Chapter 5
Acceleration of Classical Molecular Dynamics Simulations

Y. Andoh, N. Yoshii, J. Jung and Y. Sugita

Abstract In this chapter, we describe the acceleration and parallelization in classical molecular dynamics simulations. As electrostatic interactions are computationally intensive, the importance of the particle mesh Ewald (PME) method and the fast multipole method (FMM) will increase. These methods will be described here. In addition, general techniques for hierarchical parallelization on the latest general-purpose supercomputers (especially connected by a three-dimensional torus network), together with the critical importance of the data array structure, are explained. We show the optimization and benchmark results in the parallel environments of the molecular dynamics calculation programs, MODYLAS and GENESIS.

5.1 Classical Molecular Dynamics Simulations

Molecular dynamics (MD) calculation is a method to investigate the thermodynamic properties, structures, and dynamics of molecular assemblies, such as liquids, solids, gases, glasses, polymers, and biomolecules. By numerically solving the equations of motion of all atoms and molecules in the system, we directly obtain their trajectories. When the motion of the atoms follows the equation of motion as described by classical mechanics (Newtonian mechanics), it is called classical MD calculation.

To solve the equations of motion of atoms, it is necessary to obtain the force acting on each atom. When the force is sufficiently accurate and a numerical solution of the equation of motion is obtained with sufficient precision, it can be regarded that the

Y. Andoh · N. Yoshii (✉)
Center for Computational Science, Graduate School of Engineering, Nagoya University,
Furo-cho, Chikusa-ku, Nagoya, Aichi 464-8603, Japan
e-mail: yoshii@ccs.engg.nagoya-u.ac.jp

N. Yoshii
Department of Materials Chemistry, Nagoya University, Furo-cho, Chikusa-ku, Nagoya,
Aichi 464-8603, Japan

J. Jung · Y. Sugita
RIKEN Advanced Institute for Computational Science, Kobe, Japan

M. Geshi (ed.), *The Art of High Performance Computing for Computational Science, Vol. 2*,
https://doi.org/10.1007/978-981-13-9802-5_5

trajectory of the atom is equivalent to that of the real system. Various thermodynamic quantities and statistical mechanical functions can be obtained from the trajectories of the atoms. We can discuss the physics and chemistry of the target system.

Due to the recent widespread use of highly parallel computers, target systems for MD calculations become larger. MD calculations are now used in various fields, including materials science and bioscience. The development of high-precision general-purpose potential functions, such as AMBER [1], CHARMM [2], OPLS [3], and the development of high performance and multifunctional free software, such as GROMACS [4], NAMD [5], LAMMPS [6], has made it easier to perform high-quality MD calculations. However, using massively parallel machines with more than 1,000 nodes, it is difficult to efficiently execute these programs because of the reduction in parallelization efficiency caused by internode communication. It is necessary to choose algorithms and optimize data structures in programs to maintain high parallel efficiency in highly parallel environments. By applying various techniques, we can use the high performance of a massively parallel computer. In this chapter, we will explain the various algorithms and schemes used in the MD calculation software, MODYLAS [7], and GENESIS [8] developed by the authors, suitable for the massively parallel computers.

5.1.1 Molecular Dynamics Calculation Flow

Here, we outline the MD calculation flow (Fig. 5.1). First, we set the initial conditions. The coordinates and velocities of atoms are input as an initial condition. When atomic coordinates are obtained experimentally, their structures should be used as the initial coordinates. As a preparation for MD calculation, addition of the solvent molecules and energy optimization is applied, if necessary. In the simulation of liquids, we first prepare a lattice structure as an initial configuration. We obtain the equilibrated liquid structure by performing a high-temperature MD simulation. The initial velocity of each atom is assigned to reproduce a Maxwell distribution corresponding to the target temperature. In addition, we must set up the calculation conditions for each MD calculation, such as the parameters for mass and intermolecular interaction of each atom, statistical ensembles concerning temperature and pressure control, number of MD steps, time steps, and so on.

Next, we calculate the forces acting on each atom, which are assigned to intramolecular interactions acting through chemical bonds within the molecule and intermolecular interactions acting between atoms in different molecules or distant atoms in the same molecule. This intramolecular interaction consists of stretching potential which is a two-body interaction acting between bonded atoms, bending potential which is a three-body interaction, torsional potential which is a four-body interaction, and so on. The computational complexity of these interactions is $O(N)$, where N is the number of atoms in the system. However, the calculation amount for intermolecular interactions, such as electrostatic and van der Waals interactions, is $O(N^2)$. As a system becomes large, the computational amount increases drastically.

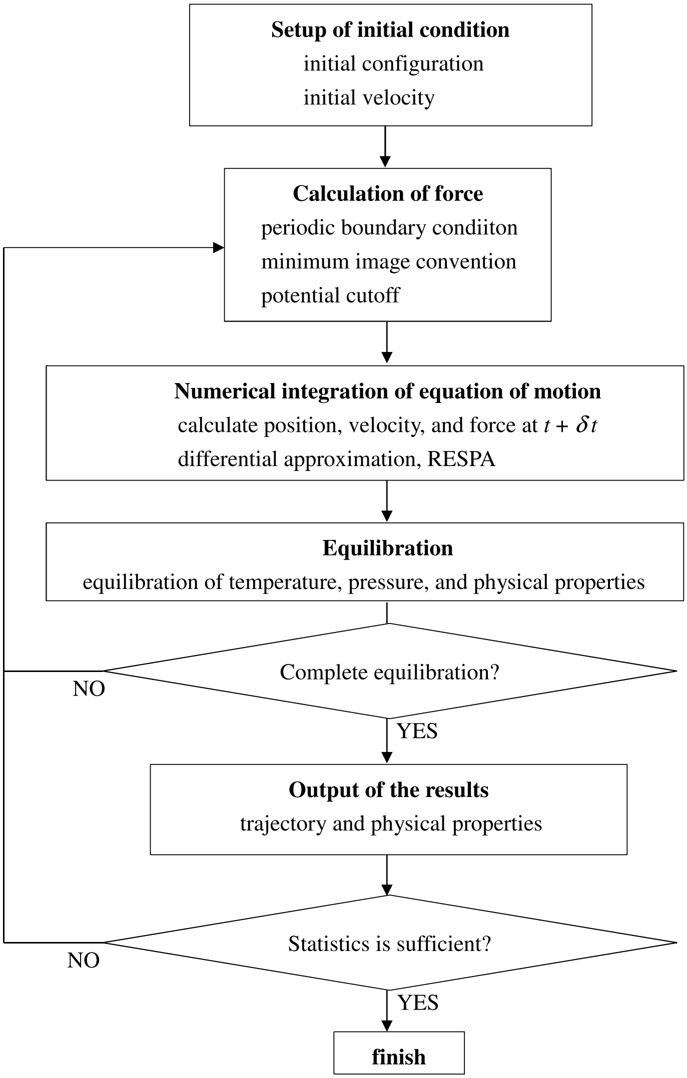

Setup of initial condition
initial configuration
initial velocity
Calculation of force
periodic boundary condiiton
minimum image convention
potential cutoff
Numerical integration of equation of motion
calculate position, velocity, and force at $t + \delta t$
differential approximation, RESPA
Equilibration
equilibration of temperature, pressure, and physical properties
Complete equilibration?
NO
YES
Output of the results
trajectory and physical properties
Statistics is sufficient?
NO
YES
finish

Fig. 5.1 MD calculation flow

The intermolecular interaction is the important factor in determining the size of the target system. Various techniques have been developed to accelerate MD calculations. This chapter focuses on the acceleration of intermolecular interaction calculations.

The numerical integration of the equation of motion is performed using the intermolecular force at time t, and the new velocity and position of the atoms at time $t + \Delta t$ are obtained. Difference approximation and predictor-corrector methods have been often used as numerical integration algorithms for the equation of motion [9, 10]. In

Sect. 5.1.4 , we discuss the reference system propagator algorithm (RESPA) method, which enables multiple time steps and efficient numerical integration.

In the numerical integration of the equation of motion, rapid intramolecular degrees of freedom are often removed. For example, stretching and bending motions in water molecules are not so important, and they are often treated as rigid molecules that eliminate intramolecular degrees of freedom. The motion of the water molecules can be decomposed as the translational motion of the center of mass and rotational motion around the center of mass. Each motion can be obtained by numerically integrating Newton's equation of motion and Euler's equation [9, 10]. Alternatively, constraint dynamics may be used where the equations of motion are numerically solved with distance- or angle-constraints among atoms. By these methods, it is possible to introduce a larger time step Δt, which greatly improves the efficiency of the numerical integration of equations of motion in MD simulations. In Sect. 5.1.5, we discuss the constraint dynamics.

The physical quantity can be calculated using the position and velocity obtained from MD calculations. If the MD calculations reach an equilibrium state where the physical quantity of interest fluctuates around a certain value, the equilibration from the initial configuration is completed. The MD calculations are continued and we output the molecular trajectory and physical quantity at each step. If sufficient statistics are obtained, the MD calculations are terminated.

5.1.2 Algorithms to Reduce the Calculation Amount for Nonbonded Interactions

In general, the amount of calculation for nonbonded interactions is considerably larger than that for bonded interactions. The Lennard-Jones (LJ) interaction is composed of $(1/r_{ij})^{12}$ and $(1/r_{ij})^6$, where both converge rapidly to zero with r_{ij} and r_{ij} is the distance between the atom i and j. The LJ interaction can be truncated if r_{ij} is greater than a cutoff distance r_{cut}. Typically, r_{cut} is chosen to be $3\sigma - 4\sigma$, where σ is the LJ parameter that corresponds to the diameter of the atom.

By using a cutoff technique, most of the candidate j atoms located in a region distant from the i atom can be omitted without calculating the r_{ij} values. There are two established methods for this purpose: i.e., the Verlet neighbor list and linked-list cell (LLC) methods [11].

In the Verlet neighbor list method, a list for candidate j atoms is stored for each i atom by using a longer cutoff radius, $r'_{\text{cut}} = r_{\text{cut}} + \Delta r_{\text{cut}}$, than the actual r_{cut}. The list is updated per tens of MD steps. In the LLC method, a calculation unit cell is divided into subregions (subcells). Each atom is assigned to one of the subcells according to its position. The subcells where the candidate j atoms may be included are listed using the distance between subcell centers, which can reduce the search range of the candidate j atoms. The interaction is calculated between i and j atoms by applying a cutoff. These two methods are widely used in MD calculation programs in which the LLC method is sometimes used in preprocessing to create the neighbor list effectively.

However, the electrostatic interaction is a function of $(1/r_{ij})$, which converges very slowly to zero with r_{ij}. Thus, the cutoff technique causes serious artifacts in the calculation results [12], even though a relatively large r_{cut} value is chosen. To date, some algorithms to reduce the calculation cost of the electrostatic interaction have been suggested without losing the accuracy of the calculated potential energy and forces. We will explain some of these algorithms below.

5.1.3 Acceleration of Electrostatic Interactions

In most MD calculations including all biomolecular systems, the electrostatic interaction calculation is the largest hot spot. When the electrostatic interaction is calculated directly, the calculation complexity is $O(N^2)$. If a cutoff is used, the computational cost becomes $O(N)$, but a discontinuous and large change of force and potential appears at the cutoff length, and a significant artifact occurs in the calculation result. Therefore, it is not permissible to use the cutoff.

The Ewald method has been used for a long time to evaluate the electrostatic interaction under periodic boundary conditions efficiently [9, 11]. In this method, the electrostatic potential $1/r$ is multiplied by the error and complementary error functions. The sum of these two functions becomes 1. With increasing x, the error function, erf(x), attenuates slowly and the complementary error function, erfc(x), attenuates quickly. The cutoff can be used for the latter. For the former, the function can be expressed as a sum of wavenumber vectors by a Fourier series. The sum of this Fourier series is obtained through fast convergence and does not require so many terms. However, the calculation amount increases as $O(N^{2/3}) \sim O(N^2)$. Therefore, it is difficult to apply to large-scale systems. Currently, the particle mesh Ewald (PME) method, which efficiently performs Fourier series, partly using the fast Fourier transform (FFT), is the mainstream technique [13, 14]. In the PME method, the point charges distributed in the MD cell are allocated lattice points using the interpolation technique to reproduce the electrostatic potential in the system. The contribution of the reciprocal-space of the Ewald method can be evaluated by the computational complexity $O(N\log N)$ by using the FFT for the charge on this lattice. However, care must be taken when executing the FFT in a highly parallel environment. Generally, the FFT requires all-to-all communication. In massively parallel computers of 1,000 nodes or more, all-to-all communication is often a major hurdle. This problem is successfully overcome by the GENESIS software described later, which achieves high performance [8]. On the other hand, the FMM [15] and multilevel summation method [16] were developed as non-FFT algorithms. The FMM with a periodic boundary condition is described in detail in Appendix.

5.1.4 Acceleration by Multiple Time Steps

In the numerical integration of the equation of motion, the time span for MD steps is determined by the fastest atomic motion in the system. Slow degrees of freedom must also be integrated by small time steps, which is quite inefficient. It is desirable to separate the degrees of freedom according to the rapidity of the molecular motion and integrate the equations of motion separately according to their suitable time steps. Multiple time steps enable this calculation.

First, the RESPA method [10], which is a numerical integration method that enables multiple time steps, is described. m_i, $\boldsymbol{r}_i$, and $\boldsymbol{p}_i$ are the mass, coordinate, and momentum of atom i, respectively. $V(\boldsymbol{r}^N)$ is the potential energy of the system. When the Hamiltonian is given by

$$H(\boldsymbol{r}^N, \boldsymbol{p}^N) = \sum_{i=1}^{N} \frac{{\boldsymbol{p}_i}^2}{2m_i} + V(\boldsymbol{r}^N), \tag{5.1}$$

the Liouville operator corresponding to Eq. (5.1) is

$$iL = \sum_{i=1}^{N} \left(\frac{\boldsymbol{p}_i}{m_i} \frac{\partial}{\partial \boldsymbol{r}_i} + \boldsymbol{F}_i \frac{\partial}{\partial \boldsymbol{p}_i} \right). \tag{5.2}$$

The time evolution operator for time step Δt is given by $e^{iL\Delta t}$ [10]. The above Liouville operators are composed of two noncommutative operators $iL_1 = \sum_{i=1}^{N} \frac{\boldsymbol{p}_i}{m_i} \frac{\partial}{\partial \boldsymbol{r}_i}$ and $iL_2 = \sum_{i=1}^{N} \boldsymbol{F}_i \frac{\partial}{\partial \boldsymbol{p}_i}$. When the time evolution operator $e^{iL_1\Delta t + iL_2\Delta t}$ is symmetrically decomposed by Suzuki-Trotter expansion, it can be approximated as

$$e^{(iL_1+iL_2)\Delta t} = e^{iL_2\frac{\Delta t}{2}} e^{iL_1\Delta t} e^{iL_2\frac{\Delta t}{2}} + O(\Delta t^3). \tag{5.3}$$

By applying this to $\boldsymbol{r}_i(0)$ and $\boldsymbol{p}_i(0)$ at time $t = 0$, the well-known velocity Verlet method,

$$\boldsymbol{r}_i(\Delta t) = \boldsymbol{r}_i(0) + \Delta t \frac{\boldsymbol{p}_i(0)}{m_i} + \Delta t^2 \frac{\boldsymbol{F}_i(0)}{m_i} \tag{5.4}$$

$$\boldsymbol{p}_i(\Delta t) = \boldsymbol{p}_i(0) + \Delta t \frac{\boldsymbol{F}_i(0) + \boldsymbol{F}_i(\Delta t)}{m_i} \tag{5.5}$$

is obtained. For Hamiltonian dynamics systems, the numerical solution as described above becomes a symplectic integrator. The coordinates and the momentum are the canonical variables, and the discretized expression (5.4) has the conservative quantity called shadow Hamiltonian. Numerical integration is known to be stable over a long period of time [9, 10].

