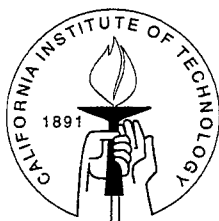


MEGA-MOLECULAR DYNAMICS ON HIGHLY PARALLEL COMPUTERS: METHODS AND APPLICATIONS

Thesis by

Kian-Tat Lim

In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy



California Institute of Technology

Pasadena, California

1995

(Submitted April 27, 1995)

Copyright © 1995
Kian-Tat Lim
All Rights Reserved

ACKNOWLEDGEMENTS

Thanks to all the people who have helped me through my (long) graduate career, including the following:

- Original CMM developers: Naoki Karasawa (Chapters 3, 4, and 8), Hong-Qiang Ding (Chapter 3).
- Collaborators: Michael Belmares (Chapters 10 and 11), Sharon Brunett (Chapters 6, 7, and 9), Guanghua Gao (Chapter 11), Mihail Iotov, Richard McClurg (Chapter 9), Changmoon Park, Vaidehi (Chapter 12).
- Friends and colleagues: Erik Bierwagen, Ken Brameld, Terry Coley, Alan Mathiowetz, Rick Muller, Charles Musgrave, Adel Naylor, Jason Perry, Kevin Plaxco.
- Replacing me and doing a better job, too: Darryl Willick.
- Friends who kept me sane and made living in Los Angeles enjoyable: Anne McCoy, Matt and Donna McKeever, Gigi Nickas, Richard and Tonja Dixon.
- Others who kept me going: Drs. Stern, Kelly, and Butler.
- Roommates who put up with me: Chris Gorman, Matt Fraser, Danny Koh.
- Mentors: Siddharth Dasgupta, Steve Taylor, Bill and Yvonne Goddard.

Most importantly, to my parents, Drs. Nyok Kheng and Teck Kah Lim, who have always been behind me.

ABSTRACT

Large-scale systems of thousands and millions of atoms are of great interest in many areas of chemistry, biochemistry, and materials science. Atomic-level simulations of such systems can provide increased accuracy and especially enhanced insight and understanding when compared with either smaller-scale model calculations or grossly-averaged macroscopic models.

Megamolecular simulations require large amounts of memory and computation, far more than can be provided by the typical scientific workstation. These resources can be most cost-effectively provided at this time by scalable massively parallel computers.

This thesis presents a large-scale, parallel, distributed-memory, general-purpose molecular dynamics code. The most time-consuming portion of the calculation, the computation of the nonbonded forces, is handled by the Cell Multipole Method, which was developed to overcome the speed and accuracy limitations of standard techniques for handling long-range power-law interactions in large molecular systems. Versions of the code for the KSR-1 and Intel Delta and Paragon parallel supercomputers are described, and performance, accuracy, and scalability results are given.

The applications section begins with a discussion of computational experiments leading to a prescription for choosing the value of the free time-scale parameter in Nosé-Hoover constant-volume, constant-temperature (NVT) canonical dynamics. This is followed by several applications of the above megamolecular dynamics codes to interesting chemical applications in the areas of argon cluster structure, polymer structure, surface tension of water drops, diffusion of gases through polymers, and viral structure.

TABLE OF CONTENTS

Chapter 1. Introduction	1
-------------------------------	---

PART I — THEORY

Chapter 2. Molecular Mechanics and Dynamics	7
Chapter 3. Cell Multipole Method.....	13
Chapter 4. Reduced Cell Multipole Method	24

PART II — DESIGN AND IMPLEMENTATION

Chapter 5. Implementation on KSR	29
Chapter 6. Implementation on Message Passing Architectures	51
Chapter 7. Performance	61

PART III — APPLICATIONS

Chapter 8. TVN Dynamics	76
Chapter 9. Argon Clusters.....	87
Chapter 10. Surface Tension of Liquid H ₂ O and H ₂ O on PTFE	93
Chapter 11. Diffusion of Gases through Polymers	104
Chapter 12. Viruses	114

LIST OF FIGURES

Figure 3-1. Two-dimensional representation of octree decomposition.....	16
Figure 3-2. Neighbors and PNCs for a level 3 cell.	20
Figure 5-1. Atom data structure.....	32
Figure 5-2. Cell data structure.....	33
Figure 7-1. Scaling vs. number of CPUs for farfield computation.....	69
Figure 7-2. Scaling vs. number of CPUs for nearfield/integration steps.	70
Figure 7-3. Scaling vs. number of atoms for farfield computation.	71
Figure 7-4. Scaling vs. number of atoms for nearfield/integration steps.	72
Figure 7-5. Scaling vs. $n_{atoms}^{-1/3}$ for farfield computation.	73
Figure 8-1. KE distribution for multi-chain PE crystal for varying τ_s	79
Figure 8-2. KE distribution for methane cluster system for varying τ_s	80
Figure 8-3. KE distribution for fcc argon with small τ_s	80
Figure 8-4. KE distribution for fcc argon with intermediate τ_s	81
Figure 8-5. KE distribution for fcc argon with large τ_s	81
Figure 8-6. KE vs. time for multi-chain PE crystal with small τ_s	82
Figure 8-7. KE vs. time for multi-chain PE crystal with large τ_s	82
Figure 8-8. KE vs. time for methane cluster with small τ_s	83
Figure 8-9. KE vs. time for methane cluster with large τ_s	83
Figure 8-10. KE vs. time for fcc argon with small τ_s	84
Figure 8-11. KE vs. time for fcc argon with intermediate τ_s	84
Figure 8-12. KE vs. time for fcc argon with large τ_s	85
Figure 9-1. $h=2$ section of log(structure factor) for initial argon cluster.	89

Figure 9-2. $h=2$ section of $\log(\text{structure factor})$ for final argon cluster	90
Figure 9-3. Comparison of potential energies for argon clusters.	91
Figure 10-1. Experimental PTFE lattice geometry.....	96
Figure 10-2. End view of 2611 chain PTFE system after dynamics.....	97
Figure 10-3. Enlargement of 2611 chain PTFE system after dynamics.	98
Figure 10-4. Water drop on CF_3 surface of PTFE.....	100
Figure 10-5. Water drop on CF_2 surface of PTFE.....	101
Figure 11-1. Structure of amorphous PE.....	106
Figure 11-2. Voids in PE unit cell and initial CO_2 position.....	107
Figure 11-3. Path of CO_2 diffusion.....	109
Figure 11-4. Path of He diffusion.	110
Figure 11-5. Mean square deviation curve for CO_2 in PE.	111
Figure 11-6. Mean square deviation curve for He in PE.....	111

LIST OF TABLES

Table 7-1. Accuracy of energy and force calculations.....	62
Table 7-2. Scalability for KSR code.	73
Table 11-1. Experimental and simulated diffusion constants.....	112

Chapter 1. Introduction

Large-scale systems of thousands and millions of atoms are of great interest in many areas of chemistry, biochemistry, and materials science. Atomistic-level simulations of such systems can provide increased accuracy when compared with either smaller-scale model calculations or grossly-averaged macroscopic models. The ability to analyze such simulations at the atomic level can lead to greater insight into structure/function relationships, effects of chemical modification, and in general the underlying physical basis of the system's behavior. Two key examples of large-scale systems include simulations of polymers and viruses.

Polymer molecular weights are typically in the range of millions, requiring at least hundreds of thousands of atoms to properly model a single polymer chain, let alone several chains at a time. Use of a shorter chain may lead to unphysical end effects; use of an infinite chain via periodic boundary conditions may ignore such effects, or artificially limit the achievable chain conformations. In particular, amorphous or partially-crystalline assemblies, typical of industrial polymers, are not readily describable using a single chain per unit cell. The study of the organization and structures of these systems and of their mechanical and thermodynamic properties thus will require models with on the order of 1 million atoms per unit cell.

The starburst dendrimer class of polymers [1] leads to a monodisperse collection of large molecules having the same topology but each with different packing of the branches and leaves. The limits of growth for these polymers depends critically upon how closely the branches and leaves can pack. For the most interesting case, the PAMAM dendrimers, this may require

molecular dynamics studies of systems ranging from 0.25 to 0.5 million atoms.

The smallest important viruses, the picornaviruses (responsible for polio, the common cold, and hoof-and-mouth disease) [2] are composed of protein coats of about 0.5 million atoms and a nucleic acid genome of about the same size. The exterior of the protein coat has approximate icosahedral symmetry. The interior surface of the coat, which must fit around the RNA, is assuredly not highly symmetric, however, and is therefore ill-resolved in X-ray diffraction studies. It is also likely that the exterior symmetry will be broken, particularly at the interfaces between the protein subunits that make up the coat. Understanding such structural details will be important for finding specific antigenic or molecular recognition sites on the exterior surface or for devising agents that could interfere with viral assembly or disassembly.

The smallest virus for which nucleic acid structural information is known is the tobacco mosaic virus, which contains about 3 million atoms in a cigar-shaped structure [3]. The approximate helical symmetry of the coat has been used to obtain structures from X-ray fiber diffraction experiments, but determination of the true structure will require simulations with no such assumed symmetry.

The most expensive computation in standard molecular mechanics and dynamics calculations is the evaluation of the nonbonded energy. Exact computation requires $O(N^2)$ operations, which is infeasible for large-scale systems. Truncation methods have been used to reduce the operation count to $O(N)$, but at the cost of significant decreases in accuracy, particularly for the long-range Coulomb interaction.

The Cell Multipole Method (CMM) [4] was developed to overcome these limitations in handling long-range power-law forces in molecular systems. In particular, it can be used to handle the R^{-1} (or R^{-2} if screened) Coulomb interaction and the R^{-6} attractive portion of the usual Lennard-Jones 12-6 or the exponential-6 van der Waals potentials. It is a true $O(N)$ -operation algorithm, with substantially better accuracy than cutoff methods of the same speed. It is thus the most suitable method for handling large-scale molecular mechanics and dynamics problems.

Improved algorithms are not sufficient for performing megamolecular simulations, however. Such large systems also require large amounts of memory and computation, far more than can be provided by the typical scientific workstation. These resources can be most cost-effectively provided at this time by scalable massively parallel computers.

The largest parallel computers available today are most efficiently programmed in a message-passing style. Unfortunately, it is not always easy to implement a set of mathematical equations, which do not by themselves specify appropriate data partitioning and communication patterns, in such a style.

We used a three step strategy to deal with this problem. First, an algorithm is implemented on a standard workstation without regard to parallelization. This allows testing on simple, small cases to ensure that the method incorporates correct physical principles. Second, we used a KSR-1 parallel supercomputer, which provides a global shared memory programming model despite its physically distributed memory, to parallelize incrementally larger portions of the calculation by partitioning data and computation across processors. During this step, calculation and

communication are separated within the code as much as possible. When the entire algorithm has been parallelized efficiently in this fashion, it is then reasonably simple to embed the resulting computational routines within a message-passing communications framework, producing an efficient, portable code that can run on the largest production machines, such as the Intel Paragon multicomputers or the Cray T-3D.

Part I of this thesis describes the theory behind molecular dynamics and the Cell Multipole Method, as well as an extension of the CMM to systems with periodic boundary conditions, the Reduced Cell Multipole Method (RCMM) [5].

Part II then discusses the implementation of a large-scale, parallel, distributed-memory, general-purpose molecular dynamics code on the KSR-1 parallel supercomputer. The code uses the CMM and RCMM to handle the nonbonded portions of the calculation. The design of the parallel aspects of the code, particularly the parallelization of the CMM, is described. Details of the implementation of the CMM in a similar, though currently slightly less general, molecular dynamics code on the Intel Touchstone Delta and Paragon multicomputers are also presented. Performance, accuracy, and scalability results are given.

Finally, Part III begins with a discussion of computational experiments leading to a prescription for choosing the value of the free time-scale parameter in Nosé-Hoover constant-volume, constant-temperature (NVT) canonical dynamics. This is followed by several applications of the above large-scale molecular dynamics codes to interesting chemical applications in the areas of argon cluster structure, polymer structure, surface tension of

water drops, diffusion of gases through polymers, and viral protein coat structure.

References

1. Tomalia, D.A.; Naylor, A.M.; Goddard, W.A. *Angew. Chem.*, **29**(2), 138 (1990).
2. Stanway, G. *J. Gen. Virol.*, **71**, 2483 (1990).
3. Namba, K.; Pattanayek, R.; Stubbs, G. *J. Mol. Biol.*, **208**(2), 307 (1989).
4. Ding, H.-Q.; Karasawa, N.; Goddard, W.A. *J. Chem. Phys.*, **97**(6), 4309 (1992).
5. Ding, H.-Q.; Karasawa, N.; Goddard, W.A. *Chem. Phys. Lett.*, **196**(1-2), 6 (1992).

PART I — THEORY

Chapter 2. Molecular Mechanics and Dynamics

2.1. Introduction

Molecular dynamics (MD) is the premier technique for simulating medium and large-scale molecular systems, including polymers, biological macromolecules, and materials. Although not as accurate as quantum mechanical calculations, MD can be applied to much larger systems and a much larger range of systems. MD makes the approximation that atoms can be treated as classical particles rather than electrons and nuclei, as in *ab initio* methods. For many systems where structure and dynamics are important but bond breaking is not, this assumption is quite reasonable.

The atoms in an MD simulation interact through a set of potentials that makes up a forcefield. These potentials are usually parameterized to reproduce experiment or more accurate theory. Summing the potentials as a function of atomic positions produces an overall energy for the system. The gradient of this energy then provides the force on each atom, which can be used to find the acceleration, velocity, and finally new positions of each atom.

2.2. Forcefields

There are two main types of energy terms used in forcefields: valence (or bonded) and nonbonded. The valence interactions include all terms related to the chemical bonds within the molecule, while the nonbonded interactions include through-space terms that are independent of atomic connectivity.

Valence terms typically include bond stretch, angle bend, torsional rotation, and inversion center contributions. Less frequently used terms

include various combinations of cross terms, pi-twist representations of torsional barriers, etc. Different forcefields may use different functional forms for the various valence terms, but most share the common characteristic that the energy is a function only of the atomic positions, together with constants depending on the types of the atoms involved in the interaction. A common set of valence terms would include the following:

$$E_{bond} = \sum_{bonds} k(R - R_0)^2 \quad (1)$$

$$E_{angle} = \sum_{angles} k(\theta - \theta_0)^2 \quad (2)$$

$$E_{torsion} = \sum_{torsions} \sum_p k_p \cos p\phi \quad (3)$$

$$E_{inversion} = \sum_{inversions} k(\cos \omega - \cos \omega_0)^2 \quad (4)$$

Nonbonded terms typically include the Coulomb and van der Waals interactions.

$$E_{Coulomb} = \sum_{A,B} q_A q_B R_{AB}^{-1} \quad (5)$$

$$E_{vdW} = \sum_{A,B} C R_{AB}^{-12} - D R_{AB}^{-6} \quad (6)$$

Some forcefields use special nonbonded terms to handle hydrogen bonds. These nonbonded components of the energy also depend only on the atomic positions and types.

Computing the energy and its gradient, the force, for a given system is thus a matter of identifying all the atoms participating in valence or nonbonded interactions, passing their positions and types to the appropriate functions, and summing the results.

Most widely-used forcefields for atomistic simulations can be placed within this general framework.

2.3. Microcanonical Dynamics

Once the forces on the atoms in the system have been computed, Newton's Second Law can be used to derive accelerations. The accelerations can then be integrated twice to generate velocities and then new positions for the atoms. We thus obtain a method for following the time evolution of the atomic positions.

Using the Verlet formulation for integrating the equations of motion [1,2], we obtain a procedure for updating the atomic velocities and coordinates that conserves the total energy (kinetic plus potential) in the system. This is known as microcanonical dynamics.

If we start from a minimum of the potential energy with a kinetic energy equal to twice the desired temperature, by equipartition, we should eventually reach equilibrium at the desired temperature. Since the potential surface may be complex, however, and since we cannot always start at a minimum, it may be impossible to maintain a fixed temperature within the dynamics. To overcome this limitation, we can rescale the temperature of the system periodically by appropriately modifying the velocities of the atoms. This allows us to simulate a physically reasonable temperature, at the cost of discontinuities in properties at the rescalings. Typically, the amount of rescaling needed decreases as the system reaches equilibrium, so it is possible to perform a long run without scaling to obtain thermodynamic averages after an initial equilibration period.

The equations of motion used are as follows, where \bar{x} , \bar{v} , \bar{f} , and \bar{m} are the atomic positions, velocities, forces, and masses, respectively, and Δt is the timestep:

$$\bar{v}_{n+1/2} = \bar{v}_{n-1/2} + \frac{\bar{f}_n}{\bar{m}} \Delta t \quad (7)$$

$$\bar{x}_{n+1} = \bar{x}_n + \bar{v}_{n+1/2} \Delta t \quad (8)$$

2.4. Canonical Dynamics

A better way of achieving constant temperature is to place the system in contact with an infinite heat bath which is fixed at the desired temperature [3,4,5,6]. If the system gains kinetic energy (and thus heats up), it will transfer the excess energy to the bath. If it loses kinetic energy, it will obtain energy from the bath. The conserved Hamiltonian in this case is the sum of the kinetic and potential energies of the system and the bath.

We use a double-half-step integration method. A half-step is taken to determine the velocity at timestep n from the velocity at timestep $n-1/2$, and another half-step is taken to go from the velocity at time n to the velocity at time $n+1/2$, which is needed for the next step. This method allows the velocity at each half-step to be determined, which is essential for computing the system's kinetic energy at both time n and $n+1/2$, while at the same time not requiring additional storage for velocities. The kinetic energy at time n is reported and contributes to the Hamiltonian, while the kinetic energy at time $n+1/2$ is used to compute $\dot{\zeta}$ at time $n+1/2$.

The equations of motion used are as follows, where ζ is the friction coefficient that controls heat exchange between the system and the bath, σ is

its integral, and τ_s is a free parameter that is related to the time scale for heat transfer to the bath:

$$\bar{v}_n = \frac{1}{1 + \zeta_n \frac{\Delta t}{2}} \left(\bar{v}_{n-1/2} + \frac{\bar{f}_n}{m} \frac{\Delta t}{2} \right) \quad (9)$$

$$\bar{v}_{n+1/2} = \bar{v}_n \left(1 - \zeta_n \frac{\Delta t}{2} \right) + \frac{\bar{f}_n}{m} \frac{\Delta t}{2} \quad (10)$$

$$\bar{x}_{n+1} = \bar{x}_n + \bar{v}_{n+1/2} \Delta t \quad (11)$$

$$\sigma_n = \sigma_{n-1/2} + \zeta_n \frac{\Delta t}{2} \quad (12)$$

$$\sigma_{n+1/2} = \sigma_n + \zeta_n \frac{\Delta t}{2} \quad (13)$$

$$\zeta_{n+1} = \zeta_n + \frac{1}{\tau_s^2} \left(\frac{T_{n+1/2}}{T_{bath}} - 1 - \frac{1}{3N} \right) \Delta t \quad (14)$$

$$KE_{bath} = \frac{3N}{2} k_B T_{bath} \tau_s^2 \zeta_n^2 \quad (15)$$

$$PE_{bath} = (3N+1) k_B T_{bath} \sigma_n \quad (16)$$

2.4. Minimization

Finding minima of complex, multidimensional potential surfaces is a heavily-researched field. Three approaches are presented here.

First, we can run dynamics at zero temperature. In this method, the atoms in the system will always move along the direction of their force vector, which is the gradient of the energy. This method uses a fixed step size.

Improved convergence can be obtained by varying the step size. In particular, projecting the location of the nearest minimum along the gradient direction can be very useful. This method is known as steepest descent.

Even better convergence properties can be obtained if the past history of the minimization is used. Successive search directions are constructed so that they form a set of mutually conjugate vectors with respect to the (positive-definite) Hessian of a general convex quadratic function. Convergence can be quadratic, rather than linear as for steepest descent, and is especially improved near the minimum.

Both steepest descent and conjugate gradient minimization require additional storage to enable the directed search procedure, but this additional storage is linear in the number of atoms.

References

1. Verlet, L. *Phys. Rev.*, **159**, 98 (1967).
2. Rahman, A. *Phys. Rev.*, **136**, A405 (1964).
3. Nosé, S. *Mol. Phys.*, **52**(2), 255 (1984).
4. Nosé, S. *J. Chem. Phys.*, **81**(1), 511 (1984).
5. Hoover, W.G. "Molecular Dynamics", *Lecture Notes in Physics* 258, Springer-Verlag, New York, 1986.
6. Hoover, W.G.; Ladd, A.J.C.; Moran, B. *Phys. Rev. Lett.*, **48**(26), 1818 (1982).

Chapter 3. Cell Multipole Method

3.1. Introduction

The most expensive computation in standard molecular mechanics calculations is the evaluation of the nonbonded energy. Exact computation requires $O(N^2)$ operations, which is infeasible for large-scale systems. We must thus find a way to represent groups of atoms in an approximate way, reducing the operation count while maintaining as much accuracy as possible. Traditionally, cutoff methods have represented all atoms sufficiently far away from an atom of interest as not being present at all, assuming that their effects would be negligible. Unfortunately, for many interesting systems this assumption may not be valid and accuracy suffers, even though the operation count is reduced to $O(N)$.

The Cell Multipole Method (CMM) [1] was developed to overcome these limitations in handling long-range power-law forces in molecular systems. In particular, it can be used to handle the R^{-1} (or R^{-2} if screened) Coulomb interaction and the R^{-6} attractive portion of the usual Lennard-Jones 12-6 or the exponential-6 van der Waals potentials. It is a true $O(N)$ -operation algorithm [2], with substantially better accuracy than cutoff methods of the same speed. It is thus the most suitable method for handling large-scale molecular mechanics problems.

The key feature of the CMM and other multipole methods [3,4,5,6] is that they replace the effects of atoms with multipole expansions representing the fields due to those atoms. This replacement reduces the number of computations that must be performed, while not ignoring these effects altogether as in cutoff methods. In order to maintain accuracy, stronger,

nearby interactions are represented more accurately than weaker, more distant interactions.

Computing and using these multipole expansions is the major computational task in all multipole methods. The CMM is a particularly regular, easily parallelizable multipole method that enables this task to be performed efficiently. It uses Cartesian coordinates only, unlike the other fast multipole formulations that use spherical harmonics [5,6,7], further simplifying the method. We can divide the CMM into four parts:

- Octree decomposition. The space occupied by the system of interest is divided into cells that form a tree. Each cell's effects will be represented by multipole expansions.
- Multipole expansion computation. The multipole expansions are computed for each cell at each level within the tree, starting from the leaves, or smallest cells, and working upwards in the tree to the root, which represents the entire system.
- Taylor series expansion computation. The multipoles represent the effects of atoms within a given cell. What we require, however, is the effect on an atom within a cell of all the other atoms in the system. We represent the long-range component of this effect with a Taylor series expansion that applies to all atoms within a given cell. The coefficients of this series are computed from the multipoles from the previous step, starting at the root of the tree and working downwards to the leaves.
- Farfield and nearfield computation. Once the Taylor series expansion coefficients have been computed for each leaf cell, the force on each atom can be computed as a combination of the

effects represented by the Taylor series, evaluated at the atom's location, and the effects of nearby atoms, which are calculated explicitly.

3.2. Octree Decomposition

In the CMM, the system of interest is surrounded by a bounding box, typically a cube, but in general a parallelepiped specified by three side lengths and three corner angles. This box, the level 0 cell, is then subdivided into eight octants by bisecting each side. The eight "child" cells are then themselves further subdivided into octants. This process continues recursively, forming an octree decomposition of the original box. The maximum level of the tree is a parameter of the method and is chosen to best maximize computational speed and accuracy. Increasing the maximum level will tend to increase computational speed but decrease accuracy; decreasing the maximum level will have the opposite effect. The cells at the maximum tree level ("lowest" level) will be referred to as "leaf" cells. Note that the decomposition is carried out to the same level across the entire system, unlike other adaptive multipole methods. This regularity makes it easier to determine the neighbors of any given cell and may increase the number of timesteps that can be executed before rebuilding the cell tree.

Figure 3-1 shows a two-dimensional projection of some of the cells in the octree. The bounding box has been divided into level 1 cells, one of which has been further subdivided into level 2 cells. A level 2 cell has in turn been split into level 3 cells. All of the level 1 and level 2 cells would be subdivided in this fashion.

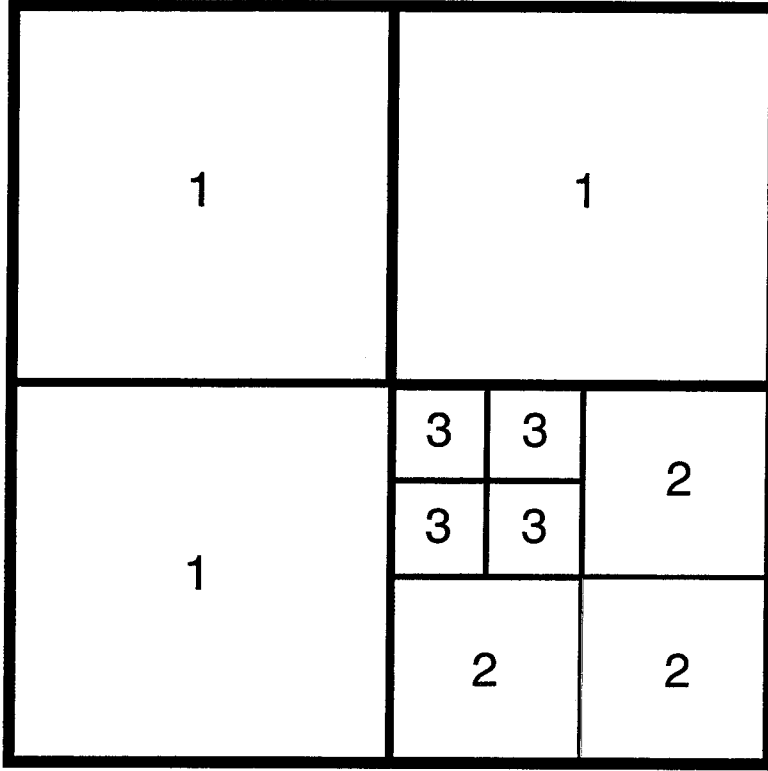


Figure 3-1. Two-dimensional representation of octree decomposition.

