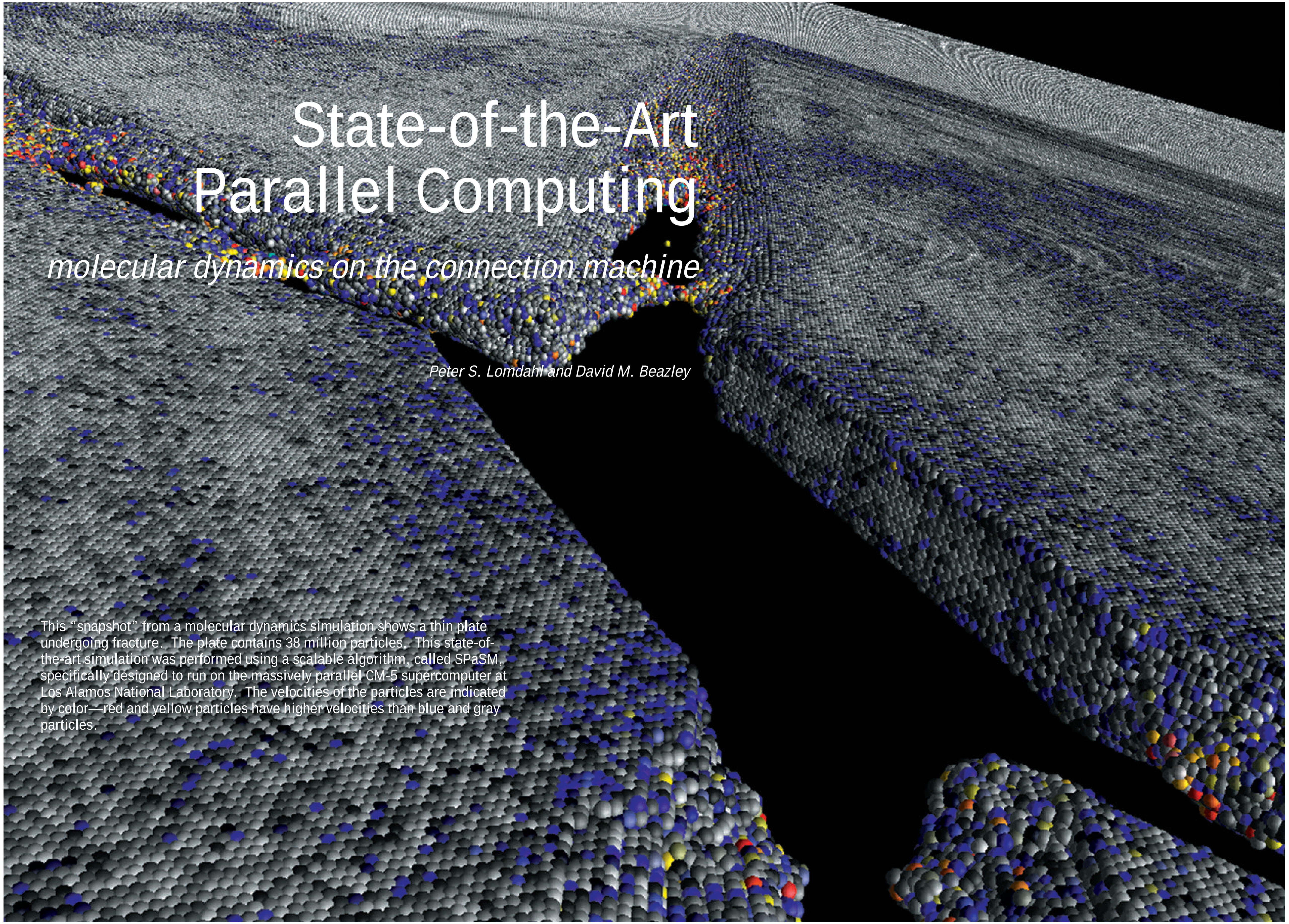


State-of-the-Art Parallel Computing

molecular dynamics on the connection machine

Peter S. Lomdahl and David M. Beazley

This "snapshot" from a molecular dynamics simulation shows a thin plate undergoing fracture. The plate contains 38 million particles. This state-of-the-art simulation was performed using a scalable algorithm, called SPaSM, specifically designed to run on the massively parallel CM-5 supercomputer at Los Alamos National Laboratory. The velocities of the particles are indicated by color—red and yellow particles have higher velocities than blue and gray particles.

The availability of massively parallel computers presents computational scientists with an exciting challenge. The architects of these machines claim that they are scalable—that a machine containing 100 processors arranged in a parallel architecture can be made 10 times more powerful by employing the same basic design but using 1000 processors instead of 100. Such massively parallel computers are available, and one of largest—a CM-5 Connection Machine containing 1024 processors—is here at Los Alamos National Laboratory. The challenge is to realize the promise of those machines—to demonstrate that large scientific problems can be computed with the advertised speed and cost-effectiveness.

The challenge is significant because the architecture of parallel computers differs dramatically from that of conventional vector supercomputers, which have been the standard tool for scientific computing for the last two decades.

The massively parallel architecture requires an entirely new approach to programming. A computation must somehow be divided equally among the hundreds of individual processors and communication between processors must be rapid and kept to a minimum. The incentive to work out these issues was heightened by Gordon Bell, an independent consultant to the computer industry and well-known skeptic of massively parallel machines. Six years ago he initiated a formal competition for achievements in large-scale scientific computation on vector and parallel machines. The annual prizes are awarded through and recognized by the Computer Society of the Institute of Electrical and Electronic Engineers (IEEE). The work reported here was awarded one of the 1993 IEEE Gordon Bell prizes. We were able to demonstrate very high performance as measured by speed: Our simulation, called SPaSM (scalable parallel short-range molecular dynamics), ran on the massively parallel CM-5 at a rate of 50 gigaflops (50 billion floating-point operations per second). That rate is nearly 40% of the theoretical maximum; typical programs achieve only about 20% of the theoretical maximum. Also, the algorithm we developed has the property of scalability. When the number of processors used was doubled, the time required to run the simulation was cut in half.

Our achievement was the adaptation of the molecular-dynamics method to the architecture of parallel machines. Molecular dynamics (MD) is a first-principles approach to the study of materials. MD starts with the fundamental building blocks—the individual atoms—that make up a material and follows the motions of the atoms as they interact with each other through interatomic forces. Even the smallest piece of material that is useful for experimental measurement contains over ten billion atoms. Therefore, the larger the number of atoms included in the calculation, the more interesting the results for materials science. Scientists would very much like to perform simulations that predict the motions of billions of atoms during a dynamical process such as machining or fracture and compare the predictions with experiment. Unfortunately, the realization of that prospect is not possible today. However, if the machines can be scaled up in both memory capacity and speed, as the computer manufacturers have promised, then simulations can be scaled up proportionally so that billion-atom simulations may become a reality. Furthermore, as we show here, the current generation of massively parallel machines has made it possible to simulate the motions of tens of millions of atoms. Such simulations are large enough to perform meaningful studies of the bulk properties of matter and to improve the approximate models of those properties used in standard continuum descriptions of macroscopic systems. Molecular-dynamics simulations on massively parallel computers thus represent a new opportunity to study the dynamical properties of materials from first principles.

