Massive Parallelization of First Principles Molecular Dynamics Code

● Hidemi Komatsu ● Takahiro Yamasaki ● Shin-ichi Ichikawa

(Manuscript received April 16, 2008)

PHASE is a first principles molecular dynamics simulation program for explaining and predicting various properties of semiconductors and metals through electron-state calculations based on quantum dynamics. Its code was developed by the Revolutionary Simulation Software project sponsored by the Ministry of Education, Culture, Sports, Science and Technology (MEXT), and Fujitsu has been participating in the development. As part of the recent trend toward large-scale computing, the need for simulating tens of thousands of atoms has been increasing, and to this end, there is a need for high-speed massively parallel processing on tens of thousands of central processing units (CPUs) simultaneously. This, in turn, requires that simulation programs be converted to massively parallel code to support such highly parallel operations. In this paper, we describe the method that we used to massively parallelize the PHASE code; it is a method for achieving massive parallelization of a two-dimensional decomposition of kernel sections having high computational load in large-scale problems. We show that the amount of data transferred between CPUs (processes) here is small compared with the computational complexity, indicating that the performance for massively parallel processing should be high.

1. Introduction

First principles molecular dynamics is a simulation method that attempts to explain and predict various properties of semiconductors and metals by numerically solving electron states based on quantum dynamics. PHASE[1] is representative of code for performing such simulations. It has been developed under the Multiscale Simulation System for Function Analysis of Nanomaterials project in the Revolutionary Simulation Software project sponsored by the Ministry of Education, Culture, Sports, Science and Technology (MEXT).[2] Fujitsu participates in these projects, and one of the authors (T. Yamasaki) was one of the PHASE code developers.

Calculations by first principles molecular dynamics are performed directly according to quantum dynamics without using approximations based on classical dynamics. This means that the computational complexity increases dramatically as the scale of the problem being simulated increases: specifically, by the third power of the number of atoms. As a result, simulating tens of thousands of atoms, as is now desired, calls for a computer having high-speed computational abilities. Current simulation programs, however, typically handle only a few hundred atoms.

At the same time, high-speed scientific computers that are now being conceived will link many scalar central processing units (CPUs) on the order of tens of thousands of units and allot a computational load and data to each CPU with the aim of achieving massively parallel computing that, on the whole, can process large-scale problems at high speed. This, in turn, will require that simulation programs themselves be modified to a format suitable for massively paral-

lel processing.

In this paper, we first outline PHASE processing and describe the current parallelization method and a problem it has in high-speed execution. We then describe a massive parallelization method that we have developed to overcome this problem. Next, we describe a method that we apply before direct massive parallelization of the PHASE program to extract sections having a particularly large computational load in a large-scale problem and to prepare and massively parallelize three types of kernel code that models those sections. We also describe a method for transposing and transferring data between sections of kernel code in order to link high-computational-load kernel codes in PHASE. Finally, we evaluate data transfer complexity and computational complexity per process and show that data transfer between processes can be made sufficiently small compared with the computational complexity, which indicates that the scalability of the degree of parallelism should be high.

2. Outline of PHASE processing

The pseudopotential method used by the PHASE program groups together the atomic nucleus and core electrons, treating them as a frozen ion core, and solves only the state of valence electrons. This approach enables the prediction of many material properties. This method is called "pseudopotential" because the potential is treated on condition that the wave function agrees with the true wave function outside the bonding radius and changes smoothly in the vicinity of an atomic nucleus. Suppressing large fluctuations in the proximity of an atomic nucleus in this way makes it easy to expand wave function $\Psi(\vec{r})$ as plane wave functions $\exp(-i\vec{G}\cdot\vec{r})$, where $\vec{G}$ is the wavenumber in momentum space and $\vec{r}$ is a real space vector (the number of required wavenumbers $\vec{G}$ may be small). The pseudopotential consists of non-local terms having different scattering powers[note 1)] corresponding to each of the components of the wave function, such as the s, p, and d orbitals,[note 2)] and local terms having the same scattering power. Here, band (i) and the k-point are used as subscripts to distinguish the wave functions of individual valence electrons. Bands represent different energy levels of wave functions, while a k-point, as a quantum-state index reflecting the fact that crystals have translational symmetry,[note 3)] gives a distribution to a band. The result of multiplying this k-point index by Planck's constant[note 4)] and dividing by 2π yields the "crystal momentum". In semiconductors and insulators, electron occupied bands and unoccupied bands are determined without regard to k-points, and in large-scale systems, band distribution is small, which means that the electron state can be sufficiently reproduced with a small number of sampling k-points.

