2003

Scalable parallel molecular dynamics algorithms for organic systems

Satyavani Vemparala
Louisiana State University and Agricultural and Mechanical College

SCALABLE PARALLEL MOLECULAR DYNAMICS ALGORITHMS FOR ORGANIC SYSTEMS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Systems Science

in

The Interdepartmental Program in Computer Science

by
Satyavani Vemparala
M.Sc. University of Hyderabad, Hyderabad, India, 1996
M.Tech. Indian Institute of Technology, Kharagpur, India, 1998
December 2003

ACKNOWLEDGMENTS

I am thankful to my advisor, Dr. Aiichiro Nakano, for his support and encouragement in my research at Louisiana State University. I am grateful to Dr. S. S. Iyengar for serving as a co-chair on my committee.

I am thankful Drs. Donald Kraft and Bijaya Karki for serving on my committee. Many thanks go to Dr. Kikuchi for the help given by him during the benchmarking tests. Thanks to Dr. Joel Tohline and Monika Lee for the usage of *SuperMike*.

Finally thanks to my family for all their support and encouragement.

TABLE OF CONTENTS

ABSTRACT

A scalable parallel algorithm, Macro-Molecular Dynamics (MMD), has been developed for large-scale molecular dynamics simulations of organic macromolecules, based on space-time multi-resolution techniques and dynamic management of distributed lists. The algorithm also includes the calculation of long range forces using Fast Multipole Method (FMM). FMM is based on the octree data structure, in which each parent cell is divided into 8 child cells and this division continues until the cell size is equal to the non-bonded interaction cutoff length. Due to constant number of operations performed at each stage of the octree, the FMM algorithm scales as $O(N)$. Design and analysis of MMD and FMM algorithms are presented. Scalability tests are performed on three tera-flop machines: 1024-processor Intel Xeon-based Linux cluster, *SuperMike* at LSU, 1184-processor IBM SP4 *Marcellus* and the 512-processor Compaq AlphaServer *Emerald* at the U.S. Army Engineer Research and Development Center (ERDC) MSRC. The tests show that the Linux cluster outperforms the SP4 for the MMD application. The tests also show significant effects of memory- and cache-sharing on the performance.

CHAPTER 1
INTRODUCTION

Recent advances in linear-scaling simulation algorithms and scalable parallel-computing frameworks have made it possible to carry out large-scale atomistic simulations of inorganic materials (e.g., semiconductors and ceramics) based on the molecular dynamics (MD) method on thousands of processors [1,2,3]. For example, a recent benchmark test [4] has demonstrated a 6.44-billion-atom MD simulation on a 1024-processor Teraflop architecture. These MD simulations of inorganic materials are coarse grained (large number of atoms/processor), typically involving 10^4-10^6 atoms per processor, and they are based on spatial decomposition [5], in which the physical system is partitioned into subsystems of smaller volumes. In spatial decomposition, load balancing may be addressed easily with a computational-space-decomposition scheme [6,7].

Significant progress has also been made for parallel MD simulations of organic macromolecules such as polymers and proteins [8,9,10,11,12]. To handle the sequential nature of requisite constrained MD algorithms [13] and the complex computation of bonded interactions among atomic *n*-tuples (pairs, triplets, and quadruplets), earlier parallel MD algorithms of organic macromolecules were based on atom decomposition, in which the list of atoms are partitioned into sub-lists of smaller numbers, and the replicated data strategy [14,15], in which these sub-lists are replicated on all processors. Another parallelization scheme includes force decomposition, which applies block decomposition to the force matrix for atomic pairs [16]. A state-of-the art parallel algorithm for biological MD simulations employs hybrid force/spatial decomposition and prioritized message-driven execution to achieve a 327000-atom simulation on

3000 processors [17]. However, these parallel MD simulations of organic macromolecules are typically fine-grained (small number of atoms/processor), including ~10^2 atoms per processor.

While parallel MD algorithms for inorganic and organic materials are based on distinct parallelization schemes, recent advances in biomedical technologies have made it imperative to develop parallel MD algorithms for very large-scale biological macromolecular simulations with coarse granularity. For example, a viable self-assembly approach to semiconductor quantum-dot architecture involves large 2D arrays of proteins as a template [18]. Each unit of such an array contains ~10^5 atoms and even a small 10×10 array simulation requires 10^7-atom MD simulations. Large-scale MD simulations of bio-molecular systems, from the single-molecule to the cellular level, are also needed so that a virtual cell may be constructed to enable *in silico* evaluation of systems responses to novel drugs [19]. Again even a small 10×10 array of membrane proteins and ion channels, embedded in a cell membrane with solvent molecules such as water, requires 10^7-atom MD simulations. Scalability consideration at large granularities of these large-scale macro-molecular simulations argue for spatial decomposition, but this requires nontrivial handling of constrained dynamics and bonded interactions.

A spatial-decomposition-based parallel MD algorithm for organic macromolecular simulations is developed in this thesis. The macro-molecular dynamics (MMD) algorithm employs penalty-function-based constrained MD [20] combined with a multiple time-scale method [21], and dynamic management of distributed data structures for force computations involving atomic n-tuples. The most time consuming part of an MD simulation of macromolecules involves the computation of non-bonded interactions such as van der Waals and long-range Coulombic interactions, direct computation of which requires $O(N^2)$ operations for N atoms. The MMD code reduces this complexity to $O(N)$ using: linked-cell [22,23] and Verlet-

neighbor lists for van der Waals interactions; and the fast multipole method (FMM) [24,25,26] for Coulomb interactions. For anticipated large-scale (10^7-10^9 atoms) applications, the asymptotic $O(N)$ scaling of the FMM is preferable to other algorithms, such as the $O(N\log N)$ particle mesh Ewald method [17, 27].

Another important design consideration for parallel MD simulations is performance portability among high-end computing architectures. There is a particular interest in commodity-based Linux clusters constructed from off-the shelf components. For example, Louisiana State University (LSU) has recently acquired a Linux cluster consisting of 512 dual Intel Xeon 1.8 GHz nodes (i.e., 1024 processors) connected by Myricom's Myrinet interconnect [28]. The performance of this $ 2.6 million cluster, *SuperMike*, is rated as 2.21 Tflops, according to the standard High Performance Linpack benchmark [29], and *SuperMike* was ranked as the 11th fastest supercomputer in the world in August 2002 [30]. Although the performance of low-cost multi-Teraflop Linux clusters has thus been confirmed by standard benchmark tests, there is a continuing concern regarding the scalability of such architecture for real high-end scientific/engineering applications, in comparison with conventional higher-end parallel supercomputers such as IBM SP4. A comparative performance study of the MMD code on the *SuperMike* and the 1184-processor IBM SP4 system, *Marcellus*, at the Naval Oceanographic Office (NAVO) Major Shared Resource Center (MSRC) and 512-processor Compaq Alpha Server *Emerald* at the U.S. army Engineer Research and Development Center (ERDC) has been performed.

This thesis is arranged as follows. In chapter 2, the methodology of molecular dynamics and the force fields incorporated in the code are described. In chapter 3 a description of the various dynamic list management techniques for efficient calculations and the algorithms

detailing the same are presented. The parallelization scheme and the multi-resolution techniques employed in the code are discussed in chapter 4. The fast multipole method, which is used to compute the long-range coulomb interactions, is discussed in chapter 5. Finally the scalability tests, the test beds, results and conclusions are presented in the chapter 6.

CHAPTER 2
SIMULATION METHODOLOGY

2.1 Formulation of Molecular Dynamics

In MD simulations, the physical system is represented by a set of N atoms. The trajectory, i.e., positions, $\{\vec{r}_i | i=1,...,N\}$, and velocities, $\{\vec{v}_i | i=1,...,N\}$, of all the atoms is followed by numerically integrating the Newton's equations of motion,

$$m_i \frac{d^2\vec{r}_i}{dt^2} = \vec{F}_i \ , \tag{2.1}$$

where m_i is the mass of atom i and the force on atom i is defined as

$$\vec{F}_i = -\frac{\partial V(\{\vec{r}_i\})}{\partial \vec{r}_i} . \tag{2.2}$$

The interatomic potential energy, V, encodes interactions among atoms and therefore is the essential ingredient of MD simulations. In organic macromolecular systems, in addition to the atomic positions, connectivity information $\{Adj(i) | \ i=1,...N\}$ is also needed to calculate the interatomic potential energy. $Adj(i)$ is the list of atoms connected to atom i, *i.e.*, a macromolecular system is represented as a graph data structure, where atoms are vertices and $\{Adj(i)\}$ represents edges.

2.2 Force Field Parameterization

A force field model [31,32] in which the interatomic potential energy consists of the bonded and non-bonded interaction terms was used,

$$V = V_{bonded} + V_{non-bonded} . \tag{2.3}$$

Here V_{bonded} represents bond-stretching, bond-bending, and torsion interactions:

$$V_{bonded} = V_{stretch} + V_{bend} + V_{torsion}, \tag{2.4}$$

which are sums over atomic pairs, triplets, and quadruplets, respectively:

$$V_{stretch} = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ (i<Adj(i)_j)}}^{|Adj(i)|} v_2(\vec{r}_i, \vec{r}_{Adj(i)_j}), \tag{2.5}$$

$$V_{bend} = \sum_{j=1}^{N} \sum_{i=1}^{|Adj(j)|-1} \sum_{k=i+1}^{|Adj(j)|} v_3(\vec{r}_{Adj(j)_i}, \vec{r}_j, \vec{r}_{Adj(j)_k}), \tag{2.6}$$

$$V_{torsion} = \sum_{j=1}^{N} \sum_{\substack{k=1 \\ (j<Adj(j)_k)}}^{|Adj(j)|} \sum_{\substack{i=1 \\ (Adj(j)_i \neq Adj(j)_k)}}^{|Adj(j)|} \sum_{\substack{l=1 \\ (Adj(k)_l \neq j)}}^{|Adj(k)|} v_4(\vec{r}_{Adj(j)_i}, \vec{r}_j, \vec{r}_{Adj(j)_k}, \vec{r}_{Adj(k)_l}). \tag{2.7}$$

In Eqs. (2.5)-(2.7), $Adj(j)_i$ denotes the ith element in the list $Adj(j)$, which has the connectivity information of atom j.

The $V_{non-bonded}$ represents the van der Waals and Coulombic interactions:

$$V_{non-bonded} = V_{vdw} + V_{col}, \tag{2.8}$$

$$V_{vdw} = \sum_{i<j} v_{\mathrm{vdw}}(r_{ij}), \tag{2.9}$$

$$V_{col} = \sum_{i<j} \frac{q_i q_j}{\varepsilon r_{ij}}, \tag{2.10}$$

where q_i is the partial atomic charge of atom i, $r_{ij} = |\vec{r}_{ij}|$, $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j$, and ε is the dielectric constant. A pair of atoms i and j are considered to be non-bonded if they belong to different molecules or are atoms in the same molecule separated by more than two atoms. The Newton's equations of motion in Eq. (2.2) are integrated using the velocity Verlet algorithm [22]:

$$\vec{r}_i(t+\Delta t) = \vec{r}_i(t) + \vec{v}_i(t)\Delta t + \frac{1}{2}\left(\frac{\vec{F}_i(t)}{m_i}\right)\Delta t^2 + O(\Delta t^3) \quad (i = 1,\ldots,N) \tag{2.11}$$

$$\vec{v}_i(t+\Delta t) = \vec{v}_i(t) + \frac{1}{2}\left[\frac{\vec{F}_i(t+\Delta t) + \vec{F}_i(t)}{m_i}\right]\Delta t \ + O(\Delta t^3) \ (i = 1,\ldots,N) \qquad (2.12)$$

The MMD algorithm employs different strategies to compute the bonded and non-bonded interactions. Efficient algorithms have been developed for computation of atomic *n*-tuple contributions to V_{bonded} using adjacency lists, *Adj*. The $V_{non\text{-}bonded}$ is computed using linked-cell and neighbor lists, and V_{col} in particular is computed using the fast multipole method (FMM). The following chapters describe the techniques of evaluating the bonded and non-bonded interactions.

CHAPTER 3
LIST MANAGEMENT ALGORITHMS

The interaction potentials, based on the interaction distance, can be divided into short- and long-range potentials. The bonded interactions have a range of 3-4 Å and hence are short-ranged in nature. The non-bonded interactions, van der Waals, can also be short-ranged (~9-10 Å), if cut-off distance is used. The Coulomb interactions can be divided into near-field (short-range) and far-field (long-range) contributions. The near-field contribution to the Coulomb interactions is calculated in a manner similar to the calculation of van der Waals interactions. This chapter describes the algorithms that are used in MMD code to compute the short-range interactions (both bonded and non-bonded) efficiently. Chapter 4 describes methods of handling long-range interactions.

3.1 Short-Range Bonded Lists

The bonded interactions, in Eqs. (2.5)-(2.7), are calculated using three lists: *Adj, list_bend* and *list_tors*. The list *Adj* contains the connectivity information of all the atoms, *list_bend* is the list of all triplets and *list_tors* is the list of all quadruplets that are constructed, both of which are constructed using the list *Adj,* by the algorithms in Figures 3.1 and 3.2.

Input : list *Adj*, number of atoms *N*
Output: list *list_bend*, *Nbend*
MAKE_LIST_BEND(*Adj*, *N*)
ibend ← 0
for *j* ← 1, *N*
  for *i* ← 1, |*Adj*(*j*)|-1
    for *k* ← *i*+1, |*Adj*(*j*)|
      ibend ← *ibend*+1
      list_bend(*ibend*,1) = $Adj(j)_i$
      list_bend(*ibend*,2) = *j*
      list_bend(*ibend*,3) = $Adj(j)_k$
      Nbend ← *ibend*

Figure 3.1. List construction algorithm for *list_bend*.

```
Input : list Adj, number of atoms N
Output: list list_tors, Ntors
MAKE_LIST_TORS(Adj, N)
itors ← 0
for j ← 1, N
    for k ← 1, |Adj(j)|
        if (j < k) then
            for i ← 1, |Adj(j)|
                if (i ≠ k) then
                    for l ← 1, |Adj(k)|
                        if (l ≠ j) then
                            itors ← itors+1
                            list_tors(itors,1) = Adj(j)_i
                            list_tors(itors,2) = j
                            list_tors(itors,3) = Adj(j)_k
                            list_tors(itors,4) = Adj(k)_l
                            Ntors ← itors
```

Figure 3.2. List construction algorithm for *list_tors*.

