Available online at www.sciencedirect.com
ScienceDirect
Computer Physics Communications 177 (2007) 426–431
Computer Physics Communications
www.elsevier.com/locate/cpc

An efficient parallel implementation of the smooth particle mesh Ewald method for molecular dynamics simulations

Kwang Jin Oh [a,*], Yuefan Deng [b]

[a] *Supercomputing Center, Korea Institute of Science and Technology Information, Daejeon, Republic of Korea 305-806*
[b] *Department of Applied Mathematics, Stony Brook University, Stony Brook, NY 11794-3600, USA*

Received 16 December 2006; received in revised form 16 April 2007; accepted 13 May 2007

Available online 24 May 2007

Abstract

This paper focuses on the implementation and the performance analysis of a smooth particle mesh Ewald method on several parallel computers. We present the details of the algorithms and our implementation that are used to optimize parallel efficiency on such parallel computers.

PACS: 02.70.Ns; 07.05.Tp

Keywords: Parallel molecular dynamics simulation; Smooth particle mesh Ewald method; 3D fast Fourier transform

1. Introduction

The advent of massively parallel computer systems with thousands or more processors [1] makes developing applications algorithms, such as molecular dynamics simulation (MD) programs, and implementing them for substantial performance and scalability a serious challenge.

In this paper, we will describe the implementation of the parallel smooth particle mesh Ewald (SPME) method [2] using the volumetric decomposition scheme, instead of more conventional and easier slab decomposition scheme, for the 3D fast Fourier transform (FFT) and incorporate the parallel SPME routine into the MD package [3] we developed earlier. We will present, and analyze, the performance data and scaling behavior of the MD package with parallel SPME routine based on our volumetric decomposition scheme.

2. MD algorithms and parallel implementation

Molecular dynamics (MD), when it involves non-bounded long-range forces for proteins, is one of the most time-consuming applications of supercomputer resources. Many efforts have been invested to design efficient algorithms to reduce the complexity from $O(N^2)$ to $O(N \log N)$ involving fast Fourier transforms, for a system with N particles. However, these new algorithms that appear efficient on serial computers post serious challenges for parallel implementation.

2.1. Basic observations

Two popular decomposition schemes [4–6] for parallelizing MD, i.e., atom decomposition and spatial decomposition, possess very distinct properties. The atom decomposition scheme allows each processor equal partition of the atoms regardless of their localities which, obviously, requires it to communicate with almost all processors to get the coordinates of the atoms distributed to other processors. Thus, the communication cost becomes significant degradation to the parallel efficiency for large systems. On the other hand, the spatial decomposition scheme divides the simulation domain into sub-domains, usually, according to atom locality. For MD involving short-ranged interactions, the sub-domain is larger than the potential cutoff distance and thus it requires no communication for computing forces among atoms within one processor and occasional communication among the nearest neighbor sub-domains. In general, this spatial decomposition, requiring less communication, may result in higher parallel efficiency than the atom decom-

* Corresponding author.
E-mail address: koh@kisti.re.kr (K.J. Oh).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2007.05.005

position scheme, abet the fact that communication is needed when decomposing the domain into sub-domains. In fact, the details of spatial decomposition can also affect communication and it is desirable to reduce surface-to-volume ratio of the sub-domain [4]. The computational cost scales as $O(N/P)$ while the communication cost is proportional to the surface of the sub-domain boundaries and scales as surface-to-volume ratio $O((N/P)^{2/3})$. Then the ratio between communication and computation scales as $O((N/P)^{1/3})$. This means that volumetric decomposition with smaller surface-to-volume ratio scales better than slab-like decomposition. The relatively low communication cost of the volumetric spatial decomposition scheme has lead to it being adopted by many researchers.

Recently, Oh and Klein [7] described an efficient domain decomposition scheme for constant pressure and constant temperature MD simulation of molecular systems with bond constraints. In their study, atomic groups instead of atoms were distributed over domains, eliminating the communication steps during SHAKE/RATTLE procedure [8,9]. Depending on the size of atomic groups, more atoms may require communication from neighboring processors and thus more atoms may require non-bonded interaction calculations. However, such occurrence is sparse and the scheme showed a strong scalability.

