50 GFlops Molecular Dynamics on the Connection Machine 5

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

Abstract

We present timings and performance numbers for a new short range three dimensional (3D) molecular dynamics (MD) code, SPaSM, on the Connection Machine-5 (CM-5). We demonstrate that runs with more than 10^8 *particles are now possible on massively parallel MIMD computers. To the best of our knowledge this is at least an order of magnitude more particles than what has previously been reported. Typical production runs show sustained performance (including communication) in the range of 47-50 GFlops on a 1024 node CM-5 with vector units (VUs). The speed of the code scales linearly with the number of processors and with the number of particles and shows 95% parallel efficiency in the speedup.*

1 Introduction

The use of molecular dynamics (MD)[1] to study dynamical properties of solids and liquids has been known for decades, but it is only the recent proliferation of powerful massively parallel computers that begins to makes detailed studies of realistically sized systems possible. A cube of material 1000 atoms on the side measures roughly $0.5\mu m \times 0.5\mu m \times 0.5\mu m$ - while this may seem like a very small piece of material - it contains 10^9 atoms. Solving Newton's equations for a billion interacting atoms still represents a formidable problem for MD. However, realistic calculations in materials science require system sizes in this range if the dynamics of defects like dislocations and grain-boundaries is to be studied. An additional problem is presented by the short time scale that is accessible by the MD method, which is typically tens or maybe hundred of *nano*-sec. at best. Ideally one would like to use the MD method for second long simulations with at least 10^9 atoms. While this goal is still very far away, there is substantial current interest in the development of fast MD algorithms[2, 3, 4, 5, 6, 7] which allow for the simulation of at least million atom systems.

We have developed an new scalable MD algorithm based on a message-passing multi-cell approach which allows for simulating at least 10^8 particles interacting via a relative short range potential. We have implemented the algorithm in a code, SPaSM (Scalable Parallel Short-range Molecular dynamics), on the Connection Machine 5 (CM-5) and demonstrated that simulations with tens of millions of atoms can now be performed routinely. In addition, it is clear that simulations with 10^8 particles are now possible at a sustained rate of 50 GFlops. To our knowledge the performance numbers are the best reported to date for any MD simulation and may well be the highest for any 3D production code implementing a substantial amount of unstructured communication. In preliminary 2D studies, we simulated the fracture dynamics of a piece of material with 2 million atoms that is being pulled apart in a tensile experiment. We are currently using SPaSM to study dynamic fracture physics with millions of atoms in 3D.

2 MD Simulations

The MD method concerns the solution of Newton's equation of motion for N interacting particles. This general N-body problem involves the calculation of $N(N-1)/2$ pair interactions in order to compute the total force on any given particle:

$$m_i \frac{d^2\mathbf{r}_i}{dt^2} = -\sum_{j\neq i} \frac{\partial V_{ij}(|\mathbf{r}_j - \mathbf{r}_i|)}{\partial \mathbf{r}_i}, \qquad (1)$$

here $\mathbf{r}_i$ indicates the instantaneous position and m_i the mass of particle i. The complexity of the force calculation is simplified considerably if the potential $V_{ij}(r)$ has a finite range of interaction. This is a reasonable approximation of the atomistic interactions in many solids and fluids. In our timings here we have used the Lennard-Jones 6-12 (LJ) potential

$$V(r) = \begin{cases} 4\epsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) & 0 < r \leq r_{max} \\ 0 & r_{max} < r \end{cases} \tag{2}$$

Here σ and ϵ are the usual LJ parameters. The potential is cut-off at r_{max}, i. e. no particles interact beyond this range. We include here timings and performance numbers for two values of r_{max}. We stress that while more complicated and accurate potentials which include many-body effects are available, the amount of work needed in the force calculation is represented well by the LJ potential especially when the cut-off is $r_{max} = 5\sigma$. The number of interacting neighbors for each particle depends on the value of the cut-off distance r_{max} and the particle density ρ.

Our code is written in ANSI C with explicit calls to the CMMD message-passing library. The kernel of the force calculation is coded in the CM-5 vector unit (VU) assembler language, CDPEAC, and consists of approximately 60 lines of code. All our calculations were performed in *double precision*.

3 Multi-cell algorithm

Our algorithm has been described in detail in[7]. Here we briefly outline its main features, illustrating the algorithm in 2D, but it extends naturally to 3D. Space is considered to be a rectangular region with periodic boundary conditions.

