INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

H4.SMR/854-8

College on Computational Physics

15 May - 9 June 1995

Parallel Algorithms for Short-Range Molecular Dynamics

R. Giles

Boston University
Boston, USA

revert

Parallel Algorithms for Short-Range Molecular Dynamics

David M. Beazley, Peter S. Lomdahl, Niels Grønbech-Jensen
Theoretical Division and Advanced Computing Laboratory
Los Alamos National Laboratory, Los Alamos, NM 87545

Roscoe Giles
Department of Electrical, Computer and Systems Engineering
and Center for Computational Science
Boston University, Boston, MA 02215

and

Pablo Tamayo
Theoretical Division,
Los Alamos National Laboratory, Los Alamos, NM 87545
and
Thinking Machines Corp.
Cambridge, MA 02142.

January 14, 1995

Abstract

Over the last three years short-range parallel Molecular Dynamics methods have converged to two approaches: the *cell method*, which uses spatial decomposition to map rectangular regions of space to processors, and the *Verlet neighbor table method*, which combines the use of these cells with local neighbor tables. In this article we review the basics of these methods and analyze in detail one example of each: Los Alamos's SPaSM and Boston University-TMC's MD program. We show that these methods are fast, scalable, efficient and fully exploit the power of parallel machines even for thousands of processors. The article also contains an introduction to Molecular Dynamics, its history, traditional methodology, state-of-the-art and a list of problems in Statistical Mechanics and Materials Science that can be addressed with current methods.

1 Introduction.

After more than thirty years of existence Molecular Dynamics (MD) has become an important and widely used technique for the study of liquids, solids, and complex molecular systems in Chemistry, Biology, Statistical Physics and Materials Science. During the last decade vector supercomputers made possible the application of MD to more realistic and challenging problems. The recent introduction of scalable parallel computers has provided us with unprecedented computational power and at the same time with the challenge to make efficient use of it.

A MD simulation consists of the integration of Newtonian equations for a system of N point-like particles which represent atoms, molecules or larger units and is an example of a conceptually simple but computationally intensive numerical problem: a "high school physics problem from hell" as one of us likes to characterize it. Computationally speaking, the most important issues for short-range MD are the identification of interacting pairs and the computation of the forces. The basic tradeoff is between time invested in identifying the non-zero

interactions, so as to reduce time spent computing zero forces, and the actual force calculation itself. Traditional methods for serial and vector computers have been based on the use of *spatial cells* and/or *Verlet neighbor tables.* Parallel MD algorithms are based on similar concepts but new problems arise not present in traditional approaches.

Not many years ago it was unclear if parallel computers could be used efficiently to run MD simulations. Today, thanks to the work of many people, we know that there are efficient and scalable MD algorithms that fully exploit the power of parallel machines even for thousands of processors. Parallel MD methods have converged to two basic approaches: the *cell method*, based on using spatial decomposition to map rectangular regions of space (cells) to processors, and the *Verlet neighbor table method*, which combines the use of spatial cells with Verlet neighbor tables in each processor. In this article we will show the result of this convergence by describing these two methods and by analyzing in detail one example of each. It is our hope that the student who is interested in implementing these methods will obtain enough information to get started on his or her favorite parallel computer and the expert will find some of the tables, figures or bibliographical references useful. These methods have proven to be fast, scalable, efficient, easy to implement, and have allowed us to simulate one or two orders of magnitude larger and faster systems than older algorithms on vector supercomputers. We will limit our subject to short-range large-scale simulations of relatively homogeneous systems, where load balance issues are not critical.

We will analyze these issues in more detail in the next sections: section 2 is an introduction, including a brief history and a short review of traditional methods. In section 3 we discuss parallel computing and general approaches for parallel MD. Section 4 describes a representative cell method (Los Alamos' SPaSM) and discuss its characteristics and scaling properties. Section 5 describes and analyses a Verlet neighbor table method (the Boston University-TMC program), and finally section 6 discusses the perspectives for parallel MD simulations and lists a set of interesting problems that can be addressed in the near future.

2 Molecular Dynamics.

In this section we make a brief introduction to MD. For more complete introductions we will refer the reader to other works. A basic introduction to computer simulation methods is Gould and Tobochnik's *Computer Simulation Methods*[1] (chapter 6) and Stauffer *et al Computer Simulation and Computer Algebra*[2]. For a comprehensive introduction to MD simulations see Allen and Tildesley's *Computer Simulations of Liquids*[3], Hockney and Eastwood's *Computer Simulations Using Particles*[4], and the collection of articles in *Simulation of Liquids and Solids*[5]. Articles about the conceptual foundations of MD can be found in *Molecular Dynamics Simulations of Statistical-Mechanical Systems*[6]. Other useful references are Hover's *Computational Statistical Mechanics*[7], *Molecular Dynamics*[8] and the collection *Molecular Dynamics Simulations*[9] edited by Yonezama.

For other review articles about MD on vector and parallel computers we refer the reader to van Waveren[10], Fincham[11], Rapaport[12,13,14], Abraham[15], Smith[16], Gupta[17], Boghosian[18], Plimpton[19,20,21] and Esselink[105,106].

The program library CCP5 at Daresbury Laboratory U.K. contains many examples of MD programs including the ones from Allen and Tildesley's book, K. Refson's "Moldy," and Forester and Smith's "DL POLY."

Other references and links can be found in our "Molecular Dynamics Related Resources and Information" World Wide Web page at http://conx.bu.edu/CCS/md/md.html.

2.1 A Brief History.

The first computer simulations to study statistical mechanical systems, or any physical system for that matter, were done on the first-generation vacuum tube computers, such as the ENIAC and MANIAC, after World War II at Los Alamos National Lab. The first MD simulations, properly speaking, were done by Alder and Wainwright[22] in 1957. They simulated systems with a few hundreds of hard-sphere particles and discovered the existence of a fluid-solid phase change. This was somewhat surprising because it was thought that an attractive potential was necessary to produce the fluid-solid transition. At the same time Wood and Parker[23] studied the properties of simple fluids using the Monte Carlo method. These early results attracted the attention of physicists and chemists who saw in the numerical methods a new tool to test models and improve the understanding of many body systems.

In those, the early days of computational science, simulations were done on state-of-the-art UNIVAC and IBM 704 computers using assembler programming. Debugging consisted of looking at endless piles of listings with "dump core", something young readers are hopefully unaware of. In 1964, Rahman[24] performed the first MD simulations with the Lennard-Jones potential. He determined pair correlations and diffusion constants and obtained very good agreement with experimental data. This was one of the first breakthroughs that opened the way for future work. A few years later, in 1967, Verlet[25] determined the phase diagram for the Lennard-Jones fluid and introduced the use of neighbor tables to save computation. By then, computers were made out of transistors and were much easier to use thanks to the introduction of the first fortran compilers. In 1969, Alder and Wainwright[26] discovered an algebraic long-time tail in the velocity autocorrelation function of hard spheres. This important and unexpected discovery stimulated continuing interest in MD simulations as a tool of exploration. In 1971, Rahman and Stillinger[27] addressed the problem of simulating more complex, and at the same time more realistic, molecular interactions as in the case of liquid water (see review of F. Sciortino, S. Sastry and P. H. Poole in vol. II of this series).

Once the basic methodology and confidence in the technique were established the horizons of MD expanded. The 70's saw further improvements in methodologies and algorithms, for example, practical algorithms to compute molecular rotations were developed by Evans and Murad[28] and Ciccotti and co-workers[29], and new methods to measure free energies, such as the umbrella method, were introduced by Torrie and Valleau[30], Frenkel and Ladd[31], Bennett[32] and others. State-of-the-art MD programs used cell partitioning and Verlet neighbor table methods in fast sequential machines such as the CDC 7600 and the IBM 370. In the early 80's Andersen[33], and Parrinello and Rahman[34], developed methods to perform constant pressure and constant temperature MD. In 1984, Nose[35] developed equations to simulate constant temperature MD by introducing additional degrees of freedom in the form of a "thermostat". These equations were later reformulated and simplified by Hoover[36]. All these improvements extended the applicability of MD to other ensembles besides the constant energy ensemble of the original simulations.

The advent of the first vector machines motivated the search for efficient vectorization techniques, in particular the ones relevant to the use of Verlet neighbor tables[37,38,39,12,40]. The first generation of vector supercomputers, such as the Cyber 205, Cray 1 and IBM 3090, produced a significant improvement in relation to the serial machines of the previous generation. By the mid 80's MD methods and algorithms were mature enough to address problems in non-equilibrium statistical mechanics. For example, Erpenbeck and co-workers[41] discovered the "string phase" in planar coulette flow in 1984. Further progress came in 1985 when Car and Parrinello[42] introduced a method combining MD and electronic structure calculations. In 1984, Abraham and co-workers[43] studied phase separation and spinodal decomposition in 2D systems with up to 161,604 particles.

The late 80's and early 90's saw the appearance of parallel MD algorithms. In 1990 the first multi-million particle simulations were done by a group at Livermore[44], by Swope and Andersen[45] and by Rapaport[13,14,46].

Large scale simulations have contributed to the understanding of new and complex phenomena. For example, Swope and Andersen[45] made a comparative study of crystalization in systems with 15,000 and 10^6 particles and found the existence of competing crystal structures that could only be seen in the larger system.

2.2 State of the Art.

In the last five years many parallel MD methods have been studied and introduced in the literature. See for example the methods of Rapaport[12,13,14,46,47], Smith[16], Brown *et al*[48], Kalia *et al*[117], Scott *et al*[118], Plimpton[20,21], Plimpton and Heffelfinger[19], Form *et al*[52], Buchholtz, and Pöschel[53], Raine *et al*[54], Fincham and Smith[54], Esselink *et al*[55], Hedman and Laaksonen[56], Bruge and Forlini[57], Melcuk *et al*[58], Beazley and Lomdahl[59], Lomdahl *et al*[60,61], Beazley *et al* [62], Tamayo *et al*[63], Tamayo and Giles[64].

Almost all of these methods use spatial decomposition to define coarse-grained cells sometimes in combination with Verlet neighbor tables. Most of them have reasonable scaling properties and despite some differences the partitioning techniques and basic algorithms are similar.

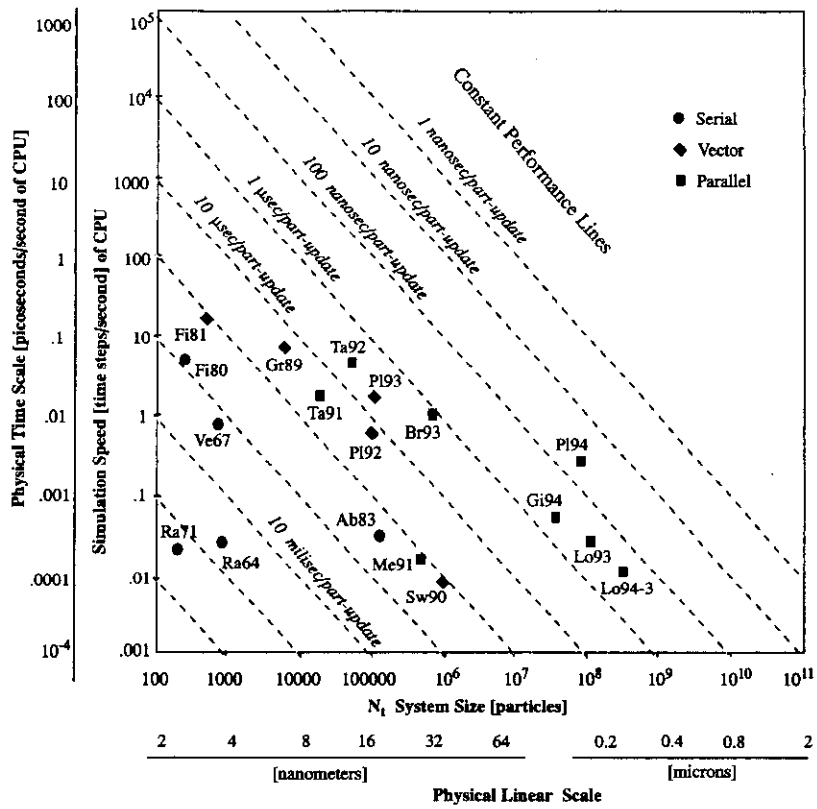


Figure 1: Space and time scales of short-range 3D Molecular Dynamics simulations. See table 1 for legends

It is interesting to analyze the progress in short-range MD algorithms (Lennard-Jones) as measured by the updating speed versus system size as is shown in Fig. 1. The figure also shows physical space and time scales. There are several interesting observations we can make about this plot. Since the mid-sixties up to nowadays the performance per atom update has increased by about six orders of magnitude. This is the result not only of faster machines and better compilers but also of more efficient implementations and algorithms with appropriate scaling properties. One can roughly distinguish three different stages: the first from the mid-sixties to the late seventies is the period of the fast serial machines (UNIVAC, CDC 6600 and 7700, IBM 360, etc.). In the 80's the vector supercomputers (Cray-1, Cray XMP, Cyber 205, IBM 3090 etc.) produced another two orders of magnitude improvement over the previous era. The last period from the late 80's until today has seen a dramatic increase in performance produced by the parallel machines (nCUBE, iSPC/860, Paragon, T3D, VP1000, CM-2, CM-5 etc.).

The diagonal lines in the plot are *constant performance* lines. A given machine and implementation of the algorithm gives a performance point and these lines are defined assuming there is a linear trade off between speed and size. One can make the system size smaller to increase the speed or viceversa. This trade off holds within some central region. If one changes the system size so as to move far up or down the line one encounters practical limitations. For example, moving up by decreasing system size increases the speed but reduces the data set in each processor and consequently the efficiency goes down and at the end one faces the limit imposed by the computation speed of a single processor. Moving down increasing system size is limited by the total amount of memory in the machine.

There are two ways to increase performance: by adding more processors –and moving to the right on a horizontal line– or by increasing the speed of the processors, and in this way moving up vertically. It is easier to add more processors than making them faster. This has been one of the motivations for parallel computing in the first place. Adding processors is an easy way to increase the performance but one still faces a problem of *scalability*: the efficiency –measured as the *effective* divided by the *actual* number of processors in the machine– will decrease as the number of processors increases. This is the reason algorithms with good scaling properties are needed.

Table I. A sample of State-of-the-Art MD algorithms.

Code	Researcher(s) year	Algorithm DP=DataParallel SP=SinglePrec.	Machine (# of procs.)	ρ/σ_c	Size N	t_{update} secs	$t_{particle}$ μsecs	t_{pair} nano- secs	$t_{pair}^{p=1}$ μsecs
Serial and Vector Machines:									
Ra64	Rahman[24] 64	Direct N^2	IBM 704 (1)	0.8/2.25	864	45.5	52662	10^6	-
Ve67	Verlet[25] 67	Tables	CDC 6600 (1)	0.45/2.5	864	1.2	1389	47157	-
Fi80	Fincham[37] 80	Direct N^2	CDC 7600 (1)		256	0.21	8172		-
Fi81	Fincham and Ralston[39] 81	Tables	Cray 1 (1)		500	0.06	118		-
Es94-4	Esselink[106] 94	Cells	HP 9000/735(1)	0.84/2.5	2,000	0.106	53	964	-
Wa87	van Waveren[10] 87	-	Cyber 205 (1)	1.0/2.5	2,500	2.73	1092	16684	-
He88	Heyes and Smith[86] 88	Cells	Cray 1s (1)		6,912	1.60	230		-
Sc89	Schoen[40] 89	Direct N^2	Cyber 205 (1)		6,912	0.53	76.5		-
Gr89	Grest *et al*[107] 89	Cells/Tables	Cray XMP (1)	0.84/2.5	6,912	0.16	23.1	421	-
Pl93	Plimpton[20] 93	Cells/Tables	Cray YMP (1)	0.84/2.5	100,000	1.47	14.7	267	-
Pl93-2	Plimpton[20] 93	Cells/Tables	C-90 (1)	0.84/2.5	100,000	0.59	5.9	107	-
Sw90	Swope and Andersen[45] 90	Cells/Tables	IBM 3090 (1)	0.95/2.3	1,000,000	108	108	2231	-
Parallel Machines:									
Sc93	Scott *et al*[118] 93	Replic. Data^{SP}	MUSIC (20)	0.89/2.5	1,000	0.02	20.0	343	6.87
Ba94	Batista and Coker[128] 94	Cells/Tables	SGI (IP22) (8)	1.0/2.5	4,000	1.17	293	4469	35.8
Ta91	Tamayo *et al*[63] 91	Cells^{DP}	CM-2 (32K/32)	0.8/2.5	18,000	0.57	31.7	605	619
Es93	Esselink *et al*[55] 93	Cells	T800 (400)	0.7/2.5	39,304	0.86	21.9	478	191
Ta92	Tamayo and Giles[64] 92	Cells Tables	CM-5 (512)	0.84/2.5	51,200	0.20	3.91	71.1	36.4
Es94-4	Esselink[106] 94	Cells	IBM SP1 (12)	0.84/2.5	54,000	0.73	13.5	246	2.95
Ka93	Kalia *et al*[117] 93	Cells	iPSC/860 (8)	1.0/2.5	108,000	5.2	48.1	736	5.86
Es94-4	Esselink[106] 94	Cells	SGI Chal. (28)	0.84/2.5	126,000	0.83	6.59	120	3.36
Me91	Mel'čuk *et al*[58] 91	Cells^{DP}	CM-2 (32K/32)	0.95/2.3	512,000	58.8	115	2372	2429
Os94	Ossadnik and Gyure[120] 94	Cells^{DP}	CM-5E (32)	1.0/2.5	524,288	10	19.1	291	9.33
Br93	Brown *et al*[48] 93	Cells/Tables	AP1000 (512)	0.8/2.5	729,000	0.93	1.28	24.4	12.5
He93	Hedman *et al*[56] 93	Cells^{DP}	CM-200 (8K/32)	0.82/2.8	1,000,000	5.9	5.9	78.1	20.0
Ly94	Lynch and Tamayo[109] 94	Cells^{DP}	CM-5E (32)	1.0/2.5	1,000,000	14.6	14.6	223	7.14
Ra91	Rapaport[14] 91	Cells	iPSC/860 (64)	.71/1.12	2,050,000	2.75	1.34	321	20.5
Pl94-3	Plimpton[21] 94	Cells/Tables	T3D (256)	0.84/2.5	5,000,000	2.25	0.45	8.19	2.10
Pl94-2	Plimpton[21] 94	Cells/Tables^{SP}	nCUBE2 (1024)	0.84/2.5	10,000,000	10.2	1.02	18.6	19.0
Ra93	Rapaport[47] 93	Cells	CM-5 (64)	.71/1.12	10,976,000	12.5	1.14	273	17.4
Lo94-2	Lomdahl *et al*[126] 94	Cells	CM-5E (32)	1.0/2.5	16,384,000	91.7	5.60	85.5	2.74
Gi94	Giles and Tamayo[119] 94	Cells/Tables	CM-5E (256)	0.84/2.5	43,904,000	19.65	0.448	8.15	2.09
Be93	Beazley and Lomdahl[59] 93	Cells	CM-5 (1024)	1.0/2.5	67,108,864	87.7	1.31	20.0	20.4
Lo94-4	Lomdahl *et al*[126] 94	Cells	T3D (128)	0.84/2.5	75,000,000	46.9	0.625	11.4	1.46
Pl94	Plimpton[21] 94	Cells/Tables^{SP}	Paragon (3680)	0.84/2.5	100,000,000	3.5	0.035	0.636	2.34
Lo93	Lomdahl *et al*[60] 93	Cells	CM-5 (1024)	0.84/2.5	131,072,000	34.0	0.259	3.96	4.06
Lo94-3	Lomdahl *et al*[126] 94	Cells	CM-5 (1024)	0.84/2.5	300,800,000	90.6	0.301	5.48	5.61
Lo94	Lomdahl *et al*[126] 94	Cells^{SP}	CM-5 (1024)	0.84/2.5	600,000,000	242	0.403	7.34	7.51

