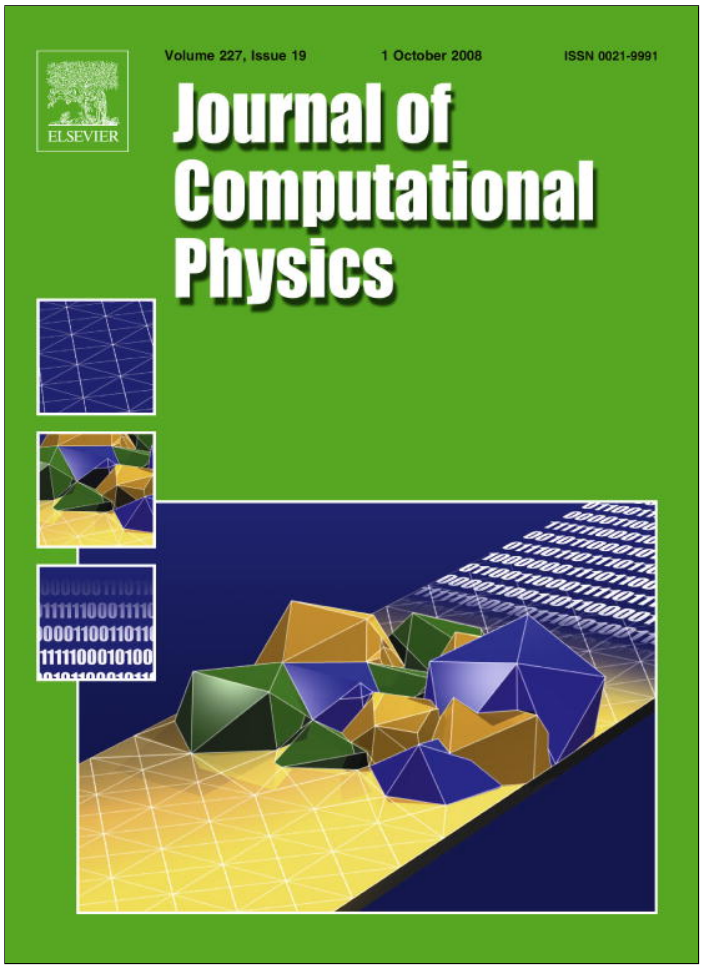
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Journal of Computational Physics 227 (2008) 8551–8567

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Fast electrostatic force calculation on parallel computer clusters

Amirali Kia [a,*,1], Daejoong Kim [b,1], Eric Darve [a]

[a] Mechanical Engineering, Stanford University, Mechanics and Computation Division, 496 Lomita Mall, Stanford 94305-4040, United States
[b] Mechanical Engineering, Sogang University, 1 Shinsu-dong, Mapo-gu, Seoul 121-742, Republic of Korea

ARTICLE INFO

Article history:
Received 15 November 2007
Received in revised form 25 May 2008
Accepted 11 June 2008
Available online 27 June 2008

Keywords:
Fast multipole method
Particle mesh Ewald
Plane wave
Parallel algorithm

ABSTRACT

The fast multipole method (FMM) and smooth particle mesh Ewald (SPME) are well known fast algorithms to evaluate long range electrostatic interactions in molecular dynamics and other fields. FMM is a multi-scale method which reduces the computation cost by approximating the potential due to a group of particles at a large distance using few multipole functions. This algorithm scales like $O(N)$ for N particles. SPME algorithm is an $O(N \ln N)$ method which is based on an interpolation of the Fourier space part of the Ewald sum and evaluating the resulting convolutions using fast Fourier transform (FFT). Those algorithms suffer from relatively poor efficiency on large parallel machines especially for mid-size problems around hundreds of thousands of atoms. A variation of the FMM, called PWA, based on plane wave expansions is presented in this paper. A new parallelization strategy for PWA, which takes advantage of the specific form of this expansion, is described. Its parallel efficiency is compared with SPME through detail time measurements on two different computer clusters.

© 2008 Elsevier Inc. All rights reserved.

1. Introduction

Typical biological molecules contain thousands of atoms and when water molecules are included the system size can go up to hundreds of thousands of atoms and more. In such systems, it becomes crucial to avoid computing all pair interactions between atoms, which would lead to a computational cost proportional to $O(N^2)$. This is the case for Coulombic potentials. Even though this potential is often truncated beyond a certain distance, accurate simulations require computing Coulombic interactions between all atoms.

Various techniques have been developed to handle this calculation efficiently among which we can mention Ewald summation, the fast multipole method (FMM) and particle–particle/particle–mesh (PPPM) whose computational cost reduces from $O(N^2)$ to $O(N^{3/2})$, $O(N)$ and $O(N \log N)$, respectively. FMM is therefore most efficient on very large problems ($N > 10^6$) while PPPM performs well on mid-size problems ($N \sim 10^3 - 10^5$) [1].

Particle mesh Ewald (PME) and SPME, which are both variations of PPPM, transfer charges from the particle positions to a Cartesian grid and solve the Poisson equation on this grid to approximate the potential. They both use FFTs to reduce the cost to $O(N \log N)$ [2–5].

The fast multipole method is based on the idea that a group of particles at a large distance can be approximated using few multipole functions. By clustering the system into bigger and bigger groups and using a tree structure, this eventually leads to an $O(N)$ algorithm [6].

* Corresponding author.
E-mail addresses: akia@stanford.edu (A. Kia), daejoong@sogang.ac.kr (D. Kim), darve@stanford.edu (E. Darve).
[1] Kia and Kim have contributed equally to this paper.

0021-9991/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2008.06.016

Author's personal copy

The emergence of large parallel machines demands scalable algorithms. An issue with biological systems is that the problem size is often, to a large extent, fixed and depends for example on the number of residues in the protein at hand. This is somewhat different from computational fluid dynamics where one can increase the mesh size to get more accurate results. Consequently, in molecular dynamics of bio-molecules, the amount of computational work to perform concurrently is often relatively small which can lead to poor efficiency unless specific optimizations are considered. Both FMM and PPPM have scalability issues for large distributed memory machines, as both these methods require significant communication between all processors. Hence, it would be advantageous to develop methods which have improved scalability and overall shorter runtimes on large parallel machines.

Fast multipole method falls into the category of tree-based methods (see e.g. Barnes and Hut [7]). There has been a lot of effort on the parallelization of tree methods [8–12]. Important issues include load balancing and data locality; the optimal algorithm is usually a compromise between these two requirements [12]. *Orthogonal recursive bisection* [8–10] and *costzones* [12] are among the proposed algorithms to partition the domain while keeping a good load balance. Although algorithms use different approaches for partitioning, they all decompose the physical domain into partitions and distribute them among processors. Every processor maintains a local view of the tree and traverses that tree performing operations on the multipole expansions. These algorithms require significant inter-processor communication [9,12]. On the order of $\ln N$ communication steps are typically required, where N is the number of particles.

Parallel algorithms for the specific case of FMM face similar issues as tree-based codes while having its own complexity. The tree building phase and the upward and downward passes through the tree for multipole and local expansions requires inter-processor communication and synchronization. In addition, inter-processor communication (but no synchronization) is required for the interaction list evaluations and direct potential calculations of the near neighbors at the leaf nodes [12]. Various techniques and algorithmic changes have been exploited to improve the load balancing and scalability of the algorithms [12–18]. Velamparambil and Chew address the issue of load balancing by replicating a few top level clusters in all processors and splitting the far-field patterns among processors to address the load balancing issue [13,14]. The scalability and load balancing have also been improved by overlapping communication with computation with a specific ordering of the communication steps [16,17].

We are proposing to improve the scalability of the FMM, especially in cases where communication occupies a significant fraction of the overall runtime, by taking advantage of a recently developed fast multipole formulation based on plane wave expansions [19]. Originally, the FMM uses spherical harmonics and Bessel functions to compute far-field expansions [6,19–21]. A recent variation of the FMM uses exponential or plane wave functions to approximate the long range interactions. These approximations have been used to accelerate transferring multipole to local coefficients in the FMM [19,22,23], and to solve Helmholtz and Maxwell equations with better stability and accuracy [24–29]. In this paper, we use an FMM based on plane wave functions. Using certain properties of this expansion, we propose a new parallelization strategy based on decomposing the quadrature points associated with the plane wave expansion among the processors. This is different from the classical domain decomposition approach. We show that this reduces considerably the amount of communication required and hence leads to better scalability of the algorithm.

This paper is structured as follows. In the first section we review the original FMM based on spherical harmonics. We introduce the plane wave approximation (PWA) and describe the relation between traditional FMM and PWA [30]. Accuracy of both FMM and PWA is reviewed [19]. Then, we discuss SPME and the various parameters involved in this method. This is important to understand the numerical results. In the third and fourth sections, we discuss parallelization of PWA and SPME method, respectively. We present the challenges and propose some strategies.

At the end, we present time measurements for both PWA and SPME on different parallel clusters and study communication vs. computation time in detail. We also discuss the effect of different networks on the communication time and consequently overall efficiency of the algorithms.

