A Generic Approach for Developing Highly Scalable Particle-Mesh Codes for GPUs

W. Hönig, F. Schmitt, R. Widera, H. Burau, G. Juckeland, M. S. Müller, and M. Bussmann

I. Introduction

We present a general framework for GPU-based low-latency data transfer schemes that can be used for a variety of particle-mesh algorithms [8]. This framework allows to hide the latency of the data transfer between GPU-accelerated computing nodes by interleaving it with the kernel execution on the GPU. We discuss as an example the fully relativistic particle-in-cell (PiC) code PIConGPU [5] currently used to simulate particle acceleration by extremely short high-energy laser pulses. The PiC algorithm is a versatile algorithm used frequently in plasma physics—especially for large-scale simulations of fusion plasmas [13]—, in astrophysics [7], [9], or for the simulation of particle accelerators [11]. A special Cell processor version is used as a benchmark code for the Roadrunner system at Los Alamos National Lab [4].

In general, particle-in-cell codes show very good weak scaling in both mesh size and particle number and thus are good candidates for massively-parallel computing approaches [10]. However, when considering vector parallelization as used for example by GPU accelerators one has to take into account that particles can cross the boundaries between mesh domains. Such particle crossings may cause non-local memory access every time mesh-related data such as fields and currents and particle data such as velocities and positions are accessed by the same kernel [7]. Random and non-local memory access can seriously degrade the overall performance on modern accelerator hardware [3], [12]. Thus, performant implementations of the PiC-algorithm with vector parallel memory access accelerator hardware have been sparse [1], [12]. The need for large-scale simulations of plasmas calls for a general approach towards vector-parallel PiC implementations.

The presented hybrid GPU-CPU data transfer and access framework is grid-based and can, furthermore, be used for general particle-mesh schemes. GPU memory access to particle data and regular grid data are efficiently separated, while data that has to be exchanged between domains located on different GPUs is transferred during computing steps using GPU-CPU memory transfers and MPI. A simulation of laser-wakefield acceleration of electrons in an underdense plasma serves as a real-world benchmark for the performance of the framework.

H. Burau and M. Bussmann are with the Forschungszentrum Dresden-Rossendorf e.V., D-01328 Dresden, Germany.

W. Hönig, F. Schmitt, R. Widera, G. Juckeland, and M. Müller are with Technical University Dresden, Center for Information Services and High Performance Computing (ZIH), D-01062 Dresden, Germany.

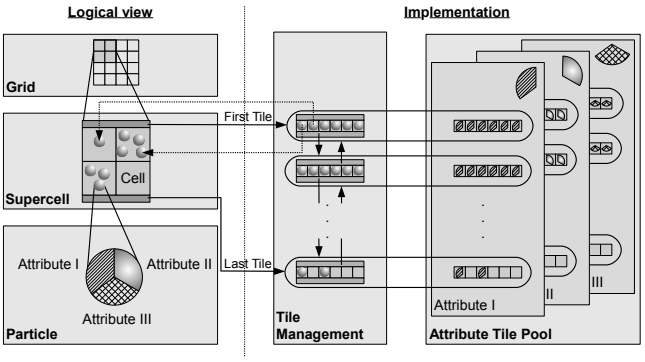

Fig. 1. Schema for tiling particles: A regular grid is divided into supercells, which contain a fixed number of cells. Each cell can hold an arbitrary number of particles. The attributes (e.g. velocity) are stored separately in an attribute tile pool where each attribute is stored in fixed-size vectors (at the same index as the particle in the particle array). Tiles consist of an 1D-array and two pointers which form a double-linked-list. The mapping between a particle and its cell is handled via special attributes in the tile management.

II. Single GPU Optimizations

A. Memory Model - Tiling Particles

For Particle-Mesh algorithms a mapping from cells to particles and vice versa is necessary. Our first basic solution was a particle list where each particle element holds a pointer to its owning cell and its successor [5]. Usually a PiC algorithm will run for a large number of iterations which leads to movement of many particles. This results in heavy fragmentation of the particle list which turns fast regular memory access patterns into slow random memory accesses and, thereby, to continuously increasing time needed for one iteration of the algorithm (figure 2).

The solution to this fragmentation problem is the introduction of a new data structure which uses a three stage memory hierarchy as shown in figure 1. This approach uses coalesced global memory accesses for all particle and cell data except tile domain border regions. Therefore, the global fragmentation does not affect the performance and we have no limit in the amount of particles per cell. Problem specific data has to be copied only if a particle moves to a neighboring supercell. Otherwise, only the cell index has to be adjusted.