To give a better idea about the actual performance of state-of-the-art MD programs Table I shows performance data for some state-of-the-art MD algorithms on a variety of new and old machines. This is not intended as an exhaustive enumeration but just a sample of algorithms and implementations we are aware of. As we can see in the table the fastest programs attain performances in the *nano*second per pair interaction range.

When comparing different MD algorithms one has to be careful because no single measure of performance gives the full picture. Mflop rates do not give a reasonable measure of performance because they depend upon the particular method being used. A more useful quantity, frequently reported in the literature, is the *time per particle update* $t_{particle}$, which is simply the *update time* t_{update} for one time step divided by the *total number of particles* N. This is in general a useful quantity but the use of different densities and cutoffs makes comparisons between machines, algorithms and implementations difficult. A better quantity to characterize the performance of an MD algorithm is the *effective time per pair interaction*, t_{pair},

$$t_{pair} = \frac{t_{update}}{\frac{4}{3}\pi\sigma_c^3\rho N} = \frac{t_{particle}}{\frac{4}{3}\pi\sigma_c^3\rho} \tag{1}$$

where σ_c is the cutoff, the distance beyond which the potential is taken to be effectively zero, and ρ the density. This quantity multiplied by the *number of processors*, p, gives the *one-processor effective time per pair interaction*,

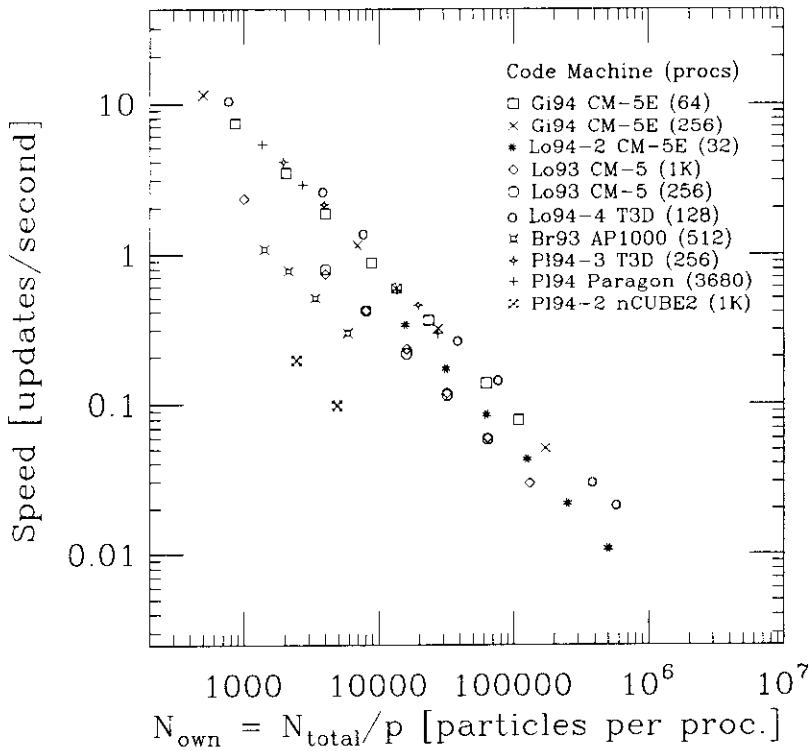
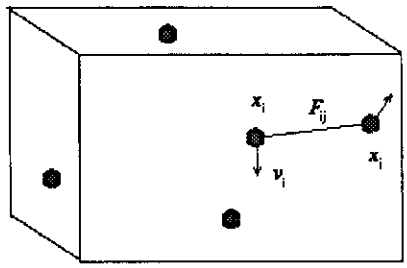


Figure 2: Speed (updates per second) *vs* number of particles per processor for some state-of-the-art parallel MD algorithms. Scalability implies that data for different machines sizes should collapse on a single line as can be seen for some of the examples shown.

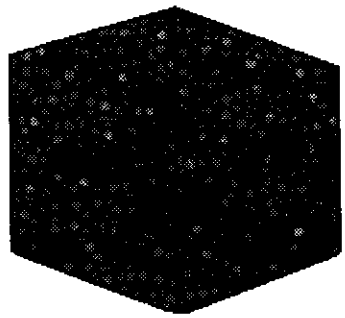
$$t_{pair}^{p=1} = p\, t_{pair}, \tag{2}$$

which is also useful to compare performance independent of the number of processors. These quantities measure the time to compute an "effective unit of work" in a MD simulation: a single Lennard-Jones pair-interaction in double precision. If the calculation is done in single precision we suggest to multiply $t_{pair}^{p=1}$ by an appropriate factor to make comparisons. This quantity is roughly independent of the density, cutoff, machine and system sizes within reasonable limits. Of course, no single number can convey the scaling properties of an algorithm but t_{pair} and $t_{pair}^{p=1}$ provide useful figures of merit when used judiciously. The last two columns on the table include t_{pair} and $t_{pair}^{p=1}$ for the algorithms and implementations shown.

To study scaling properties it is useful to plot the speed of a program *vs* the number of particles per cell-processor, N_{own}. If the algorithm, and the parallel architecture, are scalable the data should fall roughly onto the same line independent of the number of processors. This is indeed the case for some algorithms as can be seen in Fig 2. When MD algorithms are scalable, and keep the communications costs to a minimum, the dominant factor for performance becomes the effective computational speed of the local processing node. We will come back to this subject when we discuss each of the parallel methods in detail.



Simulation with 5 particles
Problem: high school homework
Machine: hand calculator



Simulation with 2000 particles
Problem: study of liquid structure
Machine: workstation

Simulation with 38 million particles
Problem: study of crack formation
Machine: parallel supercomputer

Figure 3: Three examples of Lennard-Jones 3D Molecular Dynamics Simulations.

2.3 A Short Introduction.

One of the main objectives of Statistical Mechanics is to explain the macroscopic behavior and properties of matter (temperature, phase, density, etc.) from the microscopic properties of atoms and molecules. For example, if we look at a glass of beer[65] –not a fruitless experiment if one remembers how the idea of the invention of the Bubble Chamber came about– we will realize that it has well defined homogeneous properties despite the fact it contains about 10^{25} particles. One is able to predict and control the behavior of this system without knowing anything about the individual trajectories of the myriad of molecules inside the glass.

The connection between the macroscopic emergent properties and the microscopic laws is complex and far from trivial in most cases. The particles interact dynamically and as a consequence complex physical behavior appears at different space and time scales. The analytical techniques to explain the connections between the micro and the macro world constitute the formal content of Statistical Mechanics, Chemical Physics and Condensed Matter Physics.

Today, scientific research in the physical sciences is based on a triad of *theory*, *experiment* and *simulation*. The subject of Computational Science is about simulations and algorithms and the best way to implement them in real computers. MD is a direct simulational approach based on the numerical calculations of particles trajectories according to classical mechanics and simplified semi-empirical molecular interaction forces. If the simulations are performed with enough numbers of particles, and if the potential captures enough of the relevant physics, then the properties of interest will emerge as a consequence of the dynamics. Fig. 3 shows three examples of Lennard-Jones 3D MD simulations.

2.3.1 The Main Computational Tasks.

Suppose we have N classical particles in our system and that the state of affairs can be described by a collection of positions and velocities,

$$\{\vec{x_1}, \vec{x_2}, \vec{x_3}, ..., \vec{x_N}\} \quad \{\vec{v_1}, \vec{v_2}, \vec{v_3}, ..., \vec{v_N}\}. \tag{3}$$

Now lets define r_{12} as the distance between the first and the second particles, r_{13} between the first and third etc. and assume the total energy will consist on pair interactions only,

$$E = U(r_{12}) + U(r_{13}) + U(r_{23}) + ... \tag{4}$$

where $U(r)$ denotes the inter-particle potential for two particles separated by a distance r. The idea is to choose a simple model for $U(r)$ that will produce the physical behavior of interest. For example, to simulate the thermodynamic behavior of a simple liquid the basic ingredients of an inter-particle potential are: a *hard core*, that provides repulsive forces and exclusion effects if particles get too close, and an *attractive force*, that models the *van der Waals* inter-molecular attraction. One of the most widely used potentials with these characteristics is the Lennard-Jones potential,

$$U(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right]. \tag{5}$$

This potential is parameterized by σ, the length at which the potential crosses zero, and by ϵ which defines the energy scale. For liquid Argon atoms, which this potential models relatively well[24], $\sigma = 3.4\mathring{A}$ and $\epsilon/k_B = 119.8K$. The potential decays rapidly and a cutoff distance can be defined beyond which the potential is very small and can be taken to be effectively zero. In typical MD simulations this cutoff r_c varies from 1.12 σ for hard core simulations up to 2.5-5 σ for simulations of simple liquids or solids. The force is given by the gradient of the potential,

$$\vec{F_{ij}} = -\nabla U(r_{ij}). \tag{6}$$

The distance r_{ij} between each pair of particles is needed to compute the force,

$$F(r_{ij}) = 24\epsilon \frac{\sigma}{r_{ij}} \left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6} \right]. \tag{7}$$

This force is used only if $r_{ij} < r_c$ and it is made equal to zero otherwise. Notice that $F_{ij} = -F_{ji}$ so Newton's *3rd* law can be used to save computation. From the forces one finds the accelerations,

$$\vec{a}_{ij} = \frac{F_{ij}}{m} \hat{r}_{ij}, \tag{8}$$

and then one integrates the equations of motion to find the new set of coordinates and velocities for time $t + \Delta t$ from the values at time t. The study of integration algorithms is a subject by itself[66,67,68,69,70]. Here we will just show one example: the Verlet velocity integration scheme.

$$\vec{r}_i(t + \delta t) = \vec{r}_i(t) + \delta t\, \vec{v}_i(t) + \frac{1}{2}(\delta t)^2 \vec{a}_i(t), \tag{9}$$

$$\vec{v}_i(t + \delta t) = \vec{v}_i(t) + \frac{1}{2}\delta t \left[\vec{a}_i(t) + \vec{a}_i(t + \delta t)\right]. \tag{10}$$

These equations are simple to implement and only require the storage of the current and previous positions, velocities and accelerations[3].

The actual number of interacting neighbors is a function of the molecular density. For a typical liquid-state simulation using the Lennard-Jones potential the computation of a single pair-interaction requires about 30-40 floating-point operations, therefore a complete force calculation will require of the order of 2,000 floating-point operations per particle. This is much more manageable than a full N-body calculation but still computationally intensive. There are ways to produce faster dynamics by using multiple-time-steps[71,72] or variable step size[68,70], however there are limitations to these approaches.

In summary, the main three computational tasks of a short-range MD calculation are: *finding the interacting neighbors*, the *computation of forces* and the *integration of the equations of motion*. The most computational intensive part is always the computation of forces which accounts for 70-90 % of the time in serial, vector and parallel MD programs.

2.3.2 Direct N^2 Solvers and Multipole Methods.

When the number of particles to be simulated is not large, it is possible to take the direct N^2 approach to the calculation of forces. Sometimes this is done for simulations of celestial objects in Astrophysics[73,74,75]. As part of the Japanese GRAPE 4 project for stellar dynamics a machine with teraflop capability is being constructed[76].

Systolic algorithms[77,78] for MD have been studied by Nelson *et al*[79,80]. Optimal parallel schemes for N-body simulations have been studied by Brunet *et al*[81,82] and Greenberg *et al*[83]. Direct N-body solvers[75] are also relevant to vortex methods in hydrodynamics[84,85]. Fincham[37] and Schoen[40] have used the direct N^2 approach in MD simulations. Schoen's program was about three times faster than a link cell method for 6,912 particles[86].

In the last decade there has been significant progress in the development of techniques to approximate the effect of long range forces. This is relevant to MD simulations where long-range electrostatics is important. One class of approximation methods is based on imposing a grid structure and dividing the contributions to the force in two parts: an atomistic short-range piece and a interpolated long-range contribution. This approach is known as P^3M *particle-particle particle-mesh* method[4]. The problem of computing long range electrostatic forces for Coulombic systems can also be addressed with the use of Ewald summation techniques[3,87,88,89,90,91] A different class of approximations is based on hierarchical methods some of them based on multipole expansions similar to the ones used in electromagnetism[92]. The best known hierarchical algorithms are the methods of Appel[93], Greengard and Rokhlin[95], Barnes and Hut[94], Zhao[96] and Anderson[97].

The *Barnes-Hut* algorithm is an $O(N \log N)$ algorithm based on a hierarchical spatial octree where interactions are computed using a first order approximation. Greengard and Rokhlin's *Fast Multipole Method* is $O(N)$ requires

more complex data structures. The methods of Zhao, Anderson and Appel are also $O(N)$. For examples of parallel implementations of these algorithms see refs. 98,99,100,101.

A recent study of the effects of different cutoff methods for long range forces can be found in ref. 104. A comparison of different algorithms for long range interactions has been done in ref. 105. There are also new alternative approaches to deal with long range forces based on domain decomposition and Taylor polynomials [102,103] which circumvent some of the problems of traditional multipole methods. The study and development of multipole and related methods is a very active area of research.

2.3.3 Cell Partitioning Methods.

In a system of n particles interacting with a potential of range r_c, and average particle density ρ, each particle will interact with $N_{neig} \sim \frac{4}{3}\pi r_c^3 \rho$ neighbors on average. If for each particle one performs a global search for these neighbors over the rest of the system the total process will be $O(N^2)$. However, for short-range rapidly decaying potentials, such as Lennard-Jones, the interaction is treated as zero for particles whose separation is larger than r_c. This reduces the computational complexity to $O(N)$ and provides a way to restrict the search for interacting neighbors to the vicinity of each particle. A widely used technique to do this is by dividing the physical simulation space in cells or rectangular domains in such way that if one is computing the force acting on a given particle i, then one looks for neighbors, particles j for which $|r_i - r_j| < r_c$, only inside surrounding cells. Fig. 4 a) shows a $2D$ system in which physical space has been divided into square cells of size r_c. The dark-shaded region shows the interacting neighborhood of radius r_c for a given particle i. All the interacting neighbors for the central cell can be found inside the light-shaded region defined by the 8 neighboring cells. In b) the linear cell size is $r_c/2$ and then the search involves two layers of neighboring cells.