2. Fast multipole method

The FMM divides the N^2 pairwise interactions into far-field interactions and near-field interactions and then treats the former using multipole expansions and the latter by direct calculations. Assume we want to calculate a sum of the form:

$$\mathscr{U}(\mathbf{r}) = \sum_{i=1}^{N} \frac{q_i}{|\mathbf{r} - \mathbf{r_i}|}$$

The far-field is formally approximated by an expansion of the type:

$$\mathscr{U}(\mathbf{r}) \approx \sum_{k=1}^{p} A_k M_k(\mathbf{r}) \tag{1}$$

where

$$A_k = \sum_{i=1}^{N} q_i O_k(\mathbf{r}_i) \tag{2}$$

Author's personal copy

A_k's are called *moments* or *multipole coefficients*. For efficiency the integer p must be much smaller than N. The definition of the functions M_k and O_k will be presented in the next section.

The FMM algorithm is implemented using a hierarchical subdivision of the space in which clusters of points are recursively subdivided into smaller clusters until some criterion is met [20]. Before reviewing the main steps in the FMM, we recall the usual terminology:

Near neighbor:	two clusters are near neighbors if they are at the same refinement level and share a boundary point.
Well separated:	two clusters are well separated if they are at the same refinement level and are not near neighbors.
Interaction list:	for each cluster C, an interaction list is associated which consists of the children of the near neighbors of C's parent which are well separated from C.
Multipole expansion:	an expansion which approximates the far-field due to particles in a cluster.
Local expansion:	an expansion which approximates the potential in a cluster due to far away particles.

The FMM algorithm starts by computing multipole expansions at the finest level for every cluster. Then multipole expansions are computed at all levels by gathering expansions computed at finer levels. The multipole expansions are converted into local expansions and scattered down the hierarchical tree of clusters. Finally, the local expansions are used to evaluate the potential at all particle locations. This approximation is added to the potential due to particles in near neighbors which is directly computed.

2.1. FMM using spherical harmonics

2.1.1. Theory

Assume two particles at locations $\mathbf{r} = (r, \theta, \phi)$ and $\mathbf{a} = (a, \alpha, \beta)$. Let γ be the angle between $\mathbf{r}$ and $\mathbf{a}$. The multipole expansion uses spherical harmonics and associated Legendre functions $P_l^m(x)$ [6]:

$$\frac{1}{|\mathbf{r}-\mathbf{a}|} = \sum_{l=0}^{\infty} P_l(\cos\gamma)\frac{a^l}{r^{l+1}} = \sum_{l=0}^{\infty}\sum_{m=-l}^{l}\frac{(l-|m|)!}{(l+|m|)!}\frac{a^l}{r^{l+1}}P_l^{|m|}(\cos\alpha)P_l^{|m|}(\cos\theta)\mathrm{e}^{-\mathrm{i}m(\beta-\phi)} \tag{3}$$

Following the notation in [21], the multipole coefficients $\omega_l^m(q;\mathbf{a})$ for a charge q at $\mathbf{a}$ can be defined as follows:

$$\omega_l^m(q;\mathbf{a}) = qO_l^m(\mathbf{a}) \stackrel{\text{def}}{=} qa^l\frac{1}{(l+|m|)!}P_l^{|m|}(\cos\alpha)\mathrm{e}^{-\mathrm{i}m\beta} \tag{4}$$

Then, we have the following multipole expansion for the potential:

$$\frac{1}{|\mathbf{r}-\mathbf{a}|} = \sum_{l=0}^{\infty}\sum_{m=-l}^{l}\omega_l^m(q;\mathbf{a})\frac{1}{r^{l+1}}(l-|m|)!P_l^{|m|}(\cos\theta)\mathrm{e}^{\mathrm{i}m\phi} \tag{5}$$

The local coefficients μ_l^m for a far away particle with charge q at $\mathbf{r}$ can be defined as

$$\begin{aligned}\mu_l^m(q;\mathbf{r}) &= qM_l^m(\mathbf{r}) \stackrel{\text{def}}{=} q\frac{1}{r^{l+1}}(l-|m|)!P_l^{|m|}(\cos\alpha)\mathrm{e}^{\mathrm{i}m\phi}\\ \frac{1}{|\mathbf{r}-\mathbf{a}|} &= \sum_{l=0}^{\infty}\sum_{m=-l}^{l}\mu_l^m(q;\mathbf{r})a^l\frac{1}{(l+|m|)!}P_l^{|m|}(\cos\alpha)\mathrm{e}^{-\mathrm{i}m\beta}\end{aligned} \tag{6}$$

2.1.2. Translation and conversion operators

Assume that we have multipole expansions $\omega_l^m(\mathbf{O}) = \sum_i \omega_l^m(q_i;\mathbf{a}_i - \mathbf{O})$ where the sum is over all particles in a cluster with center $\mathbf{O}$. To translate those multipoles to center $\mathbf{O}+\mathbf{T}$, the following operator G is used [21]:

$$\begin{aligned}\omega_l^m(\mathbf{O}+\mathbf{T}) &= \sum_{j=0}^{p}\sum_{k=-j}^{j}G_{jk}^{lm}(\mathbf{T})\omega_l^m(\mathbf{O})\\ G_{jk}^{lm}(\mathbf{T}) &= O_{l-j}^{m-k}(\mathbf{T})\end{aligned}$$

Operator S translates local expansions μ_l^m from $\mathbf{O}$ to $\mathbf{O}-\mathbf{T}$:

$$\begin{aligned}\mu_l^m(\mathbf{O}-\mathbf{T}) &= \sum_{j=0}^{p}\sum_{k=-j}^{j}S_{jk}^{lm}(\mathbf{T})\mu_l^m(\mathbf{O})\\ S_{jk}^{lm}(\mathbf{T}) &= G_{jk}^{lm}(\mathbf{T}) = O_{l-j}^{m-k}(\mathbf{T})\end{aligned}$$

Author's personal copy

Operator T converts the multipole expansions into local expansions:

$$\mu_l^m(\mathbf{O}-\mathbf{T}) = \sum_{j=0}^{\infty}\sum_{k=-j}^{j} T_{jk}^{lm}(\mathbf{T})\omega_j^k(\mathbf{O})$$
$$T_{jk}^{lm}(\mathbf{T}) = M_{j+l}^{k+m}(\mathbf{T})$$

2.2. Plane wave approximation

PWA is based on the exponential expansion of $1/|\mathbf{r}|$. PWA differs from FMM in the sense that instead of a multipole expansion it uses a quadrature of an integral to approximate the potential. Beyond that, the other steps are similar to those of the FMM.

2.2.1. Theory

The kernel $1/|\mathbf{r}|$ can be written as an integral involving plane waves:

$$\frac{1}{|\mathbf{r}|} = \frac{1}{\pi}\int_0^{\infty}\int_0^{2\pi} \mathrm{e}^{-\chi^2 z}\mathrm{e}^{\mathrm{i}\chi^2(x\cos\phi + y\sin\phi)}\chi \mathrm{d}\chi\mathrm{d}\phi, \quad z>0 \tag{7}$$

where $\mathbf{r} = (x,y,z)$. By denoting $K_{\mathbf{r}}(\chi,\phi) \stackrel{\text{def}}{=} \exp(-\chi^2 z + \imath\chi^2(x\cos\phi + y\sin\phi))$ we have

$$\frac{1}{|\mathbf{r}|} = \frac{1}{\pi}\int_0^{\infty}\int_0^{2\pi} K_{\mathbf{r}}(\chi,\phi)\chi \mathrm{d}\chi\mathrm{d}\phi \tag{8}$$

The integral in (8) can be approximated by a discrete quadrature [31]:

$$\frac{1}{|\mathbf{r}|} \approx \sum_n v_n K_{\mathbf{r}}(\chi_n,\phi_n), \quad z>0 \tag{9}$$

where the χ_n and ϕ_n are the quadrature points and v_n's are the corresponding weights. (Note that v_n includes the pre-factor χ/π.) With this approximation, the potential can be approximated as

$$\sum_i \frac{q_i}{|\mathbf{r}-\mathbf{r}_i|} \approx \sum_n v_n \sum_i q_i K_{\mathbf{r}-\mathbf{r}_i}(\chi_n,\phi_n) \tag{10}$$

for $\mathbf{r}$ sufficiently far away from $\mathbf{r}_i$. Note that, since Eq. (7) holds only for $z>0$, the sum in Eq. (10) actually needs to be split in six different directions $(+z,-z,+x,-x,+y,-y)$ [19]. To simplify the discussion, we will ignore this dependence on the direction and assume that a single kernel K is needed.

Similar to the definition of $\omega_l^m(q;\mathbf{a})$ in Eq. (4), we define plane wave coefficients as

$$\omega_{\chi_n,\phi_n}(q;\mathbf{a}) \stackrel{\text{def}}{=} qK_{\mathbf{a}}(\chi_n,\phi_n)$$
$$\frac{1}{|\mathbf{r}+\mathbf{a}|} \approx \sum_n v_n \omega_{\chi_n,\phi_n}(q;\mathbf{a})K_{\mathbf{r}}(\chi_n,\phi_n)$$

Local expansions are defined as

$$\mu_{\chi_n,\phi_n}(q;\mathbf{r}) \stackrel{\text{def}}{=} qK_{\mathbf{r}}(\chi_n,\phi_n)$$
$$\frac{1}{|\mathbf{r}+\mathbf{a}|} \approx \sum_n v_n \mu_{\chi_n,\phi_n}(q;\mathbf{r})K_{\mathbf{a}}(\chi_n,\phi_n)$$

The desired accuracy is achieved by varying the number of quadrature points being used. This will be explained in details later when we discuss accuracy issues.