2.2. Smooth particle mesh Ewald method

Smooth particle mesh Ewald (SPME) method is an efficient method for calculating the long-range forces and has been widely used as an alternative to the standard Ewald summation [10] mainly due to its computational efficiency. The computational time of the standard Ewald summation method scales as $O(N^2)$ while the best optimized Ewald summation method scales as $O(N^{3/2})$. On the other hand, the computational time of the SPME method scales as $O(N \log N)$, enabling the users to avoid using fast, but inaccurate, truncation method that ignores forces beyond the potential cutoff distance. This practice causes the well-known artificial effects on simulation results.

The SPME method utilizes interpolation and 3D fast Fourier transform (FFT) to approximate structure factor in reciprocal space sum of the standard Ewald summation method. To parallelize the SPME method, we use the volumetric decomposition scheme for 3D FFT. In the slab decomposition, each processor is assigned a slab of size $N \times N \times N/P$ for computing an $N \times N \times N$ FFT on P processors. Though this slab decomposition is effective in reducing communication for 3D FFT itself, one would need to communicate between processors for transferring data when the scheme is switched from the volumetric decomposition for domain to the slab decomposition for the 3D FFT and vice versa. More importantly, the scalability of the slab decomposition scheme is limited by N, the extent of the data along a single axis. To avoid this problem of poor scalability for large number of processors, Eleftheriou et al. designed the parallel schemes using volumetric decomposition and implemented it on IBM BlueGene/L [11,12], while Deng and his coworkers on QCDOC [13,14], which can distribute $N \times N \times N$ FFT over a maximum of N^2 processors.

In this paper, we concentrate on the charge–charge interaction. The charge–charge interaction energy of N-particle system is given by

$$V = \sum_{i}^{N-1} \sum_{j>i}^{N} \frac{1}{4\pi\varepsilon_0} \frac{q_i q_j}{r_{ij}}, \tag{1}$$

where q_i is charge of ith particle, r_{ij} is the distance between ith and jth particles, and ε_0 is dielectric constant in vacuum.

According to the standard Ewald summation method, the charge–charge interaction is given by

$$V = \sum_{i}^{N-1} \sum_{j>i}^{N} \frac{1}{4\pi\varepsilon_0} \frac{q_i q_j \operatorname{erfc}(\alpha r_{ij})}{r_{ij}} + \frac{1}{\varepsilon_0} \sum_{\vec{k}\neq 0} \frac{\exp(-k^2/4\alpha^2)}{k^2} \left|S(\vec{k})\right|^2 - \frac{1}{4\pi\varepsilon_0} \sum_{i}^{N} q_i^2 \frac{\alpha}{\sqrt{\pi}}, \tag{2}$$

where $\vec{k}$ is the reciprocal lattice vector and α is the Ewald parameter, and *erfc* represents complementary error function. The first term of Eq. (2) represents the real space sum and is summed over all pairs, while the second term represents the reciprocal space sum, and the third term represents the self energy term. The structure factor $S(\vec{k})$ is given by

$$S(\vec{k}) = \sum_{i} q_i \exp(i\vec{k} \bullet \vec{r}_i) \tag{3}$$

and is approximated by

$$S(\vec{k}) \approx b_1(k_1) b_2(k_2) b_3(k_3) F(Q)(k_1, k_2, k_3), \tag{4}$$

where

$$b_i(k_i) = \exp\left(2\pi i (n-1) k_i / K_i\right) \times \left[\sum_{j=0}^{n-2} M_n(j+1) \exp(2\pi i k_i j / K_i)\right]^{-1} \times \left[\sum_{j=0}^{n-2} M_n(j+1) \exp(2\pi i k_i j / K_i)\right]^{-1}, \tag{5}$$

and

$$Q(k_1, k_2, k_3) = \sum_{i} \sum_{n_1, n_2, n_3} q_i M_n(u_{1i} - k_1 - n_1 K_1) \times M_n(u_{2i} - k_2 - n_2 K_2) \times M_n(u_{3i} - k_3 - n_3 K_3). \tag{6}$$

Here the scaled fractional coordinates u_i ranges from 0 to K_i for $i = 1, 2$, and 3. $M_n(u_i)$ is Cardinal B-splines of order n. $F(Q)$ is Fourier transform of Q.