Cells are labelled with their level.

3.3. Multipole Expansions

Within the leaf cells, the effects of the atoms are replaced by multipole expansions. The level of multipole expansion (quadrupole, octupole, etc.) is another free parameter. Increasing the level of expansion will increase accuracy at the cost of extra computation time. The equations for the cell charges, dipoles, and quadrupoles are as follows:

$$\begin{aligned}
 q_{cell} &= \sum_{atoms} q_{atom} \\
 \mu_{cell,\alpha} &= \sum_{atoms} \epsilon R_{\alpha} q_{atom} \\
 Q_{cell,\alpha\beta} &= \sum_{atoms} \epsilon(\epsilon+2) R_{\alpha} R_{\beta} q_{atom} - \epsilon R^2 q_{atom} \delta_{\alpha\beta}
 \end{aligned} \tag{1}$$

where q , μ_α , and $Q_{\alpha\beta}$ are the charge, dipole, and quadrupole moments, respectively; $R_\alpha = R_{atom,\alpha} - R_{center,\alpha}$; \bar{R}_{atom} is the position of the atom; \bar{R}_{center} is the position of the center of the expansion; the power-law for the energy is $E \propto R^{-\epsilon}$; and $\alpha = x, y, z$.

3.4. Choice of Expansion Centers

The centers of the multipole expansions can be chosen in several different ways. The simplest choice is to use the geometric centers of the cells. If the atoms within a cell are not distributed evenly, however, expanding about the centroid (or average) of the atom locations produces improved accuracy for a given level of multipole expansion.

$$\bar{R}_{centroid} = 1/n_{atoms-in-cell} \sum \bar{R}_{atom} \quad (2)$$

Another possible alternative is to use a weighted average of the atom locations, with the weights being, for example, the absolute values of the charges on the atoms. This tends to place the center close to atoms with large charges, which will again tend to reduce the higher-order multipole coefficients.

The centroids or weighted averages can easily be computed in a hierarchical fashion using the existing cell tree, as the centroid of a higher level cell is equal to the weighted average of the centroids of its children, where the weights are the number of atoms in each child cell.

We have found that the increased accuracy from the centroid formulation allows the use of more highly truncated expansions than would otherwise be required. Further details are given in chapter 7.

3.5. Combination of Multipoles

Once the leaf cell multipoles have been computed, the expansions may be translated to the centers of the next higher-level (“parent”) cells and combined. In this way, the multipole expansion representing the parent cell is determined.

$$\begin{aligned} q_{parent} &= \sum_{children} q_{child} \\ \mu_{parent,\alpha} &= \sum_{children} \mu_{child,\alpha} + \epsilon R_{\alpha} q_{child} \end{aligned} \quad (3)$$

$$Q_{parent,\alpha\beta} = \sum_{children} Q_{child,\alpha\beta} + (\epsilon + 2) R_{\alpha} R_{\beta} q_{child} - (2 \vec{R} \cdot \vec{\mu} + \epsilon R^2 q_{child}) \delta_{\alpha\beta}$$

where $\vec{R} = \vec{R}_{center,parent} - \vec{R}_{center,child}$.

This process continues up the tree until the root (level 0 cell) is reached. At this point, every cell in the system has associated with it a multipole expansion representing the field due to all of the atoms contained within the cell.

3.6. Taylor Series Expansions

In order to compute energies and forces on a given atom within the system, we need to obtain the field due to all of the other atoms in the system. This can be broken into two components: the “nearfield” interaction due to all of the atoms in the same cell or neighboring cells, and the “farfield” interaction due to all of the rest of the atoms. In the CMM, the nearfield interactions are evaluated explicitly to ensure high accuracy while the farfield interactions are evaluated using effective fields.

A Taylor series expansion of the farfield for a given cell is used to enable the computation of the farfield at any point within the cell. The

coefficients of this expansion are determined from the multipole expansions computed in the previous step.

We define the Taylor series expansion of the field at the position of an atom to be

$$V(\vec{R}) = V_0 + \sum_{\alpha} V_{\alpha} R_{\alpha} + \sum_{\alpha\beta} V_{\alpha\beta} R_{\alpha} R_{\beta} + \dots \quad (4)$$

where $\vec{R} = \vec{R}_{atom} - \vec{R}_{center}$ and V_0 , V_{α} , and $V_{\alpha\beta}$ are the zeroth, first, and second order expansion coefficients, respectively. The centers used for the Taylor series expansion are the same as those used for the multipole expansion.

3.7. Computation of Taylor Series Coefficients

If we assume that a cell's parent's Taylor expansion has been computed and properly represents the field due to all atoms in the system outside the parent's neighbors, the remaining contribution we need to add to obtain the cell's farfield expansion is just the fields from the children of the parent's neighbors and from the other children of the parent, minus the fields from the cell itself and its immediate neighbors at its level. This is 27 cells (parent and its neighbors) times 8 (children per cell) minus 27 cells (cell and its neighbors), or 189 cells. Note that all of the children of the same parent are immediate neighbors of each other and can thus be omitted. We thus need to add to the parent's farfield expansion the fields from the parent cell's neighbors' children (which we will call PNCs), with the understanding that the other immediate neighbors of the cell in question are also excluded. Figure 3-2 shows a two-dimensional projection in which the immediate neighbors (excluded from the farfield) and the PNCs (included in the farfield) of a level 3 cell are shown. All of the shaded cells are with the parent of the level 3 cell or the parent's neighbor cells. The remaining unshaded level 2

cells are within the farfield of the parent cell, and are included in its contribution to the level 3 cell's farfield.

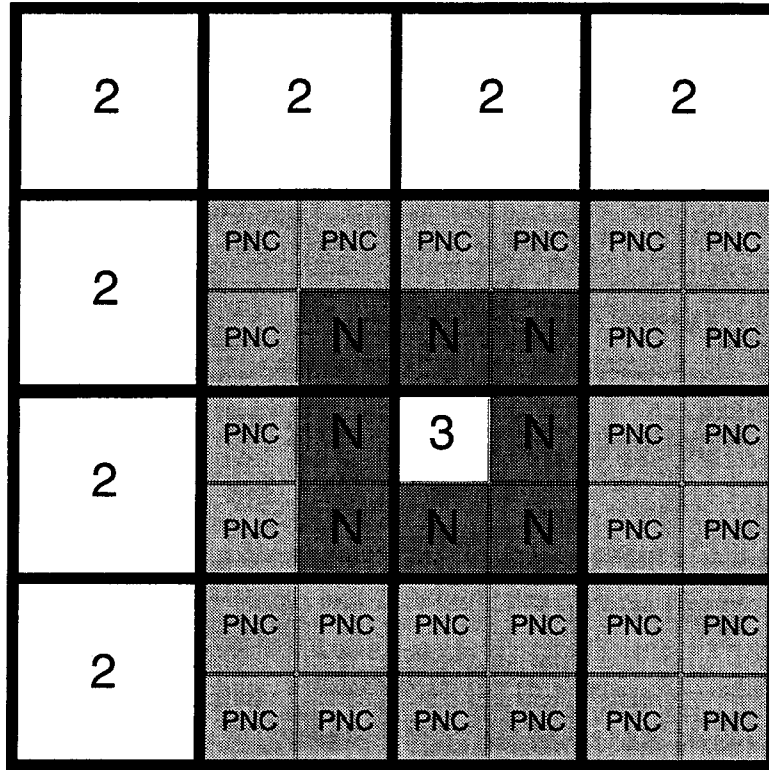


Figure 3-2. Neighbors and PNCs for a level 3 cell.

To start this process, note that all cells at level 1 (the first set of eight child cells) are immediate neighbors of each other, and thus there is no farfield contribution from the parent or the PNCs of these cells.

By induction, we can continue generating Taylor expansions all the way to the leaf cells, at which point every cell has a Taylor expansion representing the farfield within the cell.

We define the following useful terms:

$$\begin{aligned}
\bar{R} &= \bar{R}_{center, PNC} - \bar{R}_{center, cell} \\
\mu &= \sum_{\alpha} \mu_{\alpha} R_{\alpha} \\
Q &= \sum_{\alpha\beta} Q_{\alpha\beta} R_{\alpha} R_{\beta}
\end{aligned} \tag{5}$$

The contributions from each PNC to a cell's Taylor expansion coefficients are then:

$$V_0 = R^{-\varepsilon} q - R^{-\varepsilon-2} \mu + \frac{1}{2} R^{-\varepsilon-4} Q \tag{6}$$

$$\begin{aligned}
V_{\alpha} &= \varepsilon q R^{-\varepsilon-2} R_{\alpha} - R^{-\varepsilon-2} ((\varepsilon+2) R^{-2} R_{\alpha} \mu - \mu_{\alpha}) \\
&\quad + R^{-\varepsilon-4} \left(\frac{1}{2} (\varepsilon+4) R^{-2} R_{\alpha} Q - \sum_{\beta} Q_{\alpha\beta} R_{\beta} \right)
\end{aligned} \tag{7}$$

$$\begin{aligned}
V_{\alpha\alpha} &= \frac{1}{2} \varepsilon q R^{-\varepsilon-2} ((\varepsilon+2) R^{-2} R_{\alpha}^2 - 1) + R^{-\varepsilon-4} (\varepsilon+2) \mu_{\alpha} R_{\alpha} \\
&\quad - \frac{1}{2} (\varepsilon+2) \mu R^{-\varepsilon-4} ((\varepsilon+4) R^{-2} R_{\alpha}^2 - 1) + \frac{1}{2} R^{-\varepsilon-4} Q_{\alpha\alpha} \\
&\quad + \frac{1}{4} (\varepsilon+4) R^{-\varepsilon-6} \sum_{\beta} Q_{\alpha\beta} R_{\alpha} R_{\beta}
\end{aligned} \tag{8}$$

$$\begin{aligned}
V_{\alpha\beta} &= \varepsilon (\varepsilon+2) q R^{-\varepsilon-4} R_{\alpha} R_{\beta} \\
&\quad + (\varepsilon+2) R^{-\varepsilon-4} (\mu_{\alpha} R_{\beta} + \mu_{\beta} R_{\alpha} - (\varepsilon+4) R^{-2} \mu R_{\alpha} R_{\beta}) \\
&\quad + R^{-\varepsilon-4} Q_{\alpha\beta} \\
&\quad - (\varepsilon+4) R^{-\varepsilon-6} \sum_{\gamma} (Q_{\alpha\gamma} R_{\beta} R_{\gamma} + Q_{\gamma\beta} R_{\alpha} R_{\gamma}) \\
&\quad + \frac{1}{2} (\varepsilon+4) (\varepsilon+6) Q R^{-\varepsilon-8} R_{\alpha} R_{\beta}
\end{aligned} \tag{9}$$

where the q , μ , and Q multipole expansion coefficients are those of the PNC.

The contributions from the parent cell's Taylor series coefficients to one of its child cells' coefficients are:

$$\begin{aligned}
V_{child,0} &= V_0 + \sum_{\alpha} V_{\alpha} R_{\alpha} + \sum_{\alpha\beta} V_{\alpha\beta} R_{\alpha} R_{\beta} \\
V_{child,\alpha} &= V_{\alpha} + V_{\alpha\alpha} R_{\alpha} + \sum_{\beta} V_{\alpha\beta} R_{\beta} \\
V_{child,\alpha\beta} &= V_{\alpha\beta}
\end{aligned} \tag{10}$$

where $\bar{R} = \bar{R}_{center,child} - \bar{R}_{center,parent}$ here and the V coefficients on the right are from the parent cell.

3.8. Farfield Evaluation and Nearfield Computation

Once the cell Taylor expansions have been determined, computing the interaction energy and force due to the farfield at each atom is then simply a matter of evaluating the Taylor series at the atom's position (given by equation 4 above) and calculating the interaction of the field and the atomic charge.

Since the farfield changes much more slowly than the nearfield, we have found it feasible to perform the farfield calculation at intervals instead of every timestep. The centers and coefficients of the Taylor series expansions representing the farfield are kept constant during the interval.

The remaining nearfield interactions between an atom and the other atoms in the same cell and in neighboring cells are computed explicitly, using the appropriate charge-charge interaction equations (Coulomb or van der Waals).

References

1. Ding, H.-Q.; Karasawa, N.; Goddard, W.A. *J. Chem. Phys.*, **97**(6), 4309 (1992).
2. Esselink, K. *Information Processing Lett.*, **41**, 141 (1992).
3. Appel, A.W. *SIAM J. Sci. Statist. Comput.*, **6**(1), 85 (1985).

References (cont.)

4. Barnes, J.; Hut, P. *Nature*, **324**(6096), 446 (1986).
5. Greengard, L.; Rokhlin, V. *J. Comput. Phys.*, **73**(2), 325 (1987).
6. Carrier, J.; Greengard, L.; Rokhlin, V. *SIAM J. Sci. Statist. Comput.*, **9**(4), 669 (1989).
7. Schmidt, K.E.; Lee, M.A. *J. Statist. Phys.*, **63**(5/6), 1223 (1991).

Chapter 4. Reduced Cell Multipole Method

4.1. Introduction

Extending the CMM to handle systems with periodic boundary conditions substantially expands the range of problems that can be attacked. PBC is especially important for the simulation of bulk materials.

The Reduced Cell Multipole Method (RCMM) [1] is a relatively simple way of extending the CMM to periodic systems that maintains the overall scaling and memory usage advantages of the CMM. It builds on the idea of representing distant atoms with multipole expansions.

4.2. Enhanced Multipoles

First, an enlarged set of multipoles (through at least the hexadecapole moments) is computed from the atoms in the unit cell. This is an extended version of the standard CMM multipole computation and uses the same octree structure to ensure that it is $O(N)$ in the number of atoms.

Next, a random set of points within the unit cell is chosen. Charges are placed on these points so as to reproduce the enlarged multipole set determined above. The number of random points is equal to the number of multipole coefficients, so this amounts to merely solving a set of simultaneous linear equations. This “reduced set” of random points and charges then can be used to substitute for the unit cell at large enough distances without substantially decreasing accuracy.

4.3. Ewald Summation

In order to handle the infinite periodic lattice, a standard Ewald summation method is used [2,3]. In this method, we divide the effective field

of the infinite system at a given point into two pieces: a sum over atoms close to the unit cell in real space and a sum over terms in reciprocal space.

Computing an Ewald sum over all of the atoms in the unit cell would be computationally infeasible, so we only compute the sum over the reduced set, which is much smaller. This still reproduces all of the effects of the infinite system.

The real space part of the effective field is composed of the fields due to each of the nearby (reduced set) atoms, modified by a screening function.

$$V_{real} = \sum_{atoms} \frac{q_{atom}}{R} \text{erfc}(R/\eta) \quad (1)$$

where $R = |\bar{R}_{atom} - \bar{R}_{eval}|$ and η is a parameter that determines the cutoff range for the real space sum.

If there is an atom that is very close to the point at which the field is being evaluated, a different expression is used for the term due to that atom to account for the singularity in the field.

$$\left(-2 + \frac{2}{3} \frac{R^2}{\eta^2} - \frac{R^4}{\eta^4} \right) \frac{q_{atom}}{\eta \sqrt{\pi}} \quad (2)$$

The sum of these screened terms converges very rapidly as the distance from the evaluation point increases.

To correct for the screening functions introduced in real space, we must add terms that represent their complement. This sum now converges slowly in real space, but its Fourier transform converges rapidly in reciprocal space.

$$V_{recip} = \sum_{\vec{h}} \frac{e^{-\pi^2 \eta^2 h^2}}{\pi \Omega h^2} e^{i \vec{h} \cdot \bar{R}_{eval}} \exp \left(i \sum_{atoms} q_{atom} \vec{h} \cdot \bar{R}_{atom} \right) \quad (3)$$

where \vec{h} is the reciprocal lattice vector and Ω is the volume of the unit cell.

Finally, a correction is added to handle the case of charged unit cells.

$$V_{charge} = -\frac{\pi\eta^2}{\Omega} q_{cell} \quad (4)$$

$$V = V_{real} + V_{recip} + V_{charge} \quad (5)$$

4.4. Exclusion of Neighbors

At shorter ranges, the reduced set may not be an adequate representation of the atoms in the unit cell. Therefore, we will use a different technique (an extended CMM) to compute interactions with the nearest-neighbor unit cells. The Ewald sum over the reduced set already attempts to include these interactions, however, so we subtract out the field contribution from the nearest-neighbor unit cell images. Since this is within the range of the real-space portion of the Ewald sum, this subtraction is quite easy; the screening function in equation 1 above is simply modified by subtracting 1.

4.5. Combination with CMM

The Ewald sum is evaluated at a specific set of points within the unit cell. Taylor series coefficients representing the resulting field values are then computed by interpolation. These coefficients compose the farfield due to the infinite array of atoms, except for those in the nearest-neighbor unit cells. This is exactly the correct farfield to use for the (level 0) unit cell at the beginning of the induction step of the standard CMM.

To compute the interactions with the neighboring unit cells, we can extend the CMM into those cells. We use the multipole expansions and fields calculated for the unit cell, but translated to the neighboring images. This adds extra calculations for CMM cells that used to be at the edge of the bounding box but now have image cells as neighbors, as well as additional

calculations at level 1 in the octree, since the cells at that level now have PNCs in the image unit cells.

4.6. Noncubic Unit Cells

If the unit cell is not cubic, adjustments have to be made to the CMM. The method chosen is to compute a transformation matrix that converts the unit cell to a unit cube. The octree decomposition is then performed on the cube, with the cell coordinates being mapped back to real-space Cartesian coordinates by the inverse of the transformation matrix. The points at which the Ewald sum is computed are expressed in terms of transformed (unit cube) coordinates; the resulting interpolated Taylor series coefficients are then transformed back to real-space coordinates.

References

1. Ding, H.-Q.; Karasawa, N.; Goddard, W.A. *Chem. Phys. Lett.*, **196**(1-2), 6 (1992).
2. Tosi, M.P. *Solid State Phys.*, **16**, 107 (1964).
3. Karasawa, N.; Goddard, W.A. *J. Phys. Chem.*, **93**, 7320 (1989) and references therein.

PART II — DESIGN AND IMPLEMENTATION

Chapter 5. Implementation on KSR

5.1. Introduction

Numerous other groups have developed parallel molecular dynamics codes [1,2,3,4,5,6,7,8,9,10,11,12,13]. All of these codes suffer from various limitations, however. Some [1,2,3,4] limit the range of the forces, keeping the number of interactions low and thereby minimizing communication, but eliminating the possibility of handling interesting chemical systems which rely upon the long-range Coulomb interaction. Others [5,6,7] use replicated data methods, in which a copy of each atom is stored on each processor. Such codes are obviously unusable for large-scale systems which may not fit on a single CPU. Still others [8,9,10,11] use a ring for communication, which allows every atom to interact with every other atom, but also requires that half the memory of each CPU be reserved for incoming atoms and that every atom be sent to every CPU, maximizing communication. Kalia [12,13] has developed codes that handle nonbonded forces well but require special three-body potentials instead of standard valence forcefield terms. We set out to develop a code that would be parallel, distributed-memory (limited only by the total memory in the system), large-scale (able to handle systems of millions of atoms), and general-purpose (accepting standard forcefields). The heart of such a code is the nonbonded energy calculation, which is performed by the CMM.

The primary parallel implementation of the CMM algorithm has been performed on the Caltech Materials and Process Simulation Center's KSR-1 parallel supercomputer using the C language. This code uses the KSR's shared-memory programming model and is robust, efficient, and modular. It

has been used for production calculations on systems of up to 2 million atoms on the MSC 64-CPU, 2-gigabyte-memory machine and on systems of up to 4 million atoms on the Cornell Theory Center 128-CPU, 4-gigabyte-memory machine, with the main limitation being memory capacity. The program has been structured to permit the easy addition of new science (e.g. new forcefield terms and new integration methods), while still remaining efficient for large-scale computations.

5.2. KSR Architecture

The KSR AllCache architecture [14] presents a shared memory programming model to the user, even though it is implemented on top of a physically distributed memory. This model greatly simplifies the initial parallel programming task and, when attention is paid to the actual location of data, can still provide high parallel efficiency without requiring massive rewriting of code.

Two key features of the KSR architecture are used in the code. First, all data is stored in a single global address space. This means that data that is not specified as private is automatically sharable by all processors, merely by accessing a global variable or dereferencing a pointer. Second, any 128-byte “subpage” (the basic quantum of memory sharing in the machine) may be locked atomically by a processor so that only it retains access rights to the subpage. Any other processor attempting to lock (or even reference) the subpage must wait until the locking processor explicitly releases it. Though an exclusive lock primitive is often provided in parallel programming systems, since the KSR version is associated with 128 bytes of data, it is substantially more powerful. In particular, the cost of a lock is small in terms of execution time and zero in terms of memory usage (provided that the data

being locked is at least 128 bytes). These values are significantly cheaper than most implementations provide.

An additional feature provided by the KSR architecture is a fast reciprocal square root approximation instruction. The KSR FORTRAN compiler issues this instruction, followed by two Newton-Raphson steps to ensure accuracy, when computing $x^{-1/2}$. This is much more efficient than calling a library function to determine the square root and then performing a reciprocal operation. Since the C compiler does not make the hardware instruction accessible and does not properly optimize an explicit expression for the function, inlined code generated by the FORTRAN compiler is used.

The caching strategy of the machine allows data to be passed from CPU to CPU merely by referencing it. As long as the new CPU continues to use the data, it will remain in its cache. The copy of the data in the old CPU will be flushed when necessary or when the data is modified. Because of the global address space, no bookkeeping needs to be done in the software to keep track of where a given piece of data is. This contrasts with message-passing architectures, in which data must explicitly be sent from one CPU to another, and it is often necessary to maintain elaborate structures to identify the location of a desired piece of data.

5.3. Memory

In order to handle large-scale systems of millions of atoms, the amount of memory used per atom must be kept to a minimum. With existing machines' capacities ranging up to a few tens of gigabytes, maintaining an average memory usage of about one kilobyte per atom is essential for the simulation of million-atom systems. The KSR implementation uses 128 bytes (one subpage) to store the most important information about each atom and

additional memory of up to about 800 bytes per atom to store information about atomic connectivities, cell multipole and Taylor series coefficients, and valence forcefield information.

5.4. Data Structures

Critical atom data is stored in a single subpage for maximum efficiency. In addition, the most frequently used data is stored in the first 64 bytes of the subpage, or subblock, to enable it to be more efficiently cached in the onboard processor cache.


Offset	
0	cnext (link to next atom) 
8	x[0] (X coordinate)
16	x[1] (Y coordinate)
24	x[2] (Z coordinate)
32	f[0] (X force)
40	f[1] (Y force)
48	f[2] (Z force)
56	q[0] (Coulomb charge)
64	q[1] (London charge)
72	v[0] (X velocity)
80	v[1] (Y velocity)
88	v[2] (Z velocity)
96	m (atomic mass)
104	vchg2 (vdW repulsion charge)
112	n (global atom number) cell (CMM cell number)
120	type (atom type) flags (flag bits)

Figure 5-1. Atom data structure.

The cell data structure contains the position of the cell's center (or centroid), its multipoles, and its Taylor series coefficients. In addition, if the

cell is a leaf cell, a pointer to the cell's atoms is maintained. The “done” variable is used as a validity or “presence” tag to help avoid explicit global synchronization (see section 5.11).

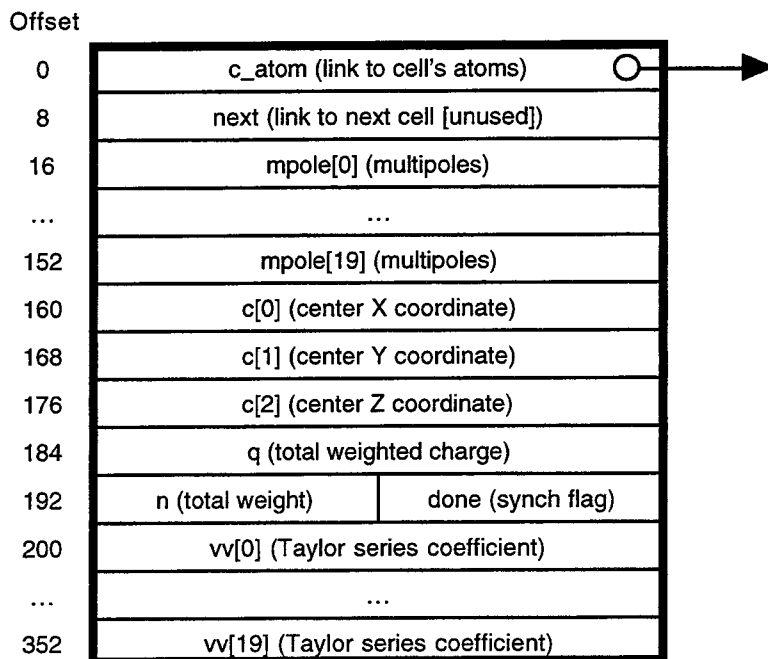


Figure 5-2. Cell data structure.