2.2.2. Translation and conversion operators

Obtaining the operators G, T and S for the PWA method is easier than for the FMM. They are simply defined as

$$G_{\chi_n,\phi_n}(\mathbf{T}) = S_{\chi_n,\phi_n}(\mathbf{T}) = T_{\chi_n,\phi_n}(\mathbf{T}) = K_{\mathbf{T}}(\chi_n,\phi_n) \tag{11}$$

The translation formulas are simple multiplications:

$$\omega_{\chi_n,\phi_n}(\mathbf{O}+\mathbf{T}) = G_{\chi_n,\phi_n}(\mathbf{T})\omega_{\chi_n,\phi_n}(\mathbf{O})$$
$$\mu_{\chi_n,\phi_n}(\mathbf{O}+\mathbf{T}) = S_{\chi_n,\phi_n}(\mathbf{T})\mu_{\chi_n,\phi_n}(\mathbf{O})$$
$$\mu_{\chi_n,\phi_n}(\mathbf{O}+\mathbf{T}) = T_{\chi_n,\phi_n}(\mathbf{T})\omega_{\chi_n,\phi_n}(\mathbf{O})$$

2.2.3. Discussion

In the paper by Greengard and Rokhlin [19], the plane wave integral is used to accelerate the transformation of multipole coefficients into local coefficients. This essentially replaces a multiplication by a dense matrix into a multiplication by a diag-

Author's personal copy

onal matrix, thereby reducing significantly the computational cost. However the basic expansion remains the multipole expansion, which is used to propagate expansion coefficients from the leaf nodes to the root of the tree and back down from the root to the leaf nodes. In this paper, we took a different approach where the same plane wave expansion is used at all levels. This requires using a quadrature accurate at all levels. In particular, this quadrature now depends on the total number of levels in the method which is not the case in the method of Greengard and Rokhlin [19]. This is not optimal from the standpoint of the total number of floating point operations to perform but we will see that this approach leads to improved parallel performance and smaller overall runtime. When the number of levels becomes very large, the number of quadrature points grows significantly, which is a drawback. Note, however, that the improvement in parallel performance advocated in this paper will be observed mostly for mid-size problems, for the reason that large problems are inherently easier to parallelize. The method is still of significant importance, for example in the field of molecular dynamics of proteins where most systems fall in fact in the mid-size category.

The approach in this paper can be extended to handle problems of arbitrary sizes while maintaining the same parallel efficiency. The extension consists in using a quadrature for the plane wave expansion which spans a fixed number of levels, say 3 or 4, and uses multipole expansions to propagate plane wave expansions to all levels. This leads to an algorithm which is efficient for problems of any size and with improved scalability. However this extension is not described in this paper because the authors decided to focus on mid-size problems which offer the biggest challenge in terms of their parallelization. Results are presented for systems with sizes ranging from 10,000 to 100,000 particles.

The key feature of the plane wave expansion is that to obtain a single local coefficient, say at (χ_n, ϕ_n), only plane wave coefficients at (χ_n, ϕ_n) are required. Information at other quadrature points (χ_m, ϕ_m), $m \neq n$, is not needed. Hence the calculation can be performed concurrently across quadrature points with no communication. This is the key advantage of this approach. This will be explained in more details in the section on parallelization.

2.2.4. Plane wave algorithm

We use the following notation:

C_k^l cluster k *at level* l
$\mathbf{O}_k^l$ center of cluster k at level l
L total number of levels in the tree

We briefly present the algorithm for PWA:

(1) *Initialization*:

$$f_{C_k^L}(\chi_n, \phi_n) = \sum_{\mathbf{r}_j \in C_k^L} q_j K_{\mathbf{O}_k^L - \mathbf{r}_j}(\chi_n, \phi_n) \tag{12}$$

(2) *Gather*: loop over $l = L : 3$

$$f_{C_k^{l-1}}(\chi_n, \phi_n) = \sum_m f_{C_m^l}(\chi_n, \phi_n) K_{\mathbf{O}_k^{l-1} - \mathbf{O}_m^l}(\chi_n, \phi_n) \tag{13}$$

where the sum is over all C_m^l which are the children of C_k^{l-1}.

(3) *Transfer*:

$$h^*_{C_k^l}(\chi_n, \phi_n) = \sum_m f_{C_m^l}(\chi_n, \phi_n) K_{\mathbf{O}_k^l - \mathbf{O}_m^l}(\chi_n, \phi_n) \tag{14}$$

where the sum is over all C_m^l in the interaction list of C_k^l.

(4) *Scatter*: loop over $l = 3 : L$

$$h_{C_k^l}(\chi_n, \phi_n) = h^*_{C_k^l}(\chi_n, \phi_n) + h_{C_m^{l-1}}(\chi_n, \phi_n) K_{\mathbf{O}_k^l - \mathbf{O}_m^{l-1}}(\chi_n, \phi_n) \tag{15}$$

where C_m^{l-1} is the parent of C_k^l.

(5) *Aggregation and direct summation*:

$$\mathscr{U}(\mathbf{r}_i) \approx \sum_n \omega_n h_{C_k^L}(\chi_n, \phi_n) K_{\mathbf{r}_i - \mathbf{O}_k^L}(\chi_n, \phi_n) + \text{short range} \tag{16}$$

for any $\mathbf{r}_i$ in cluster C_k^L.

2.2.5. Accuracy and generalized Gaussian quadratures

Yarvin and Rokhlin [31] have developed an algorithm to calculate generalized Gaussian quadrature points and weights for specific kernels. Here this algorithm is used to find a quadrature along χ for Eq. (8). The quadrature along ϕ is a set of uniformly distributed points. The quadrature for χ is designed to ensure a certain error ϵ for (x, y, z) in a specific set [19,31]. For each level, a quadrature can be computed which correctly approximates the integral in Eq. (8) at that level. It is also possible to design a quadrature which can be used for multiple levels with error ϵ. In the case of $1/r$, the number of quadrature points

Author's personal copy

is then a function of the total number of levels for which we want the quadrature to be accurate. As was explained previously, in this paper, we chose to use a single quadrature applicable at all levels in the tree. We discussed in Section 2.2.3 how this approach can be extended to relax this requirement.

2.2.6. Conversion between FMM and PWA

Multipole coefficients obtained using Eq. (4) in the FMM can be related to the plane wave coefficients through [19]:

$$f_{C_k^l}(\chi_n,\phi_n) = \sum_{m=-\infty}^{\infty} (-\mathrm{i})^{|m|} \mathrm{e}^{\mathrm{i}m\phi_n} \sum_{l=|m|}^{\infty} (-1)^m \omega_l^m (\chi_n^2)^l \tag{17}$$

A similar relation holds for the local coefficients in PWA and the local coefficients in the FMM defined by Eq. (6):

$$\mu_l^m \approx (-\mathrm{i})^{|m|} \sum_n v_n (-\chi_n^2)^l h_{C_k^l}(\chi_n,\phi_n) \mathrm{e}^{\mathrm{i}m\phi_n} \tag{18}$$

where v_n is the quadrature in PWA for (χ,ϕ). Note that in (17) we use truncated sum to approximate the plane wave coefficients but using the appropriate number of terms we can keep the error within desirable limit.

3. Particle mesh Ewald

3.1. Concept and theory

The particle mesh Ewald (PME) [3] and the smooth particle mesh Ewald (SPME) [2] are both descendants of the particle–particle/particle–mesh method [32]. These methods reduce the cost of computing the Ewald sum by first assigning charges to a Cartesian grid using interpolation and then applying the fast Fourier transform to evaluate the potential at all grid points. In this section, we briefly review the Ewald formulation and the SPME method. Finally, we examine the important parameters in this method. Because PME and SPME are very similar techniques, we shall refer exclusively to the SPME method bearing in mind that almost all conclusions are applicable to both methods. Moreover, since the SPME method or the Ewald method are not the main focus of this article, we avoid the mathematical details of these methods and refer the reader to other texts and articles on the subject.

3.1.1. Ewald sum

The total energy due to a set of N particles in a box with side length L and periodic boundary conditions is

$$E = \frac{1}{2} {\sum_{\mathbf{n}\in\mathbb{Z}^3}}' \sum_{i,j=1}^{N} \frac{q_i q_j}{|\mathbf{r}_{ij} + \mathbf{n}L|} \tag{19}$$

where the prime indicates that in the case of $i = j$ the term $\mathbf{n} = \mathbf{0}$ must be omitted.

