SAND2016-1051C

Parallel Accelerated Cartesian Expansions for Particle Dynamics Simulations

M. Vikram*, A. Baczewzki*, B. Shanker* and S. Aluru†

*Dept. of Electrical and Computer Engineering
Michigan State University
East Lansing, MI 48824, USA
Email: vikramr@egr.msu.edu, baczewsk@egr.msu.edu, bshanker@egr.msu.edu

†Dept. of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011, USA
Email: aluru@iastate.edu

Abstract

Rapid evaluation of potentials in large physical systems plays a crucial role in several fields and has been an intensely studied topic on parallel computers. Computational methods and associated parallel algorithms tend to vary depending on the potential being computed. Real applications often involve multiple potentials, leading to increased complexity and the need to strike a balance between competing data distribution strategies, ultimately resulting in low parallel efficiencies. In this paper, we present a parallel accelerated Cartesian expansion (PACE) method that enables rapid evaluation of multiple forms of potentials using a common Fast Multipole Method (FMM) type framework. In addition, our framework localizes potential dependent computations to one particular operator, allowing reuse of much of the computation across different potentials. We present an implicitly load balanced and communication efficient parallel algorithm and show that it can integrate multiple potentials, multiple time steps and address dynamically evolving physical systems. We demonstrate the applicability of the method by solving particle dynamics simulations using both long-range and Lennard-Jones potentials with parallel efficiencies of 97% on 512 to 1024 processors.

1. Introduction

Considerable research in high performance computing has been devoted to simulation of complex physical phenomena, modeled using different potential and force fields. Numerical methods for computing potentials and computational strategies for their parallel execution have been an active area of research. For example, potentials of the form $\frac{1}{R}$ can be efficiently computed using the Fast Multipole Method (FMM) [1, 2]. Rapidly decaying short-range potentials are often approximated by considering a sphere around each point where potential needs to be evaluated, and taking into account only interactions due to entities within the sphere. Even when a computational method is applicable to more than one potential, the underlying numerical methods require a completely different set of equations, requiring that each potential be computed separately.

Many real-world applications involve physical phenomena where multiple potentials are simultaneously involved. When these potential computations call for different parallel algorithms, each with its own set of data structures and data distribution strategies to optimize parallel run-time, the application developer is forced to strike a balance between these competing choices. Should one use an optimal distribution strategy for one method, at the expense of inferior performance on another? Should multiple data structures be maintained, each optimal for a particular computational method? If so, how should boundary conditions be imposed across these multiple evaluations? Will the communication expense in translating between multiple data structures and domain decomposition strategies derail the gains accrued by faithfully implementing optimal strategies targeted to each specific potential? While these issues are all performance related, a similar host of issues occur in code development and software complexity. Clearly, a single computational method that can compute a diverse set of potentials is

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

of great benefit from both performance and application development perspectives.

In this paper, we present an efficient parallel framework that can compute many classes of potentials using the same computational method. Our framework is based on the recently invented accelerated Cartesian expansions (ACE) algorithm [3], which among other things, can compute any potential of the form $R^{-\nu}$. ACE is a hierarchical computational method similar to the Fast Multipole Method (FMM) that reduces the cost of computation from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. It utilizes totally symmetric Cartesian harmonics for optimal representation of multipole and local expansion coefficients. Consequently, all operators and definitions except the translation operator are independent of the form of the kernel or potential function. This fact enables one to consider multiple types of potentials without much change or modification to the algorithm.

As our framework is based on the FMM, we briefly review literature pertinent to this topic. The FMM is a highly efficient computational method, but hitherto limited only to computing potentials of the form $\frac{1}{R}$. Since its introduction by Rokhlin and Greengard two decades ago [1, 2], considerable research efforts are devoted to its efficient execution on parallel computers by several researchers, including us (for example, see [4–10]). Parallel FMM based algorithms have been applied in a number of application areas including molecular dynamics [11], capacitance extraction [12] and a variety of electromagnetic simulations [13, 14]. The FMM has been recognized as one of the top 10 algorithmic contributions in the last century and is acknowledged to be a highly efficient computational method for large systems.

Our focus in this work is to significantly extend the reach of FMM by developing an FMM-like framework using the ACE methodology for simultaneous evaluation of multiple potentials using the same computational method and parallelization scheme. We make the following specific contributions:

- present a parallel method for the fast hierarchical computation of multiple potentials using ACE
- provide support for multiple potentials, multiple time scale integrators, and dynamically evolving particle systems
- develop full-scale application software and demonstrate its scalability and high parallel efficiency

We demonstrate the applicability of the proposed method by developing a parallel particle dynamics simulation incorporating both long range electrostatic and short-range Lennard-Jones potentials. While this is a commonly studied problem, some of the existing parallel tree codes yield parallel efficiencies as low as 44% on 256 processors [15]. Our experimental results show that the proposed algorithm yields efficiencies as high as 98% up to 512 processors, and continues to provide high parallel efficiencies beyond that up to the size of systems available to us for testing.

The rest of the paper is organized as follows: Section 2 contains a brief description of the ACE method. In Section 3, we present our parallel ACE framework that is used to compute multiple potentials simultaneously. Section 4 contains a brief overview of particle dynamic simulations. In Section 5, we present an implementation of the particle dynamics simulation using our framework. Experimental results are presented in Section 6. Section 7 concludes the paper.