The introduction of this data structure reduced the time needed for one iteration as well as the slowdown over time as shown in figure 2.

B. Comparing CUDA and OpenCL

Code written in OpenCL (Open Computing Language) is intended to work on multiple platforms with recompilation

Performance impact of old vs. new particle model

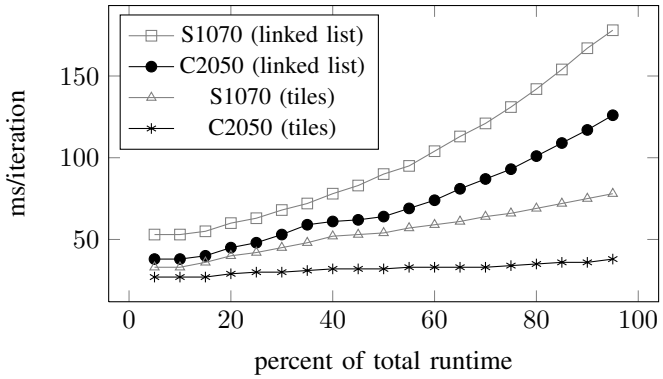

Fig. 2. Performance comparison of old (linked list) [5] and new (tiles) memory models on Tesla S1070 and C2050 (Fermi) GPUs. The Simulation uses one GPU, a 1024×2048 grid, about 30 mill. particles, 40,000 iterations. The memory-fragmentation and resulting performance drop can be reduced to almost no influence using the new particle tiling data structure and the latest NVIDIA hardware.

being the only necessity. Since OpenCL supports GPGPU as well, former CUDA applications can be extended to varying platforms and heterogeneous systems.

By our experience, OpenCL follows most of CUDA's design principles so that problems which emerged in the migration process rarely resulted from different language and API capabilities but from immature driver implementations. Examples are the tested Cell/B.E. driver by IBM which was still in early alpha stage as well as the AMD/ATI OpenCL driver. With the latter, we were unable to compute large problems on the ATI HD5770 as it could not access the full available memory.

For performance comparison we extended results from [6] by running several synthetic benchmarks which showed that current OpenCL implementations on GPU architectures give similar results to CUDA but cannot compete with the latter if highly optimized codes are used. Testing matrix multiplications, CUDA and OpenCL on the Tesla S1070 produced identical results when only global memory was accessed while CUDA surpassed OpenCL by factor 8 for an optimized version which utilized shared memory. We extend previous work on this topic by comparing two implementations of the described PIConGPU code on various NVIDIA and ATI graphic cards (figure 3).

After evaluating tests on GPU and CPU systems, we can conclude that though codes are portable in terms of correctness between platforms, optimization strategies are not as long as the underlying hardware architecture does not suite them.

III. Multi GPU Optimizations

A. Domain Decomposition and Boundary Exchange

Domain decomposition is a widely usable technique for particle-mesh computation [8]. The domain that is to be explored is partitioned into small parts called cells which contain particles (e.g. electrons). These can move between cells in each iteration and, thus, have to be exchanged between the cells' data structures as well. Our code works on multiple GPUs so that particle exchanges between different GPUs and hosts are

Performance of CUDA and OpenCL versions of PIConGPU

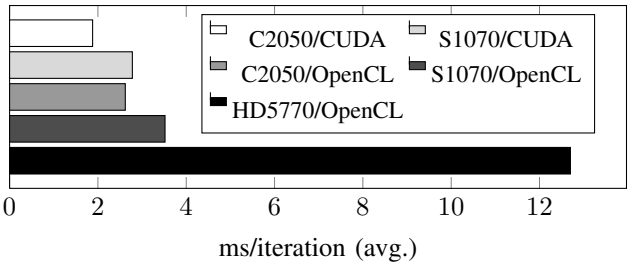

Fig. 3. Average iteration times for different GPU platforms using one GPU. The simulation uses a 512×512 grid and about 700,000 particles. The OpenCL version of the code produces similar results compared to CUDA on NVIDIA platforms, while the same code takes about three times as long on and AMD/ATI GPU.

required. The usage of MPI enables the reuse of previously established communication techniques between hosts [2]. In this special problem domain there are several appropriate performance optimizations to achieve a higher throughput and lower latency. The basic idea is to start each communication as soon as possible. Because all GPUs do not inevitably compute at the same speed, sending and receiving of data should be separated. The developed C++ library uses different buffers for all boundaries for sending and receiving and is capable of supporting 3D communication. The communication is done in the background using an event based programming mechanism as described in the next section.

