Parallel Implementation of the Treecode Ewald Method

Dongqing Liu*, Zhong-Hui Duan*, Robert Krasny†, Jianping Zhu‡§

* Department of Computer Science, The University of Akron, Akron, OH 44325, USA.
† Department of Mathematics, University of Michigan, Ann Arbor, MI 48109, USA
‡ Department of Theoretical and Applied Mathematics, University of Akron, Akron, OH 44325, USA
§ Corresponding author: jzhu@math.uakron.edu

Abstract

In this paper, we discuss an efficient parallel implementation of the treecode Ewald method for fast evaluation of long-range Coulomb interactions in a periodic system for molecular dynamics simulations. The parallelization is based on an adaptive decomposition scheme using the Morton order of the particles. This decomposition scheme takes advantage of the data locality and involves minimum changes to the original sequential code. The Message Passing Interface (MPI) is used for inter-processor communications, making the code portable to a variety of parallel computing platforms. We also discuss communication and performance models for our parallel algorithm. The predicted communication time and parallel performance from these models match the measured results well. Timing results obtained using a system of water molecules on the IA32 Cluster at the Ohio Supercomputer Center demonstrate high speedup and efficiency of the parallel treecode Ewald method.

1. Introduction

The rapid advancement in computational science has dramatically changed the way researchers conduct scientific investigations, particularly in the studies of molecular interactions in a biomolecular system [1]. However, the current modeling and simulation capabilities available to researchers are still vastly inadequate for the study of complex biomolecular systems, although there has been tremendous progress in computer hardware, software, modeling, and algorithm development [1, 2]. For example, the typical time step in a molecular dynamics (MD) simulation is on the order of 10^{-15} second, but the time interval of biological interest is typically on the order of 10^{-6} - 1 second. This translates to a simulation involving 10^{9} - 10^{15} time steps, where each step includes the computation of short-range interactions and extremely time consuming long-range Coulomb interactions. As a result, the longest simulation time that has ever been reported was 3.8×10^{-5} seconds [1, 3], which was achieved using a distributed parallel system with 5000 processors. To bridge the gap between the time interval of biological interest and that accessible by MD simulations, there has been extensive effort to improve the computational efficiency. One approach is to increase the time step size [4]. Another approach is to decrease the time needed for the force evaluations at each time step. This paper is concerned with the second approach.

In an MD simulation of a system of N particles, the computational cost is dominated by the frequent force evaluations. The calculation of long-range Coulomb interactions is the most time-consuming part of the evaluations. A straightforward implementation of the calculation requires $O(N^2)$ operations. In recent years, several methods have been developed to reduce the cost of computing Coulomb interactions while maintaining accuracy of N-body MD simulations, notably the multipole expansion based tree-codes and Ewald summation based fast algorithms [5-20]. Two of the earliest tree-codes were developed for problems involving gravitational interaction by Appel [7], Barnes and Hut [8]. Both algorithms employed monopole approximations with a complexity of $O(N\log(N))$. Appel used a cluster-cluster evaluation procedure, while Barnes and Hut used a particle-cluster procedure. Greengard and Rokhlin's fast multipole method reduces the operation count from $O(N^2)$ to $O(N)$ [9]. To obtain higher accuracy, the fast multipole method uses higher order multipole approximations and a procedure for evaluating a multipole approximation by converting it into a local Taylor series. There is much ongoing effort to optimize the performance of the fast multipole method. The latest version of the fast multipole method uses more sophisticated analytical techniques that combine the use of exponential expansions and multipole expansions [10]. The Ewald summation method has been widely used to handle Coulomb interactions for systems with periodic boundary conditions [15, 16]. The method converts a conditionally convergent series into a sum of a constant and two rapidly convergent series, a real space sum and a reciprocal space sum. The relative computational cost of the two series is controlled by an adaptive splitting

parameter a. The popular particle-mesh Ewald method [17] (PME) reduces the complexity of Ewald summation by choosing a large value for a; the real space sum is computed in O(N) and the cost of evaluating the reciprocal space sum is reduced from O(N^2) to O(Nlog N) using a particle-mesh interpolation procedure and the fast Fourier transform (FFT). PME was a major success in the field of biomolecular computing, but parallelizing PME has been a great challenge due to the high communication cost of 3D FFT. With a relative small value of a.? the Ewald summation based tree-code [18] approximates the real space sum using a Barnes-Hut tree and the multipole expansion of the kernel $erfc(|\mathbf{x}|)/|\mathbf{x}|$ and reduces the computational cost from O(N^2) to O(Nlog(N)).