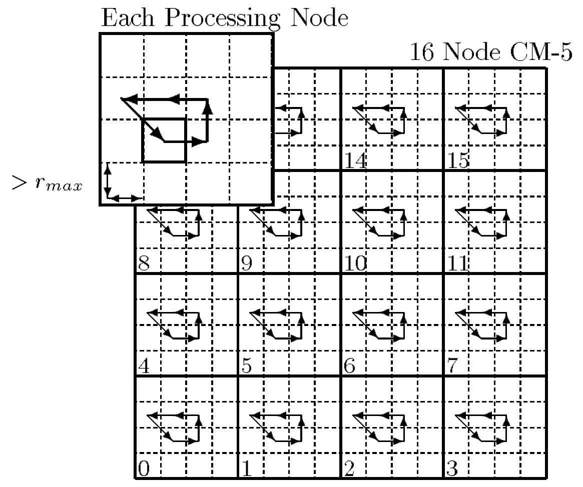

Figure 1. Processor layout and force calculation.

This region is subdivided into large cells that are assigned to the processing nodes (PNs) on the CM-5. The region assigned to each PN is further subdivided

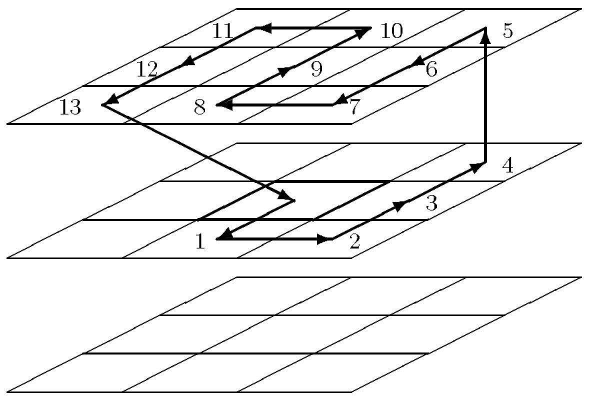

Figure 2. 3D interaction path

into small cells with dimensions slightly larger than the cutoff distance. Particles are assigned to a particular cell geometrically according to the particle's coordinates. In Fig. 1, solid lines represent processor boundaries while dashed lines represent the cells created on each PN. For large simulations, many thousands of cells per PN may be created (this does not explicitly depend on the number of PNs being used). Associated with each cell is a small block of memory for storing a sequential list of particles. To compute the forces for particles in a cell, we first compute all of the interactions between particles in that cell. Afterwards, forces between particles in neighboring cells are calculated by following an interaction path that visits neighboring cells. The path in 2D is shown in Fig. 1 and in 3D in Fig. 2. As we follow the path, accelerations are accumulated by the original cell and any visited cells (using Newton's third law). To calculate all of the forces, this procedure is carried out on all cells on all of the PNs. Cells will accumulate accelerations from their lower neighbors when they calculate their interactions. Whenever the interaction path crosses a processor boundary, message passing is used to communicate particle data. After all forces have been calculated, the particle positions are updated. Since our algorithm is geometrically based, all of the data structures must be updated to account for positional changes. The particle coordinates are checked and if a particle is in the wrong cell it is moved to the proper cell. If the new cell is on a different PN, asynchronous message passing is used to send the particle to its new PN. Each PN checks for incoming particles and places them in the proper cell when received. Since a large number of cells may be created on each PN (even for moderately sized systems) hundreds or even thousands of message-passing calls may be re-