B. Computation Communication Overlap

The main problem of parallel applications is that even the smallest communication/computation imbalance will ultimately limit the scalability of the program. Our presented solution communicates and computes in parallel exploiting the fact that one iteration of the code has various steps that also compute on different data. We start by computing the first step (electrical field for PIConGPU) including the border regions. While the second step, the magnetic field, is processed without the borders, we simultaneously transfer the previously calculated borders for the electrical field. After that, the remaining magnetic field can be computed. The presented libGPUGrid library has an easy to use interface to accomplish this: Instantiate `GridBuffer` and use `startSend()` or `startReceive()` to exchange its borders. All internals (especially MPI) are hidden. The two functions return an `EventTask` that can be used to test or wait for the communication to finish. A schematic timeline visualization (Fig. 4) illustrates this overlap.

C. Decoupling MPI and CUDA

All MPI and low level CUDA details (i.e. memory management) are hidden in the presented grid library. The CUDA kernels (user code) need not to know that several GPUs are involved. The user must only specify possible transfer directions by defining a communication mask and can start all needed communication with one function call. Internally employing several design patterns like `Observer`, `Strategy`

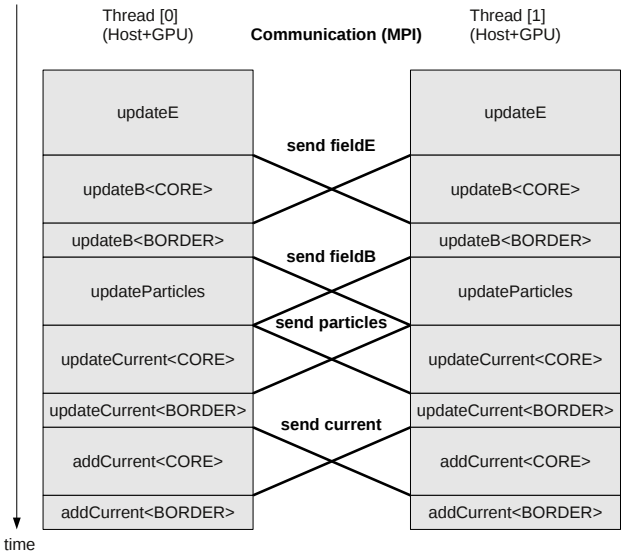

Fig. 4. The schematic program execution timeline of one iteration of the PIConGPU code shows how the communication/memory transfer (bold lines) of all data structures is interleaved with the corresponding computation steps to hide the transfer latency.

and `Adapter` allows the replacement of MPI with another inter-node communication paradigm but also OpenCL instead of CUDA for acceleration (see also Section II-B).

D. Performance

The performance was analyzed using up to 16 GPUs (4 Tesla S1070 boards with 4 GPUs each). Figure 5 shows our results for strong (i.e. fixed total problem size) and weak (i.e. fixed problem size per GPU) scaling.

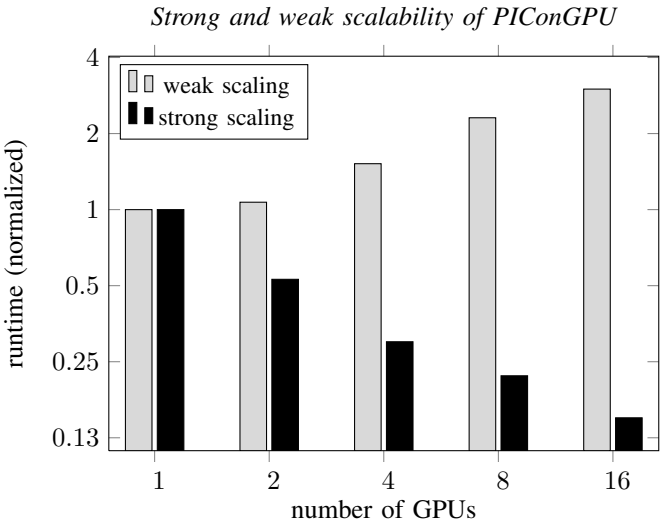

Fig. 5. Calculation time for PIConGPU with [5] memory model: strong (2048×2048 grid, about 28 mill. particles) and weak scaling (2048×2048 grid, about 7.5 mill. particle per GPU) both over 20000 iterations. Using 16 GPUs, a parallel speedup of 6.7 is reached for strong scaling.