2. Accelerated Cartesian Expansion (ACE) Method

The mathematical engine behind the proposed parallel framework is the theory of Accelerated Cartesian Expansions (ACE). This was initially developed for kernels of the form $R^{-\nu}$ [3] and has been extended to frequency domain sub-wavelength kernels [16], retarded potentials [17], Yukawa (or shielded Coulomb) potentials [16] and diffusion potentials (Gauss transform), and dispersion and Klien-Gordon kernels [18]. While similar methodologies have been introduced earlier [19], they are either not generalizable or offer only some of the advantages of this scheme. In what follows, we provide a brief overview of this numerical method.

Tensors are an integral part of the ACE algorithm. In the rest of the paper, $\mathbf{M}^{(n)}$ denotes a tensor of rank n, the *polyadic* associated with $\mathbf{r} = \{r_x, r_y, r_z\}$ is given by $\mathbf{r}^{(n)} = \{r_x^{n_1} r_y^{n_2} r_z^{n_3}\}$ where $n = \sum_{i=1}^{3} n_i$ and $n_i > 0$, an m fold contraction between two tensors $A^{(n)}$ and $B^{(m)}$ is denoted by $A^{(n)} \cdot m \cdot B^{(m)} = C^{(n-m)}$ when $n > m$; for more details on these definitions and operations see [3]. ACE is a hierarchical tree based computational method in the vein of FMM in that it uses octree for geometry processing and derives *equivalent* operators for tree computation. In contrast to FMM, ACE employs Cartesian harmonics for multipole and local expansions. Consider a source and observation domain denoted by Ω_s and Ω_o, respectively. Without loss of generality, these domains are assumed to be spherical with radius a. The following three theorems form the crux of ACE and enable the computation of fields in Ω_o due to sources in Ω_s.

Theorem 2.1 (Multipole Expansion) *The total potential at any point* $\mathbf{r} \in \Omega_o$ *due to* k *sources* q_i, $i = 1, \cdots, k$ *located at points* $\mathbf{r}_i \in \Omega_s$ *is given as*

$$\begin{aligned} \psi(\mathbf{r}) &= \sum_{n=0}^{\infty} \mathbf{M}^{(n)} \cdot n \cdot \nabla^n \psi(\mathbf{r}) \\ \mathbf{M}^{(n)} &= \sum_{i=1}^{k} (-1)^n \frac{q_i}{n!} (\mathbf{r}_i - \mathbf{r}_s)^n \end{aligned} \tag{1}$$

where $\mathbf{M}^{(n)}$ is the multipole tensor.

Theorem 2.2 (Multipole to Local Translation) *Assume that the domains* Ω_s^p *and* Ω_o^p *are sufficiently separated, and the distance between their centers* $r_{os}^p = |\mathbf{r}_{os}^p| = |\mathbf{r}_o^p - \mathbf{r}_s^p|$ *is greater than* $diam\,\{\Omega_s^p\}$ *and* $diam\,\{\Omega_o^p\}$. *If a multipole expansion* $\mathbf{M}^{(n)}$ *is located at* $\mathbf{r}_s^p$, *then another expansion* $\mathbf{L}^{(n)}$ *that produces the same field* $\forall \mathbf{r} \in \Omega_o^p$ *is given by*

$$\begin{aligned} \psi(\mathbf{r}) &= \sum_{n=0}^{\infty} \boldsymbol{\rho}^n \,.\, n \,.\, \mathbf{L}^{(n)} \\ \mathbf{L}^{(n)} &= \sum_{m=n}^{\infty} \frac{1}{n!} \mathbf{M}^{(m-n)} \,.\, (m-n) \,.\, \widetilde{\nabla}^m \psi(r_{os}^p) \end{aligned} \tag{2}$$

where $\boldsymbol{\rho} = \mathbf{r} - \mathbf{r}_o^p$.

Theorem 2.3 (Local to observer) *Given a local expansion* $\mathbf{L}^{(m)}$ *that exists in the domain* Ω_o *centered around* $\mathbf{r}_o$, *the potential at any point* $\mathbf{r} \in \Omega_o$ *is given as*

$$\psi(\mathbf{r}) = \sum_{m=0}^{\infty} (\mathbf{r} - \mathbf{r}_o^c)^m \,.\, m \,.\, \mathbf{L}^{(m)} \tag{3}$$

In the above theorems, it is important to note that all definitions except for the translation operator are independent of the form of the kernel. Thus ACE is an *almost kernel independent* algorithm and since these expansions are based on Taylor series, they are rapidly converging for any function that is *non-oscillatory*. Totally symmetric tensors provide an optimal representation of Cartesian harmonics and this in turn leads to considerable savings in terms of memory and complexity. The multipole-to-multipole and local-to-local translation operators in the multilevel version of ACE are exact [3]. This implies that the error incurred in ACE algorithm does not increase with the resolution of the domain decomposition, or equivalently the height of the tree. Due to paucity of space, readers are referred to [3] for details of this algorithm, proofs and salient features.

3. Parallel Accelerated Cartesian Expansion (PACE) Framework

3.1. Hierarchical Computation of Multiple Potentials

We first provide a basic overview of how the computational method permits rapid potential evaluation using a hierarchical decomposition of the domain, as given by the octree data structure. The octree is constructed by first embedding the entire domain in a fictitious cube that is then divided into eight subcubes, and so on. This process is recursively applied to non-empty subcubes until the desired level of refinement is reached; an N_l-level scheme implies $N_l - 1$ recursive divisions of the domain. At any level, the domain that is being partitioned is called the *parent* of all the eight *children* that it is being partitioned into. At the lowest level, all source/observers are mapped onto the smallest boxes, *leaf boxes*. This hierarchical partitioning of the domain is referred to as a regular octree data structure.

