Chapter 11

Particle Simulator Research Team

11.1 Members

Junichiro Makino (Team Leader)

Masaki Iwasawa (Research Scientist)

Daisuke Namekata (Postdoctoral Researcher)

Miyuki Tsubouchi (Technical Staff)

11.2 Overview of Research Activities

We are developing particle-based simulation software that can be used to solve problems of vastly different scales.

Simulation schemes for hydrodynamics and structural analysis can be divided into grid-based and particle-based methods. In grid-based methods, the computational region is mapped to regular or irregular grids. Continuous distributions of physical values are represented by discrete values at grid points, and the governing partial differential equation is approximated to a set of finite difference equations.

In the case of the particle-based methods, physical values are assigned to particles, while the partial differential equation is approximated by the interactions between particles.

Both methods are widely used, and they have their advantages and disadvantages. The computational cost of grid-based schemes is generally lower than that of particle-based methods with similar number of freedoms. Thus, if an near-uniform grid structure is appropriate for the problem to be solved, grid-based methods perform better.

The advantage of the particle-based methods comes from the fact that they use "Lagrangian" schemes, in which the particles move following the motion of the fluid in the case of the CFD calculation. In the case of grid-based methods, we generally use "Eulerian" schemes, in which the grid points do not move.

There are three points in which the Lagrangian schemes are better than Eulerian schemes. One is that the Lagrangian schemes are, to some extent, adaptive to the requirement of the accuracy, since when a low-density region is compressed to become high density, Second one is that the timestep criteria are quite different. In the case of the Lagrangian schemes, the timestep is determined basically by local sound velocity, while in the Eulerian scheme by global velocity. Thus, if a relatively cold fluid is moving very fast, the timestep for the Eulerian schemes can be many orders of magnitude shorter than that for Lagrangian schemes. Finally, in the case of fast-moving low-temperature fluid, the required accuracy would be very high for Eulerian scheme, since the error comes from the high velocity, while that error would be transferred to internal energy of the fluid element which is much smaller than that of the kinetic motion.

Of course, there are disadvantages of Lagrangian schemes. The primary one is the difficulty of construction of such schemes in two or higher dimensions. In the case of one-dimensional calculation, it is easy to move grid points following the motion of the fluid, but in two or higher dimensions, the grid structure would severely deform if we let the grid points follow the flow. Thus, we have to reconstruct the grid structure every so often. This requirement causes the program to become complex. Moreover, reconstruction of the grid structure (so called remeshing) means we lose numerical accuracy.

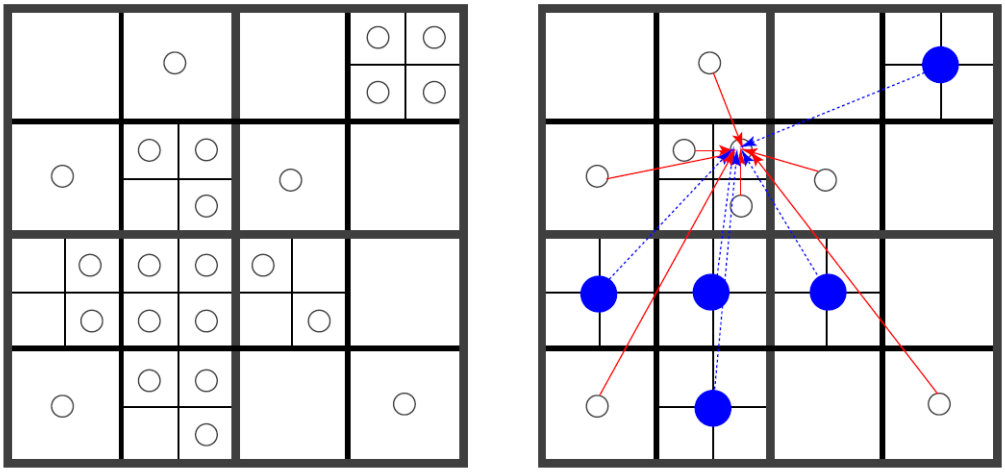

Figure 11.1: Basic idea of tree algorithm

Particle-based methods "solve" this difficulty by not requiring any mesh. In particle-based methods, particles interact with its neighboring particles, not through some connection through grid, but through distance-dependent kernel functions. Thus, there is no need of remeshing. As a result, particle-based schemes are simple to implement, and can give reasonable results even when the deformation is very large. Another important advantage is that it is relatively easy to achieve high efficiency with large-scale particle-based simulation.