The Ewald method splits the evaluation of the potential into *real space* and *reciprocal space* parts. The reciprocal space part is performed in the Fourier space and takes care of periodic boundary conditions. The total potential at every point is the sum of these three terms [1]:

$$\mathscr{U} = \mathscr{U}_{\mathrm{real}} + \mathscr{U}_{\mathrm{rec}} + \mathscr{U}_{\mathrm{corr}} \tag{20}$$

where

$$\mathscr{U}_{\mathrm{real}}(\mathbf{r}_i) = \sum_{j=1}^{N} {\sum_{\mathbf{n}\in\mathbb{Z}^3}}' q_j \frac{\mathrm{erfc}(\beta|\mathbf{r}_j - \mathbf{r}_i + \mathbf{n}L|)}{|\mathbf{r}_j - \mathbf{r}_i + \mathbf{n}L|} \tag{21}$$

$$\mathscr{U}_{\mathrm{rec}}(\mathbf{r}_i) = \frac{1}{L^3} \sum_{\mathbf{k}\neq 0} \sum_{j=1}^{N} \frac{4\pi q_j}{\mathbf{k}^2} \exp(-\mathbf{k}^2/4\beta^2) \exp(\mathrm{i}\mathbf{k}\cdot(\mathbf{r}_i - \mathbf{r}_j)) \tag{22}$$

$$\mathscr{U}_{\mathrm{corr}}(\mathbf{r}_i) = -2q_i \frac{\beta}{\sqrt{\pi}} \tag{23}$$

3.1.2. SPME

The particle–mesh schemes in general interpolate the charge of the particles to a grid and solve the discretized Poisson equation on that grid. Since it uses the fast Fourier transform (FFT), SPME scales as $O(N_{\mathrm{grid}} \log N_{\mathrm{grid}})$ where N_{grid} denotes the number of points used in discrete Fourier transform [1].

The main steps involved in a particle–mesh method and in particular the SPME method are:

- *Charge assignment*: As mentioned before, the charge density should be transferred to a grid using a proper interpolation scheme.
- *Solving poisson equation*: The discretized Poisson equation is solved using fast Fourier transforms.

Author's personal copy

- *Differentiation*: Once the potentials are obtained on the grid, they can be differentiated to obtain forces. This can be done either in the Fourier space, or in the real space from the values at the grid points [5].
- *Back-interpolation*: Potential and force values on the mesh are transferred to the particles' location using the same interpolation scheme.

3.1.3. Parameters in SPME

SPME has many parameters that control both the accuracy and the speed of the method. Some of these parameters also depend on each other. For example, to maintain a certain accuracy, the number of grid points depends on the cut-off radius in real space. Hence the optimum value for one parameter may vary depending on the other parameters. Here we explore the role of these parameters in relation to the parallel efficiency of the algorithm; we refer the reader to other articles for more elaborated analysis of these parameters [1–5]. The real space cut-off radius r_c, the Ewald coefficient β, the interpolation order n and number of grid points $\mathbf{N} = (N_x, N_y, N_z)$ are the major parameters.

The Ewald coefficient controls the relative computation cost of the reciprocal space vs. the real space. Eqs. (21) and (22) show that changing β has opposite effect on those equations in terms of the accuracy. For instance, increasing β requires a larger cut-off frequency in the reciprocal space and a smaller cut-off radius in the real space to maintain the accuracy. An optimal value for β can be defined by minimizing the computational cost at a specified accuracy. This optimal value of β depends on both the number of grid points and the interpolation order [1].

The interpolation order and scheme has direct influence on the algorithm. The particle mesh Ewald (PME) uses Lagrange polynomials while the smooth particle mesh Ewald uses B-splines, which leads to higher accuracy [2]. Changing the interpolation order also changes the optimal value of β. By increasing the order we have to use slightly higher values of β to keep the computation time minimum [5].

The number of the grid points controls the accuracy of the potential computed in the reciprocal space and also has a considerable effect on the computational time because the FFT scales as $N_{\text{grid}} \ln(N_{\text{grid}})$. An optimal performance is achieved through a suitable combination of both the number of grid points and the interpolation order [5].

4. Parallel algorithm

To have an efficient parallel simulation, we need a proper ratio of computation to communication. For example in computational fluid dynamics (CFD) simulations, because of the large number of mesh points we can usually decompose the computational domain so that the volume to surface ratio is large enough. In addition, we can refine the grid to get better resolution if necessary. Hence, adding more processors often does not reduce the efficiency of the parallel simulation.

On the other hand, in a molecular dynamics (MD) simulation, especially for a medium size macromolecule with 10^4 to 10^5 number of atoms, adding more processing units often lead to marginal improvements because the ratio of computation to communication is not favorable.

In this paper, we try to devise an algorithm which improves the ratio of computation vs. communication and reduces the overall runtime especially as more processors get applied to the calculation.

In the following sections we review the ideas behind parallelizing PWA with the goal of minimizing the communication steps. We also present the classical approach to parallelize SPME. Furthermore, we explore the effects of the different parameters on both methods. This is of practical importance because non-trivial settings are necessary to achieve shorter total runtime when utilizing many processors.

4.1. Parallelizing PWA

In the PWA algorithm the potential is divided into a potential due to the far-field particles and nearby particles. To make the first part parallel, we take advantage of the unique feature of the PWA formulations which resides in Eq. (14). For a pair (χ_n, ϕ_n) at level l, the local coefficient $h_{C_k^l}(\chi_n, \phi_n)$ in C_k^l can be computed from the plane wave coefficients $f_{C_m^l}(\chi_n, \phi_n)$ in C_m^l where m is varied. Hence, the calculation can be decomposed for parallelization purposes over the quadrature points (χ_n, ϕ_n). This approach is different from the standard domain decomposition approach. This is possible only because of the special form assumed by Eqs. (13)–(15).

Fig. 1 shows two different approaches to make the algorithm parallel. In the first approach, as shown in Fig. 1 (top row) each processor is assigned a subset of the clusters. It computes all the coefficients for these clusters. This method requires communication steps to transfer (and possibly gather and scatter) the coefficients. The second approach, as depicted in Fig. 1 (bottom row), uses the (χ, ϕ) space to make the algorithm parallel. Every processor will compute a subset of the coefficients (χ_n, ϕ_n) in the plane wave expansion but for all the clusters. This means that every processor computes only part of the potential for each particle. The most efficient way to do this is for each processor to store the position of all the particles.

The second part of the potential which is the potential due to the nearby particles, can be evaluated easily since every processor has the information on all the particles. A reduction operation is required at the end to sum up the partial potentials computed by each processor for every particle.

Author's personal copy

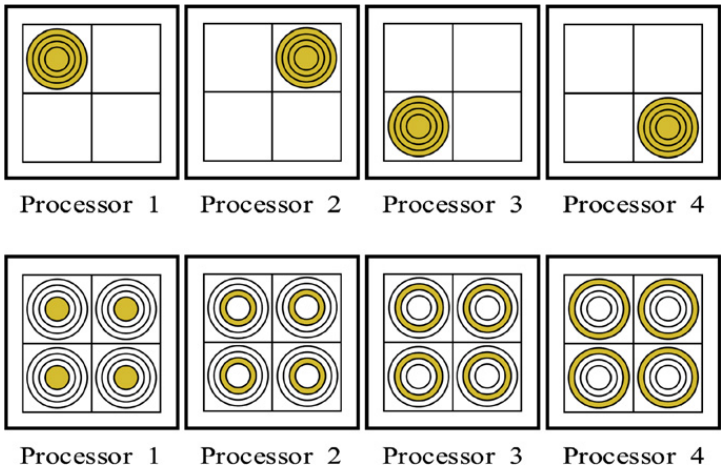


Fig. 1. The nested circles represent the (χ, ϕ) space. χ is similar to a radial coordinate and ϕ to an angular one. The 2×2 boxes represent the entire domain divided into 4 clusters. *Top row*: each processor has the coefficients associated to all pairs of (χ, ϕ) for a subset of the clusters. In the figure, each processor owns a single cluster. This is the usual domain decomposition approach. *Bottom row*: every processor owns a subset of the coefficients (χ, ϕ) but for all the clusters. This is the new approach advocated in this paper for PWA. It reduces the amount of communication in a significant fashion.

To summarize, a schematic parallel algorithm for PWA is as follows.

(1) *Initialization*: Each processor computes its subset of the plane wave coefficients at the finest level of clusters.
(2) *Gather*: Each processor computes its plane wave coefficients at all levels by going up through the hierarchical tree structure of clusters.
(3) *Transfer*: Every processor converts the assigned plane wave coefficients to the corresponding local coefficients at each level.
(4) *Scatter*: Every processor computes the assigned local coefficients at the finest level by adding up the coefficients and going down the hierarchical tree.
(5) *Aggregation*: Each processor computes some of the terms in the plane wave expansion approximation to the potential for all particles.
(6) *Direct*: Each processor computes the potential due to nearby particles for a subset of particles.
(7) *Communication*: A reduction operation is performed over all the particles to sum up the total potential.

There is only a single communication step at the end. The total number of particles is usually much smaller than the total number of multipole coefficients, and the final reduction at the end can be performed efficiently.