While parallel implementations of hierarchical tree-codes, including the parallel fast multipole method and the parallel Barnes-Hut tree based multipole method, have been extensively studied [21-26], the parallel implementation of the Ewald summation based treecode [18] for fast evaluation of long-range Coulomb interactions in a periodic system has never been reported. In this paper, we present an efficient parallel implementation of this method. We describe an adaptive decomposition scheme based on the Morton order of the particles. This decomposition scheme takes advantage of the data locality and involves minimum changes to the original sequential code. We then discuss Amdahl's law and illustrate the performance improvement gained by making a common sequential part faster. The parallel algorithm has been implemented using the Message Passing Interface (MPI), making it portable to a variety of parallel platforms [27, 28]. We then report and discuss detailed numerical results based on the force evaluation for a water system on the IA32 Cluster at the Ohio Supercomputer Center. The next section describes the Ewald summation based treecode. The parallel implementation of the method and timing results are given in Section 3 and Section 4, respectively, followed by the conclusions in Section 5.

2. Treecode Ewald method

The total electrostatic energy of a periodic system of N particles can be described in terms of a conditionally convergent series:

$$E = \frac{1}{2}\sum_{\mathbf{n}}{}' \sum_{i=1}^{N}\sum_{j=1}^{N} q_i q_j \frac{1}{|\mathrm{r}_i - \mathrm{r}_j + L\mathrm{n}|} \qquad (1)$$

where q_i is the charge of particle i, L is the size of the simulation box, $\mathbf{r}_i$ is the position of particle i, $|\mathbf{r}|$ denotes the Euclidean norm, $\mathbf{n} = (n_1, n_2, n_3)$ are integers, and the prime indicates that the $i=j$ terms are omitted when $\mathbf{n}=\mathbf{0}$. A detailed description of the Ewald summation method is given in [15]. Essentially, the method splits the above conditionally convergent series into a sum of a constant, $E^{(0)}$, and two rapidly convergent series, a real space sum $E^{(r)}$ and a reciprocal space sum $E^{(k)}$,

$$E = E^{(0)} + E^{(r)} + E^{(k)} \qquad (2)$$

where

$$E^{(0)} = \frac{a}{p^{1/2}}\sum_{j=1}^{N} q_j^2 \qquad (3)$$

$$E^{(r)} = \frac{1}{2}\sum_{\mathbf{n}}{}' \sum_{i=1}^{N}\sum_{j=1}^{N} q_i q_j \frac{\mathrm{erfc}(a\,|\mathbf{r}_i - \mathbf{r}_j + L\mathbf{n}|)}{|\mathbf{r}_i - \mathbf{r}_j + L\mathbf{n}|} \qquad (4)$$

$$E^{(k)} = \frac{1}{2pL}\sum_{\mathbf{n}\neq\mathbf{0}} \frac{1}{|\mathbf{n}|^2}\exp\left(-\frac{p^2|\mathbf{n}|^2}{L^2a^2}\right)\left|\sum_{j=1}^{N} q_j \exp\left(\frac{2pi}{L}\mathbf{n}\cdot\mathbf{r}_j\right)\right|^2 \qquad (5)$$

and a is a positive parameter. The complementary error function in the real space sum and the exponential function in the reciprocal space sum decay rapidly with the index $\mathbf{n}$, therefore cutoffs r_c and k_c can be used to compute the Ewald sum, i.e. only terms satisfying

$$|\mathbf{r}_i - \mathbf{r}_j + L\mathbf{n}| \le r_c \qquad (6)$$

for $E^{(r)}$ and

$$|\mathbf{n}| \le k_c \qquad (7)$$

for $E^{(k)}$ are retained for the computation. The magnitude of the Ewald parameter a controls the relative rates of convergence of the real space sum and the reciprocal space sum. When a is large, $E^{(r)}$ converges rapidly and can be evaluated to a given accuracy in O(N) operations using an appropriate cutoff r_c; however in this case $E^{(k)}$ converges slowly and O(N^2) operations are required since the cutoff k_c must be large enough to attain the desired accuracy. The situation is reversed when a is small, and therefore in either case, the classical Ewald method requires O(N^2) operations. The cost can be reduced to O($N^{3/2}$) by optimizing the parameters a, r_c, k_c as a function of N [16]. The hidden constant in front of $N^{3/2}$ can be further reduced using the linked-cell method to reduce the computational cost of locating the particles that are within the cutoff radius of a given particle [6].