In the case of grid-based schemes, in order achieve some adaptivity to the solution, we have to use either irregular grid or regular grid with adaptive mesh refinment. In both cases, adaptivity breaks the regularity of the mesh structure, resulting in non-contiguous access to the main memory. In the case of the particle-based schemes, it does require some irregular memory access, but it is relatively straightforward to make good use of spacial locality, and thereby achieving high efficiency. Similarly, very high parallel performance can be achieved.

However, it has its own problems. In the case of the SPH method, it has been known that the standard scheme cannot handle the contact discontinuity well. It also require rather strong artificial viscosity, which results in very low effective Reynolds number.

Thus, in many fields of computational sciences, many groups are working on implementation of high-performance particle-based simulation codes for their specific problem.

One serious problem here is that, high-performance, highly-parallel simulation codes for particle-based simulations are becoming more and more complex, in order to make full use of modern supercomputers. We need to distribute particles to many computing nodes in an appropriate way, so that the communication between nodes is minimized and at the same time near-optimal load balance is achieved. Within each nodes, we need to write an efficient code to find neighbor particles, rearrange data structure so that we can make good use of the locality, make good use of multiple cores and SIMD units within each core.

Even for the case of very simple particle-particle interaction such as the Lenard-Jones potential or Coulomb potential, the calculation code tends to be very large, and since the large fraction of the code is written to achieve a high efficiency on a specific architecture, it becomes very hard to port a code which is highly optimized to one architecture to another architecture.

Our goal is to develop a "universal" software that can be applied to a variety of problems whose scales are vastly different. In designing such universal software, it is important to ensure that it runs efficiently on highly parallel computers such as the K computer. Achieving a good load balance for particle-based simulations is a difficult task, since using a regular spatial decomposition method causes severe load imbalance, though this works well for grid-based software. Consequently, we have developed an adaptive decomposition method that is designed to work in a way that the calculation time on each node is almost the same, resulting in the near-optimal load balance.

The strategy to develop such a universal software is as follows.

We first construct an highly parallel and very efficient implementation of the TreePM algorithm for gravitational N-body problem. This is actually not a completely new implementation, but the GreeM code developed by researchers of the Strategic Program for Innovative Research (SPIRE) Field 5 "The origin of matter and the universe. In collaboration with the Field 5 researchers, we improve the efficiency of the code and study the

issues of the data structure, domain decomposition, load balance strategy etc.

In the second stage, we will develop a prototype of the parallel particle simulation platform. We will design the platform so that it can be used for multiple physical systems. In practice, we consider the following three applications as the initial targets.

1. Gravitational N-body simulation
2. Smoothed Particle Hydrodynamics
3. Molecular Dynamics

In the meantime, we will also investigate the way to improve the performance and accuracy of the current particle-based algorithms for hydrodynamics.

11.3 Research Results and Achievements

11.3.1 High-performance gravitational N-body solver.

We use the TreePM algorithm as the basic method for the evaluation of gravitational interaction between particles. TreePM is a combination of the tree method and the P^3M (particle-particle particle-mesh) scheme. Figure 1 shows the basic idea of the tree algorithm. The space is divided into a hierarchical octree structure (quadtree in the figure). Division is stopped when a cell contains one or no particle. When we calculate the force on a particle, we evaluate the force from a group of particles, with size larger for more distant particles. In this way, we can reduce the calculation cost from O(N2) to O(N log N).

The tree algorithm is widely used, but when the periodic boundary condition is applied, we can actually use a more efficient efficient scheme, since we can calculate the long-range, periodic term using FFT. The P^3M scheme has been used for such problem, but it has the serious problem that when the density contrast becomes high, the calculation cost increases very quickly. The TreePM scheme solves this difficulty by using the tree algorithm to evaluate the forces from nearby particles. Even when there are very large number of neighbor particles, the calculation cost does not increase much, since the calculation cost of the neighbor force is proportional to the logarithm of the number of neighbors.

In order to map the problem to the distributed-memory parallel computer such as the K computer, we adopted the approach to divide the space into domains and assign particles in one domain to one calculation node. We used the orthogonal recursive multisection method developed by the team leader some years ago. It is the generalization of the orthogonal recursive bisection (ORB), which has been widely used in many parallel implementations of the tree algorithm.

With ORB, we recursively divide space into two halves, each with the same number of particles. An obvious disadvantage of the ORB approach is that it can utilize the computing nodes of integral powers of two. Thus, in the worst case we can use only half of the available nodes.

The difference between the multisection method and the ORB is that with the multisection method we allow the divisions to arbitrary number of domains, instead of bisection. This would allow too many possible divisions. In our current implementation, we limit the number of levels to three, and make the numbers of divisions at all levels as close as possible. Thus, our domain decomposition is topologically a simple three-dimension grid. This fact makes the multisection method well suited to the machines with the 3D torus network like the K computer.