4.2. Parallelizing SPME

The potential computation in the SPME method is done in the real space and the reciprocal space. To minimize the number of communications, all the processors are assumed to hold information for all the particles. This is the same choice as

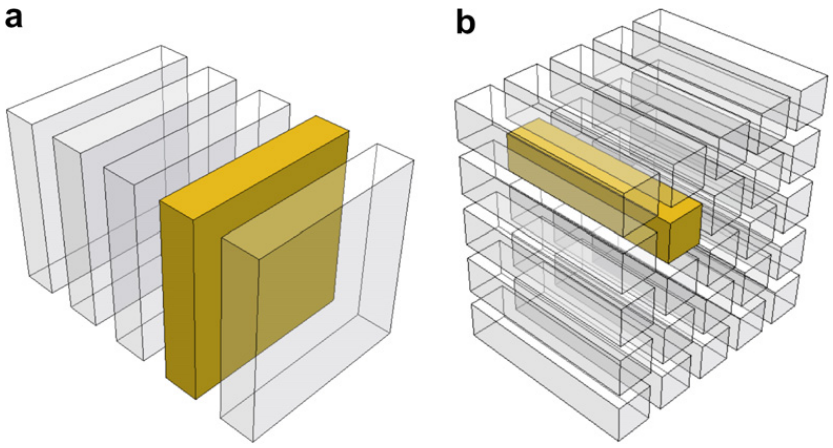


Fig. 2. Each processor receives portion of the grid for the Fourier transforms.

Author's personal copy

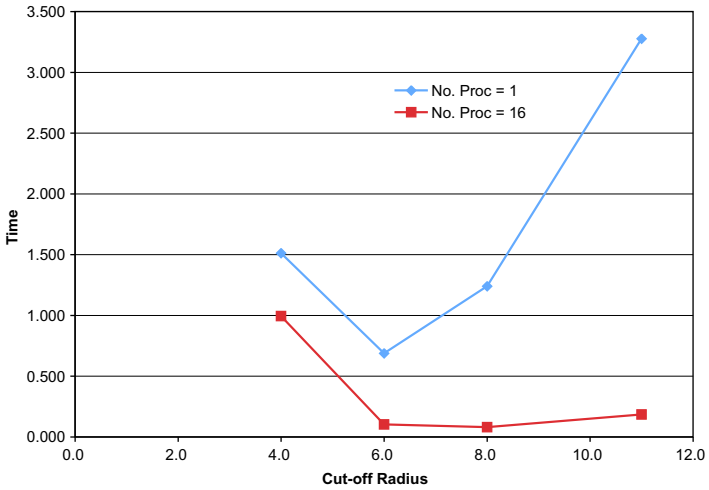


Fig. 3. SPME method, total runtime for single processor vs. 16 processors, cluster A, 10 K problem size.

Table 1
The number of FFT grid points used for different cut-off radii in the two test cases

	10 K	100 K
$r_c = 6.0$	–	$256 \times 256 \times 256$
$r_c = 8.0$	$64 \times 64 \times 64$	$144 \times 144 \times 144$
$r_c = 11.0$	$64 \times 64 \times 64$	$96 \times 96 \times 96$
$r_c = 14.0$	$32 \times 32 \times 32$	–

with PWA. This was found to be preferable as it avoids two extra communication steps during the charge interpolation and potential back-interpolation. In addition, this makes the parallel computation of the real space part easy. After creating the cell lists and Verlet lists, a subset of the particles (or in other words, a subset of the cell lists) is assigned to every processor, which then computes U_{real} (or the corresponding force) using Eq. (21).

However, the evaluation of the potential in the reciprocal space requires FFTs. To perform the FFT in 3 dimensions, we perform a 2D FFT in *yz* planes followed by a one dimensional FFT in the *x* direction. To do that the domain is split into *yz* slabs and every slab is assigned to a processor as is shown in Fig. 2a. These slabs may contain one or more *yz* planes. Then, every processor performs a 2D FFT on the *yz* planes which it has data for.

Afterward, by communicating among all the processors, the data is transposed so that every processor receives grid data for a number of *x* axes as shown in Fig. 2b. Now, the processors perform 1D FFTs in the *x*-direction. Thereafter, an embarrassingly parallel kernel computation is performed in reciprocal space by each processor. The inverse transforms are done in a similar way resulting in another communication step among all the processors.

Here is a schematic parallel algorithm used for SPME, where the communication steps are highlighted in bold:

(1) *Charge interpolation.*
(2) *Forward 2D FFT.*
(3) *Communication*: a block matrix transpose is performed with the FFT arrays.
(4) *1D FFT and kernel computation in the reciprocal space.*
(5) *Communication*: a block matrix transpose is performed with the FFT arrays.
(6) *Backward 2D FFT.*
(7) *Real space.*
(8) *Communication*: A reduction operation is performed over all the particles to sum up the total potential.

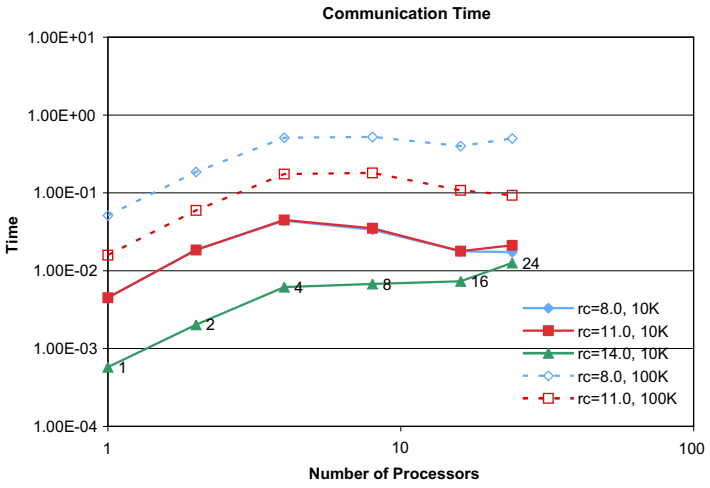
This algorithm requires three communication steps involving all processors while the PWA algorithm requires just one. Furthermore, note that in this algorithm, the size of the data transferred during the communication steps not only depends on the number of particles but it also depends on the number of grid points. In addition, if the number of available processors is larger than the number of grid points in the *x*-direction, either additional communication steps are required or not all processors can be used.

5. Numerical results

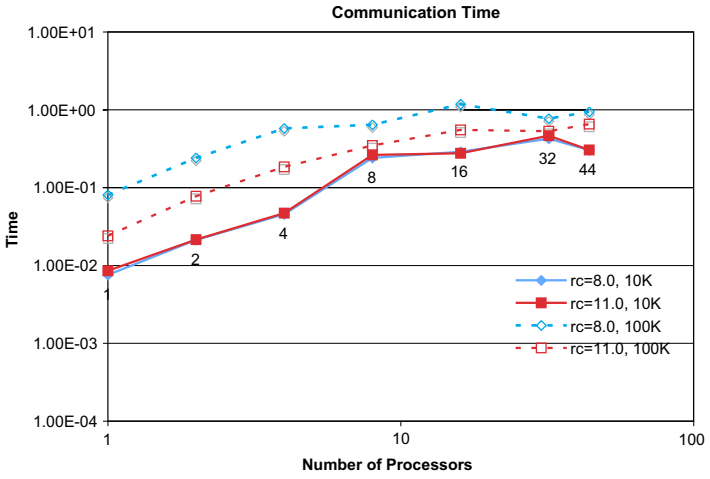
The timing benchmarks of the two algorithms are presented in this section. First, we will explore the effect of various parameters on each method. As mentioned earlier, minimizing the total runtime is the ultimate goal. First we study each individual method and then perform a comparison.

Author's personal copy

Three different machines were used for this purpose. Cluster A is a small size machine with 24 processors on 12 nodes. Cluster B is a medium size machine with 92 processors on 46 nodes. Cluster C is a large machine with 2208 cores on 552 nodes. All machines have an Ethernet interconnection.



(a) Cluster A



(b) Cluster B

Fig. 4. SPME Communication time.

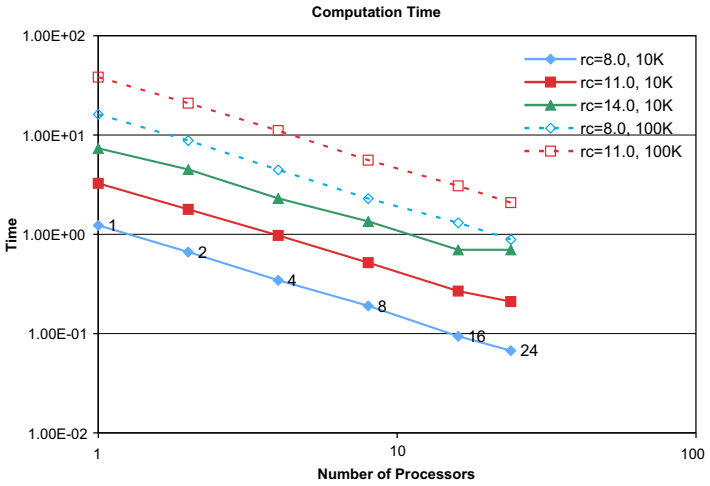


Fig. 5. SPME Computation time on cluster A.

Author's personal copy

Two problem sizes were chosen for the time measurements. First one is a $49 \times 49 \times 49$ box with 10,488 atoms. The second one is a $116 \times 116 \times 116$ box with 128,832 atoms. For the sake of simplicity we refer to them as the 10 K and 100 K problems, respectively.