Next, iL is divided into fast iL_{fast} and slow degrees of freedom iL_{slow}. Then, the time evolution operator $e^{iL\Delta t}$ in Eq. (5.4) can be divided as

$$\begin{aligned} e^{(iL_1+iL_2)\Delta t} &\approx e^{iL_{\mathrm{fast}}\frac{\Delta t}{2}} e^{iL_{\mathrm{slow}}\Delta t} e^{iL_{\mathrm{fast}}\frac{\Delta t}{2}} \\ &= \left(e^{iL_{\mathrm{fast}}\frac{\delta t}{2}}\right)^n e^{iL_{\mathrm{slow}}\Delta t} \left(e^{iL_{\mathrm{fast}}\frac{\delta t}{2}}\right)^n \end{aligned} \tag{5.6}$$

where $\Delta t = n\delta t$, δt is the time step for the fast degree of freedom, and Δt is the time step for the slow degree of freedom. In Eq. (5.6), on the one hand, the fast part is divided into n so that we calculate the fast motion in a small time step. On the other hand, the part that moves slowly does not calculate the interaction. In general, the calculation cost of the slow part is higher; therefore, multiple time steps are very efficient. With respect to $e^{iL_{\mathrm{fast}}\frac{\Delta t}{2}}$ and $e^{iL_{\mathrm{slow}}\Delta t}$ in Eq. (5.6), $\boldsymbol{r}_i$ and $\boldsymbol{p}_i$ are updated according to the formula of the velocity Verlet method represented by Eq. (5.4).

Here, division of the Liouville operator can be done for each degree of freedom. However, there is no restriction on this division. For example, it is possible to separate total force acting on an atom into fast changing force caused by stretching, bending, and torsional motions, and a slowly changing one by LJ and electrostatic interactions.

In multiple time steps, a force acts on the atom for every δt and another force acts for each Δt. A certain node in the system may resonate with the force acting on every Δt. Some atomic motions may become unstable or the trajectory may greatly differ from the original trajectory [17]. In recent years, various schemes have been proposed to avoid this phenomenon [18]. It is necessary to pay attention to whether such things have happened.

5.1.5 *Constraint Dynamics*

The degrees of freedom in the molecule are often removed and the molecule is treated as a rigid body. For example, water molecules have three intramolecular degrees of freedom. By fixing the distance between oxygen and hydrogen atoms, the OH stretching motion can be removed. Furthermore, when the distance between the two hydrogen atoms is fixed, we can remove the bending motion of ∠HOH. By constraining these degrees of freedom, Δt can be increased and efficient calculation can be performed.

There are two methods to constrain the intramolecular degree of freedom. One is to handle a molecule as a rigid rotor model and the other is to use constraint dynamics. Here we describe constraint dynamics, which is applicable to various constraints and widely used in general-purpose MD calculation software.

Now, we consider that there is a chemical bond in a molecule and its bond length is fixed. In constraint dynamics, a constraint condition, i.e., the distance between atoms i and j is constant, is imposed. The binding force between atoms is considered to act on atoms to satisfy the constraint condition,

$$g = \left(\boldsymbol{r}_i - \boldsymbol{r}_j\right)^2 - {d_{ij}}^2 = 0. \tag{5.7}$$

The constraint condition expressed by Eq. (5.7) is called a holonomic constraint. The equation of motion of the atom is

$$m_i \frac{d^2 \boldsymbol{r}_i}{dt^2} = \boldsymbol{F}_i + \lambda \frac{\partial g}{\partial \boldsymbol{r}_i}, \tag{5.8}$$

which includes the constraint force $\lambda \frac{\partial g}{\partial \boldsymbol{r}_i}$ by the constraint condition g. Here, λ is an undetermined Lagrange multiplier. When this equation of motion is numerically integrated according to the velocity Verlet method, the position coordinate at time $t = \Delta t$ is

$$\begin{aligned} \boldsymbol{r}_i(\Delta t) &= \boldsymbol{r}_i(0) + \Delta t \frac{\boldsymbol{p}_i(0)}{m_i} + \frac{\Delta t^2}{2m_i} \left\{ \boldsymbol{F}_i(0) + \lambda \frac{\partial g}{\partial \boldsymbol{r}_i} \right\} \\ &= \boldsymbol{r}'_i(\Delta t) + \frac{\Delta t^2}{2m_i} \lambda \frac{\partial g}{\partial \boldsymbol{r}_i} \end{aligned} \tag{5.9}$$

where $\boldsymbol{r}'_i(\Delta t) = \boldsymbol{r}_i(0) + \Delta t \frac{\boldsymbol{p}_i(0)}{m_i} + \frac{\Delta t^2}{2m_i} \boldsymbol{F}_i(0)$. The new position $\boldsymbol{r}_i(\Delta t)$ must satisfy the constraint condition g. By substituting $\boldsymbol{r}_i(\Delta t)$ into Eq. (5.7), we have

$$g = \left[\left\{ \boldsymbol{r}'_i(\Delta t) + \frac{\Delta t^2}{2m_i} \lambda \frac{\partial g}{\partial \boldsymbol{r}_i} \right\} - \left\{ \boldsymbol{r}'_j(\Delta t) + \frac{\Delta t^2}{2m_j} \lambda \frac{\partial g}{\partial \boldsymbol{r}_j} \right\} \right]^2 - d_{ij}^2 = 0. \tag{5.10}$$

Since this is a quadratic equation for λ, we can evaluate the constraint force $\lambda \frac{\partial g}{\partial \boldsymbol{r}_i}$ by solving for λ.

This is a simple case where the constraint on each atom is one. In the case of two or more constraints on one atom, the Eq. (5.10) is replaced by simultaneous quadratic equations including the same number of undetermined multipliers λ as the number of constraints. When the target molecule becomes large, such as macromolecules and proteins, the constrained atoms are connected to each other; therefore, the simultaneous equations are increased and the calculation cost for obtaining λ becomes very high. Thus, in general, Eq. (5.10) is linearized by ignoring the λ^2 term. Iterative calculations are performed until the constraint condition g is satisfied. This is the SHAKE method [19], which is widely used in general-purpose MD software. Various schemes to accelerate the convergence of iterative calculations in the SHAKE method have been proposed [20].

In addition, the velocity constraint was also obtained by differentiating Eq. (5.7) with respect to time. A constraint algorithm, the RATTLE method [21] for velocity similarly to the SHAKE method, was developed to satisfy the constraint condition.

Accelerated algorithms for constraint dynamics designed for water molecules, such as LINCS [22] and SETTLE [23], have been developed and are used widely.

These techniques can be introduced separately to the site to be constrained. It is not necessary to introduce new coordinates (e.g., Euler angle, quaternion), which are necessary for a rigid rotor model. Therefore, it is possible to keep the structure of the program simple; it is widely used for many general-purpose MD software. In contrast, a rigid rotor model can increase the time step Δt in MD calculations [24]. As described above, several methods for constraining the intramolecular degree of freedom are available; therefore, we must choose an effective method for the target system.

5.1.6 Developments of Classical MD Simulations Related with HPC

5.1.6.1 Development of Parallelization

The main bottleneck in MD is the evaluation of nonbonded interactions, i.e., electrostatic and van der Waals interactions. Fortunately, the van der Waals interaction decreases rapidly as the interparticle distance increases. Therefore, if a sufficiently large cut-off distance is assigned, energy and force values in the van der Waals interaction can be regarded as zero when the interparticle distance is greater than the defined cutoff distance. Using Ewald summation description for electrostatic energy/force, interactions can be split into those in real- and reciprocal-space, and cutoff value can be assigned like van der Waals case [13, 14]. To deal with interactions with long-range distance, we perform energy and force calculation using fast Fourier transform (FFT) in the reciprocal-space. Therefore, it is necessary to consider parallelization of nonbonded interactions in both real- and reciprocal-spaces. Basically, there are three parallelization schemes for the real-space interactions: atomic, force, and spatial decompositions. Parallelization based on the atomic decomposition is a relatively simple algorithm, and historically introduced earliest in MD programs. The algorithm of parallelization based on spatial decomposition is more complicated than the atomic decomposition, but it is currently introduced in most MD programs because of high parallel efficiency. Detailed parallelization schemes based on spatial decomposition are slightly different in each program. NAMD applies a parallelization method combining spatial and force decomposition in the real-space interactions, and furthermore it schedules interactions between subdomains by CHARMM++ [5]. In addition, pencil (two-dimensional) decomposition is used to parallelize the FFT calculation that is required in the reciprocal-space calculation. In DESMOND, spatial decomposition based on the midpoint method is applied for parallelization in the real-space interaction. In addition, all-to-all communications are replaced by butterfly communications in one-dimensional FFT for the reciprocal-space interaction [25, 26]. Parallelization scheme of the real-space interaction in BLUE MATTER is similar to DESMOND, but the volumetric decomposition (three-dimensional decomposition) scheme is used for parallelizing the reciprocal-space interaction [27]. In GENESIS, the midpoint cell method [8] was introduced to improve the performance from the midpoint method, and FFT is parallelized using volumetric decomposition with one-dimensional all-to-all communications [28]. In addition, hybrid parallelization scheme combining OpenMP and MPI is introduced. In CHARMM and GRO-

MACS, a spatial decomposition method based on the eighth-shell scheme is adopted and non-bonded interactions in the real-space and reciprocal-space are performed in different CPU nodes [29, 30]. CHARMM only performs MPI communication, but GROMACS can make use of a hybrid parallelization combining MPI and OpenMP. In some programs, PME is replaced by another long-range force calculation method to increase the parallel efficiency. MODYLAS showed excellent parallelization efficiency on the K computer by introducing the fast multipole method (FMM) [7]. Recently, NAMD introduces a new scheme called multilevel summation method to increase parallelization efficiency [14].

5.1.6.2 Development of MD in Specialized Platforms

In recent years, dedicated computers have become a great influence for speeding up the MD energy/force calculation. MDGRAPE-3 accelerates MD by using specialized hardware in computing non-bonded interactions [31]. In ANTON [32], developed by D. E. Shaw Research, specialized hardware is used for the entire MD simulation. Reciprocal-space interactions are accelerated by using the Gaussian-split Ewald method. By such a contrivance, ANTON achieved 100 times faster performance. However, MD calculation using ANTON has limitations on the size of the molecular system. It is difficult to calculate a large-scale molecular system exceeding 1 million atoms with ANTON. In the latest ANTON 2, this restriction was relaxed, and high-speed calculation was possible for 2 million atoms system [33].

5.1.6.3 Development of MD for GPUs

An important development in MD from the viewpoint of hardware utilization is the use of a graphics processing unit (GPU). GPUs were originally developed as arithmetic chips dedicated to graphics, but today they have been used for various scientific and technical calculations. MD program using GPUs is roughly divided into two categories. The first one is to send all necessary information to GPU and perform all operations with GPUs. This method is adopted in AMBER [34], ACEMD [35], OpenMM [36], MOIL [37], DESMOND, and so on. Despite this method is very efficient when single node is used, there are restrictions in the system size that can be calculated due to the memory limitation of the GPU. It is also difficult to parallelize on multiple computers. The other method is to divide the overall calculation into a computation-intensive part, that is, calculation of the real-space non-bonded interactions and a communication-intensive part including reciprocal-space interaction with FFT. The former is mainly computed by GPUs and the latter is computed by CPUs. In this case, communication between the CPU and the GPU is required at every integration step. NAMD [38], GROMACS [39], and GENESIS [40] adopt this method. In this method, parallelization across multiple computers can be performed, so it is possible to handle a large molecular system. On the other hand, when using a single node, the calculation speed of the latter is slower than computing using only GPUs.

5.2 Hierarchical Parallelization in the Latest Supercomputers

5.2.1 Hierarchical Hardware Structure

The latest supercomputers have a hierarchical structure for not only their calculation units, but also their devices to store calculation data.

Calculation units are composed of thousands or tens of thousands of calculation nodes, which are connected by a high-speed internode network. A calculation node sometimes has plenty of CPUs. The independent arithmetic units inside the CPUs are called the cores. In each core, units to execute various kinds of arithmetic operations are installed. Modern CPUs are also equipped with vector operation units (the same arithmetic operations on inputted multiple data), which work by single-instruction-multiple-data (SIMD) processing. A coprocessor to accelerate a part of the calculation is optionally installed onto the PCI-e bus at the same level as the CPUs, which is beyond the scope of this section.

As data storage devices, the main memory or random-access memory (RAM) devices (with tens of gigabytes (GB) of memory) are installed on each calculation node and accessed by the CPU(s). When one node has more than two CPUs, the access speed from each CPU to each main memory is mostly nonuniform; this is called nonuniform memory access (NUMA) architecture. In each CPU, a small but high-speed data storage device called cache memory is shared by several of the cores. These caches have levels: i.e., the level 1 cache with tens of kilobytes (kB) is the nearest cache from the arithmetic unit, and the level 2 cache with hundreds of kB is the second nearest one. The lowest data storage module is a processor register, which connects directly to the arithmetic units in the CPU, such as floating-point arithmetic unit (FPU). The data access speed of this module is the fastest, but it only stores a fixed amount of one integer or floating-point data. Vector registers can treat a large amount of data for SIMD operations.

Any parallelized software aiming at high parallelization efficiency must be designed considering the hierarchical structures of both calculation units and data storage devices in modern supercomputer systems.