Atoms are stored in a globally-accessible array so that they may easily be retrieved by atom number. This is particularly useful in computing the valence forcefield portion of the energy, where each interaction is specified only in terms of the numbers of the atoms participating. The atoms are simultaneously linked into lists attached to the appropriate leaf cells to allow easy cell-by-cell handling in the CMM portion of the code.

Since each atom occupies one subpage, the atom array will be distributed element-by-element across the processors. A CPU performing computations using an atom's data will tend to maintain a copy of that data in its cache.

Pointers to cells are stored in globally-accessible arrays, one for each level. This allows a cell to be accessed easily given a pair of integers representing the level and cell number. Cells without atoms have no memory allocated for their data, but they do have pointer array elements (set to null) associated with them. As the number of levels increases, these arrays grow rapidly in size. For some systems containing significant amounts of open space, many of the cells will remain unoccupied, and it may be worthwhile to convert to a hash table or other sparse vector implementation rather than the explicit array. At levels up to about 7 ($2^{21} = 2$ million cells), however, the overhead of keeping empty elements in the array is outweighed by the speed of access.

Iterator functions are used to step through all atoms or all leaf cells. Each such iterator is first initialized, and then the iterator returns a new atom or cell at each call. The end of the list of cells is indicated by returning a null pointer.

Connectivity is stored in an array of six elements for each atom. Each element is composed of an atom number, a translation to a neighboring unit cell, if necessary, and a bond type index used for computing bond energies.

Valence forcefield entries are stored in another list associated with each atom containing the interaction type, the atom numbers (and possibly translations) used in the interaction, and a type index from which the forcefield parameters can be obtained.

Both the connectivity arrays and the valence forcefield entries are globally accessible. This simplifies checking for connectivities of neighboring atoms and allows slightly less complex updating when an atom moves across unit cell boundaries.

5.5. Modularity

Most parameters to the program are specified in a control file containing commands. The control file input routine is designed to accept one command per line, switch based on the first word of the command, and pass the remainder of the command to option routines associated with each module to further parse keywords. This allows the control file reader to be essentially isolated from the state variables maintained by each module.

The forcefield parameters are specified in a separate file, the format of which is defined by BIOGRAF [15]. The forcefield file input routine makes two passes through the file. In the first pass, the number of entries in each section of the file is counted. Section dividers are used to trigger calls to specific readers that understand the format of each section. In the second pass, memory is allocated for each forcefield parameter array in which data parsed from the file is stored.

Different functions may be used for different interactions in the forcefield. When a set of forcefield parameters has been read from the file, it is passed to a routine that looks up the function type in a dictionary and calls the appropriate setup routine to store the data, after any preprocessing, in the forcefield parameter array. A pointer to the routine that actually computes the function is also stored. The use of the dictionary and function pointers allows additional functions to be added easily.

The overall sequence of events within a timestep is controlled by a single routine (`real_nodes`). Additional computational routines may be added, with calls placed in the master routine at appropriate places during the timestep. It is hoped that eventually this master routine can be replaced

by an embedded language interpreter (such as Tcl or Perl) to allow even greater flexibility in the use of computational modules and analysis tools.

Integrators are also modular. Each integrator is called through a well-defined interface from the master sequencing routine. This interface is a list of pointers to functions provided by the integrator module. In addition, functions must be provided to save and restore any state information required by the integrator, and some additional functions used by the rigid molecule code are also required. This technique allows any integration method that can be structured to fit the (very general) interface to be used, and the integrator may keep any amount of private state information. Additional integrators may be added by merely defining the interface and providing a command to choose the appropriate function pointer list.

Atoms and cells could easily be reworked into C++ objects. Although many of their data elements are “public,” most manipulations of these data structures are handled through access functions. As an example, atomic forces may only be updated through one of two calls: a version that locks the atom to prevent access by any other CPU, performs the update, and then unlocks the atom, or a version that avoids the locking. The latter version (which is actually inlined for efficiency) is used primarily in the CMM code, where the structure of the algorithm makes it impossible for two CPUs to be attempting to update the same atom. Nevertheless, it could be replaced by the locking version, perhaps for debugging purposes, without modification to the rest of the code.

5.6. Features

Even a fast, efficient code is useless unless it can solve problems of interest to chemists. To that end, the code supports general forcefields,

including all normal valence terms, cross terms, and variant functional forms for each type of term. The commonly-used DREIDING [16] and AMBER [17,18] forcefields may be used (except for DREIDING hydrogen bond terms). Many additional features have been implemented to increase the problem domain that can be handled by the code.

5.6.1. Coordinate Transformations

Two types of coordinate transformations need to be performed in the code when periodic boundary conditions are used. The first type is necessary when noncubic unit cells are used. As described in Chapter 4, coordinates in the unit cell in real space are transformed to lie within a unit cube. The reverse transformation is also required, as are utilities for slightly more complex transformations related to reciprocal-space vectors used in the RCMM.

The second type of coordinate transformation occurs when a bonded interaction spans a unit cell boundary. All atomic coordinates are kept within the unit cell by the code. If an atom moves across a unit cell boundary, it is mapped to the image position on the other side of the unit cell. As a result, each atom participating in a valence interaction crossing a cell boundary may need to be remapped into its image position outside the unit cell before the interaction energy is computed. Such displacements are stored with the atom number in the bond and valence force field structures; the appropriate corrections are made to the atomic coordinate values within the various interaction computation routines through a standard utility function.

These displacements are always to neighboring unit cells. The Cartesian coordinate adjustments for each of the 26 neighbors can be precomputed, resulting in a very fast transformation routine that merely

performs three additions. Future versions of the code with variable unit cell parameters (e.g. for NPT dynamics) will require recomputation of these coordinate adjustments whenever the cell changes.

5.6.2. Moving Atoms

During the course of the dynamics simulation, atoms will move throughout the simulation volume. At intervals, a scan through all the atoms is performed, and any atom outside its assigned cell is collected for reassignment to a new cell. If the new cell does not exist, it will be created, and if the old cell is left empty, it will be deleted. Note that the farfield must be recalculated after this reassignment process to ensure that it is consistent with the new atom locations.

At the same time, atoms outside the unit cell for PBC systems are remapped to image locations within the unit cell.

5.6.3. PBC Displacement Updating

When atoms move across unit cell boundaries in PBC calculations, the displacements associated with them and the atoms with which they participate in bonded interactions must be updated. Two routines are provided that scan through all the bond and other valence force field interactions of a moving atom and modify the displacements to reflect the new location of the atom. The displacement encoding (an integer from -13 to 13) is chosen so as to allow displacements to be simply added, provided that no displacement extends beyond the immediate neighbor unit cells.

5.6.4. Atom Tracking

During, for example, diffusion calculations, it is important to know how far in real space a given atom has moved since the beginning of the simulation, despite remappings due to periodic boundary conditions. Since

the displacements above only maintain relative information, it is not possible to use them alone to determine coordinates in the initial frame of reference. A facility has been added to the code to designate certain atoms as being tracked. The state (coordinates, velocities, and forces) of the tracked atoms may be written out more frequently than that of the other atoms in the system, reducing the amount of data that needs to be saved. In addition, absolute displacements from the original frame of reference in each of the Cartesian coordinate directions are maintained for the tracked atoms, allowing recovery of the true distance moved by an atom.

5.6.5. General Valence Forcefield

All bonded interactions are calculated in parallel by iterating through the list of atoms on each processor. Bond interactions are calculated twice (from the point of view of each atom participating in the bond); all other interactions are calculated only once, with the results communicated to the appropriate atoms. To date, this portion of the calculation has not been highly optimized. In particular, the amount of communication could be reduced by performing all calculations for each non-local atom at one time.

To set up the valence force field interactions, it is necessary to determine which interactions are to be computed and to assign each interaction a type from the forcefield. This type is just a key specifying the appropriate function and set of parameter values to be used when calculating the energy and forces due to the interaction.

Selecting interactions is a relatively simple process. From the input, we know the molecular connectivity: which atoms are bonded to which other atoms. Any atom that is bonded to two other atoms forms an angle. Any atom bonded to exactly three other atoms also forms an inversion center.

Any angle plus an additional bond on one side forms a torsion. Some additional checking needs to be done to ensure that only one instance of each interaction is generated and that systems with loops (e.g. cyclopropane) are handled properly.

Once the atoms participating in an interaction have been identified, we can use their atomic types to determine the type of the interaction. To simplify this process, a set of trees, one each for bonds, angles, torsions, and inversions, is built from the forcefield file information. Each interaction is placed in a canonical order based on its atomic types. The first type is then used to choose a branch from the root of the appropriate tree; the second type selects a branch from the resulting node; etc. Additional branches are added to the trees to handle wildcards in the type specifications from the input file. The wildcard branches are only tried if there is no type-specific branch.

An additional tree is built for hydrogen bond (or off-diagonal nonbond) interactions, which are further described in section 5.6.8. Although the bond, angle, torsion, and inversion trees may be disposed of after the program is initialized, hydrogen bonds are not completely predetermined by the initial geometry and so this small additional tree must be retained throughout the simulation.

5.6.6. Nonbond Exclusions

Many forcefields define the nonbond interactions so as to exclude any components stemming from atoms that are participating in bonded interactions. Such exclusions could be handled in the CMM nearfield computation step by checking each interaction to be calculated, but such checking would be quite expensive. In addition, if a bonded interaction

spanned a leaf cell, the most distant atoms in the interaction would only interact through the CMM farfield, not the nearfield.

To overcome these difficulties, we compute the CMM normally, with no checking for exclusions. We then compute the excluded interactions explicitly, subtracting them from the energy and forces computed by the CMM. The list of excluded interactions can be built once, since it depends on the molecular topology, which doesn't change as long as no bonds are broken or created. The calculation of the exclusions can occur in parallel, with each CPU handling the computations for the atoms that are in the cells it owns.

The excluded energy is often dominated by Coulomb and van der Waals repulsion terms at short ranges and can thus be quite large. The net energy is then the difference between two large numbers (the unexcluded energy and the excluded energy), which could potentially lead to loss of accuracy. We have found, however, that use of double-precision (64-bit) floating point numbers provides more than adequate accuracy for non-pathological cases. Since this precision is all that the KSR floating point unit provides, there is no loss in performance.

5.6.7. Spline Nonbonds

The code also handles spline-type nonbonds for comparison with this older method. As in traditional programs, a list of all nonbond interactions to be computed is generated periodically during the simulation. To speed up the list generation process, the code uses the same cell structure generated for the CMM. The sizes of the cells are set to at least the length of the spline cutoff distance. Then any interaction that could potentially be included on the nonbond list must be between atoms in neighboring cells. Each CPU generates a nonbond list for the atoms it owns; each interaction is included

twice. The lists are stored in a packed form, removing redundancies in the storage of the numbers of local atoms. Exclusion tests are performed while building the nonbond list; no separate excluded force is calculated in this case. A standard cubic spline is used to smooth the energy function at the cutoff radius.

5.6.8. Hydrogen Bonds and Off-Diagonal Nonbonds

AMBER-type hydrogen bonds are supported through a general facility that allows special off-diagonal nonbond parameters and functions to be applied. As for the spline nonbonds, a list of potential off-diagonal interactions is generated. Like the nonbond exclusions, the CMM-computed interaction is first subtracted; the new off-diagonal energy and forces are then added in.

5.6.9. Initial Velocities

The initial velocities of the atoms in the system are generated by sampling a random Gaussian distribution for each Cartesian component. The standard deviation of the Gaussian is determined by the simulation temperature. Once selected, the velocities are rescaled to ensure that the initial temperature exactly matches the desired one. The selection and rescaling process is simple to parallelize; each CPU computes initial velocities for the atoms that have been assigned to it.

Use of truly random velocities makes debugging the code or methods for using it difficult, as no run may be repeated exactly. To overcome this, we need to preserve the seed used for the pseudorandom number generator. If we saved a seed for each CPU, however, we would not be able to execute the same run on a differing number of CPUs.

The solution used was to determine the generator seed algorithmically, by computing a function of each atom's coordinates. The generator is reinitialized for each atom. Use of a function of the coordinates ensures that there is no dependency on the number of CPUs, the assignment of atoms to CPUs, or any other aspect of the parallelism.

There is, of course, an option for production runs to use true random numbers seeded by the time-of-day clock on each CPU.

5.6.10. Rigid Molecules

Rigid molecules are handled with a quaternion method [19]. The quaternion represents the rotational state of the molecule. The forces applied to atoms in the molecule are decomposed into a center-of-mass translational force and a torque. The translational force is applied to the center of mass of the molecule, which can be integrated normally. The torque is used to update the angular momentum of the molecule, which is in turn used to generate the new rotational quaternion through an iterative procedure.

5.6.11. Perturbation Thermodynamics

A limited capability to perform perturbation thermodynamics calculations has been implemented. The program allows individual atoms to be mapped from one atom type to another. Only the nonbonded parameters (van der Waals and charge) are affected. The value of λ , the percentage blend of the two atom types, may be specified. If two values of λ are given, the program evaluates energies with each value at each timestep and prints the resulting energy difference along with various statistical information. Only one value of λ is used to determine the forces that control the dynamics.

5.7. Input/Output

To ease the transition from workstation-based programs, the program reads BIOGRAF-format forcefield and structure files, with only minor modifications needed to simplify the code or support additional functionality.

Input data to the simulation includes a control file giving parameters for the dynamics, a forcefield file giving parameters for the various terms in the energy expression, and a structure file containing the atomic position, charge, and connectivity information. The input data set totals 80–120 bytes per atom (depending on the system's connectivity).

Output includes the potential and kinetic energy at each timestep, as well as other parameters of the dynamics, such as the total Hamiltonian. At user-specified intervals, snapshots of the system are taken containing positions, velocities, and forces on each atom. These allow analysis of the properties of the system (including evolution with time) and also serve the important purpose of allowing the simulation to be restarted. The output files contain 72 bytes per atom, plus a small header, and a possible trailer for periodic boundary condition systems.

5.8. Parallelization

The CMM was initially implemented on single-processor workstations. Transferring the code to a single processor of the KSR-1 was simple, but achieving an efficient parallel implementation required substantial work.

The primary method of parallelization used is domain decomposition. We partition the cells (at all levels) across the set of CPUs. Each CPU then computes all relevant information for the cells and atoms it has been assigned, communicating with other CPUs as necessary. The two major

obstacles to peak efficiency are the amount of communication required and imbalances in the amount of computation required on each CPU.

5.9. Communications

Since the CMM decomposes space into an octree of cells, it is natural to decompose the MD data across the processors of a parallel machine in the same spatial manner. Each processor is then responsible for a volume of space, and communication is only required across the surface area of that volume.

Minimizing this surface area while distributing the data is highly desirable to keep communications costs low. To achieve this goal, each cell is assigned a number. We use an octree numbering system, in which a cell's number is equal to its parent's number multiplied by 8 plus an index varying from 0 to 7. This system ensures that consecutively-numbered cells at the same level are generally close to each other in space. In particular, any range of cell numbers tends to form one or two approximately-cubic domains. Assigning such a range of cell numbers to each CPU will then tend to keep the surface area associated with each CPU small. Although this may not be the ideal partitioning, it works well, even on highly irregular (non-cubic) systems and is simple to implement.

A cell's number can thus be represented by a sequence of octal (base 8) digits, each corresponding to the child index at a different level.

Within this numbering system, the numbers of a cell's children or parent can be computed using simple expressions:

$$\begin{aligned} n_{child} &= n_{parent} \times 8 + index_{child} \\ n_{parent} &= \lfloor n_{child} / 8 \rfloor \end{aligned} \tag{1}$$

where the square brackets denote the greatest-integer function.

Determining the number of a cell's neighbor in, say, the $-x$ direction is slightly more complex. Conceptually, we wish to subtract one from an integer composed of the bits forming the x coordinate of the cell. These bits are the lowest-order bits of each octal digit (three bit group). We can thus use the following C code to mask out the desired bits and perform the subtraction with borrows, if necessary, through the intervening bits, which are then restored:

```
mask = 01111111111; /* octal */
nmask = ~mask;
x = cell & mask;
if (x > 0) {
    return (x - 1) & mask | (cell & nmask);
}
else {
    return -1; /* no such neighbor */
}
```

Similar code applies to the other directions (the mask need merely be shifted) and to the positive directions (requiring additions with carries rather than subtractions with borrowing).

5.10. Dynamic Load Balancing

Because this implementation of the CMM does not store information for unoccupied cells, and because systems of interest are often irregularly shaped, load balancing is a particular problem with this code.

The simplest approach is to assign the same number of leaf cells to each CPU. This fails, however, since many cells may be empty. Particularly with regular cell numberings, assigning equal-sized ranges of cells to CPUs

can often lead to some CPUs not having any atoms (and thus computations) at all.

To a first approximation, the computational time per timestep is dominated by the nearfield interactions, which in turn are dependent on the number of atoms (or occupied cells) assigned to each processor. Therefore, we arrange for each processor to be responsible for a consecutively-numbered range of cells containing no less than n_{atoms}/n_{cpus} atoms (except for the last CPU). The use of ranges keeps the surface area low, as mentioned above, and also limits the size of the tables needed to determine on which CPU a given cell resides to merely one integer per CPU. This approach led to satisfactory load balancing.

Atom movements may cause the load to become unbalanced again. Each cell can determine how many atoms it contains; a simple, rapid linear sweep through the CPUs then can readjust the cell ranges and communicate the cell and atom data to their new locations. The KSR architecture makes this last step trivial: data migrates to each processor's cache as it is referenced in the next timestep, so it is not necessary to communicate it ahead of time.

The final code uses an even more sophisticated approach. After the initial load balance using the above technique, the amount of time each CPU spends waiting at synchronization points is measured. CPUs with longer waiting times do not have enough work. At intervals in the simulation, each CPU compares its accumulated waiting time with the average value across all the CPUs. Those with shorter waiting times give up cells and atoms to those that have longer waiting times, in proportion to the ratio of the waiting

time to the average. The proportionality constant is adjustable and was empirically chosen to be 2.

This more advanced repartitioning strategy led to significant performance improvements on irregular problems. See Chapter 7 for more details.

Upper levels in the tree are assigned in such a way as to generally minimize the amount of communication required during tree traversals. A parent cell is assigned to the same CPU as its 0-th numbered child. Given the load-balancing-determined ranges of leaf cells on each CPU, it is simple to determine the CPU containing a higher-level cell by a simple shift and binary search.

5.11. Avoiding Synchronization

Since the KSR architecture provides a shared memory programming model, we can avoid some synchronizations that would otherwise be required to maintain data dependencies by allowing processors to explicitly check for the availability of needed data. This is accomplished by placing a “volatile” variable in each cell data structure that is set to a different value depending on what portions of the cell data have been computed. Processors requiring cell data to proceed can check the variable and wait if the values they need have not yet been calculated. The structure of the algorithm guarantees that deadlock will never occur.

Avoiding global synchronizations, in which processors that may not yet have data dependencies nevertheless must wait for slower processors, substantially improves the efficiency and speed of the code. On large problems with sufficient numbers of cells and atoms per processor, almost no

waiting for data dependencies occurs at the lower, most populous levels of the tree.

When global synchronizations do need to be performed, they are implemented using the KSR-supplied, POSIX-compatible barrier primitive.

References

1. Liem, S.Y.; Brown, D.; Clarke, J.H.R. *Comput. Phys. Commun.*, **67**(2), 261 (1991).
2. Wagner, N.J.; Holian, B.L.; Voter, A.F. *Phys. Rev. A*, **45**(12), 8457 (1992).
3. Lomdahl, P.S.; Beazley, D.M.; Tamayo, P.; Gronbechjensen, N. *Int. J. Modern Phys. C*, **4**(6), 1075 (1993).
4. Plimpton, S. *J. Comput. Phys.*, **117**(1), 1 (1995).
5. Sato, H.; Tanaka, Y.; Yao, T. *Fujitsu Sci. Tech. J.*, **28**(1), 98 (1992).
6. Skeel, R.D. *J. Comp. Chem.*, **12**(2), 175 (1991).
7. Mertz, J.E.; Tobias, D.J.; Brooks, C.L.; Singh, U.C. *J. Comp. Chem.*, **12**(10), 1270 (1991).
8. Li, J.; Brass, A.; Ward, D.J.; Robson, B. *Parallel Computing*, **14**(2), 211 (1990).
9. Smith, W. *Comput. Phys. Commun.*, **62**(2-3), 229 (1991).
10. Janak, J.F.; Pattnaik, P.C. *J. Comp. Chem.*, **13**(9), 1098 (1992).
11. Gupta, S. *Comput. Phys. Commun.*, **70**(2), 243 (1992).
12. Kalia, R.K.; Jin, W.; Deleeuw, S.W. *Int. J. Quant. Chem.*, 781 (1993).
13. Nakano, A.; Vashishta, P.; Kalia, R.K. *Comput. Phys. Commun.*, **77**(3), 303 (1993).
14. Kendall Square Research. "KSR/Series Principles of Operation," Revision 7.0, March 15, 1994.
15. BIOGRAF version 3.21, Molecular Simulations, Inc.

References (cont.)

16. Mayo, S.L.; Olafson, B.D.; Goddard, W.A. *J. Phys. Chem.*, **94**(26), 8897 (1990).
17. Weiner, S.J.; Kollman, P.A.; Case, D.A.; Singh, U.C.; Ghio, C.; Alagona, G.; Profeta, S.; Weiner, P. *J. Am. Chem. Soc.*, **106**(3), 765 (1984).
18. Weiner, S.J.; Kollman, P.A.; Nguyen, D.T.; Case, D.A. *J. Comp. Chem.*, **7**(2), 230 (1986).
19. Goddard, W.A. "Rigid Body Dynamics," California Institute of Technology, Materials and Process Simulation Center, *MSC Technical Note 105*, October 13, 1993.

Chapter 6. Implementation on Message Passing Architectures

6.1. Introduction

The KSR architecture is unique among existing massively parallel machines, and the demise of the company promises to maintain that status for the near term. A more common architecture uses the message-passing model, in which each processor has its own dedicated memory, inaccessible to any other processor, and communications are explicitly performed through messages passed from one processor to another. Examples of message-passing architectures include the Intel Touchstone Delta and Paragon XP-S, the IBM SP-1 and SP-2, the Cray T-3D when programmed using PVM, and the experimental MIT J-Machine.

Current multicomputers can be classified as coarse-grain systems in which tens to hundreds of processors are relatively loosely coupled, communicating using (optimally) large messages. At one end of this grouping are workstation clusters and machines like the IBM SP-2, which run a complete operating system on each node; at the other end are machines like the Cray T3D, Intel Delta/Paragon, or KSR (which can also be programmed using a message-passing model) which are more tightly coupled and use a distributed OS or a simple runtime system.

A major shift of emphasis in architectural design for parallel computers is occurring, however. New fine-grain multicomputers built from thousands of processors, coupled using a low-latency, small, active message paradigm, show great promise in improving the capability of parallel systems. Their primary advantages lie in the ability to scale well to orders of

magnitude more processors than existing machines, the low overhead cost of the active-message communications, and the ease of implementation of the object-oriented paradigm that is becoming the fundamental basis of software technology. This new class of machines is exemplified by the MIT J-Machine prototype.

It is not yet clear how to use these new machines for non-trivial applications, how best to program them, or how best to compile programs for them. Previous work in this area has led to the development of prototype programming systems for the J-Machine and portable programming abstractions that can be used to implement a wide variety of irregular concurrent algorithms. These methods are now at the point where they can be applied to real-world problems in materials simulation.

The computational routines developed from the KSR code have been incorporated into a second code currently running on the Caltech/CSC Intel Touchstone Delta and Paragon XP/S machines, and potentially portable to the JPL Cray T3D. This code uses a new communications architecture based on a portable active message library, and thus also compiles on the prototype MIT J-Machine. It is suitable for calculations on very large-scale systems of up to 20 million atoms. The code also serves as a testbed for flexible optimization strategies designed to give efficient performance on traditional coarse-grain as well as fine-grain processors despite the highly asynchronous, multithreaded nature of the communications strategy.

6.2. Message Types

Five types of messages implement the heart of the CMM algorithm:

1. Cell center sent from a child cell to its parent cell.
2. Multipoles sent from a child cell to its parent cell.

3. Multipoles sent from a cell to its PNCs.
4. Taylor series coefficients sent from a parent cell to its children.
5. Atoms sent from a leaf cell to its neighbors.

The first four of these are used to compute the farfield and occur only at intervals; the last type of message must be sent at every timestep.

Note that the cell centers need to be sent up the tree before the multipoles because each parent cell needs to determine what its center is before it can process incoming multipoles from its children.

Additional message types are used for initialization, synchronization, flow control, and atom reassignment.

6.3. Active Messages

An active message model was used for the development of the code. In this model, reception of a message triggers the execution of a function specified in the message with the message contents passed as an argument or arguments to the function. This model allows low-latency communications through the avoidance of copying. It also leads to a natural, asynchronous, multithreaded style of programming. These advantages of the active message model may make it the preferred programming model for future generations of multicomputers.