Massively parallel machines have only recently become popular, so the vendors have not yet developed the software—the compilers—needed to optimize a given program for the particular architecture of their machines. Consequently, to develop a scalable parallel algorithm for molecular dynamics that would yield high performance, we needed to understand the inner workings of the CM-5 and incorporate that knowledge into the details of our program. Much of our work focused on implementing a particular technique for interprocessor communication known as "message passing." In this approach programmers use explicit instructions to initiate communication among the processors of a parallel system, and the processors accomplish the communication by sending messages to each other.

Message-passing programming is advantageous not only in achieving high performance, but also in increasing "portability," because it is easily adapted for use on a number of massively parallel machines. We were able to run our message-passing algorithm on another parallel computer, the Cray T3D, with minimal modification.

Following brief descriptions of MD and the architecture and properties of the CM-5, we present the method by which we have mapped a molecular-dynamics simulation to the architecture of the CM-5. Our example illustrates message passing and other techniques that are widely applicable to effective programming of massively parallel computers.

Molecular Dynamics

Molecular dynamics is a computational method used to track the positions and velocities of individual particles (atoms or groups of atoms) in a material as the particles interact with each other and respond to external influences. The motion of each particle is calculated by Newton's equation of motion, force = mass × acceleration, where the force on a given particle depends on the interactions of the particle with other particles. The mutual interaction of many particles simultaneously is called a "many-body problem" and has no analytical, or exact, solution because the force on a particle is always changing in a complex way as each particle in the system moves under the influence of the others.

The MD algorithm provides an ap-

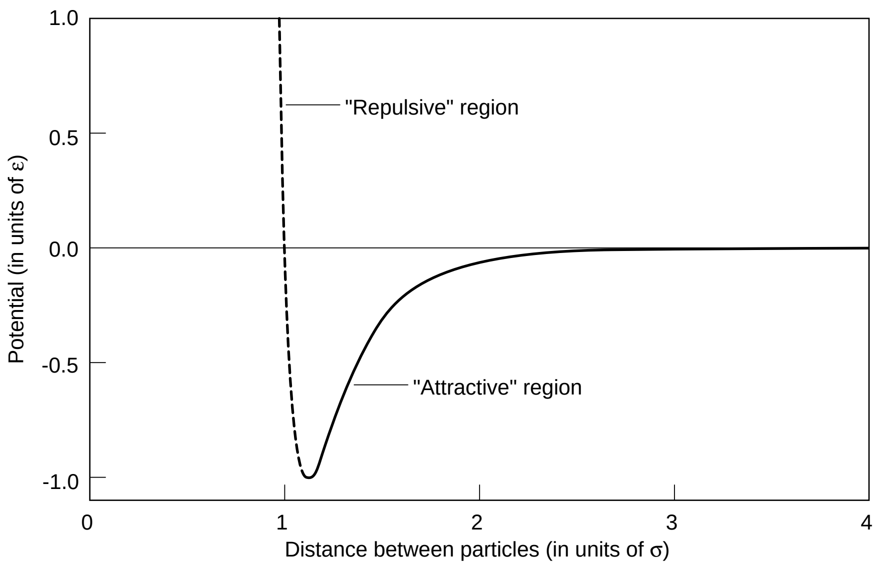


Figure 1. The Lennard-Jones Interaction Potential

The Lennard-Jones interaction potential V between two particles is plotted as a function of the distance r between the particles. The potential is in units of ϵ, and the distance between the particles is in units of σ. The parameter σ is equal to the value of r at which the potential function crosses zero. The potential has a positive slope at longer distances, corresponding to an attractive force, and a steeply negative slope at short distances, corresponding to a strong repulsive force. Separating the two regions is the local minimum of the potential—the deepest point in the potential "well"—which has a depth equal to ϵ and is located at $r = 2^{1/6}\sigma$. At the potential minimum the force between the particles is zero. Therefore, in the absence of effects from any other particles, the separation between two particles will tend to oscillate around $r = 2^{1/6}\sigma$.

proximate solution to the many-body problem by treating time as a succession of discrete timesteps and treating the force on (and therefore the acceleration of) each particle as a constant for the duration of a single timestep. At each timestep the position and velocity of each particle is updated on basis of the constant acceleration it experiences during that timestep. For example, at time t, at the beginning of a timestep of length Δt, particle i has position $\mathbf{r}_i$ and velocity $\mathbf{v}_i$. The force on particle i from the other particles is calculated; the force determines the acceleration $\mathbf{a}_i$ of particle i for the duration of this timestep. From elementary physics, the position $\mathbf{r}_i'$ and velocity $\mathbf{v}_i'$ at the later time $t+\Delta t$ are given by:

$$\mathbf{r}_i' = \mathbf{r}_i + \mathbf{v}_i\Delta t + \tfrac{1}{2}\mathbf{a}_i(\Delta t)^2$$

and

$$\mathbf{v}_i' = \mathbf{v}_i + \mathbf{a}_i\Delta t.$$

Sometimes, these equations are replaced by more sophisticated approximations for the new positions and velocities; the approximations involve interpolations from positions and velocities at time t and the earlier $t-\Delta t$. Such schemes conserve energy and momentum more effectively than the simple update of position and velocity given above.

Whatever the scheme, the equations yield a new position and velocity for each particle at the new time $t+\Delta t$. The procedure is repeated for each succeeding timestep, and the result is a series of "snapshots" of the positions and velocities of the particles at times $t=0$, Δt, $2\Delta t$, … , $n\Delta t$.

In MD simulations particles are usually treated as point particles (that is, they have no extent), and the force between any two particles is usually approximated as the gradient of a potential function V that depends only on the distance r between the two particles. The force on particle i from particle j can be written

$$F_{ij} = -\frac{dV}{dr}\bigg|_{r_{ij}}, \text{ where } r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|,$$

and that force is directed along the line connecting the two particles. If the force is negative, particles i and j attract one another; if the force is positive, the particles repel each other. The net force exerted on particle i by all other particles j is approximated by summing the forces calculated from all pair-wise interactions between particle i and each particle j.

The standard pair-wise potential used in MD simulations is the Lennard-Jones potential, which is known empirically to provide a reasonably good "fit" to experimental data for atom-atom interactions in many solids and liquids. As shown in Figure 1, this potential has both an attractive part and a repulsive part:

$$V(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right],$$

where r is the distance between the two particles and σ and ϵ are adjustable parameters. When the distance between two particles is at the minimum of the potential function, the force between the two particles is zero and the particles are in their equilibrium positions. In the absence of other particles nearby, the separation of two particles will tend to oscillate around this distance. Physically, the average nearest-neighbor distance of atoms in many solids and fluids corresponds very closely to the position of the minimum in the atom-atom potential.