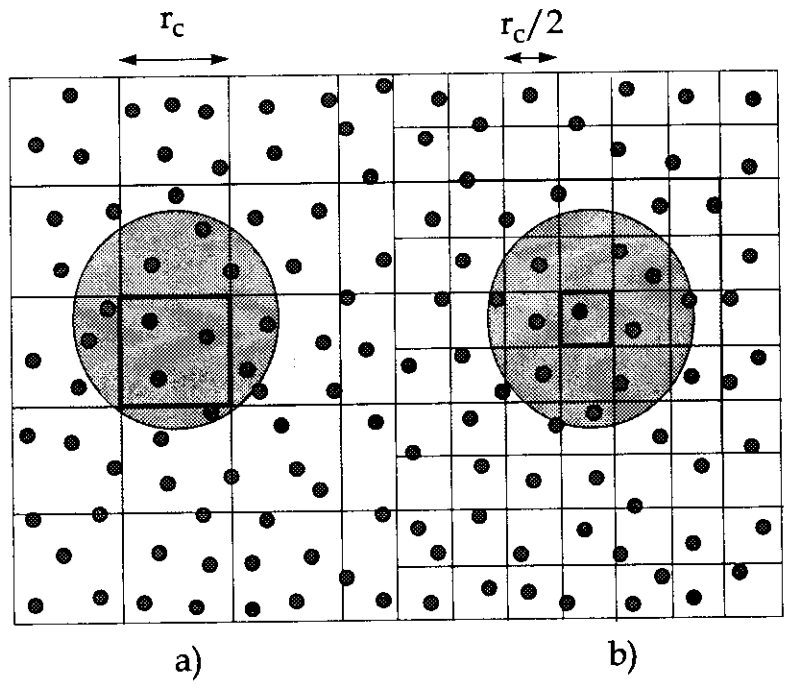


Figure 4: Cell partitioning method: a) large cells of size r_c; b) small cells of size $r_c/2$.

The use of cells reduces the complexity of the search but does not eliminate all unnecessary computation. Due to the geometrical mismatch between the rectangular shape of the cells and the spherical interacting neighbor-

hood region, not all particles inside the cells necessarily contribute to the force, and therefore, some unnecessary interactions will still be considered.

What is the optimal size for the cells ? Small cells are better to approximate the spherical shape of the interacting neighborhood region but they also introduced overhead. The overhead is produced by the decrease in occupation numbers, the higher probability of empty cells and the greater complexity of the search path. In practice, most MD programs employ large cells of size r_c, or sometimes $r_c/2$, so the search is restricted to the first or second layer of neighbors. This tradeoff is significant for serial programs but it is even more important for parallel programs where the search path is partially done off processor. The goal of reducing communications favors the choice of cells of size r_c. A *link list* data structure is often used to access a given cell's particles from a global array of particles[3].

2.3.4 Verlet Neighbor Table Methods.

If there are on average N_{neig} interacting neighbors per particle then the average number of non-zero pair forces will be $\frac{1}{2}N \cdot N_{neig}$ taking Newton's third law into account. If at each stage of the computation the program maintains a table of the interacting neighbors for each particle, or alternatively a list of interacting pairs for which the interaction forces have to be computed, a substantial amount of computation can be saved. This is the idea behind the Verlet neighbor table method first introduced by Verlet[25].

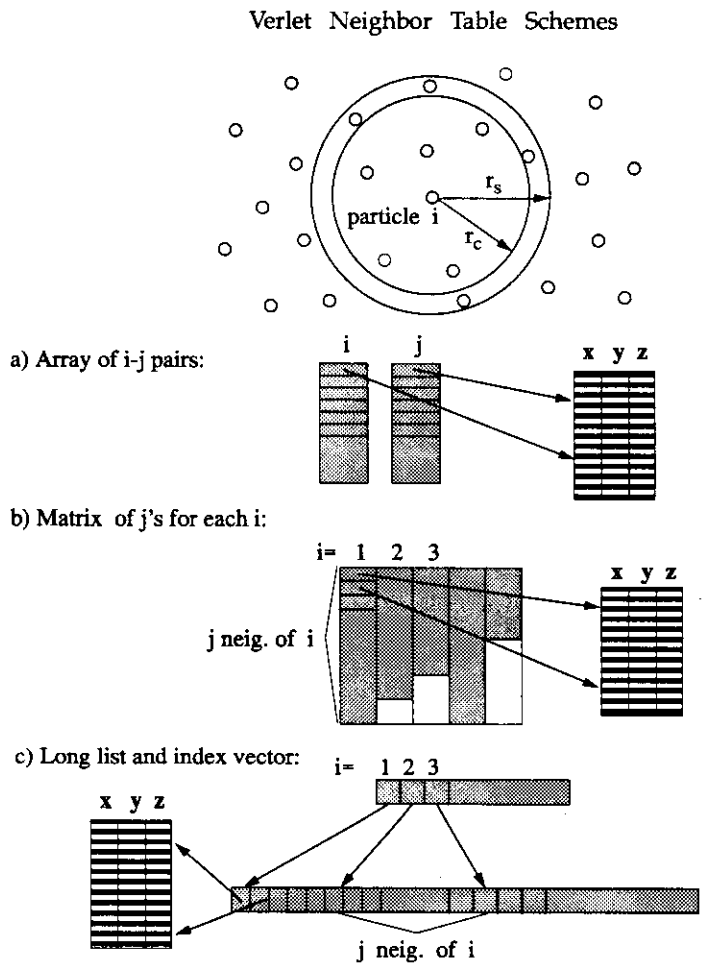


Figure 5: Three commonly used data structures for Verlet neighbor tables.

The overhead for constructing this table is significant compared to the force computation, and therefore, the

tables must be reused for several time-steps to amortize the cost of their construction. At a given time step, the tables are constructed by finding all pairs of particles whose separation is less than $r_s = r_c + \delta_s$, where δ_s is a "safety" distance. The tables can be reused for as long as no pair of particles originally further apart than r_s get closer together than r_c. This is a time of order δ_s/v, where v is a typical velocity. In principle the table construction is an $O(N^2)$ process but if one uses the cell method described in the previous section then the process becomes $O(N)$. The main difference is that in the inner loop of the cell method instead of calculating forces one computes *table entries* and saves them to be used later in the force computation.

There are different ways to represent the Verlet tables in terms of data structures. Fig. 5 shows three of the commonly used data structures. In approach (a) the neighbor table consists of "pairs" of indices into the particle arrays. The force calculation is done by fetching –or vector loading– the coordinates of the particles according to the $i-j$ indices in the table. Another approach (b) is to have a matrix in which each column contains the indices of the j neighbors for a given i particle. A memory efficient scheme is shown in (c) where a single one-dimensional array holds all the indices to particles' neighbors and an auxiliary *index array* gives entry points as a function of particle number. Scheme (a) is well-suited for fast long-vector operations (gather) but the data dependencies could produce the lost of results when one stores the accelerations. One solution to this problem is Rapaport's "layers" method[12,107]. In this approach one constructs "pockets" of pairs which are free from data dependencies by restructuring the loops over cells and particles[12,107]. Scheme (b) avoids the data dependency problem of scheme (a) but reduces the length of vectors to be of the order of the number of neighbors of a single particle. Scheme (c) is the best although it implies some amount of additional indirection. A comparison of link cell *vs* Verlet tables on vector computers is done in ref. 108.

3 Parallel Molecular Dynamics.

The key issues for implementation of molecular dynamics on modern parallel computers are communication and load-balancing. In short range molecular dynamics applications, as discussed in detail in the next two sections, each processor handles the coordinate updates of some number of particles (typically a few hundred up to a few tens of thousands) and a corresponding fraction of the force calculations. The amount of arithmetic needed to perform a time step depends only on the algorithm and is regarded as the "useful" work done. By contrast, time spent with processors idle and time spent moving data between processors or between CPU's and memory is regarded as overhead to be minimized.

Processors must synchronize with each other at critical points within the program for algorithmic reasons. For example, all processors must complete the distributed force calculation before any processor can begin to update its particles' coordinates. The load balancing issue is whether the work to be done is sufficiently uniformly distributed among processors that there are not significant periods spent with idle processors waiting for a slower companion at such a barrier. At the most basic level, load balance can be achieved by distributing the particles and force calculations as uniformly as possible. For systems which are more or less homogeneous, we can achieve this basic level of load balancing by regular spatial partitioning of the volume among cells.

Additional load balancing considerations arise depending on how often the processors must synchronize during the force calculation. In the extreme case of SIMD hardware, the processors would synchronize after every instruction. In a more typical implementation of the data-parallel model, they would synchronize at each communication step. The longer the processors can run without synchronizing, the better they can average out minor variations of execution time between them and the better the load balance.

In order to calculate forces, each pair of interacting particles must have their mutual force calculated by bringing together the two sets of coordinates at a single CPU. This requires interprocessor communication for those pairs of particles that are assigned to different processors. In the data parallel model, where the parallelism is expressed at the finest granularity that is associated with the problem independent of the number of processors, it is often hard for the programmer to conveniently distinguish the "real" communications between processors from the virtual communications that occurs when data at the fine granularity is moved around. For this reason, most of the efficient parallel implementations of MD force calculations use a MIMD model where the interprocessor

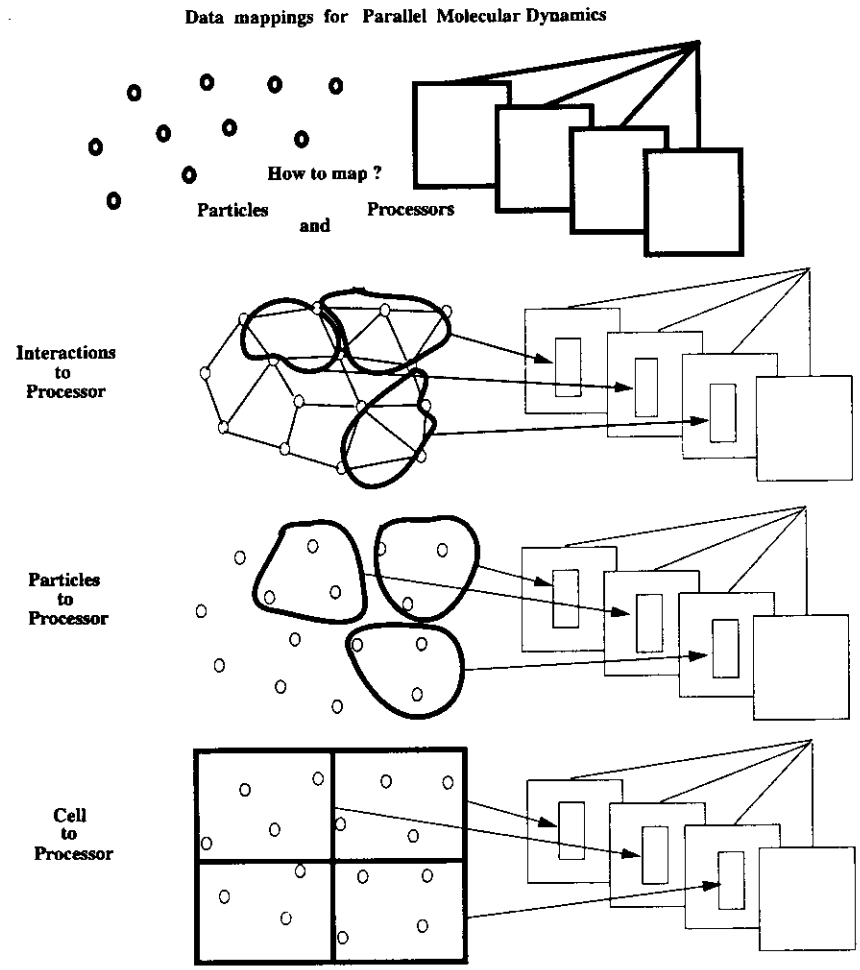


Figure 6: Three examples of data mappings for parallel Molecular Dynamics.

communication is directly visible and controlled at the top level. The programming paradigm is that of the SPMD (Single Program Multiple Data) model. The computational model that these methods use is that of a network of processing nodes. These nodes are off-the-shelf processors of an MPP computer or a cluster of workstations. Since the main goals is to minimize communications costs and exploit locality, one explicitly deals with the processor granularity rather than subsume it in a higher fine-grained model. Long data blocks are transmitted through the network more efficiently than short blocks. The two parallel methods described in sections 4 and 5 are implemented according to this SPMD paradigm.

In the linked-cell method used by Mel'cuk, Giles and Gould on the SIMD Connection Machine CM-2[58], the particles were distributed among fine grained spatial cells. The number of cells used depended only on the interaction range and was independent of the number of physical processor nodes. Each cell contained only a few particles. During the force computations, the particles in each cell were "moved" in a regular pattern past their neighboring cells. Between moves, all the forces involving particles in the cells are computed. Because of the SIMD nature of the machine, all cells have to wait for the slowest cell to complete its work, and therefore the efficiency due to load imbalance is proportional to $(n_{cell}/n_{cell-max})^2$, where n_{cell} is the typical number of particles in a cell and $n_{cell-max}$ the maximum number of particles in the cell with highest density. As the number of cells grows, this number gets smaller and smaller due to density fluctuations. Lynch and Tamayo[109] observed this problem in a CM Fortran implementation of the cell method where the program's speed was 2.6 times slower than an equivalent MIMD method (see Ly94 and Lo94-2 in Table I). Other studies of Data Parallel implementations have been done by Rapaport[46], Nielsen *et al*[144], and Ossadnik and M. Gyure[120]. An interesting issue that deserves further study is

that regarding the extensions to the data parallel programming model, for example in the direction of global-local capabilities[110] (such as the ones offered in CMFortran and HPF), that would allow data parallel programs to call message-passing local subroutines and in this way attain the same performance as MIMD programs.

Now we will briefly review some of the most commonly used partitioning and mapping strategies for parallel MD.

3.1 Cells-to-processors Mapping

The most common partitioning strategy is to map spatial cells to processors. The main reason for this is because interprocessor communications costs, as measured by the latency and bandwidth, are significantly larger that in-processor memory data transfer costs. The implication for MD simulations is that in addition to minimizing the number of floating point operations –which reflect local processor-memory interactions– as in serial algorithms, one has to organize data so as to minimize communication requirements across processors and get the most reuse out of any data that must be sent from processor to processor. The most natural way to achieve this is to capture the spatial locality inherent in the problem in the computational locality of the parallel processing system by mapping processors to spatial cells, where the granularity of the cells is related to the number, complexity and connectivity of the processors in the machine. Particles in a given cell-processor will necessarily interact with particles in neighboring cell-processors. Computation of such interactions requires that the coordinates of the particles be communicated across cell-processor boundaries. Efficient MD algorithms will minimizes the net amount of data going across cell-processor boundaries. There are different implementations of cell-to-processor algorithms. For example one approach is to allocate the cells as dynamic arrays that change size according to occupation numbers. Many examples of cell methods can be found in the literature: Kalia *et al*[117], Rapaport[12,13,14,46,46], Smith[16], Brown *et al*[48], Lin *et al*[49], Plimpton[20,21], Plimpton and Heffelfinger[19], Form *et al*[52], Buchholtz, and Pöschel[53], Plimpton[20,21], Raine *et al*[54], Fincham and Smith[54], Esselink *et al* [55], Hedman and Laaksonen[56], Ossadnik and Gyure[120], Bruge and Forlini[57], Melcuk *et al*[58], Beazley and Lomdahl[59], Lomdahl *et al*[60,61], Beazley *et al* [62], Tamayo *et al*[63], Tamayo and Giles[64,119]. Spatial decomposition is also used in the implementation of EulerGromos[111]. The partitioning of the computational domain in cells gives rise to some additional problems related with initialization schemes and random number generation[121].

3.2 Particles-to-processors Mapping

In this scheme one finds a partitioning strategy to divide the particles evenly across processors. This mapping is sometimes used in combination with Verlet tables so the amount of global communication is minimized. Tamayo *et al*[63] developed a SIMD method in which global communications are used to send the coordinates of each particle to its neighbors. This is done over several routing cycles by using a communication primitive that sends a message from each processor to a destination queue in any other processor. Once that all the messages have been received then the force computation is performed. This method is direct and easy to implement but ignores locality and as a consequence communication times dominated the execution times[63]. It will be interesting to see if fine-grained neighbor table methods like this can be made practical by using light-weight *active-messages* [112] or shared memory schemes. For other approaches using Particles-to-processor mappings see for example the EGO program from K. Schulten and collaborators[113,114,115,116], Kalia *et al*[117], Scott *et al*[118] and the atom-decomposition method of Plimpton[21].

3.3 Interactions-to-processor Mapping

In this approach the partitioning is done by distributing the interactions across the processing nodes. Mapping interactions to processors is usually done for relatively small systems. See for example the force-decomposition methods of Plimpton[20,21]. For larger systems it is better to map particles-to-processors or cells-to-processor. Sometimes the interactions-to-processor approach is combined with the use of the *replicated data* method which we describe in the next paragraph.

3.4 Replicated Data Method

Particles coordinates are replicated in each processor, usually by doing a broadcast operation, and then each processor computes a subset of the interactions. The problems with this scheme are its lack of scalability and how to obtain good load balance. This method is often used to simulate systems of a few thousand particles that can fit in a single node's memory. It has been used successfully in several molecular modeling and simulation programs such as CHARMM[122], where a hash coding scheme is used to achieve load balance, and GROMOS[124,125]. Variations of this method have been studied by Smith and Forester[16,50,51] and by Plimpton[20,21].

The use of parallel multi-color algorithms for internal forces constraints have been investigated by Müller-Plathe and D. Brown[123].

4 A Parallel Cell Method: Los Alamos's SPaSM.

In this section we will describe and analyze a representative parallel cell method. This method is based on the SPaSM MD program developed at Los Alamos Natl. Lab.[59,60,61,62,126].

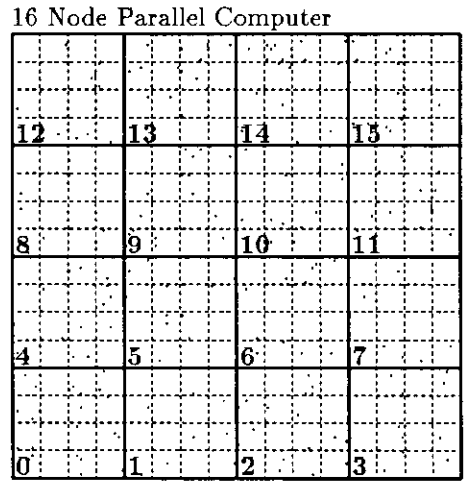


Figure 7: SPaSM: Processor and cell assignment.