Currently, the active message model is the preferred model for programming experimental fine grain parallel processing hardware such as the MIT J-Machine and M-Machine [1,2,3,4,5], which are among the targeted platforms for this code.

As an example, the second type of message from the previous section, involving the communication of multipoles from a cell to its parent. This is implemented through an active message of type CHILD, with the destination

node, destination (parent) cell, and sending cell's multipoles as arguments. When such a message is received on the destination node, it triggers the invocation of the `child()` routine which parses the arguments, obtains the appropriate destination cell, and calls a purely computational routine, identical with the KSR code's corresponding routine, to combine the child's multipoles with the parent's.

Although active messages have not been extensively used on traditional multicomputers such as the Intel's Delta and Paragon [6,7], we have found that refinements can be added to improve performance on such architectures, primarily by gathering together multiple small messages into a few large messages. This buffering significantly reduces per-byte overhead; improvements in wall clock time of factors of 2 to 5 for a five million atom case were observed when buffering was implemented.

Since the fundamental messaging primitives have no flow control, we implemented a flow control system on top of the active messages. Each processor is allowed to send a certain number of messages (a window) to each other processor in the system. When the window is full, it must wait until it has received acknowledgements before sending more messages. Keeping the windows on a per-processor basis allows more messages to be outstanding (not acknowledged) than if a single window were used on each processor. This in turn reduces the amount of nonproductive busy-waiting on each processor.

The acknowledgement can often be packed with message data needed to further the calculation. In addition, we could optimize further by sending buffered messages taking into account the size of the system message buffer, as indicated during program invocation. The buffering strategy we use not

only handles flow control on the underlying message system, but also reduces the overall latency costs by sending fewer larger buffers, rather than more smaller buffers.

There are two possible messaging strategies: a “pull” strategy in which data is requested from another processor and a “push” strategy in which data is sent to a destination processor. The “pull” strategy requires two messages per data transfer, while the “push” strategy could possibly be optimized to only use one message per transfer, if empty acknowledgements were not required. The expected savings led us to use the latter strategy.

Portability across a wide range of message-passing machines, including the experimental MIT J-Machine, was desired. To achieve this goal, the message-passing code is organized as a set of procedures executed via what amount to remote procedure calls. The main program is simply a dispatcher that executes the appropriate procedure for a given message type, with arguments obtained from the message data.

The active message strategy was implemented on top of the Intel-provided messaging system, NX.

One additional optimization can be performed within the context of the CMM. Often, the same data (such as a cell’s multipoles or its atoms) must be sent to multiple neighboring cells that are not on the local node. In many cases, all of those remote cells will be assigned to the same CPU. In that event, we can eliminate the redundant data transmissions by sending only one copy of the data but specifying that multiple cells as destinations. Such redundancy removal can provide significant performance gains.

6.4. Load Balancing

The message-passing code has not yet had the sophisticated load balancing advances of the KSR code implemented in it. Currently, an external program divides up cells among processors so that the number of atoms per processor is approximately equal, generating a map. The atoms in the input BIOGRAF file are then distributed to multiple “split” BIOGRAF files, one for each CPU, according to this map. This initial load balancing step is the only one that occurs; there is no dynamic load balancing during the course of the simulation.

6.5. Input/Output

Input data to the simulation, as for the KSR code, includes a control file giving parameters for the dynamics, a forcefield file giving parameters for the various terms in the energy expression, and a set of structure files, one per CPU, containing the atomic position, charge, and connectivity information. These structure files are generated by the load balancing preprocessor from a single, unified structure file based on the number of CPUs to be employed. The input data set totals 80-120 bytes per atom (depending on the system’s connectivity).

Since large systems of atoms can span hundreds of megabytes per input file per processor, we found this approach to be stressful for the relatively small number of I/O nodes handling the disks on the Intel Paragon and Delta. When each processor opens its own file, the read requests are funneling through very few (16 on the 512 compute node Paragon we ran our timings on) I/O nodes. The I/O nodes get overloaded retrieving information spread across the disks, so we must throttle our requests. Another approach would have been to reorganize the single, unified structure file such that each

processor could seek to its own place in the file and read. This would have allowed us to take advantage of the Intel optimized global read/write calls, but moved away from the single reader/writer mode which the J-Machine prefers. Having all the processors read all the atoms from the structure file, discarding those that are not local, is probably too expensive for the multi-gigabyte data sets needed for very large systems.

Output is again similar to the KSR code and includes the potential and kinetic energy at each timestep, as well as other parameters of the dynamics. At user-specified intervals, snapshots of the system are taken containing positions, velocities, and forces on each atom. Each snapshot is composed of one file written from each CPU.

We found that many hours of production runs not only needed the flexibility of restarting, due to machine crashes and varying scheduling policies, but would have also benefited from resuming on a different number of processors than the original run. Since the startup and checkpoint files were written on a per-processor basis, we would have had to reassign cells and atoms (and rebalance the load) if the number of processors changed. This was infeasible given the current I/O architecture; use of a unified input file might make this easier.

Although calculation per timestep might take longer on smaller numbers of processors, available CPU time for small to medium numbers of processors is often times much greater than similar blocks of time for large numbers of processors. Thus, time to solution for systems of atoms which would fit on various sized partitions could have been shortened by flexible restart capabilities.

6.6. Data Structures

Cells are stored in hash tables local to each processor. This allows rapid lookup based on the cell's level and number. The tree structure is maintained by utility functions that return the cell numbers for the parent, children, or neighbors of a given cell, rather than through explicit pointers.

Since cells on different nodes often need to interact, we need a method for determining which node a given cell at a given level is on. Since each processor is assigned a consecutive range of leaf cells, and since there is a fixed rule for assigning parent cells, all that is required is that each node have a copy of the table describing the leaf cell range limits for all the nodes. A simple binary search through this table (at most 9 steps for 512 nodes) produces the number of the node containing a desired cell.

While the nonbond calculations do not depend explicitly on the identity of atoms within the cells, the bonded calculations must in order to maintain the correct molecular topology. This requires that a processor calculating a bonded interaction know the coordinates of the atoms participating in that interaction. On the KSR, obtaining data from non-local atoms can be left to the memory system. On message-passing machines, we make the assumption that all bonded interactions only involve atoms in the same leaf cell or the nearest-neighbor leaf cells. Since the coordinates of the atoms in those cells need to be communicated anyway for the nearfield interaction computation, all we must do is save them until the valence calculations have been completed.

This portion of the code is just now being implemented; the current version stores all atom data in hash tables, taking care to keep data from remote nodes and data from the local node distinct. After each nearfield

calculation, all of the data required for the valence calculations may be found in the hash tables. After the valence portion of the code, the remote atoms may be removed.

On the KSR, each processor may update the forces for non-local atoms as long as locking is used to prevent simultaneous updates by multiple processors. The message-passing code updates the partial forces in the local copies of remote atoms and then communicates these partial sums back to the “home” node after all of the bonded calculations have been performed. No locking is needed since only the local processor can access the atoms’ memory.

6.7. Computational Routines

The key feature of the Delta/J-Machine port is the constancy of the computational routines developed originally on the KSR. Since computation and communication were separated during the incremental parallelization on that machine, large portions of the KSR code could be incorporated directly into the new message-passing structure without modification.

References

1. Dally, W.J., et al. “The J-Machine: A Fine-Grain Concurrent Computer,” *Information Processing 89*, G.X. Ritter (ed.), Elsevier Science Publishers B.V., North Holland, IFIP, 1989.
2. Dally, W.J.; Fiske, J.A.S.; Keen, J.S.; Lethin, R.A.; Noakes, M.D.; Nuth, P.R.; Davison, R.E.; Fyler, G.A. *IEEE Micro*, **12**(2), 23 (1992).
3. Dally, W.J., et al., “M-Machine Architecture v1.0,” Massachusetts Institute of Technology, Artificial Intelligence Laboratory, *Concurrent VLSI Architecture Memo 58*, February, 1994.

References (cont.)

4. Maskit, D. "A Message-Driven Programming System for Fine-Grain Multicomputers," Masters Thesis, California Institute of Technology, February, 1994.
5. Maskit, D., et al. "System Tools for the J-Machine," California Institute of Technology, Department of Computer Science, *Technical Report CS-TR-93-12*, 1993.
6. URL: <http://www.ssd.intel.com/homepage.html>.
7. URL: http://www.ccsf.caltech.edu/annrep94/facil_1.html.

Chapter 7. Performance

7.1. Introduction

There are three important aspects to the performance of a program. First, determining the accuracy of the results is crucial. Second, the overall time to solution is a key criterion. Third, for parallel programs, the scalability of the code with respect to the size of the system and the number of processors is important.

The accuracy of the KSR and message-passing codes is identical, as they use the same computational routines. Their speeds and scalabilities are evaluated separately below.

7.2. Accuracy

There are two standards to compare against to determine the accuracy of the CMM. One is of course the exact nonbond calculation using all N^2 pairwise interactions. The other is the *de facto* standard method used for small systems, spline cutoff.

The spline cutoff method derives from the simplest possible way of reducing the scaling of the nonbond calculation: ignoring all interactions between atoms farther apart than a certain distance. Since this leads to a discontinuity in the energy function at the cutoff distance, a cubic spline function is used to smooth the energy in that region. Typical distances used for small systems are 8.0 Å for the inner radius of the spline, where the energy is equal to the unmodified energy, and 8.5 Å for the outer radius of the spline, where the energy has been reduced to zero.

Given a constant density of atoms, the number of atoms within the cutoff radius is approximately constant, thereby making the scaling of the spline cutoff method $O(N)$.

The asymmetric unit of the human rhinovirus-14 protein coat was used as a test case. The structure was obtained from the Brookhaven Protein Data Bank (file 4RHV). The forcefield parameters and atomic charges were obtained from AMBER [1]. This system contains 8,530 atoms, including crystallographic water molecules.

Method	Energy (kcal/mol)	Error (kcal/mol)	Rel. Error (%)	Max. Force Error (kcal/mol/Å)	RMS Force Error (kcal/mol/Å)
Exact	-1.44298×10^4				
Spline-Cutoff	-1.39980×10^4	+431.8	+3.1%	40.17	7.06
CMM/center	-1.43682×10^4	+61.6	+0.4%	2.68	0.34
CMM/centroid	-1.44097×10^4	+20.1	+0.1%	1.99	0.28

Table 7-1. Accuracy of energy and force calculations.

Table 7-1 presents the energy and force results for the various methods. The exact calculation was performed using all 72,752,370 pairwise interactions in the system. The spline cutoff method used 8.0 Å and 8.5 Å inner and outer distances. The CMM used a 128 Å cube bounding box and a tree depth (maximum level) of 5, resulting in 3.9 atoms per occupied leaf cell. The multipole expansions were truncated at the level of quadrupoles.

The results show that for this representative system, the CMM outperforms the traditional spline-cutoff method by an order of magnitude in all categories. Of particular importance is the much smaller force error.

Using cell centroids instead of geometric centers significantly improves the energy, with a small additional improvement to the force errors, at virtually no additional cost.

Finally, the CMM is also much faster than the spline cutoff method, even with its improved accuracy. On a single (KSR-1, 20 MHz) CPU, the test case required 84.2 sec to set up and 28.1 sec to calculate using spline cutoff; the CMM required only 14.3 sec to set up and 6.0 sec calculation time. Since both methods are approximately $O(N)$, this ratio of times should scale to larger systems as well.

From a purely numerical standpoint, then, the CMM is far superior to spline-cutoff. There is one small disadvantage to using the CMM, however: it is not guaranteed to produce forces which satisfy Newton's Third Law. Since atoms that are distant from one another interact only through fields, and since those fields themselves are only approximate representations of the effects of groups of atoms, the forces generated may not be decomposable into a set of pairwise, equal and opposite forces. It is interesting to note that the Ewald sum commonly used for handling systems with periodic boundary conditions also uses fields and hence might produce non-Newtonian forces. The spline-cutoff method always deals with pairs of atoms and so must rigorously satisfy the Third Law.

Such errors in the forces have been observed to produce three effects. First, an artifactual net force and net torque may be applied to the system. Second, integration of the net force may lead to a net velocity, which can appear as a directional flow in the system. Third, errors in the velocities can contribute to the system's kinetic energy, which in turn affects the total Hamiltonian.

Code has been added to the program to remove all three of these effects, as desired by the user. Net translational forces are removed by subtracting a small corrective force vector equal to the net force divided by the number of atoms from each atom in the system. Similarly, net velocities can be removed by subtracting (mass-weighted) corrective velocity vectors from each atom. Finally, an experimental strategy for rescaling the system velocities to produce a rigorously conserved Hamiltonian was implemented.

Rather than instituting such *ad hoc* corrections, however, the user can also increase the accuracy of the computed forces by reducing the interval between farfield updates or by reducing the timestep of the simulation. We have found that a farfield update interval of 5 works well for most systems, with a reduction to 2 being necessary in a few cases. Performing farfield updates every step virtually eliminates the Hamiltonian drift in almost all cases.

7.3. Timing

Time to solution for the message-passing code is currently rather poor due to lack of optimization of various parts of the communication routines. A naive implementation of active messages on current hardware produces many small messages, each with considerable latency (ironically, the exact opposite of the intended effect of active messages on future hardware). To overcome this, we can bundle together multiple messages that are destined for the same node. Analyzing how best to do this is an ongoing research project.

With the current version of the code, a 1 million atom argon cluster system (calculating only nonbonded forces) takes approximately 35 sec per timestep on all 512 nodes of the Intel Paragon, using a farfield update

frequency of 5. A 10 million atom system, the largest run to date, takes approximately 330 sec per timestep on all 512 nodes.

The KSR version, in contrast, does substantially better in performance per CPU. On a 1 million atom virus dimer, including all valence forcefield terms, we obtain a time of 64.7 sec on 60 CPUs, or about four times the performance of the Paragon code.

The dynamic load balancing implemented in the KSR code can have a substantial effect on the timing. On a very small, 463 atom system running on 4 nodes at CMM level 2, we see an improvement in the farfield computation time of 13% (from 55 ms to 48 ms) due to the reduced load imbalance.

7.4. Scalability

Each of the steps comprising the CMM is linear and scalable, or nearly so.

There are seven steps in the CMM; these may be divided into two major parts. The five steps of the first part compute the farfield (the Taylor series expansions representing the field from atoms far away from each atom), while the two steps of the second part compute the nearfield (the explicit calculation of effects due to atoms near each atom).

The first step, generation of the leaf cell multipoles, is fully linear and runs in parallel since there are no data dependencies.

The second and third steps, computation of the cell centers and propagation of the multipoles upward through the tree, both require a traversal of the octree. Since the number of cells in the system is the sum of a geometric series with a logarithmic number of terms, it is essentially proportional to the number of atoms.

$$\begin{aligned}
n_{leaves} &= n_{atoms} / \kappa \\
n_{levels} &= \log_8 n_{leaves} \\
n_{cells} &= \sum_{i=0}^{n_{levels}} 8^i = \frac{8n_{atoms}}{7\kappa} - \frac{1}{7}
\end{aligned} \tag{1}$$

where κ is the number of atoms per cell at the finest level, a constant.

Each pass through the tree, whether upward or downward, involves a constant number of computations per cell and therefore is linear in the number of atoms in the system. The tree traversals cannot be made fully parallel, however, as there are increased data dependencies near the root of the tree. On the other hand, since the number of computations to be done near the root is relatively small, due to the high degree (8) of the octree, the tree traversal time is dominated by the computations near the leaves, which are highly parallel.

The fourth and fifth steps, the PNC computation and the propagation of the Taylor series coefficients downward through the tree are also linear and highly parallel as argued above.

The two steps of the nearfield computation (computing explicit interactions with atoms in the same cell and with atoms in neighboring cells) are perfectly linear and can also execute in a parallel fashion, limited only by the communications overhead of transmitting atoms from leaf cells to their neighbors.

Finally, the dynamics step has only one data dependency, a global sum to determine the overall kinetic energy, with the rest being perfectly parallel and linear.

The valence computations are essentially linear in the number of atoms, since each atom is only connected to a limited number of other atoms and can thus participate in only a limited number of valence interactions. In

the message-passing code, we assume that no additional communications will be required to compute the valence interactions; this should also hold true for the KSR code, since the valence computations occur after the nearfield step in which neighboring atoms are accessed and brought into the processor's cache.

The total amount of computation that occurs is thus linear in the number of atoms. Nonlinearities in the scaling of computation with number of CPUs are the result of load balancing inefficiencies, which lead to waiting at global synchronization points.

The total amount of communication that is required is almost linear in the number of atoms, except for the tree effects described above. The fraction of this communication that occurs off-node, however, varies depending on the number of CPUs used, and will in general vary as the total surface area of the boundaries between cells assigned to each CPU, which is approximately $n_{atoms}^{2/3} n_{cpus}^{1/3}$. Further complicating the analysis, though, is the fact that much of this communication can itself occur in parallel. The amount of communication can also be decreased by taking into account the fact that an atom or PNC may need to interact with multiple cells on the same destination node. This avoidance of redundant transmissions has been implemented in the message-passing code for the PNC multipole communication step, but not yet for the nearfield atom communication step. On the KSR, this redundancy is automatically eliminated because the data is cached on the destination node.

The best case time is thus

$$t_{timestep} = C + t_{comp} n_{atoms} / n_{cpus} + t_{comm} \quad (2)$$

while the worst case time is

$$t_{\text{timestep}} = C + t_{\text{comp}} n_{\text{atoms}} / n_{\text{cpus}} (1 + k_{\text{loadbal}}) + t_{\text{comm}} n_{\text{atoms}}^{2/3} n_{\text{cpus}}^{1/3} \quad (3)$$

where C is constant setup overhead, t_{comp} is the computation time per atom, t_{comm} is the communication time per cell, and k_{loadbal} represents the overhead due to imperfect load balancing.

The message-passing program was tested for performance on a series of multi-million atom argon cluster systems. Although these systems do not include Coulombic charges and their interactions, all Coulomb terms were still calculated (and correctly resulted in zero energy and zero force) and hence are included in the timing results. The calculations were run on the CSC Paragon XP/S using OSF/1 Release 1.0.4. Five cluster sizes were used: 1 million, 2 million, 5 million, 8 million, and 10 million atoms.

For a constant number of atoms, if we plot the logarithm of the time versus the logarithm of the number of CPUs, we will ideally get a line of slope -1 by equation 2. As n_{cpus} gets large, the slope should level off and eventually begin increasing to a value of at most $1/3$ as in equation 3, assuming that imperfect load balancing does not depend much on the number of CPUs used.

Figure 7-1 shows such a graph of $\log(\text{time})$ against $\log(\text{CPUs})$ for the farfield Taylor series generation process. The number of CPUs along the X axis ranges from 64 to 512. Three lines are drawn to show the scaling for systems of different sizes ranging from 1 million to 5 million atoms. The 8 million and 10 million atom systems could only be run on all 512 CPUs. The thick line shows the slope that would be achieved for ideal (perfectly linear) scaling.

This portion of the calculation contains all of the tree manipulations. The effects of the data dependencies inherent in the tree, which cause

imperfect parallelization, can be seen by the less negative slope of the lines, especially for the smallest, 1 million atom system. Larger systems, in which the amount of computation per node increases, show better scaling which is more nearly parallel to the ideal line. Note that we would only reach the regime of zero or positive slope in pathological cases with much too little computation for the amount of communication required (i.e. too few atoms spread across too many CPUs).

Note that the farfield computation is only performed at intervals, so its imperfect scaling has a relatively small effect on the overall time to solution.

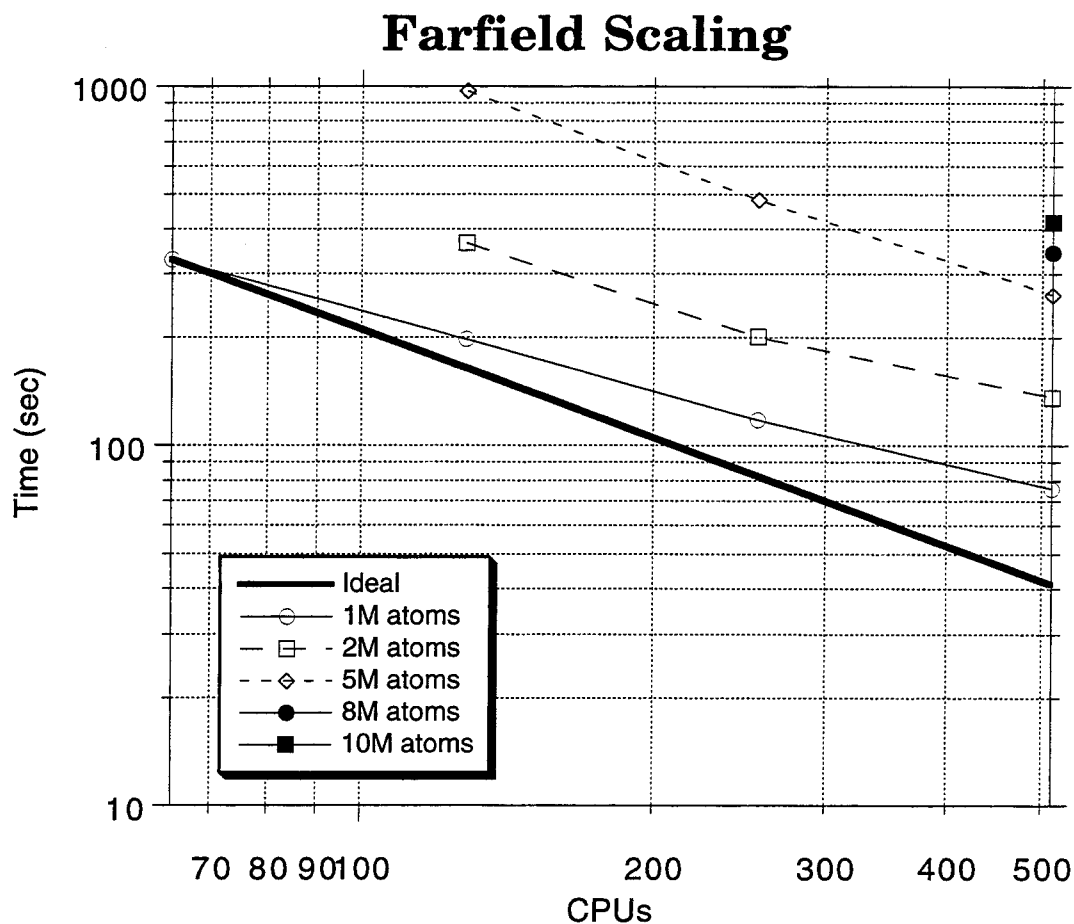


Figure 7-1. Scaling vs. number of CPUs for farfield computation.

Figure 7-2 shows the same type of graph, but for the nearfield and integration computations. This portion of the calculation contains no tree-derived data dependencies and its scaling curves are close to parallel with the ideal line.

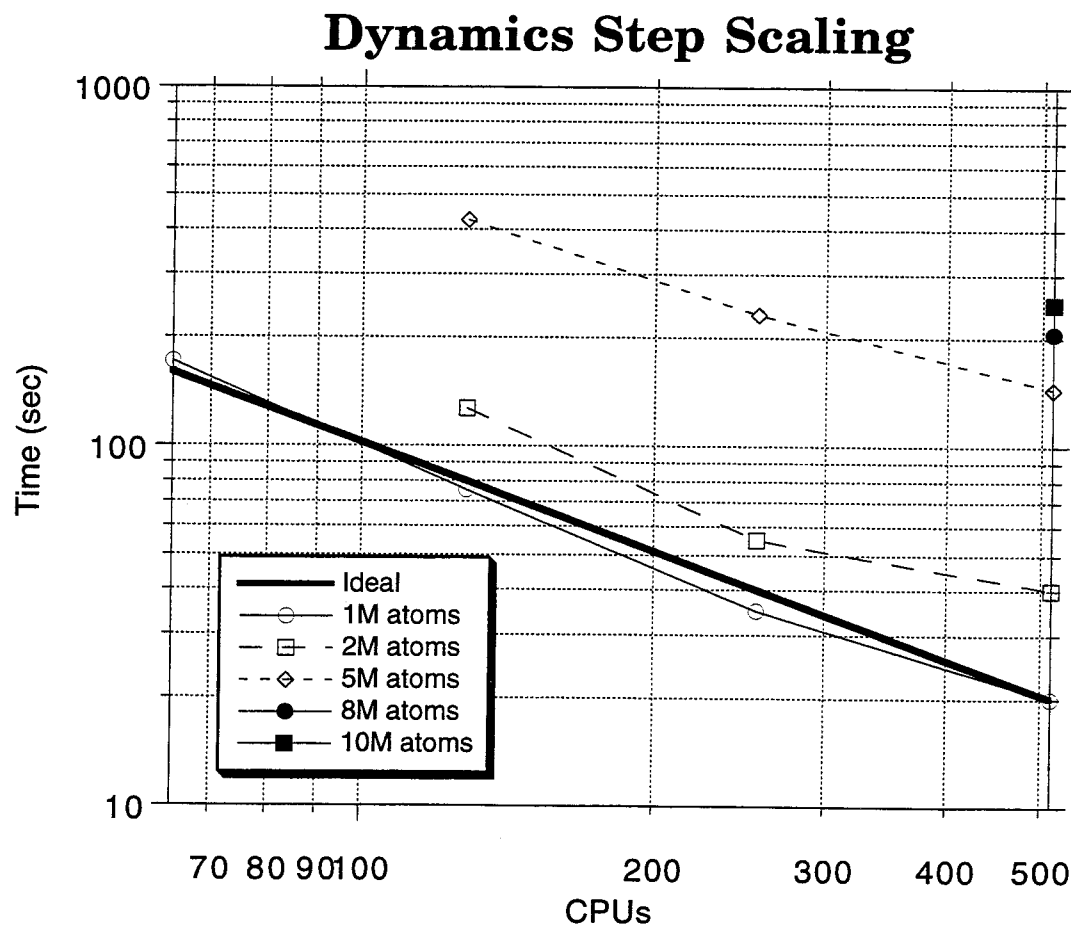


Figure 7-2. Scaling vs. number of CPUs for nearfield/integration steps.