The Ewald summation based multipole method is a tree-code [18]. A typical tree-code has three basic features: (a) the particles are divided into nested clusters, (b) the far-field influence of a cluster is approximated using a multipole expansion, and (c) a recursive procedure is applied to evaluate the required force or potential. The Ewald summation based treecode uses an oct-tree data structure and Cartesian multipole expansions to approximate the real space sum in the Ewald summation. With a relatively small value for a and an appropriate cutoff r_c, the method reduces the computational complexity of the real space part from O(N^2) to O(Nlog(N)). The reciprocal part can be computed in O(N) operations using a cutoff k_c.

In the Ewald summation based treecode, the oct-tree is a Barnes-Hut tree. The root node is the cubic box

containing all the particles in the center simulation box. The root is subdivided in each coordinate direction into a total of eight children. The children define the next level of nodes in the tree. The subdivision continues until the number of particles in a node is less than or equal to a specified value N_0. These nodes form the leaves of the tree. In the implementation of the tree construction, the particle positions are sorted in a one-dimensional list based on their Morton order to preserve the spatial locality of the data [21, 29]. A 2-D case is illustrated in Figure 1. Each tree node includes a field, (head, tail), indicating that this tree node contains all the particles between head and tail on the list. Other bookkeeping steps are also performed during the tree construction. Attributes associated with a node such as the center and the radius of the node box as well as its multipole moments up to a chosen order p are computed.

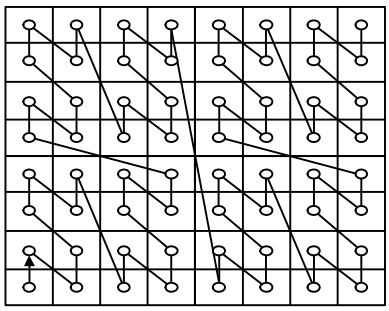

Figure 1. The Morton order of particles in a 2-D domain.

The far-field interaction --- the real space interaction between particle j and a distant cluster A is approximated using a p-th order Cartesian multipole approximation:

$$E_{j,A}^{(\mathbf{r})} \approx \frac{a}{\sqrt{p}} q_j \sum_{\|\mathbf{k}\|=\mathbf{0}}^{p} a_{\mathbf{k}} m_A^{\mathbf{k}} \qquad (8)$$

where $\|\mathbf{k}\| = k_1 + k_2 + k_3$, $a_{\mathbf{k}} = \frac{1}{\mathbf{k}!} D_{\mathbf{x}}^{\mathbf{k}} f(\mathbf{x}_A - \mathbf{x}_j)$ is the Taylor coefficient of the function

$f(\mathbf{x}) = \frac{\sqrt{p}}{2}\,\mathrm{erfc}(|\mathbf{x}|)/|\mathbf{x}|$, $\mathbf{x}_j = a\,\mathbf{r}_j$, $\mathbf{x}_A$ is the scaled center of cluster A, $\mathbf{k}! = k_1!k_2!k_3!$

$D_{\mathbf{x}}^{\mathbf{k}} = \frac{\partial^{\|\mathbf{k}\|}}{\partial x_1^{k_1} \partial x_2^{k_2} \partial x_3^{k_3}}$, $m_A^{\mathbf{k}} = \sum_{\mathbf{x}_i \in A} q_i (\mathbf{x}_i - \mathbf{x}_A)^{\mathbf{k}}$

is the $\mathbf{k}$th multipole moment of cluster A, and $(\mathbf{x} - \bar{\mathbf{x}})^{\mathbf{k}} = (x_1 - \bar{x}_1)^{k_1} (x_2 - \bar{x}_2)^{k_2} (x_3 - \bar{x}_3)^{k_3}$. The Taylor coefficients $a_{\mathbf{k}}$ are computed using a set of simple recurrence relations:

$$\|\mathbf{k}\|\mathbf{x}|^2 a_{\mathbf{k}} + (2\|\mathbf{k}\| - 1)\sum_{i=1}^{3} x_i a_{\mathbf{k}-\mathbf{e}_i} + (\|\mathbf{k}\| - 1)\sum_{i=1}^{3} x_i a_{\mathbf{k}-2\mathbf{e}_i} = \|\mathbf{k}\| b_{\mathbf{k}} \qquad (9)$$

and

$$b_{\mathrm{k}} + \frac{2}{k_i} x_i b_{\mathrm{k}-\mathrm{e}_i} + \frac{2}{k_i} b_{\mathrm{k}-2\mathrm{e}_i} = 0, \quad i = 1,2,3. \qquad (10)$$

With the constructed oct-tree, the force acting on a particle due to the interactions with particles within a cutoff range is computed by traversing the tree and using a divide-and-conquer strategy. The recursive evaluation procedure cycles through the particles and computes the interaction between a particle j and a cluster of particles A recursively. Interaction between a particle and a leaf cluster of particles is computed directly using the Ewald method. The multipole approximation is performed when the particle is well-separated from a nonleaf cluster, i.e. the following multipole acceptance criterion (MAC) is satisfied,

$$\frac{r_A}{R_{jA}} \le s \qquad (11)$$

where r_A is the cluster radius, R_{jA} is the particle-cluster distance, and s is a user-specified parameter for controlling the computational accuracy. If the MAC is not satisfied, the children of the given cluster are examined; this procedure is continued until a leaf cluster is encountered or the MAC is satisfied.