We have developed a "reference code" for gravitational N-body simulation on the K computer. This code is fairly well optimized for the K computer, and shows quite good scalability for even for relatively small-size problems. The asymptotic speed per timestep for large number of nodes is around 7ms. This speed is comparable to that of highly optimized molecular dynamics codes on K, even though our code is designed to handle highly inhomogenous systems.

We used this code as the reference implementation for more generalized particle simulation platform which will be described in the next subsection.

11.3.2 Particle Simulation Platform.

In FY 2014, We have completed and released Version 1.0 of the particle simulation platform, which we call FDPS (Framework for Developing Particle Simulator). In FY 2015, we have applied a number of improvements to FDPS.

The basic idea of FDPS is that the application developer (or the user) specified the way the particles interact with each other, and the rest is taken care by FDPS. Here, "the rest" includes domain decomposition and re-distribution of particles, evaluation of interactions between particles, including those in different domains (different MPI processes, for example).

In practice, there are many additional details the user should give. Consider a relatively simple case of particles interacting with softened 1/r potential. There are a number of small but important points one has to decide on. For example, what algorithm should be used for the interaction calculation? Even if we limit the possibilities to reasonably adaptive schemes for open boundary problems, we have the choice between Barnes-Hut tree and FMM. For both algorithms, there are many different ways to parallelize them on distributed-memory parallel computers. Also, there are infinitely many variations for the time integration schemes.

The base layer of FDPS offers the domain decomposition based on the recursive multisection algorithm, with arbitrary weighting function for the load balancing. It also offers the parallel implementation of interaction calculation between particles.

The domain decomposition part takes the array of particles on each node as the main argument. It then generates an appropriate domain for each node, redistribute particles according to their locations, and returns.

The interaction calculation part takes the array of particles, the domain decomposition structure, and the specification of the interaction between particles as main arguments. The actual implementation of this part need to take into account a number of details. For example, the interaction can be of long-range nature, such as gravity, Coulomb force, and interaction between computational elements in the boundary element method (BEM). In this case, the user should also provide the way to construct approximations such as the multipole expansion and the way to estimate error. The interaction might be of short-range nature, with either particle-dependent or independent cutoff length. In these cases, the interaction calculation part should be reasonably efficient in finding neighbor particles.

We have successfully implemented all of these functionalities in FDPS version 1.0. (https://github.com/FDPS/FDPS). Using FDPS, a gravitational N-body simulation code can be written in 120 lines, and that code is actually fully scalable even to full-node runs on K computer. For SPH calculations, we have also achieved similar scaling.

FDPS is implemented as a class template library in C++ language. It receives the class definition of particles and a function (or multiple functions in the case of complex interactions) to evaluate the interaction between particles. When a user program is compiled with the FDPS library, the class template is instantiated with the user-specified definition of the particle class. Thus, even though the FDPS library functions are generic ones not specialized to a particular definition of particles, it behaves as if it is a specialized one.

The measured performance of applications developed using FDPS is quite good. Both for gravity-only calculation and SPH calculation, weak-scaling performance is practically perfect, up to the full-node configuration of K computer. Moreover, the measured efficiency, in terms of the fraction of the peak floating-point performance, is also very high. It is around 50% for gravity-only calculation. For SPH calculations, at the time of writing the performance is around 10%.

In FY 2015, we have extended FDPS in several important directions. The first one is the improvement of the strong scaling. The algorithm used for the domain decomposition contains one serial bottleneck. The "sampling" algorithm used in FDPS 1.0 works well only when the average number of particles per MPI process is significantly larger than the total number of MPI processes. We developed a new parallel algorithm, in which $O(p^{1/3}$ MPI processes are used to decompose the computational domain. Here p is the total number of MPI processes. Thus now the requirement for the number of particle is relaxed from larger than p to larger than $p^{2/3}$. Now we can achieve pretty good performance for around 1 billion particles, on the full nodes of K computer. Previously we need near 100 billion particle to achieve good efficiency.

The second one is the addition of new interface method to interaction calculation function, which allows efficient use of accelerator hardware such as GPGPU or Intel MIC. In order to achieve high performance on accelerators, it is important to pass a large chunk of work at one time. In order to achieve this goal, in the current version of FDPS the CPU creates the list of multiple interaction lists, and send all of them at once so that the overhead of the initialization of the accelerator would not become a bottleneck. This interface has been tested on NVIDIA GPGPUs as well as the PEZY-SC processor.

In FY 2016, we have released FDPS 3.0. The most important new feature of this release is the interface to application programs written in Fortran. FDPS itself is implemented using C++. The reason why we adopted C++ is to use its "template" functions. Using templates, we can write library functions which accept user-defined data types as template arguments. This means we can effectively generate "specialized" libraries for user-specified particle data types, without knowing the data types beforehand.