In the algorithm in Figure 3.1, *list_bend* was constructed by going over the 'central' atom, *j*, of triplet *ijk,* which has two or more atoms adjacent to it. The condition $i<k$ ensures that any triplet *ijk* is counted only once. The algorithm in Figure 3.2 uses a similar strategy to construct *list_tors*. For every edge *jk* (the condition $j<k$ ensures that each edge is counted only once), vertices *i* and *l* that satisfy the conditions, $i \neq k$ and $j \neq l$ are found. *list_bend* and *list_tors*, are used to compute the forces on atoms due to the bonded interactions, $\vec{F}_{stretch}$, $\vec{F}_{bend}$, and $\vec{F}_{torsion}$, based on the algorithms in Figures 3.3-3.5.

Input: list *Adj*, number of atoms *N*, positions $\{\vec{r}_i\}$
Output: V_{stretch}, $\vec{F}_{stretch}$
COMPUTE_STRETCH_POTENTIAL(*Adj*, *N*, $\{\vec{r}_i\}$)

$V_{\text{stretch}} = 0$
$\vec{F}_{stretch} = 0$
for $i \leftarrow 1, N$
  for $k \leftarrow 1, |Adj(i)|$
    $j \leftarrow Adj(i)_k$
    if $(j > i)$ **then**

$$V_{\text{stretch}} = V_{\text{stretch}} + v_2(\vec{r}_i, \vec{r}_j)$$

$$\vec{F}_{stretch}(i) = \vec{F}_{stretch}(i) + \frac{\partial v_2(\vec{r}_i, \vec{r}_j)}{\partial \vec{r}_i}$$

$$\vec{F}_{stretch}(j) = \vec{F}_{stretch}(j) - \frac{\partial v_2(\vec{r}_i, \vec{r}_j)}{\partial \vec{r}_i}$$

Figure 3.3 Bond-stretching potential and force calculations.

Input: number of triplets *Nbends*, list *list_bend*, number of atoms *N*, positions $\{\vec{r}_i\}$

Output: V_{bend}, $\vec{F}_{bend}$

COMPUTE_BEND_POTENTIAL(*Nbends*, *list_bend*, *N*, $\{\vec{r}_i\}$)

$V_{\text{bend}} = 0$

$\vec{F}_{bend} = 0$

for *ibend* = 1,*Nbends*

i = *list_bend*(*ibend*,1)

j = *list_bend*(*ibend*,2)

k = *list_bend*(*ibend*,3)

$V_{\text{bend}} = V_{\text{bend}} + v_3(\vec{r}_i, \vec{r}_j, \vec{r}_k)$

$$\vec{F}_{bend}(i) = \vec{F}_{bend}(i) - \frac{\partial v_3(\vec{r}_i, \vec{r}_j, \vec{r}_k)}{\partial \vec{r}_{ij}}$$

$$\vec{F}_{bend}(j) = \vec{F}_{bend}(j) + \frac{\partial v_3(\vec{r}_i, \vec{r}_j, \vec{r}_k)}{\partial \vec{r}_{ij}} + \frac{\partial v_3(\vec{r}_i, \vec{r}_j, \vec{r}_k)}{\partial \vec{r}_{jk}}$$

$$\vec{F}_{bend}(k) = \vec{F}_{bend}(k) - \frac{\partial v_3(\vec{r}_i, \vec{r}_j, \vec{r}_k)}{\partial \vec{r}_{jk}}$$

Figure 3.4 Bond-bending potential and force calculations.

Input: number of quadruplets *Ntors*, list *list_tors*, number of atoms *N*, positions $\{\vec{r}_i\}$

Output: V_{torsion}, $\vec{F}_{torsion}$

COMPUTE_TORSION_POTENTIAL(*Ntors, list_tors*, *N*, $\{\vec{r}_i\}$)

$V_{\text{torsion}} = 0$

$\vec{F}_{torsion} = 0$

for *itors* = 1, *Ntors*

i = *list_tors*(*itors*,1)

j = *list_tors*(*itors*,2)

k = *list_tors*(*itors*,3)

l = *list_tors*(*itors*,4)

$V_{\text{torsion}} = V_{\text{torsion}} + v_4(\vec{r}_i, \vec{r}_j, \vec{r}_k, \vec{r}_l)$

$$\vec{F}_{torsion}(i) = \vec{F}_{torsion}(i) - \frac{\partial v_4(\vec{r}_i, \vec{r}_j, \vec{r}_k, \vec{r}_l)}{\partial \vec{r}_{ij}}$$

$$\vec{F}_{torsion}(j) = \vec{F}_{torsion}(j) + \frac{\partial v_4(\vec{r}_i, \vec{r}_j, \vec{r}_k, \vec{r}_l)}{\partial \vec{r}_{ij}} - \frac{\partial v_4(\vec{r}_i, \vec{r}_j, \vec{r}_k, \vec{r}_l)}{\partial \vec{r}_{jk}}$$

$$\vec{F}_{torsion}(k) = \vec{F}_{torsion}(k) + \frac{\partial v_4(\vec{r}_i, \vec{r}_j, \vec{r}_k, \vec{r}_l)}{\partial \vec{r}_{jk}} - \frac{\partial v_4(\vec{r}_i, \vec{r}_j, \vec{r}_k, \vec{r}_l)}{\partial \vec{r}_{kl}}$$

$$\vec{F}_{torsion}(l) = \vec{F}_{torsion}(l) + \frac{\partial v_4(\vec{r}_i, \vec{r}_j, \vec{r}_k, \vec{r}_l)}{\partial \vec{r}_{kl}}$$

Figure 3.5 Torsion potential and force calculations.

Time complexities for computing bond-stretching, bond-bending, and torsion forces are $O(Nd)$, $O(Nd^2)$, and $O(Nd^3)$ (d is the node degree, $|Adj(i)|$, averaged over i), respectively, since the corresponding force formulas, Eqs.(2.5)-(2.7), involve singly-, doubly-, triply-nested loops over *Adj,* in addition to a loop over atoms *N*.

3.2 Short-Range Non-Bonded Lists

For van der Waals contribution, Eq. (2.9), to the non-bonded potential energy, a cutoff distance r_c is introduced beyond which the potential and force are set to zero. The MMD code uses two popular methods [33] to compute the van der Waals interaction and near-field contribution to Coulomb interactions in $O(N)$ time: (i) linked-cell- and (ii) Verlet-neighbor-list methods.

3.2.1 Linked-Cell-List Method

In the linked-cell-list method [5], the physical system is divided into $(M_x \times M_y \times M_z)$ cells of cell edge > r_c. There are on average $N_c = N/(M_x \times M_y \times M_z)$ atoms in each cell. Because the cell edge is at least r_c, an atom in a particular cell m has to search only the 26 neighboring cells (in 3D case) in addition to cell m to find all the atoms within the distance r_c.

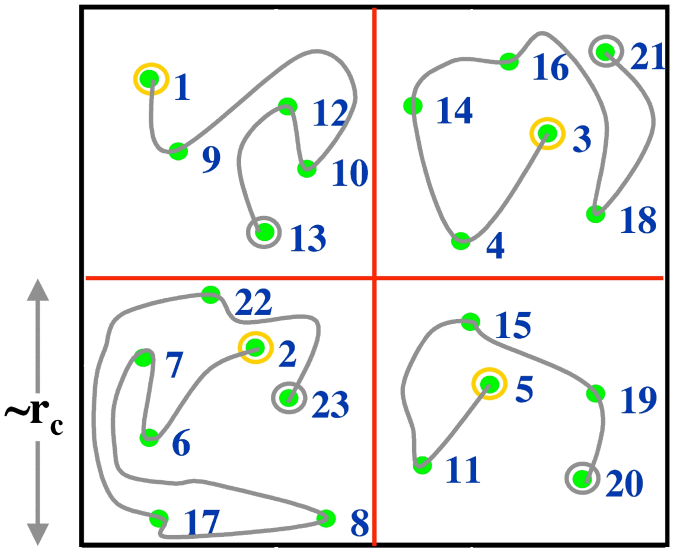

Figure 3.6 The cell lists in the MD algorithm. The atoms, which have yellow circles, are the head atoms. The solid line connects all the atoms in each cell.

This computation scales as $27NN_c$, compared with the $O(N^2)$ brute force search. Within each cell, a linked-list *lscl* is constructed over atoms where *lscl*(m) is the linked list of atoms (see

Figure 3.6) in cell *m* and *lshd*(*m*) points to the first element of *lscl*(*m*). An algorithm to construct linked-cell-list uses two data structures, *lscl* and *lshd* is given in Figure 3.7.

Input: number of atoms *N*, positions $\{\vec{r}_i\}$, cutoff radius r_c, number of cells *M*.
Output: linked list *lscl*, header list *lshd*
MAKE_LINKED_CELL(*N*, $\{\vec{r}_i\}$, r_c, *M*)
    $\forall m$, *lshd*(*m*) = 0
    for *i* = 1, *N*
        $m_{i\alpha} = \lfloor (r_{i\alpha} + r_c)/r_c \rfloor$ ($\alpha = x, y, z$; $r_{i\alpha}$ is the αth component of $\vec{r}_i$)
        $m = m_x \times M_y M_z + m_y \times M_z + m_z + 1$
        lscl(*i*) = *lshd*(*m*)
        lshd(*m*) = *i*

Figure 3.7. Linked-cell-list construction algorithm.

3.2.2 Verlet Neighbor List Method

The neighbor list [34] of atom *i* is a simple data structure containing all the atoms within the distance r_c from atom *i*, after certain exclusion rules are applied. In case of macromolecules, typically 1-4 exclusion rule [31,32] is implemented, which states that any atom *j* that participates in a quadruplet with atom *i* is excluded from the neighbor list of *i*. Depending on the time step used in the simulation, this list can be constructed every *k* steps, which further reduces the computation time. With in the *k* steps, the same list can be used for the non-bonded computation. This parameter *k* can be either set at the beginning of the simulation, or can be automated with in the simulation, if any pair of atoms crosses a particular distance. An algorithm to construct the neighbor list is given in Figure 3.8, where *InsertList*(*lsnb*(*i*),*j*) is a function that inserts the atom *j* in the neighbor list, *lsnb* of atom *i*.

Input: positions $\{\vec{r}_i\}$, cutoff radius r_c, linked-cell list *lscl*, header-cell list *lshd*
Output: neighbor list *lsnb*
MAKE_LSNB($\{\vec{r}_i\}$, r_c, *lscl*, *lshd*)
    for m_z = 1, M_z
    for m_y = 1, M_y
    for m_x = 1, M_x
        $m = m_x \times M_y M_z + m_y \times M_z + m_z + 1$
        for k_z = -1, 1

Figure 3.8 Neighbor list construction algorithm (Fig. Cont'd).

for k_y ← -1, 1
for k_x ← -1, 1
m ← $(m_x + k_x) + M_y M_z + (m_y + k_y) + M_z + (m_z + k_z) + 1$
i ← *lshd*(m)
while ($i \neq 0$)
j ← *lshd*(m)
if ({i,j} ∈ any *list_tors*()) **then**
while ($j \neq 0$)
if ($j > i$ & $\left|\vec{r}_i - \vec{r}_j\right| \leq r_c$) **then**
InsertList(*lsnb*(i), j)
j ← *lscl*(j)
i ← *lscl*(i)

The Verlet-neighbor list, defined in Figure 3.8, is used to calculate van der Waals interaction and near-field contribution to Coulombic interaction in the algorithm in Figure 3.9.

Input: list *lsnb*, number of atoms N, position $\{\vec{r}_i\}$
Output: V_{vdw}, $\vec{F}_{vdw}$, V_{Col}^{near}, $\vec{F}_{Col}^{near}$
COMPUTE_VDW_COL_POTENTIAL(*lsnb*, N, $\{\vec{r}_i\}$)
$V_{vdw} = 0$
$\vec{F}_{vdw} = 0$
$V_{Col}^{near} = 0$
$\vec{F}_{Col}^{near} = 0$
for i ← 1, N
for k ← 1, |*lsnb*(i)|
j ← *lsnb*(i)$_k$
$V_{vdw} = V_{vdw} + v_{vdw}(\vec{r}_i, \vec{r}_j)$
$\vec{F}_{vdw}(i) = \vec{F}_{vdw}(i) + \dfrac{\partial v_{vdw}(\vec{r}_i, \vec{r}_j)}{\partial \vec{r}_i}$
$\vec{F}_{vdw}(j) = \vec{F}_{vdw}(j) - \dfrac{\partial v_{vdw}(\vec{r}_i, \vec{r}_j)}{\partial \vec{r}_i}$
$V_{Col}^{near}(i) = V_{Col}^{near}(i) + V_{Col}^{near}$
$\vec{F}_{Col}^{near}(i) = \vec{F}_{Col}^{near}(i) + \dfrac{\partial v_{Col}^{near}(\vec{r}_i, \vec{r}_j)}{\partial \vec{r}_i}$
$\vec{F}_{Col}^{near}(j) = \vec{F}_{Col}^{near}(j) - \dfrac{\partial v_{Col}^{near}(\vec{r}_i, \vec{r}_j)}{\partial \vec{r}_i}$

Figure 3.9. van der Waals and near-field Coulomb potential and force calculations.

The far-field contributions to the Coulomb interactions are handled via Fast Multipole Method and are described in the next chapter.