To reduce the cost of neighbor-finding, i.e. locating the particles that are within the cutoff radius of a given particle, the tree-code uses a 3-D Hockney-Eastwood lattice of ($M_1 \times M_2 \times M_3$) cells as a mesh to cover the center simulation box and the images of its nearest neighbors [6]. Each of the cells in the center simulation box is linked with a list of cells within its cutoff radius. The contributions to the force on a particle in a cell A are only from particles within cell A and in the cells that are linked to A. In addition, the hierarchical tree data structure also eliminates a cluster of particles with a single distance calculation when

$$|\mathbf{x}_\mathrm{j} - \mathbf{x}_\mathrm{A}| > r_\mathrm{c} + r_A. \qquad (12)$$

3. Parallel implementation

The main computation of an MD simulation includes force evaluations and velocity and position updates. The calculation of new velocities and positions can be readily parallelized once the force acting on each particle is obtained. The force evaluations of the MD simulations include the computations of short-range forces (bond-bond interactions and van der Waals interactions) and long-range forces (Coulomb interactions). Parallelizing the

computation of the short-range forces can be achieved through spatial decomposition and exchange of particle positions with neighboring processors [30]. However, parallelizing the computation of long-range forces is a challenge. For the Coulomb interactions, a particle in a charged system interacts with all other particles in the system. This means that to compute the Coulomb interactions, all-to-all data communications between different processors of a parallel computer is required. This situation is even more cumbersome when the simulation system is non-homogeneous and/or a typical pointer-based tree data structure is used.

Atom decomposition and spatial decomposition are the two commonly used methods for parallelization of an MD code [30]. In the atom decomposition method (also known as replicated-data method [31]), identical copies of particle information are stored on each processor and a pre-determined set of force evaluations is assigned to each processor for the entire duration of the simulation. The particles assigned to a processor may not have any spatial relationship to each other. One of the main advantages of this method is its simplicity. As a result, it has been widely used by many major MD programs including Amber and Charmm [32, 33]. In addition, because each processor has a copy of the entire data set, the computation of many body effects such as polarizability can be relatively easily added to the simulation. Spatial decomposition is another popular parallelization method [34]. In this method the domain is decomposed spatially and particles are assigned to each processor based on their geometrical positions. One clear advantage of this method is the consideration of the data locality and scalability. The adaptive domain decomposition scheme presented in this paper is a variant of the atom decomposition method, in which the spatial position of each particle is considered in the decomposition to ensure good data locality.

The sequential code of the Ewald summation based treecode uses a pointer-based oct-tree. The tree is constructed adaptively to deal with non-homogeneous systems, i.e. tree branches are taller in regions of high particle density and shorter in regions of low density. The implementation of the data structure is achieved by utilizing a list of particles in their Morton order. Based on our experience with the sequential code, the tree construction, even in a straightforward implementation, takes only a very small fraction, typically less than 0.4%, of the total computation time. To obtain an efficient parallel code for a moderate number of processors and minimize changes to the sequential code, we implemented a variant of the atom decomposition method in the parallelization of the Ewald summation based treecode. In our method, particle information, including particle positions and charges as well as input parameters such as the order of the approximation and multipole acceptance criterion, are replicated on each processor. Each processor constructs its own copy of the oct-tree and computes the multipole moments associated with each node of the tree. As a result, a unique list of the particles in Morton order is obtained on each processor when the tree construction is finished. Through the duplicated tree construction performed on each processor, the cumbersome all-to-all communication of the pointer-based oct-tree structure can be avoided.

With the one-dimensional list of the particles in spatial order, the force computation can be distributed across the processors by simply splitting the list into groups of the same computational workload, which can be approximated based on the computational cost of each particle at the previous time step. With our domain decomposition, the actual computation for the particles assigned to each processor can be performed in parallel without exchanging data with any other processors. After the force calculation is complete, each processor sends the computed forces acting on the particles assigned to the processor to all other processors and receives the force acting on other particles from other processors.

Table1. Morton order based atom decomposition algorithm.

