are overlaid onto the same axes to allow for comparison. It is clear from the the figure that the trajectories computed in MAD and GPMAD agree to a very high accuracy.

Figure 3 is a quantitative evaluation of the deviations due to GPU processing. In this particular example, the deviation from MAD;

$$\frac{\triangle p_y}{p_y} = \frac{p_y^{MAD} - p_y^{GPMAD}}{p_y^{MAD}} \tag{11}$$

is plotted across the beamline. Throughout all the 94 magnetic elements, the deviation is kept to less than 0.025%, which is small. Note also that the deviation is largely oscillatory about the origin indicating the deviations are not cumulative, and to a large extent will cancel with each other. This is particularly important for long-term stability in lengthy beamlines. This deviation is most likely due to the non-standard floating-point precision of the GPU.

5.2.2 High Statistic Tracking

In the particle accelerator, a distribution of particles at input is generally Gaussian in form. The particle bunch retains a largely Gaussian distribution for the length of the

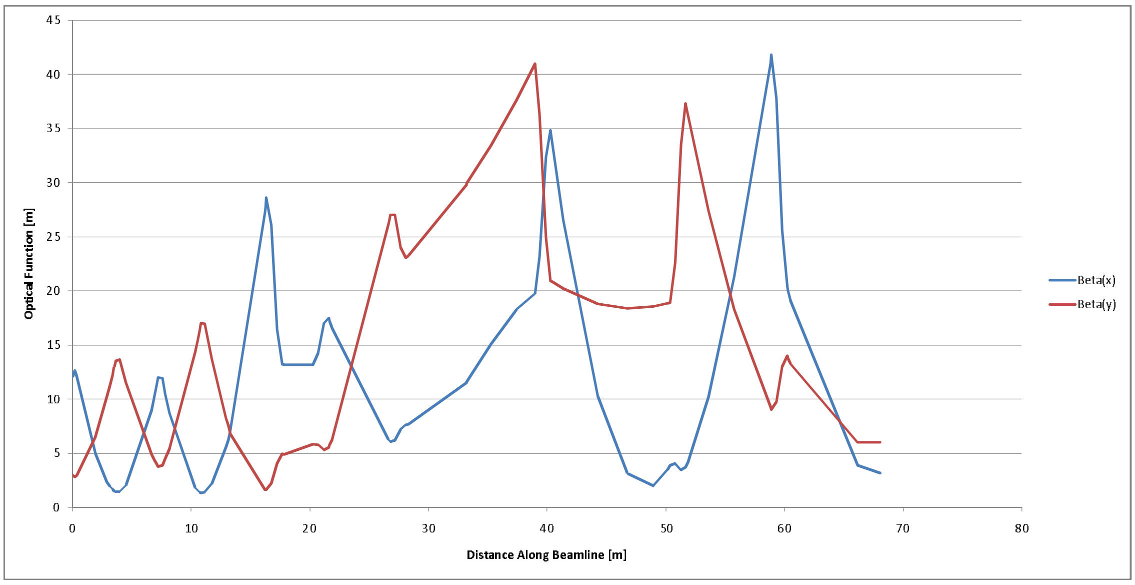

Figure 2:
Beta Function evolution through the BTS Lattice

beamline under study. To confirm that high-statistic tracking is acceptably accurate, 64000 particles were evolved through the BTS lattice. Both codes were run, and the distribution calculated at points of interest.

As is evident in Figure 4, the distributions produced by both MAD and GPMAD are practically identical. The calculated mean of x for GPMAD is 0.008173096 whereas the mean for MAD is 0.008173098. This shows that to within approximately 2.44×10^{-5} %, GPMAD is suitable for calculating the mean position of the beam. The standard deviation from GPMAD is 0.3440153 compared with 0.3440155 in MAD, thus confirming that the high-statistic results of GPMAD agree to those of MAD to within 6×10^{-5} %

5.2.3 Performance Gains

The reason for using GPMAD over other codes is for the reduction in processing time for a given job. GPMAD has been tested up to 8,192,000 particles in a single job, whereas other codes such as MAD and DIMAD, would require this to be processed in multiple jobs. For this many particles, GPMAD was shown to be using approximately 150 MB of main system memory and, given the latest graphics cards are available with up to 1 GB of memory, equates to a limit around 24,000,000 particles for these cards. This makes GPMAD a particularly efficient code where batch numbers are large. As well as this GPMAD uses the high-performance GPU processor for the intensive linear algebra. The benchmarking was performed on a laptop containing an Intel Core 2 Duo 1.5 GHz processor and an nVidia 8600m GT GPU (chosen for this case study to represent a typical desktop computer in 2008).