For a constant number of CPUs, if we plot the time per atom versus the number of atoms, we should ideally get a constant. Deviation from the constant line should be most apparent at small numbers of atoms, since the deviation is expected to scale as, at worst, $n_{atoms}^{-1/3}$. These graphs are shown in Figure 7-3 for the farfield computation and Figure 7-4 for the nearfield and

integration steps, in which the X axis is the number of atoms in the simulated system in millions and the Y axis is the time spent in the indicated portion of the code divided by the number of atoms.

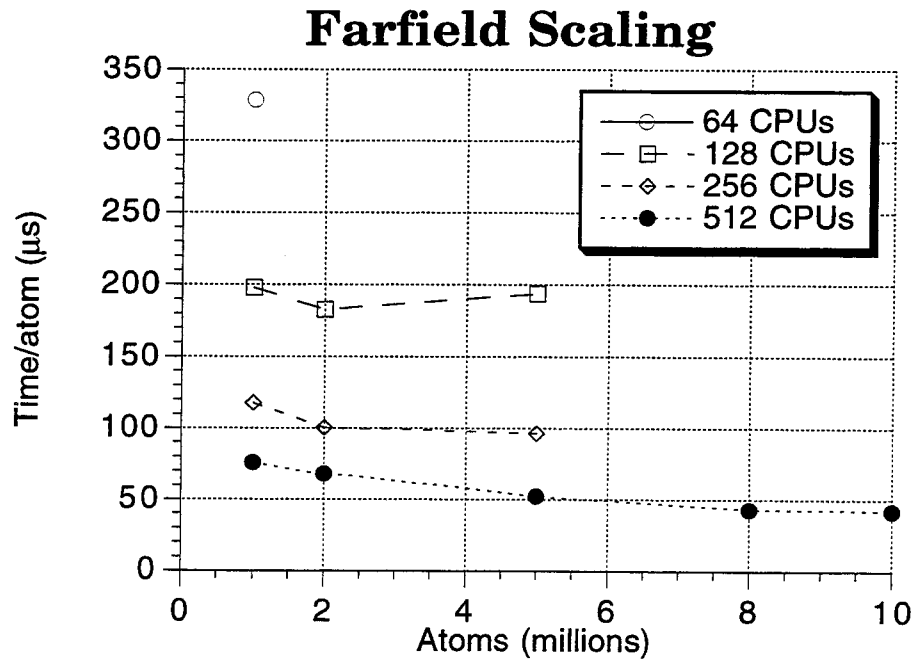


Figure 7-3. Scaling vs. number of atoms for farfield computation.

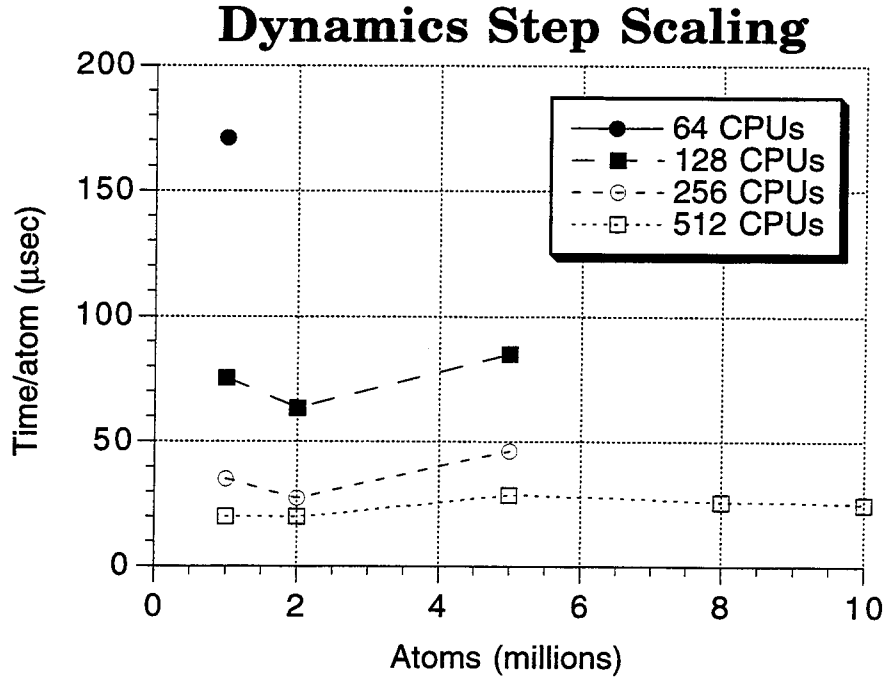


Figure 7-4. Scaling vs. number of atoms for nearfield/integration steps.

We see that the lines for 128, 256, and 512 CPUs are close to flat, with the expected upturn for the farfield computation at small system sizes. To show that this is in fact due to the communications overhead, as described in the theoretical scaling formula (equation 3), we can plot the time per atom against the number of atoms to the $-1/3$ power. This should give lines with a slope of zero in the best case, or $t_{comm} n_{cpus}^{1/3}$ (a positive constant) in the worst case. In Figure 7-5, the lines of zero or constant positive slope show that the imperfect scaling for the farfield computation is in fact due to the communications term in the theoretical scaling formula.

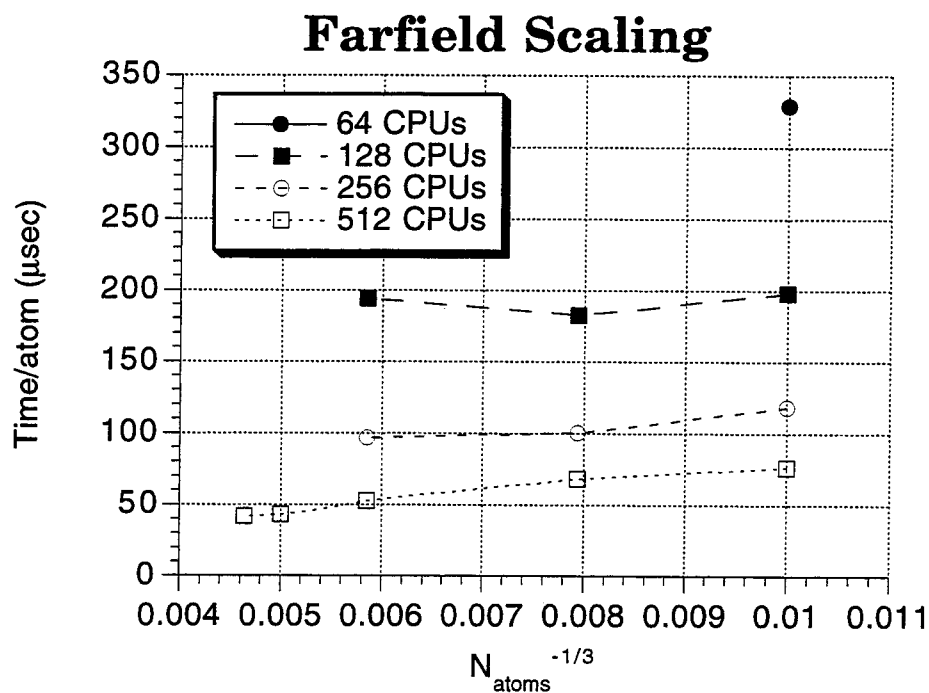


Figure 7-5. Scaling vs. $n_{atoms}^{-1/3}$ for farfield computation.

On the KSR, we get close to linear scaling with both CPUs and atoms. Inverting equation 2 above, we can compute an expected value of t_{comp} from the time per timestep, the number of CPUs, and the number of atoms. Doing so gives the approximately equal values in the following table for a viral system, including valence forcefield interactions:

Atoms (millions)	0.5	1.0	1.0	0.5	1.0
CPUs	30	30	45	60	60
Time (CPU-ms/atom)	3.07	3.55	3.35	4.39	3.98

Table 7-2. Scalability for KSR code.

For comparison, fitting lines to the above graphs for the message-passing code gives t_{comp} values ranging from 5.6 to 18 CPU-ms/atom for the farfield step and 12 to 18 CPU-ms/atom for the nearfield and dynamics step.

References

1. Weiner, S.J.; Kollman, P.A.; Case, D.A.; Singh, U.C.; Ghio, C.; Alagona, G.; Profeta, S.; Weiner, P. *J. Am. Chem. Soc.*, **106**(3), 765 (1984).

PART III — APPLICATIONS

Chapter 8. TVN Dynamics

8.1. Introduction

Questions exist as to whether constant-temperature, constant-volume (TVN) dynamics properly reproduces physical properties of systems. Although the equations of motion guarantee that a proper canonical distribution will be generated eventually, the primary concern is whether the dynamics reaches ergodicity, and thus an adequate sampling of the distribution, in a reasonable amount of time.

The Nosé-Hoover formulation has one free parameter, τ_s . This variable controls the rate of transfer of kinetic energy from the system to the bath and vice versa and thus the rate of equilibration. There have been no clear guidelines as to how this parameter should be chosen [1].

We investigated how τ_s affects dynamics in a wide variety of systems, including argon and methane clusters and periodic poly(ethylene) models.

8.2. Procedure

Five structures were generated for testing. Argon atoms were arranged in a Mackay icosahedron [2] of five shells (561 atoms). An additional argon system was built using a 256 atom face centered cubic (fcc) unit cell, 21.6204 Å on a side. A methane cluster was built by replacing each argon atom in a three-shell Mackay icosahedral structure with a methane molecule; the resulting cluster of 735 atoms was then minimized using BIOGRAF [3]. Two poly(ethylene) (PE) systems were also built. One was an infinite system composed of a minimized single chain fragment of 398 atoms in an 18 Å cubic unit cell. The ends of the chain fragment were connected

through the unit cell boundary to form the infinite chain. The other PE system was a 2x6x4 supercell of 576 atoms (16 chain fragments, each of 6 monomer units).

The argon systems are a very simple, finite cluster with only heavy atoms and weak forces and an extension of that system to periodic boundary conditions. The methane system is similar to the argon cluster, but it also includes high-frequency C–H bonds. The PE cases have both periodic boundary conditions and high-frequency bonds, first using only one chain and then with inter-chain interactions.

Each system was simulated for 100 ps, using 1 fs timesteps. τ_s values between 0.01 ps and 1.0 ps were used. The bath temperature was set to 20 K for the argon and methane cases; it was fixed at 300 K for the PE systems.

The kinetic energy for each run was plotted against time to observe how quickly the system converged to the desired temperature. In addition, the kinetic energy distribution was computed by dividing the range of kinetic energies into 100 bins and counting the number of values falling into each bin. In the ideal case, these distributions should be of Gaussian form [1].

8.3. Results

The kinetic energy distributions for various values of τ_s are plotted in Figures 8-1 through 8-5. The argon cluster results were essentially the same as those for the fcc argon system and are not shown; the single-chain PE results similarly were essentially identical to the multi-chain PE results and are omitted.

Two particular features are notable. At longer values of τ_s , the distributions have two distinct peaks and are often skewed so that the lower-

energy peak is sharper and taller. For the argon systems, the distributions for very short values of τ_s in Figure 8-3 also show double-peaked behavior.

The first feature is explained by looking at sample plots of the kinetic energy versus time in Figures 8-6 through 8-12. The system is coupled to the bath via what is essentially a harmonic oscillator of period $2\pi\tau_s$. For long values of τ_s (Figures 8-7, and 8-9), relatively few cycles of this oscillator have occurred by the end of the 100 ps simulation. The oscillator frequency is outside the range of the modes of the system, so transfer of energy from the system to the bath occurs only through the anharmonicities of the system's oscillators. This process is slow. Thus, by the end of the simulation the system is still equilibrating and has not yet reached ergodicity. The resulting distribution is dominated by the turning points of the Nosé oscillator, where the system spends the most time.

At shorter values of τ_s (Figures 8-6, 8-8, and 8-10), the oscillator frequency couples well with the modes of the system. Energy is transferred easily between the system and the bath, and ergodicity is reached more quickly, both in terms of simulation time and in terms of the number of oscillator periods.

At very short values of τ_s , as seen in the argon cases (Figure 8-3), the distributions again become double-peaked. Note, however, that the range of kinetic energies in these cases is much smaller than for long τ_s . Again, reference to the kinetic energy versus time plots (Figure 8-9) shows that the system is now highly constrained to a very limited range of KE values. In these cases, with heavy atoms, energy is transferred too rapidly from the system to the bath or vice versa, and the system has no opportunity to reach ergodicity.

The short τ_s cases for argon were run with an unusually small timestep of 1 fs. Using a more typical longer timestep caused numerical instabilities as the Nosé oscillator failed to be properly integrated.

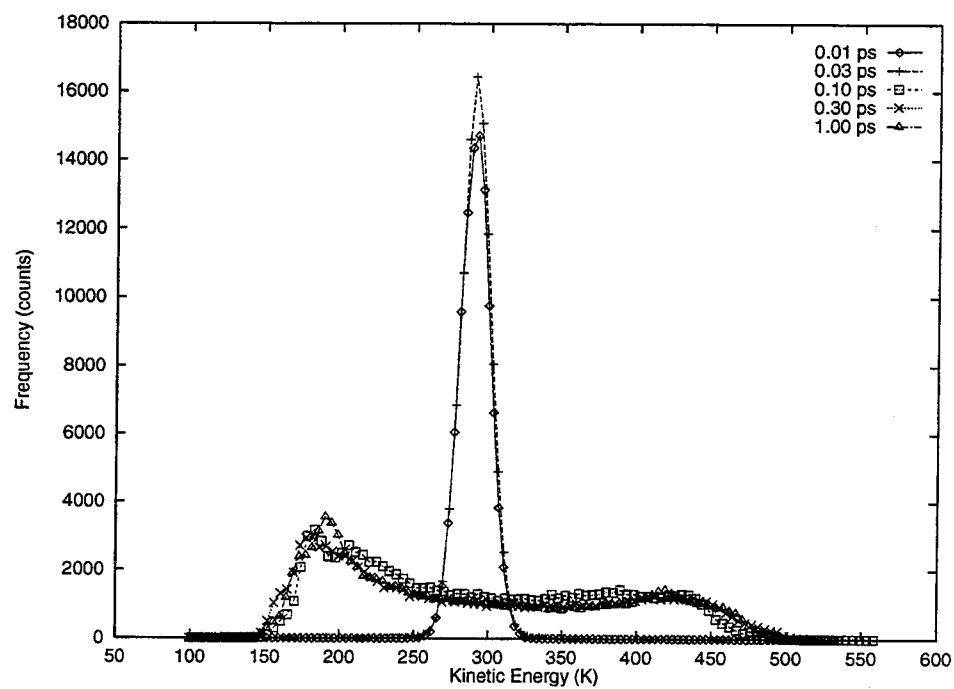


Figure 8-1. KE distribution for multi-chain PE crystal for varying τ_s .

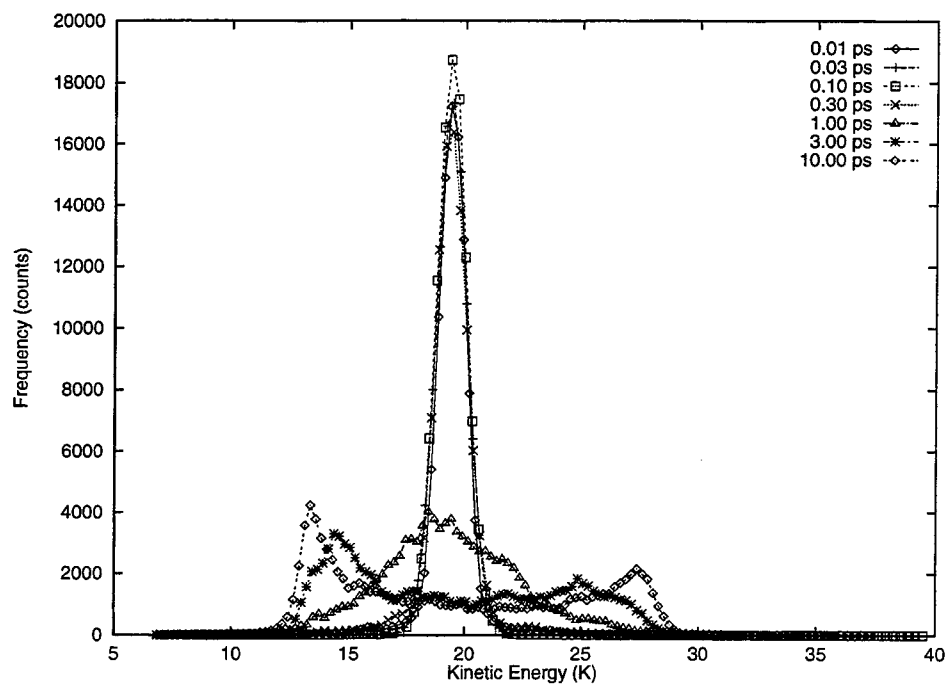


Figure 8-2. KE distribution for methane cluster system for varying τ_s .

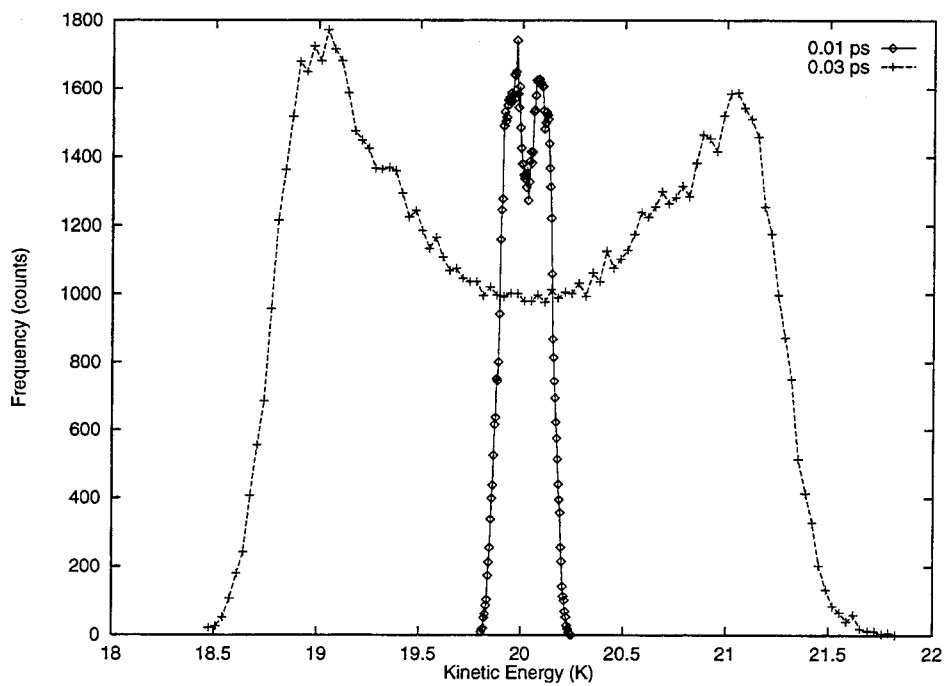


Figure 8-3. KE distribution for fcc argon with small τ_s .

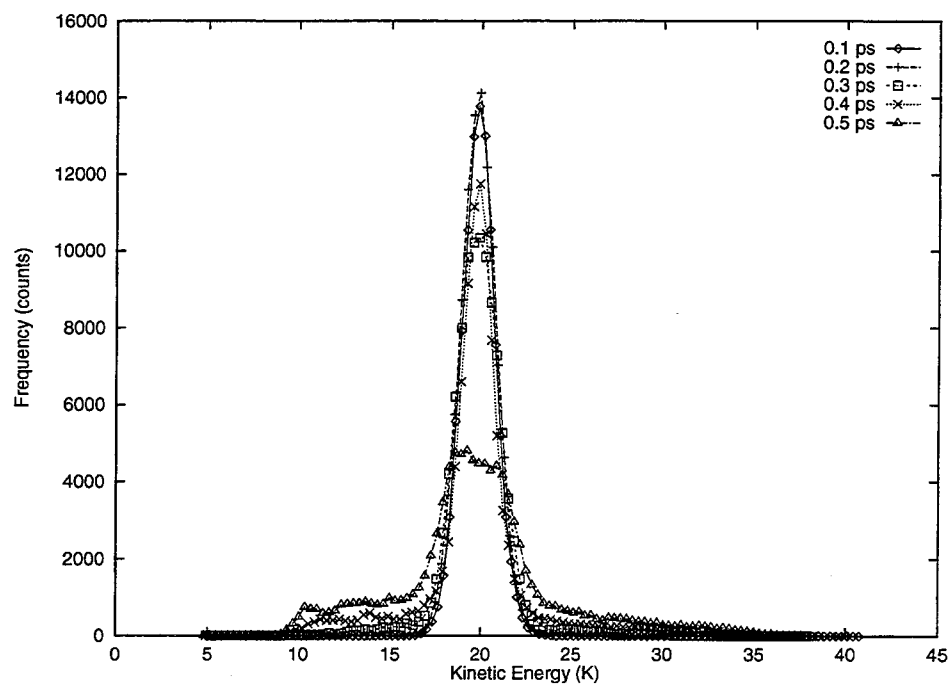


Figure 8-4. KE distribution for fcc argon with intermediate τ_s .

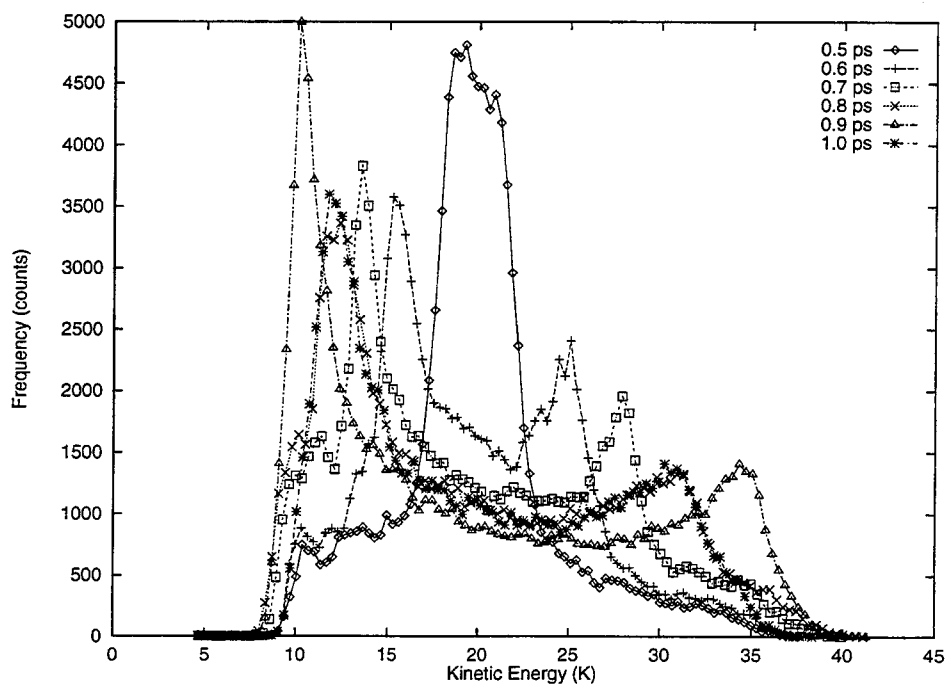


Figure 8-5. KE distribution for fcc argon with large τ_s .

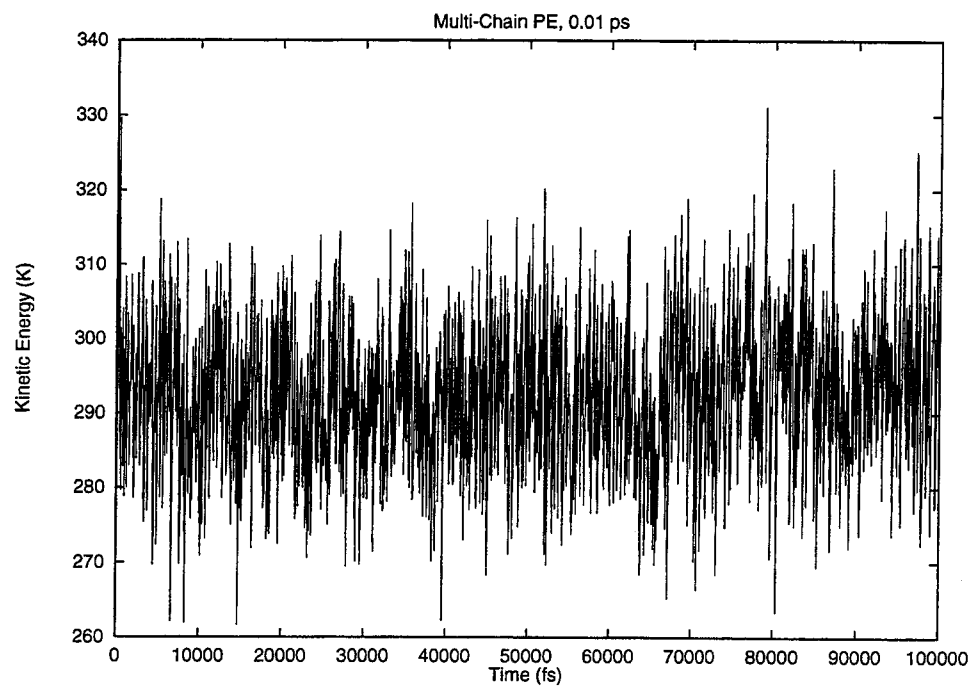


Figure 8-6. KE vs. time for multi-chain PE crystal with small τ_s .