5.2.2 Parallelization Strategy

Parallel execution between calculation nodes involves a distribution of calculation data onto the main memory devices on each calculation node. In standard operations, such a parallelized execution is realized by the message-passing interface (MPI) by inserting additional MPI functions onto the original code. The unit of parallelization by the MPI is called the MPI process. In the case of several CPUs in one node or one CPU with the NUMA structure, allocating one MPI process onto one CPU or one NUMA unit often provides better parallelization efficiency.

Next, in parallel execution between the cores, the required calculation data on the cache(s) can be shared between the cores. This style of parallelized execution is usually realized by the OpenMP language extension, which is a standard component of modern Fortran/C language compilers. By adding a set of OpenMP directives before and after a DO loop (Fortran) or for loop (C), the arithmetic operations in the loop are distributed among the parallelized units (threads). Typically, one thread is assigned to one physical core.

Finally, when the CPU equips vector arithmetic devices, parallel execution by the SIMD instruction is possible for the deepest loops. For highest parallelization efficiency, it is essential to code a series of arithmetic operations at hot spots by vectorized assembler language or in the intrinsic instructions, although this requires high coding skills. A second-best method is to use the compiler's automatic SIMD parallelization function, in addition to the vectorized messages provided by the compiler. From these messages, we can observe which loops are vectorized, and if not vectorized, what elements inhibit SIMD parallelization of the loop. Better SIMD efficiency could be obtained by modifying the loop structure of problematic loops repeatedly.

Consider that hierarchical parallelization starts from the lowest MPI level, followed by the thread level, to the SIMD processing because the efficiency of the higher level parallelization is strongly restricted by the manner of implementation of the lower level parallelization. In particular, the parallelization efficiency of the code is dominated not by arithmetic efficiency (i.e., equal partitioning of tasks among parallelized units), but by the data transfer efficiency between hierarchical data devices. If the access time to data on a register is 1, it is 10 for data on the cache(s), and 100 on the main memory. Therefore, without establishing a smoothed data-flow path from the main memory to the register, an ideal parallelization efficiency could not be obtained at higher levels. Arithmetic devices waste time waiting for data to be calculated.

Section 5.3.2 describes an example of a sophisticated data structure that enables a smoothed data-flow path for MPI/OpenMP/SIMD hierarchical parallelization on supercomputer systems with a three-dimensional (3D) torus network.

5.2.3 Parallelization Techniques Based on Torus Network Characteristics

5.2.3.1 A Torus Network

A torus network is a network to connect calculation nodes in series and circularly along one direction. A 3D torus network is comprised of calculation nodes placed at 3D lattice points connected by a torus network along the x, y, and z axes. There can be more than three dimensions of network connections; e.g., the Tofu interconnect developed by Fujitsu Co. Ltd. provides a six-dimensional torus network for supercomputer systems with high fault tolerance to network troubles. A torus network is superior to other network structures, such as fat-tree networks from the following perspectives.

- High-speed and low-latency data transfer to first neighbor nodes along each axis.
- High affinity to MPI parallelization in spatial domain decomposition style.

However, it also has the following disadvantages:

- Need for relay nodes for communication with nodes other than the nearest neighbors.
- Increase of communication count, if software requires wide range communication.
- Frequent occurrence of collisions between two data transfers in opposite directions, if without a communication procedure design.

As schematically shown in Fig. 5.2a, a direct communication with calculation nodes on the diagonal line can be made possible by a two-step communication; i.e., the first communication along the x-axis followed by y-axis communication with a relay node. The situation becomes more severe when the target node is more distant from

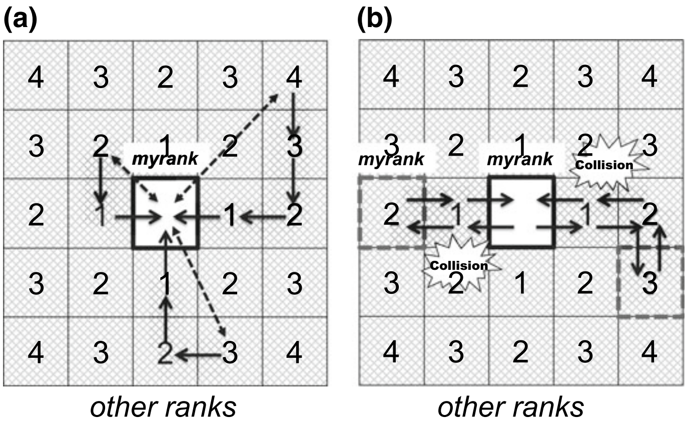


Fig. 5.2 Problems that can occur in a torus network. **a** Increase in communication count and **b** occurrence of collisions between two data transfers in opposite directions

the source node. The integer numbers in Fig. 5.2a indicates a communication count required for data transfer to the centered myrank from each rank. In total, 60 counts of communication are necessary to gather data from the surrounding hashed region to myrank. On the other hand, as shown in Fig. 5.2b, if each communication is not scheduled, collisions of two data transfers in opposite directions occur frequently everywhere. The occurrence of a collision event delays communication time significantly because one communication started after the other communication ended.

For parallelized calculations aiming at strong scalability, such as MD calculations, these delays in communication due to collisions are fatal. Thus, it is strongly desirable to implement MPI parallelization algorithms to minimize the communication counts without any collision events on the 3D torus network.

5.2.3.2 MPI Function Suited to a Torus Network

The MPI_SENDRECV is a MPI function to hide the disadvantages of a torus network as described above and to realize highly effective parallelization by the MPI.

```
call MPI_SENDRECV (sendbuf,scount,stype,dest,stag,
    recvbuf,rcount,rtype,source,rtag,comm,status,ierr)
```

sendbuf	: initial address of send buffer
scount	: element number in sendbuf
stype, stag	: type of elements in sendbuf, send tag
recvbuf	: initial address of receive buffer
rcount	: element number in recvbuf
rtype, rtag	: type of elements in recvbuf, receive tag
dest, source	: rank of destination, rank of source
comm, status, ierr	: communicator, status code, error status

This function executes MPI_SEND and MPI_RECV in one call; i.e., myrank sends scount elements of sendbuf with stype to dest rank, while it receives rcount elements of recvbuf with rtype from source rank. By calling this subroutine from every MPI process along one axis on a torus network, a circularly shifted communication is realized.

Figure 5.3 shows how the MPI_SENDRECV function can hide the disadvantages of a torus network in a two-dimensional (2D) case. Assuming that data on ranks 0–24, except for rank 12 (myrank), are transferred to the myrank. First, by calling the MPI_SENDRECV function, data on each rank are shifted along the $+x$ direction. Integer numbers below s and d in squares are values for rank of source and rank of destination as set in subroutine arguments, respectively. In this process, the function is called from all ranks simultaneously. As a result of these collective MPI communications, any collision events do not occur. Second, after the communication, the transferred data are merged with the originally owned data. The merged data are then shifted again along the $+x$ direction by calling the MPI_SENDRECV function. By applying the same communications along the $-x$ direction, distributed data on the ranks along the x-axis are gathered to the centered ranks (ranks 2, 7, 12, 17, and 22). Third, the merged data within each x line are shifted twice along the $+y$

direction by calling the MPI_SENDRECV function twice with setting source and destination numbers as shown in the right panel of Fig. 5.3. Finally, by shifting the merged data along the $-y$ direction twice in the same manner, all data on ranks 0–24 are transferred to myrank without any collision events.

In this technique, the communication count required to gather the data is only 8, which is 13% of the count for a series of fundamental one-to-one communications as shown in Fig. 5.2a. In a 3D case, the degree of reduction is more remarkable, i.e., from 450 to 12, which is a reduction rate of 97%. Furthermore, with this style of collective communication, all involved MPI processes gather the same range of data at once, which is in most cases a favorable situation for the subsequent arithmetic operations. If the torus network equipment has bidirectional communication architecture, such as the Tofu interconnect, the communication count can be further reduced to half.

For example, MODYLAS [7] software implements this kind of collective communication routine to gather the data of atom coordinates and coefficients of multipole expansions required to calculate potential energy and forces. The same type of communication algorithm is available in a process to send data back to the original processes, such as a scattering of reaction forces by the nonbonded two-body interactions.

The communication technique described here is widely applicable to any MPI parallelized software on a 3D torus network.

5.2.4 Necessity of Sophisticated Data Structure for Highly Efficient Parallelization

As described briefly in Sect. 5.2.2, the parallelization efficiency of hybrid parallelization is determined by the smoothness of data transfers between memory storage devices. First, an element controlling the smoothness of data flow is the data array structure, followed by proper coding at each parallelization level. An ideal data structure for hierarchical parallelization should have the following properties simultaneously:

1. Data are sequentially accessed at the time of arithmetic operation.
2. Data are localized, compartmentalized by small areas, and a data-blocking technique can be applied without using temporary arrays.
3. No data sorting and copying operations are required through a series of MPI communications.

Property 1 is important for the working efficiency of arithmetic units by smoothed data transfer from the cache(s) to registers. If not, software pipelining cannot be applied; therefore, the units waste time waiting for input data. If vector units are assumed, data sequentiality is more important to realize highly efficient vectorized operations. Data sequentiality is not normally achieved in particle-based calculations, such as MD calculations. For example, when the Verlet neighbor list method is adopted for pairwise additive LJ interaction calculations, the access to coordinate

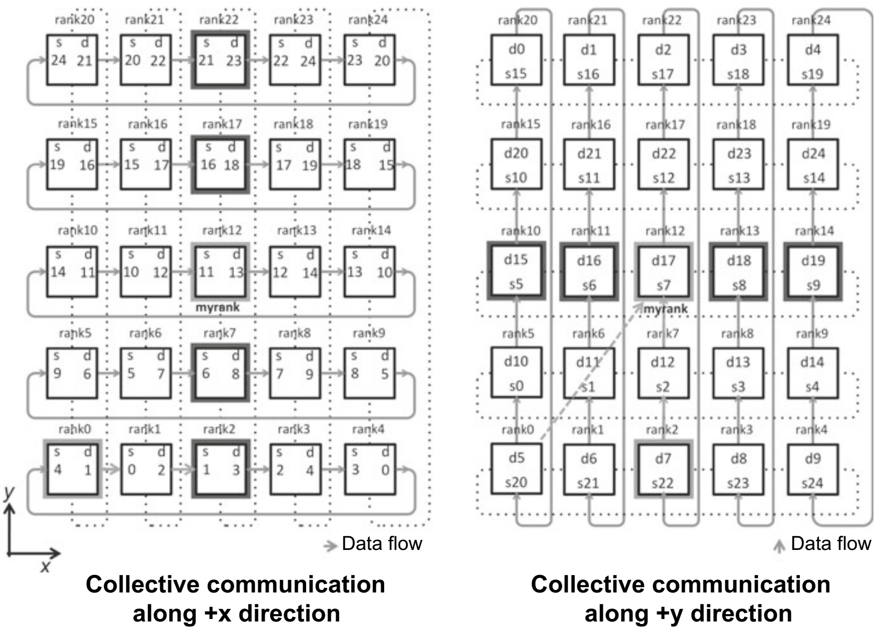


Fig. 5.3 An example of a circularly shifted communication on a 2D torus network realized by the MPI_SENDRECV function. First collective communication along the ±x-axis (left) and successive collective communications along the ±y-axis (right). In each stage, data are shifted by twice to plus or minus direction. Consequently, data on ranks 0–24 are transferred to rank 12 with 4 counts of communication

data based on a list of candidate j atoms becomes random in general, as shown in Fig. 5.4. This is because an arrangement of coordinate data on memory devices does not consider the relative distance between atoms. Much effort is needed to realize sequential access to coordinate data with the Verlet neighbor list method. In contrast, when the cell-link method is used, sequential access on data can be realized relatively easily with a well-designed data structure as described in Sect. 5.3.2.

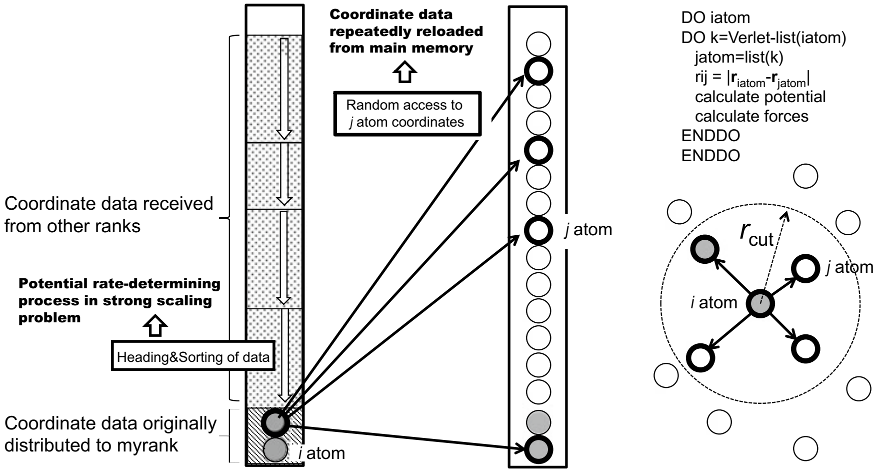
Property 2 is important from the perspective of effective utilization of data on the cache(s) without reloading data from the main memory. In a calculation hot spot, it is desirable to load the data required for a series of arithmetic operations just once from the main memory to the cache. An established method, e.g., the data-blocking technique, is usually adopted for this purpose. The data-blocking technique assumes that a whole data set, such as atom coordinates stored in myrank, is subdivided into smaller data sets. The manner of division is chosen so that accesses to data in one data set are concentrated and data loading is performed with a unit of each data set. Unnecessary data reloading can be reduced by applying some modifications to the nested loop structures.

Property 3 is important from the perspective of strong scaling at process-level parallelization by omitting essentially unnecessary data sorting and duplication operations. Some types of parallelized calculations, such as MD calculations, in their scalability to MPI process number with a fixed problem size (strong scalability) is more important than the scalability by making problem size per process constant (weak scalability). Decreasing the overall elapsed time with large MPI process numbers, an accumulation of small times costed for pre- and post-processing of interprocess communications by the MPI functions could become a rate-determining process in parallelized calculations. This extra wasted time comes from the implementation style, which can be removed by devising the data structure appropriately.

Various data structures could have all of the features listed above, depending on the degree of hierarchy of hardware, kind of interconnections between calculation nodes, and whether coprocessors are used or not. Section 5.3.2 shows an example of the data structure suitable for MD calculations, which is hierarchically parallelized on multicore CPU nodes connected by a 3D torus network.