CHAPTER 4
SPACE-TIME MULTIRESOLUTION ALGORITHMS

4.1 Multiple Time Scale Method

The techniques described in chapter 3 reduce the computational complexity for the non-bonded interactions, which are the most time consuming part in MD simulations. The next problem is to choose the time discretization unit, Δt, for numerical integration of Eqs. (2.11) and (2.12). The bonded interactions vary much more rapidly than the non-bonded interactions and Δt needs to be chosen smaller than the smallest time scale. The multiple time-scale (MTS) method reduces the number of force computations significantly by separating the fast (small time-scale) modes from the slow (large time-scale) modes. The non-bonded interactions, which are more time consuming and slowly varying, are computed less often than the bonded interactions, which are rapidly varying, without sacrificing the accuracy of the computation. An MTS algorithm called rRESPA [21,35,36], which is reversible since it is based on the symmetric Trotter expansion of the time evolution operator, is used in the MMD code. The rRESPA algorithm is also symplectic, i.e., the phase space volume occupied by the atoms is a loop invariant, which results in long time stability of solutions. Figure 4.1 shows our 2-level velocity-Verlet rRESPA algorithm.

compute non–bonded forces $\{ \vec{F}_{non\_bond,i}(t) \}$

for *outer* = 1 **to** *nstp* / *mmts*

$$\vec{v}_i(t + mmts \bullet \frac{\Delta t}{2}) = \vec{v}_i(t) + \frac{mmts}{2}\Delta t \frac{\vec{F}_{non\_bond,i}(t)}{m_i} \quad (i = 1,...,N)$$

compute initial bonded forces $\{ \vec{F}_{bond,i}(t) \}$

for *inner* = 1 **to** *mmts*

$$\vec{v}_i(t + \frac{\Delta t}{2}) = \vec{v}_i(t) + \frac{1}{2}\Delta t \frac{\vec{F}_{bond,i}(t)}{m_i} \quad (i = 1,...,N)$$

Figure 4.1 2-level velocity-Verlet rRESPA MTS algorithm. (Fig. Cont'd)

$$\vec{r}_i(t+\Delta t) = \vec{r}_i(t) + \Delta t\,\vec{v}_i(t+\frac{\Delta t}{2}) + \frac{1}{2}\Delta t^2\left[\frac{\vec{F}_{bond,i}(t)}{m_i}\right] \quad (i = 1,...,N)$$

update bonded forces $\{\vec{F}_{bond,i}(t+\Delta t)\}$

$$\vec{v}_i(t+\Delta t) = \vec{v}_i(t+\frac{\Delta t}{2}) + \frac{1}{2}\Delta t\,\frac{\vec{F}_{bond,i}(t+\Delta t)}{m_i} \quad (i = 1,...,N)$$

update non-bonded forces $\{\vec{F}_{non\text{-}bond,i}(t + mmts \bullet \Delta t)\}$

$$\vec{v}_i(t + mmts \bullet \Delta t) = \vec{v}_i(t + mmts \bullet \frac{\Delta t}{2}) + \frac{mmts}{2}\Delta t\,\frac{\vec{F}_{non\text{-}bond,i}(t + mmts \bullet \Delta t)}{m_i} (i=1,...,N)$$

This algorithm has doubly-nested loops. The outer loop is used to compute the non-bonded potential and forces, while the inner loop computes the bonded potential and forces. The variable *nstp* is the total number of simulation steps and *mmts* is the number of inner loops per single outer loop. Typically we set *mmts* to ~10 within desired numerical accuracy. A typical value of Δt = 0.3fs was used for the inner loop and thus the time step for the outer loop is *mmts*•Δt= 3fs. Since the non-bonded interactions are computationally more intensive than the bonded interactions, evaluating the former less often reduces the computational time significantly.

To conserve the bond lengths, constrained dynamics based on the SHAKE or RATTLE [13] algorithm is widely used. These algorithms allow one to use a large Δt, but they are inherently sequential in nature and are difficult to parallelize. The penalty-function approach [20] we have adopted to enforce the bond constraints through V_{stretch} combined with the MTS algorithm, on the other hand, is very easy to parallelize.

4.2 Formulation of Fast Multipole Method (FMM)

The fast multipole method (FMM) developed by Greengard and Rokhlin [37] calculates electrostatic energies and forces for a collection of *N* charged particles with *O*(*N*) operations and with predictable error bounds, whereas the optimal Ewald method is an

$O(N^{3/2})$ algorithm [38]. The FMM enables atomistic simulations of realistic materials involving millions to billions of charged particles under various settings of boundary conditions [39]. A scalable and portable FMM code named FMMP for materials simulations on parallel computers using the Message Passing Interface (MPI) standard [40] was recently developed. The FMMP features the calculation of microscopic stress tensors, which is needed for pressure-controlled simulations and local stress analyses. It also supports various boundary conditions, including two- and three-dimensional periodic-boundary conditions, which are used for slab and bulk systems, respectively. The FMMP has been used in various materials simulations, including oxidation of aluminum nanoparticles [41] and sintering of titania nanoparticles [42] in which interatomic interaction is modeled with variable-charge potentials [43,44].

A collection of N particles with charges $\{q_i \mid i = 1,...,N\}$ at positions s; $\vec{a}_i = (a_i, \alpha_i, \beta_i)$ with $a_i = |\vec{a}_i|$ in polar coordinates was considered. Greengard and Rokhlin [37] proposed an $O(N)$ algorithm to compute Coulomb potentials for all particles, *i.e.*, $\sum_{j \neq i} q_j / |\vec{a}_i - \vec{a}_j| \ (i = 1,...,N)$. The direct calculations to evaluate Coulomb potentials at N particle positions requires $O(N^2)$ operations. In this section, we summarize equations that are necessary to understand the present implementation of the FMM.

The electrostatic potential at position $\vec{r}$ may be expressed as [45]

$$\sum_i \frac{q_i}{|\vec{r} - \vec{a}_i|} = \begin{cases} \sum_{l=0}^{\infty} \sum_{m=-l}^{l} M_{lm}(\vec{A}) L_{lm}(\vec{r}) & \text{for } r > A_{\max} \\ \sum_{l=0}^{\infty} \sum_{m=-l}^{l} L_{lm}(\vec{A}) M_{lm}(\vec{r}) & \text{for } r < A_{\min} \end{cases} \tag{4.1}$$

where

$$M_{lm}(\vec{x}) \equiv \frac{1}{(l+|m|)!} x^l P_{lm}(\cos\theta)\exp(-im\phi), \tag{4.2}$$

$$L_{lm}(\vec{x}) \equiv (l-|m|)!\, x^{-(l+1)} P_{lm}(\cos\theta)\exp(im\phi), \tag{4.3}$$

$$M_{lm}(\vec{A}) \equiv \sum_i q_i M_{lm}(\vec{a}_i), \tag{4.4}$$

$$L_{lm}(\vec{A}) \equiv \sum_i q_i L_{lm}(\vec{a}_i). \tag{4.5}$$

Here, $\vec{x} = (x,\theta,\phi)$ in polar coordinates, $\vec{A}$ denotes a collective set, $\{\vec{a}\}$, of the positions, $A_{\max(\min)}$ is the maximum (minimum) value of a_i, $\{M_{lm}(\vec{A})\}$ are the multipole moments [9] of the charges, and $\{L_{lm}(\vec{A})\}$ are the local Taylor expansion coefficients [45] of the Coulomb field at the origin. $P_{lm}(x)$ in Eqs. (4.2) and (4.3) are defined with the associated Legendre polynomials $P_l^m(x) \equiv (1-x^2)^{m/2}\frac{d^m}{dx^m}P_l(x)$ $(l \geq m \geq 0)$ as

$$P_{lm}(x) = \begin{cases} (-1)^m P_l^m(x) & \text{for } m \geq 0 \\ P_l^{|m|}(x) & \text{for } m < 0 \end{cases}. \tag{4.6}$$

The following recursion formulas are useful to calculate $P_{lm}(x)$

$$\begin{aligned} &P_{mm}(x) = (-1)^m[(2m-1)(2m-3)\cdots 1](1-x^2)^{m/2} \quad \text{for } m \geq 1 \\ &P_{0\,0}(x) = 1 \\ &P_{m+1\,m}(x) = x(2m+1)P_{mm}(x) \quad \text{for } m \geq 0 \\ &(l-m)P_{lm}(x) = x(2l-1)P_{l-1,m}(x) - (l+m-1)P_{l-2,m}(x) \quad \text{for } m \geq 0 \\ &P_{l-m}(x) = (-1)^m P_{lm}(x) \quad \text{for } m \geq 1 \end{aligned}. \tag{4.7}$$

The following relations are used to speed up computations if values of charges are real numbers:

$$\begin{aligned} &M_{l-m} = (-1)^m M_{lm}^* \quad \text{for } m > 0 \\ &L_{l-m} = (-1)^m L_{lm}^* \quad \text{for } m > 0 \end{aligned} \tag{4.8}$$

where * denotes the complex conjugate.

The above recursive relations can be used to derive transformation operators between M_{lm} and L_{lm}:

$$\sum_{l}\sum_{m} M_{lm}(\vec{A})L_{lm}(\vec{r}) = \sum_{l}\sum_{m} M_{lm}(\vec{A}-\vec{b})L_{lm}(\vec{r}-\vec{b}) \tag{4.9}$$

with the condition of $|\vec{r}-\vec{b}|>|\vec{A}-\vec{b}|$,

$$\sum_{l}\sum_{m} M_{lm}(\vec{A}-\vec{b})L_{lm}(\vec{r}-\vec{b}) = \sum_{l}\sum_{m} L_{lm}(\vec{b}'-\vec{A})M_{lm}(\vec{b}'-\vec{r}) \tag{4.10}$$

with the condition of $|\vec{b}'-\vec{A}|>|\vec{b}'-\vec{r}|$, and

$$\sum_{l}\sum_{m} L_{lm}(\vec{b}'-\vec{A})M_{lm}(\vec{b}'-\vec{r}) = \sum_{l}\sum_{m} L_{lm}(\vec{c}-\vec{A})M_{lm}(\vec{c}-\vec{r}) \tag{4.11}$$

with the condition of $|\vec{c}-\vec{A}|>|\vec{c}-\vec{r}|$. In Eqs. (4.9)-(4.11), $M_{lm}(\vec{A}-\vec{b})$ and $L_{lm}(\vec{b}'-\vec{A})$ are short hand notations for $\sum_{i} q_i M_{lm}(\vec{a}_i-\vec{b})$ and $\sum_{i} q_i L_{lm}(\vec{b}'-\vec{a}_i)$, respectively.

The following three types of transformation operations are used in the code [45]:

$$M_{lm}(\vec{A}-\vec{b}) = \sum_{j=0}^{l}\sum_{k=-j}^{j} T^{\mathrm{MM}}_{l-j,m-k}(\vec{b})M_{jk}(\vec{A}) \quad \text{[multipole-to-multipole]} \tag{4.12}$$

$$L_{lm}(\vec{b}-\vec{A}) \sim \sum_{j=0}^{p}\sum_{k=-j}^{j} T^{\mathrm{ML}}_{j+l,k+m}(\vec{b})M_{jk}(\vec{A}) \quad (p = \text{maximum of } l) \text{ [multipole-to-local]} \tag{4.13}$$

$$L_{lm}(\vec{c}-\vec{A}) = \sum_{j=l}^{p}\sum_{k=-j}^{j} T^{\mathrm{LL}}_{j-l,k-m}(\vec{b}-\vec{c})L_{jk}(\vec{b}-\vec{A}) \quad (p = \text{ maximum of } l) \text{[local-to-local]} \tag{4.14}$$

with the operators,

$$T^{\mathrm{MM}}_{l-j,m-k}(\vec{b}) = M_{l-j,m-k}(-\vec{b}), \tag{4.15}$$

$$T^{\mathrm{ML}}_{j+l,k+m}(\vec{b}) = L_{j+l,k+m}(\vec{b}), \tag{4.16}$$

$$T^{\mathrm{LL}}_{j-l,k-m}(\vec{b}) = M_{j-l,k-m}(\vec{b}). \tag{4.17}$$

The FMM calculates Coulomb potentials of charged particles contained in a simulation box in five steps [37,38,45]:

(i) The simulation box with dimensions (h_x, h_y, h_z) is successively subdivided. Let l_{bot}^x, l_{bot}^y, and l_{bot}^z be the prescribed numbers of recursive subdivisions along the x, y, and z-axes. At the finest subdivision level $l_{\text{bot}} = \max(l_{\text{bot}}^x, l_{\text{bot}}^y, l_{\text{bot}}^z)$, the simulation box is decomposed into cells with dimensions $\left(h_x / 2^{l_{\text{bot}}^x}, h_y / 2^{l_{\text{bot}}^y}, h_z / 2^{l_{\text{bot}}^z}\right)$. The largest division is the simulation box itself at level 0. At level l, the simulation box is composed of cells with dimension $\left(h_x / \max(2^{l_{\text{bot}}^x + l - l_{\text{bot}}}, 1), h_y / \max(2^{l_{\text{bot}}^y + l - l_{\text{bot}}}, 1), h_z / \max(2^{l_{\text{bot}}^z + l - l_{\text{bot}}}, 1)\right)$.

(ii) Compute multipole moments of all the cells at the finest level of subdivision. Sweep up from the smallest cells to largest cell to obtain multipole moments of cells at all subdivision levels using the multipole-to-multipole transformation formula, Eq. (4.12).

(iii) Sweep down from the largest cell to cells at the next level of subdivision to obtain local expansion coefficients in the smallest cells: First, transform local expansion coefficients of larger cell to cells at the next level of subdivision using the local-to-local transformation formula for shifting the origin of a local expansion, Eq. (4.14). Second, add to these local expansion coefficients contribution from cells at the next level of subdivision, which have not been included and are well-separated from the cell being considered, using the multipole-to-local transformation formula, Eq. (4.13).

(iv) Once the preceding step has reached the finest subdivision level, evaluate the potential for each particle, Eq. (4.1), using the local expansion coefficients, $L_{lm}(\vec{A})$, of the smallest cell containing the particle.

(v) Add contributions from other charges in the same cell and the near neighbor cells by direct computations.

The various transformations and octree data structure are shown schematically in Figure 4.2.

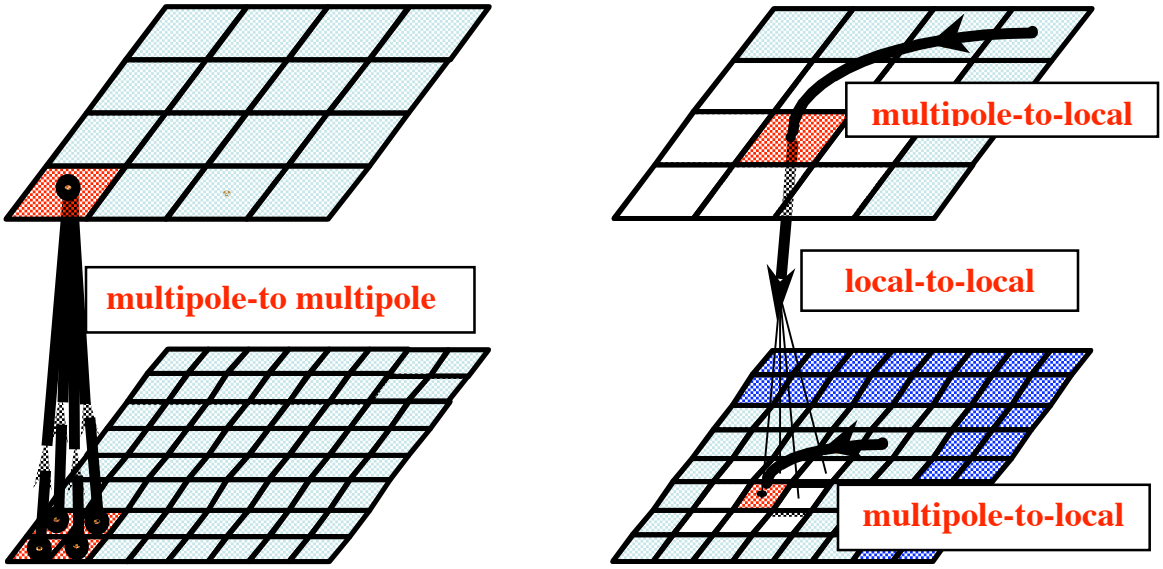

Figure 4.2. Schematic of octree data structure and different transformations, which are essential in FMM computation.