The interactions between all source and observation points are now computed using traversal up and down the tree structure. This is done using the following rule: at any level in the tree, all boxes are classified as being either in near field or far field of each other using the following dictum: two boxes at the same level are said to be in nearfield of each other if they are adjacent to each other or touch diagonally; and two boxes at the same level are said to be in the far field of each other if the distance between their centers is at least twice the sidelength of the boxes. For every box b in the tree, an *interaction list* is constructed that is made of all far field boxes at the same level such that their parents are in the near field of b's parent. Once the interaction lists have been built for all boxes, the computation proceeds as follows: At the lowest level, field between elements of boxes that are in the nearfield of each other is computed directly, i.e., using $\psi(R)$.

All other interactions are computed using a three stage algorithm:

1) Compute multipole expansions of clusters of sources that reside in each box. At the leaf level, these are computed directly from sources that reside within the leaf box. At higher levels, the multipole expansion of a box is computed from the expansions at its children.
2) For each box, compute the local expansion at the box due to boxes in its interaction list. This is done by considering the boxes in its interaction list, and using translation operators to convert

the multipole expansions at these boxes into local expansion. Membership in interaction lists is symmetric – i.e., if box b_2 occurs in b_1's interaction list, then b_1 occurs in b_2's interaction list as well. Hence, this stage can also be computed by translating the multipole expansion of a box to the local expansion of each box in its interaction list. This latter approach simplifies parallel communication by directly sending a multipole expansion rather than doing so in response to a request.

3) Compute the local expansion at each box due to all sources that are not in the nearfield. This is obtained by combining the local expansion at the parent, with the local expansion obtained by translation from multipole expansions of boxes in its interaction list, as described in previous step.

The local expansion so obtained at each leaf box now contains the effect due to all sources except those in its nearfield. This is combined with the direct nearfield computation carried out earlier, and the field at each observer is computed.

To evaluate multiple potentials, we allocate storage at the tree nodes corresponding to different potentials. It is also possible that the tree is traversed up to different heights depending on the spatial range of their influences. See section 5.2 for a more detailed exposition. We now proceed to describe the proposed parallel framework.

3.2. Parallel Construction of the Octree

The first step in the method is the parallel construction of the octree data structure reflecting hierarchical decomposition of the domain. Though this takes up a negligible fraction of the overall parallel run-time, it is important because (a) tree partitioning among the processors directly affects the load balancing of the rest of the algorithm, and (b) the various interaction lists are created at this stage and this is communication-intensive. In our implementation we store the tree in its postorder traversal order. It will be shown that this ordering of nodes enables load balanced computation of various tree operations, obviating the need for explicit load balancing.

Let N denote the total number of points (sources and observers) distributed within a cubical domain of side length dx and P be the number of processors. The average number of points per processor is denoted by $n = N/P$. Given the smallest side length dx_0 associated with leaf boxes, the total number of levels or height of the tree is $H = \log_2(dx/dx_0)$. To uniquely represent the tree nodes at all levels we employ the integer key coding scheme introduced in [6]. This representation has several advantages as (a) the keys encode a wealth of information such as the center position of the box represented by the node, level of the node, its entire ancestral lineage etc., and (b) the sorted keys conform to Morton ordering.

Given a computational domain we construct the tree in a bottom up fashion, starting from the leaf nodes. Each point in a processor is associated with a leaf node based on its position, and the leaf keys are sorted in parallel. The Morton ordering of the sorted leaf nodes distributed across processors results in a *self-similar* structure in each processor [14, 15]. This is crucial to parallel processing as *self-similarity* ensures same number of tree operations in each processor and hence this scheme is implicitly load balanced. The parent nodes are collected from their children and thus the full postorder tree is constructed in a recursive manner. In rest of the paper, given any two nodes A and B, we say A is *less than* B if node A appears earlier than node B in the postorder traversal ordering. This notion of comparison between tree nodes simplifies implementation of several of the processes detailed below.

Thus, in each processor we have a part of the tree with nodes at every level as shown in Figure 1. It is evident that some nodes can occur in multiple processors. When considering the global postorder traversal tree across processors, each such node has a processor where its occurrence is appropriate (the processor which has the rightmost leaf box in the subtree of the node). The node is considered *native* to this processor and the processor is referred to as the *native processor* to this node. All other occurrences of the node are termed *duplicate nodes*.

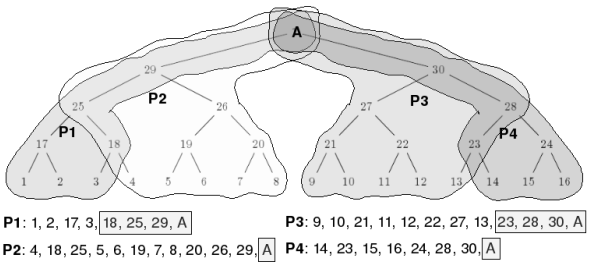


Figure 1. Partitioning of a tree among 4 processors. The postorder traversal tree at each processor is shown below and the *duplicate nodes* are highlighted.

Lemma 3.1 *The number of duplicate nodes in each processor is bounded by the height of the tree, and will appear sequentially at the end of the local postorder traversal tree.*

Proof: Let H denote the height of the tree. A processor can have at most one duplicate nodes per level in the tree. To see why, suppose a processor has at least two duplicate nodes at the same level in the tree. Let v_1 and v_2 be two such nodes, with v_2 occurring to the right of v_1 in the tree. A processor has a node in its local tree only if at least one of the leaf boxes in the subtree under the node falls in the same processor. Also, all the leaf boxes in a processor are consecutive in Morton ordering. Taken together, these two observations imply that the rightmost leaf box under v_1 must reside in the same processor. Thus, v_1 is native to this processor and cannot be a duplicated node. This argument demonstrates that a processor can have at most one duplicate node per level, shared with the next processor. Similarly, one can show that the number of multiply occurring nodes that are native to a processor are limited to one per level.