5.3 MODYLAS

This section introduces general techniques of a hybrid parallelization of large-scale calculations by MPI, OpenMP, and SIMD suited for supercomputers connected by a 3D torus network, with practical examples of implementation of the techniques on MD calculations that adopts the FMM (see Appendix) to calculate the long-range electrostatic interaction under a 3D periodic boundary condition. With a sophisticated data structure, it becomes possible to perform massively parallelized calculations with excellent parallelization and arithmetic efficiency [7].


Coordinate data repeatedly reloaded from main memory
Random access to j atom coordinates
Coordinate data received from other ranks
Potential rate-determining process in strong scaling problem
Heading&Sorting of data
Coordinate data originally distributed to myrank
i atom
j atom
r_{cut}
j atom
i atom

```
DO iatom
DO k=Verlet-list(iatom)
  jatom=list(k)
  rij = $|\mathbf{r}_{iatom}-\mathbf{r}_{jatom}|$
  calculate potential
  calculate forces
ENDDO
ENDDO
```

Fig. 5.4 General issues to inhibit efficient hierarchical parallelization with standard data structure for atom coordinates

5.3.1 Parallelization Characteristics of the FMM

From the perspective of massive parallelization, the FMM is superior to the PME method in the following aspects.

First, the data required to calculate both short- and long-range interaction are spatially localized at each level by its nested structure. Thus, adjacent communications can be used by the MPI instead of all-to-all type communications. In particular, if a 3D torus network is installed onto a supercomputer, the techniques described in Sect. 5.2.3 are directly applicable by minimizing a number of hops and avoiding data-transfer collisions. Second, the boundary of two kinds of potential energy calculations (P2P and L2P operations) is not spherical as in the Ewald method; therefore, access to $\boldsymbol{r}_j$ does not require a list of candidates of $\boldsymbol{r}_j$s. List access could become a barrier to achieving highly efficient vectorized arithmetic operations with SIMD instructions. Third, the FMM assumes a partitioning of a calculation unit cell into subcells; thus, the established LLC method (Sect. 5.1.2) can be used for the P2P operation. Furthermore, because atoms have metadata of the relative position of their belonging subcells, continuous access to data of atom coordinates $\boldsymbol{r}_j$s is possible with their well-designed data structure as shown in Sect. 5.2.4. Finally, since its calculation order is $O(N)$, there is a greater possibility to treat a much larger number of atoms in MD calculations using over 10,000 calculation nodes of exascale supercomputers [41] without losing calculation accuracy. The techniques described in this section can be the basis for such massively parallelized calculations in the near future.

5.3.2 A Metadata Structure of Coordinates and Multipoles

The interatomic interactions treated in MD calculations are distance dependent and reduce to zero with increasing distance between the source and destination atoms. Most of these interactions become completely zero at ten and a few angstroms, except for the electrostatic interaction. Therefore, it is beneficial to add supplementary information about spatial decomposition onto the atom coordinate data, which is one of the *metadata* sets. These metadata are also useful for other primary data used in electrostatic interaction calculations, such as multipoles in the FMM and grid charges in the PME method.

Figure 5.5 shows the data structure of atom coordinates **meta_xyz** with metadata of the relative distances between each subcell. Relative indices rather than absolute indices of each subcell are more favorable as metadata for homogeneous coding among different myranks. The data are represented in two ways: i.e., a one-dimensional representation as a copy of the original array (left panel of the figure) and a multidimensional representation emphasizing the metadata of the relative address of each subcell (right panel of the figure). The data originally stored by myrank are placed in a continuous manner not at the top of the array, but at the center of the array. The blanks arranged above and below the data are storage regions to receive

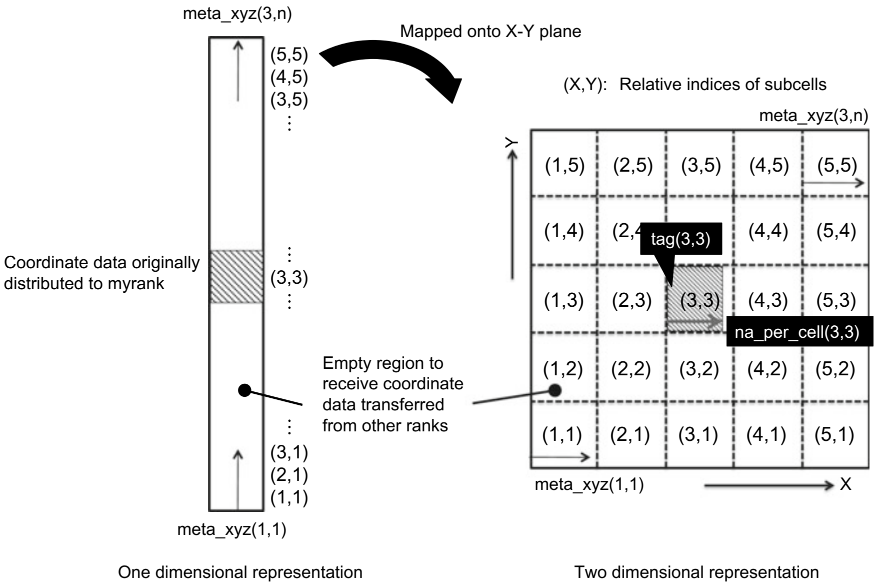


Fig. 5.5 Two kind of data array representations for atom coordinates with metadata structure

coordinates transferred from other surrounding MPI processes. In the multidimensional representation, blanks form a square halo in a 2D representation and a cubic halo in a 3D representation.

Access to the data is always performed using metadata by preparing two additional arrays. The first is a tag array, which stores initial addresses of a series of coordinate data included in each subcell (**tag** in Fig. 5.5), and the second is a counter array, which stores the number of atoms packed in each subcell (**na_per_cell** in Fig. 5.5). The dimensions of these arrays are the same as the metadata information; e.g., (5,5) in Fig. 5.5. With these arrays, the data originally distributed to myrank with an address of (3,3) are accessed by

```
DO i0=tag(3,3), tag(3,3)+na_per_cell(3,3)-1
  meta_xyz(1:3, i0)
ENDDO
```

Data to be transferred by a series of MPI communications are stored as shown in Fig. 5.6. These data are spatially localized around the original myrank data according to the following series of MPI communications. First, data transferred along the ±x-axis are placed at both sides of the original data by connecting its left ($+x$) and right ($-x$) end to the original data. As a result, data blocks are laid along the x-axis in which all coordinate data are arranged serially. Next, data transferred along the ±y-axis with data-block units from (1,3) to (5,3) are placed at the top and bottom of the line at address 3, where voids at both sides of the block are also transferred to avoid any heading and sorting of data in each communication process. With **tag**

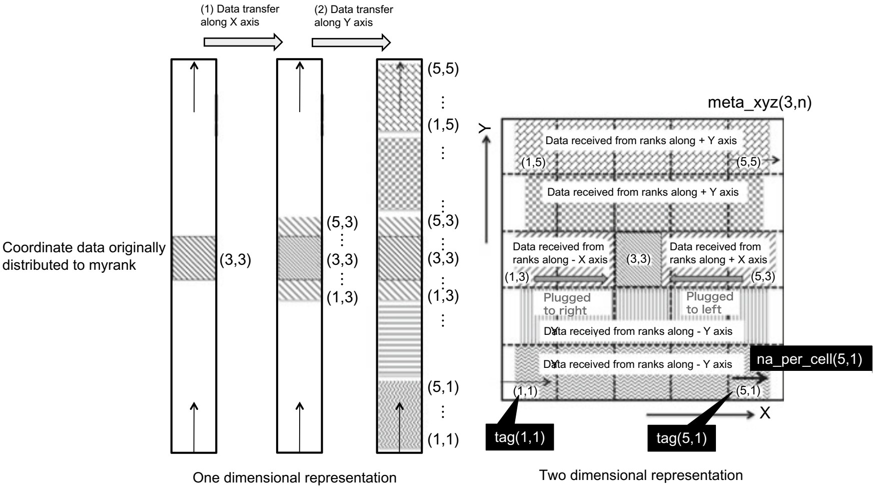

(1) Data transfer along X axis
(2) Data transfer along Y axis
Coordinate data originally distributed to myrank
(3,3)
(5,3)
(3,3)
(1,3)
(5,5)
(1,5)
(5,3)
(3,3)
(1,3)
(5,1)
(1,1)
meta_xyz(3,n)
Y
Data received from ranks along + Y axis
(1,5)
(5,5)
Data received from ranks along + Y axis
Data received from ranks along - X axis
(3,3)
Data received from ranks along + X axis
(1,3)
(5,3)
Plugged to right
Plugged to left
Data received from ranks along - Y axis
Data received from ranks along - Y axis
na_per_cell(5,1)
(1,1)
(5,1)
X
tag(1,1)
tag(5,1)
One dimensional representation
Two dimensional representation

Fig. 5.6 Storage style for coordinate data transferred from other ranks

and **na_per_cell** reconstructed after the communications, any data stored in the halo region can be accessed in the same way as a method to access data on (3,3). For example, the coordinate data packed in a set of five subcells at line address 1 are accessed by

```
DO i0=tag(1,1), tag(5,1)+na_per_cell(5,1)-1
  meta_xyz(1:3, i0)
ENDDO
```

with a completely serial access favorable for the following vectorized arithmetic operations. In addition, a set of five subcells designated according to **tag** and **na_per_cell** with the start and end address of subcell block can be a data-blocking unit without introducing any temporary arrays.

Generally, several subcells are distributed to each rank, depending on the given total subcell number and MPI process. In such a case, the coordinate data are separately stored with a unit of a line along one axis, which is chosen as same as the first axis in a series of MPI communications optimized for a 3D torus network described in Sect. 5.2.3.2 and Fig. 5.3.

For the multipole data, a similar data structure with metadata of relative indices of subcells or supercell addresses is considered.

The M2M operation refers to multipoles on eight subcells or supercells with the same parent cell. The M2L operation requires multipoles on surrounding 875 (=$10^3 - 2^3$) subcells or supercells for each level where interactions with centered 2^3 cells are calculated at the lower level. Within this range of multipole information, the other operation is available. In the torus network and collective communications described in Sect. 5.2.3.2, data not only on target ranks, but also on intermediate ranks, are automatically gathered to myrank. Therefore, it is more efficient to receive

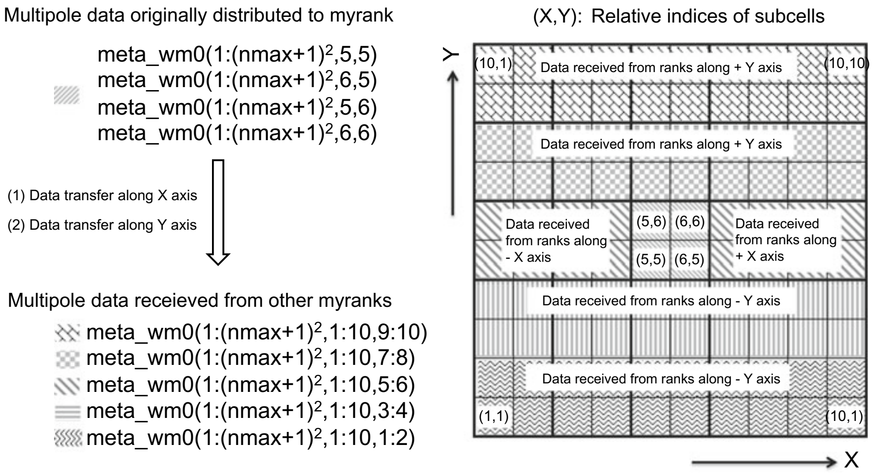

Multipole data originally distributed to myrank
meta_wm0(1:(nmax+1)^2,5,5)
meta_wm0(1:(nmax+1)^2,6,5)
meta_wm0(1:(nmax+1)^2,5,6)
meta_wm0(1:(nmax+1)^2,6,6)
(1) Data transfer along X axis
(2) Data transfer along Y axis
Multipole data receieved from other myranks
meta_wm0(1:(nmax+1)^2,1:10,9:10)
meta_wm0(1:(nmax+1)^2,1:10,7:8)
meta_wm0(1:(nmax+1)^2,1:10,5:6)
meta_wm0(1:(nmax+1)^2,1:10,3:4)
meta_wm0(1:(nmax+1)^2,1:10,1:2)
(X,Y): Relative indices of subcells
Y
(10,1)
Data received from ranks along + Y axis
(10,10)
Data received from ranks along + Y axis
Data received from ranks along - X axis
(5,6)
(6,6)
(5,5)
(6,5)
Data received from ranks along + X axis
Data received from ranks along - Y axis
Data received from ranks along - Y axis
(1,1)
(10,1)
X

Fig. 5.7 Two-dimensional representations of data array for multipoles with metadata structure

redundant multipoles on surrounding 10^3 cells than to extract multipoles on 875 (for M2L) and 8 (for M2M) cells explicitly.

Figure 5.7 shows the data structure of multipoles at level 0 (**meta_wm0**) in a 2D representation where multipoles on four subcells at (5,5), (6,5), (5,6), and (6,6) addresses are distributed to myrank. The FMM is based on a hierarchical partitioning of a calculation unit cell where metadata of relative distance among subcells or supercells are equipped originally. Unlike the case of atomic number, the number of elements for each cell is constant, which corresponds to the number of expansion terms $(\text{nmax} + 1)^2$. Thus, it is not necessary to prepare additional arrays to store the metadata of data numbers for each cell.

In **meta_wm0** array, data are serially packed along the X-axis in each line. Data transferred from other ranks along the $\pm X$-axis by MPI communications are placed at both sides of data in myrank. A data block in each line, e.g., meta_wm0(1:$(\text{nmax} + 1)^2$,1:10,5) is sent to the adjacent rank by a MPI communication along the $+Y$-axis, receiving a data block meta_wm0(1:$(\text{nmax} + 1)^2$,1:10,4) from the adjacent rank in $-Y$ direction. By repeating send/receive communications, the empty region in the halo area is filled with multipole data necessary for the M2M and M2L operations.

5.3.3 Techniques to Realize Highly Effective Arithmetic Operations

5.3.3.1 Data Blocking

Data blocking is a general method to utilize the data on the cache(s). The data to be calculated are partitioned into a group of patches, which are small enough to be placed on the cache, especially level 1 cache (a few 10 kBs). It is desirable for calculation efficiency that the same data are loaded from the main memory to the cache only once. In other words, the data loaded on the cache are consumed thoroughly without flowing back.