In FY 2017, we have released several new versions of FDPS, up to 4.0a. There are a number of improvements, mostly for improved performance, For example, we have implemented the reuse of the interaction list. In the

case of the calculation of short-range interactions, the neighbor-list method or so-called bookkeeping method has been used in many applications. On the other hand, to our knowledge, such a method has not been applied to Barnes-Hut treecode or FMM. There is no fundamental difficulty in combining the two methods, and the reason why such a combination has not tried before is probably it was not really necessary. The calculation cost of constructing the tree structure and traversing the tree to construct the interaction lists is, int the case of Barnes-Hut algorithm, a small fraction of the total calculation cost. Thus, it is usually unnecessary to try to reduce the cost of the tree construction and tree traversal.

However, some of recent high-performance computers have rather extreme ratios in various aspects, and thus require the performance improvements which were not necessary. One example is the Sunway SW26010. It's architecture is rather extreme in two aspects. First, its "core group" consists of one MPE (management processing element) and 64 CPEs (computing processing elements). MPE has data cache and runs the primary thread, and CPEs do not have cache. Thus, it is difficult and very time-consuming to develop the program which runs on CPEs, in particular for complex operations like tree construction. On the other hand, MPE is very slow compared to CPE, and thus we need to minimize the computational work of MPE.

Another aspect is the rather low memory bandwidth. The B/F number of SW26010 is around 0.03, which is around 1/15 of that of K computer. Thus, in order to achieve reasonable performance on SW26010, we need to minimized the main memory access per timestep.

The reuse of the interaction list turned out to be very effective on SW26010 and other machines with relatively low memory bandwidth, such as NVIDIA P100/V100 and PEZY-SC2.

In FY 2018, we have improved the API of FDPS and also worked on further performance improvement. Concerning the improvement of API, we added the API in the C language in FDPS 5.0. Before 5.0, user programs should be written either C++ or Fortran. Although these two languages cover a fair fraction the needs of of HPC users, it is clearly desirable to have API in C language, since that would allow users to write programs not only in C languages but also in any other languages with FFI (foreign function interface), since FFI is usually defined in C. The C language API works much in the same way as Fortran API.

In FY 2019, we have worked mainly on new functionalities for FDPS. The first one is the support of PM^3 (Particle-Mesh Multipole) method. As its name suggests,PM^3 method is a combination of Particle-Mesh method and Fast Multipole method. In the Particle-Mesh method, long-range and short-range interactions are separated using smooth splitting function, and the long-range interaction is calcualted using FFT. In the case of PM^3 method, Long-range interaction is defined as the interactions calculated using the tree higher than a certain level in FMM. Then, instead of using hieratchical FMM method, the long-range interaction is obtained as the convolution of the muitipole expansions and Green's function, and then FFT is used to evaluate convolution. Compared to the traditional particle-mesh (or even particle-mesh Ewald) method, PM^3 method can achieve much higher accuracy with significantly smaller amount of communication, and thus suited to large-scale parallel machines with relative weak network. We also worked on the optimization of FDPS on Fugaku. Compared to K, Fugaku has relatively weak network and CPU core with large latency for arithmetic operations. Thus, bottlenecks appear in several unexpected places. We have implemented various new algorithms to remove these bottlenecks.

11.4 Schedule and Future Plan

We plan to improve the performance of FDPS further in FY 2020. The main issue will be the performance tuing on Fugaku supercomputer. Other issues include the automatic generation of high-performance computing kernel for particle-particle interactions.

11.5 Publications

11.5.1 Articles/Journal

[1] Iwasawa, M., D. Namekata, K. Nitadori, K. Nomura, L. Wang, M. Tsubouchi, and J. Makino, *Accelerated FDPS: Algorithms to use accelerators with FDPS*, Publications of the Astronomical Society of Japan, 2020, **72**,.

[2] Ebisuzaki, T., H. Katori, J. Makino, A. Noda, H. Shinkai, and T. Tamagawa, *INO: Interplanetary network of optical lattice clocks*, International Journal of Modern Physics D, 2019, **29**,.

[3] Hirai, Y., S. Wanajo, and T. R. Saitoh, *Enrichment of Strontium in Dwarf Galaxies*, The Astrophysical Journal, 2019, **885**,.

[4] Hirai, Y., T. R. Saitoh, S. Wanajo, and M. S. Fujii, *Enrichment of Heavy Elements in Chemo-Dynamical Simulations of Dwarf Galaxies*, Dwarf Galaxies: From the Deep Universe to the Present, 2019, **344**, 197-200.

11.5.2 Software

[5] FDPS github.com/FDPS
[6] Formura github.com/Formura

11.5.3 Patents