ISBN: 978-3-95450-184-7 doi:10.18429/JACoW-IPAC2018-THPAK043

PERFORMANCE OPTIMIZATION OF A BEAM DYNAMICS PIC CODE ON HYBRID COMPUTER ARCHITECTURES

Zhicong Liu[1*], Ji Qiang[†], Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA
[1]also at Key Laboratory of Particle Acceleration Physics and Technology,
Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China

Abstract

The self-consistent multi-particle tracking based on particle-in-cell method (PIC) has been widely used in particle accelerator beam dynamics study. However, the PIC simulation is time-consuming and needs to use modern parallel computers for high resolution applications. In this paper, we implemented and optimized a parallel beam dynamics PIC code on two types of hybrid parallel computer architectures: one is the GPU and GPU cluster, while the other is the "Knight Landing" CPU cluster.

INTRODUCTION

The particle-in-cell (PIC) method is widely used as the self-consistent space charge solver in the simulation codes in the accelerator community [1–7]. The PIC code is usually computationally expensive. A number of parallel quasi-static PIC codes using Message Passing Interface (MPI) had been developed in the accelerator community for high intensity/high brightness beam simulations [2–5].

The Graphics Processing Unit (GPU), which was originally developed for computer graphics and video game, now becomes a general-purpose computer processor [8]. In contrast to the Central Processing Unit (CPU), one GPU contains several hundreds or even thousands of cores , as shown in Fig. 1 . The Compute Unified Device Architecture (CUDA) library is a parallel computing platform and programming model for GPUs developed by the NVIDIA [9]. It enables a fast implementation of numerical model on GPUs and dramatically increases computing performance by harnessing the power of the GPU.

Following the previous work [10–13], a multi-particle tracking code based on PIC method is under development using CUDA running at both single GPU and GPU cluster. Using a single home-use GPU GTX 1060, the code speeds up by more than 50 times compared with that running on an AMD Opteron 6134 CPU core. Also, it shows good scalability on a cluster when particle number is large.

Besides the GPU implementation, the code is also ported and tuned on an advanced CPU cluster - Cori Knight Landing(KNL), which is based on the Intel Many Integrated Core (MIC) Architecture. The implementation and test on single GPU and different clusters give a comprehensive evaluation of the performance of PIC code on different hybrid computer architectures.

* liuzhicong@ihep.ac.cn
† jqiang@lbl.gov. Work supported by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 and the Ministry of Science and Technology of China under Grant No.2014CB845501.

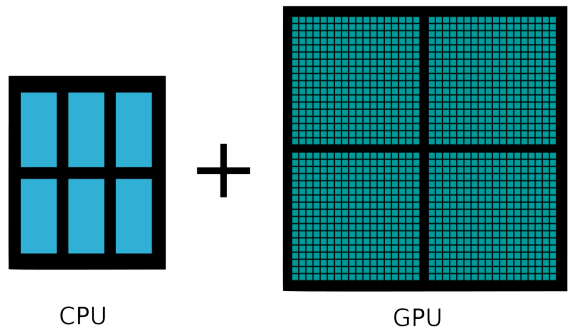

Figure 1: GPU vs CPU

In this paper, after the Introduction, the GPU implementation of PIC code, especially the parallel particle reordering, is presented in Section 2. After that, the performance of the tracking code running on single GPU, GPU cluster, and CPU cluster is presented in Section 3. Finally, conclusions are drawn in Section 4.

PIC CODE ON GPU

In a typical PIC method, the particles are deposited onto the grid points firstly. Then, the field at grid points is obtained by solving the Poisson equation using the Fast Fourier Transformation (FFT). Finally, the particles are pushed and kicked by the electromagnetic field. The loop continues until we attain enough number of steps. When porting the PIC method onto hybrid multi-core architecture, the second and third steps are similar to a regular serial PIC code. However, as to the first step (depositor) using multi-thread, the race condition rises and may lead to wrong results. To avoid the race condition, it's necessary to reorder the particle before the depositor at each steps by dividing the girds into smaller tiles.

To reorder the particles, firstly, the arrays `nhole` and `ndirec` are declared to handle the indices and the number of particles that would leave the current tile to each direction, as shown by the orange arrows in Fig. 2. The `nhole` is pre-allocated at given size, which determines the maximum number of particles leaving these current tiles. The size is calculated by the available GPU memory size.

Secondly, the particles leaving a tile are copied into an ordered global buffer: `pbuff`. With a running sum to the `ndirec`, we can know the memory address where we would put the particles to, so the particles going to the same direction are stored contiguously.