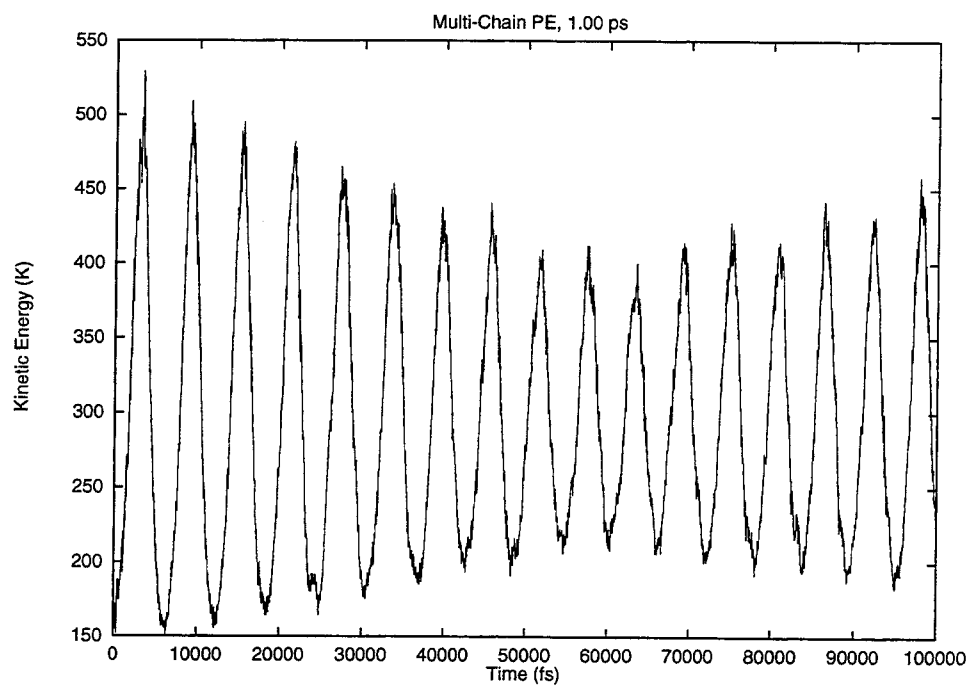


Figure 8-7. KE vs. time for multi-chain PE crystal with large τ_s .

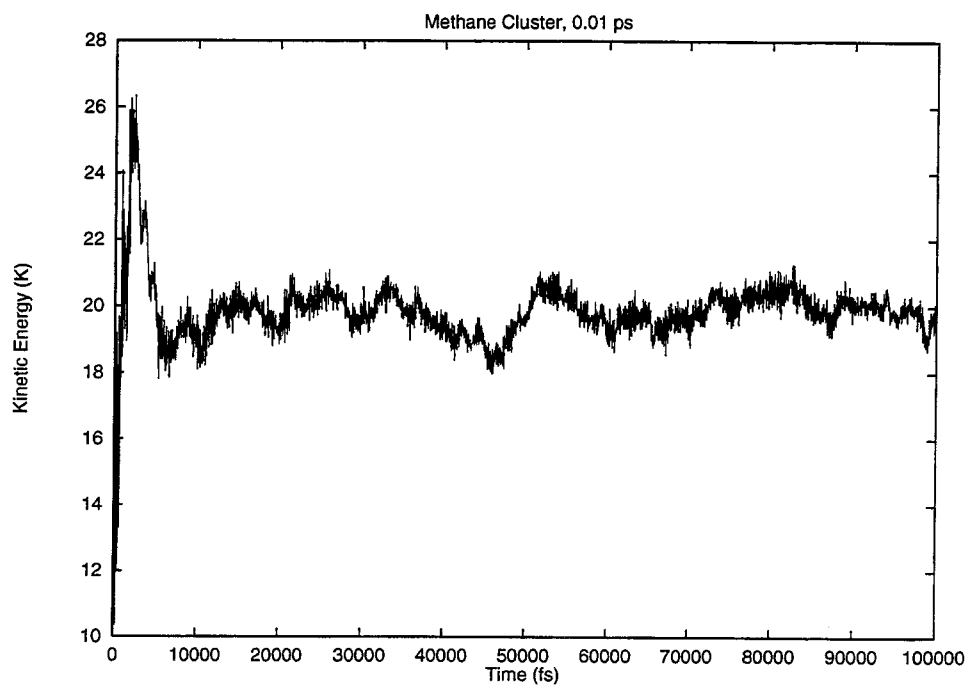


Figure 8-8. KE vs. time for methane cluster with small τ_s .

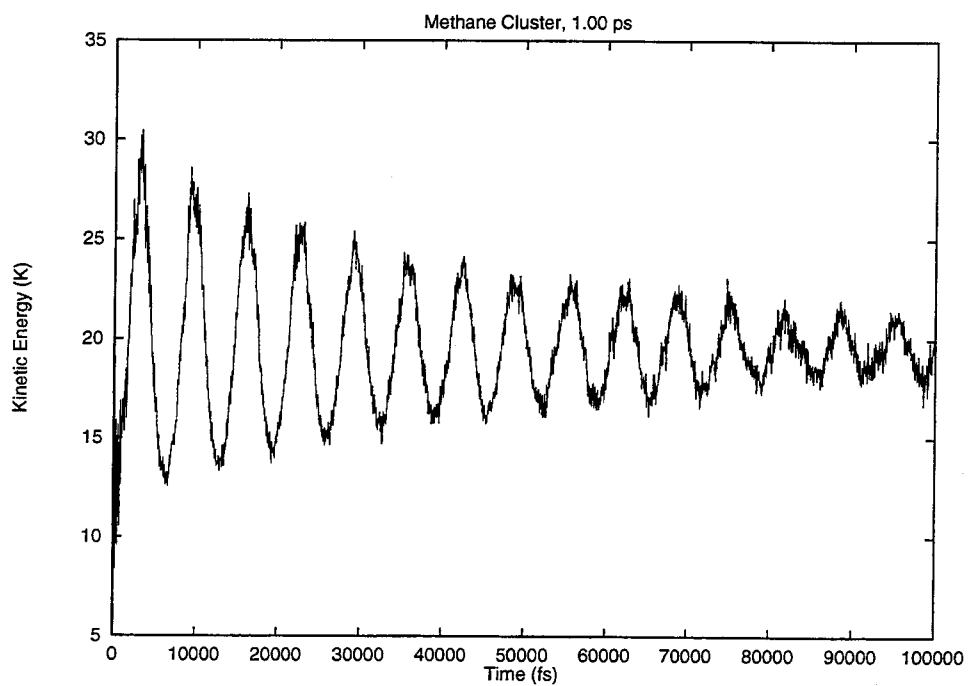


Figure 8-9. KE vs. time for methane cluster with large τ_s .

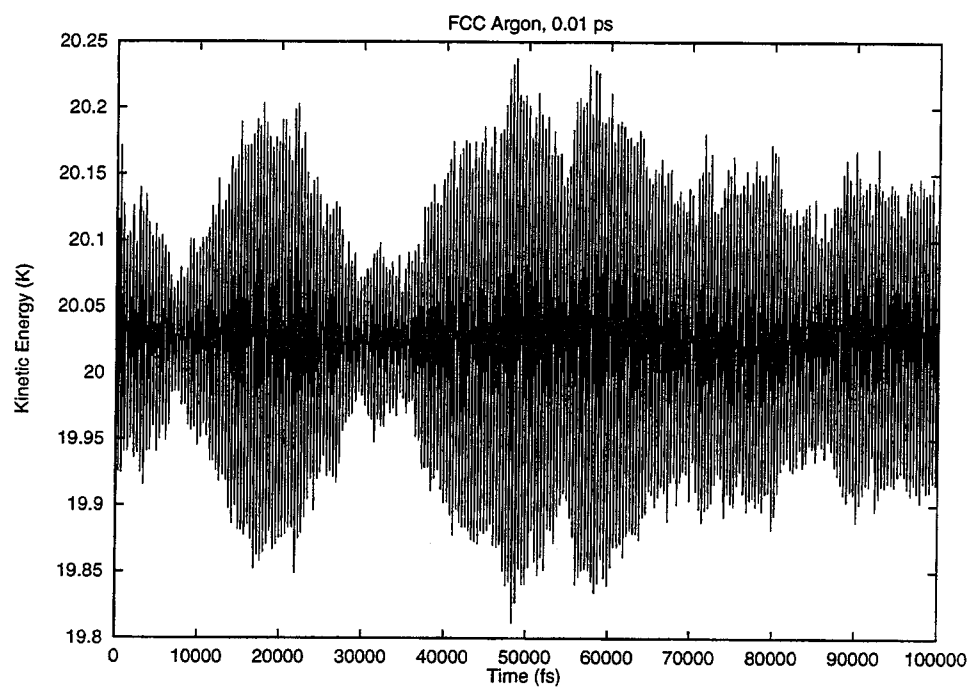


Figure 8-10. KE vs. time for fcc argon with small τ_s .

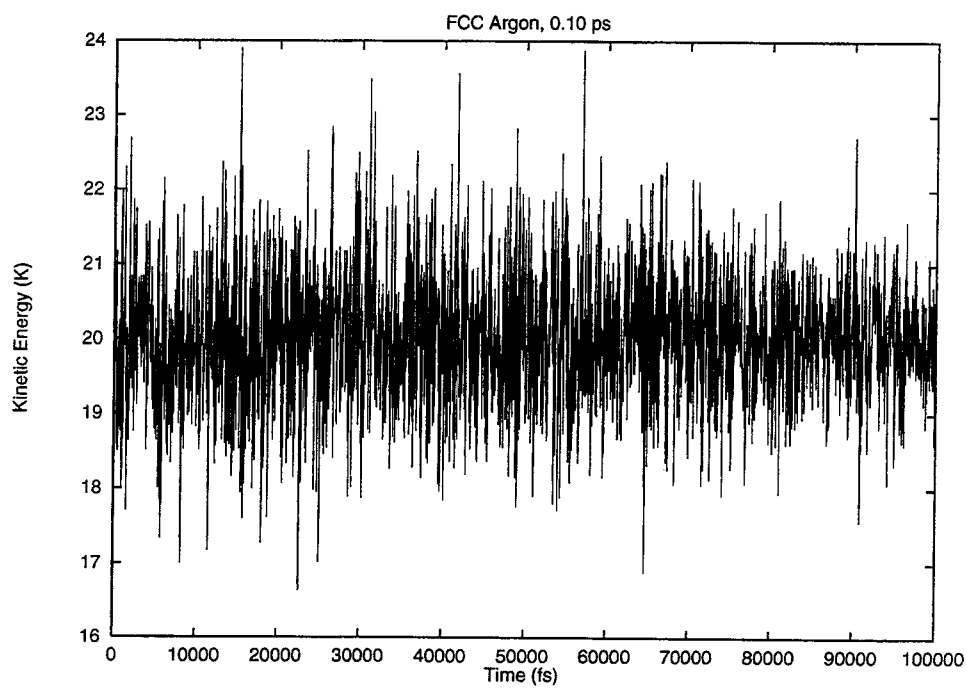


Figure 8-11. KE vs. time for fcc argon with intermediate τ_s .

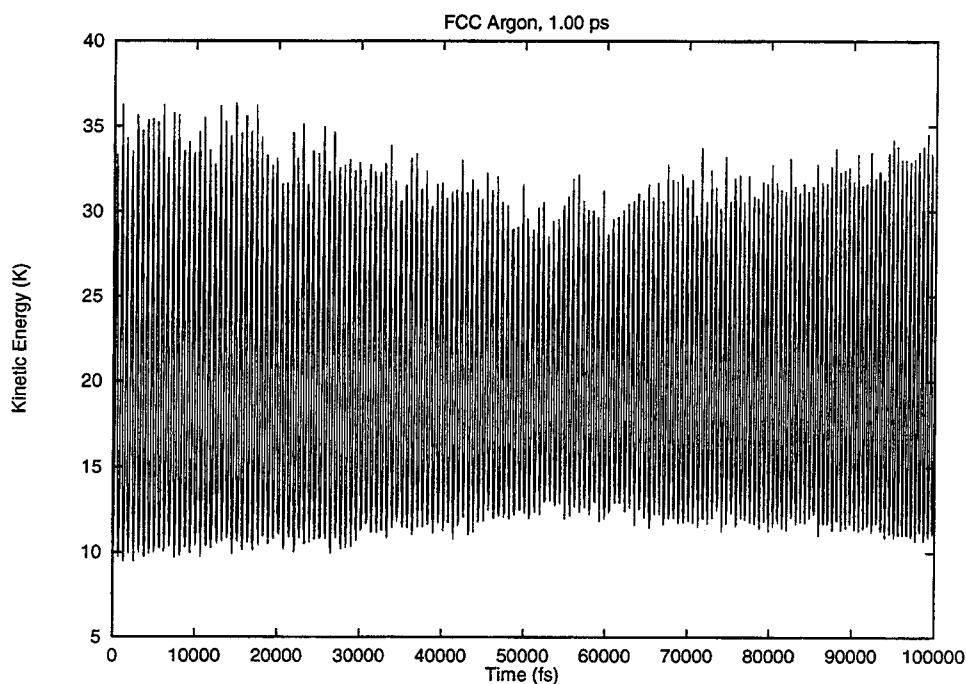


Figure 8-12. KE vs. time for fcc argon with large τ_s .

8.4. Conclusions

Since rapid convergence is almost always desired over slower convergence, the shortest possible τ_s should be used. From the argon cases, it is apparent that there is a limit to how short this can be, however. Between the twin considerations of integrability of the Nosé oscillator and allowing the system to vary somewhat in KE to maintain ergodicity, a suitable choice of τ_s appears to be 10 times the normal dynamics timestep, or 0.01 ps for typical systems involving hydrogen atoms. Using such a value, convergence to the desired temperature can be achieved in a few hundred timesteps.

References

1. Nosé, S. *Prog. Theor. Phys. Suppl.*, **103**, 1 (1991).
2. Mackay, A. *Acta Cryst.*, **15**, 916 (1962).

References (cont.)

3. BIOGRAF version 3.21, Molecular Simulations, Inc.

Chapter 9. Argon Clusters

9.1. Introduction

Argon clusters have long been studied to determine the structures of van der Waals aggregates. For small clusters, certain “magic numbers” [1] of atoms have been found which lead to more stable structures than clusters with even one more or one fewer atom. The most stable structure for these clusters is typically the Mackay icosahedron [2].

Bulk solid argon, however, is known [3] to be most stable as a face-centered cubic structure, which is not compatible with the Mackay icosahedral symmetry. To investigate the transition between the small finite cluster behavior and the bulk structure, we modelled a “magic number” cluster of about five million argon atoms using a quenched dynamics process.

9.2. Procedure

We began with a Mackay icosahedral structure of 114 shells, comprising a total of 5,003,879 atoms. The forcefield used was a simple Lennard-Jones 12–6 potential with equilibrium radius (R_e) of 3.82198 Å and well depth (D_e) of 0.23725 kcal/mol, chosen to agree with the bulk lattice spacing and the 0 K heat of vaporization (after correcting for zero-point energy). CMM level 8 was used with a bounding cube 1440 Å on a side; the farfield was updated every 5 timesteps. The initial potential energy of the starting structure using this potential was -9.20×10^6 kcal/mol.

We performed 400 steps (4 ps) of Nosé-Hoover constant-temperature (NVT) dynamics at 80 K with a Nosé time constant τ_s of 0.1 ps. The kinetic energy of the system rapidly converged to the desired temperature, which

was selected to be close to the melting point of bulk argon (at 1 atm), thereby allowing melting of the initial structure while limiting boil-off of surface atoms from the cluster.

After the dynamics run, we quenched the system using dynamics at a temperature of zero K, with all velocities removed after each integration step. Adequate convergence was achieved after 1350 steps. The final energy was -9.29×10^6 kcal/mol, nearly 1% lower than that of the starting structure, indicating that a more stable structure was found. The RMS force on the final structure was 5×10^{-4} kcal/mol/Å.

9.3. Results

The final structure was analyzed by comparing simulated diffraction structure factors computed from it to those computed from the initial Mackay icosahedral structure. To look at gross structural aspects, we computed the cluster structure factor (the magnitude of the Fourier transform of the atomic positions) for $0 \leq h \leq 10$ and $-10 \leq k, l \leq 10$, using the CMM bounding box as the unit cell. Certain peaks show significant changes, as much as several orders of magnitude, from the initial to the final structure. For example, the peak at (0,7,8) decreases in intensity by a factor of 22, while (2,10,10) increases by a factor of 752.

Sample contour plots of the logarithms of the structure factors for the initial and final structures are shown in Figures 9-1 and 9-2. Both figures show sections through the $h=2$ plane.

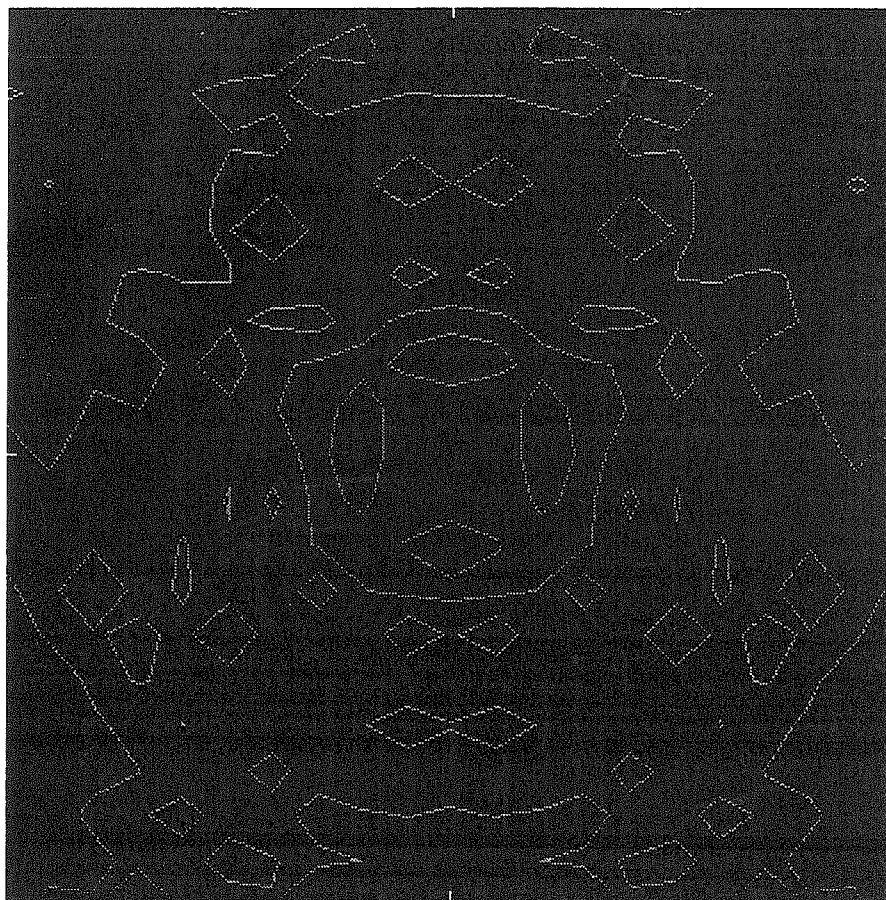


Figure 9-1. $h=2$ section of $\log(\text{structure factor})$ for initial argon cluster.

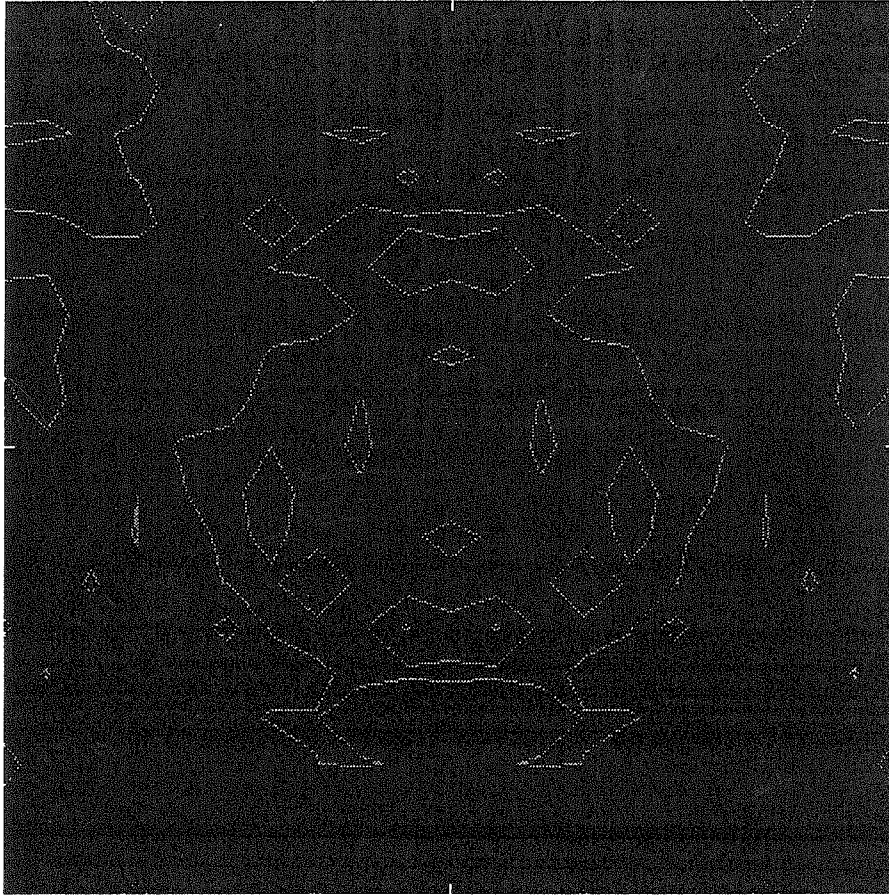


Figure 9-2. $h=2$ section of $\log(\text{structure factor})$ for final argon cluster

Additional comparisons were made with the energies of smaller, optimized Mackay icosahedra and fcc spheres. A number of small (55 to 3925 atom) clusters were generated. Each cluster was minimized for 1000 steps, which brought all of them essentially to convergence. All energies were computed using exact nonbonds. The potential energy is expected to contain two components: one that varies with the number of atoms (the binding energy) and one that varies with the amount of surface area, or the number of atoms to the $2/3$ power (the surface energy). The energy per atom should then be close to linear in the number of atoms to the $-1/3$ power, and this is in fact observed in Figure 9-3 for both the Mackay icosahedra and fcc spheres.

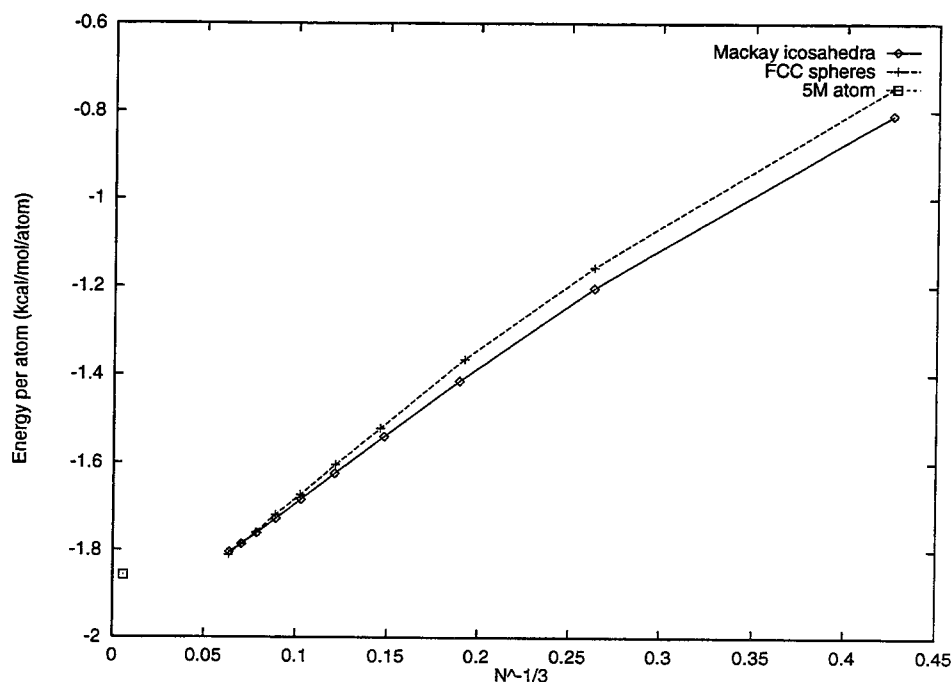


Figure 9-3. Comparison of potential energies for argon clusters.