The second part follows from the fact that the postorder traversal order of the tree always places nodes before their parents. The parent of a duplicate node is also a duplicate node in the same processor. Hence all duplicate nodes in a processor appear in sequence at the end of the local postorder traversal tree. □

Note that even when the tree is full (i.e., all potential boxes at every level are present, i.e., are non-empty), the number of leaf boxes per processor need not be identical. This is because the points are distributed evenly to processors, and the number of points per leaf box need not be uniform.

3.3. Construction of Interaction Lists

As mentioned in previous section, ACE requires the construction of interaction and nearfield lists to perform the hierarchical computations. Interaction lists are built for all the nodes in the local tree except for the duplicate nodes. This operation is split into serial and parallel parts. In the serial part, we build the interaction list of each node assuming that the *full* tree is constructed, and later eliminate non-existent nodes using communication with processors that should have contained these nodes. Given a node's key code, straightforward bit manipulation yields its parent node, the parent's neighbor nodes and their children. These information are used to construct the interaction list of each local node. In the parallel part we construct different communication *maps* for information exchange during hierarchical computation. We create a list of local nodes to be sent to different processors based on the nodes in their interaction list and the processors they reside in. The local nodes and relevant parts of their interaction lists are exchanged among the processors. Since the interaction lists were compiled with the assumption of full tree, non-existent nodes among the received node lists are identified and removed. The entire process is efficiently implemented with the use of a binary tree search algorithm to identify nodes in postorder traversal. The nearfield list of each local leaf node is constructed and exchanged among processors in a similar fashion.

3.4. Multipole Expansion Computation

In each processor, the multipole expansions are computed at every node in the local postorder traversal tree as we scan from left to right. The postorder traversal order ensures that a parent node appears only after all its children (in case of duplicate nodes, all children that reside in the same processor's memory). Thus, when a parent node occurs the necessary children multipoles are already computed. Multipole expansion is computed for all the local nodes, including the *duplicate nodes*. However, the multipole expansions at the duplicate nodes are only partially filled as they account for sources in that processor only. After the local computation, all processors with duplicate nodes send their multipole expansions to the appropriate native processors of the duplicate nodes they host. The native processor of a node simply adds the received multipole expansion data to the appropriate local node. This algorithm is a one step update process with the following bound on communication overhead.

Lemma 3.2 *Total number of nodes received by a processor during multipole computation is bounded by* $(P-1)*H$.

Proof: This follows from the fact that the number of *duplicate nodes* in a processor is bounded by H (see Lemma 3.1). Since only the *duplicate nodes* are exchanged during multipole computations, the maximum number of nodes received by any processor will be no more than $(P-1)*H$. □

Cost analysis: Note that each processor has at least one node from every level of the tree and their multipole expansions are computed in every processor as we traverse the local postorder traversal tree. This part of the process is load balanced if every processor has the same number of leaf nodes. This is true even in the case of *dynamic* trees where the number of multipole

harmonics increases as the level increases. Since the number of duplicate nodes per processor is bounded, the communication overhead involved in exchange of their multipole information is also bounded. Hence the overall process is load balanced.

3.5. Translation Operation

At each node in the global postorder traversal tree we compute the local expansion using the multipole expansions of the nodes in its interaction list. This process is divided into a parallel and serial part. In the initial parallel part we exchange multipole information between the processors. while building the interaction lists, for each node in the local postorder traversal tree we identify the set of processors that require their multipole expansions. This list of local nodes and processors is sorted according to the processors. At every processor we traverse through this list and *pack* a character array with the multipole expansion data of the appropriate nodes using *MPI_PACK*. This array is then exchanged in blocks whose size is pre-determined so that communication time is optimum. Given the huge amount of data exchange among processors at this stage, packing the multipole data into blocks offers two advantages: (a) the number of communication calls is greatly reduced when compared to a scheme where the multipole data is exchanged one node at a time, and (b) the block size can be adjusted according to the communication architecture to ensure optimum performance. Once the required multipole expansion data is received the actual translation operation is performed in a serial manner. The local expansion of nodes in the local tree is computed using the received multipole expansions.

Cost analysis: Notice that the translation operation is reciprocal, thus if two interacting nodes are in different processors, then both processors need to exchange same amount of information. Hence the communication overhead associated with this process is load balanced and approximately same for all processors. If the tree is distributed uniformly across processors then the number of translation operations performed is same for all processors except for those that are the native processors of multiply occurring nodes. Note that the number of native processors is the same as the number of unique duplicate nodes. In case of static tree, where the number of multipole harmonics is same at all tree levels, this additional computation cost is not significant and would be masked by difference in distribution of tree among processors. However this difference will be noticeable, though may not be significant, in dynamic trees where the number of harmonics grows proportionally as we traverse up the tree. If desired this offset in load balance can be rectified by ensuring that the native processors have fewer leaf nodes to begin with.

3.6. Local Expansion Computation

In the downward tree traversal, the child node local expansions are updated with the local expansion at their parent node. This is exactly the reverse process of upward tree traversal. First, the processors with duplicate nodes obtain their local expansion from their native processors. Once the local expansion of duplicate nodes are filled, the downward tree traversal is performed locally in each processor as we traverse the local postorder tree from right to left.