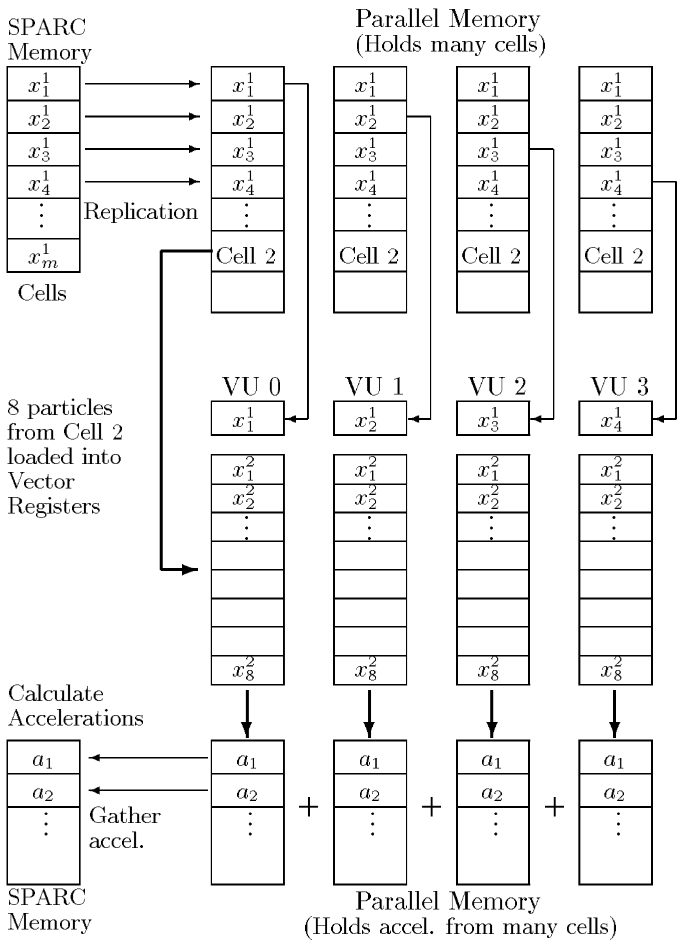

Figure 3. Calculating forces on the VUs.

quired for each time step. The amount of communication and calculation for 3D is substantial due to an increased number of neighboring cells and a more complicated interaction path.

4 Using The Vector Units and Parallel Memory

On each processing node, the CM-5 has four vector units (VUs) that perform fast vector arithmetic in a SIMD mode. Each VU has a peak speed of 32 Mflops for a combined speed of 128 Mflops per node. In addition to performing arithmetic operations, the VUs also act as memory controllers with each VU controlling an 8 Mbyte bank of memory. The 32 Mbytes of memory on each PN can be accessed by both the SPARC processor and the VUs, but the memory is divided into two separate areas for this purpose. SPARC memory is memory that has been allocated for use by the SPARC processor. All usual SPARC operations perform normally in this area. Parallel memory is a special memory allocation that allows the four VUs to perform simultaneous load/store operations. Each VU allocates an identically structured memory region in the 8 Mbyte memory bank that they control. When loads or stores are performed, each VU accesses its particular bank. This allows the VUs to operate on four different data sets in a SIMD mode. The SPARC processor can access any particular bank of parallel memory, but the VUs can not directly access SPARC memory. Transferring data between the two memory regions can be done using special instructions, but accessing SPARC memory from the VUs is slow and should be avoided as much as possible. As a general rule, any operations involving the VUs must use parallel memory for optimal performance.

To access the VUs, we have implemented the force calculation in CDPEAC (the assembler language for controlling the CM-5 vector units). The kernel of the force calculation takes two cells of particles and calculates the resulting accelerations between the particles. This calculation is described in detail in Fig. 3. First, the particle coordinates from the two cells are copied from SPARC memory and replicated across all four VUs in parallel memory. Eight particles from cell 2 are then loaded onto all four VUs. We then loop over all of the particles in cell 1 and calculate the accelerations between these particles and the eight particles loaded from cell 2. At each step, four different particles from cell 1 are loaded (a different particle on each VU). This allows the VUs to calculate 32 interactions simultaneously. Once all particles in cell 1 have been processed, the next set of eight particles from cell 2 is loaded and the process is repeated. The calculation continues, calculating 32 interactions per step, until all accelerations have been calculated. Afterwards, the resulting accelerations are gathered from parallel memory and saved back to SPARC memory.

All internal data structures in our code are stored in SPARC memory. This allows the data to be easily accessible to functions that do not require the VUs. Whenever particles are used in the force calculation, they are copied to parallel memory. For simulations with a cutoff of $r_{max} = 5\sigma$, each cell may contain several hundred particles and most of the time is spent calculating interactions in the force kernel. The extra overhead associated with copying the particles from SPARC memory to parallel memory is small so we pay a minimal penalty for using SPARC memory in this case. For simulations with a smaller cutoff such as $r_{max} = 2.5\sigma$, the number of cells per processor increases dramatically while the number of particles per cell decreases. In this case, it is more difficult to keep the VUs busy and our use of parallel memory becomes more critical.