IV. Summary and Outlook

We presented the internals of a general framework for using clusters of GPU nodes for particle-mesh algorithms. The capabilities of this framework include abstraction of all communication (between hosts and between host and accelerator devices), the possibility to easily interleave computation and communication, as well as a GPU oriented memory management for particle data. One particular implementation using this framework—PIConGPU—proves the efficiency of the approach by providing to our knowledge the lowest time to compute an iteration compared to other (accelerated) Particle-in-Cell codes. Additionally, we presented experience in porting our PiC code to heterogeneous systems using OpenCL.

Further work on the framework includes a library to abstract particle handling as well.

References

[1] Paulo Abreu, Ricardo Fonseca, Joao Pereira, and Luis Silva. PIC codes in new processors: a full relativistic PIC code in CUDA enabled hardware with direct visualization. In *21st International Conference on Numerical Simulation of Plasmas*, page submitted. IEEE Press, 2009.

[2] K. J. Bowers, B. J. Albright, B. Bergen, L. Yin, K. J. Barker, and D. J. Kerbyson. 0.374 Pflop/s trillion-particle kinetic modeling of laser plasma interaction on Roadrunner. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, 2008. IEEE Press.

[3] K. J. Bowers, B. J. Albright, L. Yin, B. Bergen, and T. J. T. Kwan. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation. *Physics of Plasmas*, 15(5):055703–1–055703–7, March 2008.

[4] K. J. Bowers, B. J. Albright, L. Yin, W. Daughton, V. Roytershteyn, B. Bergen, and T. J. T. Kwan. Advances in petascale kinetic plasma simulation with VPIC and Roadrunner. *Journal of Physics: Conference Series*, 180:012055+, 2009.

[5] Heiko Burau, René Widera, Wolfgang Hönig, Guido Juckeland, Alexander Debus, Thomas Kluge, Ulrich Schramm, Tomas E. Cowan, Roland Sauerbrey, and Michael Bussmann. PIConGPU – A fully relativistic particle-in-cell code for a GPU cluster. *Special Issue of the IEEE Transactions on Plasma Science on Numerical Simulation of Plasma*, 2010. accepted for publication.

[6] A. Danalis, G. Marin, C. McCurdy, J. Meredith, P. Roth, K. Spafford, V. Tipparaju, and J. Vetter. The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite. *Proceedings of the Third Workshop on General-Purpose Computation on Graphics Processors (GPGPU 2010)*, 2010.

[7] R. A. Fonseca, S. F. Martins, L. O. Silva, J. W. Tonge, F. S. Tsung, and W. B. Mori. One-to-one direct modeling of experiments and astrophysical scenarios: pushing the envelope on kinetic plasma simulations. *Plasma Physics and Controlled Fusion*, 50(12):124034+, December 2008.

[8] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Taylor & Francis, January 1989.

[9] Peter Messmer. Par-T: A Parallel Relativistic Fully 3D Electromagnetic Particle-in-Cell Code. In Tor Sørevik, Fredrik Manne, Assefaw H. Gebremedhin, and Randi Moe, editors, *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, volume 1947 of *Lecture Notes in Computer Science*, chapter 41, pages 350–355. Springer Berlin Heidelberg, Berlin, Heidelberg, April 2001.

[10] K. Paul, D. L. Bruhwiler, B. Cowan, J. R. Cary, C. Huang, F. S. Tsung, W. B. Mori, E. Cormier-Michel, C. G. R. Geddes, E. Esarey, S. Martins, R. A. Fonseca, and L. O. Silva. Benchmarking the codes VORPAL, OSIRIS and QuickPIC with Laser Wakefield Acceleration Simulations. In *Advanced. Accel. Concepts Workshop*, volume LBNL-2293E, pages 315–320, 2008.

[11] David B. Serafini, Peter Mccorquodale, and Phillip Colella. Advanced 3D Poisson solvers and particle-in-cell methods for accelerator modeling. *Journal of Physics: Conference Series*, 16(1):481+, January 2005.

[12] George Stantchev, William Dorland, and Nail Gumerov. Fast parallel Particle-To-Grid interpolation for plasma PIC simulations on the GPU. *J. Parallel Distrib. Comput.*, 68(10):1339–1349, October 2008.

[13] J. L. Vay, P. Colella, P. Mccorquodale, Van, A. Friedman, and D. P. Grote. Mesh refinement for particle-in-cell plasma simulations: Applications to and benefits for heavy ion fusion. *Laser and Particle Beams*, 20(04):569–575, 2002.