Then the reciprocal space sum in the charge–charge interaction is given by

$$V = \frac{1}{2} \sum_{m_1=0}^{K_1-1} \sum_{m_2=0}^{K_2-1} \sum_{m_3=0}^{K_3-1} Q(m_1, m_2, m_3) \bullet (\theta_{\text{rec}} * Q)(m_1, m_2, m_3), \tag{7}$$

where

$$\theta_{\text{rec}} = F(B \bullet C) \tag{8}$$

$$B(m_1, m_2, m_3) = \left|b_1(m_1)\right|^2 \left|b_2(m_2)\right|^2 \left|b_3(m_3)\right|^2 \tag{9}$$

and

$$C(m_1, m_2, m_3) = \frac{1}{\pi V} \frac{\exp(-\pi^2 m^2/\beta^2)}{m^2}. \tag{10}$$

2.3. Implementation of the parallel 3D FFT

Implementation of 3D FFT on parallel computers is a multi-step process.

First, we decompose the data into sub-domains following volume decomposition scheme. For example, to solve a $64 \times 64 \times 64$ 3D-FFT problem on 8 processors, we map $64 \times 64 \times 64$ complex numbers to a mesh of $2 \times 2 \times 2$ processors. This could be the actual network topology of the computer such as BlueGene/L or an imagined mesh on a typical Beowulf cluster. After the mapping, each processor contains a $32 \times 32 \times 32$ complex-number sub-domain.

Second, we carry out the actual FFT computation. To do this, we do 1D FFT one dimension at a time in order of x-axis, y-axis and z-axis. When doing x-axis, one needs a whole line of original data along x-axis to be located in one processor. To avoid multi-processors doing the same line of 1D FFT, the communication between two nodes is required in the first machine dimension. For example, the 8-node machine has been imagined or mapped into $2 \times 2 \times 2$ mesh. One can put a half to the first processor with the other half in the second processor. After the communication, perform the actual 1D FFT within each of the processors independently, to complete the first-dimensional FFT. Before doing the second dimension, one moves the data back to where they were.

Third, we perform the same operation along the second dimension, the y-axis. We transfer the data along the second machine dimension to allow another 1D FFT to be performed in each processor independently. Actually, if FFTW is adopted, we can use its 1D FFT function for all three one-dimensional FFT similarly. The only possible difference is that the number of mesh points in each of the dimension, if it were not a cube like $64 \times 64 \times 64$. After this step, transfer the data back to where they were just like the first dimension.

Fourth, we perform similar steps as above for the third dimension, the z-axis. It needs the communication along the third machine dimension. So in total, one needs 6 global communication steps with each machine dimension two global communications. No matter how many processors in one dimension, global communication among processors in this one dimension is necessary.

In performing FFT for PME to calculate long-range electrostatic potential, one needs to use real-to-complex 1D FFT instead of complex-to-complex 1D FFT. This function is given in FFTW package. Special care must be given to the data layout. Actually, for the PME case, since one has to do the kernel multiplication followed by an inverse FFT to transfer back to real space, one can eliminate the last communication and the first communication in the inverse 3D FFT. Thus, the total number of communication steps is 10 instead of 12. One implementation specific comparison of blocking and non-blocking operations shows that the non-blocking operations perform better [15]. This observation suggests that one can achieve better performance by using ISEND and IRECEIVE in MPI so as to overlap the computational steps with the communicational ones. So in total, the communicational time is the only time it cost for small machine with fewer than 100 processors. If one uses more than 512 processors, overhead from software and hardware becomes significant.

2.4. Incorporation of parallel 3D FFT into SPME

The overall flow of parallel SPME routine is given by

1. Get charges on grid points,
2. Block the charge array,
3. Backward FFT,
4. Calculate energy and all-reduce,
5. All-gather and Forward FFT,
6. Calculate forces.

Since the charge array contains charges on the 3-dimensional grid of each domain and also contribution from atoms in neighboring domains, the data inside the array is not contiguous. That is to say, the data includes (large or small) groups of 0's between contiguous non-zero data. For this reason, we grouped the useful data (non-zero data) together into one contiguous array, while keeping track of how to reform the original array. This step requires $O(N)$ operations. An all-gather operation is performed so that each processor knows how to reform the data coming from each other processor. Next another all-gather is performed on the contiguous data block with no zero's. Using the information on how to reform the original array, the data is summed back into the original array, which requires $O(N \times P)$ operations. Using information for the FFT, only a subset of this data needs to actually be summed, which decreases the complexity of the summing part to $O(N/P)$.

In energy calculation, the loop index in x-direction ranges from 0 to NX/PX. The upper bound is NX/PX, not NX. Similarly, the upper bounds of the loop indices in y-direction and z-direction are NY/PY and NZ/PZ, respectively. The $(NX \times NY \times NZ)/(PX \times PY \times PZ)$ data set is the one that the FFT produced as output. After these calculations are performed, one global reduction needs to be done to have global energy and other properties like pressure tensor.