Step	Description
1	Construct the oct-tree, sort the particles based on their Morton order to form a 1-D list, and compute the multipole moments of each tree node.
2	Compute forces on particles assigned to processor P_i based on the workload in the previous time step. The computation uses the oct-tree structure and the linked cells method. The distant particle-cluster interactions are approximated using multipole expansions.
3	Exchange forces with all other processors
4	Consolidate other parts of the force acting on each particle, integrate the equation of motion, and update the particle positions.

Table 1 outlines the Morton order based atom decomposition algorithm. The cost of the tree construction is of order $N\log(N)$, but as mentioned above, the proportionality constant is very small, making the cost of step 1 only a very small fraction of the total computation time. The second step is the most time consuming part of the computation. Each

processor is responsible for the force calculation of N/P particles, where P is the number of processors. Consequently, the computation time on each processor is proportional to $(N\log(N))/P$. The third step of the algorithm involves an all-to-all communication, but fortunately this is the only communication required in the method. Once the long-range force evaluation is finished, the forces due to other short-range interactions, including the reciprocal part of the Ewald summation, can be added to obtain the total force on each particle. Clearly the cost of this last step is proportional to N.

In our parallel implementation of the tree-code, the tree construction part is sequential although the whole computation has substantial parallelism. To minimize the computation in tree construction, we optimized this common sequential part of the tree-code. In the original sequential implementation, eight temporary linked lists are used to construct the tree and sort the particles in Morton order. When a tree node is divided into eight child nodes, the particles belonging to a child node are copied into the corresponding linked list. When the partition is finished, the particles on the eight linked lists are then copied back into the main list according to the order illustrated in Figure 1. In the current implementation, a variant of quicksort partition is used and accordingly the sorting is done in place with no need for any linked lists. This partition technique is an analog of the K-d tree construction technique. The initial list of the particle positions are partitioned along the x-axis into two sublists. Each sublist is then partitioned along y-axis, and then z-axis respectively. The enture process is repeated recursively until the number of particles in each node is less than or equal to a pre-determined number N_0. With this new tree construction technique, a substantial amount of tree construction time can be saved, thereby reducing the sequential computation time in the parallel code.

Load balancing is a key issue in the parallelization of any MD code, especially when the simulation systems under consideration are non-homogenous. The workload for computing the force at each particle depends on its local particle density and the particle's geometrical position. As pointed out earlier, the domain decomposition scheme splits the main list for the simulation system into groups of particles. The number of particles in each group is approximated based on the number of interactions that each particle was involved in the previous time step. With this weighted equal-size splitting technique a good load balancing can be achieved.

4. Numerical Results

The parallel treecode Ewald method is implemented using the C programming language and MPI on the IA32 Cluster at the Ohio Supercomputing Center [35]. The cluster is a distributed/shared memory hybrid system constructed from commodity PC components running the Linux operating system. Each compute node has two 1.4GHz Athlon MP processors, 2 GB of memory, and 70 GB of local scratch space. The nodes are connected using Myrinet 2000, a switched 2 Gbit/s network.

The test data is a set of 15,625 water molecules, which consists of 46,875 particles. The TIP4P water model [36] is used and a 1.6 picosecond molecular dynamics simulation is performed to generate the configurations of the water molecules [37]. The real space part of the potential and forces are computed based on this set of data, and the performance of the parallel implementation of the Ewald summation based treecode is also measured using this set of data.

Figure 2 and 3 show the execution times (wall-clock times) of the real space sum calculations for one time step using $a = 5.6/L$, $r_c = L/2$, and $N_0=20$. The order of the multipole expansion was taken to be $p=6$ and $p=9$ respectively. It is clear that significant reductions in runtime have been achieved as more processors are added. In the case of $p=6$, the execution time is reduced from about 100 seconds on one processor to less than 2 seconds on 64 processors. In the case of $p=9$, the execution time is reduced from about 200 seconds on one processor to less than 3.4 seconds. Note that the higher the order of the multipole approximation, the more computations are needed to calculate the forces while the communication cost remains the same. Hence, we expect better speedup and computational efficiency with higher order multipole approximations.

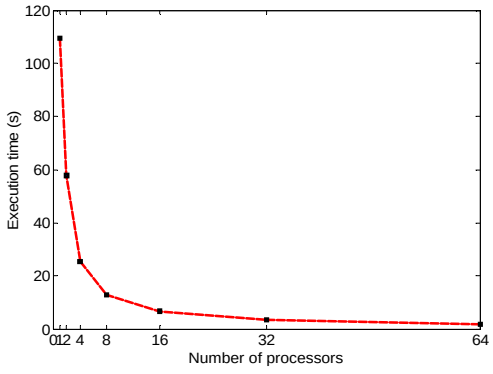

Figure 2. Execution time or calculating the real space sum using the parallel tree-code with the order of the multipole approximation p=6.