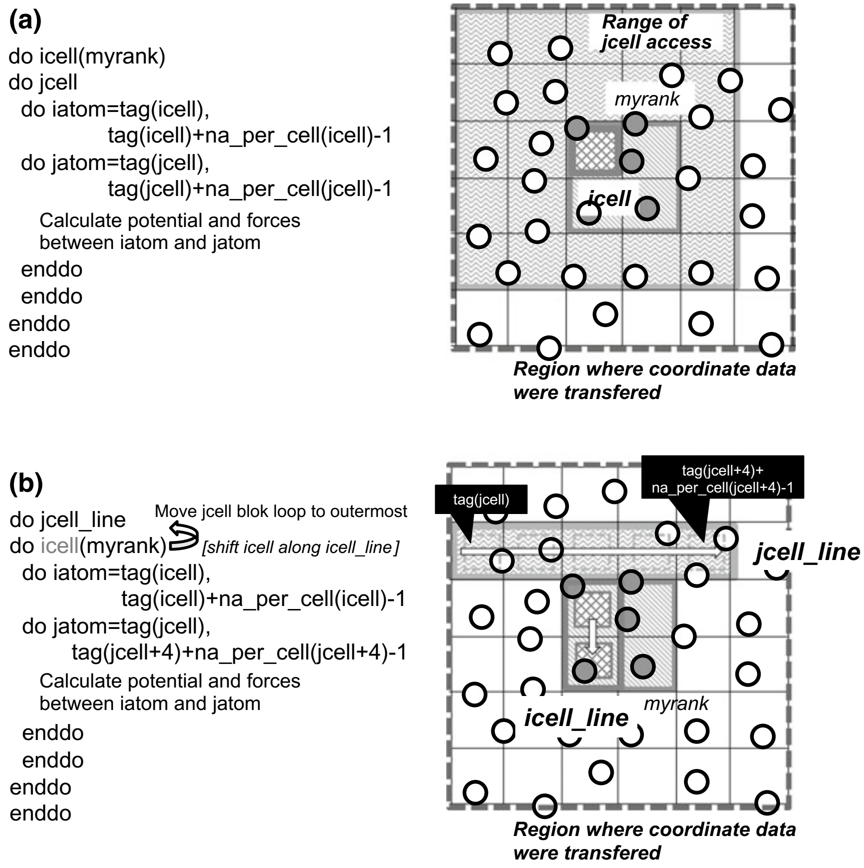
In MD calculations, a data-blocking target is the coordinate data in the pairwise additive interaction calculations. If the FMM is adopted for the calculation method of the long-range electrostatic interaction, the multipole data on each subcell or supercell and partial elements of the transformation matrix in the M2L operation are to be blocked.

An issue in the pairwise additive interaction calculation by the LLC method is summarized in Fig. 5.8a. In the usual implementations of the LLC method, DO loop for *i*-cells owned by myrank is located at the outermost position, followed by *j*-cells loop. Inside these cell loops, loops for *i* atoms in the *i*-cell and *j* atoms in one specified *j*-cell are included. This loop structure is easy to understand, although there is a large amount of wasted time loading the coordinate data of *j* atoms; i.e., the range of access is determined separately by the position of each *i*-cell and most of them overlap each other. Thus, the same data of the *j* atom coordinates are repeatedly loaded from the main memory to the caches.

On the other hand, Fig. 5.8b shows a DO loop structure considering the data blocking of *j* atom coordinates. The outermost DO loop orders a shift of blocked data, i.e., **jcell_line** in the figure. A length of the **jcell_line** is five subcells in the direction where data are stored serially. The second inner loop orders a shift of *i*-cells in a direction vertical to the **jcell_line** within the decomposed domain (**icell_line**). The third DO loop selects *i* atoms in each subcell, followed by the fourth DO loop, which selects *j* atoms in each **jcell_line**. With this DO loop structure, coordinate data in each **jcell_line** are loaded only once, and all relevant pairwise interactions are calculated properly.

In typical biological systems, if a side length of subcell is about 6–7 Å, the number of atoms in each subcell is nearly 40. Thus, the data size of atom coordinates in each **jcell_line** is about 5 kB, which is small enough to be placed in the level 1 cache (typically, 32 kB in modern CPUs). For calculations of the LJ and electrostatic interactions, the force field parameters (σ, ε, and q) are also required. In general, the same parameters are reused for different atoms; therefore, the required data size for these is relatively small. The data size for *i* atoms is one fifth of *j* atoms, which costs only a few kBs of data.

In the case of M2L operation, the same loop structure is adopted for a blocked arithmetic operation, as shown in Fig. 5.9. In the conventional loop structure, desti-



(a)

```
do icell(myrank)
do jcell
  do iatom=tag(icell),
            tag(icell)+na_per_cell(icell)-1
  do jatom=tag(jcell),
            tag(jcell)+na_per_cell(jcell)-1
    Calculate potential and forces
    between iatom and jatom
  enddo
  enddo
enddo
enddo
```

(b)

```
do jcell_line   Move jcell blok loop to outermost
do icell(myrank)  [shift icell along icell_line]
  do iatom=tag(icell),
            tag(icell)+na_per_cell(icell)-1
  do jatom=tag(jcell),
        tag(jcell+4)+na_per_cell(jcell+4)-1
    Calculate potential and forces
    between iatom and jatom
  enddo
  enddo
enddo
enddo
```

tag(jcell)
tag(jcell+4)+ na_per_cell(jcell+4)-1
jcell_line
icell_line
myrank
Region where coordinate data were transfered

Fig. 5.8 An example of the data blocking in the pairwise additive calculation of interatomic interactions

nation cells are shifted by the outermost DO loop, followed by a shift of source cells. For each selected cell pair, matrix (M2L transformation matrix) times vector (multipole elements) arithmetic operations are executed. However, in the data-blocking arithmetic operations, the multipole data on the surrounding 875 subcells or supercells are partitioned into a set of small patches. Similarly to the blocked pairwise additive calculation, the outermost DO loop shifts a patch of source cells. The second inner DO loop shifts a destination cell within the domain distributed to myrank. After selecting source and destination cell pairs, a matrix-times-vector operation is executed. The optimal size of each block depends on the truncated degree of multipole expansions $n_{\max}$. The expansion coefficient and translation matrix elements are complex numbers (16 bytes (B) in double precision), and the required data size for blocks is an integer multiple of 16 B × $(n_{\max} + 1)^2$, which should be less than the cache size on the hardware used.

When the data-blocking technique is adopted, the data in each block are also required to be serially packed on the main memory to reduce the frequency of data transfer to the cache and for the following vectorized arithmetic operations by SIMD instructions. An easy way is to prepare a temporary array for blocked arithmetic operations by copying the original data to the temporary array with partitions between each block. However, as described below, the time consumed in a data copy operation from the original array to the temporary array could become a rate-determining process when the elapsed time is greatly reduced by hierarchical parallelization of a code. With the metadata structure of coordinate data as described in Sect. 5.3.2, the temporary array and copy operation are omitted completely because the coordinate data are already partitioned based on small units (subcells) and data are serially packed along one axis by a series of MPI communications. Consequently, the atom coordinates in an arbitrary number of subcells in one direction can be serially accessed by a combination of **tag** and **na_per_cell**.

In conclusion, the metadata structure is excellent from the perspectives of not only low latency MPI communications, but also the data-blocking technique for the following arithmetic operations.

5.3.3.2 Thread-Level Parallelization

Thread-level parallelization is realized by the OpenMP language extension, which is activated by adding the OpenMP directives. Since the latest hardware development trend is toward many cores and wide SIMD architectures, it becomes critically important to apply efficient thread-level parallelization at every hot spot.

Briefly, the range of thread-level parallelization in a Fortran code is defined by a pair of !$omp parallel and !$omp end parallel directives (#pragma omp parallel and #pragma omp end parallel in a C code). In a defined parallelized region, any DO loops sandwiched by !$omp DO and !$omp end DO directives are the target of thread-level parallelization (#pragma omp for in a C code, without end statement). In addition, a proper compile option to a compiler (e.g., -fopenmp for gfortran, or -qopenmp for ifort, -mp for pgfortran) is required to interpret the inserted directives. Modern compilers are equipped with the auto thread-level parallelization function, which is activated by the defined option (-parallel for ifort, -Mconcur for pgfortran). In practice, however, it is only applicable to simple codes, such as setting of initial array values because the compiler assigns priority to keep the auto parallelized code accurate. Therefore, an explicit thread-level parallelization of calculation hot spots by inserting the OpenMP directives is required.

A general point to be noted for efficient thread-level parallelization is the uniform load balancing between threads. The easiest but effective way is to elongate the length of the target loop for thread-level parallelization compared with a given thread number. By default, a block of a whole loop length divided by N_t is assigned to each thread. Thus, the longer the loop, the smaller the load imbalance between threads becomes. Frequently, a fusion technique of two or more loops is used to elongate the loop length. The collapse clause in the OpenMP is prepared for this purpose; i.e.,

```
!$omp parallel
!$omp do
do iblk =1,nblock
do icy =icyblkst (iblk ),icyblkend (iblk )
do icx =icxblkst (iblk ),icxblkend (iblk )
do icell (myrank )
if(icx,icy is within 2 nearest neighbors) cycle
do m1=1,(nmax+1)^2
do m2=1,(nmax+1)^2
   wl(m1) =wl(m1)+ m2l (m1,m2)* wm(m2)
enddo
enddo
enddo
enddo
enddo
enddo
!$omp end do
!$omp end parallel
```

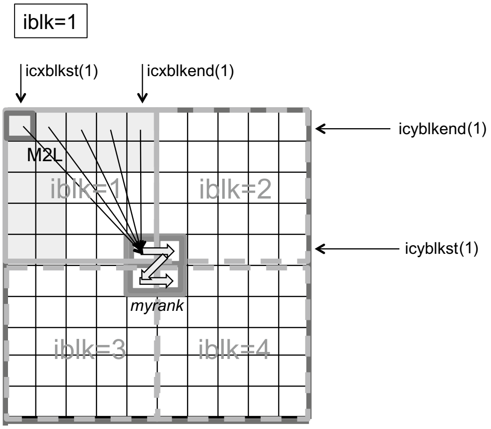


Fig. 5.9 Blocked arithmetic operations parallelized with the OpenMP directives in the M2L operation

multiple loops are merged into one loop in the compiling process and thread-level parallelization is performed on the merged loop.

The next method is to design the size of loop blocks assigned to each thread explicitly, if possible, considering blocked arithmetic operations. In addition, it is desirable that the number of blocks is integer times the number of threads from the perspective of load balancing of parallelized calculations between threads. This method achieves excellent load balancing, especially when the amount of calculations between each pair of source block and destination patch is uniform.

Figure 5.9 shows an example of the blocked arithmetic operations parallelized with the OpenMP directives in the M2L operation. The outermost loop with DO variable **iblk** shifts the block of source cells with a total nblock number (nblock = 4 in this case). If given thread number N_t is 4 (determined by an environmental variable of operation system OMP_NUM_THREADS when a code is executed), one block composed of 25 cells is assigned to each thread. The block shape is controlled by the index arrays icxblkst, icxblkend, icyblkst, and icyblkend with an argument, **iblk**. In a 3D case, an additional two index arrays are introduced, iczblkst and iczblkend.

When the amount of calculations is not uniform for each block, which is the general case for pairwise additive interaction calculations, a more sophisticated thread-level parallelization algorithm is required to achieve high parallelization efficiency with good load balancing.

5.3.3.3 Vectorized Arithmetic Operations

For operations within the innermost loop, vectorized arithmetic operations could be applied by the SIMD instructions on modern CPU architectures. Its prerequisite is that the same kinds of operations are performed on a series of input data. In MD calculations, e.g., pairwise additive calculations of the LJ and electrostatic interactions fully satisfy this prerequisite. That is, the data of atom coordinates and force field parameters for *j* atoms in the jcell_line and *i* atoms in *i*-cell are loaded, and then the potential energy and force are calculated by the equations in the same functional form. The M2L operation is also a good candidate for vectorized arithmetic operations because multiplications of the coefficient matrix and multipole element row are repeated for each pair of source and destination subcells or supercells.

There are several ways to implement vectorized arithmetic operations. The easiest way is to use the compilers' auto-SIMD optimization or autovectorized functions, which are activated by adding specific compiler options (e.g., -msse for gfortran, -vec for ifort, or -fastsse for pgfortran). Compilers are also equipped with an option to output vectorized reports of a code, sometimes with a function annotating the elements to inhibit the generation of vectorized instructions (e.g., -qopt-report for ifort). Performance of vectorized arithmetic operations in a calculation hot spot could be improved step by step, modifying the code being more suited to vectorized arithmetic operations based on a vectorized report. Directives for SIMD optimization (!$OMP SIMD [42]) are now prepared from the OpenMP 4.0, which are activated in another compiler option (e.g., -qopenmp-simd for ifort). With these SIMD directives, more detailed information to enhance autovectorization can be added to the compiler.

The second, but highly professional way is to use SIMD's intrinsic functions [43]. An optimized set of vectorized codes for basic mathematical operations are prepared for each vector technology (e.g., SSE, AVX, AVX-512) and precision. However, intrinsic coding does not always result in highly effective vectorization of a hot spot. It is only valid when the code is rate-limited by arithmetic operations themselves: If not, vectorized arithmetic units waste most of the time waiting for input data. Therefore, in advance of optimization for vectorized arithmetic operations, hierarchical coding based on a sophisticated data structure must be realized to achieve excellent total parallelization efficiency of the code. For example, our proposed metadata structures for coordinates and multipoles satisfy the condition for well-vectorized arithmetic operations. The blocked coordinate data for *j* atoms in jcell_line or multipoles on source cells are placed serially in the level 1 cache without delays in loading time for data from the main memory.