After the inverse FFT is performed, due to the decomposition of the FFT algorithm, each processor does not have the data it needs to move forward. It still has the slab decomposition data set from the FFT. To fix this, another improvised all-gather operation is performed on this data set. Then each processor has the correct part on the grid and will end with the same subset of the grid that it started with, which is necessary for the force calculations.

3. Performance and analysis

Our example system contains a protein (PDB id: 5DHFR) with CHARMM22 force field [16] and 7023 TIP3P [17] waters (see Fig. 1). The system contains a total of 23 558 atoms in simulation box having dimension 62.23 Å × 62.23 Å × 62.23 Å. The potential cutoff distance was set to 9 Å and the switching function was applied from 7.5 Å for the Lennard–Jones force calculation in real space. For the SPME calculation, K_1, K_2, and K_3 in Eq. (7) were set to 64 and the interpolation order was set to 4. To do the fast Fourier transform, we used FFTW [18]. The NPT ensemble with isotropic fluctuation of the simulation box was used for the numerical integration of the equations of motion [19,20]. The Nosé–Hoover chain is coupled to the system globally and another Nosé–Hoover chain to the barostat. Each Nosé–Hoover chain has 5 thermostats. We used 3 and 5 for n_c and n_{ys}, respectively. Here we chose NPT MD simulation because NPT MD simulation is more challenging than NVT MD simulation for parallel implementation [7]. All bonds including hydrogen atoms were constrained using SHAKE/RATTLE algorithms with a tolerance of 10^{-8}. The integration time step was set to 1 fs.

The example system has been simulated on Hamel cluster in KISTI (a Linux cluster, which is composed of 256 IBM x355 servers with Intel Pentium IV Xeon DP 2.8 GHz connected by a Myrinet2000 network). The maximum number of processors used in this study is limited to 64 processors due to the domain size. Note that the side length of the domain should be larger than the potential cutoff length in our domain decomposition implementation [7]. The domain dimensions used in the simulations were given in Table 1. In all simulations, the extended domain parameter δ was set to 1.5 Å. The details on δ was described elsewhere [7]. To obtain wall clock times in this study, we used clock() function in standard C library.

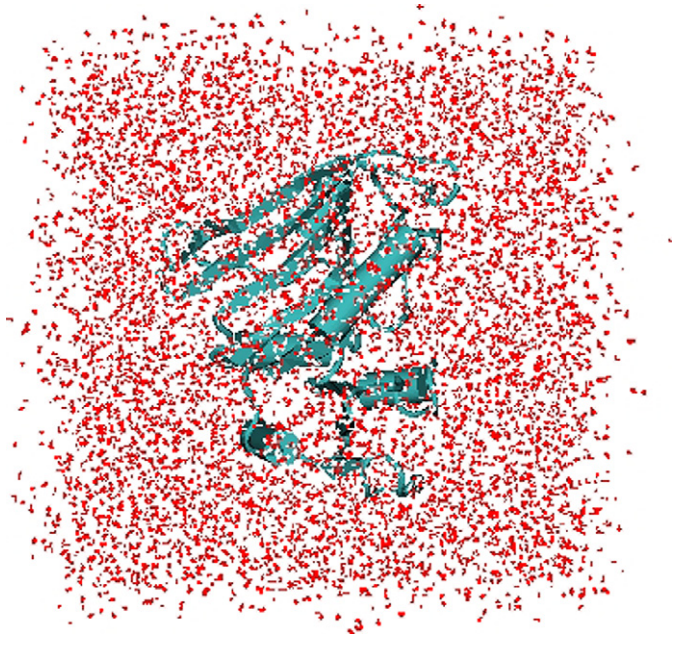

Fig. 1. A protein (PDB id: 5DHFR) plus 7023 TIP3P system used for performance test. The system contains total 23 558 atoms and the simulation box dimension is 62.23 Å × 62.23 Å × 62.23 Å.

Table 2 shows timing result from each step described in the previous section as a function of the number of processors and Fig. 2 shows its scaling behavior. Here we exclude the step (1) (to get the charges on the grid points) which does not affect scaling behavior.

From the table, we can find that two FFT steps are dominant for the total timing of the SPME routine. However the FFT steps show different scaling behaviors. As can be seen clearly in Fig. 2, the backward FFT step scales better than the forward FFT step. After 8 processors, the scaling of the forward FFT step was saturated. This is due to the all-gather operation (see 6th column in Table 2) to transfer the data set from the intrinsic slab-like layout of the FFT algorithm to the volumetric layout.