Cost analysis: Since this is the *reverse* analogue of multipole expansion evaluation the same cost analysis applies here. Communication in this phase happens from native processors to processors holding corresponding duplicate nodes. As noted earlier, a processor is native to at most one multiply occurring node per level. Thus, the communication cost is identical as in the multipole expansion phase.

3.7. Evaluation of Potential

The farfield potential at the observation points are evaluated from the leaf node local expansion they reside in. However the evaluation of the potential is completed only after accounting for the nearfield interactions. These are interactions only among leaf boxes as specified by the nearfield list. Similar to translation operation, for each leaf node a list of processors that require its information is created and then sorted by processor number. At every processor we traverse through this list and communicate the leaf box information to appropriate processors. The information associated with each leaf box would include number of points, their positions, mass, charge and other attributes. When all the communication is completed, at every processor the potential at their local points is updated with nearfield contribution in a serial manner. This completes the evaluation of potential at every point across all processors using the ACE algorithm within the parallel framework.

4. Particle Dynamics Simulations

To demonstrate the power of our proposed framework, we developed and tested a sample particle dynamics (PD) application. In PD simulations, a variety of physical and chemical process are modeled by

considering atomic or molecular scale interactions. These simulations have been widely used in the study of protein folding, plasma dynamics, crystal dynamics etc. In PD simulations, at each simulation time step j, the force acting on a particle is first computed. Based on these forces the particle positions are updated, and these serve as the input for next time step $j+1$. Formally, consider a volume V filled with N particles whose charges q_i and other attributes like position and velocity $\{\mathbf{r}_i, \mathbf{v}_i\}$ are already known. Let N_t be the total number of time steps in the simulation with Δ_t increments. Then, at a particular simulation time step j, the force (potential) computation and position and velocity updates can be expressed in their simplest form as,

$$\begin{aligned} Step1: \quad & \Phi(\mathbf{r}_i)|_j = \sum_{k=1}^{K} \sum_{n=1, i\neq n}^{N} g_k(\mathbf{r}_i, \mathbf{r}_n) q_n q_i \\ & i \in [1, N] \\ Step2: \quad & \mathbf{r}_i|_{j+1} = \mathbf{r}_i|_j + \mathbf{v}_i|_j \Delta_t \\ & \mathbf{v}_i|_{j+1} = \mathbf{v}_i|_j - (\nabla\Phi(\mathbf{r}_i))\,|_j \Delta_t^2 / m_i \end{aligned} \tag{4}$$

where g_k is the kth type Green's function and each of the K Green's functions characterizes a different type of interaction. For example g_k can be used to compute the short range Lennard-Jonnes potential or long-range potentials like electrostatic and gravitational potentials,

$$g_k(\mathbf{r}_i, \mathbf{r}_n) \in \left\{ \left(\frac{A}{R^6} - \frac{B}{R^{12}} \right), \frac{1}{R} \right\} \tag{5}$$

where $R = |\mathbf{r}_i - \mathbf{r}_n|$ and A and B are constants in Lennard-Jonnes potentials based on particles species. Step 2 of equation (4) is referred to as *time integrators* and different schemes are available for accurate and stable implementation. The simplest form of simulation that we described here involves computing three different potentials $\frac{1}{R}$, $\frac{1}{R^6}$, and $\frac{1}{R^{12}}$. Often the size of time steps in PD simulation are very small, in orders of femto- to nano-seconds, to capture the dynamics of the system at sufficient resolution to maintain a stable evolution. Thus a large number of time steps is required to simulate any observable phenomena and this poses a computational bottleneck for large scale PD simulations.

5. Particle Dynamics Simulations Using PACE Algorithm

In this section, we present an implementation of the PD simulation described above using the PACE framework. In particle dynamics, particle attributes (position, velocity, etc.) evolve at each simulation time step. This poses an immediate complication to the use of multipole methods as they rely on pre-computation of various quantities. Thus one is required to efficiently perform these tasks and here we mention a variety of techniques to achieve this.

5.1. Construction of the Full Tree

As mentioned in section 3.1, by convention we build the tree based on the distribution of points. In other words, any node (box) exists in the tree only if the box is non-empty. In PD, sticking to such a notion implies that one is required to discard and reconstruct the tree every time the particles change position and move in and out of the leaf boxes. This also means that we need to update the interaction and nearfield lists every time the tree changes. Though these computations are of $\mathcal{O}(N)$ complexity, they can impose significant overhead when computed at each time step. Alternatively, here we propose to construct the *full* tree based on the maximum domain size, i.e., even empty leaf nodes will be accounted for along with their parents etc., while building the tree. This would mean a marginal increase in the pre-computation cost but eliminates all pre-computations at each time step. Further, it is common to expect the particles to move outside the bounding box and to accommodate this scenario we expand the initial bounding box prior to tree construction. We discard and rebuild the tree whenever the geometry changes considerably or when a particle moves out of the computational domain.

5.2. Incorporating Multiple Scale Time Integrators

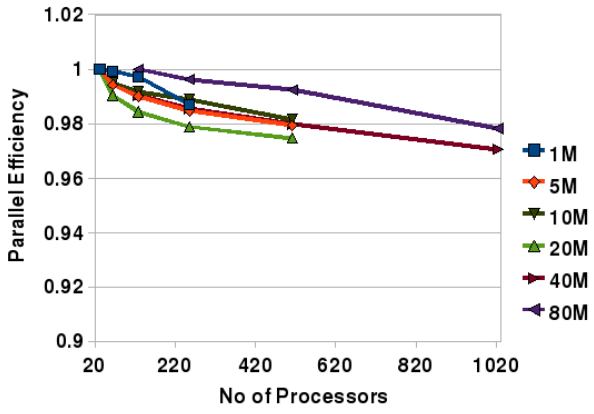
In general, particle dynamics simulations contain particle species that interact through many different force fields with varying ranges. Due to this the dynamics of certain species evolve at a rate very different from another species in simulation. A straightforward implementation of step 2 in (4) would require one to choose the shortest time step required to capture the fastest dynamics. However, in these time stepping schemes all forces are computed at each time step in spite of the fact that some of them do not contribute to the dynamics at the shortest time scale. Such redundant computation can be avoided using multi-scale time integrators [20]. In these schemes the force contribution is separated into long-range and short-range forces. This separation is performed in a manner such that the long-range forces need to be evaluated at coarser time scale only while the short range forces are computed at finer time scales also. Here we propose a scheme to efficiently implement the same within the