Thirdly, for each tile, we could know how many particles would move in and where they are located in `pbuff` by the

ISBN: 978-3-95450-184-7 doi:10.18429/JACoW-IPAC2018-THPAK043

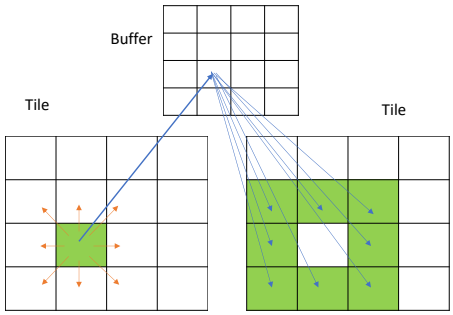

Figure 2: Particle reordering.

`nhole` and `ndirec` of the neighbor tiles. If there is a particle that leaves this tile, the hole left would be filled by incoming particle firstly. After all holes are filled, the particle data is written at the end of the array. If there are still holes after we wrote all incoming particles, the last several particles are moved to fill the holes.

In this way, we ensure that the particles are ordered in this tile and always occupy a continues memory. After reordering and depositing the particles onto the grid, the next step is to solve the Poisson equation on the grid. In the GPU implementation, we use NVIDIA's CUDA Fast Fourier Transform Library (cuFFT) [14] to do this.

We tested domain decomposition parallel method by using multi-GPUs to process different spatial sub-domains, so each GPU would have less computation and thereby the speed of the program would be increased. However, domain decomposition requires communication among different GPUs. Since the GPUs cannot directly exchange information among each other currently, especially between different nodes, we need to copy the data from the GPU back to the CPU memory and communicate on the CPU side, which will takes extra time. So the efficiency of the domain decomposition parallel method will depend on the balance of extra data moving time and the reduced computation time. According our test, domain decomposition parallel method doesn't show any advantage.

PERFORMANCE

The performance of PIC code was tested on a single GPU, GPU cluster Titan and CPU cluster Cori Knight Landing [15, 16]. In the following of this section, the performance running on each computer platform is presented.

Single GPU Speedup

The speed of the PIC code running on single GPU is tested firstly. The GPU code uses an NVIDIA GeForce GTX 1060 6GB. For comparison, the CPU code runs on an AMD Opteron(TM) Processor 6376, 2.3GHz. The speedup is calculated by the CPU runtime divided by the GPU runtime. In this performance test, the grid number is $64 \times 64 \times 64$ while the particle number varies from 16 thousand to 1.6 million.

As shown in Fig. 3, generally, we achieved a speedup of more than 40 for the whole PIC code. The speedup of the Poisson solver, colored as orange in Fig. 3, is about 64 and almost keep constant as expected. The speedups of some other functions, like depositor, kicker, pusher, and output, become larger as the increase of particle number.

The depositor of GPU code also includes the particle reordering operation. Because of the irregularity of reordering, the speedup of depositor is relatively low. The diagnostic output also contains the calculation of the statistics of the beam parameter, and the reason for low speedup is due to the limit of output bandwidth. The relatively small speedups for depositor and output reduce the speedup of the entire code.

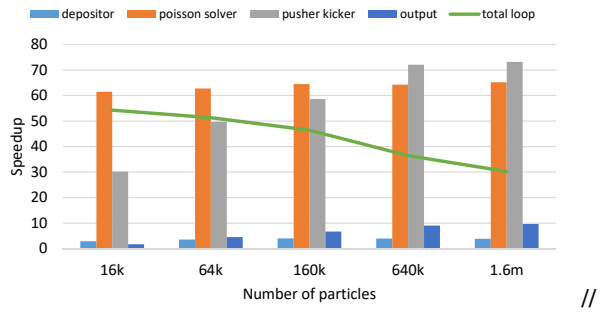

Figure 3: The speedup of PIC Code using single GPU.

The speedup of the total time decreases when the particle number becomes larger. The reason is that the time consumed by the depositor, which has a lower speedup, dominates when the particle number becomes larger.

Overall, we attain a speedup of more than 60 for the Poisson solver, and about 40 for the whole code using a home-use GPU. This is about two times faster than its MPI version running on a 64 cores computer.

GPU Cluster Speedup - Titan

After testing on a single GPU, a scaling test for PIC code on GPU cluster was done on TITAN. Figure 4 shows the results with $1.6M$ particles. The total time decreases with more GPUs, and reach minimum at 32 GPUs. The reason is that the time consumed by pusher, kicker, and depositor dominates in the large particle number case, which can be well speeded up by multi-GPUs.

