ACCELERATION OF SYMPLECTIC INTEGRATOR WITH GRAPHICAL PROCESSING UNITS

J. Rowland, I. P. Martin, Diamond Light Source, UK

Abstract

The study of non-linear effects in storage rings requires massively parallel particle tracking over a range of initial conditions. Stream processing architectures trade cache size for greatly increased floating point throughput in the case of regular memory access patterns. The symplectic integrator of Tracy-II [1] has been implemented in CUDA [2] on the nVidia stream processor and used to calculate dynamic apertures and frequency maps for the Diamond low-alpha lattice. To facilitate integration with existing workflows the lattice description of Accelerator Toolbox [3] is re-used. The new code is demonstrated to achieve a two orders of magnitude increase in tracking speed over a single CPU core and benchmarks of the performance and accuracy against other codes are presented.

INTRODUCTION

The Graphics Processing Unit (GPU) has previously been demonstrated as a useful tool for single particle dynamics despite the single precision hardware [4]. The nVidia GPU is a stream processing architecture which differs primarily from CPUs in the optimization of memory access patterns. In short, cache space is traded for extra floating point units and registers which enforces a regular access pattern to achieve good performance. The main and serious disadvantage of this architecture is the lack of double-precision floating point capability. Also the memory bandwidth while good is unable to compete with the combined bandwidth of a cluster of processors. The most recent generation of GPUs have a few double-precision units for use as accumulators but the bulk of the computation must be done in single precision to fully utilize the hardware. The GPU is also a multi-threaded architecture, with each processor running a large number of threads similarly to the hyper-threading of recent Intel processors. Fast context switches between threads hide memory latency, while the first thread is waiting for memory access to complete the second thread is scheduled and can perform a calculation. There should be at least as many threads as required to hide all the memory latency in the code to achieve maximum throughput. A reasonable number of parallel tasks is 10000 (100 processors, 100 threads per processor). Single particle dynamics is a good fit for this architecture as the phase space co-ordinates for each particle are held at a fixed stride in memory and there are no complex data structures, and parallel tracking for frequency map analysis and dynamic aperture studies provides the motivation for using a large number of particles.

While techniques such as Kahan summation or multi-precision arithmetic would allow more accurate results this paper takes the path of least resistance and considers what can be achieved with a small amount of effort using the device in the most efficient way, and whether the result is useful as is.

SOFTWARE

nVidia provide a C++ compiler with some extra syntax for the GPU. A program is compiled into two parts, one runs on the CPU and one runs on the GPU. The output is the usual binary for the system (Windows and Linux are supported) containing both machine codes. The important operations on the host machine are allocating buffers on the GPU, direct memory access transfer from the CPU to the GPU and back, and invoking calculation 'kernels' on the GPU. This operation is similar to the parallel for loop available in some parallel programming systems such as OpenMP. In the case of parallel tracking the grid of particles is distributed with one thread per particle. As each kernel invocation must return within 5ms (the screen also freezes when the GPU is busy) the CPU must repeatedly initiate a few hundred turns. Functions to run on the GPU must be labelled with void __global__ myname and are called with the syntax myname(params)<<<nthreads, nblocks>>> where nthreads is the number of threads per processor and nblocks is the number of processors in simple usage. Inside the function the thread and block number are available as special variables and can be used to index into the data for that part of the calculation. The integrator code is taken from the symplectic bending multipole in Tracy-II and used for all elements (it may be more efficient to use the standard lattice data structure with more than one pass method as there is no real penalty for taking branches as long as all threads take the same path, but for code brevity a single combined element is used).

The GPU has a small low-latency constant cache memory, the __constant__ declaration causes the compiler to target this memory. The Diamond lattice coefficients are small enough to fit into this so memory access for the lattice is essentially free and GPU main memory access is limited to writing phase space co-ordinates once per particle per turn. Intermediate phase space co-ordinates are stored in the large on-chip register files.

The lattice description is taken from Accelerator Toolbox (AT) [3] and written as multipole and bending coefficients to a text file. The parallel tracking program reads this and outputs another text file containing the phase space co-ordinates over each turn, for interactive use a Matlab extension would be more practical. It is not possible to make the GPU tracking an AT pass method as such as the GPU must track over many elements for efficiency but the AT lattice data structure could be passed directly to the tracking routine through the Matlab API. For batch mode operation a standard IO format such as Self-Describing Data Sets (SDDS) [5] would be desirable.

RESULTS

The code was used to perform fast previews of the dynamic aperture of candidates for the Diamond low-alpha lattice, one of which is shown below. The shape of the dynamic aperture and frequency map is preserved but the tune diffusion rate is inaccurate.

The single precision performance of one mid-range GPU with 64 stream processors is comparable to a 8 node, 64 core Intel Xeon cluster. No extra effort has been made to optimize the CPU software by using vectorized instructions. These figures do not include the NAFF [6] calculation time which is not a limiting factor. The purchase cost of the cluster is approximately 100 times that of the GPU, in addition to higher power consumption and administrative costs.