The FMMP performs steps (i)-(iv). The direct computations in step (v) should be performed separately in a subroutine supplied by the user. Such a separation in the FMM steps is appropriate since usual MD simulation codes have a linked list [46] of neighboring particles for fast computations of short-range forces and the linked list can be exploited for the direct calculations in step (v).

In FMMP, the most time-consuming part is the multipole-to-local transformation operation in step (iii). If terms only up to $|j+l| \le p$ in Eq. (4.13) are taken, the computation time becomes approximately a half with some loss of accuracy [45].

Forces acting on particles are calculated by differentiating the potential, using the following formulas:

$$\frac{\partial M_{lm}}{\partial x} = M_{lm}\left[\frac{lx}{x^2+y^2} + im\frac{y}{x^2+y^2}\right] M_{l-1,m}\frac{zx}{x^2+y^2}, \tag{4.18}$$

$$\frac{\partial M_{lm}}{\partial y} = M_{lm}\left[\frac{ly}{x^2+y^2} - im\frac{x}{x^2+y^2}\right] M_{l-1,m}\frac{yz}{x^2+y^2}, \tag{4.19}$$

$$\frac{\partial M_{lm}}{\partial z} = M_{l-1,m} \ . \tag{4.20}$$

A scalable and portable FMM code was recently implemented on parallel computers [26] using Message Passing Interface (MPI) standard. Scalability tests of the FMMP code have exhibited a parallel efficiency as high as 98% on 512 processors, for a system size of 512 million atoms, the results of which will be described in chapter 6.

CHAPTER 5
PARALLELIZATION METHODS

5.1 Parallel Macro-Molecular Dynamics (MMD) Algorithm

The MMD algorithm has been parallelized using spatial decomposition [5,25], in which the physical system is divided into P subsystems, $P_x \times P_y \times P_z$, in the x, y, and z directions, and each subsystem is assigned to a processor (see Figure 5.1). Each processor p thus stores arrays containing the positions $\vec{r}_i (i = 1,...,N_p)$, velocities $\vec{v}_i (i=1,...,N_p)$, and species α_i $(i = 1,...,N_p)$ of atoms, where N_p is the number of atoms residing in the pth subsystem.

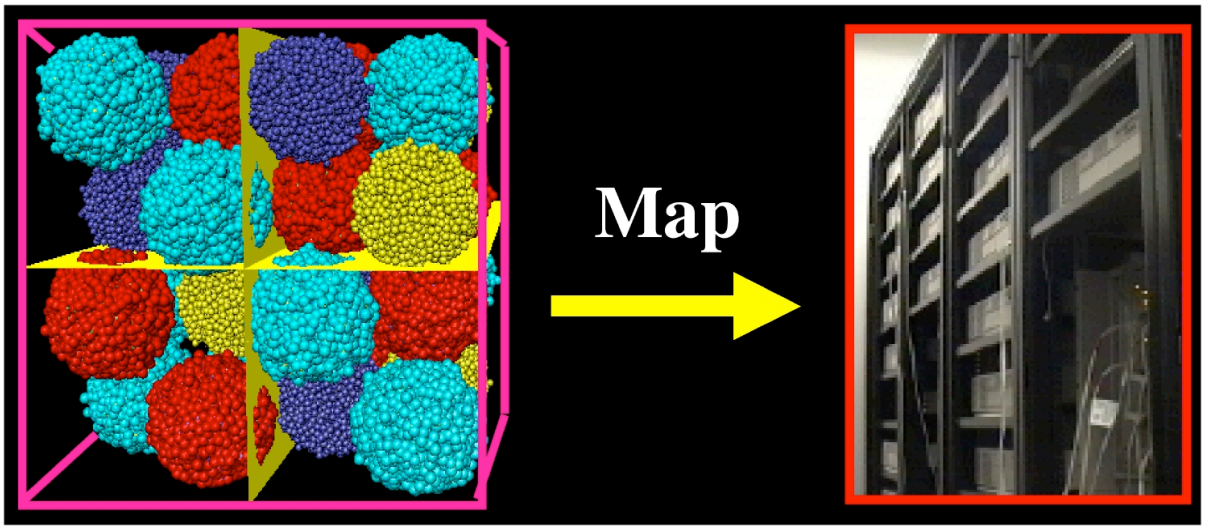

Figure 5.1 Spatial decomposition method, where each sub system is mapped to a processor.

When time-stepping procedure updates the atomic positions, some atom i in subsytem p may have moved out to a neighboring subsystem p'. This atom is 'migrated' to processor p', i.e., the information, $\vec{r}_i$, $\vec{v}_i$ and α_i, is removed from processor p and sent to processor p', where it is appended to position, velocity and species arrays.

To calculate bonded and van der Waals interactions as well as the near-field contribution to Coulombic interaction in processor p, all atoms, which are in the neighboring processors but are close to p, are 'cached'. Namely the positions and species of these atoms must be received

from the neighboring processors, and then appended to local arrays in p. More precisely, a skin of thickness r_c (the cutoff radius of the van der Waals potential function, $v_{vdw}(r)$) around each processor p is defined, and cache the information about the atoms within this skin from neighboring processors to p. The set of atoms in this 'secondary' skin is denoted by $S_s(p)$, as $B_s = \{i \in S_s(p)\}$. To minimize the communication time for the bonded-force calculations, another skin, 'primary' skin $S_p(p)$, of thickness r_p ($<r_c$) is defined, where r_p is at least twice the longest bond length. The set of atoms in this skin is denoted by $B_p = \{i \in S_p(p)\}$. Figure 5.2 schematically shows these two skins.

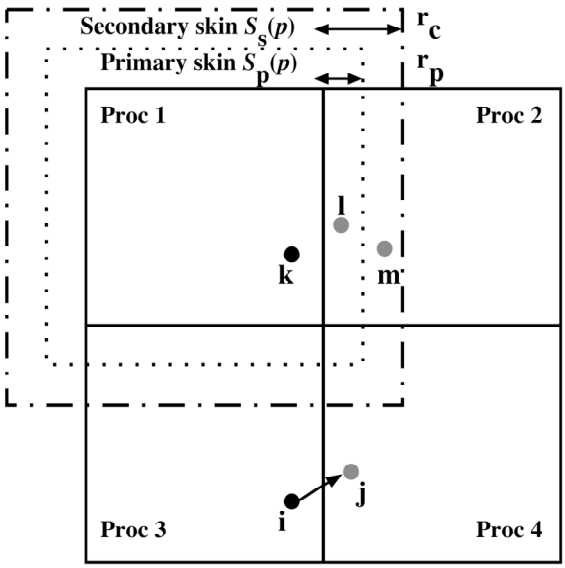

Figure 5.2. Schematic of spatial decomposition in MMD algorithm. The figure shows the primary skin $S_p(p)$ of thickness r_p and the secondary skin $S_s(p)$ of thickness r_c for processor 1. Atom i has migrated from processor 3 to processor 4 due to time-stepping procedure.

Using the cached information, the force computation is partitioned as follows. For bonded and non-bonded pair interactions, the processor that has the atom computes the force on each atom. For van der Waals interaction, any atomic pair km crossing the boundary between processors 1 and 2 (see Figure 5.2), processor 1 computes half of $v_{vdw}(r_{km})$ and force on atom k and processor 2 computes the other half of $v_{vdw}(r_{km})$ and force on atom m. The same strategy is

used for near-field contribution to Coulomb interaction and for any bonded pair *uv* crossing a processor boundary (see Figure 5.3).

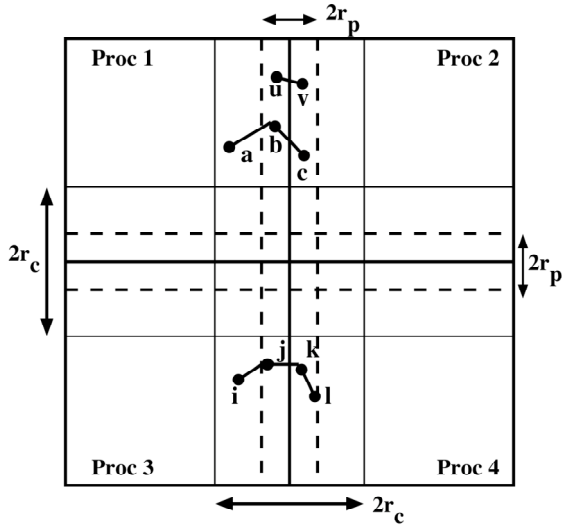

Figure 5.3. Primary and secondary skins in MMD algorithm Atomic pair *uv* crosses the boundary between processors 1 and 2. Processor 1 computes the forces on all atoms *a*, *b*, and *c* in triplet *abc* and send the computed force on non-resident atom *c* to processor 2. Processor 3 computes the forces on all atoms *i*, *j*, *k*, and *l* in quadruplet *ijkl* and send the computed forces on non-resident atoms *k* and *l* to processor 4.

For triplet (*ijk*) and quadruplet (*ijkl*) interactions in Eqs. (2.6) and (2.7), the processor which has atom *j* (the index of the outermost summation) computes the forces on all the atoms in the *n*-tuple. For triplet *abc*, the processor *p* that has atom *b* computes the energy and the forces on atoms *a* and *c* in addition to the force on atom *b*. If atom *c* is non-resident (see Figure 5.3), the computed force on *c* is sent to the processor it belongs. For quadruplet *ijkl*, the processor that has the atom *j* computes the energy and forces on atoms *i*, *k* and *l*, in addition to the force on *j*. Computed forces on non-resident atoms are sent to the processors they belong to. Thus only triplet and quadruplet interactions involve communication of forces across processor boundaries.

Single-program multiple-data (SPMD) programming paradigm is used, in which all the processors execute the same program on different datasets. Figure 5.4 shows the parallelized MMD algorithm for each processor.

cache secondary skin atoms B_s
compute non–bonded forces $\{ \vec{F}_{non\_bond,i}(t) \}$
for *outer* = 1 **to** *nstp* / *mmts*

$$\vec{v}_i(t + mmts \bullet \frac{\Delta t}{2}) = \vec{v}_i(t) + \frac{mmts}{2}\Delta t \frac{\vec{F}_{non\_bond,i}(t)}{m_i} \quad (i = 1,...,N)$$

cache primary skin atoms B_p
compute initial bonded forces $\{ \vec{F}_{bond,i}(t) \}$
for *inner* = 1 **to** *mmts*

$$\vec{v}_i(t + \frac{\Delta t}{2}) = \vec{v}_i(t) + \frac{1}{2}\Delta t \frac{\vec{F}_{bond,i}(t)}{m_i} \quad (i = 1,...,N)$$

$$\vec{r}_i(t + \Delta t) = \vec{r}_i(t) + \Delta t \vec{v}_i(t + \frac{\Delta t}{2}) + \frac{1}{2}\Delta t^2 \left[\frac{\vec{F}_{bond,i}(t)}{m_i} \right] \quad (i = 1,...,N)$$

migrate moved-out atoms
cache primary skin atoms B_p
update bonded forces $\{ \vec{F}_{bond,i}(t + \Delta t) \}$

$$\vec{v}_i(t + \Delta t) = \vec{v}_i(t + \frac{\Delta t}{2}) + \frac{1}{2}\Delta t \frac{\vec{F}_{bond,i}(t + \Delta t)}{m_i} \quad (i = 1,...,N)$$

cache secondary skin atoms B_s
update non-bonded forces $\{ \vec{F}_{non\_bond,i}(t + mmts \bullet \Delta t) \}$

$$\vec{v}_i(t + mmts \bullet \Delta t) = \vec{v}_i(t + mmts \bullet \frac{\Delta t}{2}) + \frac{mmts}{2}\Delta t \frac{\vec{F}_{non\_bond,i}(t + mmts \bullet \Delta t)}{m_i} (i=1,...,N)$$

Figure 5.4. Parallelised 2-level velocity-Verlet rRESPA MTS algorithm.

With the spatial decomposition, the computation scales as N/P while communication scales as $(N/P)^{2/3}$. For deeper tree levels, $l \geq \log_2[\max(P_x, P_y, P_z)]$, in the parallel FMM (FMMP) algorithm, the calculation of the multipoles is local to each processor so that the computation scales as N/P. For lower levels, however, the number of FMM cells, 8^l, becomes smaller than the number of processors. Consequently many processors duplicate the same computation, and this computation overhead scales as log P. For a coarse-grained decomposition ($N >> P$), this log P overhead becomes insignificant.

5.1.1 Selected Subroutines in MMD

The following summarizes important subroutines in the MMD.

mmmd: The driver program with the Multiple time stepping algorithm, updates the positions of atoms using velocity Verlet algorithm.

setup: This subroutine sets up all the potential parameters for the MMD program, and assigns random velocities to atoms in the initialization run.

accel: This subroutine sets up the linked cells, makes lists for bonded and non-bonded interactions, and computes the short-ranged bonded and non-bonded interactions between all atoms.

FMMP: This subroutine calculates the far-field contribution to the Coulomb interaction, which is described in detail in section 5.2.

bacopy: This subroutine exchanges copies of the relevant information (species, coordinates and velocities of atoms), in both the primary and secondary skins of each processor, with the neighboring processors.

bamove: This subroutine sends and receives the coordinates, velocities and species of atoms that have moved-out and moved-in from neighboring processors. The atoms which have moved-out are 'removed' from the processor and the atoms that are moved-in are 'added' to the processor.