For our molecular-dynamics simulations, we simplify the force calculation greatly by using a truncated form of the Lennard-Jones potential. That is, we "cut off" the long, flat tail of the potential and assume that the force between particles is exactly zero for particle separations greater than some chosen $r_{\max}$, the interaction cut-off distance. In other words, when calculating the forces for particle i, we need consider only those particles j that are within a distance $r_{\max}$ of i. The time required to calculate the pair interactions with all the surrounding particles is reduced considerably since a given particle "feels" forces from perhaps only a hundred neighbors instead of the millions or billions of particles included in the simulation. The truncated potential provides an excellent approximation for solid materials because in solids the effects of distant atoms are "screened" by nearer atoms. The fact that we can use a short-range potential is the most significant difference between our problem and that described in "A Fast Tree Code for Many-Body Problems," in which the force is long-range and therefore each particle always interacts with all other particles in the system.

Mapping Molecular Dynamics onto the Architecture of the CM-5

The molecular-dynamics method has been implemented on electronic computers since 1957. The first such calculations were performed on the UNIVAC, the first commercially available computer. Initially only 32 particles were used, and only about 300 interactions per hour could be calculated. The rapid advance of computers has made it possible for us—using the massively parallel CM-5—to include tens of millions of atoms in our simulation of materials behavior. Since materials contain large numbers of repeating units (units that can be treated in parallel) calculations of materials behavior are ideally suited for implementation on massively parallel architectures.

Figure 2 illustrates the architecture of the CM-5, which was built by Thinking Machines Corporation. The computer was constructed from large numbers of relatively inexpensive, high-volume microprocessor and memory components. Such a scheme allows the hardware designers to build a high-performance supercomputer without a major effort in the development of new microprocessor and memory circuits. The Laboratory's CM-5 has 1024 processing nodes but partitions as small as 32 nodes may be used.

The processing nodes in the CM-5 are connected by two types of high-speed communication networks: a control network and a data network. The control network performs synchronization functions, broadcasts, and reductions. The synchronization functions are used to coordinate the work of the individual processors. Broadcasts are messages sent to all processors simultaneously, such as a set of instructions to be executed by each processor. Reductions are operations in which data are collected from all processors for the calculation of some overall value such as the total energy of a system.

The data network carries simultaneous point-to-point communication between multiple processing nodes at a rate of over 5 megabytes per second per node. Both networks have a "fat-tree" topology, a scalable structure in which the total bandwidth (capacity for data transmission) of the network increases in proportion to the number of processors. When new processing nodes are added, the networks are expanded so that the effective bandwidth between any two processors does not decrease. Maintaining a high bandwidth is critical to the scalability of the CM-5. A machine having more processing nodes will have a greater total amount of network "traffic." Therefore the larger computer must also have a greater network capacity in order to realize the expected gain in performance.

Our MD algorithm uses the data network extensively—large amounts of particle data are transmitted between processors. The control network is also very important to our algorithm. A single program is broadcast over the control network to the processors, each of which executes that program for the appropriate subset of the particles in the simulation. The processing nodes are allowed to operate independently during most of the calculations. However, there are steps in our algorithm that require synchronization of the processors. For example, the calculation of new particle positions and velocities cannot proceed until the total force acting on each particle has been determined. A special synchronization function ensures that the latter calculation is complete before the former is initiated.

Access to the communication networks is provided by a message-passing library supplied by Thinking Machines. The library allows two types of communication—synchronous and asynchronous. We use both communication styles in our MD algorithm.

All programs on the CM-5 use some form of message passing between processing nodes. Data-parallel languages such as C* and FORTRAN-90 perform