Table 1: Timing Results

Machine	Particles	Turns	Time (s)	Cost
9600GT	4096	2048	80	1
64 x L5430	4096	2048	80	100

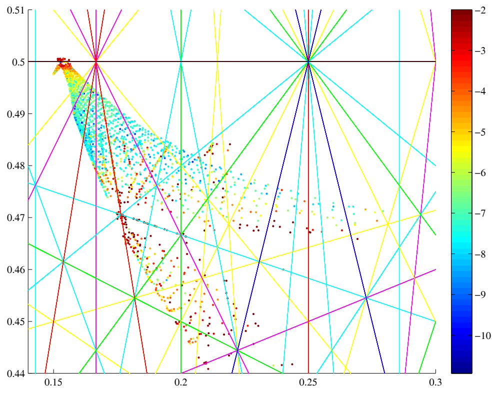

Figure 1: Frequency Map (CPU).

CONCLUSIONS

Comparing Fig. 1 with Fig. 2 and Fig. 3 with Fig. 4 it is clear that the dynamic range of the tune diffusion is limited in single precision mode but the bounds of the aperture and the shape of the map are preserved. Having access to a fast interactive tool to explore the shape if not the details of frequency maps and dynamic apertures should be useful for design studies even as a quick check before starting a large job on a cluster. It also opens up the possibility of global optimization of nonlinear features such as the dynamic aperture or resonance driving terms where the increased number of function evaluations may allow a rapid convergence to a reasonable point before continuing with a higher precision routine. The unfamiliar programming model, single precision arithmetic and general immaturity of the platform preclude any serious investment in converting existing software to make use of the GPU. The difficulty is similar to programming parallel message-passing algorithms before MPI. Special-purpose accelerators have

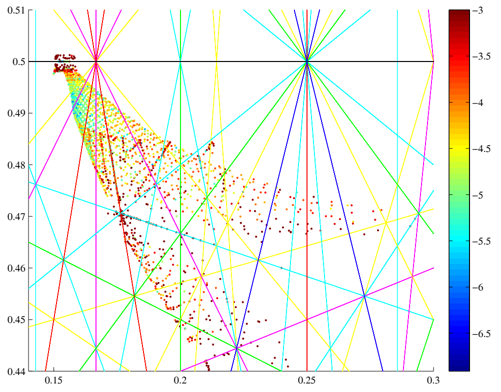

Figure 2: Frequency Map (GPU).

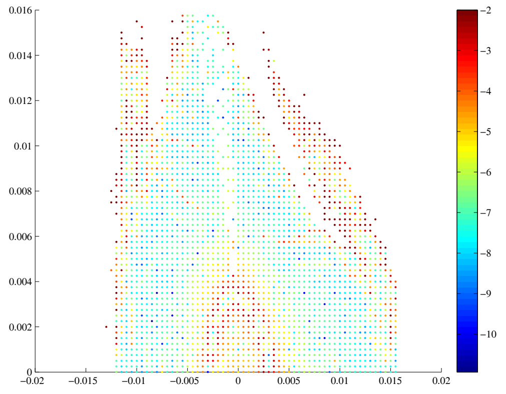

Figure 3: Dynamic Aperture (CPU).

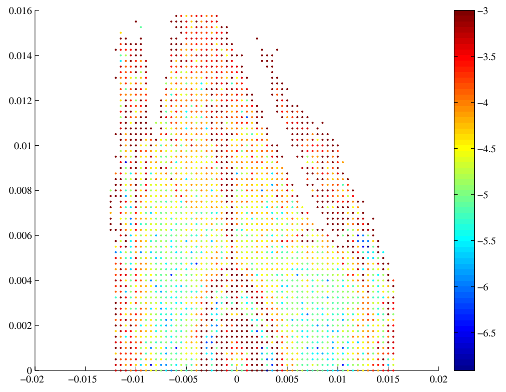

Figure 4: Dynamic Aperture (GPU).

regularly failed to keep up with the CPU due to economies of scale and engineering resources in the PC industry, however the programmable GPU has the benefit of being almost as common as the CPU, and with the increasing number of processor cores in CPUs the distinction may blur. A hybrid programming model addressing tightly coupled multi-core processors and message passing in the cluster will be the norm.

The authors have not considered other GPU programming libraries such as ATI's CTM or the cross platform OpenCL which is likely to replace both vendor specific APIs this year. Equally the cell broadband engine offers high performance but the benefit of the GPU is that it is a standard low-cost component in desktop computers.

Considering the extremely low cost of the GPU and the good match for single particle dynamics it may be worth reconsidering multi or mixed precision schemes. The limited implementation of C++ features in CUDA may not support an efficient implementation of operator overloading with expression templates but the combined element integrator code is small enough that replacing arithmetic operators with multi-precision arithmetic macros is not a limiting factor. However this is likely to incur at least a 10 times overhead [7] negating some of the performance gains of using the GPU.

REFERENCES

[1] J. Bengtsson, "TRACY-2 User's Manual", SLS Internal Document, February 1997

[2] NVIDIA, "NVIDIA_CUDA_Programming_Guide_1.1.pdf"

[3] A. Terebilo, "ACCELERATOR MODELING WITH MATLAB ACCELERATOR TOOLBOX", PAC 2001

[4] M. D. Salt, R. B. Appleby, D. S. Bailey, "Beam Dynamics Using Graphical Processing Units", EPAC 2008

[5] M. Borland, L. Emery, "The Self-Describing Data Sets File Protocol and Program Toolkit", ICALEPCS 1995

[6] J. Laskar et al., Physica D, 56, 253, (1992).

[7] T. Fukushima, "Reduction of Round-off Error in Symplectic Integrators", The Astronomical Journal, 121:1768-1775, 2001 March