5.1.2 Parameters, Input and Output of MMD

The size of the system (*N)* and the number of compute nodes in each direction (`nx`, `ny`, `nz`) should be set by the user prior to the starting of the simulation. The scalar node-index `myid`, and the vector node-index (`myx`, `myy`, `myz`) are calculated. Each processor will read the set of species, coordinates of the atoms that are assigned to it. At the beginning of the program, a set of control parameters is read and they are explained below:

`input = 0` if the simulation is starting from the initial configuration (read only species and coordinates)

`= 1` if simulation restarts from previous configuration (read species, coordinates and velocities)

`treq` : The initial temperature to be given to the system if `Input = 0`.

`dt`: Time step of the innermost loop.

`nstp`: Number of total time steps of the simulation.

`mmts`: Number of inner steps to be performed for every outer step.

The flow of information in MMD code is shown schematically in Figure 5.5. Depending on the value of `input`, the velocities are either read from the input file along with coordinates or random velocities scaled to temperature `treq` are assigned to the atoms in the subroutine *setup*. The Figure 5.5 shows the 2-level multiple-time scale scheme with non-bonded and bonded interactions calculated in the outer and inner loops respectively. The subroutines, which are responsible for each of the processes, are marked in italics and red. The updating of positions and velocities is according to the time stepping algorithm described in Eqs. (2.11) and (2.12). The lists described in chapter 3 are made every outer step, and the same are used for all the inner steps. At the end of each outer step, the *bamove* subroutine is called which migrates the moved out atoms and reconfigures the system based on new positions.

5.2 Parallel Fast Multipole Method (FMMP) Algorithm

The FMMP is applicable to systems of both isolated clusters of charges and periodically repeating simulation boxes. For the periodic boundary conditions, the reduced-cell multipole method (RCMM) [48] was previously implemented. In the RCMM, a well-separated image of a

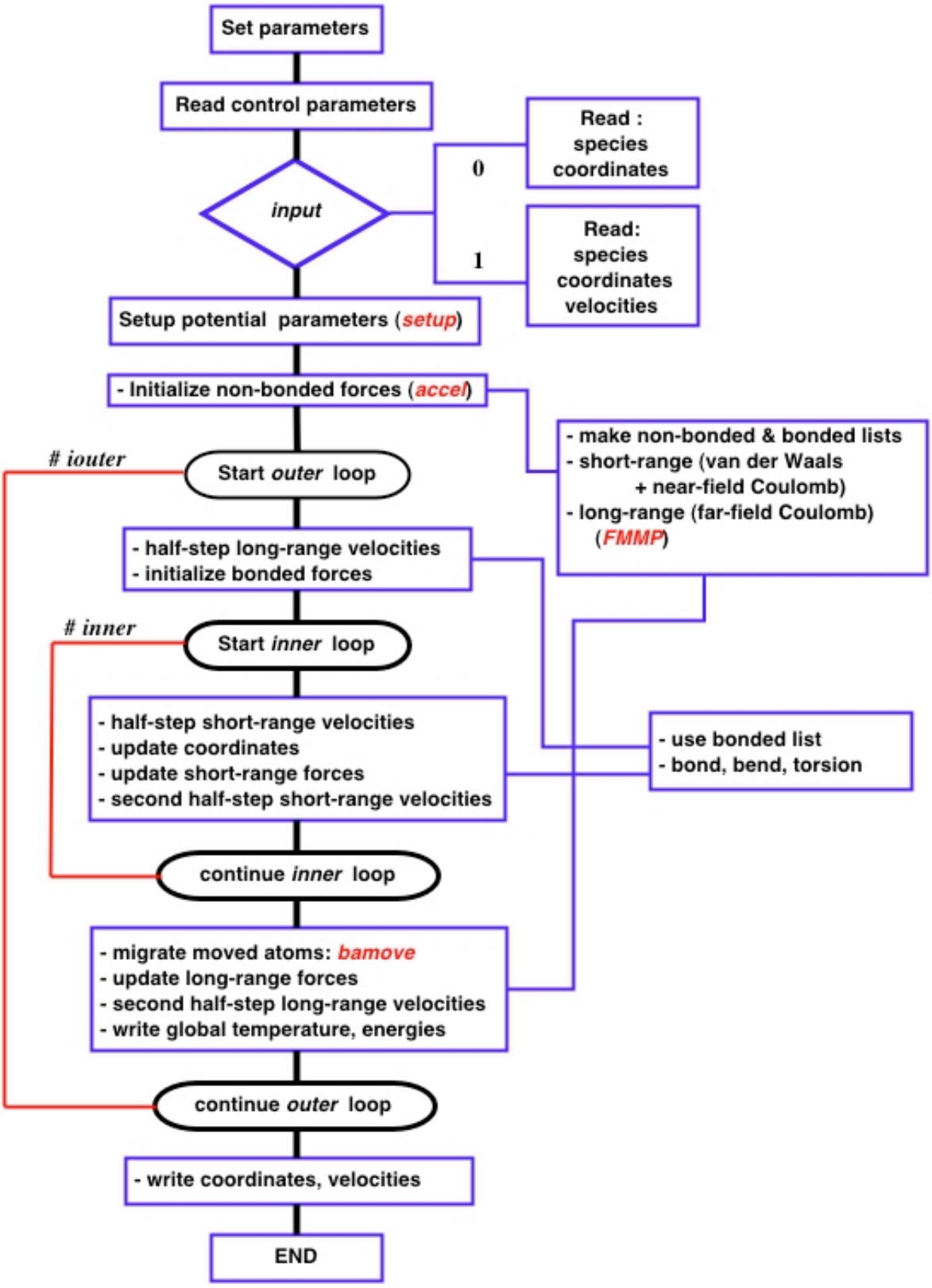

Figure 5.5 Flow chart of the information flow in MMD program.

simulation box is represented by a small number of charged points; those positions are chosen at random and magnitudes of the charges are determined to reproduce lower-order multipoles of the original simulation box [49]. The Ewald summation technique is used to calculate Coulomb potential energy between such periodically repeating, reduced-cell charges. However, the potential energy calculated with the RCMM was found to be sensitive to the positions of representative reduced-cell particles. In the FMMP, a different method that resembles the macroscopic-multipole method [48,49] is used.

It is known that Coulomb potential energy for increasing numbers of repeating simulation boxes becomes a convergent series if aggregates of the boxes form, e.g., a spherical shape [22,48-51]. The Coulomb potential energy of the aggregates includes a term arising from finite dipole moment of the box. In the usual Ewald summation method, such aggregates of boxes are regarded as embedded in a metallic environment to neglect the energy term due to the dipole moment of the box. When a periodic boundary condition is chosen in the FMMP, the code calculates the Coulomb interactions with the original simulation box wrapped by a lattice of boxes forming a sphere with radius ~7 boxes. Since the original MD box is represented by a finite number of multipoles, the computation time increases by small amounts by taking such image boxes in the FMMP. To realize the metallic environment as in the case of the Ewald summation, the user has to add dipole correction terms to the FMMP values. The potential energy in the three-dimensional Ewald summation method is then reproduced as

$$V_{\mathrm{Ewald}}^{3\mathrm{D}} = V_{\mathrm{FMM}} + \Delta V_{\mathrm{dipole}}^{3\mathrm{D}}, \tag{5.1}$$

where

$$\Delta V_{\mathrm{dipole}}^{3\mathrm{D}} = \frac{2\pi}{3V} |\vec{P}|^2 \tag{5.2}$$

with the volume of the simulation box Ω and the dipole of the simulation box $\vec{P} = \sum_{i=1}^{N} q_i \vec{a}_i$. Corresponding correction term to the force on particle i is $-\partial V_{\text{dipole}}^{\text{3D}} / \partial \vec{a}_i$. The correction term to the total stress tensor is

$$\ddot{\Xi}_{\text{Ewald}}^{\text{3D}} = \ddot{\Xi}_{\text{FMM}} + \frac{4\pi}{3\Omega} \vec{P}\vec{P}^{\text{T}} + \Omega V_{\text{dipole}}^{\text{3D}} \ddot{1} \ . \tag{5.3}$$

Recently there have been increasing interests in simulating surface and interfacial systems with a slab geometry. Conventional 3D Ewald summation formula cannot be used directly because of no periodicity in one of the three directions. A two-dimensional Ewald summation technique has been developed, giving explicit formulas for the Coulomb energy $V_{\text{Ewald}}^{\text{2D}}$ for the slab geometry [52-54]. Yeh and Berkowitz [56] performed a detailed numerical comparison between $V_{\text{Ewald}}^{\text{3D}}$ and $V_{\text{Ewald}}^{\text{2D}}$ and found a n efficient way of calculating $V_{\text{Ewald}}^{\text{2D}}$ using $V_{\text{Ewald}}^{\text{3D}}$.

Let us assume that the (neutral) slab system is periodic in x and y directions, and that an empty space is inserted with its length in z direction greater than or equal to max(L_x, L_y). The 2D Ewald result for the Coulomb energy of the system is reproduced as [56]

$$V_{\text{Ewald}}^{\text{2D}} = V_{\text{Ewald}}^{\text{3D}} + \frac{2\pi}{V} P_z^2 = V_{\text{FMM}} + \Omega V_{\text{dipole}}^{\text{3D}} + \frac{2\pi}{V} P_z^2 \tag{5.4}$$

using the FMM value V_{FMM} in the 3D periodic-boundary condition.

In the FMMP, the simulation box is spatially decomposed into `nx` × `ny` × `nz` subsystems, which are assigned to the same number of compute nodes. Here, `nx`, `ny`, and `nz` should be either 1 or a power of 2, *i.e.,* 1, 2, 4, 8, …, with constraints $\log_2(\texttt{nx}) \le l_{\texttt{bot}}^{x}$, $\log_2(\texttt{ny}) \le l_{\texttt{bot}}^{y}$, and $\log_2(\texttt{nz}) \le l_{\texttt{bot}}^{z}$. Each node has a scalar index `myid` in the range [0, `nx` × `ny` × `nz` - 1]. The vector node-index (`myx`, `myy`, `myz`) satisfies the relation `myid` = `myx` × `ny`

□ nz + myy □ nz + myz, and it specifies the (*x, y, z*) position of the node in the logical 3D array of compute nodes. In the single-node case (nx = ny = nz = 1), for example, myid = 0 with (myx, myy, myz)=(0, 0, 0); in the two-node case (nx = 2, ny = nz = 1), two nodes are myid = 0 with (myx, myy, myz) = (0, 0, 0) and myid = 1 with (myx, myy, myz) = (1, 0, 0). Each node is given a number of particles Ntotn in the corresponding subsystem, charges {chg(i); i = 1, ..., Ntotn}, normalized positions {sr(1-3,i); i = 1, ..., Ntotn}, the simulation-box tensor $\overleftrightarrow{h}$ =(hx(1-3), hy(1-3), hz(1-3)). The normalized position of particle *i*, sr(1-3,i), in a node with the vector node-index (myx, myy, myz) satisfies the following inequalities: myx/nx < sr(1,i) < (myx+1)/nx, myy/ny < sr(2,i) < (myy+1)/ny, and myz/nz< sr(3,i) < (myz+1)/nz.

Parameters that control the accuracy of the FMMP are the maximum subdivision levels (lbotx, lboty, lbotz), the maximum order of multipoles iptop, and the minimum separation between well-separated cells normalized by the simulation box iWS = 1 or 2 (see step (iii) in the FMM algorithm). Multipole and local expansion data at subdivision level $l \leq$ lglim are global and stored in all nodes, whereas the data at level $l >$ lglim are stored only in the corresponding node and transferred to different nodes through MPI_Send and MPI_Receive calls when they are required. The value of lglim is lfit (for iWS = 1) or lfit+1 (for iWS = 2) with lfit = $\log_2$[max(nx, ny, nz)]. The lfit corresponds to the level at which the cell assumes the maximum size in a single node.

5.2.1 Selected Subroutines in FMMP

The following summarizes important subroutines in the FMMP.

FMPmain: All the FMM calculations are performed in this subroutine.

Mpsetup: This subroutine sets up cell indices for upward and downward passes in the FMM.

MPup: This subroutine calculates the multipole moments for cells starting from the smallest cell toward the simulation box (level 0).

MPdown: This subroutine transforms the local expansion coefficients of a larger cell to cells at the next level of subdivision using the local-to-local transformation formula for shifting the origin of a local expansion. Then, the subroutine adds to these local expansion coefficients the contribution from cells at the next level of subdivision, which have not been included and are well-separated from the cell being considered, using the multipole-to-local transformation formula.

GetPFS: This subroutine calculates potential, force, stress tensor fields felt by particles using the local expansion coefficients for the smallest cell containing the particle.

MDwrap3: This subroutine calculates the contribution of periodic images of simulation boxes in 3 dimensions.

5.2.2 Parameters, Input and Output of FMMP

The user should set the following CPP macros in the include file, fmmp_dim.h, to determine maximum array sizes used in FMMP:

`DEBUG`: Switch to write (= 1) debugging information.

`INCLUDE_STRESS`: Switch to include (= 1) arrays for stress calculation.

`PFLM_PRECISION`: Switch for single (1) or double (2) precision representation of multipoles.

`Nsize_`: maximum number of particles in each node.

`iptop_`: maximum order of multipoles.

`msize_`: maximum size for the cell index. `msize` > $1+8+8^2+...+8^{\texttt{lbot}}$ with `lbot` = max(`lbotx`, `lboty`, `lbotz`).

`ibsize_`: maximum buffer size for data transfer between the nodes.

It should be noted that including stress calculations significantly increases required memory size of the program. The fmmp.F should be preprocessed by CPP before compilation with Fortran77.

The FMM calculations are performed in subroutine FMPmain. The user has to set the number of compute nodes in each direction (`nx`, `ny`, `nz`), the scalar node-index `myid`, and the vector node-index (`myx`, `myy`, `myz`) in a common block `node_vec`. Dimensions of the simulation box $\vec{\vec{h}}$ = (`hx(1-3)`, `hy(1-3)`, `hz(1-3)`) should be set in a common block `MDbox`. Charge and normalized coordinates of particles (`chg(i)`, `sr(1-3,i)`) and the total number of particles in each node `Ntotn` are set in a common block `node_ptcl`.

The following control parameters for FMMP are set in the common block `mpdat1`.