According to Amdahl's law [38, 39], the speedup of a parallel implementation of a computation algorithm

using P processors without communication cost is given by

$$S(P) = \frac{T_s}{(1-f)T_s + fT_s/P} \tag{13}$$

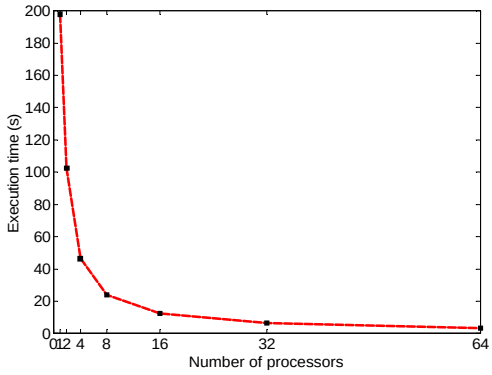

Figure 3. Execution time for calculating the real space sum using the parallel tree-code with the order of the multipole approximation *p*=9.

where T_s is the execution time of the sequential algorithm on one processor and f is the fraction of the computation that can be parallelized. The parallel efficiency is defined as

$$e(P) = \frac{S(P)}{P}. \tag{14}$$

In our calculations, the tree construction part, which takes about 0.278 seconds, is not parallelized. Since the total sequential computation times with p = 6 and p=9 are 109.3 seconds and 198.1 seconds, respectively, the fraction of parallelizable computation is 99.75% and 99.86%, respectively.

With communication cost, the Amdahl's model needs to be extended as

$$S(P) = \frac{T_s}{(1-f)T_s + fT_s/P + T_c} \tag{15}$$

where T_c is the communication time. Based on the architecture of the parallel machine used in our study, we use the following model to account for the all-to-all communication:

$$T_c = (ß + ?N/P)\log(P) \tag{16}$$

where ß is the effective startup latency, 1/? is the effective bandwidth for sending forces acting on particles, N is the number of particles, and P is the number of processors. Each processor is responsible for the force calculation of N/P particles. The term $\log(P)$ reflects the number of stages involved in an all-to-all communication for P processors. Using measured communication time and least-squares data fitting, we obtain ß = 4.6237×10^{-3} and ? = 1.3585×10^{-7}. Figure 4 shows both the measured communication time and that predicted by this model. It is clear that the model provides a good fit to the actual communication cost.

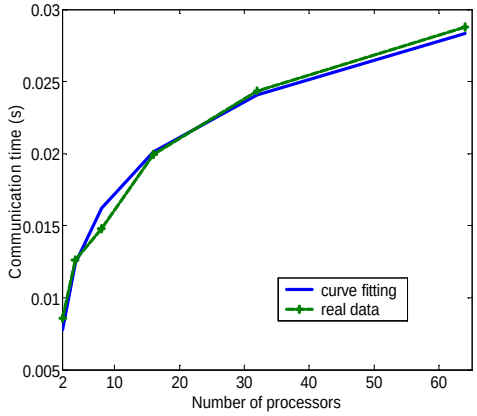

Figure 4. All-to-all communication time.

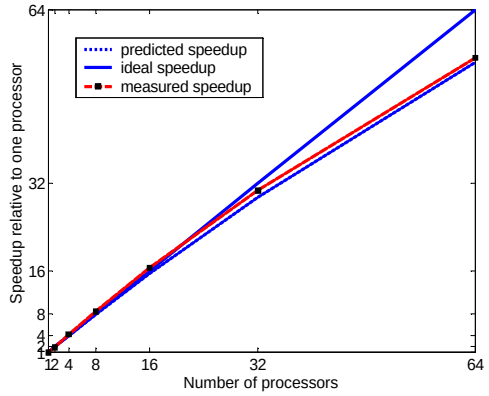

Figure 5. Speedup of the parallel tree-code with the order of the multipole approximation *p* = 6.

Figure 5 shows the ideal, measured, and predicted speedup using the speedup model Eq. 15 for the force calculation with p = 6. It is clear that very good speedup has been achieved. With 64 processors, the measured speedup is above 55, indicating a parallel efficiency of 86%. Further, we note that the speedup predicted by the model (Eq. 15) is very close to the actual measured speedup. Given the cache and memory hierarchy in the processors, computations involving smaller data arrays usually maintain better data locality and benefit more from high speed cache than those involving larger data sets, which means the decomposed subtasks can be done more efficiently on P processors than can a single task on one processor. This is not reflected in the speedup model of Eq. 15, which explains why the measured speedup is actually better than the predicted speedup.