The curves appear to cross between 2057 and 2899 atoms, indicating that the fcc structure is more stable after this point. The 5 million atom structure, however, has an energy per atom which, though more negative than all of the smaller clusters, is well above the curves. This would seem to indicate that the structure is in fact not a global minimum for that number of atoms.

9.4. Conclusions

We have shown that molecular dynamics and minimization calculations on systems as large as 5 million atoms are feasible given current software and hardware technology.

The results of these simulations on argon clusters suggest that, even at the achieved local minimum, interesting structural changes are occurring, as

evidenced by the changes in peaks in the structure factor. Further analysis of these changes will be necessary.

Although some visualization of these clusters has been performed, bringing to bear the substantial capabilities of the human visual system on the analysis of these structures will require more advanced visualization tools that permit the user to select portions of the cluster to view, rather than overwhelming the limited screen resolution with all 5 million particles.

The primary limitation on improving the search for a global minimum for the 5 million atom cluster was the available minimization technology. Dynamics at 0 K has relatively poor convergence properties, particularly when compared with standard methods such as conjugate gradient minimization. Installation of such improved minimizers in the code is a high priority task.

References

1. Herzog, R.F.K.; Poschenrieder, W.P.; Satkiewicz, F.G. *Radiat. Effects*, **18**, 199 (1973).
2. Mackay, A. *Acta Cryst.*, **15**, 916 (1962).
3. Donohue, J. "Structures of the Elements," R.E. Krieger, Malabar, FL, 1982.

Chapter 10. Surface Tension of Liquid H₂O and H₂O on PTFE

10.1. Introduction

Determining the surface tension of bulk liquids is a very difficult problem with standard small-cluster or infinite periodic boundary condition simulation methods. Large-scale molecular mechanics offers the opportunity to use large clusters which more accurately simulate bulk properties and thus promises to enable the prediction of surface tensions.

10.2. Drop Procedure

Water cubes from Jorgensen et al. [1] were used to build large supercells. In particular, a 216 molecule cube was replicated 8 times in each Cartesian direction to form a 110,000 molecule box.

A sphere of radius 50 Å was then extracted from the center of this large box. The sphere contained 17,254 molecules or 51,762 atoms. The initial sphere density was 0.94 g/mm³.

Microcanonical dynamics at 300 K was then performed on the system. A timestep of 1 fs was used, though longer timesteps should be feasible. An iterative rigid molecule procedure based on SHAKE [2] was used to constrain the bonds and angles in the water molecules. The temperature was rescaled periodically, every 10 fs. The CMM parameters used were a maximum level of 5 and farfield updates performed every timestep. The TIP3P forcefield parameters [3] were used to describe the water molecules.

Ideally, reflecting boundary conditions would be used to enable the formation of a vapor atmosphere around the drop. The performed

simulations, however, were for relatively short times and hence had relatively little boil-off of surface molecules.

10.3. Drop Results

After 2450 timesteps, the system temperature was stable and the total and potential energies were also essentially constant, indicating that equilibration was achieved.

Visualization of the system shows that the water drop has lost the periodicity derived from the initial supercell; the molecular positions have been thoroughly randomized. Some shrinkage has occurred. The RMS radius from the center of mass decreased from 38.69 Å to 35.38 Å. Using $R_{sphere} = \sqrt{\frac{5}{3}} R_{RMS}$, the computed radius of the final sphere is 45.68 Å, a shrinkage of about 10%. The resulting density is 1.29 g/mm³.

10.4. PTFE Simulation

Standard polymer simulations typically use one or a few chains or chain fragments, each composed of a handful of monomer units. These are then placed in a unit cell with periodic boundary conditions.

This approach has several significant limitations. Molecular weights are either very small or else infinite, both of which are unphysical. Interactions between a chain and the images of other chains, or even its own image, in a neighboring unit cell may lead to unphysical correlations.

Using large-scale molecular mechanics, a different model for bulk polymers may be generated that may better reproduce physical properties. Since thousands or millions of atoms may now be handled, molecular weights can be increased to the range of 10⁵ or 10⁶ typical of experimental values. In addition, the correlation problem may be avoided by using a very large, but

finite system. This has the disadvantage of introducing edge effects, but hopefully the behavior in the center of the system will accurately reproduce the bulk. Perhaps better still is to use a very large unit cell with periodic boundary conditions, reducing the correlation problem to negligible levels.

10.5. Procedure

A chain of 143 monomer units (431 atoms) was built in two forms: a left-handed helix and a right-handed helix. The helix parameters were obtained from experiment [4].

Sets of helices were then packed together into hexagonal structures using rotations (about the helical axis) and translations (both in the X-Y plane to generate the hexagonal lattice and in the Z axis). The lattice parameters, helical rotations, and Z axis translations were determined from experimental values. In figure 10-1, the left- and right-handed helices are depicted as circles with appropriate letters. The spacing between two helices of the same handedness in the vertical direction is 5.648 Å, while the spacing in the horizontal direction is 9.649 Å. If the left-handed helix at the bottom of the figure is assigned a Z coordinate of zero, then the right-handed helix above and to the left of it will be offset 0.15 Å in the positive Z direction (out of the plane of the paper). The Z offset between one helix and the next of the same handedness in the vertical direction, as indicated by the arrow, is 0.12 Å; the Z offset in the horizontal direction is 0.05 Å. These parameters are sufficient to allow a system composed of any number of helices to be built.

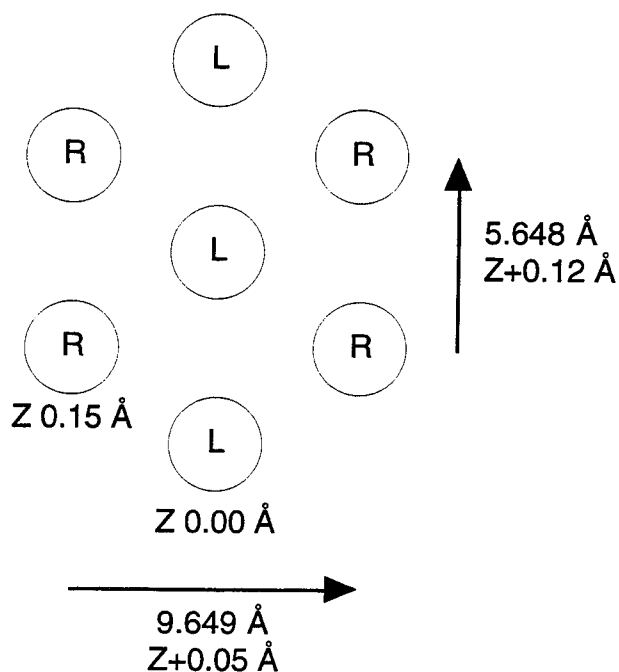


Figure 10-1. Experimental PTFE lattice geometry.

These hexagonal models expose three types of surfaces: one composed of the sides of helices having the same handedness (e.g. the right or left surfaces in Figure 10-1), one composed of the sides of helices having alternating handednesses (e.g. the angled surfaces in the figure), and one composed of the ends of the helices.

The sizes of the models used ranged from 7 chains (3000 atoms) to 2611 chains (1.1 million atoms).

Microcanonical dynamics was performed for 2.4 ps with a timestep of 2 fs. Since only heavy atoms are present, timesteps longer than the usual 1 fs are feasible for this system.

The forcefield used included accurate torsions with terms up to $\cos(12\phi)$ from quantum mechanical calculations on small model systems [5]. The charges used were also optimized based on simpler systems.

10.6. Results

The carbon backbones of the polymer chains were visualized to assess changes that occurred during the dynamics.

As expected, edge effects were seen, with the ends of surface chains and loops in the centers of surface chains moving into the vacuum. For the smaller systems, these effects dominated any bulk behavior.

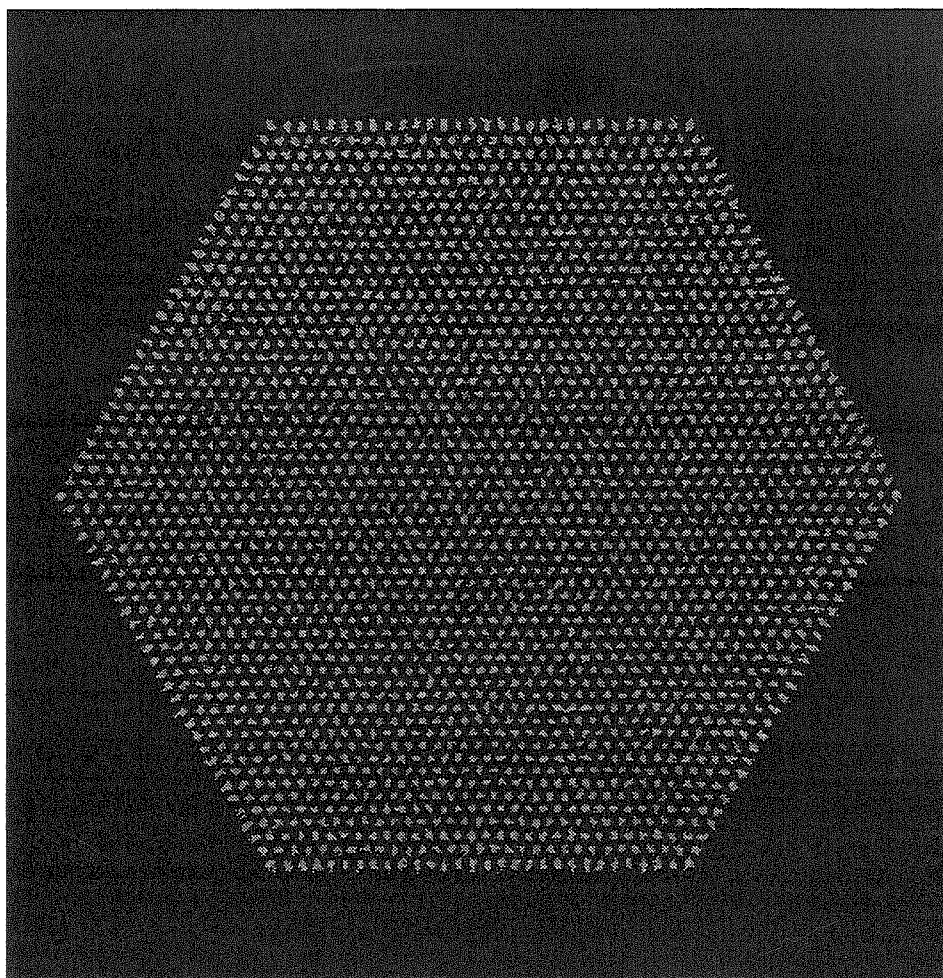


Figure 10-2. End view of 2611 chain PTFE system after dynamics.

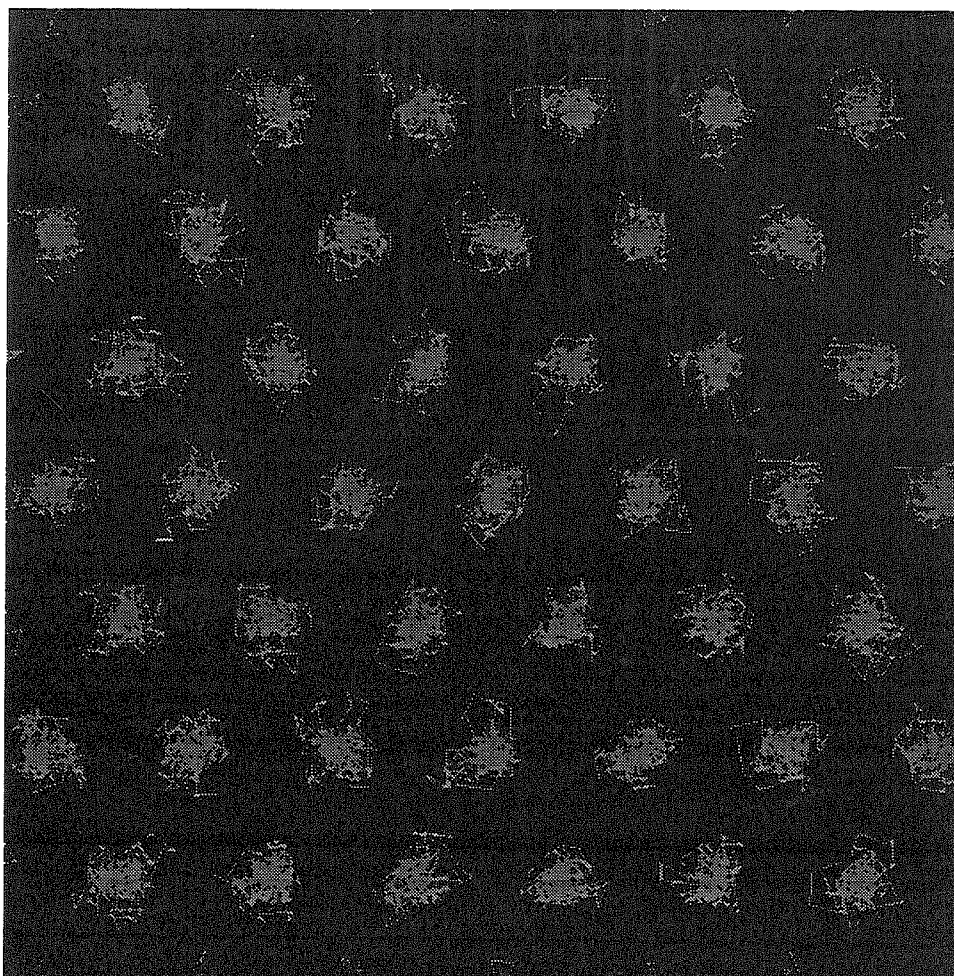


Figure 10-3. Enlargement of 2611 chain PTFE system after dynamics.

Figure 10-2 is a snapshot of the final state of the million-atom system. An enlargement of the central portion of this system is displayed in Figure 10-3.

Disorder of the helices occurred surprisingly readily, even in the central bulk portion of the system. With less accurate calculations, supercoiling of the helices was observed. It is possible that longer dynamics runs will produce the same effect in this calculation.

10.7. PTFE Surface Procedure

Two cubes of 216 water molecules were placed on the surface of a large, 1.1 million atom hexagon of PTFE. The polymer molecules were fixed in place in the experimental configuration, while the water was allowed to move. One cube was placed in the center of a surface (a YZ plane) composed of sides of helices, all with the same handedness, while the other was placed on a surface (an XY plane) composed of the ends of the helices. The side-surface drop is exposed to CF_2 groups only, while the end-surface drop is primarily exposed to CF_3 groups.

The positions of the cubes were adjusted so that the side of the cube was the TIP3P oxygen van der Waals radius away from the nearest PTFE atom.

Microcanonical dynamics at 300 K was then performed. CMM parameters were: level 6, farfield update every 5 steps. The temperature was rescaled to 300 K every 200 fs. The iterative SHAKE constraints were used. Timesteps of 2 fs were found to be feasible and were used.

The PTFE parameters from the previous section were used, along with TIP3P parameters for the water molecules.

10.8. PTFE Surface Results

After 24.4 ps, the water on PTFE had formed drops, showing no evidence of the original cubic shape. 7 molecules had boiled away from the drop on the CF_3 end surface, while 10 had left the CF_2 side surface drop. The drop shapes are shown in Figures 10-4 and 10-5. Qualitatively, we observe that the CF_3 drop appears to extend farther in the direction perpendicular to the surface (the Z direction) and also seems to have a smaller contact area

with the surface. This meshes with our expectation that the end surface should have lower surface tension, causing water to wet it less easily.

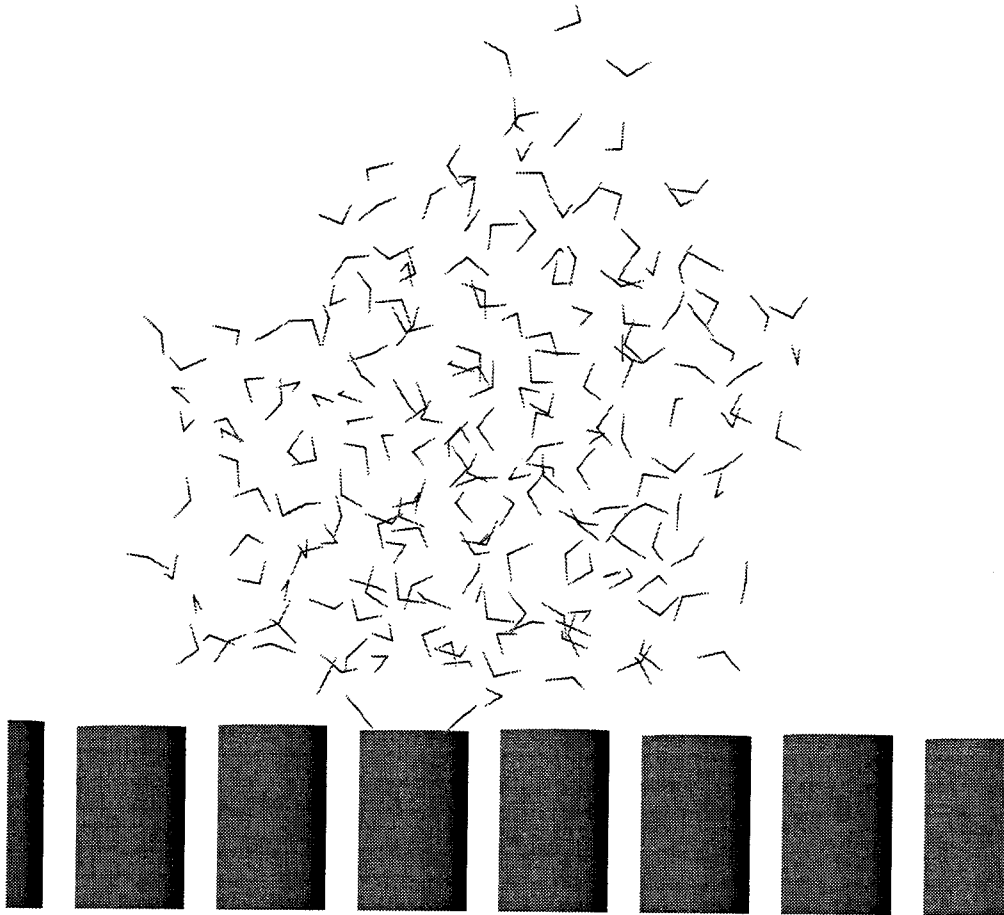


Figure 10-4. Water drop on CF_3 surface of PTFE.

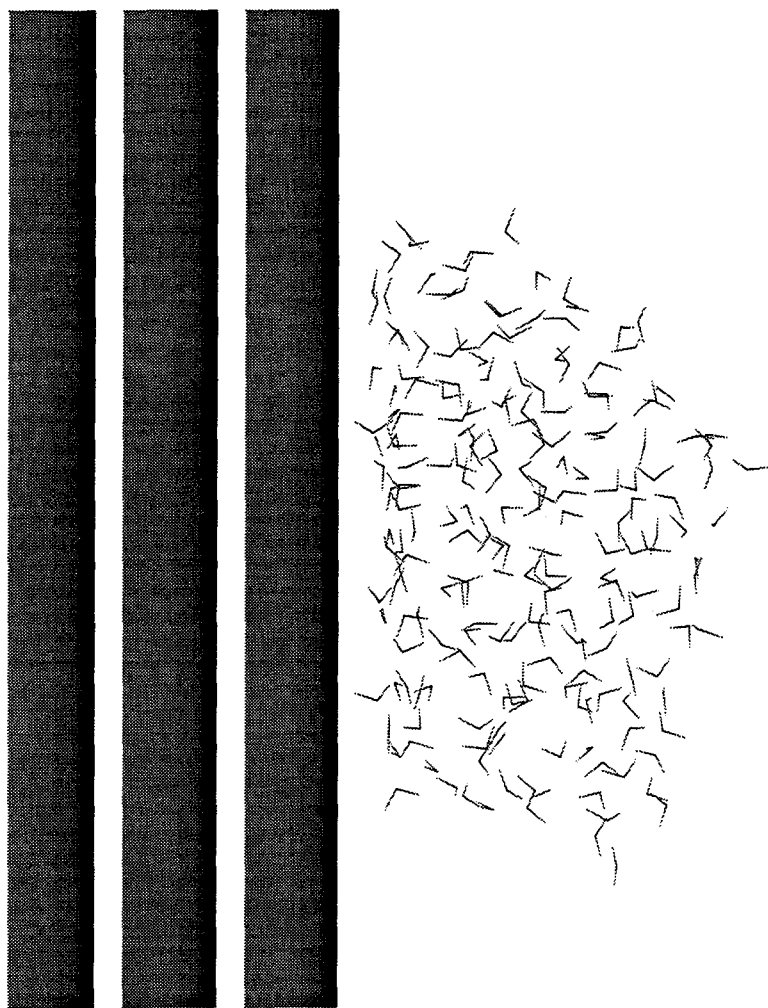


Figure 10-5. Water drop on CF_2 surface of PTFE.

To quantitate these observations, the distances of the centers of mass of the drops from the surface were computed. The surface was defined as the minimum value of the X or Z coordinate over all the drop's molecules for the CF_2 and CF_3 surfaces, respectively. The center of mass for the CF_2 drop was found to be 6.73 \AA away from the surface, while that of the CF_3 drop was found to be 9.78 \AA distant. The CF_3 drop is thus in fact elongated in the surface-normal direction compared with the CF_2 drop. Since both drops have

approximately the same number of molecules, the cross-sectional area of the CF_3 drop must also be smaller than that of the CF_2 drop.

We can compute moments of the drops along the three axes to further investigate the differences between them. We find that the root-mean-square values of the coordinates parallel to the surface are 9.35 by 8.07 Å for the CF_3 drop and 9.01 by 10.88 Å for the CF_2 drop, confirming the smaller surface area for the former.

Since the number of water molecules in each drop is relatively small, the drops are distorted from the ideal spherical sections. It is thus difficult to compute quantitative surface tension values from these results.

10.9. Conclusions

Proof-of-principle calculations were performed on a large-scale spherical drop of water and a million-atom finite PTFE system. These calculations demonstrated the ability of the code to handle systems much larger than could be used on ordinary workstations. The results of these simple calculations indicate possibly fruitful further directions for investigation.

For the water drop, future work with reflecting boundary conditions and quaternion rigid molecules should be able to make contact with experimental studies of drops on the scale of 0.1 μm . For example, a drop containing 5 million atoms, the same size as previously-simulated argon clusters, would have a diameter of about 0.05 μm .

The simulations of PTFE systems showed surprising amounts of disorder in the interior of the crystal. Further investigation of this disorder and possible supercoiling will be necessary.

The surface tension calculations used systems of greater than one million atoms. Even though most of those were fixed in place, their effects on the surface atoms were still computed rigorously. Use of the large, finite PTFE crystal allowed simulations to occur simultaneously on multiple crystal surfaces. The surfaces were large enough to avoid edge effects. The results show that drop formation can be simulated on this small scale, and they demonstrate the expected trends in drop shape, with the CF_3 surface showing a significantly lower tendency to be wetted by the drop.

Future work in this area would include removing the constraint on the polymer, allowing it to respond to motions of the water on the surface, and use of larger drops with quaternion rigid body dynamics instead of the constraint-based technique used here.

References

1. Jorgensen, W.L. Private communication.
2. Ciccotti, G.; Ferrario, M.; Ryckaert, J.-P. *Mol. Phys.*, **47**(6), 1253 (1982).
3. Jorgensen, W.L. *J. Chem. Phys.*, **77**(8), 4156 (1982).
4. Weeks, J.J.; Clark, E.S.; Eby, R.K. *Polymer*, **22**, 1480 (1981).
5. Karasawa, N.; Dasgupta, S.; Goddard, W.A. Private communication.

Chapter 11. Diffusion of Gases through Polymers

11.1. Introduction

Controlling the diffusion of gas molecules through polymer membranes is a key problem in many industrial processes. It is desirable to predict the diffusion coefficient for a given gas through a polymer of given composition. If quantitative diffusion coefficients cannot be obtained, relative diffusion rates for differing gases are still very useful.

It is generally thought that crystalline regions of the polymer matrix are too dense and closely packed to allow penetration by the diffusant molecule. The amorphous or amorphous-crystalline interface regions apparently control the diffusion process. Accurately simulating diffusion at an atomistic level thus requires the simulation of amorphous polymers.

Since amorphous regions of a polymer are not well-ordered, use of a single, short chain or chain fragment and periodic boundary conditions, as is typically done in order to limit the size of the simulation, is likely to yield inaccurate results due to the imposed short-range order. Using multiple, longer chains in a much larger unit cell should give a better approximation of the true bulk amorphous behavior. This then requires large-scale molecular mechanics, in the regime from thousands to millions of atoms, the latter particularly in cases where properties are especially sensitive to molecular weight.

Key industrial gases for which experimental data are often available include CO₂, O₂, N₂, and He. Important polymers include poly(ethylene), poly(vinyl chloride), and poly(vinylidene chloride), as well as copolymer mixtures of the latter two.

11.2. Procedure

An amorphous polymer unit cell was generated from a single chain of poly(ethylene) of 34.5 monomer units built using the BIOGRAF amorphous builder routine [1], which uses the rotational isomeric state (RIS) methodology. This single chain was equilibrated at 350 K using Nosé-Hoover constant-temperature, constant-pressure (TPN) dynamics. The final cell volume gave a density of 0.87 g/cc.