`lbotx(y,z)`: the finest level of subdivision in x (y,z) direction; 2 ≤ `lbotx(y,z)` ≤ 16.

`iWS=1` or 2: the minimum separation distance in units of the simulation box between well-separated cells.

`iTR=0` or 1: If `iTR=1`, the multipole-to-local translation is truncated.

`iPBC=0` or 1: If one sets `iPBC=0`, free b:oundary condition is assumed. It should be noted that the user is allowed to set different numbers for subdivision levels (`lbotx, lboty, lbotz`) only if `iPBC = 0`. If one sets `iPBC = 1`, the simulation box is wrapped by a 3-dimensional lattice of boxes.

iST=0 or 1: If the CPP macros `INCLUDE_STRESS = 1` and `iST = 1`, microscopic stress-tensors are also calculated.

After the subroutine FMPmain terminates, the potential field (`PF(i)`), negative of the force field (`FF(1-3,i)`), and stress tensor field (`ST(1-6,i)`) felt by each particle *i* are stored in the common block `node_result`. Definitions of PF, FF, and SF are

$$\texttt{PF(i)} = \sum_{j \neq i} \frac{\texttt{chg(j)}}{\left|\vec{a}_i - \vec{a}_j\right|}, \tag{5.5}$$

$$\texttt{FF(1-3,i)} = \frac{\partial \texttt{PF(i)}}{\partial \vec{a}_i} \tag{5.6}$$

with the first index denoting x = "1", y = "2", z = "3" components, and

$$\texttt{SF(1-6,i)} = \sum_{j \neq i} \texttt{chg(j)} \frac{\left(\vec{a}_i - \vec{a}_j\right)\left(\vec{a}_i - \vec{a}_j\right)^{\mathrm{T}}}{\left|\vec{a}_i - \vec{a}_j\right|^3}, \tag{5.7}$$

with the first index denoting xx = "1", yy = "2", zz = "3", yz = "4", xz = "5", xy = "6" components. The measured computation time for each FMM step explained in Sec. 2 is stored in the common block `fmm_time`:

`t_setup`: setup time.

`t_comm`: total communication time.

`t_up`: time for upward pass.

`t_down`: time for downward pass.

`t_wrap`: time to take care PBC.

`t_pfs`: time to compute PF, FF, and ST.

Common block names used in the FMMP are `node_vec`, `MDbox`, `node_ptcl`, `node_result`, `mpdat1`, `mpdat2`, `pseudoC`, `PBC3`, `fmm_time`, and `fmmparity`. Those names should not be used in the main program.

The present code, FMMP, is scalable and portable implementation of the FMM. It can be used not only in materials simulations but also in various fields of simulations including plasmas

and astronomical objects interacting through $1/r$ potential. The user, however, should be cautious in its applications to highly ordered systems such as a crystalline lattice of charged points. Depending on the lattice structures, values of the potential energy may vary considerably as a function of `iptop` in the case of periodic boundary conditions. This results from the fact that many of the lower-order multipoles are zero or nearly zero and only some higher-order multipoles assume non-zero values in such regular lattices. The user needs to set a rather higher order of multipoles for such systems.

CHAPTER 6
SCALABILITY TEST RESULTS AND CONCLUSIONS

The results of scalability tests performed on the two codes Fast Multipole Method Parallel (FMMP) and Macro-Molecular Dynamics (MMD) are discussed in this chapter. The portability of the two codes is demonstrated by performing the scalability tests on different platforms. The parallel FMMP results are described first and then using FMMP as a part of the MMD code, describe the scalability results of MMD code.

6.1 Parallel Fast Multipole Method Results

A sample driver program, *fmmptest.F* has been prepared to illustrate the usage of the FMMP code and test its accuracy. The executable *fmmptest* is created with the make command. The *fmmptest* is then run using *n* nodes under MPI environment [40] as `mpirun –np` *n* `fmmptest`.

Results of two tests are reported in this section. The first test estimates the accuracy of the FMM by comparing its results with those with the Ewald summation and direct calculations. In the second test, scalability of the code is demonstrated by running it on different numbers of nodes and study the effect of system size on the scalability of the code.

For the accuracy test, thousand (N = 1000) particles with charges either +1 or –1 are distributed uniformly in a cubic box (side length, L) such that the overall charge neutrality is maintained. The test is performed on a Linux PC cluster (8 nodes, 100BaseTX-connected PentiumIII/600MHz). The simulation box is divided into P = 8 nodes; (`nx`, `ny`, `nz`)=(`2`, `2`, `2`). The values of some of the parameters are set to the following values: `iWS = 1`, `iTR = 0`, `lbotx = lboty = lbotz = 3`, `iST = 0`, and `iptop = 5`. The results of the accuracy test are shown below in Table 6.1.

Table 6.1Results of accuracy tests

P	lbot	iPBC	N	t_setup	t_comm.	t_up	T_down	t_wrap	t_pfs	Rel_error
8	3	0	1000	1.45e-3	5.38e-2	5.57e-2	5.68e-1	3e-6	3.71e-3	0.1077e-3
8	3	1	1000	1.42e-3	5.49e-2	5.77e-2	1.15e0	1.47e-1	1.12	0.4799e-3

The accuracy test was performed for both `iPBC = 0` (free boundary condition) and `iPBC = 1` (the simulation box is wrapped by a 3-dimensional lattice of boxes) and, for both cases, the program compares the FMM results of the potential fields felt by the particles (that include nearest neighbor contribution) with the direct calculation results. Averaged values of the relative errors are written to the standard output with timing data. In the present case (`iptop = 5`), the averaged relative errors of the potential fields are on the order of 0.01%. When more accurate results are required, the user should set a large value for `iptop` and/or set `iWS = 2`. Averaged relative errors of the force (stress) field are 2-3 (5-10) times larger than that of the potential field.

The second set of the tests examines the scalability of the FMMP on massively parallel computers. The calculations were performed on an IBM SP3, called *HABU*, at the U.S. Naval Oceanographic Office (NAVO) Major Shared Resource Center. The HABU is configured with 375 MHz Power3 CPUs and has 334 nodes with 4 CPUs and 4 GB of memory per node, total of 1,336 processors. It runs AIX 4.3 operating system and IBM XL Fortran 7.1 compiler. The same calculations were repeated on another parallel computer, IBM SP4, called *Marcellus* at NAVO. The Marcellus is configured with 1.3 GHz Power4 CPUs and has 148 nodes with 8 CPUs and 8 GB of memory per node, total of 1,184 processors. The specifications of both the machines are given in Table 6.2. The following parameters were chosen: `iptop = 5`, `iWS = 1`, `iTR = 0`, `iST = 0`, and `iPBC = 1`.

Table 6.2 Comparison of IBM SP3 and IBM SP4.

	IBM SP3	IBM SP4
Processor	375 MHz Power3	1.3GHz Power 4
# of processors	1336	1184
Peak speed	2.0 Tflops	6.1 Tflops
L2 Cache	8MB	1.4MB/2-processor-chip
Memory	4GB/4-processor-node	8GB/8-processor-board
Latency	21.7 sec	21 sec
Band width	75MB/proc (bidirectional)	90MB/proc (bidirectional)

The compute nodes for this test were chosen as (`nx, ny, nz`) = (1, 1, 1), (2, 2, 2), (4, 4, 4), and (8, 8, 8) with the number of particles treated in each node, *N*/*P*, fixed. The *N*/*P* value is varied as 100,000, 500,000, and 1,000,000. The values of `lbotx = lboty = lbotz` are also varied from `4` to `7`. The results are shown in Tables 6.3, 6.4 and 6.5 and in Figures 6.1-6.3.

Table 6.3 Scalability test results for N/P = 100,000

P	Lbot	N	t_setup	t_comm.	t_up	t_down	t_wrap	t_pfs	Total	effeciency
1	4	1e+05	0.573	0.012	0.155	8707	0.138	1.1	89.7	1.00
8	5	8e+05	0.586	0.0514	0.195	88.4	0.139	1.12	90.4	0.99
64	6	6.4e+06	0.589	0.0725	0.218	88.5	0.138	1.12	90.6	0.99
512	7	5.12e+07	0.591	0.208	0.353	88.8	0.140	1.12	91.3	0.98

Table 6.4 Scalability test results for N/P = 500,000

P	lbot	N	t_setup	t_comm.	t_up	t_down	t_wrap	t_pfs	Total	effeciency
1	4	5e+05	2.84	0.0121	0.155	87.6	0.138	5.52	96.3	1.00
8	5	4e+06	2.92	0.0481	0.192	88.3	0.138	5.58	97.1	0.99
64	6	3.2e+07	2.92	0.0873	0.231	88.5	0.138	5.59	97.5	0.98
512	7	2.56e+08	2.92	0.222	0.367	88.8	0.139	5.60	98.0	0.98

Table 6.5 Scalability test results for N/P = 1,000,000

P	lbot	N	t_setup	T_comm.	T_up	t_down	t_wrap	t_pfs	Total	effeciency
1	4	1e+06	5.8	0.0121	0.156	87.6	0.138	11.1	105.0	1.00
8	5	8e+06	5.83	0.0595	0.204	88.3	0.139	11.2	106.0	0.99
64	6	6.4e+07	5.82	0.177	0.321	88.4	0.138	11.1	106.0	0.99
512	7	5.12e+08	5.84	0.332	0.47	88.9	0.140	11.4	107.0	0.98

Figure 6.1a shows the results on the IBM SP3. The total execution time of the FMMP code with a fixed value of N/P is nearly constant, when the total number of particles, N, is increased by varying the number of compute nodes, P. The parallel efficiency, which is defined as the ratio of time on a single processor to time on P processors, for the case of largest system size *viz.*, P = 512 and N/P = 1,000,000 is as high as 0.98. The communication time is only a small fraction of the total execution time. Figure 6.1b shows the results on the IBM SP4, where the execution time is almost 2.5 times faster than that on IBM SP3 but only a slight improvement in terms of communication time.

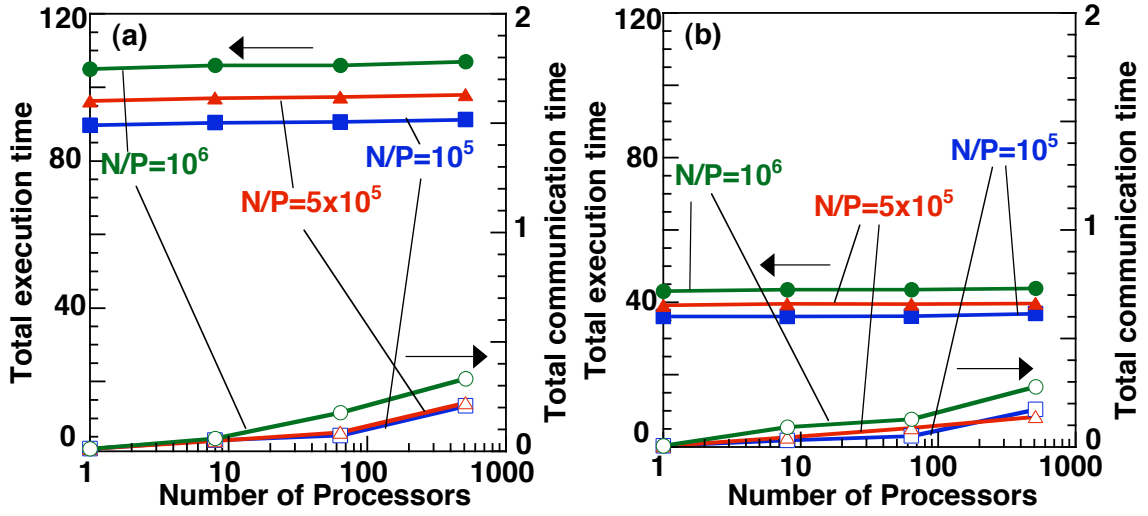

Figure 6.1 Timings as function of number of processors. (a) Total execution (solid symbols) and communication (open symbols) times are plotted for the number of particles per processor, N/P = 10^5 (squares), 5□10^5 (triangles), and 10^6 (circles) on IBM SP3. (b) The same as (a) on IBM SP4.

It can also be observed in Figure 6.1 that the total execution time is not proportional to N/P for a fixed value of P, but is of the form $a + b(N/P)$, where $a >> b$. This is understood by analyzing the individual timing for each subroutine in the FMMP code. The results of the timing results for individual subroutines are shown in Figure 6.2.

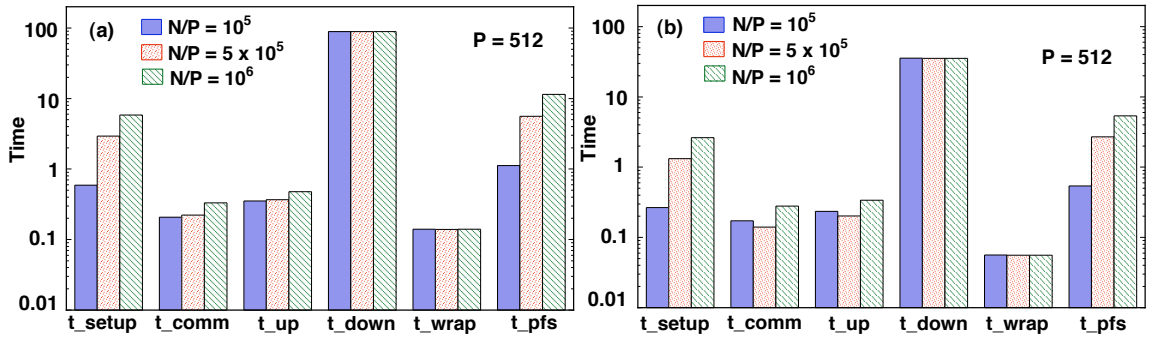

Figure 6.2. Partial timing data for different subroutines for $P = 512$ for $N/P = 10^5$, 5×10^5, and 10^6 on (a) SP3 and (b) SP4.

Figure 6.2 shows the timing data for subroutines for three different cases of N/P = 100,000, 500,000, and 1,000,000 with P = 512. The dominant computation time is for the subroutine *MPdown*, which is constant and does not change with N/P. The next two dominant computations are from the subroutines *GetPFS* and *MPsetup*, which increase with N/P. This is because the number of particles contained in a cell increases in proportion to N/P. The other subroutines *MPup* and *MDwrap3* give very small contributions to the total time. Combination of the increasing times for *GetPFS* and *MPsetup* and the constant time for *MPdown* explains the observed behavior in the total execution time as a function of N/P.