Figure 6 shows the ideal, measured, and predicted speedup using Eq. 15 for the force calculation with p = 9. Since the communication cost does not increase as more terms are used in the calculation, the speedup is better compared to that in Figure 5. With 64 processors, the measured speedup is 59.6, indicating a parallel efficiency of 93%. For a similar reason as discussed in Figure 5, the measured speedup is better than the predicted speedup with more significant difference since more computations are involved using p=9. Although we only use up to 64 processors in our computations since it is difficult to obtain access to more processors on a shared cluster, the speedup model of Eq. 15 predicts a speedup of 104 on 128 processors, indicating a parallel efficiency of better than 81%. Since the real measured performance is better than the model prediction, we expect our algorithm to perform well on 128 processors.

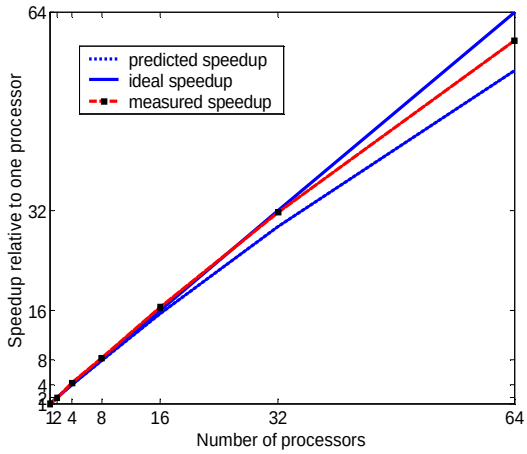

Figure 6. Speedup of the parallel tree-code with the order of the multipole approximation p = 9.

It is well known that for a given problem with fixed size, the parallel performance will eventually deteriorate as the number of processors increases. To maintain parallel efficiency, the problem size must increase as more processors are used, and a different measure, such as effective number of floating point operations per second, must be used to measure the parallel efficiency.

5. Conclusions

The work reported in this paper is a first attempt to parallelize the Ewald summation based treecode for long-range force calculations in a periodic system for molecular dynamics simulations. Our effort focused on portability, simplicity, and efficiency of the parallel code. The predicted results from our communication and performance models match the measured results well. Timing results obtained using a 46,875-particle water system demonstrates that our parallel algorithm scales well up to 64 processors. We expect the algorithm to maintain this good scalability on more processors with increased problem size. Future work includes investigation of parallel tree construction to increase the fraction f of parallelizable computations, and minimization of communication cost T_c to further improve parallel efficiency.

6. Acknowledgement

This work was supported in part by NSF Grants ACI-0081303, DMS-0075009, DMS-0107187, a start-up fund from the University of Akron, and Michigan Life Sciences Corridor grant #1515. The authors thank Dr. Timothy O'Neil for helpful discussions and the anonymous referees for their comments on the revision of this paper.

7. References

1. T. Schlick, *Molecular Modeling and Simulation*, Springer-Verlag, New York, New York, 2002.
2. T. Schlick, R.D. Skeel, A. T. Brunger, L. V. Kale, J. A. Board Jr., J. Hermans, and K. Schulten, Algorithmic challenges in computational molecular biophysics, *Journal of Computational Physics*, 151:9-48, 1999.
3. Y. Duan and P. A. Kollman, Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution, *Science*, 282:740-744, 1998.
4. T. Schlick, E. Barth, and M. Mandziuk, Biomolecular dynamics at long timesteps: bridging the time scale gap between simulation and experimentation, *Annu. Rev. Biophys. Biomol. Struct.*, 26:181-222, 1997.
5. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University, New York, 1987.
6. R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, McGraw-Hill, New York, 1981.
7. A. W. Appel, An efficient program for many-body simulation, *SIAM Journal on Scientific and Statistical Computing*, 6:85-103, 1985.
8. J. Barnes and P. Hut, A hierarchical *O(nlog(n))* force-calculation algorithm, *Nature*, 324:446-449, 1986.
9. L. Greengard and V. Rokhlin, A fast algorithm for particle simulations, *Journal of Computational Physics*, 73:325-348, 1987.
10. L. Greengard and V. Rokhlin, A new version of the Fast Multipole Method for the Laplace equation in three dimensions, *Acta Numerica*, 229-269, 1997.