4.1 Basic Description.

We illustrate the algorithm in 2D but the method naturally extends to 3D. Let us start by describing the data structures. We consider space to be a rectangular region with periodic boundary conditions. This region is then subdivided into large cells which are assigned to the processors. Particles are assigned to processors geometrically according to the particle's coordinates. For a 16 processor machine, space would be subdivided as shown with solid lines in Fig 7.

The numbers in each region indicate the processor that would be assigned to that region of space. Points correspond to a sample set of particles. For solids, the particles are usually uniformly distributed and each processor could be assigned hundreds to hundreds of thousands of particles.

For a large set of particles, the region assigned to each processor will have dimensions significantly larger than the interaction cut-off distance, r_c. In general, there will be a large number of particles on each processor that do not interact with each other. We seek a method to organize the particles so that a particle's neighbors can be quickly located for the force calculation and so the number of interactions calculated is minimized. To solve the

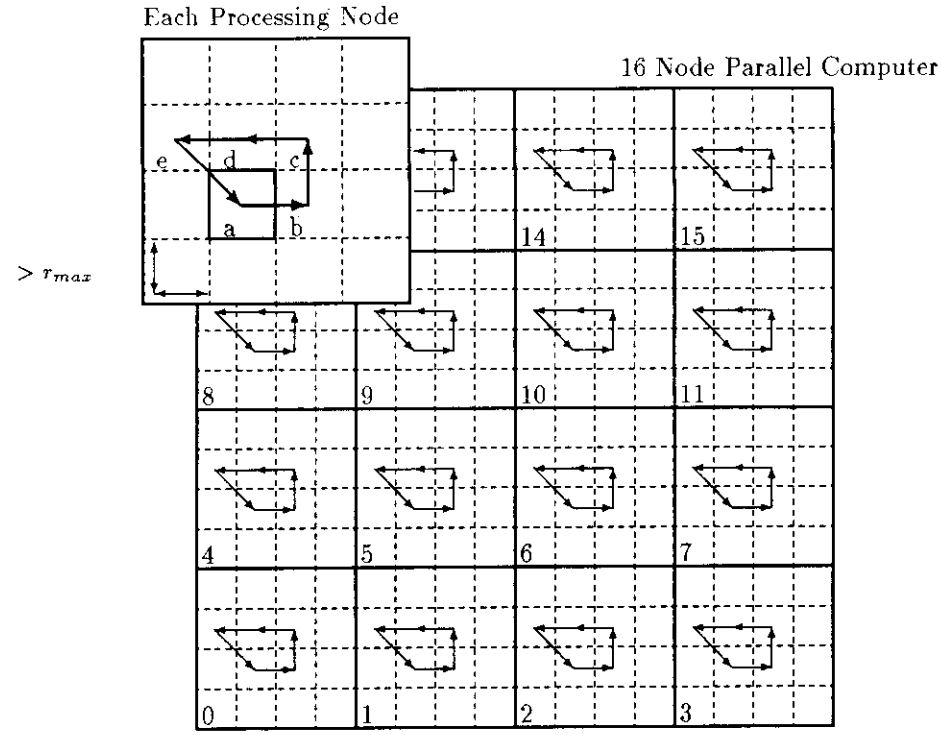


Figure 8: SPaSM: Force calculation.

problem, the region on each processor is subdivided into a large collection of small cells. Each cell created is chosen to be slightly larger than the interaction cut-off. Particles are then assigned to an appropriate cell geometrically as before. It is important to note that number of cells created depends only on the size of the region and the interaction cut-off, r_c, not on the number of processors available. For large simulations it is possible to subdivide the space on each processor into thousands of cells. In Fig. 7, the dashed lines indicate the smaller cells created on each processor. In this case, 16 cells per processor are being used. Each cell has dimensions larger than the cut-off distance. The same number of cells are created on every processor.

This cell structure forms the foundation of the algorithm. For storage, each particle consists of a C language structure containing the position, velocity, force, and a particle type. For memory management, associated with each cell is a small block of memory where the particles are stored sequentially in a list. This method of storage is important for the communication aspects of our interaction calculation since we communicate entire cells, not individual particles. The sequential nature of the data allows us to easily communicate the entire contents of a cell by simply sending a small block of memory. In our early implementations, we fixed the size of each block. Recently we have have added a dynamic memory management scheme that allows variable block sizes.

With the cell structure now in place, interactions can be effectively calculated. The calculation of forces on a single particle only involves the particles in the same cell and neighboring cells. The calculation of forces for all of the particles in a cell is a two step process. First, all of the forces between particles in the same cell are calculated. Next, forces from particles in the neighboring cells are calculated by following an interaction path[58] that describes how we compute the interactions with neighboring cells. In 2D the path is as shown in Fig. 8.

Forces are being calculated for the particles in cell (a). Interactions between all particles in cell (a) are first calculated. Interactions with particles in neighboring cells are then calculated in the order shown (b-e). Once interactions with cell (e) are calculated, the process is complete. As we calculate forces with the neighboring

cells the total force is accumulated by the original cell and the cells along the path (using Newton's third law). To calculate all forces, this procedure is carried out on all cells on all processors. Cell (a) will accumulate the interactions from its lower neighbors when they calculate their interactions.

On each processor, the calculation of forces proceeds sequentially through all of the cells. This process then occurs in parallel on all processors. Globally, the processors are assumed to be operating in a loosely-synchronous mode where each processor is calculating the forces for the same internal cell at approximately the same time. For most cells on a processor, all of the neighboring cells are on the same processor. This allows most cells to calculate interactions without any communications. However, for cells along the edge of each processor, we must calculate interactions between cells on different processors. When this occurs, message-passing communications are utilized. Particles are sent to neighboring processors and received from other processors. When communication is necessary, all of the processors synchronize and participate in a send-and-receive type communication. Since each processor has the identical cell structure and is operating on their cells in the same order, when one processor needs to calculate an interaction with particles on another processor, all processors will have to do this. Each processor sends the particle data for the cell to the appropriate processor and simultaneously receives particles from another processor. The force calculation then proceeds with each processor operating on the particles it received. The key idea is that whenever a processor boundary is crossed along the interaction path, all processors participate in a synchronous communications step and the force calculation continues. It should be noted that synchronization only occurs in message-passing. At all other times, the processors are running asynchronously.

Interaction Algorithm (Executes on all processors simultaneously)

```
 Variables:
   Cells(Xcells,Ycells) : 2d array of cells on the processor.
   Path[PathLength]     : Array of increments describing interaction path.
   CellBuffer           : Temporary storage for a single cell
   CurrentCell          : Pointer to the cell for which interactions are being calculated.
   PathCell             : Current cell on interaction path.  Can be on a neighboring processor.
   InteractCell         : Same as PathCell except with periodic boundary conditions.
   ReceiveCell          : Pointer to where incoming particles are received during communications.
   HomeProcessor        : The address number of this processor.

ComputeAllInteractions()
   For i = 0 to Xcells
         For j = 0 to Ycells {
           ComputeInteractionsSameCell(Cells(i,j))
           CurrentCell  = Cells(i,j)
           PathCell     = (i,j)
           InteractCell = (i,j)
           For k = 1 to PathLength {
                 PathCell = PathCell + Path[k]
                 InteractCell = InteractCell + Path[k]
                 If (Processor(InteractCell) != HomeProcessor) {
                          Destination = Processor(InteractCell)
(1)                       Source      = Processor((Xcells,Ycells) - InteractCell)
(2)                       If (Processor(PathCell) = HomeProcessor)
                               Then ReceiveCell = Cells(i,j)
                               Else ReceiveCell = CellBuffer
                          SendAndReceive(CurrentCell-->Destination, Source-->ReceiveCell)
(3)                       CurrentCell = ReceiveCell
(4)                       InteractCell = InteractCell mod (Xcells, Ycells)
                 }
                 ComputeInteractions(CurrentCell, InteractCell)
           }
         }
end(ComputeAllInteractions)

ComputeInteractionsSameCell(C)
```

```
    For I = 1 to (NumParticles(C) - 1)
         For J = I+1 to NumParticles(C)
            F = ComputeForce(Particle(I), Particle(J))
            Particle(I).Force = Particle(I).Force + F
            Particle(J).Force = Particle(J).Force - F
end(ComputeInteractionsSameCell)

ComputeInteractions(Cell1, Cell2)
    For I = 1 to (NumParticles(Cell1))
        For J = 1 to (NumParticles(Cell2))
            F = ComputeForce(Particle(Cell1,I), Particle(Cell2,J))
            Particle(Cell1,I).Force = Particle(Cell1,I).Force + F
            Particle(Cell2,J).Force = Particle(Cell2,J).Force - F
end(ComputeInteractions)
```

---

(1) The source processor corresponds to the processor on the opposite boundary as the destination processor.

(2) If the path leaves the processor, particles will be received from a neighboring processor and are stored in *CellBuffer*. If the path reenters this processor, particles that were sent out earlier are being sent back and are stored in their original location *Cell(i,j)*. Forces that were calculated with particles on nearby processors are returned with the particles at this time.

(3) This switches the processing to the proper set of particles after communications.

(4) This imposes periodic boundary conditions on the interaction path. When particles are received from a neighboring processor, this forces the interaction path to wrap around to the other side to work with the received particles. In one sense, this processor takes over the interaction path from the neighboring processor.

The main feature of this algorithm is that each processor must simultaneously manage its own cells and cells received from its neighbors. To do this, two separate interaction paths are utilized (these are described by the variables *PathCell* and *InteractCell*). One path (*PathCell*) is used to manage the particles that belong to this processor. It may leave this processor when processor boundaries are crossed, but always keeps track of where particles are located at any particular time in the algorithm. The second path (*InteractCell*) describes the actual computations that need to be performed by this processor. The path is identical to the first path except that we impose periodic processor boundary conditions. When the interaction path crosses a processor boundary, new particles will be received from a neighboring processor across the opposite boundary. With processor periodic boundary conditions, the second path will properly describe the required interactions with the received cells and cells on this processor.

It was noted earlier that the particles on each cell are stored sequentially in memory. This allows us to send all of the particles in a particular cell by simply sending a block of memory through the data network. This type of sending is faster than sending one particle at a time that would be required if the particles in each cell were scattered throughout memory. On each communication we send particle positions, accumulated forces, and types. The velocity of each particle is not needed to calculate forces so velocity data are not sent. The selection of the data to send is done by breaking particle data into two data structures (one used for communications and one that is only used locally). Each cell actually has two regions; the particle data which is communicated to neighboring processors and the particle data which is kept locally.

Our algorithm can also be viewed as a compromise between sending individual particles and a minimal message-passing scheme. In this latter scheme one could imagine all outgoing particles in a given processor be buffered and only sent when all particles tagged for a given destination processor are ready. However, such a scheme would be significantly more costly in terms of local memory and would probably prohibit runs with 10^8 particles.

After each force calculation, particle positions and data structures must be updated. This involves moving particles between cells and processors if necessary. Moving particles between cells on the same processor is

easily performed by copying data. When particles move between processors, one can use either asynchronous or synchronous message passing strategies. Asynchronous methods allow each processor to both send and receive particles from the network while checking all of the particle coordinates. This method is faster, but it can cause severe problems with network traffic and can introduce randomness into the propagation of round-off error (which can cause a program to generate slightly different results on "identical" runs). Synchronous methods require coordination between processors and run between 4-5 times slower. However, they eliminate all of the problems associated with an asynchronous approach while making it easier to perform dynamic memory management.

4.2 Scaling Properties and Performance

Suppose, for the purpose of analysis, that the particles are uniformly distributed and that the region assigned to each processor is square. Further suppose that this square region is subdivided into a collection of square cells of equal area and define the following variables.

N = Number of particles on this processor.
c = Number of cells in each direction.

We first determine the approximate number of interactions. The number of particles in each cell is given by $N_c = N/c^2$. The number of interactions calculated by each cell is then given by $I_s = N_c(N_c-1)/2$ and $I_n = 4N_c^2$, where I_s is the number of interactions from particles in the same cell and I_n is the number of interactions calculated from the 4 neighboring cells during the force calculation. The total interactions per cell is then given by $I_c = 9N_c^2/2 - N_c/2$. Summing over all of the cells on the processor and substituting for N_c the total number of interactions calculated is given by

$$I_t = c^2 I_c = \frac{9N^2}{2c^2} - \frac{N}{2}. \tag{11}$$

Next, we determine the amount of message-passing required by the force calculation. We examine the interaction path and count up the number of times a processor boundary is crossed. The right edge contributes $2(c-1)$ passes, the top $2(c-2)$, the left $2(c-1)$ and the two top corners contribute 7 passes. The total number of message passes is given by

$$M_t = 6c - 1. \tag{12}$$

The amount of data being sent on each message-pass depends on the size of each cell. The number of bytes per cell is given by $D_c = N_c\beta = N\beta/c^2$ where β is the number of bytes per particle. The total amount of data sent is

$$D_t = (6c-1)D_c = \frac{(6c-1)N}{c^2}\beta. \tag{13}$$

The total interaction time can now be approximated. Define the following variables :

α = Interaction time (sec/interaction).
δ = Data transfer rate (bytes/sec).
γ = message-passing overhead for each send (sec).
β = bytes per particle.

The total interaction time is given by the following

$$T = I_t\alpha + M_t\gamma + \frac{D_t}{\delta} \tag{14}$$

$$= \left[\frac{9N^2}{2c^2} - \frac{N}{2}\right]\alpha + (6c-1)\left[\frac{N\beta}{c^2\delta} + \gamma\right]. \tag{15}$$

All the constants can be defined in terms of machine architecture specifications such as the Flop rate and network transfer rate. They can also be determined empirically.

A 3D scaling model can be developed in the same was as the 2D model. The corresponding 3D equation is given by

$$T = \left[\frac{27N^2}{2c^3} - \frac{N}{2}\right]\alpha + (16c^2 - 3c + 1)\left[\frac{N\beta}{c^3\delta} + \gamma\right]. \tag{16}$$

It is interesting to note that the amount of message-passing is now a quadratic function of c. This indicates a dramatic increase in the amount of message-passing involved over our 2D simulations.

The choice of the number of cells has a dramatic effect on the overall performance. Looking at equation (5) for the total number of interactions, we see that the number I_t decreases inversely as the square of c. This alone suggests that we should have as many cells as possible to reduce the number of interactions calculated. The minimum number of interactions occurs when the cell size is exactly equal to the cut-off distance (if the cells are smaller than the cut-off the force calculation will be incorrect).

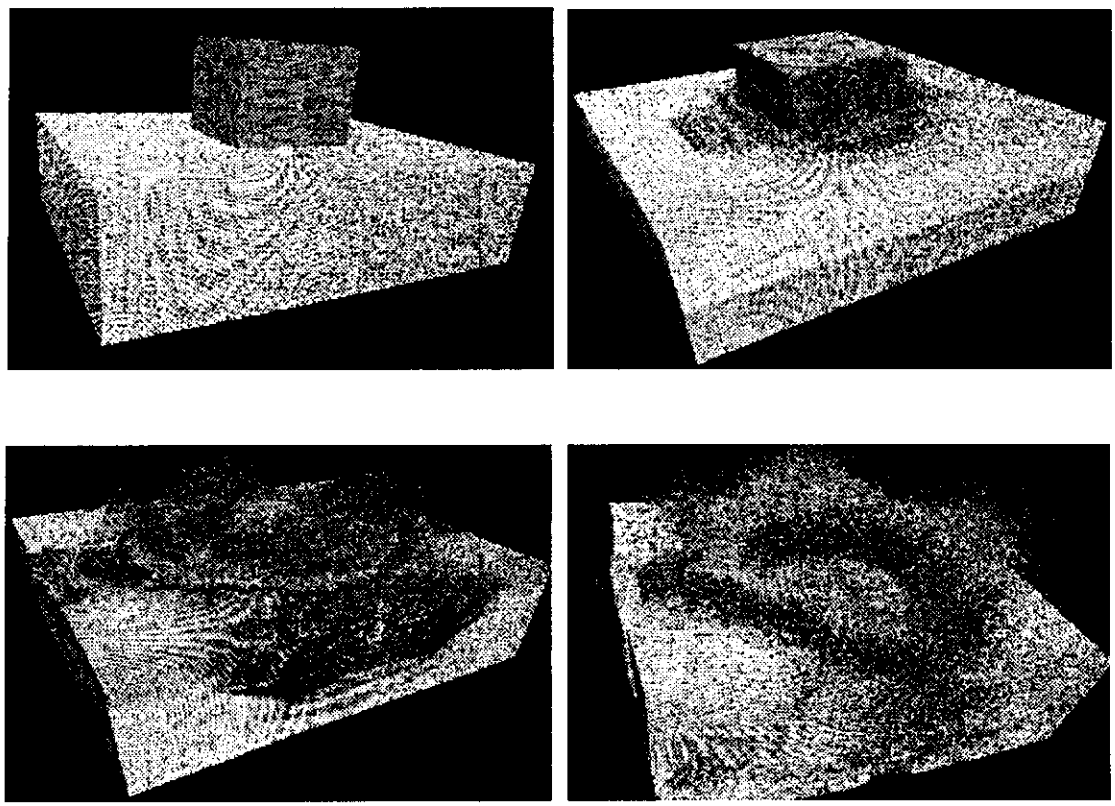


Figure 9: Molecular Dynamics impact simulation with 11.3 million particles using SPaSM. The small block impacts the larger plate at approx. 80 % of the sound velocity. The small block (projectile) contains approx. 1 million atoms and the larger block (target) 10 million atoms. Both have initially an FCC lattice structure.