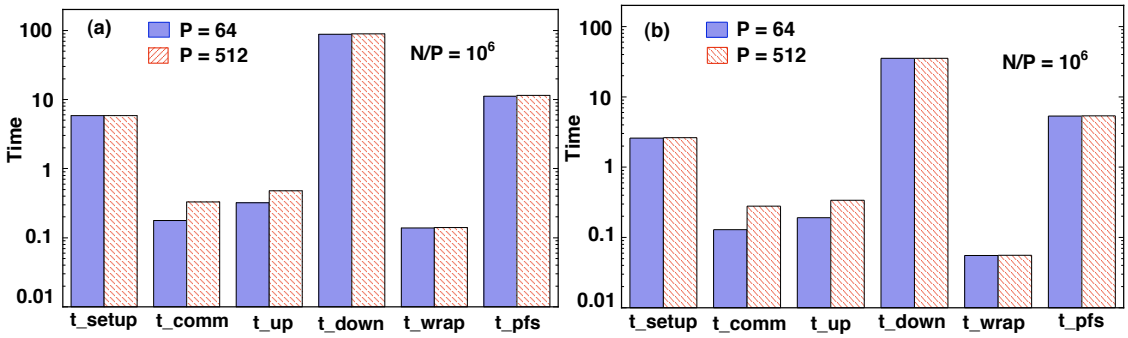

Figure 6.3. Partial timing data for different subroutines for $N/P=10^6$ on P= 64 and 512 processors on (a) SP3 and (b) SP4.

Figure 6.3 compares the timing data for various subroutines with N/P=1,000,000 between P = 64 and P = 512. The timings for all the dominant subroutines, *MPdown*, *GetPFS*, and

MPsetup are nearly the same between the two cases of *P*. Only the communication time and the timing for *MPup* increase slightly as a function of *P*.

6.2 Parallel Macro-Molecular Dynamics Results

In this section the scalability tests performed on MMD code, which incorporates the FMMP code, are described. To study the effect of granularity N/P = 5280, 10560, and 21120 was used. The number of processors *P* varies from 1 to 1024 and the maximum number of atoms is 21.6 million. (The system is self assembled monolayers of alkane-thiolates on gold surface.) The tests are performed on three multi Tera-flop machines, IBM SP4 *Marcellus* at the Naval Oceanographic Office (NAVO), Intel Xeon-based Linux cluster *SuperMike* at Louisiana State University (LSU) and Compaq Alpha Server SC45 *Emerald* at army Engineer Research and Development Center (ERDC). A comparison of three systems is given in Table 6.5.

Table 6.5 Comparison of Linux cluster, IBM SP4 and Compaq SC45.

	Linux cluster	IBM SP4	Compaq SC45
Processor	1.8GHz Intel Xeon	1.3GHz Power 4	1.0GHz Alpha 21264
# of processors	1024	1184	512
Peak speed	3.7 Tflops	6.1 Tflops	1.0 Tflops
L2 Cache	512KB/processor	1.4MB/2-processor-chip	8MB/processor
Memory	2GB/2-processor-node	8GB/8-processor-board	4GB/4-processor-node
Latency	9 □sec	21 □sec	5 □sec
Band width	250MB/proc (bidirectional)	90MB/proc (bidirectional)	125MB/proc (bidirectional)

In *SuperMike* 512 dual-processor nodes are connected by Myrinet network, which consists of 24 switch units and 64 nodes are connected within the same switch. *Marcellus* on the other hand uses a proprietary network and IBM's Colony II switch for communication. In *Emerald*, 128 four-processor nodes are connected by a 64-port, single-rail Quadrics high-speed interconnect switch.

Figure 6.4 shows the scalability of the MMD algorithm as a function of the number of processors on IBM SP4. The number of atoms per processor (N/P) is also varied to study the effect of granularity. The parallel efficiency for P processors, which is defined as the ratio of time on a single processor to time on P processors, increases with N/P. Up to 512 processors the parallel efficiency is 80%, 83% and 87% for N/P values of 5280, 10560 and 21120, respectively. The observed degradation in parallel efficiency for smaller N/P is due to larger communication to computation ratio. The computation time scales as N/P and the communication time scales as $(N/P)^{2/3}$, and hence the communication to computation ratio, $(N/P)^{-1/3}$, is a decreasing function of N/P. It can also be seen from Figure 6.4 that the communication time is much smaller compared to the total execution time in all the three cases.

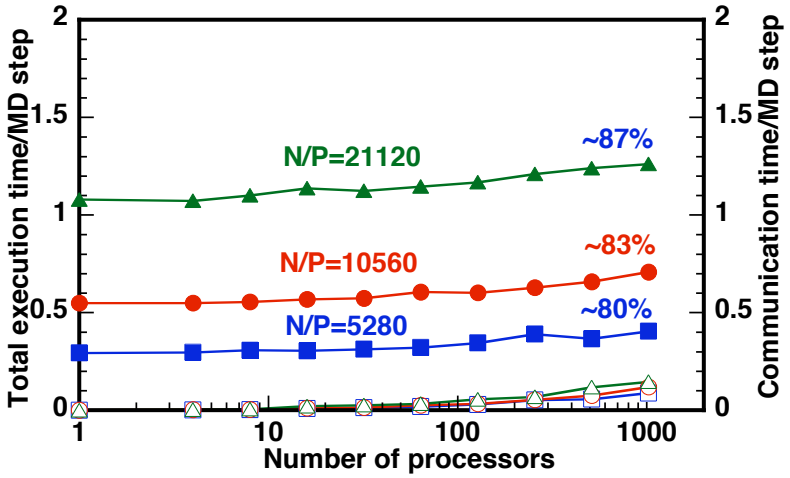

Figure 6.4. Total execution time (solid symbols) and communication time (open symbols) of MMD as a function of P for different work loads: N/P = 5280 (squares), 10560 (circles) and 21120 (triangles).

On IBM SP4, two processors share L2 cache on a chip, four chips on a board share memory, and four boards constitute a physical unit called LPAR (Logical PARtition). Thus each LPAR has 32 processors. To study the performance degradation due to the sharing of the L2 cache, a set of tools *bindUtils* were used that enables us to use only one processor per chip, thereby reducing the congestion caused by the sharing of the cache.

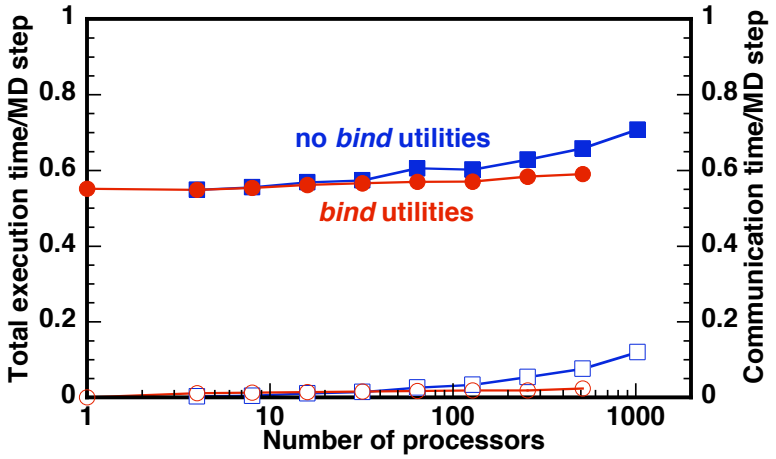

Figure 6.5. Total execution time (squares) and communication time (circles) of MMD as a function of P with (open symbols) and without (solid symbols) *bindUtils* for $N/P = 10560$.

In Figure 6.5, solid squares show the execution time when two processors share L2 cache and open squares show the execution time when only one processor per chip is used. Sharing of L2 cache increases the execution time as much as 10%. The small jump in the execution time from 32 processors to 64 processors occurs due to communication bottleneck across two LPARs.

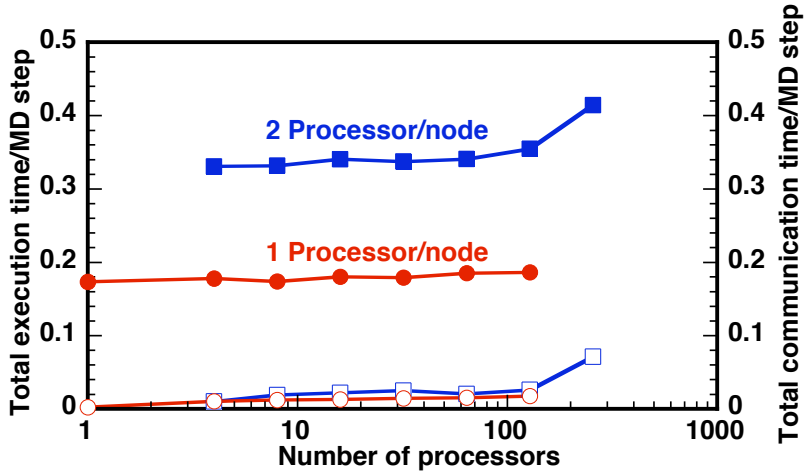

Figure 6.6. Total execution time (solid symbols) and communication time (open symbols) of the HIO-MD algorithm as a function of P with $N/P = 10560$ on the Linux cluster at LSU-BCVC, using one processor per node (circles and red) and two processors per node (squares and blue).

This result suggests a detrimental effect of sharing main memory by the two processors within a node on the Linux cluster also. To quantify this effect, Figure 6.6 compares the results of two sets of HIO-MD benchmark tests on the 256-processor Linux cluster at LSU-BCVC. In

the first set, only one processor per dual-processor node is used, whereas in the second set, both processors are used. The performance degradation due to sharing main memory in the execution time is nearly constant (~ 90%) from 2 to 128 processors.Figure 6.6 also shows performance degradation in the communication time, because the shared memory is used for communication between two processors in the same node. The resulting congestion in memory and/or internal bus degrades the performance.

On Linux cluster, the effect of choice of compiler on performance of MMD algorithm also has been studied. For this purpose two compilers, PGI 4.0 and Intel 6.0 are used and the result is shown in Figure 6.7. The Intel compiler makes better use of architecture of the cluster, including the Streaming SIMD Extensions 2 (SSE2) that augments the floating-point functional unit to deliver two results per cycle in the ideal case. Accordingly it can be seen that the total execution time with Intel compiler is smaller than that with PGI compiler, but no significant difference in communication time. In both cases, one processor per node is used up to 512 processors, and 2 processors per node for 1024 processor case.

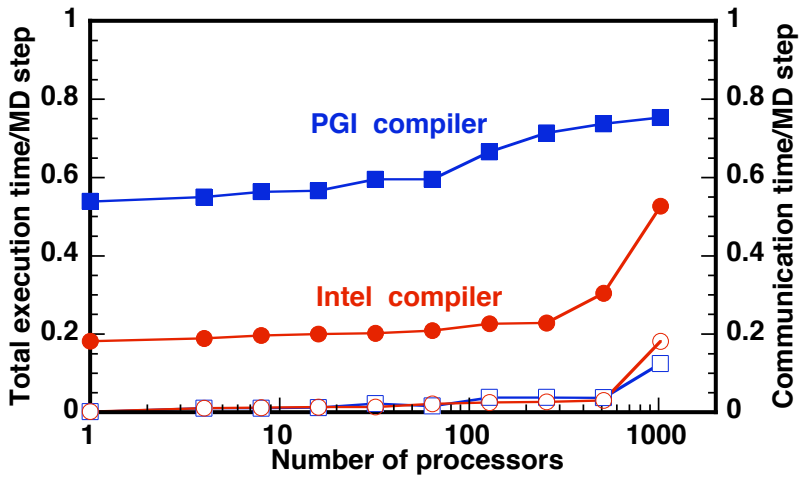

Figure 6.7. Total execution time (squares) and communication time (circles) of MMD as a function of P for different compilers, Intel (open symbols) and PGI (solid symbols) for N/P = 10560.

In Figure 6.8, the scalability of MMD on IBM SP4 and Linux cluster is compared. In both cases, the best performance of the code is used, i.e., with *bindUtils* tools on IBM SP4 and with Intel 6.0 compiler on Linux cluster. Linux cluster *SuperMike* is significantly faster than the IBM SP4 *Marcellus* for this application. No significant difference in the communication time is observed. On Linux cluster, the execution time increases only slightly with the number of processors, but a sudden jump in the execution time for 1024 processors is observed. This is due to memory-sharing effect as two processors on a motherboard share the main memory.

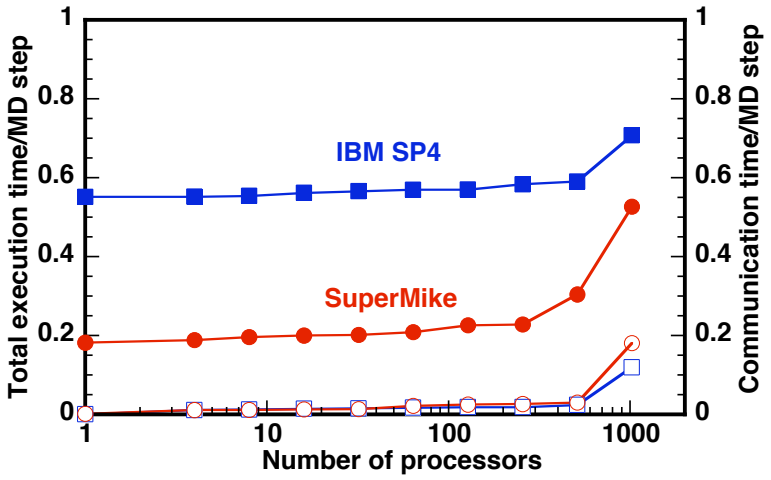

Figure 6.8. Total execution time (squares) and communication time (circles) of MMD as a function of P, at a constant granularity of N/P = 10560, for two different machines: Linux cluster *SuperMike* (open) and IBM SP4 *Marcellus* (solid).