The parallelization of the original PHASE code is done solely on the basis of bands and sampling k-points, but in large-scale systems like semiconductors and insulators, the effect of parallelizing in the k-point direction is limited. This method also suffers from an insufficient number of bands when performing paralleliza-

note 1) Scattering power is the effect that core electrons near the atomic nucleus have on the outer orbitals of valence electrons.

note 2) Individual electrons enter the s, p, and d orbitals in that order from the atomic nucleus outward.

note 3) A crystal lattice possesses periodicity in that the lattice state does not change for spatial movement through the lattice by a distance equal to the lattice constant. This gives rise to momentum being conserved in electrons, which becomes crystal momentum corresponding to k-points. Crystal momentum gives width (distribution) to band energy levels.

note 4) Planck's constant is a universal constant of the natural world that converts the wave nature of a single electron into a physical quantity. Multiplying frequency by Planck's constant gives energy and multiplying wave number by Planck's constant gives momentum.

tion on the level of tens of thousands of CPUs (processes), which prevents sufficient parallelism from being obtained. Furthermore, if we were to try parallel decomposition in only the band direction, the number of bands per process would be small, making it difficult to obtain acceptable computational performance on a scalar computer. There is also a method of one-dimensional parallelization in the wavenumber direction, but this generates inter-process data transfer proportional to the square of the total number of band elements, resulting in a large transfer load when the degree of parallelization is high. In light of the above, we used two-dimensional parallelization in the band (i) direction and wavenumber (G) direction on a plane wave basis.

3. Extraction of high-load sections in PHASE and massive parallelization

When dealing with large-scale problems in PHASE, there are three types of code called "kernel sections" in which the computational workload (complexity) is greatest:

1) pseudopotential and wave function product
2) Gram-Schmidt orthogonalization
3) three-dimensional fast Fourier transformation (FFT)

The pseudopotential and wave function product constitutes a section that calculates how individual valence electrons are affected using the non-local terms of pseudopotential. A wave function that evolves over time using the pseudopotential and wave function product must be made to satisfy the orthogonalization conditions by Gram-Schmidt orthogonalization. Three-dimensional FFT converts the representation of a wave function in wavenumber space to a representation in real space and vice versa. This transformation must be performed when calculating the effect of local potential on an electron or when configuring charge density from valence electrons. The processing of these three sections in the original code of PHASE is shown in **Figure 1**.

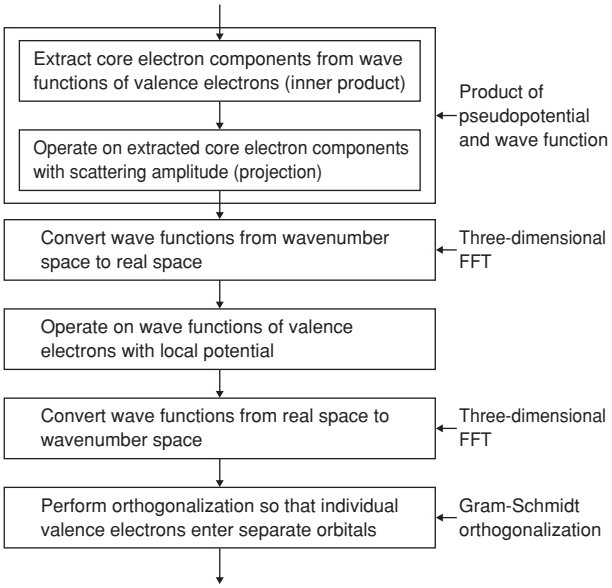

Figure 1
Outline of original PHASE code.

In recent research, we prepared code modeled on these kernel sections and massively parallelized it. The massive parallelization method is described below and the numerical formulas show computational algorithms.

1) Pseudopotential and wave function product

In PHASE, core electrons near each atomic nucleus are embedded on the potential side as a pseudopotential, and only the wave functions of valence electrons on the outside of that potential are solved. This high-load section in PHASE that takes the product of non-local pseudopotential and the wave functions of valence electrons can be expressed as follows.

$$| \Psi_i' > = \sum_n D_n \mid \beta_n >< \beta_n \mid \Psi_i >$$

Here, $D_n \mid \beta_n >< \beta_n \mid$ is the pseudopotential produced by each atom, Ψ is the wave function of an electron, n is a subscript identifying the pseudopotential orbit as well as a subscript identifying each atom, and i is a subscript denoting the wave function (band). This computation can be divided into 1st and 2nd computation sections having a computational complexity

proportional to the third power of the number of atoms.