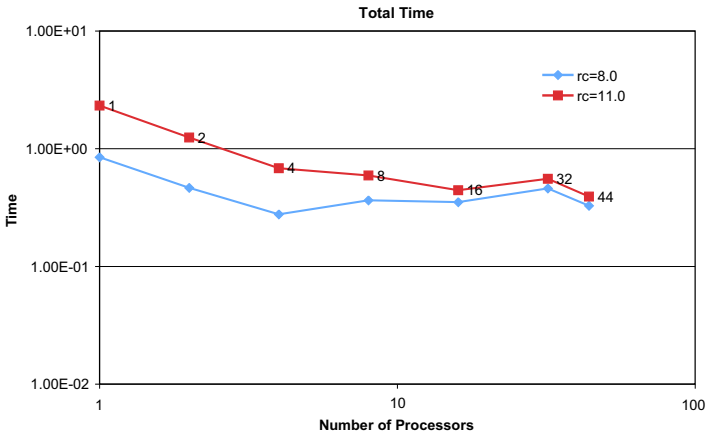
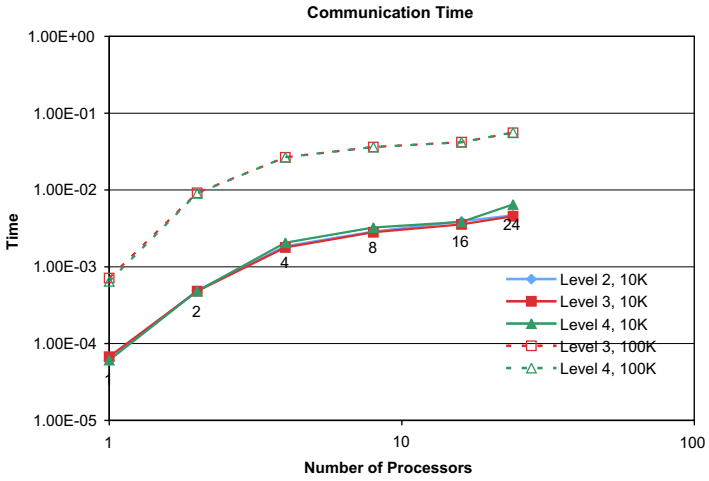
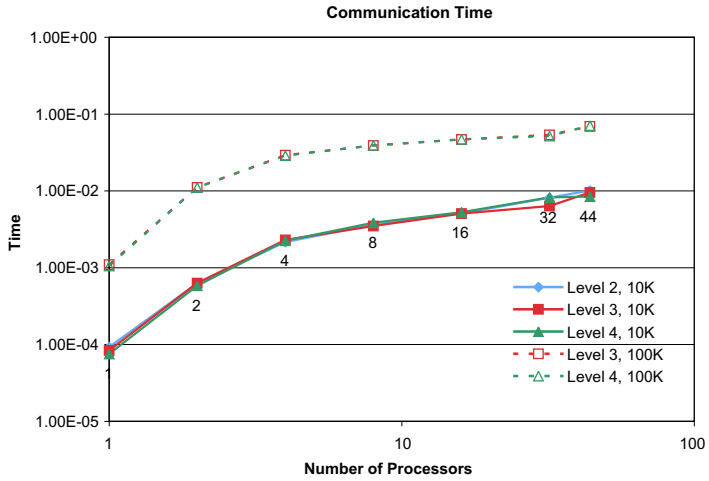


Fig. 6. SPME Total runtime on cluster B, problem size: 10 K.



(a) Cluster A



(b) Cluster B

Fig. 7. PWA Communication time.

Author's personal copy

Both methods have parameters that directly influence the computation time, the communication time and the accuracy. In all the cases, the parameters are set to reach an error below 10^{-6}.

5.1. SPME – timing benchmarks

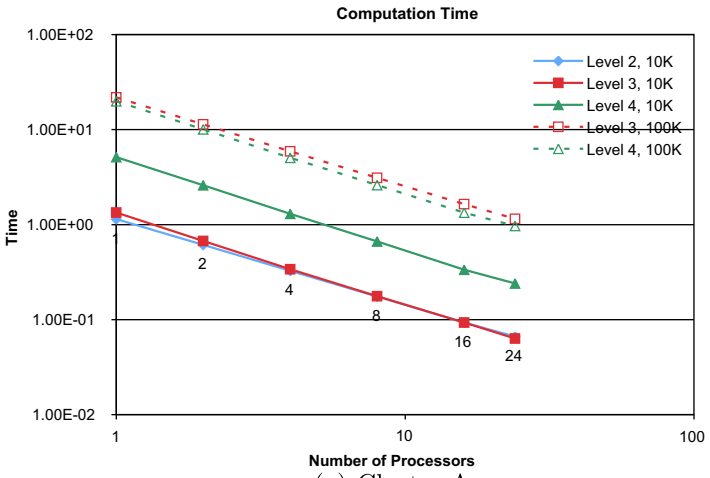
The parameters that can be changed in the SPME method are the cut-off radius which directly specifies the Ewald coefficient, the number of grid points for the FFT and the interpolation order. Changing the Ewald coefficient which is done here through the alteration of the cut-off radius determines how the computation time is split between the real space and the reciprocal space. As we decrease the cut-off radius, the computation done in real space decreases and we need more grid points in the reciprocal space to maintain the order of accuracy. Increasing the number of grid points for the FFT increases the computation done in the reciprocal space as well as the communication time.

The interpolation order is another parameter which controls the accuracy. It also has direct effect on the runtime. In the following time measurements, the interpolation order was fixed at 6. This number was found to be nearly optimal.

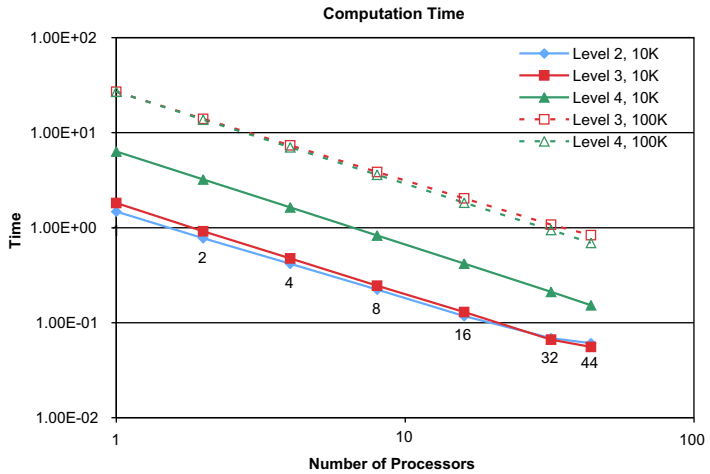
We have to choose a cut-off radius which results in the minimum total runtime. Frenkel and Smith provided an analysis of the Ewald coefficient and its effect on the total runtime in [1]. However, when we are using a computer cluster the communication time adds up to the total runtime. Hence, the cut-off radius (or the Ewald coefficient) which results in the minimum runtime for a single processor may not be the optimal one for a cluster. In fact, a larger cut-off radius must be used to get shorter total runtime.

As Fig. 3 reveals, although the minimum runtime with a single processor occurs using an approximate cut-off radius of 6.0, we have to use a larger cut-off radius of 8.0 to get shorter runtime for 16 processors. The reason to that will be more obvious when we present the detailed time measurements for this method.

The cut-off radii and the number of grid points used to meet the desired accuracy of 10^{-6} for two test cases are shown in Table 1.



(a) Cluster A



(b) Cluster B

Fig. 8. PWA Computation time.

Author's personal copy

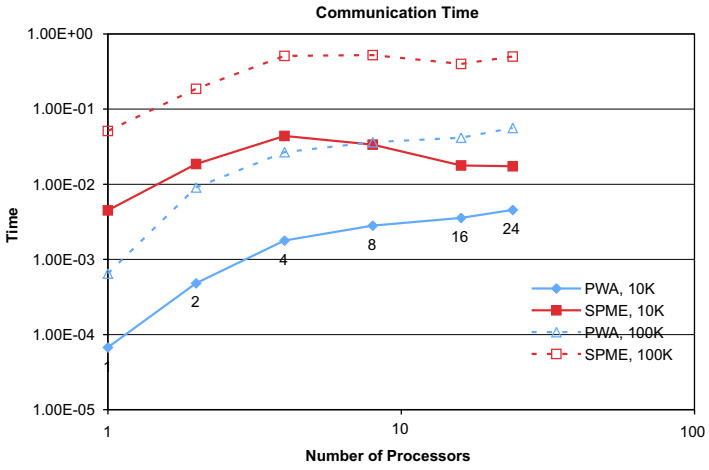
The reduction in the last step of the algorithm scales linearly with the number of atoms. In terms of the number of processors, the wall clock time scales as $\ln(P)$ where P is the number of processors. The communication time also depends on the grid size for the FFT because of the two transpose operations. The same scaling holds for this transpose operation.