We outline the general considerations for highly efficient vectorized operations as follows. First, the serial dependency between elements in data arrays should be removed by modifying a code or the algorithm itself. In general, backward or forward reference to data, such as

```
DO
  a[i]=a[i-1]+b[i]
ENDDO
```

or

```
DO
  a[i]=a[i+1]+b[i]
```

```
ENDDO
```

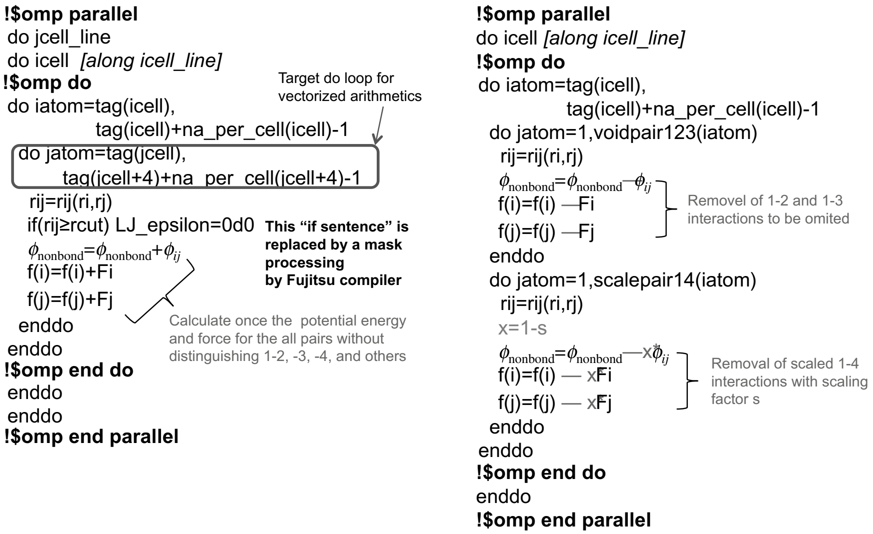
where loop index i is difficult to be vectorized because the present arithmetic operation for a[i] depends on the past a[i-1] or future a[i+1] data with the possibility of being updated. It is also difficult to vectorize a code effectively when a loop contains an indirect access to a data array element by introducing an index list array, such as

```
DO
  a[list(i)]=a[list(i)]+b[i]
ENDDO
```

with loop index i. There are some established tricks to overcome these difficulties, whereas the best way is to adopt an alternative algorithm without backward or forward reference to data.

Second, the type of arithmetic operation to input data in the target loop is as uniform as possible; e.g., in the calculation of the LJ interactions in MD calculations, a cutoff technique (Sect. 5.1.2) is usually adopted. Its execution depends on the distance between *i* and *j* atoms. Usually, an IF sentence may be introduced into the target loop of vectorized arithmetic operations, which makes arithmetic operations in the loop nonuniform. Another example in MD calculations is an exclusive calculation of interatomic interactions in the same molecule. Since interatomic interactions between atoms separated by one chemical bond (the so-called 1–2 interaction) and by two chemical bonds (the 1–3 interaction) are evaluated by bonded potential terms and angled potential terms, they must be eliminated from interatomic LJ and electrostatic interaction calculations to avoid a duplicated estimation of interactions. In addition, a scaled calculation (or calculation with special parameters) of an interatomic interaction between atoms separated by thee chemical bonds (the 1–4 interaction) in the same molecule should be considered in longer molecules, since the interaction is partially evaluated by dihedral potential terms. In the former, the code can be made uniform by introducing a cutoff variable with a value of 0 (if $r_{ij} > r_{\mathrm{cut}}$) or 1 (else) multiplied by the calculated potential and forces. In the latter, it may be possible to use a redundant calculation and removal technique that calculates all pair interactions once, and then eliminates the pair interactions from the total. In most cases, the removal process code is complicated and does not suit vectorization. However, since the calculation amount is much higher in the uniform calculation than the latter removal calculation, this method greatly improves the total performance. An issue is that a large round off could produce errors when subtracting repulsive 12-power terms in the LJ interaction. The error is acceptable with a double-precision floating-point calculation, while it is not acceptable in the case of a single-precision calculation. In such a case, another technique, such as mask processing to avoid the 1–2 and 1–3 calculations should be implemented to keep the MD calculations accurate.

Third, the loop length is long enough to conceal the pre- and post-processing time for vectorized arithmetic operations and enhance the software pipeline. It is also desirable from the perspective of load balancing between vector lines. An easy way to elongate a loop length is a fusion of the loop with upper loop(s) as in the case for thread-level parallelization by the OpenMP directives described above. However,



```
!$omp parallel
 do jcell_line
 do icell  [along icell_line]
!$omp do
 do iatom=tag(icell),
           tag(icell)+na_per_cell(icell)-1
  do jatom=tag(jcell),
       tag(jcell+4)+na_per_cell(jcell+4)-1
   rij=rij(ri,rj)
   if(rij≥rcut) LJ_epsilon=0d0
   $\phi_{nonbond}=\phi_{nonbond}+\phi_{ij}$
   f(i)=f(i)+Fi
   f(j)=f(j)+Fj
  enddo
 enddo
!$omp end do
 enddo
 enddo
!$omp end parallel
```

Target do loop for vectorized arithmetics

This “if sentence” is replaced by a mask processing by Fujitsu compiler

Calculate once the potential energy and force for the all pairs without distinguishing 1-2, -3, -4, and others

```
!$omp parallel
do icell [along icell_line]
!$omp do
do iatom=tag(icell),
           tag(icell)+na_per_cell(icell)-1
  do jatom=1,voidpair123(iatom)
    rij=rij(ri,rj)
    $\phi_{nonbond}=\phi_{nonbond}-\phi_{ij}$
    f(i)=f(i) —Fi
    f(j)=f(j) —Fj
  enddo
  do jatom=1,scalepair14(iatom)
    rij=rij(ri,rj)
    x=1-s
    $\phi_{nonbond}=\phi_{nonbond}-x\phi_{ij}$
    f(i)=f(i) — xFi
    f(j)=f(j) — xFj
  enddo
enddo
!$omp end do
enddo
!$omp end parallel
```

Removel of 1-2 and 1-3 interactions to be omited

Removal of scaled 1-4 interactions with scaling factor s

Fig. 5.10 Vectorized arithmetic operations of the pairwise additive LJ and electrostatic interactions for a molecular system, which includes the 1–2 and 1–3 interactions to be omitted and the 1–4 interactions to be scaled in the same molecule

it usually involves an introduction of an index list that relates the loop variable of a fused new loop to the loop variables in the original loops. As stated above, an indirect access by an index list makes it difficult to create an optimized vectorized code using a compiler because the data dependency between loops becomes opaque for a compiler. In any case, the best way is to elongate the target loop in the original code from the perspective of calculation algorithms.

Figure 5.10 shows an example of the SIMD-optimized code for pairwise additive calculations of the LJ and electrostatic interactions, including thread-level parallelization by the OpenMP directives adopted by the MODYLAS [7] software. The left block of the program is executed first, followed by the right block. The third DO loop with DO variable **iatom** in the left block is the target of thread-level parallelization. The innermost DO loop with DO variable **jatom** is the target of vectorized arithmetic operations; its loop length is determined as the [average number of atoms in each subcell (about 40)] × [number of subcells in jcell_block (5)] ≈ 200 for typical biological systems. The cutoff processing of the LJ interaction is performed by the IF sentence, which is replaced by mask processing in the compiling process by Fujitsu's Fortran compiler. The potential energy and forces are removed from their total by the third DO loop in the right block, which detects the atoms of the 1–2 and 1–3 interaction sources for each iatom. Similarly, the fourth DO loop in the right block detects the 1–4 interaction source for each iatom in which the calculated interaction multiplied by $x = 1 - s$ (s: original scaling factor) is removed from the total.

By a performance measurement on the K computer, which has a 128-bit SIMD with for double-precision floating-point calculations and eight cores (eight threads execution) in each CPU, the left program block gives over a 95% SIMD instruction fraction to floating-point arithmetic operations with a very low cache miss rate at less than 1% and nearly 0% for the level 1 and 2 caches. In addition, the distributed memory parallelization by the MPI keeps scalability up to 65,536 (2^{16}) processes with an input of a 10 million atom system when the long-range electrostatic interactions are calculated by the FMM [7].

5.3.4 Future Prospects

In the exascale era, general-purpose supercomputers with many cores and vectorized units with wider SIMD widths will be installed. The quality of not only thread-level parallelization, but also vectorized arithmetic operations becomes more critical in performing highly effective parallelized calculations on such supercomputers. Obviously, points to note for achieving highly effective parallelization are a good load balance and the uniformity of arithmetic operations between parallelized units. These are realized by temporarily applying some established methods, although the best way is to modify the calculation algorithm at the hot spot itself as we recently proposed for the P2P operation [44]. New algorithms for parallelization should be continuously invented to adopt to future supercomputer architectures.

5.4 GENESIS

5.4.1 Main Features of GENESIS for Optimization and Parallelization

To extend the simulation system size and time is a great challenge in MD due to small integration time step and required long simulation time. GENESIS (Generalized-Ensemble Simulation System) is implemented to address these challenges by optimizing the program and developing efficient algorithms for ensemble generations [8]. In this section, we focus on the developments in GENESIS from the point of view of optimization. Because the electrostatic interaction in GENESIS is based on particle mesh Ewald (PME), we will discuss optimization on two points: efficient parallelization of three-dimensional Fast Fourier Transform (FFT) and evaluations of finite-range interactions in the real-space.

5.4.1.1 Inverse Lookup Table [45]

In MD, the major bottleneck is the calculation of non-bonded interactions including van der Waals or electrostatics. Especially in the PME method, much of the operation cost is spent on the square root function and the error function evaluations. MD software, therefore, usually employs a calculation method that avoids calling these mathematical functions. Instead, a numerical lookup table obtained by performing numerical interpolation is generated in advance and used for non-bonded interactions. The feature of GENESIS's lookup table is that the numerical interpolation is performed evenly in proportion to the reciprocal of the distance squared. Using this lookup table, it is possible to accurately calculate the interaction energy and the force at short distance since many interpolation points are used. As for the large distance interaction, it is possible to increase the performance by using a small number of interpolation points. Given the interaction distance r and the cutoff distance r_υ, respectively, it is possible to define the interpolation point coordinates L using the interpolation point density D,

$$L = \mathrm{INT}\left(D \times \frac{r_\nu^2}{r^2}\right) \tag{5.11}$$

Then, the energy function $E(r)$ using the lookup table can be written as

$$E(r) = E_{\mathrm{tab}}(L) + t(E_{\mathrm{tab}}(L+1) - E_{\mathrm{tab}}(L)) \tag{5.12}$$

$$t = D \times \frac{r_\nu^2}{r^2} - L \tag{5.13}$$

By using the lookup table scheme with Eqs. (5.11)–(5.13), which we named inverse lookup table, CPU cache memory can be used efficiently, so high-speed nonbonded interaction evaluation can be performed. Even using small number of interpolation points, it provides accurate energy and force calculation.

5.4.1.2 Spatial Decomposition Scheme in GENESIS

In GENESIS, there are two MD simulators: ATDYN (ATomic decomposition DYNamics) and SPDYN (SPatial decomposition DYNamics). ATDYN is parallelized based on the atomic decomposition, and each MPI processor has all information of the system, such as coordinates, velocities, charges, and so on. Parallelization of SPDYN is based on the spatial decomposition scheme. SPDYN is particularly designed for large-scale MD simulations suitable for recent multicore CPUs. Therefore, in this section, we introduce the parallelization method of GENESIS, limited to SPDYN. In SPDYN, the simulation space is divided into the same subdomain as the number of MPI processors. Subdomain is further divided into smaller unit domains called cell. The length of cell in each dimension is limited to a size greater than half of the cutoff distance. Particle data like coordinates or velocities are grouped according to cell indices. Pairwise interactions are grouped by cell pairs which are distributed over OpenMP threads by shared memory parallelization scheme using OpenMP.

(1) Midpoint cell method [8]

The midpoint cell method is an extension of the existing midpoint method for hybrid parallelization of MPI and OpenMP. In the midpoint method, the non-bonded interaction between a particle pair is evaluated in the subdomain containing the midpoint of the particle pair. Therefore, a particle pair interaction is often performed in a subdomain where none of the particles exist. In the case of using the cut-off distance or pairlist cutoff distance, r_v, it is sufficient to import coordinate data within a distance of $r_v/2$ from each subdomain using the midpoint scheme. In the midpoint cell method, particle data are grouped cell-wise and the interaction subdomain is not decided from the midpoint of each particle pair but from the midpoint cell of each cell pair in which each particle resides. The midpoint cell is sometimes not uniquely defined. For example, in Fig. 5.11, the midpoint cell of the cell pair a and b is uniquely determined. On the other hand, there are two possibilities for the midpoint cell of the cell pair c and d. In such a case, the midpoint cell is determined considering the load balance in the actual operation. In the case of using the midpoint cell method, it is only necessary to communicate from/to the adjacent cells of each subdomain. In both midpoint and midpoint cell methods, communication cost is reduced by increasing the number of processors, enabling high parallel efficiency. In the existing midpoint method, it is necessary to determine the subdomains including the midpoints for all particle pairs every step or whenever pairlist is rewritten. On the other hand, in the midpoint cell method, midpoint cells are decided before running MD simulation.

(2) Volumetric decomposition FFT [28]

By using the PME method, the amount of computation required for the non-bonded interaction decreases from $O(N^2)$ to $O(N \log N)$ when the total number of particles is N. However, since the FFT calculation in PME requires global all-to-all communications, FFT becomes a main bottleneck in MD when using very large number of processors. In NAMD and GROMACS, slab (one-dimensional) or pencil (two-dimensional) decomposition schemes are used, whereas GENESIS uses a parallelization method based on volumetric (three-dimensional) decomposition. Volumetric decomposition FFT requires more frequent all-to-all communication than the case of using slab or pencil decompositions, but the number of processors involved in each communication is minimized. For this reason, FFT calculation using volumetric decomposition is generally suitable for MD simulations of big systems using many processors. Furthermore, by applying the same node topology as volumetric decomposition, volumetric decomposition can be useful for supercomputers equipped by torus network type like K. When the midpoint cell method and volumetric decomposition FFT are combined to each other, all-to-all communication of charge information can be skipped. For example, in Fig. 5.12a, the information of the charge in the reciprocal-space held by the processor 10 can be obtained without any communication from the information held by the same processor in the real-space. If different decomposition scheme is done between the real- and reciprocal-spaces, shown in

Fig. 5.11 Spatial decomposition scheme in GENESIS SPDYN

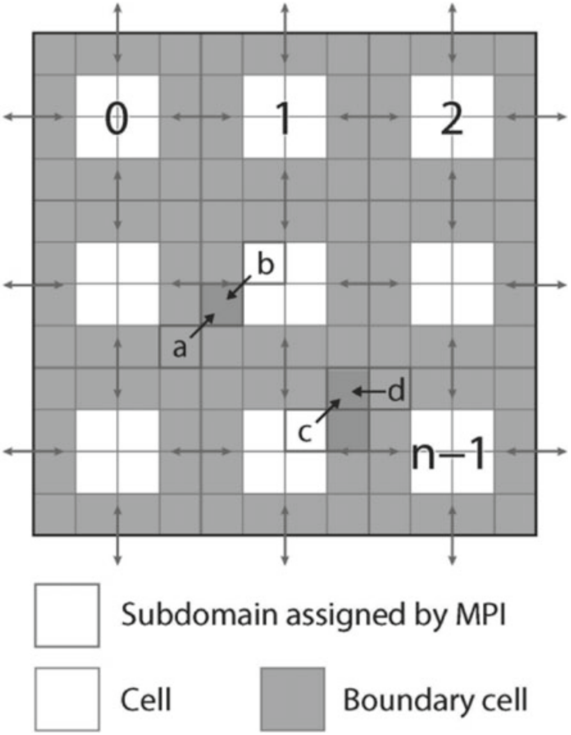


Fig. 5.12b, the charge information in the reciprocal-space in the processor 10 can be obtained by communication among processors 3, 7, 11, and 15.