Pseudopotential product—1st computation section:

$$< \beta_n \mid \Psi_i > = \sum_G \beta_n^*(G)\Psi_i(G)$$

Pseudopotential product—2nd computation section:

$$\Psi'_i(G) = \sum_n D_n \beta_n(G) < \beta_n \mid \Psi_i >$$

Here, G is the wave number in momentum space. We note here that a minor term associated with reference energy is generated with reference to pseudopotential β, but we omit it in the above computations for the sake of simplicity.

We prepared kernel code for these computation sections modeled in the above way and performed parallelization by two-dimensional decomposition of that code in the band (i) direction and wavenumber (G) direction. We incorporated coefficient D into β and reduced both of the above 1st and 2nd computation sections to matrix products. Moreover, to promote reuse of cache memory on scalar computers, we formed blocks of loops. The flow diagram for the massively parallel code of this pseudopotential product is shown in **Figure 2**. Performing parallelization by two-dimensional decomposition in this way enables a sufficiently large block size to be accommodated in each direction thereby maintaining computational efficiency. A new requirement here is an aggregate transfer (MPI_allreduce transfer) among processes in the wavenumber direction between the 1st and 2nd computation sections.

2) Gram-Schmidt orthogonalization

This section follows the pseudopotential product section described above, and like that section, it has a computational complexity proportional to the third power of the number of atoms. It is also divided into 1st and 2nd computation sections, but here they are repeated by turns. That is, all of the following is repeated for i from 1 to N, where N is the total number of bands.

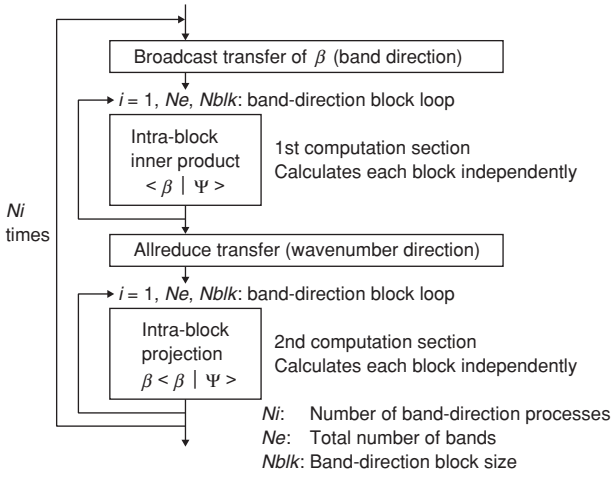

Figure 2
Massively parallel code of pseudopotential product.

Gram-Schmidt orthogonalization—1st computation section:

$$< \Psi'_j \mid \Psi_i > = \sum_G \Psi'_j(G)\Psi_i(G)$$

Gram-Schmidt orthogonalization—2nd computation section:

$$\mid \Psi'_i > = \mid \Psi_i > - \mid \Psi'_j >< \Psi'_j \mid \Psi_i >$$

Expressing the above for each G component, we get:

$$\Psi'_i(G) = \Psi_i(G) - \sum_{j=1}^{i-1} \Psi'_j(G) < \Psi'_j \mid \Psi_i >$$

In the original PHASE program, this Gram-Schmidt orthogonalization section was parallelized only in the wavenumber direction, and because this was different from the parallelization direction (band direction) of other sections, an inter-process data transpose transfer was applied before and after Gram-Schmidt orthogonalization. Here, however, to eliminate the need for this transpose transfer and obtain high parallelism, we applied a two-dimensional data distribution the same as in the other sections like the pseudopotential product and performed two-dimensional parallelization in the wavenumber (G) and band (i) directions. As in the pseudopotential product described above, loops are divided into blocks, with i in the above