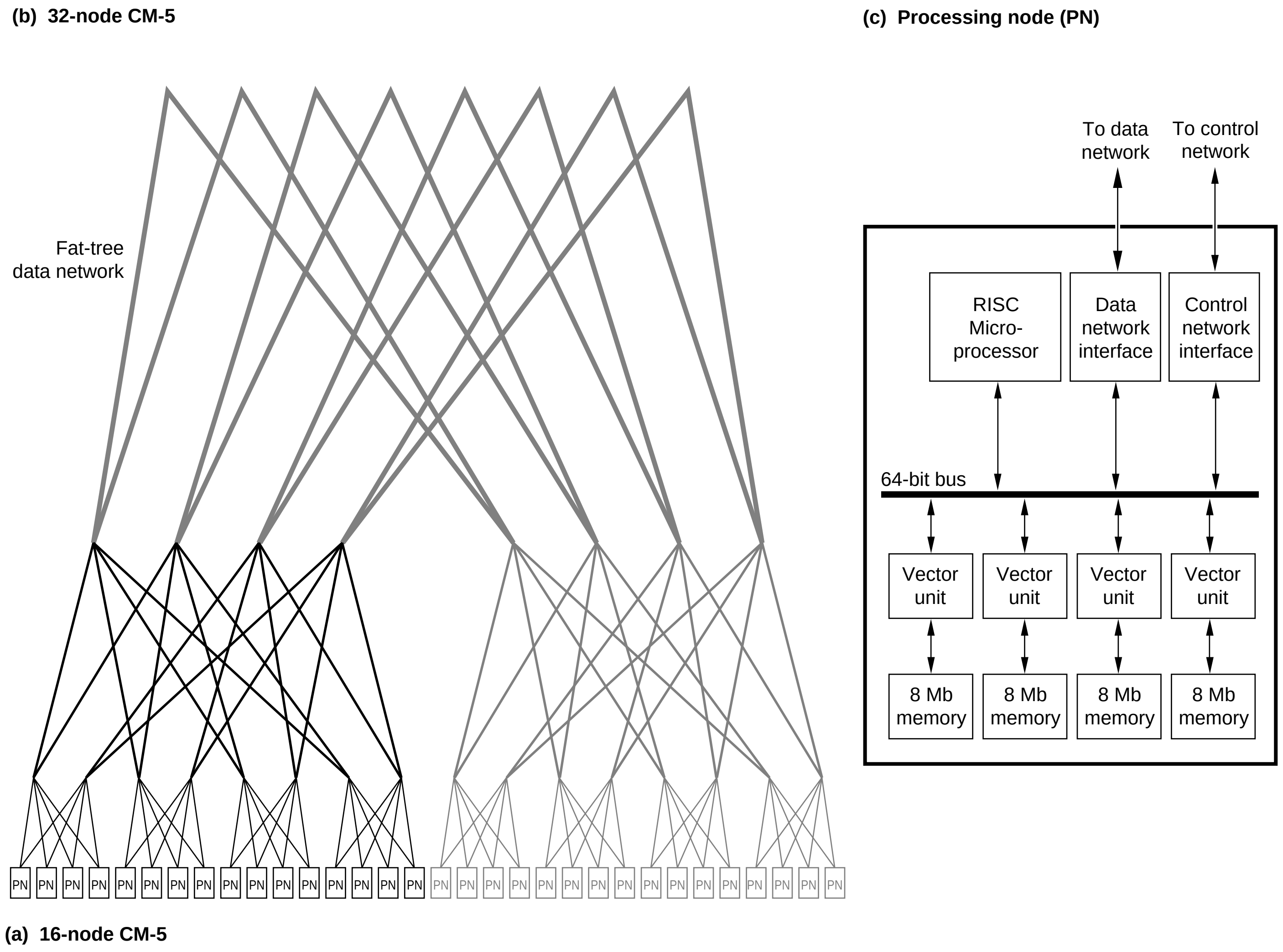


Figure 2. The Architecture of the CM-5 Connection Machine

The diagram shows the general layout of the CM-5, one of the largest massively parallel computers. Also shown in the diagram is the manner in which this parallel architecture can be scaled up in size. (a) Sixteen processing nodes are connected in parallel by a network that carries data between the processors. A control network (not shown), similar in structure to the data network, carries instructions to the nodes. The networks have a "fat tree" topology—the upper branches (which connect a large number of nodes and therefore must carry more data) have a higher bandwidth (capacity for data transmission) so that the overall point-to-point bandwidth is at least 5 megabytes per second for any pair of nodes. (b) The addition of more processing nodes to the system (shown in gray) requires that the networks be expanded to a "higher" level in order to maintain the point-to-point bandwidth. (c) Each processing node consists of a 33-megahertz SPARC RISC (Reduced Instruction Set Computer) microprocessor, 32 megabytes of memory, and four vector-processing units that perform 64-bit floating-point and integer arithmetic at a maximum combined rate of 128 million floating-point operations per second. The 1024-node CM-5 at the Laboratory also has 400 gigabytes (400 billion bytes) of disk space distributed over a number of disk drives (not shown), which are connected in parallel to the networks by input/output processors. The parallel arrangement of the disk drives allows data to be transferred rapidly to disk; however, the data of a single write operation will be spread over all disks. For more details of computer elements and architectures see "How Computers Work: An Introduction to Serial, Vector, and Parallel Computers."

message passing transparently; that is, the underlying communication is hidden from the user. However, to achieve higher performance we wrote explicit message-passing instructions into our program. It was designed to be executed on all of the processing nodes simultaneously. Once the program is broadcast to the processing nodes, each node works independently on a small part of the problem and manages its own data and interprocessor communications.

Data structures of the MD algorithm. The particles in a molecular-dynamics simulation occupy a simulated region of space called a "computational box," shown in Figure 3a (for simplicity, the algorithm is illustrated in two dimensions). The box is divided into domains of equal size, one for each processor (Figure 3b). The assignment of particles to processors is made according to the domain in which each particle is located. The data associated with each particle include its velocity and its coordinates within the computational box.

Because any simulation is finite in size we often need a method to effectively eliminate the boundaries of the simulation. Therefore, we apply periodic boundary conditions to the computational box. That is, the entire computational box with all of its particles is treated as just one unit of an infinite lattice of identical units. Thus a particle near a boundary of the computational box will be treated as if it is interacting with particles in an adjacent box containing an identical number of particles with identical positions and velocities. One result of such boundary conditions is that when a particle moves out through one side of the computational box, an identical particle with the same velocity enters through the boundary at the opposite side of the box. Therefore, both linear momentum and the number of particles in the computational box are conserved.

Typically the volume of the computational box is large enough that the processor domains have dimensions significantly larger than the interaction cut-off distance, r_{max}. In such cases each processor will have been assigned a significant number of particles that do not interact with each other. Computing the forces between all pairs of particles within each domain is unnecessary because the result for each pair of particles separated by more than r_{max} would be zero. Therefore, the domain of each processor is subdivided into an identical number of cells, each having dimensions slightly larger than r_{max} (see Figure 3c). The number of such cells depends only on the size of the domain and r_{max}, not on the number of available processors. The domains may be subdivided into thousands of cells for simulations with large spatial dimensions. In the example in Figure 3, there are 4 processor domains and each domain has 16 cells. The cell structure forms the foundation of our algorithm.

Each particle is represented in the computer by a C-programming-language data structure consisting of the particle parameters, including position, velocity, force, and type. Associated with each cell is a small block of memory containing a sequential list of the particle data for that cell. Storing particle data in this manner facilitates the communication steps of the force calculation. The entire contents of a cell can be communicated to other processors simply by sending a small block of memory.

MD force calculation. The calculation of forces acting on the particles is the most time-consuming part of the MD method. To optimize the calculation, particles that are neighbors physically should also be near one another in the memory of the computer. However, such an arrangement is difficult to maintain because each particle is free to move anywhere in space—particles that are initially neighbors may separate during the course of an MD simulation, and particles that are initially far apart may become neighbors. The cell structure just described was designed to keep the particles organized so that the force calculation proceeds efficiently.

The force on a given particle includes contributions from all the other particles that are closer than r_{max}. Because the cell size has been chosen to be slightly larger than r_{max}, all the particles that must be considered are located within the cell containing the given particle or within adjacent cells.

Each processor follows an identical, predetermined sequence to calculate the forces on the particles within its assigned domain. For each cell in its domain, the processor computes forces exerted on particles in the cell by other particles in the same cell as well as by particles located in adjacent cells. The forces due to particles in adjacent cells are calculated in the order given by the interaction path shown in Figure 4a. Use of an identical path for each cell ensures that all contributions to the total force on every particle are computed without performing redundant calculations.

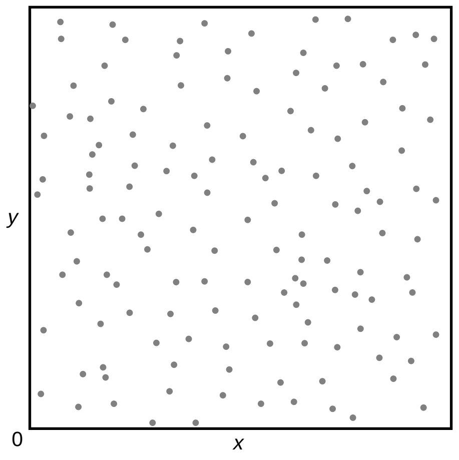
Typical calculations have hundreds or thousands of cells per processor domain, so for most of the cells of a given domain all neighboring cells are also assigned to the same domain. Therefore, the interaction calculation can be performed for most cells without any communication between processors. However, for cells along the edge of each processor domain, some of the adjacent cells are assigned to other processors (as in Figure 4b), and the message-passing features of the CM-5 must be utilized so that particle data for those adjacent cells can be sent to and received from the neighboring processors. Because each processor has been assigned an identical cell structure and follows the same sequence of operations through that structure, all processors will be required to transmit particle data for the same internal cell at approximately the same time. At such times the processors synchronize and participate in send-and-receive communication. Processors assigned to domains with fewer particles will proceed through the interaction calculations faster. Therefore, when working on cells at the edges of the domain, those processors may be required to wait for others before they can synchronize. The details of synchronized message passing are shown in Figure 4c.