Increasing the number of cells by increasing c causes the number of message-passes to increase linearly, but the amount of data sent decreases as $1/c$. Generally the transfer time is significantly longer than the overhead of setting up a message-pass so it is an advantage to increase the number of cells in this case as well. If the overhead for initiating a message send is extremely high, then it may be a disadvantage to increase the amount of message-passing. This does not seem to be the case on the CM-5.

A change in density will cause the number of particles per processor to fluctuate. From our formula, we see that the total time will vary as N^2 with a density fluctuation. The effect of changing the number of processors can also be measured here. Suppose that the number of processors is doubled while the cell sizes remain fixed. The

effect of this is to chop the region assigned to the original processor into two equal rectangles with half as many cells as before. The number of interactions on each processor is then cut in half. Since the interaction calculation tends to dominate the overall iteration time, the effect of this is to cut the iteration time in half.

In summary, the model tells us that we should make as many cells as possible on each processor. It also tells us that doubling the number of processors will cut the iteration time in half. These observations will be analyzed further with our actual timing results.

The scaling model should give reasonable results regarding the behavior of the algorithm if the constants can be determined. The constants can be determined using fixed hardware performance specifications and empirical observations. For a CM-5 without VUs, reasonable values of the constants are given below :

α = 9×10^{-6} sec/interaction
δ = 7 Mbytes/sec
γ = 100μsec
β = 60 bytes

α is the most difficult constant to determine. Our determination involved measuring the calculation rate in Flops for our code, and determining the average computation speed of each processing node. The number of floating point operations per force calculation is then determined and the time per interaction can then be approximated from this data. We obtained speeds of approximately 1.8 GFlops on the full CM-5 with approximately 16 floating point operations per interaction. These quantities were used to determine the value of α given above. The transfer rate of 7 Mbytes/sec was also a measured quantity.

Table II. Update times for 2D SPaSM simulations on 1024-node CM-5.

Particles	Actual	Scaling Model
16384	0.0057	0.0040
65536	0.016	0.013
262144	0.053	0.047
1048576	0.19	0.17
4194304	0.69	0.67
16777216	2.56	2.63
67108864	8.83	10.42

Using these constants, we compare the scaling model with actual results for various 2D runs on a 1024-node CM-5. For the model, one sets N to the number of particles/processor. In this case, the number of particles is divided by the number of nodes. The results are shown in Table II.

Table III shows the performance of the 3D version of code on both a 1024 processor CM-5 and 128 processor Cray T3D. For the timings, the atoms have been arranged in an fcc lattice with reduced density $\rho = 0.8442$ and reduced temperature $T = 0.72$. A Lennard-Jones cutoff of $r_c = 2.5\sigma$ was used.

In both cases, there is a near-linear relationship as the simulation sizes are increased. All simulations were performed in double precision except those runs denoted by (SP) which were performed in single precision. These runs were performed using a recently developed dynamic memory management scheme for allocating cells. This has allowed us to substantially increase the simulation sizes beyond those reported in [59,60,61,62,126] without a significant loss in overall performance.

SPaSM is been used in large-scale fracture, crack propagation and impact computational experiment. Fig. 9 shows a typical impact simulation with 11.4 million particles.

Table III. Update times for 3D SPaSM simulations.

N	N_{own}	machine / Num.of procs.	Time/Step [secs] T_{force}	T_{comm}	T_{redist}	T_{total}	Time/Part-Update [μsecs]
1000000	976	CM-5/1024	0.25	0.12	0.02	0.39	0.39
5000000	4883	CM-5/1024	1.16	0.33	0.11	1.60	0.32
10000000	9765	CM-5/1024	2.29	0.48	0.21	2.98	0.30
50000000	48825	CM-5/1024	11.26	1.83	1.11	14.20	0.28
100000000	97650	CM-5/1024	23.7	2.50	1.99	28.19	0.28
150000000	146484	CM-5/1024	34.8	3.52	2.94	41.26	0.27
300800000	293750	CM-5/1024	78.17	5.45	6.97	90.59	0.30
300800000	293750	CM-5/1024 (SP)	102.34	17.14	5.43	124.91	0.41
600000000	585937	CM-5/1024 (SP)	201.37	29.78	10.58	241.73	0.40
100000	781	T3D/128	0.066	0.021	0.006	0.093	0.93
500000	3906	T3D/128	0.307	0.056	0.024	0.387	0.77
1000000	7812	T3D/128	0.62	0.05	0.05	0.72	0.72
5000000	39062	T3D/128	2.96	0.61	0.29	3.86	0.77
10000000	78124	T3D/128	5.80	0.64	0.49	6.93	0.69
50000000	390625	T3D/128	28.06	2.51	2.52	33.09	0.66
75000000	585937	T3D/128	42.02	1.14	3.79	46.94	0.63

5 A Parallel Verlet Table Method: Boston University-TMC's MD program.

In this section we describe a representative parallel Verlet table method. It is an improved version of the Boston University-Thinking Machines Corp. MD program which was first introduced in ref. 64. The original program is written in C (3200 lines) with message-passing calls using the CMMD CM-5 library.

5.1 Basic Description.

The use of Verlet neighbor tables inside each processor allow us to save the computation of pair-interactions not needed for the force calculation. The cost one has to pay for this savings is the additional memory to store the tables and the additional complexity of the code to build them. The method divides the computational space in spatial domains that are assigned to each processor (cell-processors) and subdivides the space inside the cell-processors with internal sub-cells to optimize Verlet neighbor table construction. The sub-cell and cell-processor layouts are similar to the algorithm described in the previous section. At any stage of the algorithm, each particle is "*owned*" by a cell-processor. Particles are distributed in such way that each cell-processor is associated with a rectangular volume of space and the particles it owns are those contained in this "proper" volume as is shown in Fig. 10. We will assume that the processor cells are large enough that the cell length in each direction is longer than the cutoff distance ($r_s < l$). Every particle inside the interaction range of a given particle lies in that particle's own cell or one of its 26 neighbors. An "extended" space is defined to include the particles in neighboring cells lying within an interaction range from the boundary. If a particle's coordinates do have to be communicated from one cell-processor to a neighboring one, it is likely that they will be needed for *more* than one pair-interaction. This gives means to minimize the data flow required to go from cell-processor to cell-processor by using a block mode communication. This block communication can be implemented with standard message-passing functions. Similar communications schemes have produced very good results in other unstructured problems such as cluster labeling in Monte Carlo simulations[127].

A regular time step of the MD simulation consists of a *communication step* followed by the *force computation* and finally *integration of the equations of motion*. Before sending particle information to a neighboring cell-processor the relevant particles' coordinates inside the *extended* space near the boundary, that we will call "*own-shared*", are gathered in a buffer as determined by a list of pointers corresponding to that extended space region. Then

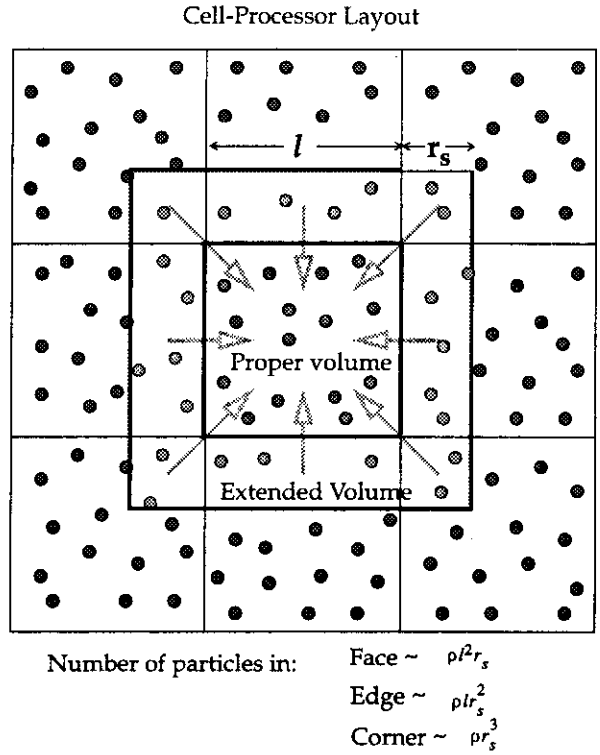


Figure 10: Proper and extended volumes for parallel Verlet neighbor table method.

each cell-processor sends them to a neighboring cell-processor using send and receive "blocking" message passing functions. At the other side the neighbor tables of some particles will include pointers to this buffer of particles that we will call "*visitors*". A set of *own-shared* particles in one cell-processor become the *visitors* in another cell-processor after communication and exchange of particles has taken place (see Fig. 11). A full exchange of particles involves 26 message-passing calls: one for each neighboring cell-processor. This procedure reduces overhead by eliminating intermediate storage and minimizes communication calls. A simple pseudo-code description of the *communication step* is as follows:

Communication Step (Executes on all cell-processors simultaneously)

```
Variables:
 Source                          :  Neighboring cell-processor sending data
 Destination                     :  Neighboring cell-processor receiving data
 Output_Buffer, Receive_Buffer   :  Buffers for own-shared and visitor particle coordinates

Communications()
  For    direction = 1   to  26 {
      Copy_Own_Shared_to_Output_Buffer(direction)
      Source        =   Processor_Neighbor(negative_direction)
      Destination   =   Processor_Neighbor(direction)
      Send_And_Receive(Output_Buffer --> Destination, Source --> Receive_Buffer)
      Copy_Receive_Buffer_to_Visitors(direction)
  }
end(Communications)
```

Once each cell has received the 26 buffers from the neighboring processors then the forces can be computed using the Verlet neighbor tables. The result is the total force on all *own* particles. To perform the force computations four basic data structures are maintained in each cell-processor:

i) An array of *own* particle coordinates and velocities. These are, on average, $\rho \cdot l^3$ *own* particles initially in the cell.

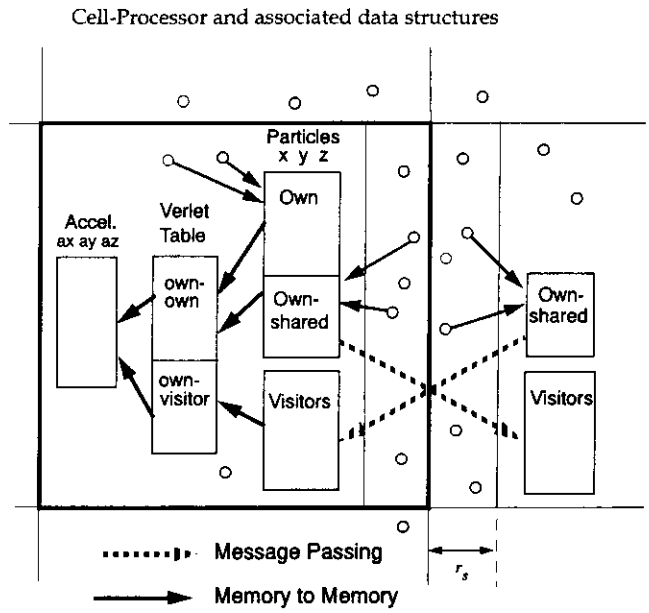


Figure 11: Cell-processor layout and data structures.

ii) An array to hold the coordinates of *visiting* particles which are particles *owned* by one of the 26 neighboring cells but lying within an interaction range of a face (extended space). There are on average $\rho((l+2r_s)^3 - l^3)$ such particles.

iii) A Verlet neighbor table with pointers to each *owned* particle's neighbors. This table includes *own-own* and *own-visitor* interactions but not *visitor-visitor* interactions.

iv) 26 arrays of pointers to *own* particle coordinates to be sent to each of the neighboring cell-processors. These data structures are represented by rectangles in Fig. 11.

The computation of the forces requires to fetch the particles' coordinates according to the Verlet neighbor tables, evaluate the distances and the Lennard-Jones forces, and finally, accumulate the resulting accelerations. An explicit implementation of this procedure is shown below.

Force Computation (Executes on all cell-processors simultaneously)

```
Variables:
  N_own                      : Number of particles in the cell-processor
  x_own[N_own]               : Array of particle coordinates (NOTE xyzxyz storage mode)
  Index[N_own]               : Array of entry points into Verlet neighbor table
  Number_of_neig[N_own]      : Array which contains the number of neighbors for each particle
  Verlet_own_own[N_own_own] : Verlet neighbor table with own-own interactions
  F_own[N_own]               : Array of particle accelerations (NOTE xyzxyz storage mode)
  E_potential, LJ_range_2    : Potential energy scalar variable and potential cutoff (squared)

  Compute_Forces()
  E_potential = 0.0
  For i = 0 to N_own {
       For  j =  Index[i]  to  Index[i] + Number_of_neig[i] {
            n2  =  Verlet_own_own[j]
            dx  =  x_own[i]     -  x_own[n2]
            dy  =  x_own[i + 1] -  x_own[n2 + 1]
            dz  =  x_own[i + 2] -  x_own[n2 + 2]
            r_sqr  =  dx*dx  +  dy*dy  +  dz*dz
```

```
            If ( r_sqr  <  LJ_range_2 )  {
                 r_2  =  1.0/r_sqr
                 r_6  =  r_2*r_2*r_2
                 force  =  24.0*r_2*r_6*(2.0*r_6 - 1.0)
                 E_potential  =  E_potential + 4.0*r_6*(r_6 - 1.0)
                 F_own[i]      =  F_own[i]      +  force*dx
                 F_own[i + 1]  =  F_own[i + 1]  +  force*dy
                 F_own[i + 2]  =  F_own[i + 2]  +  force*dz
                 F_own[n2]     =  F_own[n2]     -  force*dx
                 F_own[n2 + 1] =  F_own[n2 + 1] -  force*dy
                 F_own[n2 + 2] =  F_own[n2 + 2] -  force*dz
             }
        }
   }
   end(Compute_Forces)
```

This code segment corresponds to the *own-own* interactions. The *own-visitor* interactions are computed in a similar way, using the *own-visitor* part of the table, but Newton's third law is not used for these interactions. After the forces have been calculated each cell-processor integrates the equations of motion to obtain new velocities and positions for all the *own* particles.

Now we turn our attention to the construction of the Verlet neighbor tables. In a similar way as in the case of any serial Verlet neighbor table algorithm, the tables have to be reconstructed from time to time. The flow of control of the full program looks like this:

Main Time Loop (Executes on all cell-processors simultaneously)

```
Variables:
 Max_iterations            : Number of time steps
 N_update                  : Frequency of update for Verlet tables

Main()
   Read_Parameters()
   Initialize_Particles_and_Geometry()
   For time =  0  to  Max_iterations {
        if (time  mod  N_update = 0)) Build_Verlet_tables()
        Integrate_Positions()
        Communications()
        Compute_Forces()
        Integrate_Velocities()
   }
end(Main)
```

Building the Verlet tables adds some complexity to the code but saves precious computation time. The neighbor table construction can be summarized by the following operations:

i) A communication step as in the force calculation to update the positions of the *visitors* in each cell-processor. After the exchange each cell-processor scans through all particles (*owned* and *visitors*) and keeps only those particles that still reside in the proper volume of the cell-processor. If there has been no error, i.e. no particle has moved more that δ_s, this is guaranteed to place all the particles in the correct cell without a global resorting. An error can be detected by simply counting particles at this stage.

ii) A communication step to sent the particles' velocities to neighboring cell-processors.

iii) A search of *own-shared* particles to define the 26 lists of pointers to particles that are near the boundaries. Once the lists have been constructed their sizes are sent to the appropriate neighboring cell-processor. This allows each cell-processor to allocate memory for the *visitors* it will receive in the 26 blocks of incoming data at every regular time step.

iv) A communication step to send the newly defined *own-shared* particles' coordinates.

v) Verlet neighbor table construction. Each cell-processor sorts the *own* and *visitors* particles according to the particular sub-cell they belong to, and then scans the coordinates to produce new *own-own* and *own-visitor* tables. These tables take into account Newton's third law only for the *own-own* section.

The memory for the Verlet neighbor tables can be allocated dynamically as needed. In the next section we will see how to compute initial estimates for the table sizes using an approximate scaling model.

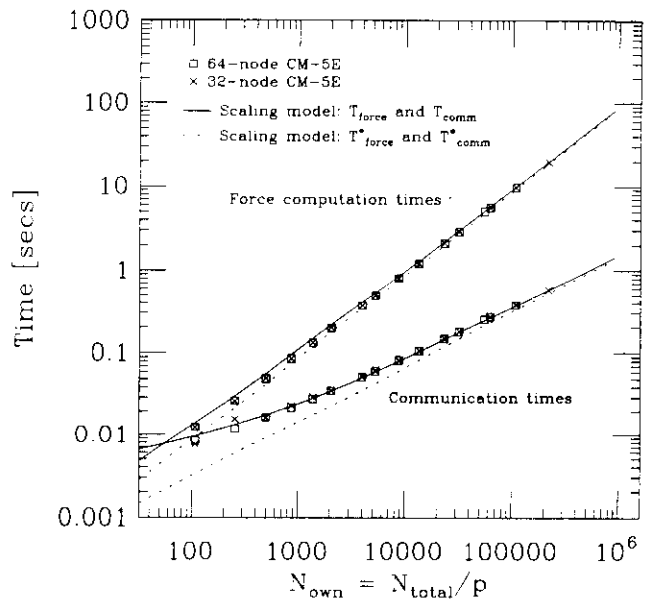


Figure 12: Scaling model: communication and computation times.