The force and energy calculation steps scale better than other steps. The poorer scalability of the energy calculation step than the force calculation step originates from the global reduction operation (see 5th column in Table 2) which is an inevitable step to get total energy and other properties.

Finally, the blocking step slows down and becomes dominant as the number of processors increases. Its scaling behavior is very poor. This is due to the all-gather operation to reform the data set to make it contiguous. For better scalability, in particular, on large number of processors, we plan to improve and optimize further this step.

To see the combined effects of all those steps, we examined scaling behavior of the MD code with the fully parallelized SPME routine. Fig. 3 shows the scalability curves which were obtained on two Linux cluster systems. One is KISTI Hamel cluster and the other is Nankai Star [21] which is composed

Table 2
Timing result from each step (in seconds for 100 steps)

Number of processors	Blocking	Backward FFT	Energy	All-reduce	All-gather	Forward FFT	Force
1	1.89	22.82	8.32	0.0	0.0	22.62	5.23
2	2.70	21.51	4.49	0.15	0.81	21.26	3.01
4	3.77	10.4	2.42	0.37	1.14	10.51	1.55
8	3.83	3.69	1.17	0.18	1.74	3.54	0.81
16	5.73	2.35	0.70	0.32	2.63	2.05	0.47
32	5.84	1.71	0.43	0.22	2.89	1.35	0.28
64	7.86	1.48	0.23	0.24	3.31	0.83	0.15

Table 1
Domain dimension used in simulation

Number of processors	1	2	4	8	16	32	64
Domain dimension	1 × 1 × 1	1 × 1 × 2	1 × 2 × 2	2 × 2 × 2	2 × 2 × 4	2 × 4 × 4	4 × 4 × 4

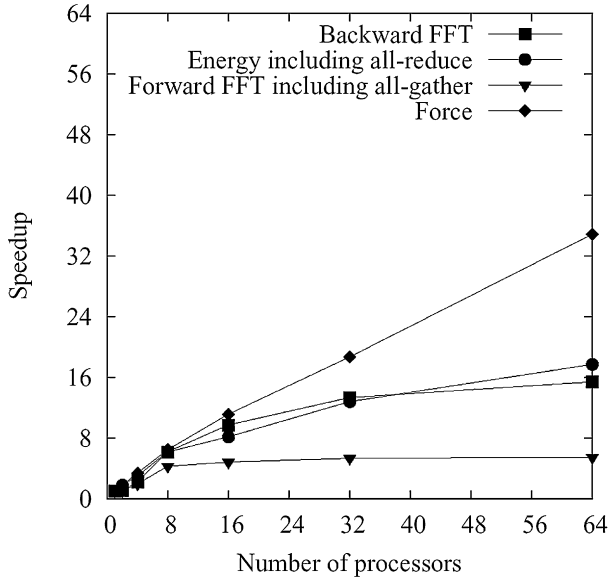

Fig. 2. The speedup from each step in SPME routine. The steps are backward FFT step, energy calculation step including all-reduce, forward FFT step including all-gather, and force calculation step.

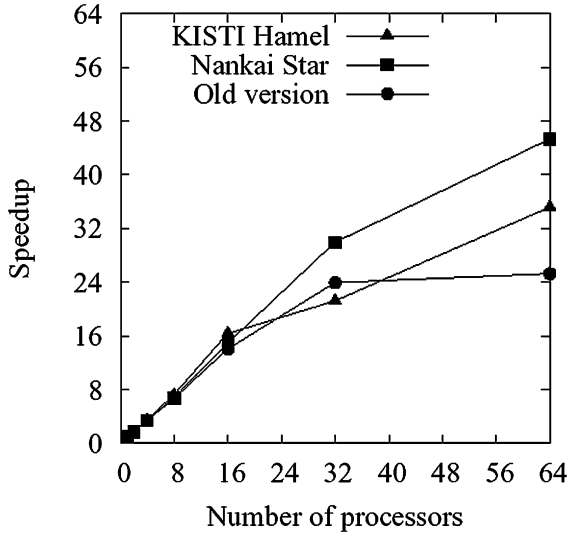

Fig. 3. The speedup of MD code with fully parallelized SPME routine on KISTI Hamel cluster and Nankai Star. The speedup of old version with partially parallelized SPME routine using serial FFT routine on KISTI Hamel cluster was also shown for comparison.