framework of hierarchical computation. It is important to ensure that the short range forces are computed between particles whose distance of separation is less than the range of the force. The same can be achieved by limiting the level of the tree up to which short range force multipole and local expansions are computed. In other words, the short range forces are computed only at levels where the maximum distance of separation in interaction list is less than range of the corresponding force. In the overall process at finer time scale the hierarchical computations pertaining to short range forces only are computed on *multiple short* trees and at coarser time scales the long range forces are computed using the entire tree structure.

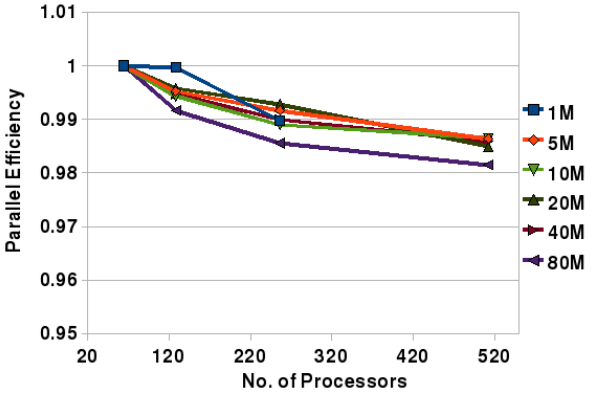
6. Experimental Results

In this section, we present experimental results on the IBM Blue Gene/L that demonstrate the scalability and efficiency of the proposed parallel ACE algorithm using the PD simulation. First, we demonstrate the performance of our algorithm in computing only *step* 1 of equation (4). Given N randomly distributed particles with arbitrary charge q_i, we evaluate the total pairwise potential $\Phi(\mathbf{r}_i)$. Both long and short range potentials in (5) are considered. Three upward traversals are used, one for each of the three potentials $1/R$, $1/R^6$, and $1/R^{12}$, respectively. This is needed due to the differences in translation operators and multipole expansions for each of these three different potentials. In contrast, one downward traversal is used to compute the cumulative potential. The particles are distributed randomly either in a volume or on a spherical surface and varied from 5 to 80 million. In all cases, the number of particles at the leaf nodes was approximately constant, and around 60. All numerical experiments were performed on 32, 64, 128, 256 and 512 processors. Some of the configurations were also executed on 1024 processors, the largest size available on our system. The parallel efficiency reported here is computed using $\nu_{eff} = \frac{T_{ref} N_{ref}}{T_p N_p}$, where T_m and N_m denote the time taken for a particular process and the number of processors in the processor set $m \in \{32, 64, 128, 256, 512, 1024\}$, and ref is the smallest size processor set for a given N. In the case of uniform distributions, $ref = 32$ was chosen in all cases expect for $N = 80 \times 10^6$. In simulations with 80 million particles memory limitation restricts the smallest number of processors to 64. For non-uniform distributions, $ref = 64$ was used.

Figures 2(a) and 2(b) show the parallel efficiency in computation of total potential for both volume and surface distribution of particles. The algorithm



(a)



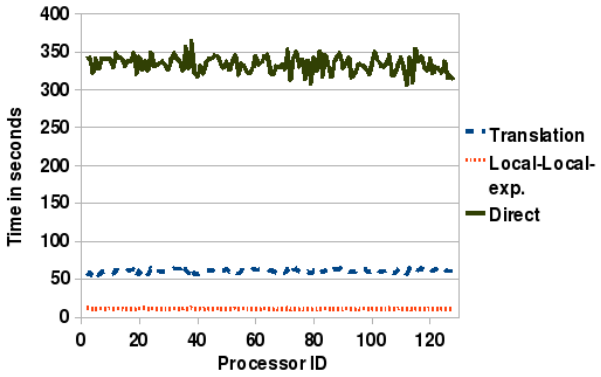
(b)

Figure 2. Parallel efficiency vs. number of processors for randomly distributed particles (a) in a cubical volume and (b) on a spherical surface.

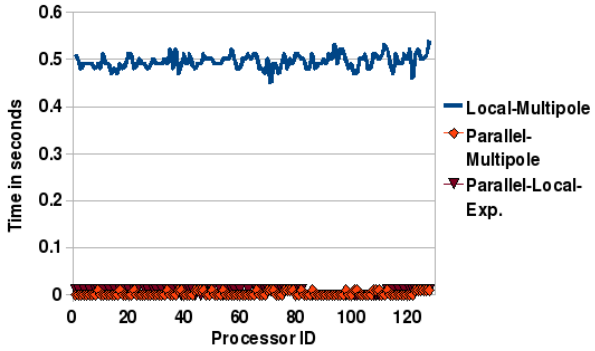
proposed here has efficiency ≈98% as the number of processors is increased from 32 or 64 to 512. Simulations of pairwise interactions between 40 and 80 million particles was computed using 1024 processors, with an efficiency greater than 97%. These results demonstrate the scalability of the proposed algorithm to several hundreds of processors. Note that this efficiency is attained without any explicit load-balancing, or tuning, or optimization mechanisms. The timing measurements for different stages of hierarchical computation on different processor sets for 20 million and 40 million particles are shown in Table 1. This data is representative and is characteristic of all other numerical experiments. From Table 1 it is evident that the time spent in parallel multipole accumulation and parallel local expansion distribution during upward and downward tree traversals, respectively, are negligible when compared to the other computations. This is in accordance with our algorithm that is designed for

Table 1. Sample average times in seconds for 20 and 40 million particles as a function of the number of processors. The two most time consuming stages are near field computation and translations, in that order.