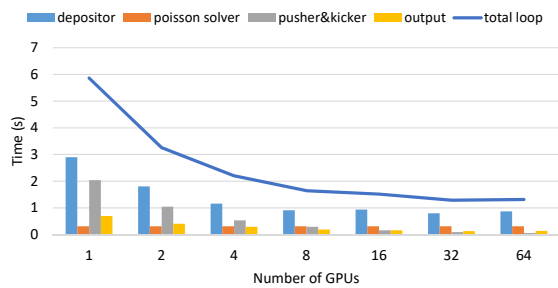

Figure 4: The scalability of the PIC code using $64 \times 64 \times 64$ grid points and $1.6M$ particles on Titan.

However, when we try a even larger number of particles, 16M particles, we cannot run the code on only 1 or 2 GPUs,

ISBN: 978-3-95450-184-7 doi:10.18429/JACoW-IPAC2018-THPAK043

as seen in Fig. 5. It should be noted that the problem size is limited by the GPU memory size. Unlike CPU memory, which can be easily extended, the GPU memory is fixed in a given GPU model. Ideally, for a GPU with a memory size of 6GB, the maximum particle number is about 60M with a uniform distribution. But it is difficult to reach this number because it's more common to use a WaterBag, or a Gaussian distribution instead of a uniform distribution in most of the accelerator applications. Except for efficiency, the limit by memory size is another reason why we need multi-GPUs.

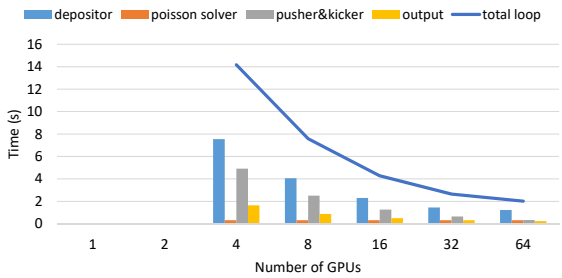

Figure 5: The scalability of the PIC code using $64 \times 64 \times 64$ grid points and $16M$ particles on Titan.

CPU Cluster Speedup - Cori-KNL

Besides GPU cluster, the PIC code is also ported and tested on Cori Knight Landing(KNL), a hybrid supercomputer based on MIC technology. The OpenMP(OMP) directives were added to the original pure MPI code to adapt to the Intel Many Integrated Core (MIC) architecture. The new hybrid MPI+OpenMP PIC code uses fine-grained parallelism, and the balance between MPI rank size and OpenMP threads number at given number of cores is finely tuned. For the potential race condition at some subroutines, like depositor, the OpenMP intrinsic reduction and atomic operator is used. Aa strong scaling test was done using gradually increasing number of nodes with different problem sizes. It should be noted that the Cori Knights Landing has 68 CPU cores per node, however, we only used 64 cores per node for easier comparison.

Using pure MPI, the code shows a good scalability in Fig. 6, where the abscissa is the number of nodes, and the left ordinate is time by seconds, while the right ordinate is memory usage by GB. The total time decrease first and reach minimum on 32 nodes, after which the total time starts to increase, and the time for communication dominates.

The comparison of total time and memory usage by pure MPI(OMP=1), OMP=2, and OMP=4 is shown in Fig. 7. The difference is small. But for some case, like node = 16, the pure MPI is faster than the OMP=4 by a factor of about 2. As to the memory usage, a larger OpenMP threads number always wins. In most cases, the pure MPI is a good choice if we have a large enough memory.

For comparison between GPU code and CPU code, a common problem size in real physics simulation, 1.6 million particles and $64 \times 64 \times 64$ grid points, is selected. The total loop time using a single Nvidia GeForce GTX 1060 GPU

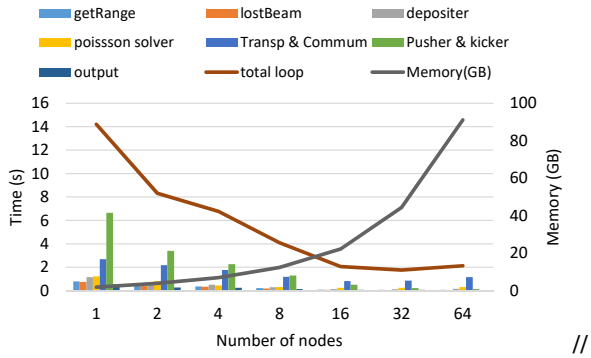

Figure 6: The time consuming of PIC code using multi-node at Cori knight Landing.