Figure 5 demonstrates the reduction in processing time achieveable using GPMAD. At low particle numbers, the performance of GPMAD is drastically reduced. For very small

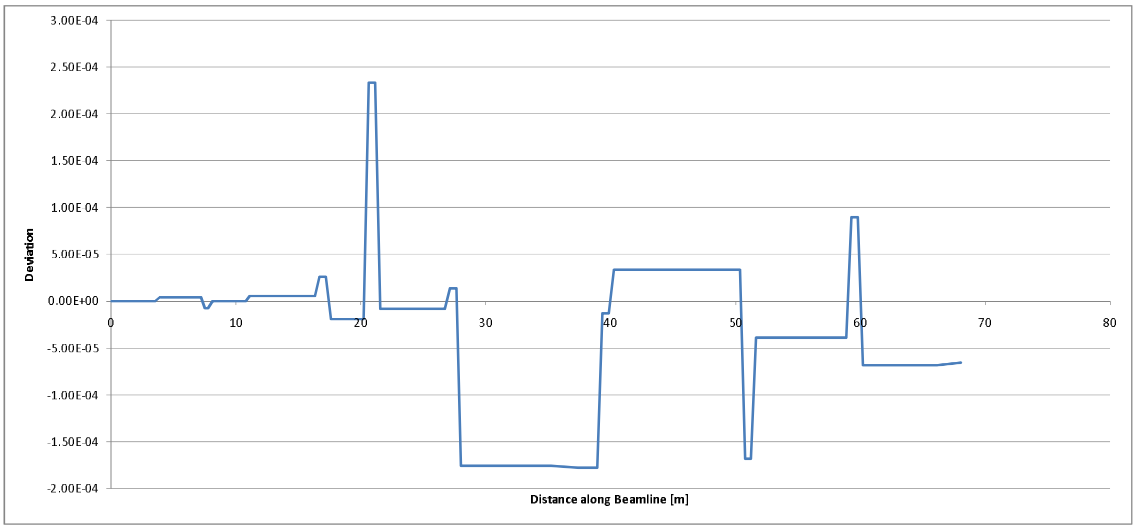

Figure 3:
Deviation, $\frac{\triangle p_y}{p_y}$ along beamline relative to MAD

batches, MAD would be the faster choice. This is due to a feature of GPU Processing known as the 'kernel overhead'. The GPU has a fixed setup time per kernel call. As a result, this delay eclipses the performance gains. When using small batches, it is recommended to alter the configuration file to process using the CPU.
At large batch sizes, upwards of 10,000 particles, the performance gains are clearly evident. For example, at 4,096,000 particles, GPMAD completes the given task in 5 minutes, 38 seconds. The same task takes 22 minutes, 27 seconds. Profiling showed that much of the time was spent reading-in, and writing-out particle data files. The fraction of input and output time, relative to the whole job time, would be much reduced if a longer beamline were to be used. In terms of performance, GPMAD is best used with high particle numbers, and long beamlines.

6 Conclusion

Based upon the theory, and the case study of GPMAD, it is clear that stream processing is an attractive computational option. Commodity GPUs offer excellent performance to price ratio, which may be utilised to significantly reduce simulation times. Many personal computers and laptops are shipped with a dedicated GPU, so it is possible that this technology is available at no further cost to the user. The important points to note is that whilst GPUs are increasing in performance, the price remains stable. Therefore, in the future, GPMAD will take advantage of a yet greater GFLOPS per £ratio, thus significantly reducing the cost of simulation.