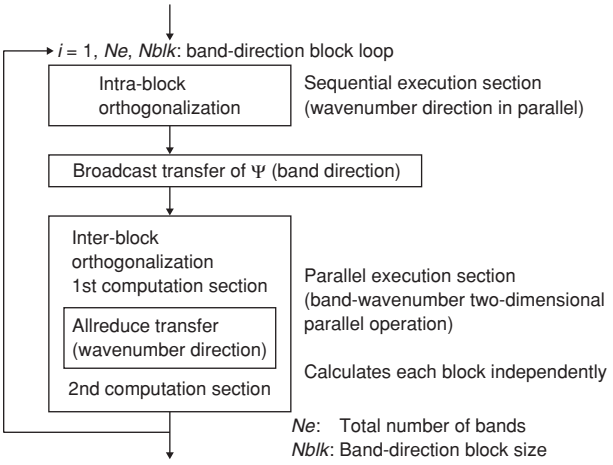

Figure 3
Massively parallel code of Gram-Schmidt orthogonalization.

	(1,1)	(1,2)	(1,3)	(1,4)	⋯	(1,10 000)
	(2,1)	(2,2)	(2,3)	(2,4)	⋯	(2,10 000)
	(3,1)	(3,2)	(3,3)	(3,4)	⋯	(3,10 000)
	(4,1)	(4,2)	(4,3)	(4,4)	⋯	(4,10 000)
↑	(1,1)	(1,2)	(1,3)	(1,4)	⋯	(1,10 000)
Band (*i*)	(2,1)	(2,2)	(2,3)	(2,4)	⋯	(2,10 000)
direction	(3,1)	(3,2)	(3,3)	(3,4)	⋯	(3,10 000)
	(4,1)	(4,2)	(4,3)	(4,4)	⋯	(4,10 000)
	⋮	⋮	⋮	⋮	⋮	⋮
	(4,1)	(4,2)	(4,3)	(4,4)	⋯	(4,10 000)

Wavenumber (*G*) direction →

In the above table:
(*i*,*G*) = process number
(*i*: band direction, *G*: wavenumber direction)

Figure 4
Data distribution method in Gram-Schmidt orthogonalization.

expressions being the inner loop and *j* being the outer loop.

The flow diagram for the massively parallel code of this Gram-Schmidt orthogonalization is shown in **Figure 3** and the way data is distributed to the processes is shown in **Figure 4**. To improve the utilization rate of each CPU here, multiple blocks are cyclically placed in a single process in the band direction.

3) Three-dimensional FFT

In PHASE, when calculating, for example, the product of local potential and electron wave function, the wave function is transformed from wavenumber space to real space, and the calculation is performed by using the FFT. As a result, a three-dimensional FFT and its inverse function will be repeated any number of times. Denoting the total number of wavenumbers as N, this section requires a computational complexity proportional to (number of bands × $N \log N$), making it the section with the third highest load in PHASE. Moreover, in a manner similar to the pseudopotential product and Gram-Schmidt orthogonalization sections, this section can perform parallelization in both the band direction and wavenumber direction or in the band direction and real-space-coordinates direction.

After parallelizing this three-dimensional FFT section in the band direction and x, y, z wavenumber directions (Gx, Gy, Gz directions), we performed parallelization by two-dimensional decomposition even within the wavenumber direction, the same as in the method presented by Eleftheriou.[3] In short, when performing a one-dimensional FFT in the x direction, for example, two-dimensional parallelization will be performed in the y and z wavenumber directions.

Here, when shifting from one-dimensional FFT in the x direction to one-dimensional FFT in the y direction, a transpose transfer of x-direction data and y-direction data for each process (y-direction transpose transfer) is necessary. Likewise, when shifting from one-dimensional FFT in the y direction to one-dimensional FFT in the z direction, a transpose transfer of y-direction data and z-direction data for each process (z-direction transpose transfer) is necessary.

4. Kernel-linking transfer section

In the previous section, we described the parallelization methods for the three kernel sections in PHASE, but each of these sections features a different way of ordering data

(wave functions). There is therefore a need for inter-process data transfer that transposes data between these three kernel sections. In the following, we describe the transfer methods in the wavenumber and band directions in the case of massive parallelization.

1) Reordering transfer in the wavenumber direction

Except for the section related to three-dimensional FFT, an upper limit (cutoff value) is set in relation to absolute values of wavenumbers in the PHASE kernel sections. Therefore, data is reordered and calculated in the order of wavenumber absolute values. Specifically, in the pseudopotential product section and Gram-Schmidt orthogonalization section, wave function data $\Psi i(G)$ is arranged in order of wavenumber absolute value ($G^2 = Gx^2 + Gy^2 + Gz^2$), while in the three-dimensional FFT section, they are arranged in the order of x, y, z (Gx, Gy, Gz). Therefore, a data transpose transfer in the wavenumber direction must be performed before and after the three-dimensional FFT section. Given that the number of processes in the wavenumber direction at the time of massive parallelization is far greater than the number of wavenumber elements associated with each process, each wavenumber element is transferred directly to its target process by means of an index.