To obtain better performance with a smaller interaction cutoff r_{max}, several modifications have been made. The main performance problem is that of loading the particles into parallel memory from SPARC memory. If we use the same scheme developed for large cutoff distances, each cell is loaded into parallel memory whenever needed in the force calculation. This results in each cell being loaded to parallel memory as many as 14 times (once when calculating self-interactions and 13 times when neighboring cells calculate their interactions). To reduce the amount of loading, a parallel memory caching scheme has been implemented. A buffer for holding a collection of cells is allocated in parallel memory. Each time a cell is encountered in the force calculation, this buffer is checked to see if that cell has already been loaded. If not, it is loaded to parallel memory and the previous contents (if any) saved back to SPARC memory (including accelerations). The loading process operates according to a FIFO scheme and eventually new cells will begin to replace previously loaded cells in the cache. As the force calculation proceeds, previously loaded cells will no longer be necessary in the calculation (after they have remained in the cache sufficiently long) and can be saved back to SPARC memory without having to be reloaded. This property is due to the fact that the interaction path has a finite range and will not see cells that were loaded much earlier in the calculation. By making the cache sufficiently large, each cell will be loaded to parallel memory only once and a substantial improvement in performance is obtained.

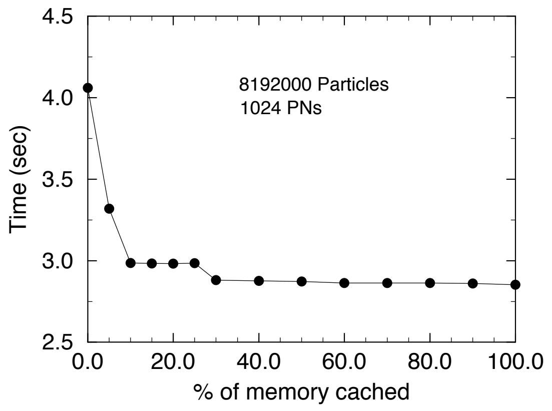

Figure 4 : Effect of caching on iteration time

In our code, the amount of memory available for caching can be adjusted. This gives us increased flexibility since our code can be optimized for memory (by using a small cache) or for speed (by using a larger cache). This allows us to run small simulations at increased speed or large simulations with more than 10^8 particles by simply adjusting our memory usage. Caching has significantly improved our code performance for simulations with a cutoff of $r_{max} = 2.5\sigma$. The speedup obtained by caching for a particular simulation is shown in Fig. 4. As more memory is added to the cache, the iteration time drops rapidly. In the figure we also see that little speedup is gained by caching more than 30% of the cells. For very large simulations, we have found that even adding a small cache of 1-2% of the cells can dramatically improve performance.

5 Timings and performance

In Table 1 ($r_{max} = 5\sigma$) we summarize the timings for runs on a variety of CM-5 processor partitions with different number of particles, N, in the range from 1 million to 131 million. The particles, in each case, were arranged in a uniform 3D cubic lattice at constant density $\rho = N/\sigma^3 = 1$. With this density and interaction cutoff, each particle has approximately 520 interacting neighbors. This configuration is unstable and will undergo a phase change where the particles rearrange in an face-center-cubic (fcc) configuration. This choice of initial conditions thus guarantees that the particles are moving between processors and realistic inter-processor communication is involved.

In the table, the update time per time step and the corresponding GFlop rates are given (the numbers in parenthesis are the GFlop rates.) The GFlop rates were obtained by counting the total number of interactions between the particles during a time step. Each interaction involves a force calculation with 42 floating point operations in the CDPEAC kernel (counting multiply, add, and compare as one operation each and divide as five)[8]. The GFlop numbers are calculated by multiplying the total interaction count by 42 and dividing this number with the time for a time step (measured with `CMMD_node_timer_elapsed`). This procedure was then repeated and averaged over many time steps. Our numbers thus include both computation and inter-processor communication and reflects the speed of realistic production runs.[1]

In Table 2 we summarize our recent timings for runs with a cutoff of $r_{max} = 2.5\sigma$. The update time per time step and the GFlop rates are given. All runs were performed using parallel memory caching. In each case, 25% of the cells were cached except for the run with 131 million particles that used a 3% cache. With a cutoff of $r_{max} = 2.5\sigma$, each particle has approximately 65 interacting neighbors. Since each particle has fewer neighbors, it is more difficult to keep the VUs busy during the force calculation. Consequently, the GFlop rates are lower. However, our best timing for $r_{max} = 2.5\sigma$ is our run with 65 million particles on 1024 PNs. In this case, the update time is 16.55 seconds which corresponds to 250 *nano*-sec. per particle update. To the best of our knowledge, this is the best reported time to date [6]. Using minimal parallel memory caching, we were also able to simulate 180 million particles with an update time of 55.6 seconds.