At every message-passing step, each processor sends the particle data for an entire cell to the appropriate processors and receives corresponding information from other processors. The force calculation then proceeds, now that each processor has available in its local memory the particle data needed to implement the next step along the interaction path. Note that synchronization occurs only during message passing. At all other times, the processors are running asynchronously.

Redistribution of particles. When the force calculation is complete the processors compute the new positions and velocities of all particles in the system. Since our algorithm is based on the cell structure of the processor domains, the possibility of changes in particle positions requires that the computer data structures for each cell are updated regularly. A special redistribution function checks the coordinates of each particle after every timestep. If the new coordinates of a particle indicate that it has moved to a new cell, the particle parameters must be transferred

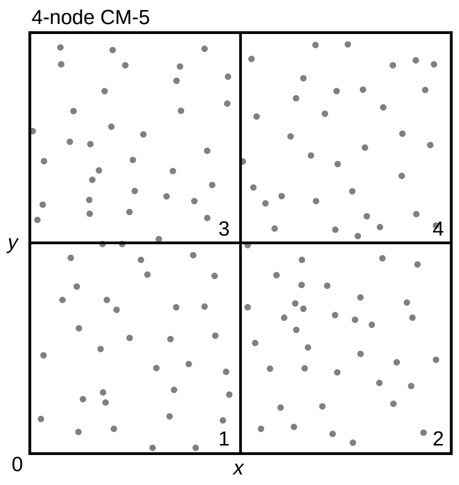
Figure 3. The Data Structure of the MD Algorithm

This figure illustrates the mapping of a molecular-dynamics problem onto a parallel processor with 4 nodes. For simplicity, the algorithm is illustrated in two dimensions.



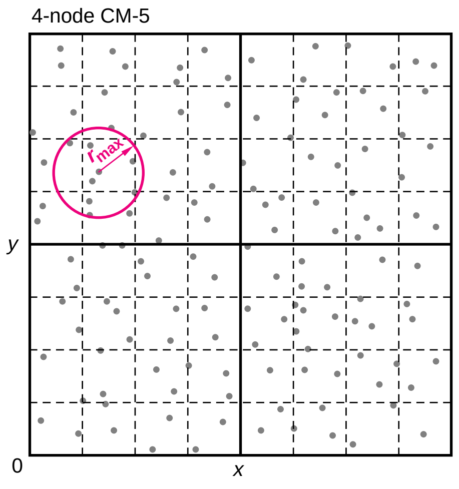
(a) The computational box

A "computational box" contains all the particles in the simulation. A single coordinate system is used for the entire computational box so that each particle has a unique set of coordinates.



(b) The processor domains

The computational box is divided into 4 equal domains, each of which is assigned to a separate processor as denoted by the numbers 1, 2, 3, and 4.

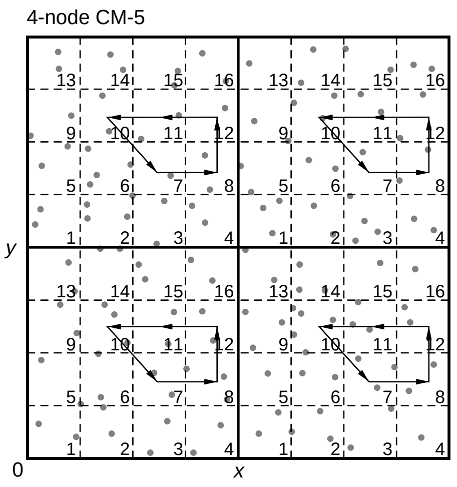


(c) The cell structures of each processor domain

The domain assigned to each processor is subdivided into cells having dimensions just larger than r_{max}, the interaction cut-off distance. In this example the domain of each processor has been divided into 16 cells. The circle of radius r_{max} is centered on a particle and indicates the area containing all the other particles that are close enough to affect the motion of the center particle. Note that all the particles contained in this circle are located either within the same cell as the center particle or within adjacent cells. Associated with each cell is a small block of memory containing a sequential list of the particle data for that cell.

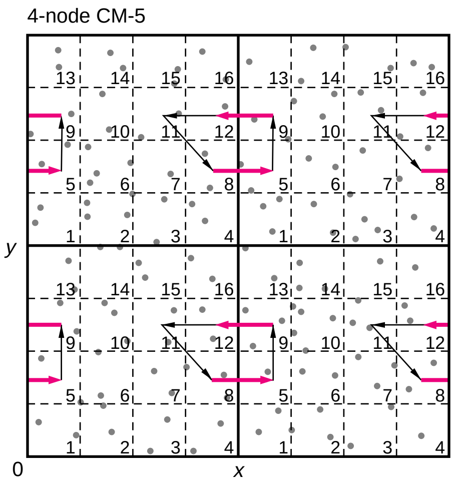
Figure 4. MD Force Calculation

This figure shows the steps that are performed for each cell in order to calculate and sum up the forces acting on all particles. Each processor works on its cells in the order 1 through 16 and follows an identical interaction pathway for each cell.



(a) Interaction path for cell 7

The processors are performing the interaction calculations for cell 7, having already done so for cells 1 through 6. First, all forces between particles within cell 7 are computed and the results are sent to an accumulator that adds all the pair-wise contributions for each particle. Then the block of memory containing the list of particle data for cell 8 is called up and used to calculate th forces exerted on particles in cell 7 by particles in cell 8. The results are sent to the accumulato for cell 7. By Newton's third law, the forces exerted on particles in cell 8 by particles in cell 7 are equal and opposite to those exerted on particles in cell 7 by particles in cell 8. Consequently, th results are also sent to the accumulator for cell 8. The process is repeated three times more for cell 7, calculating forces involving particles in cell 7 with those in cells 12, 11, and 10. The force exerted on particles in cell 7 by particles in cells 2, 3, 4, and 6 were sent to the accumulator for cell 7 when the interaction pathways were followed for cells 2, 3, 4, and 6. Because all the particles that must be considered for cell 7 are within the domain of a single processor, no message passing is required.