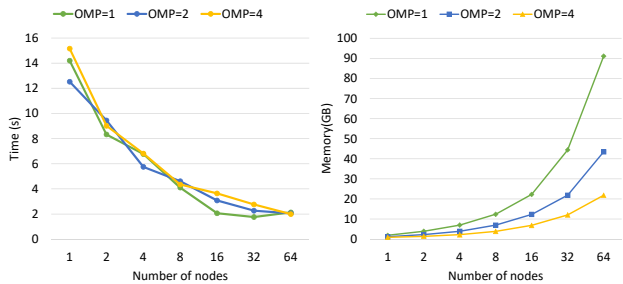

Figure 7: a) total time and b) memory usage of different parallel configuration.

card is 3.56 seconds. It is comparable with the CPU code running at around 4 or 8 nodes on Cori Knight Landing. In other words, for our PIC code, the performance of using one GPU card is comparable with that using 256 to 512 Knight Landing CPU cores.

CONCLUSIONS

A multi-particle beam dynamics simulation code using PIC method was implemented on GPU using the CUDA library. The GPU code structure and parallel strategy about how to avoid race condition were discussed. We achieved a maximum speedup of more than 50 using a common home-use GPU. The PIC code also shows good scalability on a GPU cluster Titan. We also ported this code and explored the performance optimization on Cori Knight Landing. In the future study, we will continue to extend this code and to enhance the efficiency. We would also like to compare PIC model with the gridless symplectic models on GPUs in our future work.

ACKNOWLEDGMENTS

One of the author, Zhicong Liu, would like to extend his thanks for the financial support from China Scholarship Council (CSC, File No. 201604910876). We have used computing resources at the Oak Ridge Leadership Computing Facility (OLCF) and National Energy Research Scientific Computing Center(NERSC).

ISBN: 978-3-95450-184-7 doi:10.18429/JACoW-IPAC2018-THPAK043

REFERENCES

[1] C. K. Birdsall, "Particle-in-cell charged-particle simulations, plus monte carlo collisions with neutral atoms, pic-mcc," *IEEE Transactions on Plasma Science*, vol. 19, pp. 65–85, Apr 1991.

[2] A. Friedman, D. P. Grote, and I. Haber, "Three-dimensional particle simulation of heavy-ion fusion beams," *Physics of Fluids B: Plasma Physics*, vol. 4, no. 7, pp. 2203–2210, 1992.

[3] J. Qiang, R. D. Ryne, S. Habib, and V. Decyk, "An object-oriented parallel particle-in-cell code for beam dynamics simulation in linear accelerators," *Journal of Computational Physics*, vol. 163, no. 2, pp. 434 – 451, 2000.

[4] J. Qiang, M. A. Furman, and R. D. Ryne, "A parallel particle-in-cell model for beam–beam interaction in high energy ring colliders," *Journal of Computational Physics*, vol. 198, no. 1, pp. 278–294, 2004.

[5] J. Amundson, P. Spentzouris, J. Qiang, and R. Ryne, "Synergia: An accelerator modeling tool with 3-d space charge," *Journal of Computational Physics*, vol. 211, no. 1, pp. 229 – 248, 2006.

[6] D. Uriot and N. Pichoff, "Tracewin," *CEA Saclay, June*, 2014.

[7] Y. K. Batygin, "Particle-in-cell code beampath for beam dynamics simulations in linear accelerators and beamlines," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 539, no. 3, pp. 455–489, 2005.

[8] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[9] C. Nvidia, "Programming guide," 2010.

[10] G. Stantchev, W. Dorland, and N. Gumerov, "Fast parallel particle-to-grid interpolation for plasma pic simulations on the gpu," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1339–1349, 2008.

[11] H. Burau, R. Widera, W. Honig, G. Juckeland, A. Debus, T. Kluge, U. Schramm, T. E. Cowan, R. Sauerbrey, and M. Bussmann, "Picongpu: A fully relativistic particle-in-cell code for a gpu cluster," *IEEE Transactions on Plasma Science*, vol. 38, no. 10, pp. 2831–2839, 2010.

[12] V. K. Decyk and T. V. Singh, "Particle-in-cell algorithms for emerging computer architectures," *Computer Physics Communications*, vol. 185, no. 3, pp. 708–719, 2014.

[13] X. Pang and L. Rybarcyk, "Gpu accelerated online multi-particle beam dynamics simulator for ion linear particle accelerators," *Computer Physics Communications*, vol. 185, no. 3, pp. 744–753, 2014.

[14] C. Nvidia, "Cufft library," 2010.

[15] D. Tiwari, S. Gupta, G. Gallarno, J. Rogers, and D. Maxwell, "Reliability lessons learned from gpu experience with the titan supercomputer at oak ridge leadership computing facility," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, p. 38, ACM, 2015.

[16] Y. He, B. Cook, J. Deslippe, B. Friesen, R. Gerber, R. Hartman-Baker, A. Koniges, T. Kurth, S. Leak, W.-S. Yang, *et al.*, "Preparing nersc users for cori, a cray xc40 system with intel many integrated cores," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 1, 2018.