[1]The bare kernel of the force calculation (with no communication or SPARC memory operations) runs at 68.5 GFlops.

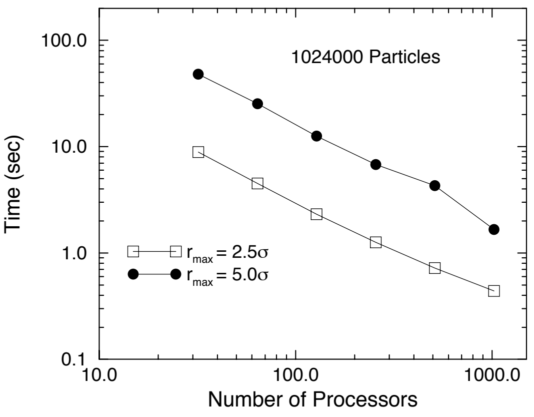

Figure 5 : MD update time vs. number of processors

In Fig. 5 we illustrate the scaling properties of our code. The data are for runs with 1 million particles and two different values of the cut-off $r_{max} = 5\sigma$ and $r_{max} = 2.5\sigma$. A near linear dependence is found for both values of the cut-off. This is of course also evident from the numbers in Tables 1 and 2. It should also be noted that our algorithm scales linearly with the number of particles for a fixed number of processors.

	$r_{max} = 5\sigma$		$r_{max} = 2.5\sigma$	
Particles	Comp.	Comm.	Comp.	Comm.
1024000	82.5%	17.5%	78.0%	22.0%
4096000	88.5%	11.5%	77.2%	23.8%
16384000	91.9%	8.1%	84.8%	15.2%
65536000	93.2%	6.8%	89.5%	10.5%
131072000	94.4%	5.6%	90.8%	9.2%

Table 3: Timing breakdown for 1024 PNs

In Table 3 the breakdown of computation and communication time is given. For the larger simulations, each processor may have many thousands of cells. As a result, calculating the accelerations may require several thousand message passing calls. Depending on the value of r_{max}, each message passing call may involve a transfer of 800-10000 bytes. Despite the large amount of communication, our algorithm is dominated by the calculation of forces for both values of r_{max}. In

	Processors					
Particles	32	64	128	256	512	1024
1024000	47.94 (1.5)	25.41 (2.9)	12.56 (5.8)	6.79 (10.7)	4.30 (16.9)	1.66 (43.7)
2048000	94.39 (1.5)	48.39 (3.0)	24.43 (5.9)	12.75 (11.4)	6.92 (21.0)	3.16 (47.0)
4096000	186.83 (1.6)	95.28 (3.1)	47.98 (6.0)	25.57 (11.4)	14.00 (20.7)	6.17 (47.0)
8192000	-	188.00 (3.1)	95.11 (6.1)	50.63 (11.5)	28.12 (20.6)	12.13 (47.8)
16384000	-	-	185.77 (6.2)	95.52 (12.2)	52.60 (22.1)	23.68 (49.0)
32768000	-	-	-	190.81 (12.2)	101.55 (22.9)	46.71 (49.7)
65536000	-	-	-	-	204.91 (22.6)	92.91 (50.0)
131072000	-	-	-	-	-	183.01 (50.7)

Table 1: Update times per time step in sec. (GFlops in parenthesis). Cut-off: $r_{max} = 5\sigma$

	Processors					
Particles	32	64	128	256	512	1024
1024000	8.90 (0.81)	4.51 (1.61)	2.32 (3.12)	1.26 (5.74)	0.72 (10.07)	0.44 (16.55)
2048000	-	8.96 (1.62)	4.44 (3.26)	2.46 (5.90)	1.36 (10.65)	0.74 (19.54)
4096000	-	-	8.79 (3.29)	4.81 (6.03)	2.67 (10.84)	1.36 (21.27)
8192000	-	-	16.83 (3.44)	8.81 (6.58)	4.80 (12.07)	2.47 (23.50)
16384000	-	-	-	16.95 (6.84)	8.74 (13.26)	4.49 (25.82)
32768000	-	-	-	-	16.90 (13.72)	8.54 (27.14)
65536000	-	-	-	-	-	16.55 (28.01)
131072000	-	-	-	-	-	34.26 (27.06)

Table 2: Update times per time step in sec. (GFlops in parenthesis). Cut-off: $r_{max} = 2.5\sigma$.

the table, computation time includes the calculation of forces and the numerical integration. The communication time includes all message passing during the interaction calculation and the time to redistribute the particles after each time step.