11. K. E. Schmidt and M. A. Lee, Implementing the fast multipole method in three dimensions. *Journal of Statistical Physics*, 63:1223-1235, 1991.
12. H.-Q. Ding, N. Karasawa, and W. A. Goddard III, Atomic level simulations of a million particles: The cell multipole method for coulomb and London interactions, *Journal of Chemical Physics*, 97:4309-4315, 1992.
13. M. Challacombe, C. White, and M. Head-Gordon, Periodic boundary conditions and the fast multipole method, *Journal of Chemical Physics*, 107:10131-10140, 1997.
14. Z.-H. Duan and R. Krasny, An adaptive tree code for potential energy evaluation in classical molecular simulations, *Journal of Computational Chemistry*, 22:184-195, 2001.
15. S. W. De Leeuw, J. W. Perram, and E. R. Smith, Simulation of electrostatic systems in periodic boundary conditions. I. Lattice sums and dielectric constants, *Proceedings of the Royal Society of London. Series A*, 373:27-56, 1980.
16. J. W. Perram, H. G. Petersen and S. W. de Leeuw, An algorithm for the simulation of condensed matter which grows as the 3/2 power of the number of particles, *Molecular Physics*, 65:875-893, 1988.
17. T. Darden, D. York, and L. Pedersen, Particle mesh Ewald: An *Nlog(N)* method for Ewald sums in large systems, *Journal of Chemical Physics*, 98:10089-10092, 1993.
18. Z.-H. Duan and R. Krasny, An Ewald summation based multipole method, *Journal of Chemical Physics*, 113:3492-3495, 2000.
19. R. Krasny and Z.-H. Duan, Treecode algorithms for computing nonbonded particle interactions, Advances in Computational Methods for Macromolecular Modeling, *Lecture Notes in Computational Science and Engineering*, T. Schlick, H. H. Gan (eds.), Springer-Verlag, 359-380, 2002.
20. A. Y. Toukmaji, and J. A. Board, Jr., Ewald sum techniques in perspective: A survey, *Computer Physics Communications*, 95:73-92, 1996.
21. M. S. Warren and J. K. Salmon, A portable parallel particle program, *Computer Physics Communications*, 87:266-290, 1995.
22. J. K. Salmon and M. S. Warren, Skeletons from the treecode closet, *Journal of Computer Physics*, 111:136-155, 1994.
23. W. Rankin and J. Board, A potable distributed implementation of the parallel multipole tree algorithm, *Proceedings of IEEE Symposium on High Performance Distributed Computing*, pp.17-22, 1995.
24. J. A. Board, J. W. Causey, J.F. Leathrum, A. Windemuth and K. Schulten, Accelerated molecular dynamics simulation with the parallel fast multipole algorithm, *Chemical Physics Letters*, 198:89-94, 1992.
25. J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy, Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-Hut, Fast Multipole, and Radiosity, *Journal of Parallel and Distributed Computing*, 27:118-141, 1995.
26. H. Zeng, J. Devaprasad, M. Krishnan, I. Banicescu, and J. Zhu, Improving load balancing and data locality in N-body simulations via adaptive weighted fractiling, *Proceedings of 2002 High Performance Computing Symposium*, Society of Computer Simultation International, CD-ROM, San Diego, CA, April 14 - 18, 2002.
27. B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall, Inc., New Jersey, 1999.
28. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, second edition, The MIT Press, Cambridge, Massachusetts, 1999.
29. H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley Publishing Company, Inc., New York, New York, 1989.
30. S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, *Journal of Computational Physics*, 117:1-19, 1995.
31. W. Smith, Molecular dynamics on hypercube parallel computers, *Computer Physics Communications*, 62:229-248, 1991.
32. J. J. Vincent and K. M. Merz, Jr., A highly portable parallel implementation of AMBER 4 using message passing interface standard, *Journal of Computational Chemistry*, 16:1420-1427, 1995.
33. B. R. Brooks and M. Hodoscek, Parallelization of CHARMM for MIMD machines, *Chemical Design Automation News*, 7:16-22, 1992.
34. L. Kale, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan and K. Schulten, NAMD2: Greater scalability for parallel molecular dynamics, *Journal of Computational Physics*, 151:283-312, 1999.
35. OSC IA32 Cluster. http://oscinfo.osc.edu/computing/beo/
36. W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, Comparison of simple potential functions for simulating liquid water, *Journal of Chemistry Physics*, 79:926-935, 1983.
37. D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, Cambridge University Press, Cambridge, 1995.
38. D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, second edition, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.
39. J. Zhu, *Solving Partial Differential Equations on Parallel Computers*, World Scientific Publishing Co. Pte. Ltd., River Edge, New Jersey, 1994.