(b) Interaction path for cell 8 (synchronous message passing steps are red)

The processors are calculating forces for cell 8. Because cell 8 is located at the edge of a processor domain, some steps of the interaction path (shown by red arrows) cross the boundary between processor domains. To carry out the entire force calculation for cell 8 (and for all other cells along the edge of processor domains) the data for the particles in cell 8 must be made available to neighboring processors. Synchronous message passing is used to transfer the particle data for cell 8 to and from neighboring processors. Note that because the computationa box has periodic boundary conditions, interaction paths leave the box at one edge while equivalent interaction paths enter the box at the opposite edge. Because each processor has the same cell structure and proceeds through the cells in the same order, all the processors will need to send and/or receive particle data for a given edge cell at approximately the same time. Those processors that are assigned to neighboring domains in the computational box will exchange data with one another during the force calculation. Therefore, synchronization of message passing between the appropriate processors is straightforward.

(c) Synchronous message passing between processing nodes (PN)

One processor is sending data to another via synchronous message passing. The source processor stops computing, issues a "Ready to send" signal, and waits for a "Ready to receive" signal from the destination processor. After the "Ready to receive" signal is issued, the data is transferred, and then both processors resume calculations.

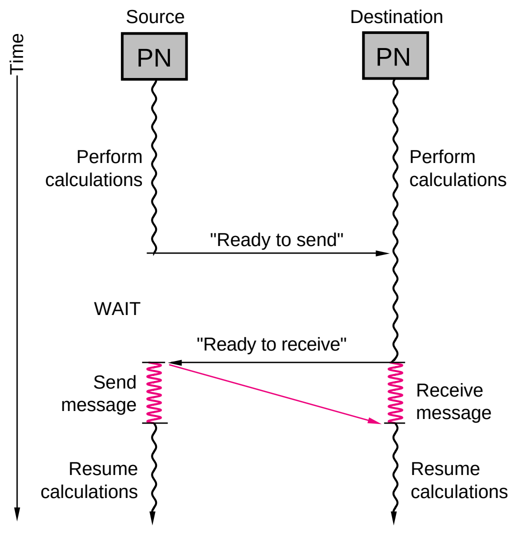
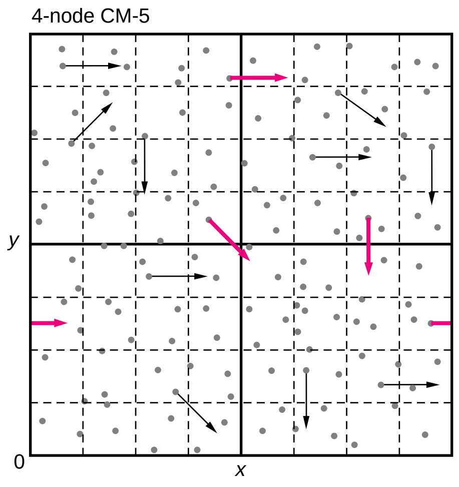
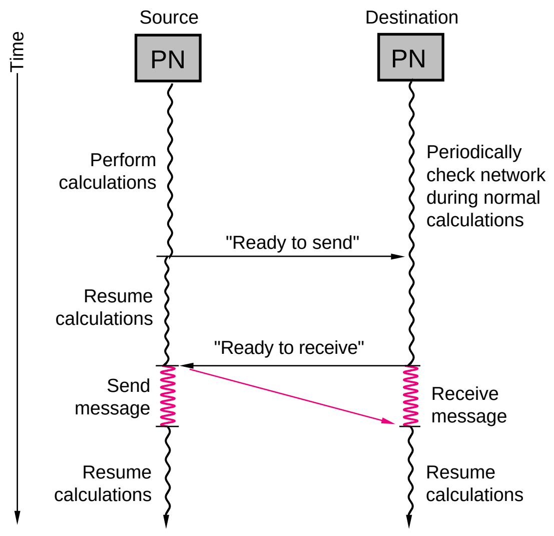


Figure 5. Redistribution of Particles
After the force calculation is completed for a given timestep, new positions and velocities are calculated for all the particles.



(a) Moving particles to proper cells (asynchronous message passing steps are red)

The new coordinates of each particle are checked by a special redistribution function. For each case in which the new coordinates of a particle correspond to a location in a different cell, the redistribution function also transfers the data for that particle to the sequential list of particle data for the new cell. Most of the transfers are between cells within the same processor domain. Those transfers that cross a boundary between processor domains are executed by means of message passing between processing nodes and are indicated by red arrows in the figure.



(b) Asynchronous message passing between processing nodes (PN)

Since there is no way to know beforehand which processors will be involved in the transfer of particles, the transfer is accomplished by an asynchronous mode of message passing. During asynchronous message passing, the source processor issues a “Ready to send” signal and immediately resumes calculation (contrast with synchronous message passing in Figure 4c). All processors periodically check the network and retrieve any waiting messages. The only significant interruption to the calculations of either node is the time used for the transfer of the message.

	Processors					
Particles	32	64	128	256	512	1024
1,024,000	8.90	4.51	2.32	1.26	0.72	0.44
2,048,000	—	8.96	4.44	2.46	1.36	0.74
4,096,000	—	—	8.79	4.81	2.67	1.36
8,192,000	—	—	16.83	8.81	4.80	2.47
16,384,000	—	—	—	16.95	8.74	4.49
32,768,000	—	—	—	—	16.90	8.54
65,536,000	—	—	—	—	—	16.55
131,072,000	—	—	—	—	—	34.26

Table 1. Update Times per Timestep in Scaling Simulations
Shown in the table are the timing results for a set of test simulations ($r_{max} = 2.5\sigma$). Each entry in the table is the CPU time (in seconds) required to compute a single timestep for the given combination of numbers of particles and processors.

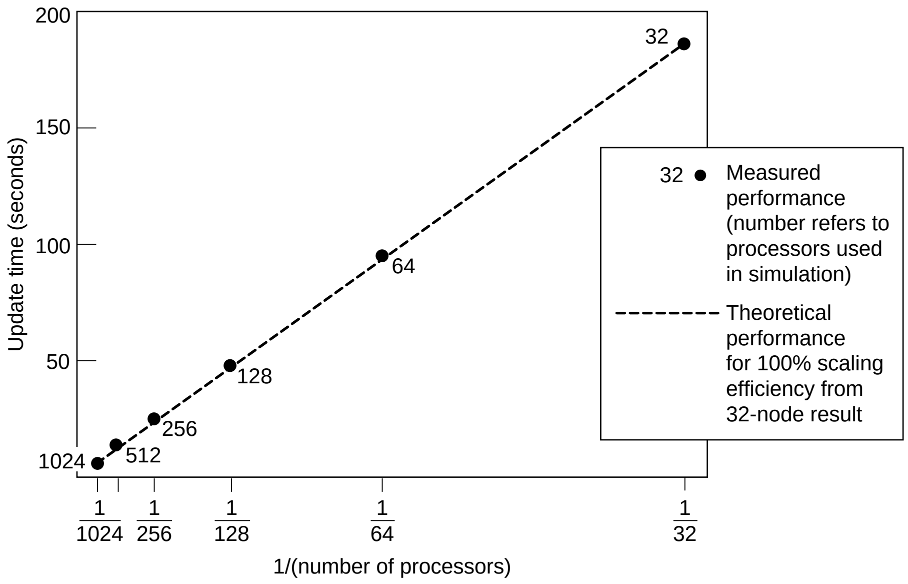


Figure 6. The Scalability of the MD Algorithm
The figure is a plot of the CPU time required to simulate a single timestep versus the reciprocal of the number of processors used for the calculation. The dots represent the results obtained from a 4-million-particle simulation ($r_{max} = 5\sigma$) in which particles rearranged from a simple cubic lattice to a face-centered cubic lattice. The line represents perfect scaling from the 32-node data point, that is, an update time that is inversely proportional to the number of processing nodes used in the simulation. The results of this series of test calculations indicate that the performance of our algorithm, at least for this type of simulation, scales very well with the number of available processors.