This unit cell was then replicated to build a 5x2x1 supercell consisting of 2090 atoms in 10 polymer chains. The dimensions of the supercell were approximately cubic: 27.429 x 27.046 x 27.255 Å, with angles of 90.0°. The supercell was relaxed using Nosé-Hoover TVN dynamics (fixed cell parameters) at 350 K for 50 ps. The resulting structure was highly disordered, with no apparent residual crystallinity. This structure is depicted in Figure 11-1.

The resulting unit cell was then scanned for voids using a program that attempts to fit a sphere of a given radius at each point on a grid within the unit cell. For CO₂, the sphere size was chosen to be 3.10 Å; for He, 2.5 Å spheres were used. Random voids of adequate size were chosen as the locations of the diffusant gas molecules. Figure 11-2 shows the starting position of the CO₂ molecule, along with the surrounding void spaces.

One CO₂ molecule or three He atoms were placed in the unit cell. The CO₂ diffusant molecule was treated as rigid using quaternions.

Dynamics was performed at 350 K for 300 ps using 1 fs timesteps; the location of the diffusant molecule was tracked at each 0.1 ps interval.

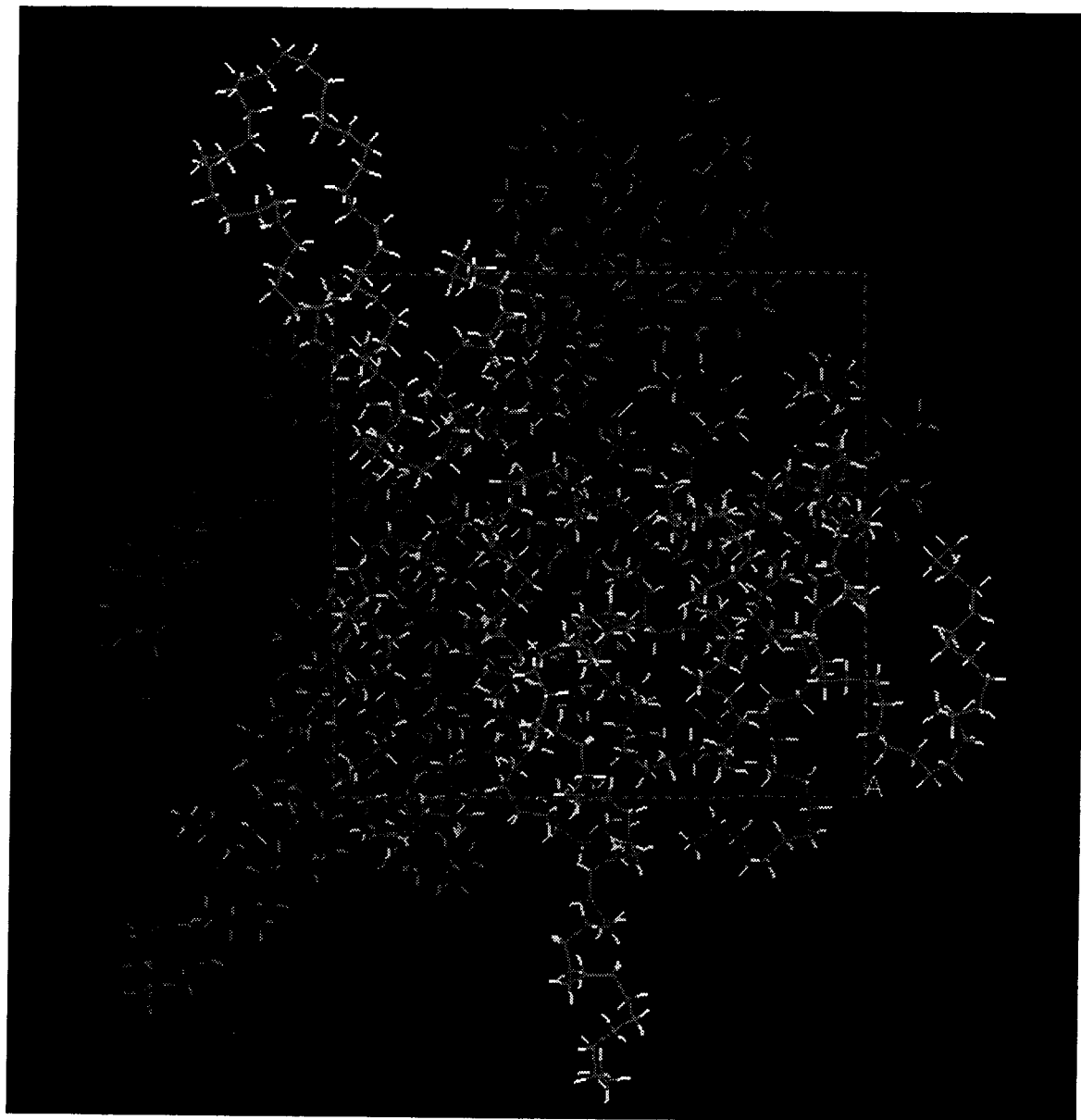


Figure 11-1. Structure of amorphous PE.

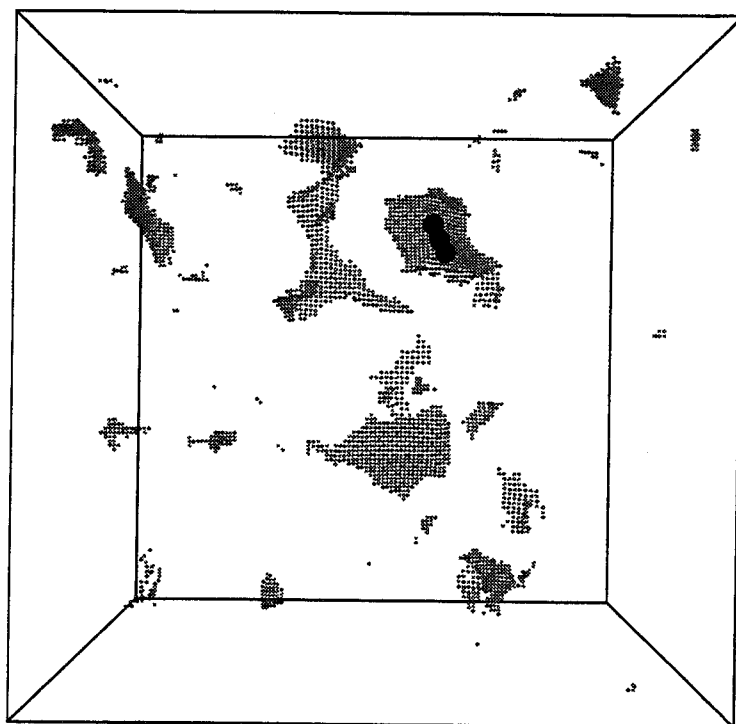


Figure 11-2. Voids in PE unit cell and initial CO₂ position.

The forcefield used had been optimized for single-chain periodic poly(ethylene) [2]. The CO₂ van der Waals parameters used were taken from the DREIDING II forcefield [3], with charges assigned so as to reproduce the experimental quadrupole moment. A CMM level of 3 was used; the farfield update frequency was every 10 steps. The Nosé-Hoover τ_s constant was set to 0.01 ps.

11.3. Results

Figure 11-3 shows the path of the CO₂ molecule through the unit cell during the diffusion process. The path appears to be composed of a few sections in which the molecule remains localized to an area, or trapped, interleaved with sections in which the molecule moves between areas. The lines crossing the path indicate the orientation of the CO₂ molecule at 10 ps intervals through the trajectory. Unexpectedly, it appears that the axis of the molecule is generally perpendicular to the direction of motion.

Figure 11-4 shows the path of the He atoms. In this case, the more mobile He atoms appear to avoid being trapped by the polymer. Note that the three He atoms went off in different directions, and that over the course of the simulation, they managed to drift through multiple unit cells.

The mean square deviation $\langle [R(t + \Delta t) - R(t)]^2 \rangle$ was computed, where the angle brackets denote the ensemble average over all possible time origins t and R is the location of the molecule center of mass or atom. A graph of this function versus values of the time interval Δt is shown in Figure 11-5 for CO₂ and Figure 11-6 for He. The slopes of these curves at large time intervals can be used to determine the diffusion constant, using

$$D = \lim_{\Delta t \rightarrow \infty} \frac{1}{6\Delta t} \langle [R(t + \Delta t) - R(t)]^2 \rangle \quad (1)$$

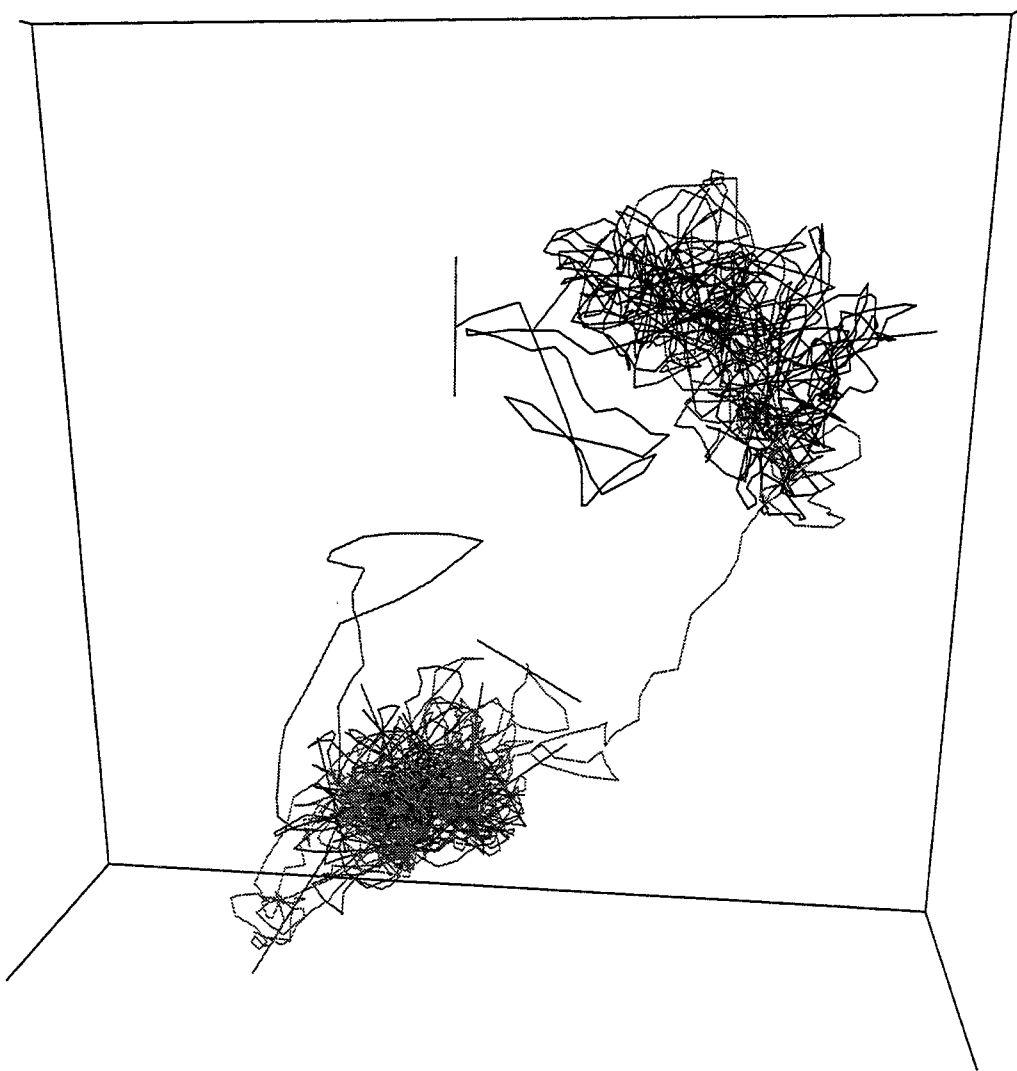


Figure 11-3. Path of CO₂ diffusion.

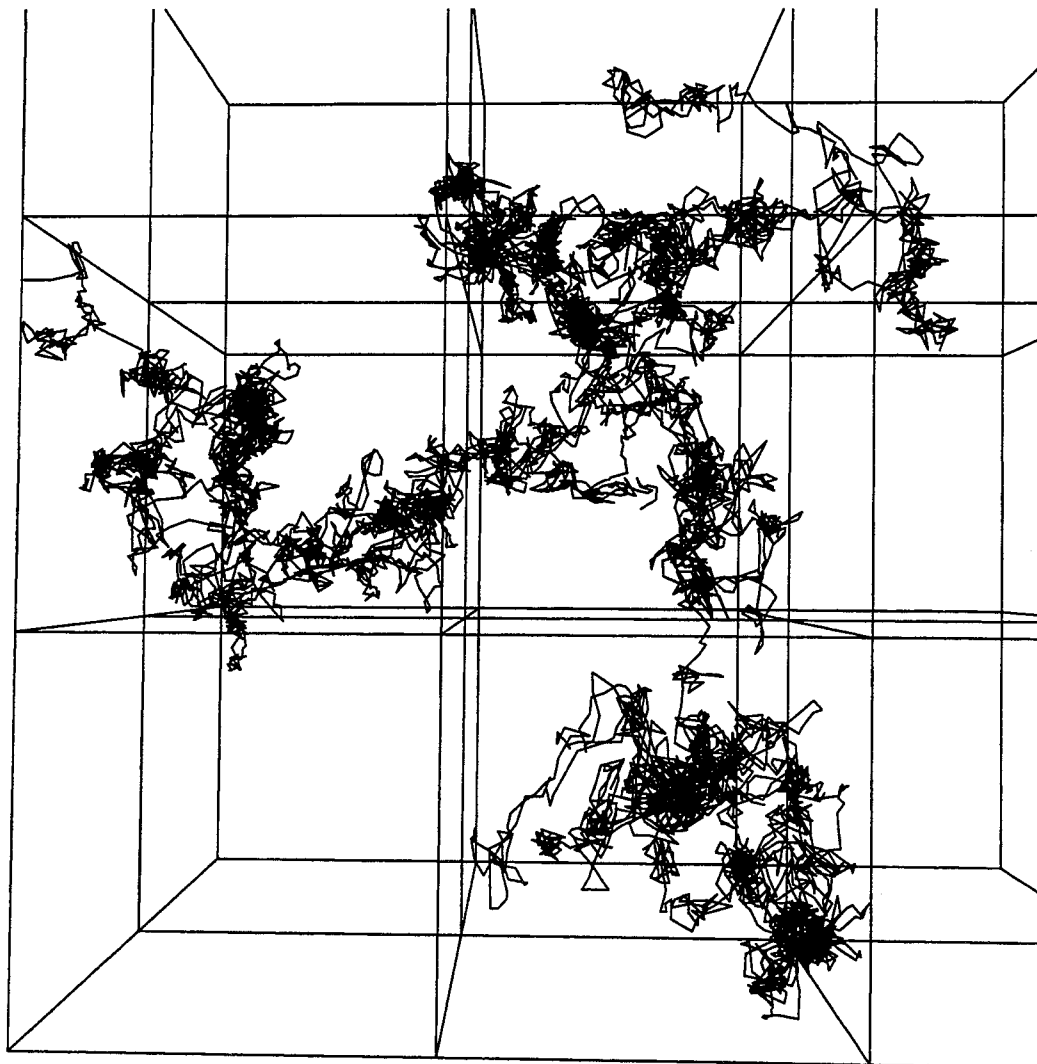


Figure 11-4. Path of He diffusion.

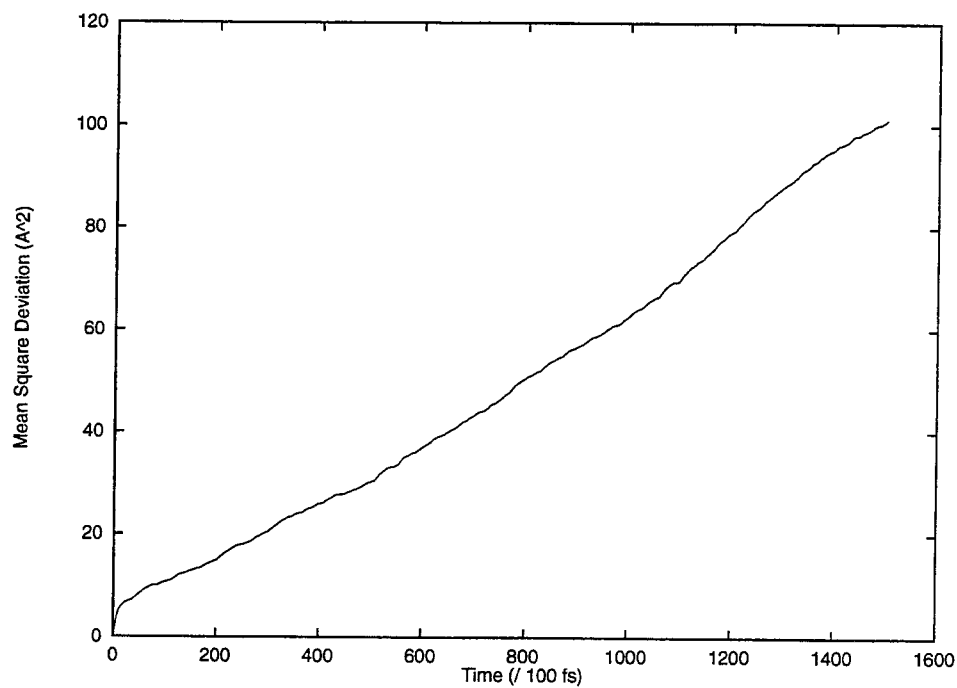


Figure 11-5. Mean square deviation curve for CO₂ in PE.

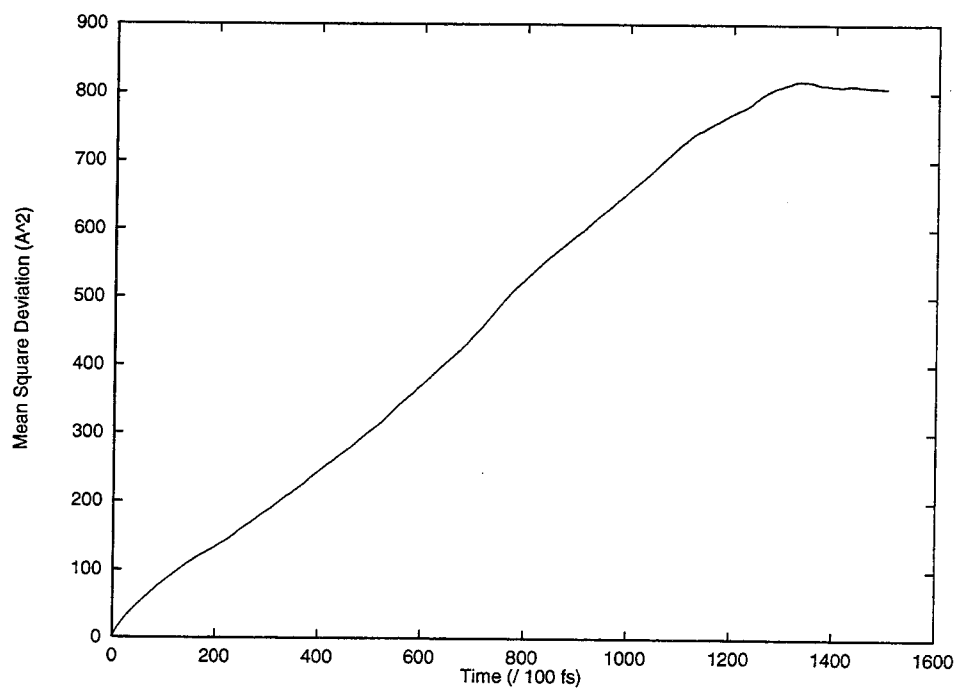


Figure 11-6. Mean square deviation curve for He in PE.

The resulting diffusion constants are given in Table 11-1 along with values for two experimental systems [4].

System	Temperature	Diffusant	Diffusion Constant (cm ² /sec)
Expt. LDPE	298 K	He	6.8×10^{-6}
(0.914 g/cc)	298 K	CO ₂	0.372×10^{-6}
Expt. HDPE	298 K	He	3.07×10^{-6}
(0.964 g/cc)	298 K	CO ₂	0.12×10^{-6}
Simulated	350 K	He	1.00×10^{-7}
(0.87 g/cc)	350 K	CO ₂	1.09×10^{-8}

Table 11-1. Experimental and simulated diffusion constants.

The simulated diffusion constants are approximately an order of magnitude lower than the experimental values, even for more dense systems.

Nevertheless, we did find that He diffusion is about 10 times faster than CO₂ diffusion, which is reasonable given the 30-fold difference in diffusion constants for a density of 0.964 g/cc, the 20-fold difference for a density of 0.914 g/cc, and our density of 0.87 g/cc.

11.4. Conclusions

The use of the efficient, parallel molecular dynamics code on smaller systems of only a few thousand atoms, but simulated for long time periods of hundreds of ps was demonstrated with this calculation.

Multiple polymer chains produced a significantly more disordered, amorphous structure than the original single chain.

We computed a ratio of diffusion constants between CO₂ and He that is of the correct order of magnitude. The low absolute values of the diffusion constants are likely related to trapping within the polymer medium. As the simulation length increases, the mean square deviation will be primarily determined by the rate of hopping between regions, rather than the rate of motion within those (presumably interstitial) regions as is currently the case.

Further investigation of the orientation of the CO₂ molecule with respect to its trajectory during the diffusion process may provide greater insight into the atomic-level processes occurring in this system. Such details can only be recovered from atomistic simulations.

References

1. BIOGRAF, version 3.21, Molecular Simulations, Inc.
2. Karasawa, N.; Dasgupta, S.; Goddard, W.A. *J. Phys. Chem.*, **95**(6), 2260 (1991).
3. Mayo, S.L.; Olafson, B.D.; Goddard, W.A. *J. Phys. Chem.*, **94**(26), 8897 (1990).
4. Brandrup, J.; Immergut, H. "Polymer Handbook," 3rd Ed., Wiley, New York, 1989, p. VI-437.

Chapter 12. Viruses

12.1. Introduction

A prototypical example of a large-scale system is a virus. Typical viral protein coats range from 0.5 million atoms up; with the addition of viral RNA or DNA, atom counts can easily exceed one million. Understanding the structure of viral protein coats is essential for investigating antigenic sites amenable to recognition by natural or synthetic agents and for understanding the process of coat assembly and disassembly that is critical to the viral life cycle. Much as with large finite simulations improving on small periodic simulations, full atomistic simulations of viruses can improve substantially on current technology, which imposes simplifying assumptions such as symmetry constraints.

As a start towards more sophisticated analyses of viral coat structure, a specific viral structure was simulated: human rhinovirus-14 (RHV). This viral coat is composed of sixty protomers arranged with icosahedral symmetry; each protomer is in turn composed of multiple subunits.

12.2. Procedure

The subunit structure at a resolution of 3.0 Å was obtained from the Brookhaven protein databank (file 4RHV) [1]. The atom types and forcefield parameters were selected from the AMBER forcefield [2], and connectivities were generated to match standard amino acids. Investigation of the fitting of asymmetric units showed that several charged groups on the surface of the protomer appeared to form salt bridges with oppositely-charged groups on the opposing surface of a neighboring protomer. After accounting for these, a net

charge of +5 remained, composed of 10 positively charged groups and 15 negatively charged groups. Counterions (sodium or chloride) were positioned near each of these charged residues. Finally, the protein coat, consisting of 60 copies of the asymmetric unit, was then built using the crystallographic symmetry operations from the structure deposited with the protein databank. The final structure contained 512,760 atoms including crystallographic waters and the added counterions.

The structure was then minimized to better position the counterions and to attempt to find the best, symmetry-free, relaxed structure for the coat. This also served to test whether the coat was stable in the absence of the viral genome. Dynamics at 0 K was used to perform the minimization. A CMM level of 6 was used with a bounding cube 360 Å on a side. The farfield was updated every 5 minimization steps.

12.3. Results

After 6000 steps of minimization, the RMS force reached a value of 0.2 kcal/mol. The energy of the final structure was -1.03×10^6 kcal/mol.

The RMS difference in coordinates between the initial and final structures was 0.495 Å, but there was no change in the radius of gyration during the course of the minimization.

Note that the structure remained relatively stable, despite the absence of RNA in the interior. This suggests that the RNA does not play an essential role in maintaining the integrity of the protein coat, though it does undoubtedly affect the structure of the internal portions of the protomer units.

12.4. Conclusions

We have demonstrated that building a symmetry-constraint-free model of a viral protein coat is now practical.

The coat structure is stable with respect to minimization from the X-ray geometry, suggesting that there is sufficient strength in the interactions between protomers to hold the coat together, even in the absence of RNA.

Future work on this project will include studying the pH dependence of the stability of the system, as experiments have suggested that acid-induced changes may be relevant to uncoating for rhinovirus-14 [3], but perhaps not for poliovirus [4]. The temperature dependence of the stability can also be determined.

References

1. Arnold, E.; Rossmann, M.G. *Acta Crystallogr. A*, **44**, 270 (1988).
2. Weiner, S.J.; Kollman, P.A.; Case, D.A.; Singh, U.C.; Ghio, C.; Alagona, G.; Profeta, S.; Weiner, P. *J. Am. Chem. Soc.*, **106**(3), 765 (1984).
3. Giranda, V.L.; Heinz, B.A.; Oliveira, M.A. *Proc. Nat. Acad. Sci.*, **89**(21), 10213 (1992).
4. Perez, L.; Carrasco, L. *J. Virol.*, **67**(8), 4543 (1993).