Fig. 4a shows almost the expected behavior. For the case of $r_c = 14.0$ which is the largest cut-off radius in the figure, a smaller grid size is required. As a result, the communication time is shorter. It should be noted that when increasing the number of processors, we are also transmitting smaller data sets (e.g. when transposing the FTT grid). This explains why the communication time goes down at some point. Fig. 4b shows the communication in the cluster B which has a poor interconnection between the nodes. Hence, the data size which is being transmitted has less effect in this cluster (larger relative latency).

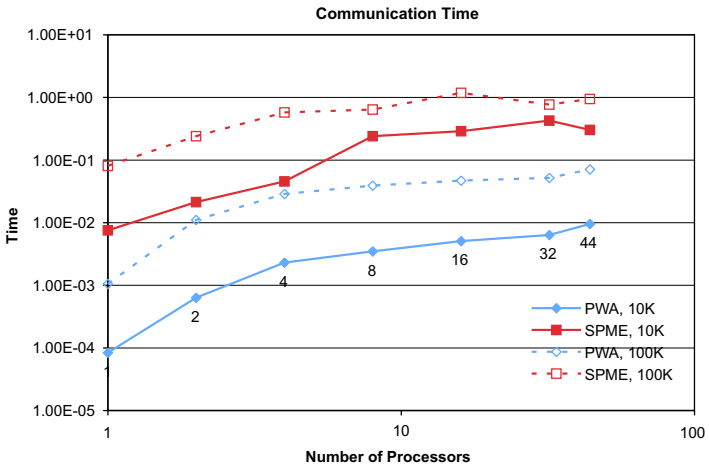
Fig. 5 shows the SPME computation time. The computation time scales very well in most cases. Although, in the case of the smaller problem size, 10 K, it cannot scale properly for the large number of processors when a large cut-off radius is used. A large cut-off radius means more computation in the real space and a small grid size for the FFT. Since the number of atoms are small here, increasing the number of processors beyond a certain point has no advantage. The computation time in the cluster B follows the same behavior.

If we compare Figs. 4a and 5, we find that for the large number of processors the communication time and the computation time are of the same order. Therefore the communication time has a significant influence on the total time. For both clusters and problems sizes, $r_c = 8.0$ gives the smallest total runtime. The low performance network in cluster B has a dramatic effect as seen in Fig. 6. The smallest runtime is achieved by 4 processors. Beyond that, we don't get any improvement in the total runtime.

In terms of the scalability, cluster A shows acceptable scalability but cluster B suffers from poor scalability. Using a larger cut-off radius leads to improve scalability (Fig. 6) since it allows reducing the size of the FFT grid. However, the increase in computational cost offsets this improvement resulting in overall longer total runtime.



(a) Cluster A



(b) Cluster B

Fig. 9. Communication time, PWA vs. SPME.

Author's personal copy

5.2. PWA – timing benchmarks

The number of quadrature points χ and ϕ and the number of levels are the parameters which can be changed in the PWA algorithm. By increasing the number of levels, we make the leaf clusters smaller. Hence, the computational cost due to the direct potential evaluation of neighboring cells decreases. At the same time, the number of quadrature points will increase which will add to the computational cost of the plane wave approximation. We have a balance between the computational time due to direct potential evaluation and the plane wave expansion. The number of levels is chosen to minimize the runtime.

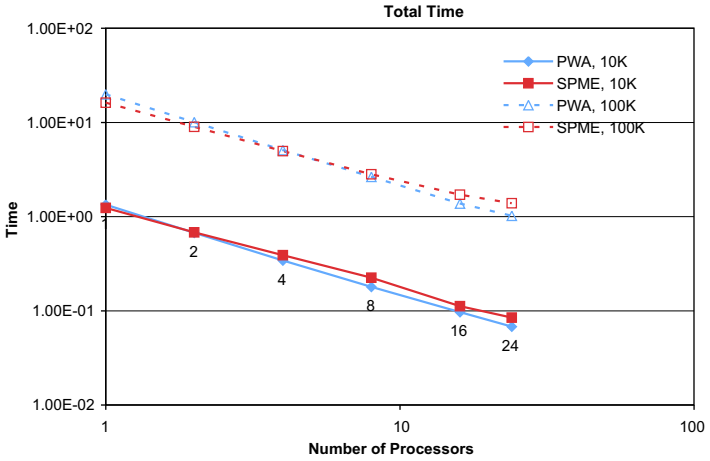
The number of quadrature points χ used for different number of levels 2,3 and 4 are 14, 18 and 22, respectively. The number of discretization points for ϕ remains 20 in all the cases.

The PWA algorithm has just one communication step at the end of the calculation. This reduction operation depends only on the number of atoms and does not depend on the number of levels.

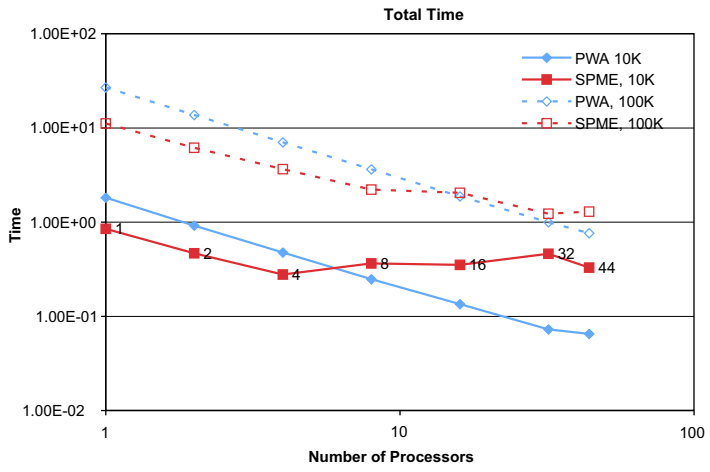
Fig. 7 shows the communication time in the PWA algorithm. The communication time depends linearly on the number of atoms and increases with the number of processors. Although cluster B has a poor interconnection, the communication time for cluster A and cluster B are almost the same.

The computation time are shown in Fig. 8. The computation time scales perfectly. As we observed in Fig. 5, in the SPME we could not scale the computation time for 10 K problem. This is not an issue here anymore. In the case of the 10 K problem, using 2 and 3 levels lead to a similar computation time. The same thing happens for the 100 K problem, with 3 levels and 4 levels. This is exactly where we are having a balance between the computation due to near neighbors and the plane wave approximation.

The total runtime was found to decrease as we increase the number of processors (see Fig. 10 in the next section). We chose 3 levels for the 10 K problem and 4 levels for the 100 K problem as the optimal number of levels.



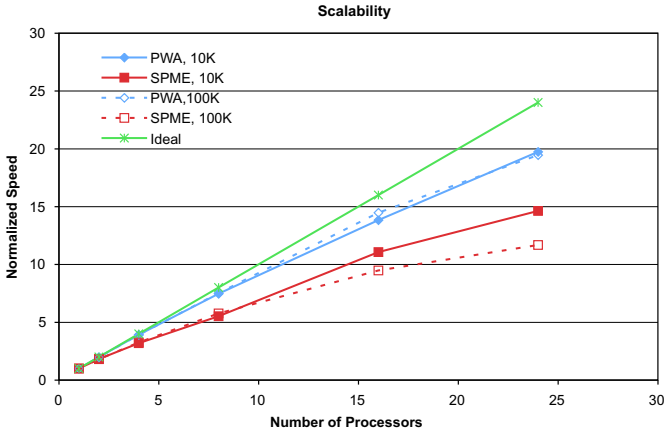
(a) Cluster A



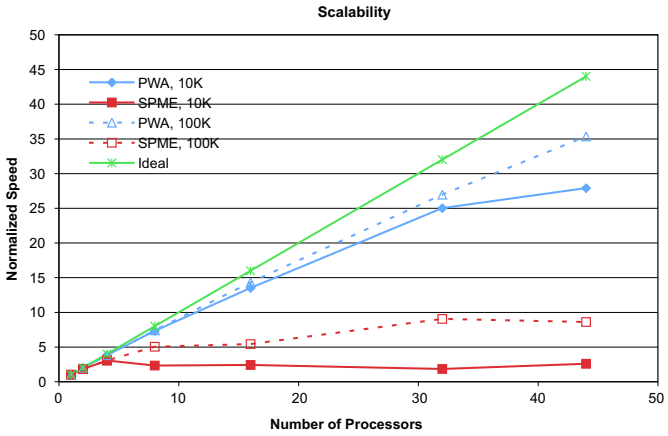
(b) Cluster B

Fig. 10. Total runtime, PWA vs. SPME.

Author's personal copy



(a) Cluster A



(b) Cluster B

Fig. 11. Scalability, PWA vs. SPME.

5.3. Comparing the PWA and SPME algorithms

Fig. 9 shows the communication time in the two clusters for the two problem sizes. PWA shows a better benchmark timing for the communication. Particularly for cluster B, which has not a good interconnecting network, the communication time is an order of magnitude less than that of SPME. Furthermore, comparing the two clusters reveals that the communication time in PWA tends to be more hardware independent.