to the particle list of the new cell (see Figure 5a).

There are two types of particle transfers to consider. One involves the simple transfer of a particle from one cell to another within a single processor domain. In this case, deleting particle data from one cell and adding it to a new cell requires only copying of memory contents within a single processing node. The second type of transfer requires moving particle data to a cell list in another domain, so the particle data is deleted from the original cell list and sent to the new processor by message passing. The redistribution process cannot use synchronized message passing since it is not known how many particles will leave each processor, how many will be received by each processor, and from what processors particles will arrive. Therefore all processors are put into an asynchronous message-passing mode. Asynchronous message passing occurs in the background while the particle coordinates are checked (see Figure 5b). The messages are addressed so that they go to the appropriate processor, and each node periodically checks the network and receives any messages that are waiting. Messages may be sent or received at any time, and, in general, all processors operate independently. The redistribution of particle data is complete when all particle coordinates have been checked and all messages have been retrieved from the data network.

Our experience indicates that the redistribution of particles is efficient and accounts for a small portion of the overall iteration time. Generally, the number of particles changing cells after any given timestep is small compared with the total number of particles in the system. In addition, since most of the particles that change cells simply move to another cell on the same processor, the inter-processor communication required for the redistribution procedure is minimal.

The architecture and capabilities of the CM-5 are well suited for molecular-dynamics applications. The high-speed communication networks allow us to use message passing to efficiently transfer data among the processors. An equivalent domain was assigned to each processor in order to divide the workload among the processors, and the cell structure and interaction path ensure that the interaction calculations are completed in a minimum amount of time.

The Scalability of our MD Algorithm

In ideal circumstances, the speed of a well-designed parallel algorithm will scale linearly with the number of processors used in the parallel computer. To test the scalability of our algorithm we repeatedly performed a test simulation using various numbers of processors and various numbers of particles ranging from 1 million to 131 million. The starting condition for the tests consisted of identical particles arranged in a simple cubic lattice. The simple cubic lattice—known to be unstable for an interaction potential that depends only on r and to undergo a phase change in which the particles rearrange to a face-centered cubic configuration—was chosen to guarantee movement of particles between domains during the calculation.

Table 1 shows the results of the scaling tests in which $r_{\max}$ was set equal to 2.5σ. Each doubling of the number of processors used for a given number of particles approximately halved the update times (the CPU time required to compute all particle motions during a single timestep). For a 4-million-particle simulation in which $r_{\max}$ was equal to 5σ, the increase in speed in going from 32 processors to 1024 processors was more than a factor of 30, corresponding to 95% parallel efficiency. The latter scaling results are shown graphically in Figure 6, where the update times for the 4-million-particle simulation are plotted as a function of the reciprocal of the number of processors used. The scalability of our MD algorithm was also demonstrated by comparing the update times for varying numbers of particles (see Table 1). For a given number of processors, the update times increased linearly with the number of particles in the simulation.

Applications

As a demonstration of the practicality of our algorithm, we performed a simulation of a 1-million-particle projectile colliding with a 10-million-particle plate. Figure 7 is a series of images taken from the simulation; each image is a "snapshot" showing the positions and velocities of the particles at the end of a timestep. The velocities are indicated by color—red and yellow particles are moving faster than blue and gray particles. The top panel in Figure 7 shows the system at timestep 1000, just after the projectile has made contact with the plate. Note that the particles in the lower third of the projectile have higher kinetic energies (higher velocities) than those in the remainder of the projectile. Some of the particles in the plate also have increased kinetic energies. At timestep 2000 (middle panel of Figure 7) the projectile has nearly penetrated the plate and part of the projectile has begun to break up. A shock wave has propagated to the edges of the plate. By timestep 2900 (bottom panel of Figure 7) part of the projectile has been absorbed into the plate, the remainder of the projectile has disintegrated into free particles, and the shape of the plate has become significantly distorted.

The impact simulation shows the inherently unstructured nature of MD simulations. Tracking the motions of individual particles often requires significant amounts of communication and data management. We have been able to achieve high performance by carefully mapping the physics of the problem to the architecture of the CM-5. The fast iteration times we have attained allow the calculation of larger MD simulations than have previously been possible. For example, the illustration at the beginning of this article was taken from one of the largest MD simulations performed to date—our 38-million-particle simulation of a plate undergoing fracture. Because our algorithm is scalable, the size of the physical system that can be modeled is expected to increase proportionally as each new generation of massively parallel machines is developed.

We are now working to incorporate new types of interaction potentials into our MD algorithm. The particular atomic interactions will be described by materials-specific interaction potentials like the "embedded-atom method" potential. Such potential functions are created by adding a "many-body" term to the pair-interaction potential presented in this paper. The many-body term varies according to the local density surrounding a given atom and thereby results in a more physically realistic interaction potential.

We are using our MD algorithm to study the physics of fracture. We are particularly interested in understanding the brittle and ductile behavior of materials. In ductile fracture the propagation of cracks is accompanied by the formation of a significant number of dislocations (discontinuities in the crystalline structure of a material) at the