Proc.	Tree-Build	Local-Multipole	Parallel-Multipole	Translation	Parallel-Local-exp	Local-Local-exp	Near-field
	N=20,000,000						
32	24.94	1.97	0	241.35	0.02	43.5	1307.56
64	14.77	0.99	0	121.31	0	21.76	660.91
128	9.96	0.5	0	60.76	0	10.89	332.83
256	7.68	0.25	0.01	30.72	0.01	5.44	167.19
512	6.62	0.12	0.01	15.62	0.01	2.72	83.75
	N=40,000,000						
32	71.12	5.02	0	645.99	0.03	87.58	2025.43
64	41.84	2.52	0	323.65	0	43.81	1019.07
128	27.85	1.26	0	162.26	0.01	21.91	512.32
256	21.2	0.63	0.01	81.57	0.01	10.96	257.37
512	18.05	0.32	0.02	41.41	0.02	5.48	129.06
1024	16.61	0.16	0.03	21.49	0.05	2.74	64.53



(a)



(b)

Figure 3. Individual processor time for different hierarchical computations for N=20 million and 128 processors.

efficient communication.

Next, Figure 3 shows run-time on individual processors for different parts of the hierarchical algorithm for a 20 million particle system on 128 processors. Except for relatively minor variation, the run-times

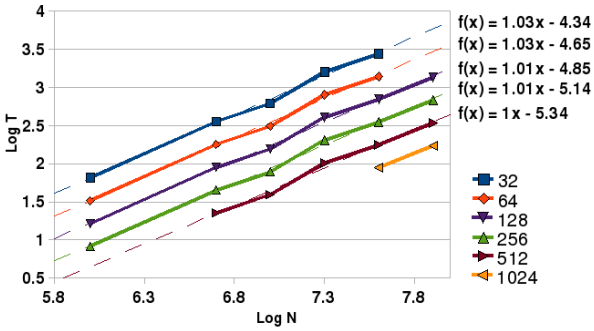


Figure 4. Time per processor vs. number of particles (N) for different processor sets.

for all steps of the algorithm are nearly identical at every processor, indicating near perfect load balancing. An examination of the total time per processor as a function of the number of particles N, shown in Figure 4, indicates that the run-time of the parallel algorithm increases linearly with problem size over a wide range of processor sets.

The next result pertains to particle simulation within the framework of the PACE algorithm. First we show that the suggested mechanism of building a full tree results in significant advantage over rebuilding the tree at every time step. Table 2 compares the time in seconds, averaged across the processors, for building the tree and for updating the leaf boxes with the new particle positions after each time step for simulations with different number of particles.

It is evident that in all cases the time spent in updating the tree is orders of magnitude less than that required for rebuilding the tree at the beginning of each time step. Next, we show the efficiency of the proposed scheme for implementation of multiple

Table 2. Average time (in seconds) spent on building the tree and updating points in the full tree at each time step for different number of particles N

N	Update Time	Tree Time
100,000	0.18733	17.7523
250,000	0.46984	15.0131
500,000	0.94673	21.4150
1,000,000	1.87960	21.2756

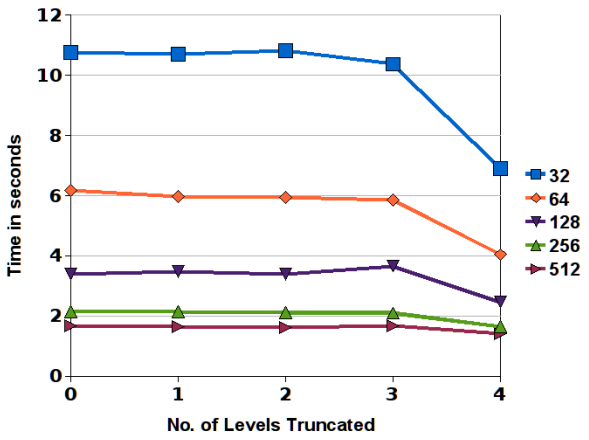
scale time integrator within the framework of ACE algorithm. From Table 1, it is evident that the dominant part in the hierarchical computation is the translation operation. The translation time, averaged across processors, against different levels of tree truncation for 1 million particles is plotted in Figure 5(a). Higher number of truncated levels correspond to smaller tree heights. For example, when the number of truncated levels is 4, only leaf box interactions are considered. Figure 5(b) shows the average time for different number of particles on a fixed processor set (32 processors) for different levels of tree truncation. In both cases the simulation time decreases as number of tree levels truncated increases. This leads to significant savings in overall simulation time as this evaluation would be repeated for multiple simulation time steps. Thus the truncated tree evaluation provides an efficient means of implementing the multiple scale time integrators within the framework of multipole methods.