2) Reordering transfer in the band direction

Before entering Gram-Schmidt orthogonalization in PHASE, convergence can be accelerated by reordering wave functions that must be orthogonalized in the order of eigenvalues (intrinsic energies) for the band in question. There is therefore a need here for data transpose transfer in the band direction. In the massive parallelization methods presented here, the number of processes allotted in the band direction is small compared with the number of band elements associated with each process. We therefore use a method that groups together the data to be transferred to each process beforehand and transfers that data once for each process. The flow diagram for the data transfer code in the band direction is shown in **Figure 5**.

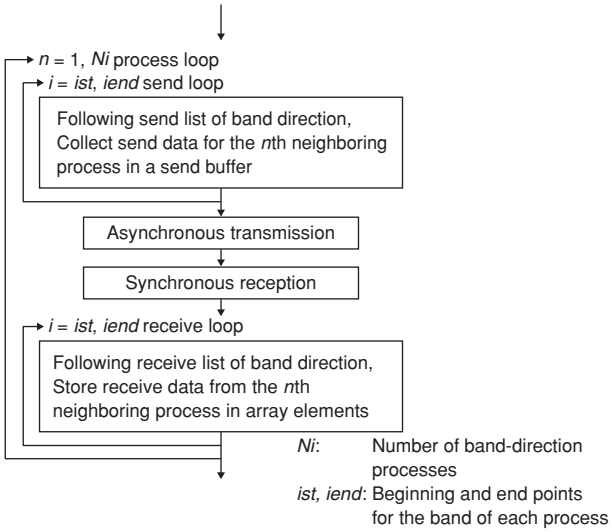

Figure 5
Data transfer code in band direction.

In the above, transposing tens of thousands of wave functions in the order of eigenvalues requires that index values be created and assigned beforehand by a sorting process that transposes eigenvalues in ascending order. Here, to carry out this sorting process in parallel, we use a method that combines the "distribution counting sort" method applicable to parallel execution and the "simple insertion sort" method. In other words, data distribution to each process is performed by the distribution counting sort while intra-process sorting of data allocated to each process is performed by the simple insertion sort. Finally, all the data is merged.

5. Computational complexity and transfer complexity in massive parallelization

Among the massively parallelized PHASE kernel sections described above, the sections with the highest loads in terms of both computational complexity and transfer complexity are the pseudopotential product section and

Gram-Schmidt orthogonalization section. The total computational complexity and total transfer complexity per process of these two sections plus the three-dimensional FFT section can be approximated as follows, where Ne denotes the total number of bands, Nf the total number of wavenumbers, Ni the number of processes in the band direction, and Nk the number of processes in the wavenumber direction. Here, we consider that the total number of pseudopotentials is nearly equivalent to the total number of bands Ne.

1) Pseudopotential and wave function product
   Computational complexity:
   $16 \times (Ne)^2 \times Nf / (Ni \times Nk)$
   Transfer complexity:
   $16 \times (Ne)^2 / Ni$ (bytes) (Allreduce transfer)
2) Gram-Schmidt orthogonalization
   Computational complexity:
   $8 \times (Ne)^2 \times Nf / (Ni \times Nk)$
   Transfer complexity:
   $8 \times (Ne)^2 / Ni$ (bytes) (Allreduce transfer)
3) Three-dimensional FFT
   Computational complexity:
   $5 \times Ne \times Nf \times logNf / (Ni \times Nk)$
   Transfer complexity:
   $16 \times Ne \times Nf / (Ni \times Nk)$ (bytes) (transpose transfer)

Here, parameter values for a scale of 10 000 atoms are of the order of Ne = 50 000 and Nf = 1 000 000 (supposed number of processes: Ni = 4, Nk = 10 000). In Gram-Schmidt orthogonalization, there is a process that performs orthogonalization within one block as preprocessing, and no parallel execution is performed in the band direction here. It is therefore desirable that the ratio of processes in the wavenumber direction be increased to improve the utilization rate of each CPU. In other words, increasing the number of band blocks associated with one CPU (one process) is equivalent to making one block's worth of preprocessing less noticeable. This is why the number of processes in the band direction Ni is made small here. The computational complexity per process of this preprocessing section is $8 \times Nblk \times Nf \times Ne/Nk$ and the transfer complexity is $8 \times (Nblk - 1) \times Ne$, where $Nblk$ is the block size. Placing multiple band blocks in one process helps to minimize the computational complexity and transfer complexity of this preprocessing section compared with later processing.