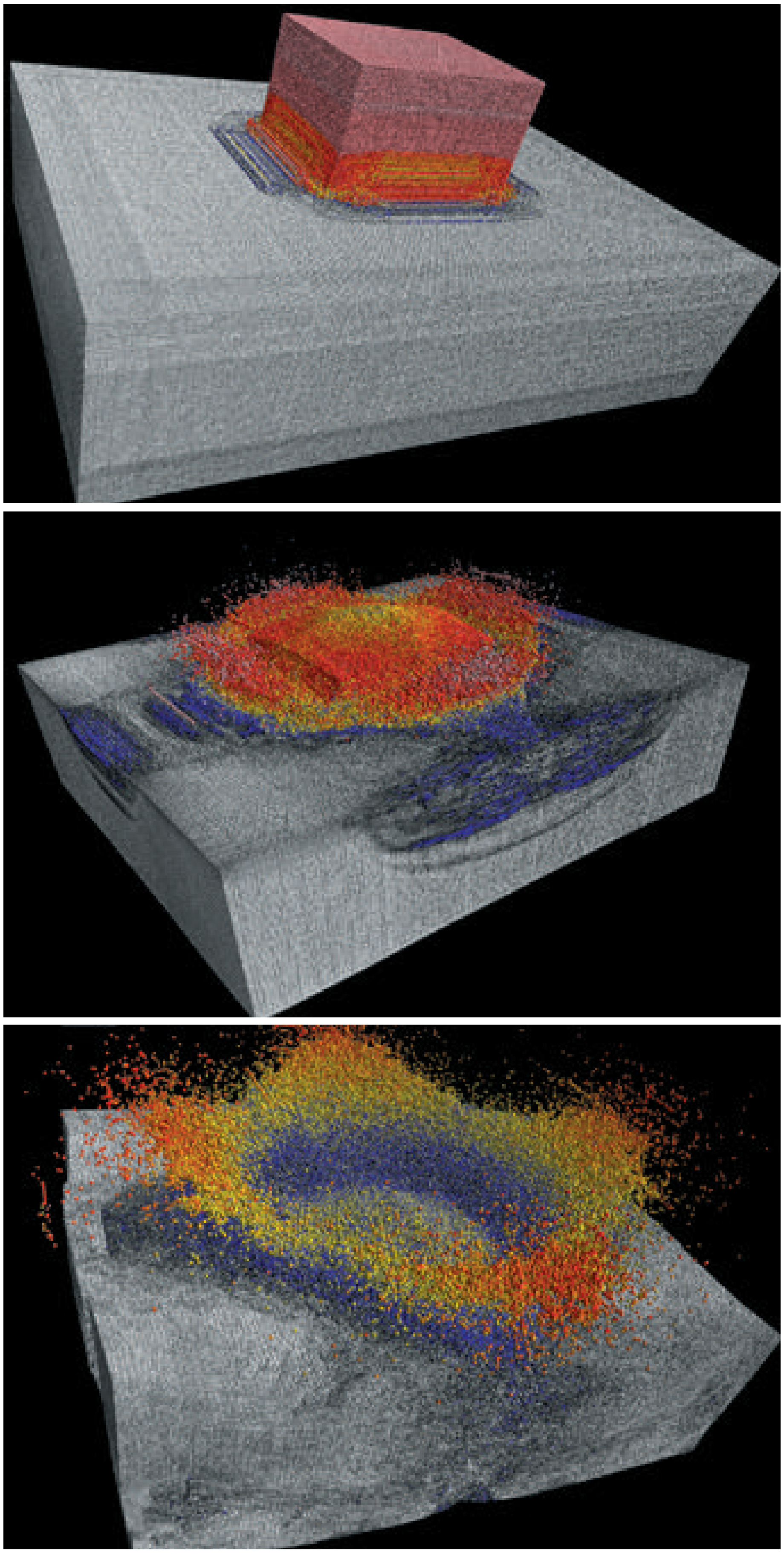


Figure 7. An 11-Million-Particle Impact Simulation
The three images show the progress of an MD simulation in which a 1-million-particle projectile impacts a 10-million-particle plate. The particles of both the projectile and the plate were initially at rest in equilibrium positions in face-centered cubic lattices. The interactions between particles were approximated by using a Lennard-Jones potential. The colors in the figure represent the velocities of the particles; red and yellow particles have higher velocities than blue and gray particles. The simulation of 2900 timesteps required 30 hours of CPU time on a 512-processor CM-5. The top, middle, and bottom panels correspond to timesteps 1000, 2000, and 2900, respectively.

crack tip. These dislocations are responsible for permanent deformations in the material. When the fracture is brittle, the crack propagates with little or no permanent deformation. Our goal is to understand and ultimately control parameters that influence these phenomena because such knowledge and ability will allow us to design materials having specific responses and behavior. Large-scale molecular dynamics is an ideal tool for the investigation of fracture phenomena because MD allows us to perform simulations of near-macroscopic samples using very few simplifications or approximations. ■

David M. Beazley (left) is a graduate student in the Center for Nonlinear Studies and the Condensed Matter and Statistical Physics Group. His research interests include massively parallel supercomputing, high-performance computer architecture, and scientific computing. Beazley began working at the Laboratory while an undergraduate in 1990. He received a B.A. in mathematics from Fort Lewis College in 1991, and is currently working on a Ph.D. in computational science at the University of Utah.

Peter S. Lomdahl is a staff member in the Condensed Matter and Statistical Physics Group in the Theoretical Division, where he has worked on computational condensed-matter and materials-science research since 1985. From 1982 to 1985 he was a postdoctoral fellow with the Center for Nonlinear Studies. Lomdahl received his M.S. in electrical engineering and his Ph.D. in mathematical physics from the Technical University of Denmark in 1979 and 1982. His research interests include parallel computing and nonlinear phenomena in condensed-matter physics and materials science.

Acknowledgements

It is a pleasure to acknowledge the contributions of Niels Grønbech-Jensen, Pablo Tamayo, and Brad L. Holian to parts of the work described here. We also acknowledge generous support from the staff of the Advanced Computing Laboratory, who provided both machine access and assistance. In particular, David O. Rich helped with the use of the CM-5, and Michael F. Krogh provided invaluable help with visualization.

Further Reading

M. P. Allen and D. J. Tildesley. 1987. *Computer Simulations of Liquids.* Clarendon Press.

D. M. Beazley and P. S. Lomdahl. 1994. Message-passing multi-cell molecular dynamics on the Connection Machine 5. *Parallel Computing* 20: 173–195.

D. M. Beazley, P. S. Lomdahl, P. Tamayo, and N. Grønbech-Jensen. 1994. A high performance communications and memory caching scheme for molecular dynamics on the CM-5. In *Proceedings of the Eighth International Parallel Processing Symposium.* IEEE Computer Society.

R. C. Giles and P. Tamayo. 1992. A parallel scalable approach to short-range molecular dynamics on the CM-5. In *Proceedings of Scalable High Performance Computing Conference 1992.* IEEE Computer Society.

P. S. Lomdahl, P. Tamayo, N. Grønbech-Jensen, and D. M. Beazley. 1993. 50 GFlops molecular dynamics on the connection machine. In *Proceedings of Supercomputing 1993.* IEEE Computer Society.

S. Plimpton. 1993. Fast parallel algorithms for short-range molecular dynamics. Sandia National Laboratory Report SAND91-1144, UC-705.

P. Tamayo, J. P. Mesirov, and B. M. Boghosian. 1991. Parallel approaches to short range molecular dynamics simulations. In *Proceedings of Supercomputing 1991.* IEEE Computer Society.