5.2 Scaling Properties and Performance.

In order to model the performance of the algorithm, we must estimate the number $N^{int}_{own-own}$ of *own-own* interactions per processor, and the number $N^{int}_{own-visitor}$ of *own-visitor* interactions per processor. Assuming uniform average density ρ these are related by: $2N^{int}_{own-own} + N^{int}_{own-visitor} = \frac{4\pi}{3} N_c N_{own}$, where $N_{own} = \rho l^3$, l is the linear size of the cell-processor region and $N_c = \rho r_s^3$.

We can calculate $N^{int}_{own-own}$ directly from:

$$2N^{int}_{own-own} = \rho^2 \int_{cell} d^3\,\vec{x} \int_{cell} d^3\,\vec{y}\,\Theta(|\vec{x}-\vec{y}| \le r_s) \tag{17}$$

The double volume integral gives,

$$\frac{4\,l^3\,\pi\,r_s^3}{3} - \frac{3\,l^2\,\pi\,r_s^4}{2} + \frac{8\,l\,r_s^5}{5} - \frac{r_s^6}{6}, \tag{18}$$

from which we deduce:

$$N_{own-shared} = 8\,N_c + 12\,{N_c}^{\frac{2}{3}}\,{N_{own}}^{\frac{1}{3}} + 6\,{N_c}^{\frac{1}{3}}\,{N_{own}}^{\frac{2}{3}} \tag{19}$$

$$N^{int}_{own-own} = \frac{-{N_c}^2}{12} + \frac{4\,{N_c}^{\frac{5}{3}}\,{N_{own}}^{\frac{1}{3}}}{5} - \frac{3\,{N_c}^{\frac{4}{3}}\,{N_{own}}^{\frac{2}{3}}\,\pi}{4} + \frac{2\,N_c\,N_{own}\,\pi}{3} \tag{20}$$

$$N^{int}_{own-visitor} = \frac{{N_c}^2}{6} - \frac{8\,{N_c}^{\frac{5}{3}}\,{N_{own}}^{\frac{1}{3}}}{5} + \frac{3\,{N_c}^{\frac{4}{3}}\,{N_{own}}^{\frac{2}{3}}\,\pi}{2} \tag{21}$$

These are the basic formulas we will use to build an scaling model for the timings. When we compare these formulas with real simulation data we can see reasonably good agreement as shown in Table IV. For large systems the values predicted by the formulas fall within 4 % of the measured values.

Table IV. Comparison of values of $N_{own-shared}$, $N^{int}_{own-own}$ and $N^{int}_{own-visitor}$ predicted by the formulas (Eq. 19-21) and simulation data for a typical simulation of a homogeneous liquid (64-node CM-5E).

N	N_{own}	$N_{own-shared}$		$N^{int}_{own-own}$		$N^{int}_{own-visitor}$	
		Model (Eq. 19)	Sim. Data	Model (Eq. 20)	Sim. Data	Model (Eq. 21)	Sim. Data
32000	500	1815	1920	12923	12416	12968	12677
256000	4000	5483	5749	127755	123140	54995	53723
4000000	62500	28489	29698	2247928	2170347	355777	347413

Now we can extend the scaling model to predict timings. We express the time per step as a sum over four terms,

$$T_{total} = T_{table} + T_{force} + T_{comm} + T_{int} \tag{22}$$

where T_{table} is the time to construct the neighbor table; T_{force} the time to perform the force calculation; T_{comm} the communications time, and finally, T_{int} the time to integrate the equations of motion. The scaling of T_{table} and T_{int} is simply a linear function of N_{own},

$$T_{table} = \frac{\sigma N_{own}}{f_{update}} \tag{23}$$

$$T_{int} = \omega N_{own}, \tag{24}$$

The coefficients σ and ω, which can be obtained from a test simulation, characterize the effective time to compute a table entry and to integrate the equations of motion for one particle respectively. f_{update} is the table update frequency. The fact that we use sub-cells to build the neighbor table implies that σ is of the form $27N_c\alpha_{table}$ where α_{table} is the time to compare two particles' distance and produce a table entry if it is less than r_s. The time to perform the force calculation is proportional to the total (*own-own* + *own-visitors*) number of pair-interactions,

$$T_{force} = (N^{int}_{own-own} + N^{int}_{own-visitors})\alpha \tag{25}$$

where α characterizes the time to compute one pair interaction. If we assume that to evaluate the Lennard-Jones force one requires n_{LJ} floating point operations, for example $n_{LJ} = 32$, and α will be equal to n_{LJ}/(flop-rate). T_{force} has a complex functional form involving different powers of N_{own} and then it is desirable to simplify the scaling model by taking only the leading term for $N^{int}_{own-own}$,

$$T^*_{force} = \frac{2}{3}N_c\pi\alpha N_{own}. \tag{26}$$

The communication times scale with the number of *own-shared* particles,

$$T_{comm} = 2N_{own-shared}\frac{\beta}{\delta} \tag{27}$$

where β is just the number of bytes per particle and δ the effective communications rate for message-passing exchanges. Here again we simplify by taking only the leading term,

$$T^*_{comm} = 2 \cdot 6N_c^{\frac{1}{3}}\frac{\beta}{\delta}N_{own}^{\frac{2}{3}} \tag{28}$$

which is equivalent to disregard "corners" and "edges" and consider communications on the "faces" only. The factor of two accounts for the bi-directional nature of the communications.

Taking these approximations into account the scaling model becomes,

$$T^*_{total} = \left(\frac{\sigma}{\mathrm{f}_{update}} + \omega + \frac{2}{3} N_c \pi \alpha \right) N_{own} + 12 N_c^{\frac{1}{3}} \frac{\beta}{\delta} N_{own}^{\frac{2}{3}} = A N_{own} + B N_{own}^{\frac{2}{3}} \tag{29}$$

Where we define $A = \sigma/\mathrm{f}_{update} + \omega + 2N_c\pi\alpha/3$ as the "computation" coefficient, and $B = 12N_c^{1/3}\beta/\delta$ as the "communication" coefficient. Their numerical values will depend upon the particular machine and implementation. The scaling behavior of the algorithm is basically determined by two terms: a "bulk" term that accounts for all the internal cell-processor computation and scales linearly with N_{own}, and a "surface" term representing particle exchanges across cell-processor boundaries and scales with $N_{own}^{\frac{2}{3}}$. Fig. 12 shows the scaling model: T^*_{comm} and T^*_{force}, compared with actual measured timings for the program. We have parameterized the model with values of α, δ, σ and ω measured in one of the simulations with $N_{own} = 32,000$ running on a 32-node CM-5E:

$$\begin{aligned}
\alpha &= 2.22 \times 10^{-6} \quad \text{computation rate in secs/interaction} \\
\delta &= 7 \times 10^{6} \quad \text{effective one} - \text{directional communications rate[Mbytes/sec]} \\
\sigma &= 5 \times 10^{-4} \quad \text{table construction time in secs/particle} \\
\omega &= 5 \times 10^{-6} \quad \text{integration time in secs/particle}
\end{aligned} \tag{30}$$

These numbers allow us to estimate $A = 0.000111$ secs., $B = 0.000145$ secs. and $B/A = 1.31$.
The other (fixed) parameters of the simulation are,

$$\begin{aligned}
\beta &= 32 \quad \text{number of bytes per particle} \\
\rho &= 0.8442 \quad \text{density : number of particles/unit volume} \\
r_s &= 2.5 \quad \text{potential cutoff distance} \\
\delta_s &= 0.3 \quad \text{security distance} \\
\mathrm{f}_{update} &= 25 \quad \text{table update frequency}
\end{aligned} \tag{31}$$

As one can see in Figs. 12 and 13, the scaling model describes reasonably well the behavior of the measured data except for small values of N_{own}. Asymptotically for larger values of N_{own}, the scaling model converges to the measured timings. In the figure we also show the values of the more exact times T_{comm} and T_{force} which fit the data better as expected. The total times, shown in Fig. 13, include the scaling of T_{table} and T_{int} parameterized with the values of σ and ω shown above.

The scaling model also provides a way to estimate the sizes of the Verlet neighbor tables. For example, if one has a homogeneous system one can estimate the initial allocation size of the tables with the leading terms of $N^{int}_{own-own}$ and $N^{int}_{own-visitor}$:

$$N^{size}_{own-own} = \frac{2}{3} N_c N_{own} \pi \tag{32}$$

$$N^{size}_{own-visitor} = \frac{3}{2} N_c^{\frac{4}{3}} N_{own}^{\frac{2}{3}} \pi. \tag{33}$$

These formulas overestimate the sizes slightly. If during the simulation more table entries are required, then the program increases the table size in fixed chunks.

The speed-up S_p of a parallel algorithm characterizes the effective number of processors being used by the simulation. It is defined as the simulation time on one processor divided by the time on p processors,

$$S_p = \frac{T_{serial}}{T_{total}}. \tag{34}$$

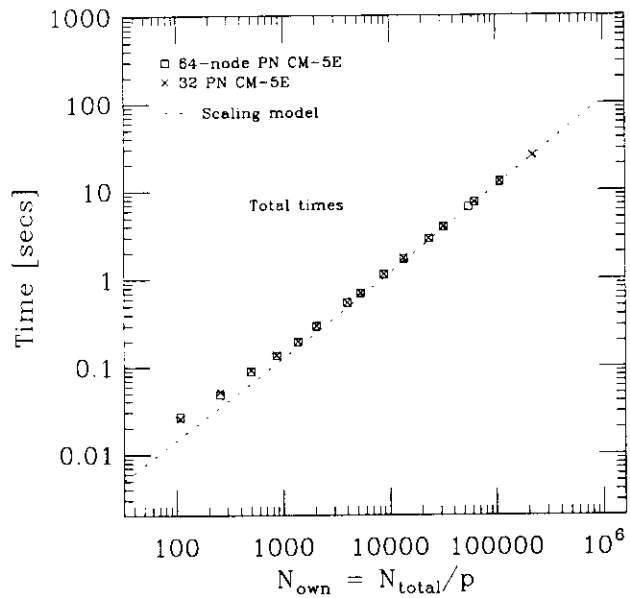


Figure 13: Scaling model: total times.

For our scaling model the speed-up is then given by,

$$S_p = \frac{AN_{total}}{AN_{own} + BN_{own}^{\frac{2}{3}}} = \frac{AN_{own}p}{AN_{own} + BN_{own}^{\frac{2}{3}}} \tag{35}$$

and multiplying numerator and denominator by $(AN_{own})^{-1}$ we obtain,

$$S_p = \frac{p}{1 + \frac{B}{A}N_{own}^{-\frac{1}{3}}}. \tag{36}$$

Now we can compute the efficiency $E_p = S_p/p$,

$$E_p = \frac{1}{1 + \frac{B}{A}N_{own}^{-\frac{1}{3}}} \tag{37}$$

One obtains a very simple scaling picture for this algorithm: for a given value of the ratio (B/A), the relative speed of communications to computation, the efficiency is a universal function of N_{own}. If one plots values of E_p for different system sizes (N_t), and number of processors (p), as a function of N_{own}; all the data should collapse onto a single curve. To test this behavior we plot E_p and compare it with actual efficiency data in Fig. 14. The agreement is better for larger N_{own} as expected. This universal behavior can also be seen in Fig. 2.

Assuming the communications network is scalable, the scaling behavior is completely determined by the number of particles per cell-processor N_{own}. The smaller the ratio B/A and the bigger N_{own} the better the efficiency. In addition one tries to make A and B as small as possible to reduce absolute timings. The *SPaSM* program described in the previous section has very similar scaling behavior. Table V gives detailed timings for different system sizes.

It is common to plot the speed-up as a function of total system size and number of processors $S_N(p)$. Fig. 15 shows $S_N(p)$ for three systems sizes $N = 128,000$, $1,750,000$ and $7,000,000$.

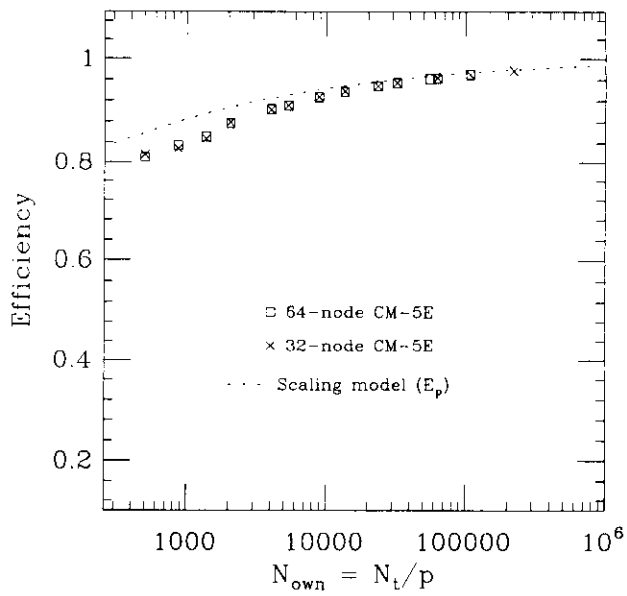


Figure 14: Scaling model: universal form of the efficiency.

To conclude this section let us add that the method described here is general and can be implemented easily in almost any parallel MPP machine or cluster of workstations[128]. By using Verlet tables in combination with coarse-grained processor cells it attains scalability and efficiency in communications and computation even with large numbers of processors. The absolute performance per processor is high as can be seen in the last columns of table I. The price to pay is an increase in the number of data structures (memory) and operations to support the Verlet neighbor tables. The increased complexity makes this method more difficult to implement than a purely cell based method; however, the effort to implement it is comparable to the one of many other typical computational science problems (hydrodynamics, plasma simulations, multigrid solvers, etc.).

6 Perspective: *Quo Vadis* Molecular Dynamics ?

With the capability to simulate faster, larger and more complex systems, we have the possibility to address a number of novel research problems in Statistical Mechanics, Material Science, Chemistry and Biology. In this section we briefly review some of the problems and specific areas in which short-range parallel MD algorithms, such as the ones discussed in this paper, can make substantial contributions before the end of the decade (many of them within the lifetime of a graduate student !).

Simulations of materials at the atomic level. Macroscopic properties of materials (strength, hardness etc.) are determined by the properties and structure of their atomic structure. The use of atomistic computer simulations is very important in the study/analysis/design of current and new models for materials. For example, the study of crack formation in tensile experiments[129,130,131]; shear-stress relaxation[133]; shock waves propagation[132,134] and their interactions with defects in crystal structures[135]; simulations of molecular indentation for elastic and plastic deformations[136,137]; the elastic properties and collision behavior of fullerenes[138]; and hydrogen embrittlement in metals[139].

Table V. Boston University-TMC 3D simulation timings.

N	N_{own}	Machine (procs)	Time/Step [secs]					Time/Part-Update
			T_{table}	T_{force}	T_{comm}	T_{int}	T_{total}	[musecs]
8192	256	CM-5E (32)	0.008	0.027	0.016	0.001	0.052	6.321
16000	500	CM-5E (32)	0.021	0.051	0.017	0.002	0.091	5.705
65536	2048	CM-5E (32)	0.054	0.196	0.036	0.010	0.299	4.562
128000	4000	CM-5E (32)	0.102	0.369	0.053	0.020	0.549	4.286
281216	8788	CM-5E (32)	0.193	0.812	0.083	0.044	1.141	4.058
432000	13500	CM-5E (32)	0.300	1.235	0.107	0.067	1.724	3.990
1024000	32000	CM-5E (32)	0.642	2.914	0.178	0.159	3.930	3.838
2000000	62500	CM-5E (32)	1.172	5.635	0.271	0.310	7.457	3.728
3456000	108000	CM-5E (32)	1.931	9.695	0.382	0.539	12.66	3.664
7023616	219488	CM-5E (32)	3.776	19.587	0.582	1.080	25.26	3.597
16384	256	CM-5E (64)	0.007	0.027	0.012	0.001	0.048	2.926
32000	500	CM-5E (64)	0.019	0.050	0.017	0.002	0.089	2.786
131072	2048	CM-5E (64)	0.052	0.196	0.036	0.010	0.297	2.266
256000	4000	CM-5E (64)	0.102	0.369	0.052	0.020	0.548	2.140
562432	8788	CM-5E (64)	0.188	0.811	0.083	0.044	1.135	2.018
864000	13500	CM-5E (64)	0.290	1.228	0.106	0.067	1.706	1.974
2048000	32000	CM-5E (64)	0.625	2.909	0.177	0.161	3.909	1.908
4000000	62500	CM-5E (64)	1.153	5.632	0.270	0.311	7.434	1.858
6912000	108000	CM-5E (64)	1.912	9.703	0.377	0.531	12.640	1.829
128000	500	CM-5E (256)	0.013	0.048	0.022	0.002	0.087	0.679
1769472	6912	CM-5E (256)	0.123	0.620	0.089	0.035	0.875	0.494
3456000	13500	CM-5E (256)	0.289	1.201	0.142	0.067	1.714	0.496
7023616	27436	CM-5E (256)	0.476	2.434	0.207	0.136	3.283	0.467
43904000	171500	CM-5E (256)	3.430	14.77	0.623	0.832	19.65	0.448

Simulation studies of homogeneous crystallization, nucleation, crystal structure and phase separation. A comprehensive understanding of these complex processes is still missing despite their potentially important industrial applications. Many fundamental questions remain open: what is the dynamics of early crystal growth and what is the relevant droplet definition[140,141,142] ? What is the structure and dynamics of phase separation, domain growth[143], melting[144] and coexistence phenomena[145] ? How important are finite size effects[146,45] ?