of 3.06 GHz dual Intel Pentium 4 Xeon processors connected by a Myrinet2000 network. The speedup at 64 processors on KISTI Hamel cluster and the Nankai Star was found to be about 35 (55% parallel efficiency) and 45 (70% parallel efficiency), respectively. From the figure, we can find that the MD code with the parallel SPME scheme shows very good scaling behavior overall despite a negative factor for the scalability, i.e. the blocking step as mentioned previously. For comparison with old version [3] where partially parallelized SPME routine using serial FFT is incorporated, we also showed scaling behavior of the old version in the figure. The speedup was about 25 (39% parallel efficiency) at 64 processors on KISTI Hamel cluster. Thus we highly improved overall scalability of the MD code using the scheme (parallel SPME with volumetric 3D parallel FFT) described in this study.

So far, we have focused on describing the improvement in the scaling behavior of the SPME routine and consequently the MD code. At this stage, it would be useful to check whether the MD code shows good single processor performance since it is an important factor for good parallel performance, i.e. good performance in timing and scaling. To that end, we compared the single-processor performance of our MD code with that of NAMD [22] (a highly optimized MD code). The timings were 1.24 seconds per step from our MD code and 0.52 seconds per step from NAMD, respectively. The MD code is about 2.4 times slower. However, the difference mainly originates from the non-bonded force calculation, not from the SPME calculation. The timings from the non-bonded calculation and the SPME calculation for the case were 1.02 seconds per step and 0.16 seconds per step, respectively. Based on this observation, we plan to optimize the non-bonded force calculation routine and improve its performance in the near future.

4. Conclusions

In this paper, we described a parallel SPME scheme based on 3D parallel FFT using volumetric decomposition method and presented performance results from a solvated protein NPT MD simulation (which would show poorer scalability than NVT MD simulation) on Linux cluster systems. From the performance results, we could find that the MD code with the fully parallelized SPME routine shows very good scaling behavior overall. We could also identify bottleneck which might reduce the scalability of the MD code, in particular, at large number of processors. Based on this observation, we plan to remove the bottleneck in the near future for application of the MD code to large scale simulation at large number of processors.

References

[1] http://www.top500.org.
[2] U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee, L.G. Pedersen, J. Chem. Phys. 103 (1995) 8577.
[3] K.J. Oh, M.L. Klein, Comput. Phys. Comm. 174 (2006) 560.
[4] S. Plimpton, J. Comp. Phys. 117 (1995) 1.
[5] S. Plimpton, B. Hendrickson, J. Comp. Chem. 17 (1996) 326.
[6] G.S. Heffelfinger, Comput. Phys. Comm. 128 (2000) 219.
[7] K.J. Oh, M.L. Klein, Comput. Phys. Comm. 174 (2006) 263.
[8] J.-P. Ryckaert, G. Ciccotti, H.J.C. Berendsen, J. Comp. Chem. 23 (1977) 327.
[9] H.C. Andersen, J. Comp. Phys. 52 (1983) 24.
[10] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids, Clarendon, Oxford, 1987.
[11] M. Eleftheriou, B.G. Fitch, A. Rayshubskiy, T.J.C. Ward, R.S. Germain, IBM J. Res. & Dev. 49 (2005) 457.
[12] M. Eleftheriou, J.E. Moreira, B.G. Fitch, R.S. Germain, Proceedings of the Conference on High Performance Computing 194 (2003).
[13] B. Fang, Y. Deng, J. Comp. Phys. (2006), accepted for publication.
[14] B. Fang, Y. Deng, G. Martyna, Comput. Phys. Comm. (2007), in press.
[15] http://www.llnl.gov/computing/tutorials/mpi_performance/.
[16] A.D. MacKerell Jr., et al., J. Phys. Chem. B 102 (1998) 3586.
[17] W.L. Jorgensen, J. Chandrasekhar, J.D. Madura, R.W. Impey, M.L. Klein, J. Chem. Phys. 79 (1983) 926.

[18] http://www.fftw.org.
[19] M.E. Tuckerman, G.J. Martyna, J. Phys. Chem. B 104 (2000) 159.
[20] G.J. Martyna, M.E. Tuckerman, D.J. Tobias, M.L. Klein, Mol. Phys. 87 (1996) 1117.
[21] Y. Deng, A. Korobka, and B. Xiang, Int. J. High Perf. Comp. Appl. (2006), submitted for publication.
[22] L. Kake, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, K. Schulten, J. Comp. Phys. 151 (1999) 282.