5.4.2 *Benchmark Performance of GENESIS*

Here we introduce the benchmark results of GENESIS using PC clusters and the K computer. As for the PC clusters, we used 32 nodes, each of which consists of two Intel Xeon E5-2670 CPUs. Since the number of cores included in this CPU is 8, the number of cores in a node is 16, so the PC cluster consists of 512 CPU cores. We used three molecular systems for the benchmark, which are listed as followings:

(1) Main porin from Mycobacterium smegmatis (MSPA, 216, 726 atoms in 126.933 × 126.933 × 131.063 Å^3 box)
(2) Satellite tobacco mosaic virus (STMV, 1,066,628 atoms in 216.83 × 216.83 × 216.83 Å^3 box)
(3) 27 STMV (28, 798, 956 atoms in 650.49 × 650.49 × 650.49 Å^3 box)

The third molecular system is obtained by magnifying three times in each dimension from (2). The number of FFT grids are (144, 144, 144), (256, 256, 256), and (512, 512,

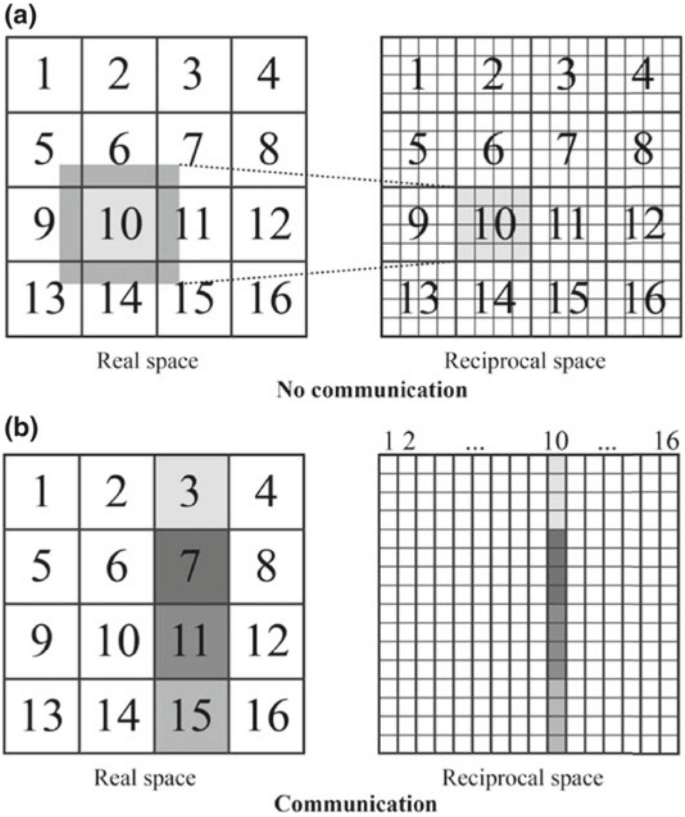


Fig. 5.12 **a** There is no communication between real- and reciprocal-space if we assign identical spatial decompositions, but **b** communication is necessary if different spatial decomposition is assigned

512), respectively. The cutoff distance and the pairlist cutoff distance were (12 Å, 13.5 Å) for (1) and (10 Å, 11.5 Å) for (2) and (3). Water molecules were treated as rigid bodies using the SETTLE method, and intramolecular bonds containing hydrogen atoms were subjected to be constrained using the SHAKE/RATTLE algorithms. The time step in MD simulation is set to 2 fs. Figure 5.13 shows the benchmark results of MSPA and STMV on the PC clusters. Parallelization efficiency is not lowered up to 512 cores despite we do not use multiple time step (MTS) integration. The maximum speeds of MSPA and STMV on the PC clusters are 17.79 ns/day and 5.87 ns/day, respectively.

Next, the performance using the K computer for STMV and 27 STMV is shown in Fig. 5.14. For STMV, the speed efficiency does not decrease up to 4096 nodes, and the maximum speed is 37.61 ns/day. When using RESPA MTS integration, the maximum speed has reached 47.77 ns/day. For 27 STMV, the speed efficiency does not decrease up to 16,384 nodes, and the maximum speeds using the velocity Verlet and RESPA MTS integrations are 14.60 ns/day and 17.51 ns/day, respectively.

Our benchmark results show excellent parallelization efficiency of GENESIS. Despite executing FFT calculation including all-to-all communication every step, high parallelization efficiency is maintained both in PC cluster and K. The performance of GENESIS shows even better values using RESPA MTS integration scheme.

Exercises

1. **Process number calculation**: MD calculations are often performed under the three- dimensional periodic boundary condition. In a circularly shifted communication using the MPI_SENDRECV function described in Sect. 5.2.3.2, it is necessary to determine process numbers **dest** and **source** with taking into account of the periodic boundary condition. Given that the total number of MPI processes is NPROCS = 2^n ($n \geq 1$), and the number of processes assigned along the x, y,

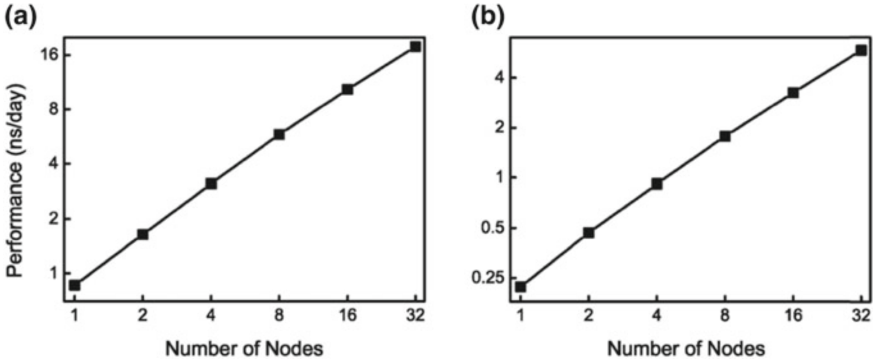


Fig. 5.13 Performance of **a** MSPA and **b** STMV systems on K computer

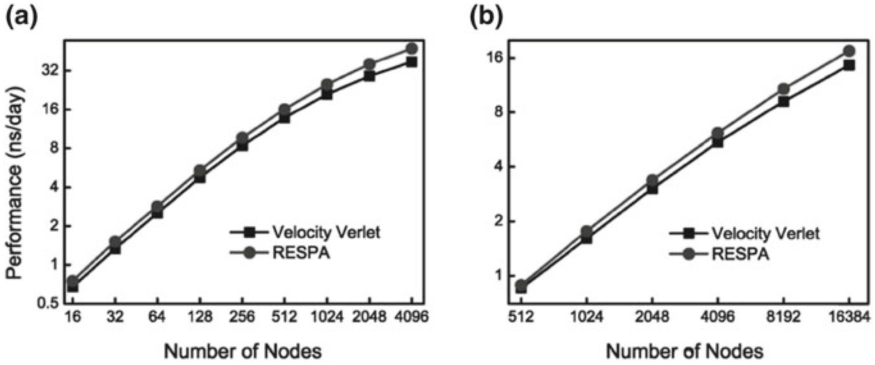


Fig. 5.14 Performance of **a** STMV and **b** 27 STMV systems on K computer

and z axis are 2^{n_x}, 2^{n_y} and 2^{n_z} (i.e., $n = n_x + n_y + n_z$). Then, derive an equation to calculate **dest** and **source** for any myrank.

2. **Reduction of array**: Code a program that takes a reduction of array parallelized by threads, without using the reduction clause prepared in OpenMP language extension (i.e., !omp reduction (+:a)).
3. **Performance measurement of vectorized operation**: (1) Specify an option to enable automatic SIMD vectorization, by applying "man" linux command to a compiler. Its name depends on a kind of fortran compiler. (2) Evaluate rate of acceleration by vectorized SIMD operations, comparing elapsed time of a code by vectorized operation with that by normal (non-vectorized) operation, by using "time" linux command. With adding "-O0" option for compilers, completely non-vectorized executable is created. Note that the latest compilers do optimization of a code to some extent at default, and that "-O0" option disables such hidden optimizations.

Appendix: FMM

The principle of the fast multipole method (FMM) is briefly explained in this Appendix. See Ref. [15] for exact mathematical treatment of the transformation of multipole moment and local expansion coefficients, such as M2M, M2L, and L2L.

Fundamentals of Multipole Expansion

As shown in Fig. 5.15, the atom i of the charge q_i in a certain area forms the potential at a distant point Q. The reciprocal $1/r_i$ of the distance between two points is calculated using the distance Δr_i from the origin O to the atom i in the region, the distance r from the origin O to the point Q, and the angle γ_i formed by the points Δr_i and r with the point O. This can be expressed as

$$\frac{1}{r_i} = \frac{1}{\sqrt{r^2 - 2r\Delta r_i \cos\gamma_i + \Delta r_i^2}} = \frac{1}{r\sqrt{1 - 2\frac{\Delta r_i}{r}\cos\gamma_i + \left(\frac{\Delta r_i}{r}\right)^2}} \tag{5.14}$$

from the cosine theorem. When this equation is rewritten by Taylor expansion, it becomes

$$\begin{aligned}\frac{1}{r_i} &= \frac{1}{r} + \frac{\Delta r_i}{r^2}\cos\gamma_i + \frac{\Delta r_i^2}{2r^3}\left(3\cos^2\gamma_i - 1\right) + \frac{\Delta r_i^3}{2r^4}\left(5\cos^3\gamma_i - 3\cos\gamma_i\right) + \cdots \\ &= \frac{1}{r}\sum_n P_n(\cos\gamma_i)\left(\frac{\Delta r_i}{r}\right)^n,\end{aligned} \tag{5.15}$$

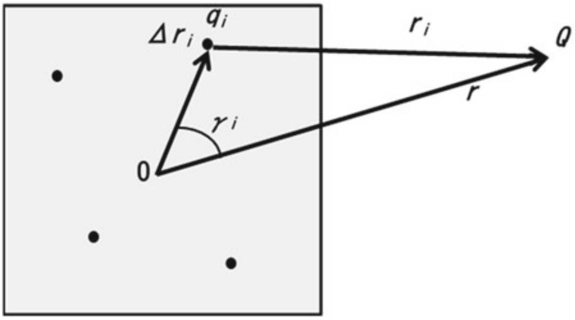


Fig. 5.15 Multipole expansion of electrostatic interaction

where $P_n(\cos\gamma_i)$ is a Legendre polynomial. This Eq. (5.15) is called multipole expansion. $P_n(\cos\gamma_i)$ is written as

$$P_n(\cos\gamma_i) = \sum_{m=-m}^{n} Y_n^{-m}(\theta_i, \phi_i) Y_n^m(\theta, \phi) \tag{5.16}$$

using the spherical harmonics, $Y_n^m(\theta, \phi)$ for spherical coordinates $(\Delta r_i, \theta_i, \phi_i)$ and (r, θ, ϕ) of the two vectors Δr_i and r. In Eq. (5.16), the coordinate variables of atom i and point Q are completely separated into two factors. Therefore, the sum with respect to the charges in a region can be represented independently of the position of the point Q. The sum

$$M_n^m = \sum_i q_i Y_n^{-m}(\theta_i, \phi_i) \Delta {r_i}^n \tag{5.17}$$

is the multipole moment formed by the charge in the region. The electrostatic field formed by all charges in the region can be expressed as

$$\sum_i \frac{q_i}{r_i} = \frac{1}{r^{n+1}} \sum_{n=0} \sum_{m=-n}^{n} M_n^m Y_n^m(\theta, \phi). \tag{5.18}$$

Division of Unit Cells

In FMM, unit cells are divided into small spaces (subcells). As described in the next section, the interaction calculation is performed using these subcells. There are several ways to divide unit cells into subcells; however, an octree structure is commonly used, in which each side of the unit cell is divided into two equal parts and the whole is divided into eight subcells (Fig. 5.16). This division is repeated until the subcell reaches the desired size of small subcells, i.e., level 0 for the unit cell and level 1 after

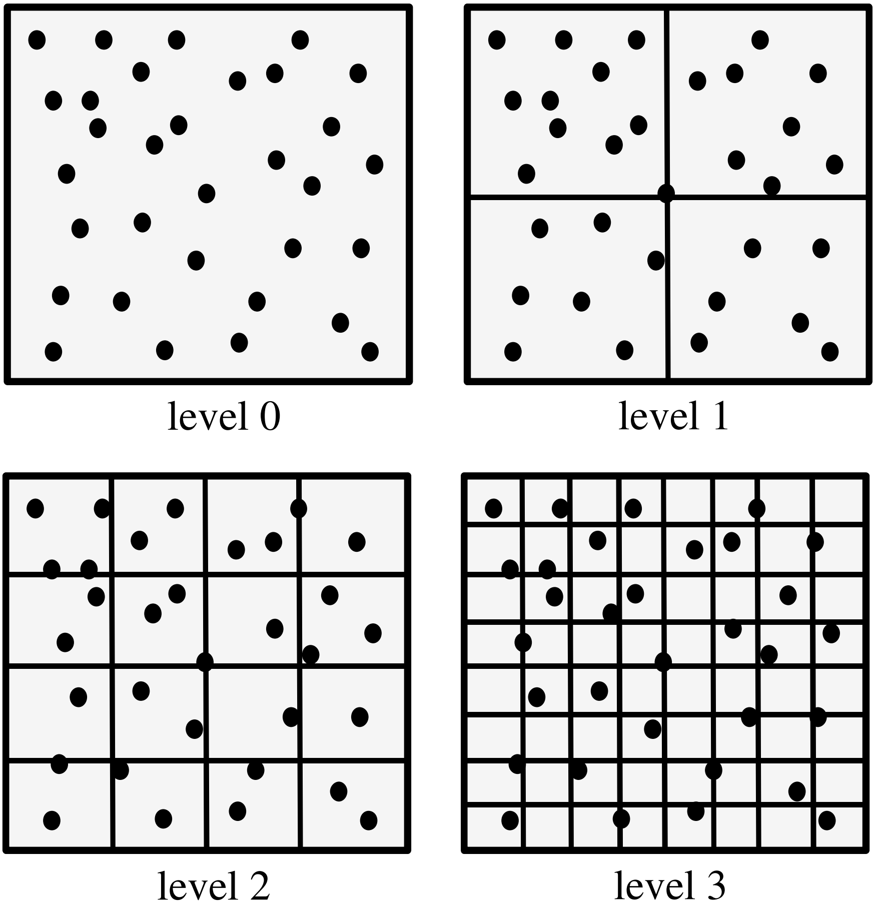


Fig. 5.16 Division of unit cell

dividing once. The number of subcells of level n is 8^n. In practical use, tens of atoms are assigned to the smallest subcell. This smallest subcell is a unit of regional division. We assign subcells to compute nodes and cores to perform parallel calculations.