Ergodic properties, relaxation and diffusion. There are many problems in the dynamical foundations of Statistical Mechanics which are related with the structural behavior of materials. For example, stochastic transitions and structural relaxation in supercooled liquids and glasses[147,148,149] and the scaling of time-dependent diffusion coefficients[155].

Dynamics of Polymers. The understanding of the Physics of polymeric materials is of great interest from the point of view of fundamental Statistical Mechanics and for its technical applications in industrial processes. Large scale simulations bridge the gap between experiments and theory and allow the building and testing of models[150,151,152].

Atomistic Hydrodynamics simulations. Large scale MD simulations offer a novel approach to the study of hydrodynamic flow instabilities and patterns[153,154] and a powerful tool for the analysis of the intermediate region between hydrodynamic and kinetic behavior[156].

The Physics of granular assemblies and complex fluids. The study of granular flow, friction and dilatancy transitions[157,158,159] is very important for many geophysical phenomena such as rockslides and earthquakes. MD provides a tool to study complex structures such as fullerenes and Si-clusters[80] and to simulate complex flows[160]. The modeling of surfactant behavior[161] (oil and water) and complex fluids in general[162] is important for the Oil and Environmental industries. For more information about granular dynamics see Ristow's review in vol. I of this series.

As we can see there is no lack of important, interesting and challenging problems that can be addressed by parallel MD algorithms. Now that we know of efficient and scalable methods for parallel short-range MD can

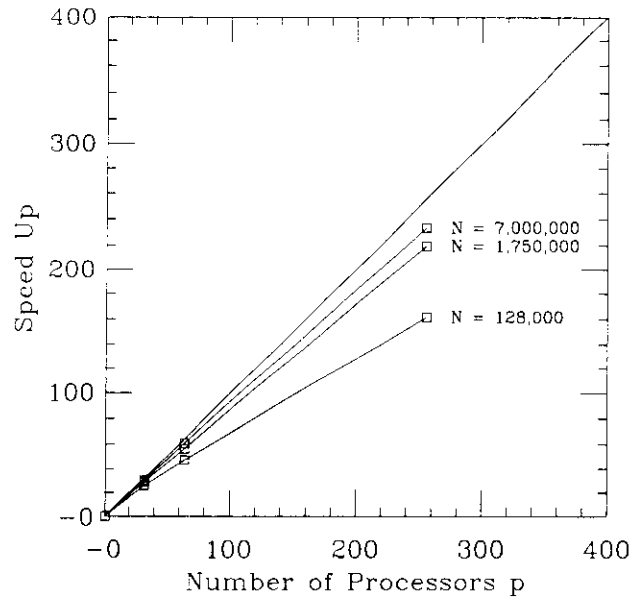


Figure 15: Speed-up as a function of system size and number of processors.

we go home and have a beer ? Perhaps, but there are outstanding problems on the table: load balance for non-homogeneous problems has to be addressed; long range and more realistic potentials have to be incorporated using multipole or alternative methods; the simulation of slow diffusive molecular processes is still a big challenge even though the gap between the time-space scales in the simulations and those analyzed in real experiments has been reduced. The wide range of time scales is still a formidable problem for the simulation of glasses, polymer blends, biomolecules and proteins. The simulation speed for small systems (1000-10,000 atoms) has not improved as dramatically as in the case of large systems. There is a clear need for better methods because even taking into account the doubling of speed of the computer hardware every two or three years, not all the remaining problems will be solved by just faster computers. This is in part a fundamental problem of the MD approach.

We have come a long way from the MANIAC days but we still have a long way to go as the editor of this series, a seasoned Monte Carlo veteran, reminds us: "Nature is still far ahead, if we look at the 10^{25} molecules in a glass of beer[65]." Cheers Mr. Editor, and with this sobering note we come to an end.

Acknowledgments

We want to express our appreciation to the many colleagues and friends who share our interest in Molecular Dynamics and Parallel Algorithms and that have contributed their comments, enthusiasm and data: J. Mesirov, D. Lynch, B. Larson, G. Batrouni, B. Boghosian, D. Rich, M. Krogh, W. Klein, H. Gould, R. Brower, R. Ramos, N. Gross, A. Mel'čuk, L. Tucker, J.-P. Brunet, Moose, A. Mainwaring, M. Drumheller, G. Drescher, S. Plimpton, S. Glotzer, M. Allen, D. Tildesley, K. Schulten, D. Brown, P. Ossadnik, M. Gyure, M. Flanigan, O. Nielsen, M. Bazant, B. Brooks, C. Marshall, R. Nagle, M. Hodošček, G. Seeley, F. Hedman, D. Rapaport, W. Smith, G. Grest, L. Monette, K. Esselink, V. Batista, D. Coker, C. Rebbi, D. Stauffer and P. P. Mario.

We also want to acknowledge the Theoretical Division and the DOE Advanced Computing Laboratory at Los Alamos Natl. Lab., Boston University's Center for Computational Science, the Mathematical Sciences Research

Group at Thinking Machines Corp., and the Naval Research Laboratory for making their computing facilities available to us and for generous support of this project.

References

[1] H. Gould and J. Tobochnik, *Computer Simulation Methods*, Addison-Wesley 1988.

[2] D. Stauffer, F. W. Hehl, N. Ito, V. Winkelmann, and J. G. Zabolitzky, *Computer Simulation and Computer Algebra*, Springer Verlag, Heidelberg, Berlin, 1993.

[3] M. P. Allen and D. J. Tildesley, *Computer Simulations of Liquids*, Clarendon Press, Oxford 1987.

[4] R. W. Hockney and J. W. Eastwood, *Computer Simulations Using Particles*, Mac Graw-Hill, New York 1981.

[5] G. Ciccotti, D. Frenkel and I. R. McDonald, eds., *Simulations of Liquids and Solids*, North Holland, Elsevier Science pub. 1987.

[6] G. Ciccotti and W. G. Hoover eds., *Molecular Dynamics Simulations of Statistical-Mechanical Systems, Procs. of the Int. School of Physics Enrico Fermi*, North-Holland 1986.

[7] W. G. Hoover, *Computational Statistical Mechanics, Studies in Modern Thermodynamics 11*, Elsevier 1991.

[8] W. G. Hoover, *Molecular Dynamics, Lecture Notes in Physics* vol. 258, Springer Verlag Berlin Heidelberg 1986.

[9] F. Yonezama ed., *Molecular Dynamics Simulations, Procs. of the 13th Taniguchi Symposium*, Kashikojima, Japan, Springer Verlag 1990.

[10] G. M. van Waveren, Application of Sparse Vector Techniques on a Molecular Dynamics Program, in *Algorithms and Applications on Vector and Parallel Computers*, H. J. J. te Riele, Th. J. Dekker and H.A. van der Vorst eds. Elsevier (North Holland) 1987.

[11] D. Fincham, Parallel Computers and Molecular Simulation, *Mol. Simul.* **1**, 1 (1987).

[12] D. C. Rapaport, *Comp. Phys. Rep.* **9**, 1 (1988).

[13] D. C. Rapaport, Multi-Million Particle Molecular Dynamics I. Design Considerations for Vector Processing. *Comp. Phys. Comm.* **62**, 198 (1991).

[14] D.C. Rapaport, Multi-Million Particle Molecular Dynamics II. Design Considerations for Distributed Processing. *Comp. Phys. Comm.* **62**, 217 (1991).

[15] F. F. Abraham, Computational Statistical Mechanics. Methodology, Applications and Supercomputing, *Advances in Physics* **35**, 1 (1986).

[16] W. Smith, Molecular Dynamics on Hypercube Parallel Computers, *Comp. Phys. Comm.* **62**, 229 (1991).

[17] S. Gupta, Computing Aspects of Molecular Dynamics Simulation, *Comp. Phys. Comm.* **70**, 243 (1992).

[18] B. M. Boghosian, Computational Physics on the Connection Machine, in *Computers in Physics*. Jan/Feb 1990.

[19] S. Plimpton and G. Heffelfinger, Scalable Parallel Molecular Dynamics on MIMD Supercomputers, *Procs. of the High Performance Computing Conference 92*, p246, IEEE Computer Society 1992.

[20] S. Plimpton, Fast Parallel Algorithms for Short-Range Molecular Dynamics, Sandia Report SAND91-1144 * UC-705, May 93.

[21] S. Plimpton, Fast Parallel Algorithms for Short-Range Molecular Dynamics, Preprint June 1994. To appear in *J. Comp. Phys.* 1995.

[22] B. J. Alder and T. E. Wainwright, Phase Transitions for a Hard Sphere System, *J. of Chem. Phys.* **27**, 1208 (1957).

[23] W. W. Wood and F. R. Parker, Monte Carlo Equation of State of Molecules Interacting with the Lennard-Jones Potential. I. A Supercritical Isotherm at About Twice the Critical Temperature, *J. Chem. Phys.* **27**, 720 (1957).

[24] A. Rahman, Correlation in the Motion of Atoms in Liquid Argon, *Phys. Rev.* **136**, A405 (1964).

[25] L. Verlet, Computer "Experiments" on Classical Fluids. I Thermodynamical Properties of Lennard-Jones Molecules, *Phys. Rev.* **159**, 98 (1967).

[26] B. J. Alder and T. E. Wainwright, Decay of the Velocity Autocorrelation Function, *Phys. Rev. A* **1**, 18 (1970).

[27] A. Rahman and F. H. Stillinger, Molecular Dynamics Study of Liquid Water, *J. Chem. Phys.* **55**, 3336, (1971)

[28] D. J. Evans and S. Murad. Singularity Free Algorithm for Molecular Dynamics Simulation of Rigid Polyatomics, *Mol. Phys.* **34**, 327 (1977).

[29] G. Ciccotti, M. Ferrario and J. P. Ryckaert. Molecular Dynamics of Rigid Systems in Cartesian Coordinates. A General Formulation, *Mol. Phys.* **47**, 1253 (1982).

[30] G. M. Torrie and J. P. Valleau, Nonphysical Sampling Distributions in Monte Carlo Free-Energy Estimation: Umbrella Sampling, *J. Comp. Phys.* **23**, 187 (1977).

[31] D. Frenkel and A. J. L. Ladd, New Monte Carlo Method to Compute the Free Energy of Arbitrary Solids. Application to the FCC and HCP Phases of Hard Spheres, *J. of Chem. Phys.* **81**, 3188 (1984).

[32] C. H. Bennett, Efficient Estimation of Free Energy Differences from Monte Carlo Data, *J. of Comp. Phys.* **22**, 245 (1976).

[33] H. C. Andersen, Molecular Dynamics Simulations at Constant Pressure and/or Temperature, *J. Chem. Phys.* **72**, 2384 (1980).

[34] M. Parrinello and A. Rahman, Polymorphic Transitions in Single Crystals: A New Molecular Dynamics Method, *J. Appl. Phys.* **52**, 7182 (1981).

[35] S. Nose, A Unified Formulation of the Constant Temperature Molecular Dynamics Methods, *J. Chem. Phys.* **81**, 511 (1984).

[36] W. Hoover, Canonical Dynamics: Equilibrium Phase-Space Distributions, *Phys. Rev. A* **31**, 1695 (1985).

[37] D. Fincham, Programs for the Molecular Dynamics Simulation of Liquids: I. Spherical Molecules with Short-Range Interactions, *Comp. Phys. Comm.* **21**, 247 (1980).

[38] R. Vogelsang, M. Schoen and C. Hoheisel, Vectorization of Molecular Dynamics Fortran Programs Using the Cyber 205 Vector Processing Computer, *Comp. Phys. Comm.* **30**, 235 (1983).

[39] D. Fincham and B. J. Ralston, Molecular Dynamics Simulation Using the Cray-1 Vector Processing Computer, *Comp. Phys. Comm.* **23**, 127 (1981).

[40] M. Schoen, *Comp. Phys. Comm.* **52**, 175 (1989).

[41] Erpenbeck

[42] R. Car and M. Parrinello, Unified Approach for Molecular Dynamics and Density-Functional Theory, *Phys. Rev. Lett.* **55**, 2471 (1985).

[43] F. F. Abraham, W. E. Rudge, D. J. Auerbach and S. W. Koch, Molecular Dynamics Simulations of the Incommensurate Phase of Krypton on Graphite Using More than 100,000 Atoms, *Phys. Rev. Lett.* **52**, 445 (1984).

[44] Livermore

[45] W. Swope and H. C. Andersen, 10^6-Particle Molecular-Dynamics Study of Homogeneous Nucleation of Crystals in a Supercooled Atomic Liquid, *Phys. Rev. B* **41**, 7042 (1990).

[46] D.C. Rapaport, Multi-Million Particle Molecular Dynamics III. Design Considerations for Data-Parallel Processing. *Comp. Phys. Comm.* **76**, 301 (1993).

[47] D.C. Rapaport, Details of Large-Scale Demonstrations of Molecular Dynamics on the CM-5. HLRZ/Julich draft report, August 1993.

[48] D. Brown, J. H. R. Clarke, M. Okuda and T. Yamazaki, A Domain Decomposition Parallelization Strategy for Molecular Dynamics Simulations on Distributed Memory Machines, *Comp. Phys. Comm.* **74**, 67 (1993).

[49] S. L. Lin, J. Mellor-Crumney, B. M. Pettitt, G. N. Phillips Jr., Molecular Dynamics on a Distributed-Memory Multiprocessor, *J. of Comp. Phys.* **13**, 1022 (1992).

[50] W. Smith and T. R. Forester, Parallel Macromolecular Simulations and the Replicated Data Strategy I: the Computation of Atomic Forces, *Comp. Phys. Comm.* **79** 52 (1994).

[51] W. Smith and T. R. Forester, Parallel Macromolecular Simulations and the Replicated Data Strategy II: the RD-SHAKE Algorithm, *Comp. Phys. Comm.* **79** 63 (1994).

[52] W. Form, N. Ito and G. A. Kohring, *Int. J. of Mod. Phys. C* **4**, 1075 (1993).

[53] V. Buchholtz and T. Pöschel, A Vectorized Algorithm for Molecular Dynamics of Short Range Interacting Particles. Preprint cond-mat/9307035.

[54] A. R. C. Raine, D. Fincham and W. Smith, *Comp. Phys. Comm.* **55**, 13 (1989).

[55] K. Esselink, B. Smit and P. A. J. Hilbers, Efficient Parallel Implementation of Molecular Dynamics on a Toroidal Network, Part I: Parallelizing Strategy. *J. Comp. Phys.* **106**, 101 (1993).

[56] F. Hedman and A. Laaksonen, Data Parallel Large-Scale Molecular Dynamics for Liquids, *Int. Jour. of Quantum Chem.* **46**, 27 (1993).

[57] F. Bruge and S. L. Fornili, *Comp. Phys. Comm.* **60**, 31 (1990); *Comp. Phys. Comm.* **60**, 39 (1990).

[58] A. I. Mel'čuk, R. C. Giles and H. Gould, Molecular Dynamics Simulation of Liquids on the Connection Machine, *Computers in Physics*, p311 May/June 1991.

[59] D. M. Beazley and P. S. Lomdahl, Message-Passing Multi-Cell Molecular Dynamics on the Connection Machine CM-5, Los Alamos tech. report LA-UR-92-3158 and *Parallel Computing* **20**, 173 (1994).

[60] P. S. Lomdahl, P. Tamayo, N. Grønbech-Jensen and D. M. Beazley, 50 Gflops Molecular Dynamics on the Connection Machine 5, Los Alamos tech. report LA-UR-93-3078 and *Procs. of the Supercomputing'93* conference, p520, Portland OR, IEEE Computer Society 1993.

[61] P. S. Lomdahl, D. M. Beazley, P. Tamayo and N. Grønbech-Jensen, Multi-Million Particle Molecular Dynamics on the CM-5. *Int. Jour. of Mod. Phys. C* **4**, 1074 (1993).

[62] D. M. Beazley, P. S. Lomdahl, N. Grønbech-Jensen and P. Tamayo, A High Performance Communication and Memory Caching Scheme for Molecular Dynamics on the CM-5, Los Alamos tech. report LA-UR-93-3467 and *Procs. of the 8th International Parallel Processing Symposium 1994*, p800, Cancun Mexico, IEEE Computer Society 1994.

[63] P. Tamayo, J. P. Mesirov and B. Boghosian, Parallel Approaches to Short-Range Molecular Dynamics Simulations, *Procs. of the Supercomputing'91* conference, p462, Albuquerque NM, IEEE Computer Society 1991.

[64] P. Tamayo and R. Giles, A Parallel Scalable Approach to Short-Range Molecular Dynamics on the CM-5,*Procs. of the High Performance Computing Conference 92*, p240, IEEE Computer Society 1992.

[65] D. Stauffer, Computers Accelerate Towards Monte Carlo, *Physics World*, p30 (1993).

[66] H. J. C. Berendsen and W. F. van Gunsteren, Practical algorithms for Dynamics Simulations, *Procs. of the Int. School of Physics Enrico Fermi*, North-Holland 1985.

[67] E. Forest and R. D. Ruth, Fourth-Order Symplectic Integration, *Physica D* **43**, 105 (1990).

[68] J. J. Biesiadecki and R.D. Skeel, Dangers of Multiple-Time-Step Methods, *J. Comput. Phys.* **109**, 318 (1993).

[69] G. Zhang and T. Schlick, LIN: New Algorithm to Simulate the Dynamics of Biomolecules by Combining Implicit-Integration and Normal Mode Techniques, *J. Comp. Chem.* **14**, 1212 (1993).