From the above, the ratio of transfer complexity to computational complexity in the pseudopotential product section and Gram-Schmidt orthogonalization section is given by

(transfer complexity) / (computational complexity)

$= Nk / Nf$.

This is the reciprocal of the number of wavenumber elements per process. In PHASE, the number of wavenumber elements is normally several hundred times the number of atoms. Thus, for an assumed number of processes on the order of Ni = 4 and Nk = 10 000, the ratio of transfer complexity to computational complexity can be made sufficiently small (about 1/100) and the transfer time can be kept short compared with the computational time. At the same time, both computational and transfer complexities in the three-dimensional FFT section are very small compared with the above two sections.

In addition, the total transfer complexity of the two transfer sections that link the above three kernel sections is given by

$16 \times Ne \times Nf / (Ni \times Nk)$ (bytes).

This is sufficiently small compared with the total transfer complexity of the pseudopotential product and Gram-Schmidt orthogonalization sections (about 1/500), so it is not expected to cause a significant drop in total computational performance.

Assuming that we use the Recursive Halving method[4] in the MPI_allreduce transfer in the pseudopotential product and Gram-Schmidt orthogonalization sections, which have the largest transfer load, and given that the data transfer

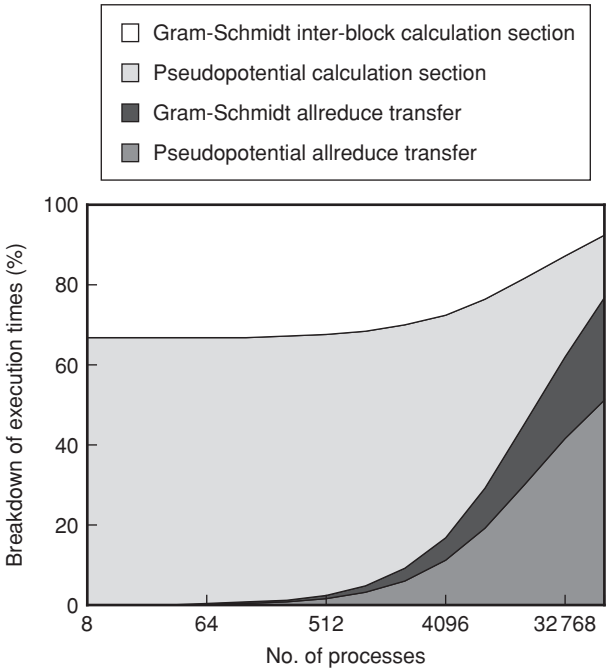

Figure 6
Breakdown of estimated execution time by one-dimensional decomposition in wavenumber direction.

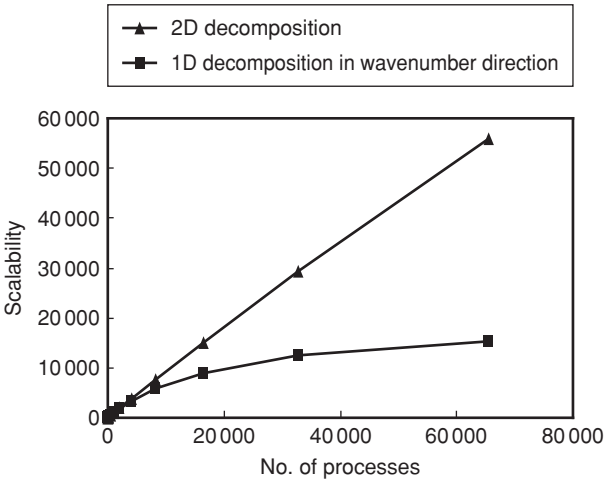

Figure 7
Estimated scalability by two-dimensional decomposition in wavenumber and band directions.

performance is uniform at 4 GB/s and computational performance is 100 Gflops (10^{11} operations per second) per process, the ratio of transfer time to computational time is given by

(transfer time) / (computational time)

$= 2 \times (Nk / Nf) \times (100/4)$.