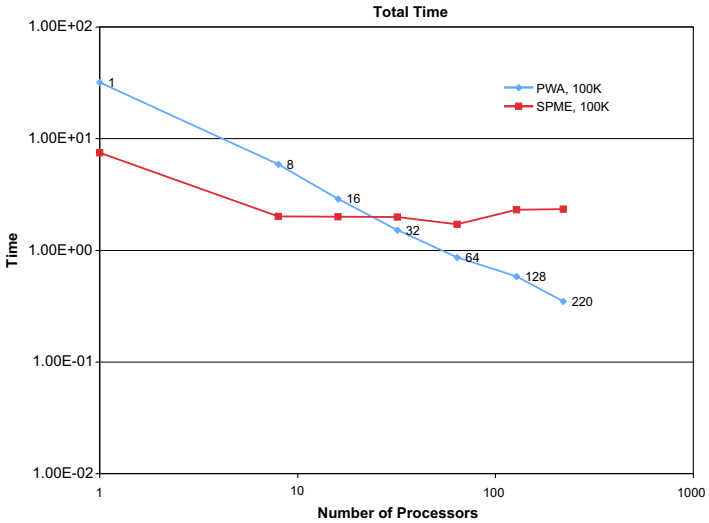


Fig. 12. Total runtime on cluster C, PWA vs. SPME.

Author's personal copy

Fig. 10 shows the total runtime for the algorithms. For a small number of processors P, although the communication time is smaller for PWA, the longer computation time keeps the total runtime somewhat close to that of SPME. However, the shorter communication time pays off when adding processors. In particular, for cluster B, PWA is faster at 10 K for $P \geqslant 8$ and at 100 K for $P \geqslant 32$.

Moreover, in the case of cluster B which has a low performance interconnection, PWA keeps scaling with the number of processors while the SPME method starts showing unpredictable behavior. This is due to the communication time which makes a huge difference for small problem sizes and/or large number of processors.

Fig. 11 shows the scalability of the two methods. Although scalability in itself is not such a good measure to compare the algorithms, it is instructive. In all cases, PWA shows better scalability which is due to its reduced communication. The difference gets even more noticeable when the communication is not very efficient, as shown in Fig. 11b.

In the case of larger clusters, SPME requires even more communication time and the communication becomes a dominant portion of the total runtime. Fig. 12 shows the total runtime for the two methods on cluster C utilizing up to 220 processors. Although SPME reaches the shortest runtime at 64 processors, it almost does not have any major improvement beyond 8 processors. PWA shows a consistent improvement in the total runtime as more processors are employed.

6. Conclusion

Techniques to parallelize SPME and PWA were presented in this paper. SPME requires relatively few floating point operations because it relies on very efficient fast Fourier transforms. However on parallel computers, the situation deteriorates significantly because a large amount of communication is required (transpose operations for the FFTs and final reduction). However trade-offs are possible where the number of floating point operations can be increased while communication is reduced. An example was shown where the cut-off radius is increased to reduce the size of the grid in Fourier space. We observed that on clusters with slow interconnect, the performance of SPME behaves erratically when the number of processors is increased beyond a certain threshold (4 processors for the 10 K problem). It is important to realize that a similar observation would be made on clusters where the processors are very fast, e.g. for multi-core processors, graphics cards or the Cell processor.

PWA is based on a recent formulation of the fast multipole method. Because it uses an expansion in terms of plane wave functions, it is possible to parallelize the algorithm differently, that is we can decompose the plane wave coefficients instead of the clusters as is usually done. This approach was explored in this paper. This leads to a single communication step at the end with a reduction operation. The scalability was found to be excellent including on clusters with slow networks (or fast processors). The number of floating point operations is somewhat larger than for SPME but the reduction in communication ultimately pays off when communication is the bottleneck. We observed that the behavior of the algorithm is more predictable than SPME and that performance is less dependent on the hardware being used.

We note that it is possible to reduce the number of operations in PWA in particular using some of the techniques developed by Demaine et al. [33]. This would improve our approach even further.

Acknowledgment

This work was supported in part by NSF award CNS-0619926 for computer resources.

References

[1] D. Frenkel, B. Smit, Understanding Molecular Simulation: From Algorithms to Applications, Academic Press, 1996.
[2] U. Essmann, L. Perera, M. Berkowits, T. Darden, H. Lee, L.G. Pedersen, A smooth particle mesh Ewald method, J. Chem. Phys. 103 (19) (1995) 8577.
[3] T.A. Darden, D.M. York, L.G. Pedersen, Particle mesh Ewald: An N log N method for Ewald sums in large systems, J. Chem. Phys. 98 (12) (1993) 10089.
[4] H.G. Petersen, Accuracy and efficiency of the particle mesh Ewald method, J. Chem. Phys. 103 (9) (1993) 3668.
[5] M. Deserno, C. Holm, How to mesh up Ewald sums (i): A theoretical and numerical comparison of various particle mesh routines, J. Chem. Phys. 109 (18) (1998) 7678.
[6] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comp. Phys. 73 (1987) 325.
[7] J. Barnes, P. Hut, A hierarchical $O(N \log N)$ force calculation algorithm, Nature 324 (1986) 446–449.
[8] J.K. Solomon, Parallel hierarchical N-body methods. PhD Thesis, California Institute of Technology, December 1990.
[9] J. Dubinski, A parallel tree code, New Astronomy 1 (1996) 133–147.
[10] R. Voldarnini, Parallelization of a treecode, New Astronomy 8 (2003) 691–710.
[11] S. Teng, Provable good partitioning and load balancing algorithms for parallel adaptive N-body simulation, SIAM J. Sci. Comput. 19 (2) (1998) 635–656.
[12] J.P. Singh, C. Holt, T. Totsuka, A. Gupta, J. Hennessy, Load balancing and data locality in adaptive hierarchical n-body methods: barnes-hut, fast multipole, and radiosity, J. Parallel Dist. Comp. 27 (2) (1995) 118–141.
[13] S. Velamparambil, W.C. Chew, Parallelization of MLFMA on Distributed Memory Computers, Proc. Int. Conf. Electromagn. Advanced Applications (ICEAA01) (2001) 141–144.
[14] S. Velamparambil, W.C. Chew, Analysis and performance of a distributed memory multilevel fast multipole algorithm, IEEE Trans. Antennas Propag. 53 (8) (2005) 2719–2727.
[15] L. Greengard, W.D. Gropp, A parallel version of the fast multipole method, Comput. Math. Appl. 20 (7) (1990) 63–71.
[16] J. Kurzak, B.M. Pettitt, Communications overlapping in fast multipole particle dynamics methods, J. Comput. Phys. 203 (2) (2005) 731–743.
[17] J. Kurzak, B.M. Pettitt, Massively parallel implementation of a fast multipole method for distributed memory machines, J. Parallel Distrib. Comput. 65 (7) (2007) 870–881.
[18] S. Ogatta, T.J. Campbell, R.K. Kalia, A. Nakano, P. Vashishta, S. Vemparala, Scalable and portable implementation of the fast multipole method on parallel computers, Comput. Phys. Comm. 153 (3) (2003) 445–461.

Author's personal copy

[19] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions, Acta Numer. 6 (1997) 229.
[20] L. Greengard, Fast algorithms for classical physics, Science 256 (1994) 909.
[21] C.A. White, M. Head-Gordon, Derivation and efficient implementation of the fast multipole method, J. Chem. Phys. 101 (8) (1994) 6593.
[22] V. Rokhlin, L. Greengard, J.F. Huang, S. Wandzura, Accelerating fast multipole method for the helmholtz equation at low frequencies, IEEE Comput. Sci. Eng. 5 (3) (1998) 32–38.
[23] L. Greengard, J.F. Huang, A new version of the fast multipole method for screened coulomp interactions in three dimensions, J. Comp. Phys. 180 (2002) 642–658.
[24] E. Darve, The fast multipole method (i): error analysis and asymtotic complexity, SIAM Numer. Anal. 38 (2000).
[25] E. Darve, The fast multipole method: numerical implementation, J. Comput. Phys. 160 (1) (2000) 195–240.
[26] E. Darve, P. Havé, Efficient fast multipole method for low-frequency scattering, J. Comput. Phys. 197 (2004) 341–363.
[27] C.C. Lu, W.C. Chew, A multilevel algorithm for solving boundary-value scattering, Microw. Opt. Technol. Lett. 7 (10) (1994) 466–470.
[28] J. Song, C.C. Lu, W.C. Chew, Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects, IEEE Trans. Antennas Propag. 45 (10) (1997) 1488–1493.
[29] W.C. Chew, J. Jin, C.C. Lu, E. Michielssen, J. Song, Fast solution methods in electromagnetics, IEEE Trans. Antenna Propag. 45 (3) (1997) 533–543.
[30] H. Cheng, L. Greengard, V. Rokhlin, A fast adaptive multipole algorithm in three dimensions, J. Comput. Phys. 155 (1999) 468–498.
[31] N. Yarvin, V. Rokhlin, Generalized Gaussian quadratures and singular value decompositions of integral operators, SIAM J. Sci. Comput. 20 (2) (1998) 699–718.
[32] R.W. Hockney, J.W. Eastwood, Computer Simulations Using Particles, McGraw-Hill, 1981.
[33] E.D. Demaine, M.L. Demaine, A. Edelman, C.E. Leiserson, P.-O. Persson, Building blocks and excluded sums, SIAM News 38 (1) (2005) 1–5.