[70] R. D. Skeel and J. J. Biesiadecki, Symplectic Integration with Variable Stepsize, *Annals of Numer. Math.* **1**, 191 (1994).

[71] W. B. Street, D. J. Tildesley and G. Saville, Multiple Time-Step Methods in Molecular Dynamics, *Molecular Physics* **35**, 639 (1978).

[72] T. R. Forester and W. Smith, On multiple timestep algorithms and the Ewald Sum, to appear in *Mol. Sim* 1994.

[73] J. F. Applegate, M. R. Douglas, Y. Gursel, P. Hunter, C. Seitz and G. J. Sussman, *IEEE Trans. Comput.* **84**, 822 (1985).

[74] G. Contopoulos, N. K. Spyrou and L. Vlahos, eds. *Galactic Dynamics and N-Body Simulations, Springer Lecture Notes in Physics* **433** (1994).

[75] J. Makino and P. Hut, Performance Analysis of Direct N-Body calculations. *Ast. J. Supp. Ser.* **68**, 833 (1988).

[76] M. Taiji, J. Makino, A. Shimuzu, R. Takada, T. Ebisuzaki, and D. Sugimoto, *Proceedings of the International Conference on Physics Computing PC '94*, edited by R. Gruber and M. Tomassini, European Physical Society, Geneva 1994.

[77] H. T. Kung, *Computer* **15**, 37 (1982).

[78] G. M. Megson, *An Introduction to Systolic Algorithm Design*, Clarendon Oxford 1992.

[79] K. M. Nelson, S. T. Smith, and L. T. Wille, in *Procs. Sixth SIAM Conf. on Parallel Processing for Scientific Computing*, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold and D. A. Reed Eds., p174 (SIAM Philadelphia, 1993).

[80] K. M. Nelson, C. F. Cornwell and L. T. Wille, Massively Parallel Computer Simulations of Fullerenes and Si-Clusters, *Computational Materials Science* **2**, 525 (1994).

[81] J.-P. Brunet, J. P. Mesirov and A. Edelman, An Optimal Hypercube Direct N-Body on the Connection Machine, *Procs. of Supercomputing'90*, p748. IEEE Computer Society press, 1990.

[82] J.-P. Brunet, A. Edelman and J. P. Mesirov, Hypercube Algorithms for Direct N-Body Solvers on the Connection Machine, SIAM *Jour. Sci. Dist. Comp.* 1993.

[83] A. Greenberg, J. A. Sethian and J. P. Mesirov, Programming Direct N-Body Solvers on Connection Machines. Thinking Machines Corp. preprint, June 1992.

[84] J. A. Sethian, A Brief Overview of Vortex Methods, in *Vortex Methods and Vortex Motion*, K. Gustafson and J. A. Sethian Eds. SIAM Publications, Philadelphia 1991.

[85] J. A. Sethian, J. P. Brunet, A. Greenberg and J. P. Mesirov, Two Dimensional, Viscous, Incompressible Flow on a Massively Parallel Processor, *J. Comp. Phys.* **101**, 185 (1992).

[86] D. M. Heyes and W. Smith, *Inf. Quat. Comp. Simul. Cond. Phases (Daresbury Lab.)* **28**, 63 (1988).

[87] J. W. Perram, H. G. Petersen and S. W. de Leeuw, An Algorithm for the Simulation of Condensed Matter which Grows as the 3/2 Power of the Number of Particles, *Mol. Phys.* **65**, 875 (1988).

[88] W. Smith, A Replicated Data Molecular Dyanmics Strategy for the Parallel Ewald Sum, Daresbury Lab. preprint DL/SCI/P764T June 1991.

[89] K. Esselink, P. A. J. Hilbers and K. Hinsen, Experience with Ewald Sums. AMER 92.014, Koninklijke/Shell-Laboratorium, Amsterdam, Sept. 1992.

[90] D. Fincham, *Inf. Quat. Comp. Simul. Cond. Phases (Daresbury Lab.)* **38**, 17 (1993).

[91] M. J. Stevens and K. Kremer, preprint Nov. 2, 1994.

[92] J. D. Jackson, *Classical Electrodynamics 2ed.*, John Wiley and Sons, 1975.

[93] A. W. Appel, An Efficient Program for Many-Body Simulations, *SIAM J. Sci. Stat. Comp.* **6**, 85 (1985).

[94] J. Barnes and P. Hut, A Hierachical $O(N \log N)$ Force-Calculation Algorithm, *Nature* **324**, 446 (1986).

[95] L. Greengard and V. Rokhlin, A Fast Algorithm for Particle Simulations, *Jour. of Comp. Phys* 73, 325 (1987); L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*, MIT Press, 1988; J. Carrier, L. Greengard and V. Rokhlin, A Fast Adaptive Multipole Algorithm for Particle Simulations, *SIAM J. Sci. Stat. Comput.* **9**, 669 (1988).

[96] F. Zhao, An $O(n)$ Algorithm for three-dimensional $N-$Body Simulations. Tech. Rep. AI MIT Memo 995, MIT Artificial Intelligence Lab. 1987.

[97] C. R. Anderson, An Implementation of the Fast Multipole Method without Multipoles, *SIAM J. Sci. Stat. Comp.* **13**, 923 (1992).

[98] F. Zhao and L. Johnsson, The Parallel Multipole Method on the Connection Machine, YALEU/DCS/TR-749, October 1989.

[99] L. S. Nyland, J. F. Prins and J. H. Reif, A Data-Parallel Implementation of the Adaptive Fast Multipole Algorithm, *Procs. of the 1993 DAGS/PC Symp.*, June 1993.

[100] J. H. Reif and S. R. Tate, The Complexity of N-Body Simulations, in *Procs. 20th International Colloquium on Automata, Languages, and Programming (ICALP'93)*, p162, Springer-Verlag (1993).

[101] Y. Hu and S. L. Johnsson, A Data Parallel Implementation of Hierarchical $N-$Body Methods, TR-26-94 Parallel Computing Research Group, Harvard University 1994.

[102] C. Draghicescu, An efficient implementation of particle methods for the incompressible Euler equations, *SIAM J. Numer. Analysis* **31**(4), 1994.

[103] M. Draghicescu, to appear in *J. Comp. Phys.*, 1994.

[104] P. J. Steinback and B. R. Brooks, New Spherical-Cutoff Methods for Long-Range Forces in Macromolecular Simulation. *J. of Comp. Chem.* **15**, 667 (1994).

[105] K. Esselink, A Comparison of Algorithms for Long Range Interactions, Shell preprint August 1994.

[106] K. Esselink, LJPaVe a Molecular Dynamics Programs for Sequential, Vector and Parallel Computers. Shell Research Amsterdam technical report AMER.94.004.

[107] G. S. Grest, B. Dunweg and K. Kremer, Vectorized Link Cell Fortran Code for Molecular Dynamics Simulations for a Large Number of Particles, *Comp. Phys. Comm.* **55**, 269 (1989).

[108] J. J. Morales and M. J. Nuevo, Comparison of Link-Cell and Neighbourhood Tables on a Range of Computers, *Comp. Phys. Comm.* **69**, 223 (1992).

[109] D. Lynch and P. Tamayo, unpublished.

[110] The global-local programming model extends the data parallel model by allowing the use of local (message-passing) subroutines. See for example the *CM Fortran Programming Guide*, version 2.2 October 1994, Thinking Machines Corp. or the High Performance Fortran Forum Home Page: http://www.erc.msstate.edu/hpff/home.html

[111] T. W. Clark, R. v. Hanxleden, J. A. McCammon and R. Scott, Parallelizing Molecular Dynamics using Spatial Decomposition, *Procs. of the Scalable High Performance Computing Conference 94*, Knnoxville TN, May 1994.

[112] T. von Eicken, D. E. Culler, S. C. Glodstein and K. E. Schauser, Active Messages: A Mechanism for Integrated Communications and Computation, in *Procs. of the 19th International Symposium on Computer Architecture*. Australia, ACM press, May 1992; L. W. Tucker and A. Mainwaring, CMMD: Active Messages on the CM-5, *Parallel Computing* **20**, 481, 1994.

[113] H. Heller, H. Grubmüller and K. Schulten, *Mol. Simul.* **5**, 133 (1990).

[114] B. Banko and H. Heller, *User Manual for EGO*, Beckman Inst. Tech. Rep. UIUC-BI-TB-92-07 (1991).

[115] H. Grubmüller, H. Heller, A. Windemuth and K. Schulten, *Mol. Simul.* **6**, 121 (1991).

[116] A. B. Sinha, K. Schulten and H. Heller, Performance Analysis of a Parallel Molecular Dynamics Program, *Comp. Phys. Comm.* **78**, 265 (1994).

[117] R. K. Kalia, S. de Leeuw, A. Nakano and P. Vashishta, Molecular Dynamics Simulations of Coulombic Systems in Distributed-Memory MIMD machines, *Comp. Phys. Comm.* **74**, 316 (1993).

[118] W. Scott, A. Gunzinger, B. Bäumle, P. Kohler, U. A. Müller, H-R. Vonder Mühll, A. Eichenberger, W. Guggenbühl, N. Ironmonger, F. Müller-Plathe and W. F. van Gunsteren, Parallel Molecular Dynamics on a Multi Signalprocessor System, *Comp. Phys. Comm.* **75**, 65 (1993).

[119] R. Giles and P. Tamayo, Thinking Machines Tech. Report (unpublished).

[120] P. Ossadnik and M. Gyure, unpublished.

[121] B. L. Holian, O. E. Percus, T. T. Warnock and P. A. Whitlock, Pseudorandom Number Generator for Massively Parallel Molecular-Dynamics Simulations, *Phys. Rev. E* **50**, 1607 (1994).

[122] B. R. Brooks and M. Hodošček, Parallelization of CHARMM for MIMD machines, *Chemical Design Automation News* **7**, 16 (1992).

[123] F. Müller-Plathe and D. Brown, Multi-Colour Algorithms in Molecular Simulation: Vectorisation and Parallelization of Internal Forces and Constraints, *Comp. Phys. Comm.* **64**, 7 (1991).

[124] H. Schreiber, O. Steinhauser and P. Schuster, Parallel Molecular Dynamics of Biomolecules, *Parallel Computing* **18**, 557 (1992).

[125] T. W Clark, J. A. McCammon and L. R. Scott, Parallel Molecular Dynamics, *Proc. of the 5th SIAM Conference on Parallel Processing for Scientific Computing* 338 (1992).

[126] P. Lomdahl and D. Beazley, unpublished.

[127] M. Flanigan and P. Tamayo, A Parallel Cluster Labeling Method for Monte Carlo Dynamics, *Int. Jour. of Mod. Phys. C* **3**, 1235 (1992).

[128] V. S. Batista and D. F. Cooker, Parallel Scalable Message-Passing Multi-Cell Molecular Dynamics on a Coupled Cluster of Workstations, BU preprint 1994.

[129] A. Zeyher, Simulated Materials Reveal Microfractures, *Computers in Physics*, p382 Jul/Aug 1993.

[130] R. M. Lynden-Bell, Computer Simulations of Fracture at the Atomic Level, *Science* **263**, 1704 (March 1994).

[131] P. Lomdahl, unpublished (T-11 current project simulations of cracks)

[132] B. L. Holian, Modeling Shock-Wave Deformation Via Molecular Dynamics, *Phys. Rev. A* **37**, 2562 (1988).

[133] B. L. Holian and D. E. Grady, Fragmentation by Molecular Dynamics: The Microscopic "Big Bang", *Phys. Rev. Lett.* **60**, 1355 (1988).

[134] N. J. Wagner, B. L. Holian and A. F. Voter, Molecular-Dynamics Simulations of Two-Dimensional Materials at High Strain Rates, *Phys. Rev. A* **45**, 8457 (1992).

[135] L. Phillips, R. S. Sinkovits, E. S. Oran and J. P. Boris, The Interaction of Shocks and Defects in Lennard-Jones Crystals, *J. of Phys. Cond. Matter* **5**, 6357 (1993).

[136] W. G. Hoover, C. G. Hoover, I. F. Stowers, A. J. DeGroot, and B. Moran, Simulation of Mechanical Deformation Via Nonequilibrium Molecular Dynamics, UCRL-101903 preprint, Aug. 1989.

[137] J. Belak and I. F. Stowers, Molecular Dynamics Studies of Surface Indentation in Two Dimensions, UCRL-JC-103866 preprint, Apr. 1990.

[138] S. G. Kim and David Tománek, Melting the Fullerenes: A Molecular Dynamics Study, *Phys. Rev. Lett.* **72**, 2418 (1994).

[139] W. Zhong, Y. Cai and D. Tománek, Computer Simulation of Hydrogen Embrittlement in Metals, *Nature* **362**, 435 (1993).

[140] R. D. Mountain and A. C. Brown, Molecular Dynamics Investigation of Homogeneous Nucleation for Inverse Power Potential Liquids and for Modified Lennard-Jones Liquid, *J. Chem. Phys.* **80**, 2730 (1984).

[141] J.-X. Yang, H. Gould and W. Klein, Molecular Dynamics Investigation of Deeply Quenched Liquids, *Phys. Rev. Lett.* **60**, 2665 (1988).

[142] J. Yang, H. Gould, W. Klein and R. D. Mountain, Molecular Dynamics Investigation of Deeply Quenched Liquids, *J. Chem. Phys.* **93**, 711 (1990).

[143] E. Velasco and S. Toxvaerd, Computer Simulations of Phase Separation in a Two-Dimensional Fluid Mixture, *Phys. Rev. Lett.* **71**, 388 (1993).

[144] O. H. Nielsen, J. P. Sethna, P. Stoltze, K. W. Jacobsen and J. K. Nørskov, Melting a Copper Cluster: Critical Droplet Theory, UNI-C preprint, Technical U. of Denmark, 1992.

[145] C. Uddink andd D. Frenkel, Orientational Order and Solid-Liquid Coexistence in the Two-Dimensional Lennard-Jones System, *Phys. Rev. B* **35**, 6933 (1987).

[146] J. D. Honeycutt and H. C. Andersen, Small System Size Artifacts in the Molecular Dynamics Simulation of Homogeneous Crystal Nucleation in Supercooled Atomic Liquids, *J. Phys. Chem.* **90**, 1585 (1986).

[147] G. Benettin, G. Lo Vecchio and A. Tenenbaum, Stochastic Transition in Two-Dimensional Lennard-Jones Systems, *Phys. Rev. A* **22**, 1709 (1980).

[148] G. Benettin and A. Tenenbaum, Ordered and Stochastic Behavior in a Two-Dimensional Lennard-Jones System, *Phys. Rev. A* **28**, 3020 (1983).

[149] D. Thirumalai and R. D. Mountain, Ergodic Convergence Properties of Supercooled Liquids and Glasses, *Phys. Rev. A* **42**, 4574 (1990).

[150] G. Krautsch, C. Dai and E. J. Kramer, Real Space Observation of Dynamic Scaling in a Critical Polymer Mixture, *Phys. Rev. Lett.* **71**, 3669 (1993).

[151] G. S. Grest and K. Kremer, Dynamics of Dense Polymers: A Molecular Dynamics Approach, *Springer Proc. in Physics* Vol 33: Computer Simulation Studies in Condensed Matter Physics, p76 (1988).

[152] K. Kremer, G. S. Grest and B. Dünweg, Computer Simulations for Polymer Dynamics, *Springer Proc. in Physics* Vol 53: Computer Simulation Studies in Condensed Matter Physics III, p85 (1991).

[153] D. C. Rapaport and E. Clementi, Eddy Formation in Obstructed Fluid Flows: A Molecular Dynamics Study, *Phys. Rev. Lett.* **57**, 695 (1986).

[154] D. C. Rapaport, Molecular Dynamics: A New Approach to Hydrodynamics ?, *Springer Proc. in Physics* Vol 33: Computer Simulation Studies in Condensed Matter Physics, p98 (1988).

[155] I. Zuñiga and P. Español, Scaling of the Time-Dependent Diffusion Coefficient by Molecular Dynamics Simulation, *Phys. Rev. Lett.* **71**, 3665 (1993).

[156] E. Enciso, N. G. Almarza, V. del Prado, F. J. Bermejo, E. López Zapata and M. Ujaldón, Molecular-Dynamics Simulation on Simple Fluids: Departure from Linearized Hydrodynamic Behavior of the Dynamical Structure Factor, *Phys. Rev. E* **50**, 1336 (1994).

[157] P. A. Thomson and M. O. Robbins, Simulations of Contact-Line Motion: Slip and the Dynamic Contact Angle, *Phys. Rev. Lett.* **63**, 766 (1989).

[158] P. A. Thomson and M. O. Robbins, To Slip or Not to Slip, *Physics World*, p35 (Nov. 1990).

[159] P. A. Thomson and G. S. Grest, Granular Flow: Friction and the Dilatancy Transition, *Phys. Rev. Lett.* **67**, 1751 (1991).

[160] G. H. Ristow, Molecular Dynamics Simulations of Granular Materials on the Intel IPSC/860, *Int. J. of Mod. Phys. C* (1993).

[161] K. Esselink, P. A. J. Hilbers, N. M. van Os, B. Smit and S. Karaborni, Molecular Dynamics Simulations of Model Oil/Water/Surfactant Systems, Shell preprint, to appear in *Colloids and Surfaces*, October 1994.

[162] K. Esselink, P. A. J. Hilbers, S. Karaborni, J. I. Siepmann and B. Smit, Simulating Complex Fluids, Shell preprint, to appear in *Molecular Simulation*, 1994.