7. Conclusions

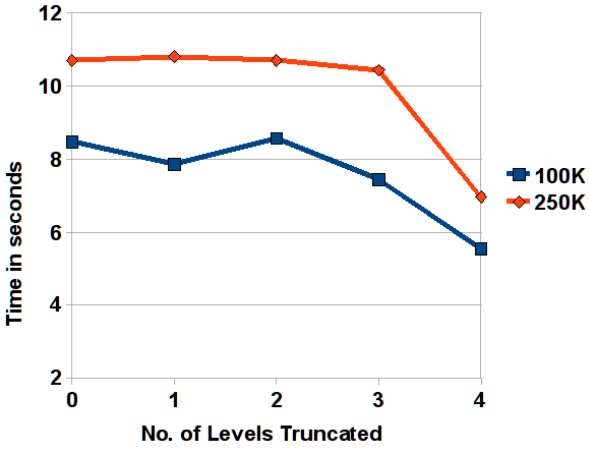
In this paper, we presented a parallel framework for physical simulations that can simultaneously evaluate multiple potentials using the same numerical and computational method. Our method uses an FMM like framework and is based on the the theory of accelerate Cartesian expansions. The advantages of our method include optimal computation of multiple potentials using the same parallel method, computational efficiencies resulting from reuse of computation across multiple potentials, considerably reduced burden on code development, and high parallel efficiency. We demonstrated these advantages by building a rudimentary particle dynamics application code and testing its performance on parallel systems.

Acknowledgements

The research is supported in part by the National Science Foundation under CCF-0729157, OCI-0835466, and DMS-0811197.



(a)



(b)

Figure 5. Translation time for varying number of active levels in tree for (a) different number of processors and (b) varying N on 32 processors.

References

[1] V. Rokhlin, "Rapid solution of the integral equations of classical potential theory," *Journal of Computational Physics*, vol. 60, pp. 187–207, 1985.

[2] L. Greengard, *The rapid evaluation of potential fields in particle systems*. Cambridge, MA: MIT Press, 1988.

[3] B. Shanker and H. Huang, "Accelerated Cartesian expansions - a fast method for computing of potentials of the form $R^{-\nu}$ for all real ν," *Journal of Computational Physics*, vol. 226, pp. 732–753, 2007. Technical Report; MSU-ECE-Report-2006-5.

[4] P. B. Callahan and S. R. Kosaraju, "A decomposition of multidimensional point sets with applications to k-nearest neighbors and n-body potential fields," *Journal of the ACM*, vol. 42, pp. 67–90, 1995.

[5] M. S. Warren and J. K. Salmon, "Astrophysical n-body simulations using hierarchical tree data structures," *Proceedings of Supercomputing*, p. 570576, 1992.

[6] M. Warren and J. Salmon, "A parallel hashed oct-tree n-body algorithm," *Proceedings of Super-computing*, pp. 1–12, 1993.

[7] J. Singh, C. Holt, T. Tosuka, 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*, vol. 27, pp. 118–141, 1995.

[8] P. Liu and S. Bhatt, "Experiences with parallel n-body simulation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, pp. 1306–1323, 2000.

[9] F. Sevilgen, S. Aluru, and N. Futamura, "A provably optimal, distribution-independent parallel fast multipole method," *Proceedings of International Parallel and Distributed Processing Symposium IEEE Computer Society Press*, pp. 77–84, 2000.

[10] S. H. Teng, "Provably good partitioning and load balancing algorithms for parallel adaptive n-body simulation," *SIAM Journal on Scientific Computing*, vol. 19, pp. 635–656, 1998.

[11] H. Schwichtenberg, G. Winter, and H. Wallmeier, "Acceleration of molecular mechanic simulation by parallelization and fast multipole techniques," *Parallel Computing*, vol. 25, pp. 535–546, 1999.

[12] Y. Yuan and P. Banerjee, "A parallel implementation of a fast multipole based 3-d capacitance extraction program on distributed memory multicomputers," *Journal of Parallel and Distributed Computing*, vol. 61, pp. 1751–1774, 2001.

[13] S. Velamparambil, J. M. Song, W. C. Chew, and G. K., "ScaleME: A portable scalable multipole engine for electromagnetic and acoustic integral equation solvers," *IEEE AP-S Int. Symp.*, vol. 3, pp. 1774–1777, 1998.

[14] B. Hariharan, S. Aluru, and B. Shanker, "A scalable parallel fast multipole method for analysis of scattering from perfect electrically conducting surfaces," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, (Los Alamitos, CA, USA), pp. 1–17, IEEE Computer Society Press, 2002.

[15] M. Griebel, S. Knapek, and G. Zumbusch, *Numerical simulation in Molecular Dynamics*. Springer-Verlag, 2007.

[16] M. Vikram and B. Shanker, "An incomplete review of fast multipole methods -from static to wideband -as applied to problems in computational electromagnetics," *ACES (accepted for publication)*, 2008.

[17] M. Vikram and B. Shanker, "Fast evaluation of time domain fields in sub-wavelength source/observer distributions using accelerated Cartesian expansions (ACE)," *Journal of Computational Physics*, vol. 227, pp. 1007–1023, 2007.

[18] M. Vikram, A. Baczewski, B. Shanker, and L. Kempel, "Accelerated cartesian expansion (ace) unified framework for the rapid evaluation of potentials associated with the diffusion, lossy wave and klein-gordon equations," *Submitted to Journal of Computational Physics*, 2008.

[19] F. Zhao, "An $\mathcal{O}(n)$ algorithm for three-dimensional n-body simulation," Master's thesis, Massachusetts Institute of Technology, 1987.

[20] M. Tuckermar, B. J. Berne, and G. J. Martyna, "Reversible multiple time scale molecular dynamics," *J. Chem. Phys.*, vol. 97, August 1992.