This result tells us that the transfer time can be kept small compared with the computational time by two-dimensional decomposition.

The estimated execution time broken down by calculation and communication sections when performing one-dimensional decomposition in just the wavenumber direction is shown in **Figure 6**. In addition, **Figure 7** shows a graph of estimated scalability (speed-up factor) versus number of processes with respect to execution time with eight processes when parallelizing by one-dimensional decomposition in only the wavenumber direction and by two-dimensional decomposition as proposed here. Since the transfer load for the allreduce transfer, which has the biggest transfer load, can be reduced to the reciprocal of the number of parallel bands by two-dimensional decomposition, its transfer efficiency is better than that obtained by one-dimensional decomposition. This graph shows how significantly better performance can be obtained by using two-dimensional parallelization instead of one-dimensional parallelization, especially for parallel execution on the level of tens of thousands of processes.

6. Conclusion

To demonstrate the feasibility of massively parallelizing the first principles molecular dynamics simulation program PHASE for tens of thousands of CPUs, we extracted three high-load sections in the calculation of large-scale problems, prepared kernel code modeling these sections, and converted that kernel code for massive parallelization. We also described a data transpose transfer section for linking these three sections of kernel code and prepared code for this purpose.

Furthermore, upon investigating the computational complexity and transfer complexity with regard to this kernel linking code, we found that the data transfer complexity between CPUs could

be kept small compared with the computational complexity for parallel execution by tens of thousands of CPUs and that massive parallelization could be achieved for tens of thousands of processes (CPUs).

In future research, we plan to carry out performance evaluations on real machines using the kernel linking code described here. In this paper, we made estimations for large-scale parallelization by assuming the same network configuration as for a low degree parallelization, but we expect that a large-scale network capable of high-speed processing can be achieved for group communications in the manner of MPI_allreduce transfer, which has found widespread use in actual applications.

This research was performed as part of the Petascale System Interconnect project of MEXT, Japan.

References

1) RSS21 MEXT R&D Program for Constructing the Next-Generation IT Infrastructure, R&D of Strategic Simulation Software: PHASE Detailed Information.
http://www.ciss.iis.u-tokyo.ac.jp/rss21/en/theme/multi/material/index.html
2) RSS21 MEXT R&D Program for Constructing the Next-Generation IT Infrastructure, R&D of Strategic Simulation Software: Software Publication.
http://www.ciss.iis.u-tokyo.ac.jp/rss21/en/result/download/index.php
3) M. Eleftheriou et al.: Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: Implementation and early performance measurements. *IBM J. Res. Develop.* Vol.49, No.2/3, pp.457-464 (2005).
4) R. Thakur et al.: Optimization of Collective Communication Operations in MPICH. *Int. J. HPC. Appli.* Vol.19, No.1, pp.49-66 (2005).

Hidemi Komatsu
Fujitsu Ltd.
Dr. Komatsu received the Ph.D. degree in Astrophysics from the University of Tokyo, Tokyo, Japan in 1988. He joined Fujitsu Ltd., Japan in 1988 and has been engaged in parallelization and parallel tuning of user programs for supercomputers. He was a member of the Petascale System Interconnect Project of the Ministry of Education, Culture, Sports, Science and Technology (MEXT) in Japan from 2005 to 2007.

Takahiro Yamasaki
Fujitsu Laboratories Ltd.
Dr. Yamasaki received the Ph.D. degree in Solid State Physics from Osaka University, Osaka, Japan in 1989. He joined Fujitsu Laboratories Ltd., Atsugi, Japan in 1989 and has been engaged in research and development of semiconductor devices. He has been participating in the Revolutionary Simulation Software project of the Ministry of Education, Culture, Sports, Science and Technology (MEXT) in Japan from 2005. He is a member of the Physical Society of Japan.

Shin-ichi Ichikawa
Fujitsu Ltd.
Mr. Ichikawa received the B.S. degree in Physics from Tokyo University of Education, Tokyo, Japan and the M.S. degree in Nuclear Physics from Tohoku University, Sendai, Japan in 1977 and 1979, respectively. He joined Fujitsu Ltd., Japan in 1979 and has been engaged in the application of high-performance computers to scientific and technical research. He is a member of the Information Processing Society of Japan.