Interaction Calculation by FMM

We assume that the unit cell is divided to level 4, as shown in Fig. 5.17. We now calculate the potential of an atom in region A. First, we evaluate the electrostatic interaction directly up to the second nearest neighbor (B) of region A of level 4. For

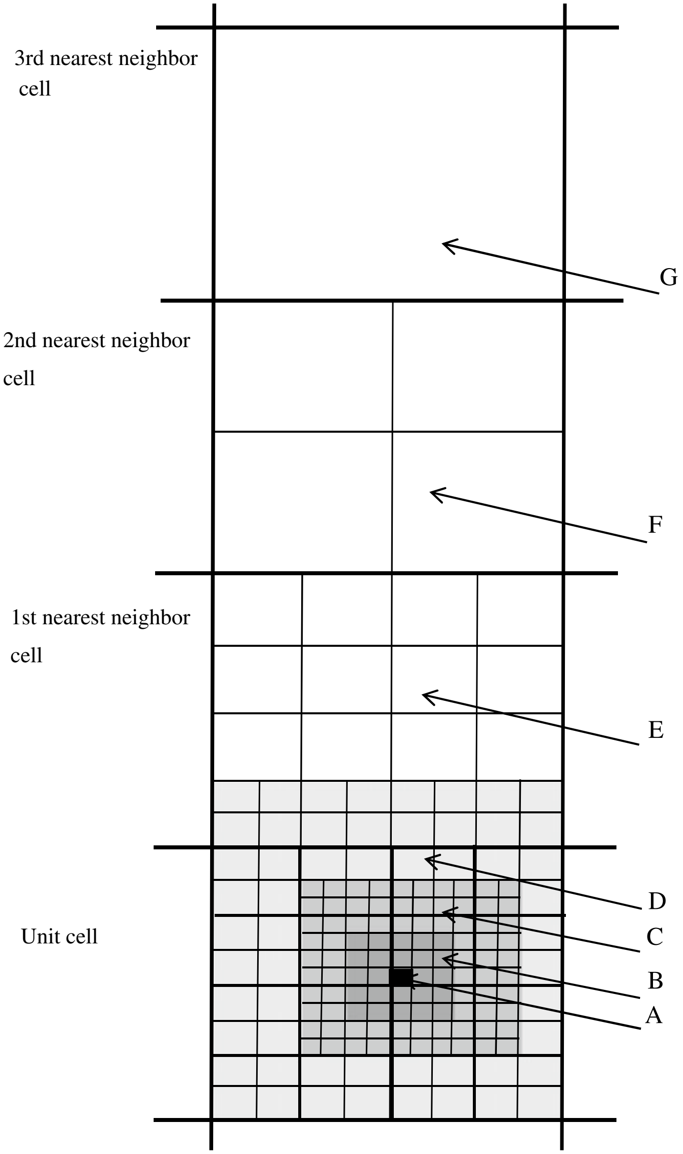

3rd nearest neighbor cell
2nd nearest neighbor cell
1st nearest neighbor cell
Unit cell
G
F
E
D
C
B
A

Fig. 5.17 Interactions between atoms and subcell

subcells beyond region B, the interaction calculation is performed using multipole moments.

Region C is within the second nearest neighbor of the subcell of level 3, but excluding region B. The interaction calculation with region C is performed using the multipole moment of level 4. Region D is within the second nearest neighbor of the subcell of level 2, but excluding region C. The interaction calculation with region B is performed using the multipole moment of level 3. Hereinafter, regions E and F are determined in the same manner. Subcells beyond region F (image cell of third or further nearest neighbor of the unit cell) are treated as those of level 0. Thus, the interactions can be calculated for each region without excess or deficiency. By using small subcells for interactions with neighboring regions and large subcells for interactions with distant regions, it is possible to efficiently perform interaction calculations without decreasing accuracy.

Multipole Expansion and Local Expansion

The procedure for determining the potential acting on the i atom in region A is shown (Fig. 5.18). First, the contributions from regions A and B is obtained by direct calculation. This procedure is called particle to particle (P2P) in FMM. For region C, the multipole moment is calculated in level 4 subcells according to Eq. (5.17). This operation is called particle to multipole (P2M). Next, the center of the multipole moment is moved to the center of subcell A. The expression obtained by this transformation is the form of Taylor expansion at the center of subcell A, which is called local expansion. This shift operation of the expansion center is called multipole to local (M2L). By using the local expansion coefficient and the relative position from the center subcell of atom i, its potential, force, and virial can be calculated, which is called local to particle (L2P). For the further subcell D, the multipole moment of the level 4 subcell is obtained by P2M similarly to C. The expansion center of the multipole moment is shifted to the center position of the level 3 subcell (i.e., multipole to multipole (M2M)). Similarly, with respect to the other seven subcells, the center of expansion is shifted by M2M. The obtained multipole moments are summed to obtain a multipole moment of level 3, which is transformed to the local expansion coefficient of the level 3 subcell, including subcell A, by M2L operation. The local expansion coefficient is shifted to the center of the subcell of level A (i.e., local to local (L2L)). The contributions from the previously obtained subcell C and the subcells D are summed and then the interaction is evaluated by L2P. The contribution from image cells more distant than region F in Fig. 5.17 can be included by applying Ewald's method to the multipole moment of image cells [46].

Accuracy of FMM

FMM is a strictly formulated analytical method that includes error evaluation. The error depends on the expansion order of the multipole moment; i.e., as the order increases, the calculation accuracy monotonically increases. Table 5.1 shows the benchmark results of potential and force in a system consisting of 10,125,000 atoms using MODYLAS software. The accuracy of FMM was evaluated by comparison

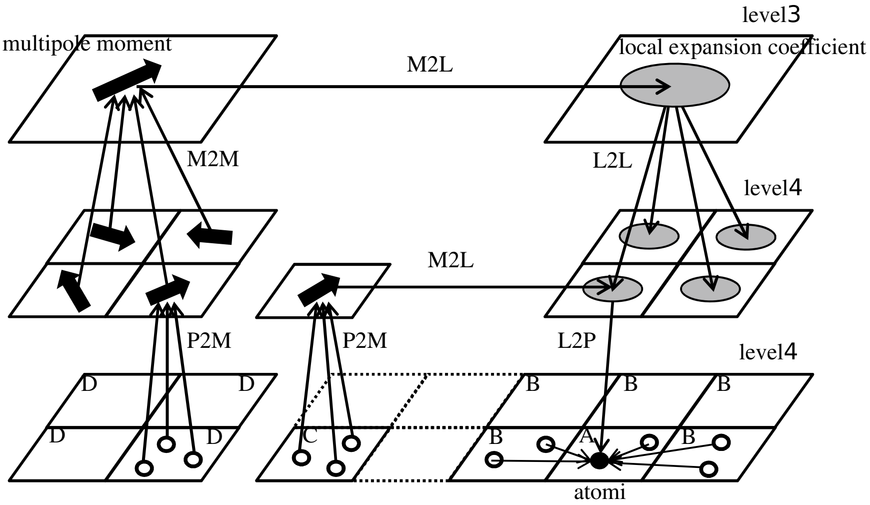


Fig. 5.18 Interaction between atoms and subcell

Table 5.1 Potential and force obtained by FMM

Expansion degree	$V/10^{-13}$ J	$F_x/10^{-10}$ N	$F_y/10^{-10}$ N	$F_z/10^{-10}$ N
Exact	−218,735,559	−4.357177	6.380146	−1.477348
4th	−218,735,742	−4.356659	6.380802	−1.475152
8th	−218,735,559	−4.357201	6.380145	−1.477340

with the result of the PME method with parameters giving a sufficiently high accuracy. The values obtained at the 4th to 8th expansion order coincide with the exact value in the range of 4–9 digits. There is also an accuracy of 4–7 digits for force; i.e., enough accuracy can be obtained for MD calculations even in expansions up to the 4th order.

References

1. W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.M. Merz Jr., D.M. Ferguson, D.C. Spellmeyer, T. Fox, J.W. Caldwell, P.A. Kollman, J. Am. Chem. Soc. **117**, 51795197 (1995)
2. A.D. MacKerell Jr., M. Feig, C.L. Brooks III, J. Comput. Chem. **25**, 14001415 (2004)
3. W.L. Jorgensen, D.S. Maxwell, J. Tirado-Rives, J. Am. Chem. Soc. **118**, 1122511236 (1996)
4. H.J.C. Berendsen, D. van der Spoel, R. van Drunen, Comput. Phys. Commun. **91**, 43–56 (1995)
5. J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kale, K. Schulten, J. Comput. Chem. **26**, 1781–1802 (2005)
6. S. Plimpton, J. Comput. Phys. **117**, 1–19 (1995)
7. Y. Andoh et al., J. Chem. Theory Comput. **9**, 3201 (2013)
8. J. Jung, T. Mori, C. Kobayashi, Y. Matsunaga, T. Yoda, M. Feig, Y. Sugita, WIREs Comput. Mol. Sci. **5**, 310 (2015)
9. D. Frenkel, B. Smit, *Understanding Molecular Simulation* (Academic Press, San Diego, 1996)

10. M.E. Tuckerman, *Statistical Mechanics: Theory and Molecular Simulation* (Oxford University Press, New York, 2010)
11. M.P. Allen, D.J. Tildesley, *Computer Simulation of Liquids* (Oxford University Press, 1989)
12. J. Wong-ekkabut, M. Karttunen, Biochim. Biophys. Acta **1858**, 2529 (2016)
13. T. Darden, D. York, L. Pedersen, J. Chem. Phys. **98**, 10089 (1993)
14. U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee, L.G. Pedersen, J. Chem. Phys. **103**, 8577 (1995)
15. L. Greengard, V. Rokhlin, J. Comput. Phys. **73**, 325 (1987)
16. D.J. Hardy, Z. Wu, J.C. Phillips, J.E. Stone, R.D. Skeel, K. Schulten, J. Chem. Theory Comput. **11**, 766 (2015)
17. J.J. Biesiadecki, R.D. Skeel, J. Comput. Phys. **109**, 318–328 (1993)
18. P. Minary, M.E. Tuckerman, G.J. Martyna, Phys. Rev. Lett. **93**, 150201 (2004)
19. J.P. Ryckaert, G. Ciccotti, H.J.C. Berendsen, J. Comput. Phys. **23**, 327341 (1977)
20. P. Gonnet, J. Comput. Phys. **220**, 740 (2007)
21. H.C. Andersen, J. Comput. Phys. **52**, 2434 (1983)
22. B. Hess, H. Bekker, H.J.C. Berendsen, J.G.E.M. Fraaije, J. Comput. Chem. **18**, 14631472 (1997)
23. S. Miyamoto, P.A. Kollman, J. Comput. Chem. **13**, 952962 (1992)
24. H. Okumura, S.G. Itoh, Y. Okamoto, J. Chem. Phys. **126**, 084103 (2007)
25. K.J. Bowers et al., in *ACM/IEEE Conference on Supercomputing (SC06)* (IEEE, Tampa, Florida, 2006)
26. K.J. Bowers, R.O. Dror, D.E. Shaw, J. Chem. Phys. **124**, 184109 (2006)
27. B.G. Fitch, A. Rayshubskiy, M. Eleftheriou, T.J.C. Ward, M. Giampapa, M.C. Pitman, J. Pitera, W.C. Swope, R.S. Germain, Comput. Model. Membr. Bilayers **60**, 159 (2008)
28. J. Jung, C. Kobayashi, T. Imamura, Y. Sugita, Comput. Phys. Commun. **200**, 57 (2016)
29. A.P. Hynninen, M.F. Crowley, J. Comput. Chem. **35**, 406 (2014)
30. B. Hess, C. Kutzner, D. van der Spoel, E. Lindahl, J. Chem. Theory Comput. **4**, 435 (2008)
31. T. Narumi et al., in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (ACM, 2006), p. 49
32. D.E. Shaw et al., in *34th Annual International Symposium on Computer Architecture (ISCA '07)* (ACM, 2007), p. 910
33. D.E. Shaw et al., in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (IEEE Press, 2014), p. 41
34. R. Salomon-Ferrer, A.W. Gotz, D. Poole, S. Le Grand, R.C. Walker, J. Chem. Theory Comput. **9**, 3878 (2013)
35. M.J. Harvey, G. Giupponi, G. De Fabritiis, J. Chem. Theory Comput. **5**, 1632 (2009)
36. P. Eastman et al., J. Chem. Theory Comput. **9**, 461 (2013)
37. A.P. Ruymgaart, A.E. Cardenas, R. Elber, J. Chem. Theory Comput. **7**, 3072 (2011)
38. J.C. Phillips, Y. Sun, N. Jain, E.J. Bohm, L.V. Kal, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (IEEE Press, 2014), p. 81
39. M.J. Abraham, T. Murtola, R. Schulz, S. Pali, J.C. Smith, B. Hess, E. Lindahl, Softw. X **1–2**, 19 (2015)
40. Jung et al., J. Chem. Theory Comput. **12**, 4947 (2016)
41. W.J. Harrod, A journey to exascale computing, in *The International Conference for High Performance Computing, Networking, Storage and Analysis, SC12* (2012), https://science.energy.gov/~/media/ascr/ascac/pdf/reports/2013/SC12_Harrod.pdf
42. https://software.intel.com/en-us/articles/explicit-vector-programming-in-fortran
43. https://software.intel.com/sites/landingpage/IntrinsicsGuide/
44. Y. Andoh, S. Suzuki, S. Ohshima, T. Sakashita, M. Ogino, T. Katagiri, N. Yoshii, S. Okazaki, J. Supercomput. **74**, 2449 (2018)
45. J. Jung, T. Mori, Y. Sugita, J. Comput. Chem. **34**, 2412 (2013)
46. T. Amisaki, J. Comput. Chem. **21**, 10751087 (2000)