Multi-Core Optimizations for Synergia and ART

[Extended Abstract]

Qiming Lu
Scientific Computing Division, Fermilab
P.O. Box 500
Batavia, Illinois 60510
qlu@fnal.gov

James Amundson
Scientific Computing Division, Fermilab
P.O. Box 500
Batavia, Illinois 60510
amundson@fnal.gov

Nick Gnedin
Particle Physics Division, Fermilab
P.O. Box 500
Batavia, Illinois 60510
gnedin@fnal.gov

ABSTRACT

New technologies emergence at a rapid pace in the realm of the high performance computing: from peta-scale to the approaching exa-scale computing which requires the collaboration from millions of closely coupled CPUs; now widely used multi-core multi-socket architectures; new but rapidly growing acceleration computing devices such as GPUs and Intel MIC architectures. They lead to new possibilities in the scientific computations and simulations, and bring them to a new level in regards of both the performance and simulation scales. Meanwhile challenges are imminent for the authors of current and future simulation packages, to understand and make the best out of these technologies.

In this poster we describe our recent work in optimizing the performance and scaling of two simulation packages: Synergia and ART, for multi-socket multi-core architectures including BlueGene/Q and GPUs.

Synergia is a beam dynamics simulation code package widely used in the community for advanced accelerator modeling and simulations. The package is capable of modeling both single-particle and multiple particle dynamics in cases when the particle-particle interactions are important, such as the simulations of beam-beam interactions, electron clouds, or the wakefield effects, etc.

The code of Synergia is developed and maintained at Fermilab. Before the optimization it is a pure MPI based parallel space charge Particle-in-Cell (PIC) program. It is not aware of the multi-core architecture, and the scalability of Synergia is largely limited by the Poisson solver of the PIC method. When running large scale simulations the strong scaling is often harmed by the spiking of the communication cost.

The Adaptive Refinement Tree (ART) is a high performance simulation package developed for computational cosmology. It is also a powerful tool of discovery in extracting insights and making precision predictions for the observations from large-scale surveys of the sky such as SDSS and LSST. It models a wide range of physical processes including *a*) detailed atomic physics of the cosmic plasma; *b*) the effects of cosmic radiation on the gas; and *c*) formation of stars and their feedback on the cosmic gas. Numerical methods used in the ART code include the PDE solver for the hydrodynamics simulations, and PIC method for processes involving particles such as the gravity and radiation transfers, etc. Both methods use adaptive mesh refinements on regular grids. Because of the on-demand and dynamic mesh generation and refinements, the computations are extremely in-balanced.

We show multiple hybridization and optimization options on both of the code packages, including communication avoidance, interchangeable multi-threading kernel using OpenMP or CUDA for different hardware architectures, customized FFT, etc., each demonstrating much better scaling behavior than the pre-optimization code.

The hybridization introduces low-level thread parallelism into the parallelization framework. The hierarchical structure allows for distributed memory parallelism at high level for massive scalability, while reducing the number of MPI tasks when shared memory parallelism is available. It reduces both the collective and point-to-point communication costs effectively by *a*) reducing the number of total MPI processors; and *b*) using the shared memory within an MPI processor to avoid explicit data passing. Hence the overall scaling has been improved significantly.

The hybrid design has also isolated the thread parallelization implementation details from the high level structures, making it possible to have interchangeable computing kernel implementations from various multi-threading libraries and hardware architectures, such as OpenMP/OpenACC/CUDA on multi-core CPUs/GPUs/Intel MIC, etc. The flexible design further supports the mixture of the computing kernels from various computing or acceleration devices simultaneously to do true heterogeneous computing. In making the transition from distributed memory to shared memory computation, the parallel PIC algorithm has been redone using both OpenMP and CUDA. Specifically, two collision-free charge/mass deposition algorithms have been implemented and assessed. The distributed deposition works better at multi-core parallel environment when the number of cores is relatively low. The interleaved deposition method is superior for its scalable overheads and fixed memory usage, when the computing device can provide massive parallelization capability, e.g., a GPU or a Intel MIC.

Other optimization approaches are exploited as well. The redundant field solver constrains the communication within a small sub-group of processors, thus has greatly simplified the communication pattern and reduced the total amount of data exchanges. Using local memory caches and pinning threads to physical cores alleviates the pressure from accessing remote memory and reduces the NUMA effects. Customized FFT parallelization based on FFTW improves the throughputs by a factor or 2.

By implementing these optimization techniques, we have extend strong scaling and peak performance by at least a factor of 2. We expect different optimization schemes to be optimal on different architectures. We have further tailored the code for BG/Q with optimized communication divider, redundant field solver, and FFT methods. The final code of Synergia scales up to 128K cores with over 90% efficiency running on Mira (BG/Q at Argonne).

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*parallel programming*; I.6.3 [**Simulation and Modeling**]: Applications; C.1.4 [**Processor Architectures**]: Parallel Architectures

Keywords

parallel programming, multi-core architecture, thread parallelization