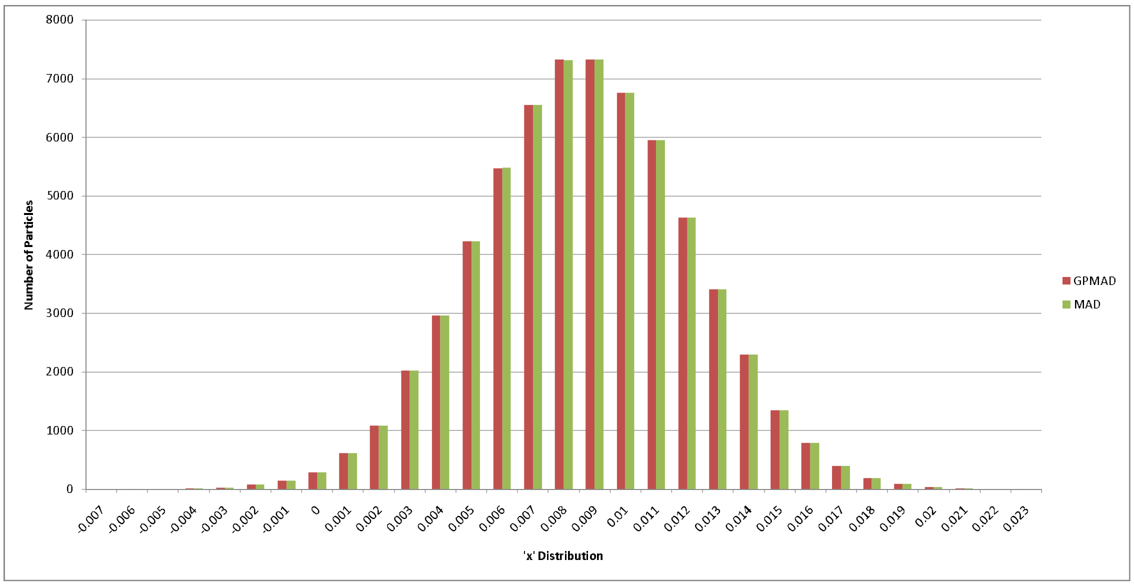

Figure 4:
Distribution of x at the end of the BTS lattice

6.1 Accuracy and Precision

Industry standard codes use at least IEEE 754 [19] floating-point precision, some may even offer double-precision. Commodity GPUs are single-precision devices, but are not truly IEEE 754 compliant. There are minor deviations from the standard, such as rounding rules and exception handling of undefined numbers. As a result, GPMAD may exhibit minor deviations to tracking codes (such as MAD) when used with GPUs. TESLA solutions are considered IEEE 754 compliant, and thus, GPMAD run on a TESLA device may also be considered to be compliant. The accuracy data and plots presented in this paper indicate high-statistic tracking variation in the 6th figure of significance.

6.2 Performance

GPMAD was demonstrated using an nVidia GeForce 8600m GT, which contains 16 stream processors. Performance of this processor is rated at less than 300 GFLOPS. GPUs are now available with up to 256 stream processors, and a peak performance of 768 GFLOPS. This performance is available for less than £400 at the time of writing. nVidia TESLA solutions offer up to 500 GFLOPS per GPU. With the fast pace of the entertainment market, new generation GPUs are getting faster, and yet still available at reasonable cost.

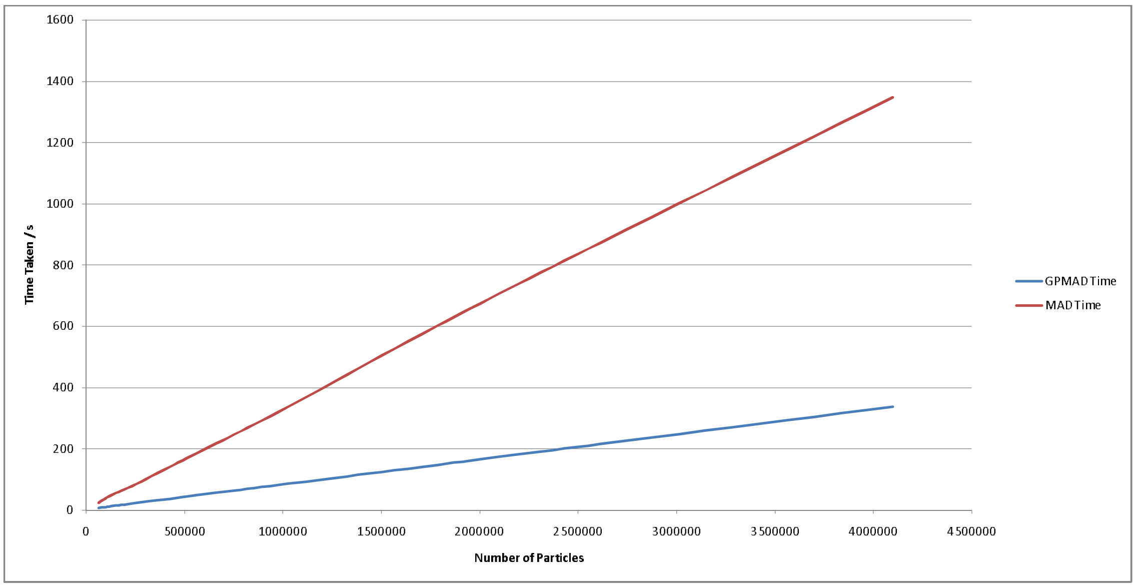

Figure 5:
Time taken to evolve particles through the BTS lattice

6.3 The Future of GPMAD

The current version of GPMAD (v0.1) includes a basic set of magnetic elements for simulation purposes. A basic feature set is provided with this release. The intention is to evolve GPMAD gradually, ensuring backwards compatibility with previous versions where possible. Much has been learned about stream processing in producing GPMAD, and this knowledge will be used to re-write later versions.