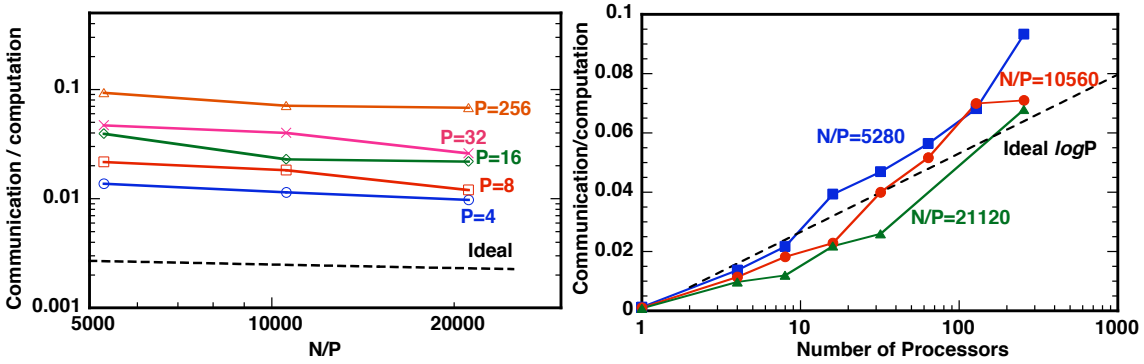

Figure 6.9: (a) Communication/computation ratio as a function of N/P for P = 4 - 256 on the AlphaServer *Emerald*. The dashed line shows the ideal $(N/P)^{-1/3}$ scaling. (b) Communication/computation ratio as a function of P for N/P = 5280 (squares), 10560 (circles), and 21120 (triangles) on *Emerald*. The dashed line shows the log$P$ dependence.

As explained in the previous section, the communication overhead of the parallel HIO-MD algorithm consists of $O((N/P)^{2/3})$ and $O(\log P)$ terms. In Figure 6.9(a), the measured ratio between communication and computation times as a function of N/P closely follows the ideal $(N/P)^{-1/3}$ dependence on the AlphaServer *Emerald*. To highlight the $\log P$ bahavior, the measured communication/computation ratio is also plotted as a function of P in Figure 6.9(b).

6.3 Conclusions

A scalable parallel code macro-molecular dynamics (MMD) algorithm has been developed for the simulation of complex organic systems. The MMD code is based on a space-time multiresolution approach combining the fast multipole method (FMM) and multiple time-scale method as well as dynamic management of distributed linked cells, neighbor lists, and atomic n-tuples. The MMD algorithm is used for the simulation of realistic systems and various scalability tests are reported in this thesis. For the computation of time consuming long-range Coulomb interaction potential of charged atoms, a highly scalable code is also developed based on FMM. The octree data structure is used in FMM, and by performing constant number of operations at each octree level, FMM algorithm scales as $O(N)$.

The scalability and portability of both FMM and MMD algorithms has been demonstrated on various tera-flop machines including 1024-processor Intel Xeon-based Linux cluster,1184-processor IBM SP4 and 512 Compaq Alpha Server . The effects of memory- and cache-sharing on the MMD algorithm are also studied and the effect is shown significant. The MMD code has the parallel efficiency of 0.87 on IBM SP4 and includes a scalable version of FMM. Separate scalability tests on FMM show the parallel efficiency of 0.98 for 512 million charged particles on 512 IBM SP3 processors. The timing results on IBM SP3 are also compared with those on IBM SP4 for FMM case. Such a scalable parallel macro-molecular simulation

algorithm is expected to play an important role in the design of hybrid quantum-device/biological-cell systems and ‘virtual-cell’ technologies.

REFERENCES

1. F. F. Abraham, *Portrait of a crack: rapid fracture mechanics using parallel molecular dynamics*, IEEE Comput. Sci. Eng. 4(2) (1997) 66-77.

2. T. C. Germann, P. S. Lomdahl, *Recent advances in large-scale atomistic materials simulations*, IEEE Comput. Sci. Eng. 1(2) (1999) 10-11.

3. P. Vashishta, R. K. Kalia, A. Nakano, *Large-scale atomistic simulation of dynamic fracture*, IEEE Comput. Sci. Eng. 1(5) (1999) 56-65.

4. A. Nakano, R. K. Kalia, P. Vashishta, T. J. Campbell, S. Ogata, F. Shimojo, S. Saini, *Scalable atomistic simulation algorithms for materials research*, in Proc. Supercomputing 2001 (ACM, New York, NY, 2001); Scientific Programming 10 (2002) 263-270.

5. D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, Cambridge Univ. Press, Cambridge, 1995.

6. A. Nakano, T. J. Campbell, *An adaptive curvilinear-coordinate approach to dynamic load balancing of parallel multi-resolution molecular dynamics*, Parallel Computing, 23 (1997) 1461-1478.

7. A. Nakano, *Multiresolution load balancing in curved space: the wavelet representation*, Concurrency: Practice and Experience 11 (1999) 343-353.

8. Y. Duan, P. A. Kollman, *Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution*, Science 282 (1998) 740-744.

9. P. E. Marszalek, H. Lu, H. Li, M. Carrion-Vazquez, A. F. Oberhauser, K. Schulten, J. M. Fernandez, *Mechanical unfolding intermediates in titin modules*, Nature 402 (1999) 100-103.

10. J. -E. Shea, C. L. Brooks, III, From folding theories to folding proteins: a review and assessment of simulation studies of protein folding and unfolding, Annual Rev. Phys. Chem. 52 (2001) 499-535.

11. M. Karplus, J. A. McCammon, Molecular dynamics simulations of macromolecules: a perspective, Nature Struct. Biol. 9 (2002) 646-652.

12. G. N. Tew, D. Liu, B. Chen, R. J. Doerksen, J. Kaplan, P. J. Carroll, M. L. Klein, W. F. DeGrado, *De novo design of biomimetic antimicrobial polymers*, Proc. Nat'l. Acad. Sci. USA 99 (2002) 5110-5114.

13. G. Ciccotti, J. P. Ryckaert, *Molecular dynamics simulations of rigid molecules*, Comput. Phys. Rep. 4 (1986) 345-392.

14. B. B. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, M. Karplus, *CHARMM: A program for macromolecular energy, minimization and dynamics calculations*, J. Comp. Chem. 4 (1983) 187-217.

15. W. F. van Gunsteren, H. J. C. Berendsen, *GROMOS: GROningen MOLecular Simulation software*, Technical Report, Laboratory of Physical Chemistry, University of Groningen, 1988.

16. S. Plimpton, *Fast Parallel algorithms for short range molecular dynamics*, J. Comp. Phys. 117 (1995) 1-19.

17. J. C. Phillips, G. Zheng, S. Kumar, L. V. Kale, *NAMD: Biomolecular simulation on thousands of processors*, in Proc. Supercomputing 2002 (IEEE, Los Alamitos, CA, 2002).

18. R. A. Mcmillan, C. D. Pavola, J. Howard, S.L. Chan, N.J. Zaluzec, J.D. Trent, *Ordered nanoparticle arrays formed on engineered chaperonin protein templates*, Nature Mater. 1 (2002) 247-252.

19. M. S. P. Sansom, I. H. Shrivastava, K. M. Renatuga, G. R. Smith, *Simulations of ion channels—watching ions and water move*, Trends in Biochem. Sci. 25 (2000) 368-374.

20. Y. Komeiji, M. Haraguchi, U. Nagashima, *Parallel molecular dynamics simulation of a protein*, Parallel Computing 27 (2001) 977-987.

21. G. J. Martyna, M. E. Tuckerman, D. J. Tobias, M. L. Klein, *Explicit reversible integrators for extended systems dynamics*, Mol. Phys. 87 (1996) 1117-1157.

22. M. J. Allen, D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press, Oxford, 1987.

23. L. H. Yang, E. D. Brooks III, J. Belak, *A linked-cell domain decomposition method for molecular dynamics simulations on a scalable multiprocessor*, Scientific Programming 2 (1993)153-162.

24. L. Greengard, V. Rokhlin, *A fast algorithm for particle simulations*, J. Comp. Phys. 73 (1987) 346-365.

25. A. Nakano, R. K. Kalia, P. Vashishta, *Multiresolution molecular dynamics algorithm for realistic materials modeling on parallel computers*, Comp. Phys. Comm. 83 (1994) 197-214.

26. S. Ogata, T. J. Campbell, R. K. Kalia, A. Nakano, P. Vashishta, S. Vemparala, *Scalable and portable implementation of the fast multipile method on parallel computers*, Parallel Computing, submitted.

27. T. Darden, D. York, L. Pedersen, *Particle mesh ewald, an Nlog(N) method for Ewald sums in large systems*, J. Chem. Phys. 98 (1993) 10089-10092.

28. H. Kikuchi, R. K. Kalia, A. Nakano, P. Vashishta, F. Shimojo, S. Saini, *Scalability of a low-cost multi-Teraflop Linux cluster for high-end classical atomistic and quantum mechanical simulations*, in Proc. 2003 Int'l Parallel and Distributed Processing Symposium (IEEE/ACM, Nice, France, 2003).

29. A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary, HPL - a portable implementation of the high-performance Linpack benchmark for distributed-memory computers, http://www.netlib.org/benchmark/hpl.

30. J. J. Dongarra. *Performance of various computers using standard linear equations software*, Technical Report, Univ. of Tennessee, August 2002, http://www.netlib.org/benchmark/performance.ps.

31. W. Cornell, P. Cieplak, C. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, P. A. Kollman, *A second generation force field for the simulation of proteins, nucleic acids, and organic molecules*, J. Am. Chem. Soc. 117 (1995) 5179-5197.

32. G. D. Smith, R. L. Jaffe, D. Y. Yoon, *A force field for simulations of 1,2-dimethoxyethane and poly(oxyethylene) based upon ab initio electronic structure calculations on model molecules*, J. Phys. Chem. 97 (1993) 12752-12759.

33. R. W. Hockney, J. W. Eastwood, *Computer Simulation Using Particles*, Adam Hilger, New York, 1981.

34. L. Verlet, *Computer experiments on classical fluids I. Thermodynamical properties of Lennard Jones molecules*, Phys. Rev. 159 (1967) 98-103.

35. M.Tuckerman, B.J. Berne, G.J. Martyna, *Reversible multiple time scale molecular dynamics*, J. Chem. Phys. 97 (1992) 1990-2001.

36. M.E. Tuckerman, D.A. Yarne, S.O. Samuelson, A.L. Hughes, G.J. Martyna, *Exploiting multiple levels of parallelism in molecular dynamics based calculations via modern techniques and software paradigms on distributed memory computers*, Comp. Phys. Comm. 128(2000) 333-376.

37. L. Greengard and V. Rokhlin, J. Comput. Phys. 60 (1985) 187; L. Greengard, *The Rapid Evolution of Potential Fields in Particle Systems* (MIT, Boston, 1987).

38. E.L. Pollock and J. Glosli, *Comments on PPPM, FMM and the ewald method for large periodic and coloumbic systems*, Comp. Phys. Commun. 95 (1996) 93-110.

39. A. Nakano et al., *Atomistic simulations of nano structured materials using parallel multi resolution algorithms*, IEEE Computational Science and Engineering 5 (1998) 68-78.

40. W. Gropp, E. Lusk, A. Skkjellum, *Using MPI Portable Parallel Programming with the Message-Passing Interface* (MIT, Boston, 1994).

41. T.J. Campbell, R.K. Kalia, A. Nakano, P. Vashishta, S. Ogata, and S. Rodgers, *Dynamics of oxidation of aluminium nanoclusters using variable-charge molecular-dynamics simulations on parallel computers*, Phys. Rev. Lett. 82 (1999) 4866.

42. S. Ogata, H. Iyetomi, K. Tsuruta, F. Shimojo, A. Nakano, R.K. Kalia, and P. Vashishta, *Role of atomistic charge transfer on sintering of TiO2 nano particles: variable-charge molecular dynamics*, J. Appl. Phys. 88 (2000) 6011.

43. F. H. Streitz and J. W. Mintmire, *Electrostatic potentials for metal-oxide surfaces and interfaces*, Phys. Rev. B 50 (1994) 11996.

44. S. Ogata, H. Iyetomi, K. Tsuruta, F. Shimojo, R.K. Kalia, A. Nakano, and P. Vashishta, *Variable-charge interatomic potentials for the molecular-dynamics simulations of TiO2*, J. Appl. Phys. 86 (1999) 3036.

45. C.A. White and M. Head-Gordon, *Derivation and efficient implementation of the fast multipole method*, J. Chem. Phys. 101 (1994) 6593.

48. H. Ding, N. Karasawa and W.A. Goddard III, *Atomic level simulations of million particles: the cell multipole method for coulomb and London interactions*, Chem. Phys. Lett. 196 (1992) 6.

49. A.Y. Toukmaji and J.A. Board Jr., *Ewald sum techniques in perspective: summary*, Comp. Phys. Comm. 95 (1996) 73.

50. C.G. Lambert, T.A. Darden, and J.A. Board Jr., *A multipole based algorithm for efficient calculation of forces and potentials in macroscopic periodic assemblies of particles*, J. Comp. Phys. 126 (1996) 274.

51. S.W. De Leeuw, J.W. Perram, and E.R. Smith, *Simulation of electrostatic systems in periodic boundary conditions.1. lattice sums and dielectric constants*, Proc. R. Soc. Lond. A373 (1980) 27.

52. M.W. Deem, J.M. Newsam, and S.K. Sinha, *The h=0 term in coulomb sums by the Ewald transformation*, J. Phys. Chem. 94 (1990) 8356.

53. D.E. Parry, *The electrostatic potential in the surface region of an ionic crystal*, Surf. Sci. 49 (1975) 433.

54. D.M. Heyer, M. Barber, and J.H. Clarke, J. Chem. Soc., Faraday Trans. II 73 (1977) 1485.

55. S.W. de Leeuw and J.W. Perram, *Electrostatic lattice sums for semi-infinite lattices*, Mol. Phys. 37 (1979) 1313.

56. I.-C. Yeh and M.L. Berkowitz, *Ewald summation for systems with slab geometry*, J. Chem. Phys. 111 (1999) 3155.

VITA

Satyavani (Vani) Vemparala was born on June 17, 1974, in Varangaon, Maharashtra, India to Subramanya Sarma and Padmavathy Vemparala. In 1996, Vani attained her master of science degree, equivalent to a bachelor's degree in United States, from the University of Hyderabad, India. In 1998, she attained her Master's degree in Solid State Technology from Indian Institute of Technology, Kharagpur, India.

In 1998, Vani entered the dual degree doctoral program in the Department of Physics and Astronomy at Louisiana State University in Baton Rouge, Louisiana. She was awarded a Graduate School Enhancement award in 1999. In 2000, she enrolled in the systems science master's program in the Department of Computer Science. She expects to graduate with both the degrees in the Fall of 2003.