6.3.1 Symplectic Mapping

GPMAD provides the user with the TRANSPORT mapping system. This is acceptable for modelling specific sections, or linear machines. For many-turn circular machines, the symplectic mapping system is required. In later versions of GPMAD, symplectic mapping will be included using Lie algebraic maps.

6.3.2 Memory-Model Performance Enhancements

In the current guise, GPMAD uses C++ for all the control steps. Through each magnetic element, particle parameters are copied from main memory to GPU memory, processed, and then returned to main memory. This is a particularly slow process due to memory latency. To improve this in later versions, the magnetic elements will be processed in advance of any kernel operations, then passed as a list to the kernel. The kernel can then process many magnetic elements in one call, rather than one call per magnetic element. Should a sampler be required, provisions can be put into place to output a copy of GPU memory to main memory for output to file. GPU memory may also use assyncronous timing to disguise memory latency.

6.3.3 Object Orientation

CUDA is based on C, and therefore is not object-oriented by default. As a result, and to ensure smooth integration with CUDA, GPMAD makes only minimal use of objects. Now that the interface between CUDA and C++ is better understood, it will be possible to make better use of objects. This will improve the modularity of later GPMAD versions, should any additional features be required.

6.3.4 Magnetic Elements

The range of magnetic elements in GPMAD will be expanded. These will require a more general kernel for processing purposes.

6.3.5 Double-Precision Support

The next generation of HPC stream processors are described as double-precision devices. When these become available, GPMAD will be converted to double-precision.

References

[1] *http://www.nvidia.com/object/geforce_9800gx2.html* accessed 08/05/2008

[2] M.J.Harris, G.Coombe, T.Scheuermann and A.Lastra *Physically-Based Visual Simulation on Graphics Hardware*

[3] SciComp Inc. *http://www.scicomp.com/parallel_computing/GPU_OpenMP*

[4] J.Michalakes and M.Vachharajani (National Center for Atmospheric Research) *GPU Acceleration of Numerical Weather Prediction*

[5] Revolutionary Technique to Recover Lost Passwords Quickly *http://www.elcomsoft.com/EDPR/gpu_en.pdf*

[6] M.D.Salt, R.B.Appleby, D.S.Bailey, *Beam Dynamics using Graphical Processing Units* - EPAC08 - TUPP085

[7] Stanford Graphics Laboratories *http://graphics.stanford.edu/projects/brookgpu/*

[8] R.B.Appleby, D.S.Bailey, J.Higham and M.D.Salt *High-Performance Stream Computing for Particle Beam Transport Simulations* - Computing for High Energy Physics Conference 2007, 66

[9] Brook+ *http://ati.amd.com/technology/streamcomputing/AMD-Brookplus.pdf* accessed 08/05/2008

[10] D. Luebke *GPU Computing: the Democratization of Parallel Computing* - http://www.gpgpu.org/asplos2008/ASPLOS08-1-intro-overview.pdf

[11] nVidia *www.nvidia.com*

[12] ATi *ati.amd.com*

[13] ATi Crossfire Technology *crossfire.ati.com*

[14] nVidia SLi Technology *sli.nvidia.com*

[15] Compute Unified Device Architecture *www.nvidia.com/object/cuda_home.html*

[16] TESLA *www.nvidia.com/object/tesla_computing_solutions.html*

[17] I.Agapov, G.Blair, J.Carter, O.Dadoun *The BDSIM Toolkit* - EUROTeV-Report-2006-014-1

[18] Methodical Accelerator Design *mad.cern.ch*

[19] Institute of Electrical and Electronics Engineers *www.ieee.org*

[20] K.Brown, *A First- and Second-Order Matrix Theory for the Design of Beam Transport Systems and Charged Particle Spectrometers* - SLAC-75

[21] DIAMOND Light Source *www.diamond.ac.uk*