With a cutoff of $r_{max} = 5\sigma$, the speedup from a run with 4 million particles on a 32 node CM-5 to the same run on a 1024 node CM-5 is over a factor 30 and corresponds to 95% parallel efficiency. We were also recently able to run SPaSM on a 4 PN CM-5 with 1 million particles. The update time here was 367 sec. The speedup achieved with same run on 1024 PNs is a factor 221, representing 86% parallel efficiency. Our best performance number, 50.7 GFlops, represents 40% of the theoretical peak performance of 128 GFlops on a 1024 node CM-5. The 50.7 GFlops also represents a cost/performance number of 1.95 GFlops/$Million.

6 Conclusion

We have demonstrated that three-dimensional multi-million particle MD simulations can now be performed routinely on massively parallel MIMD computer systems. To demonstrate the practicality of the algorithm, we present a few time frames from a million particle impact simulation. The simulated particles have an interaction cut-off of $r_{max} = 2.5\sigma$ and the time step of the integration is $\Delta t = 0.01$ time units. The system is initiated with one block of particles in a fcc lattice with $200\times200\times25$ (10^6) atoms, and a projectile in a fcc lattice with the dimensions $20\sigma\times20\sigma\times40$ layers (14000) atoms. The projectile has a velocity of 10 towards the block ($\sim$1.3 times the sound velocity in the lattice). This initial condition is shown in Fig. 6a. In Fig. 6b we show the system at $t = 2.5$. The projectile has made contact with the block and has partially penetrated. The hexagonal nature of the lattice is seen to dominate the phonons emitted on the surface of the block, even though the projectile makes contact with a square (20×20) shape. Finally, in Fig. 6c, we show the particles at $t = 5$, where part of the projectile has been absorbed in the block. Other parts of the projectile have disintegrated into almost free particles. We are currently using SPaSM to perform other multi-million atom simulations.

The impact simulation shows the inherently unstructured nature of MD simulations. In principle, every particle is free to move to any location within the system. This may require a substantial amount of communications and data management. However, we have been able to achieve high performance by carefully analyzing the problem and mapping it to the architecture of the CM-5. Our algorithm is dominated by computation with communication requiring only 5-20% of the overall time. Our algorithm scales linearly with the number of processors and the number of particles. With an interaction cutoff of $r_{max} = 5\sigma$, runs with a sustained calculation rate of 50 Gflops can be performed on a 1024 PN CM-5. With a smaller cutoff of $r_{max} = 2.5\sigma$, we can achieve an update time of 250 *nano*-seconds per particle. This fast update time allows us to run large MD simulations that have been impossible to perform in the past. Using all of the memory of the CM-5, we have also been able to simulate more than 180 million particles in 3D. While we may not be able to model 1 billion particles on current machines, this goal now seems within reach as next generation machines become available.

Acknowledgments

We thank the Advanced Computing Laboratory for generous support, in particular M. Krogh and D. Rich provided major assistance. We also express our appreciation to D. Dahl, A. Greenberg, C. Lobron, A. Mainwaring, and L. Tucker from TMC for valuable suggestions regarding CDPEAC and CMMD.

References

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

[2] A. I. Mel'čuk, R. C. Giles, and H. Gould, *Computers in Physics*, May/June 1991, p. 311.

[3] P. Tamayo, J. P. Mesirov, and B. M. Boghosian, *Proc. of Supercomputing 91*, IEEE Computer Society (1991), p. 462.

[4] B. L. Holian et al. Phys. Rev. A **43**, 2655 (1991).

[5] R. C. Giles and P. Tamayo, *Proc. of SHPCC'92*, IEEE Computer Society (1992), p. 240.

[6] S. Plimpton and G. Heffelfinger, *Proc. of SHPCC'92*, IEEE Computer Society (1992), p. 246.

[7] D. M. Beazley and P. S. Lomdahl, *Message-Passing Multi-Cell Molecular Dynamics on the Connection Machine 5*, Parall. Comp. (1993) (in press).

[8] *DPEAC Reference Manual*, Thinking Machines Corporation (1992), Cambridge, Massachusetts.

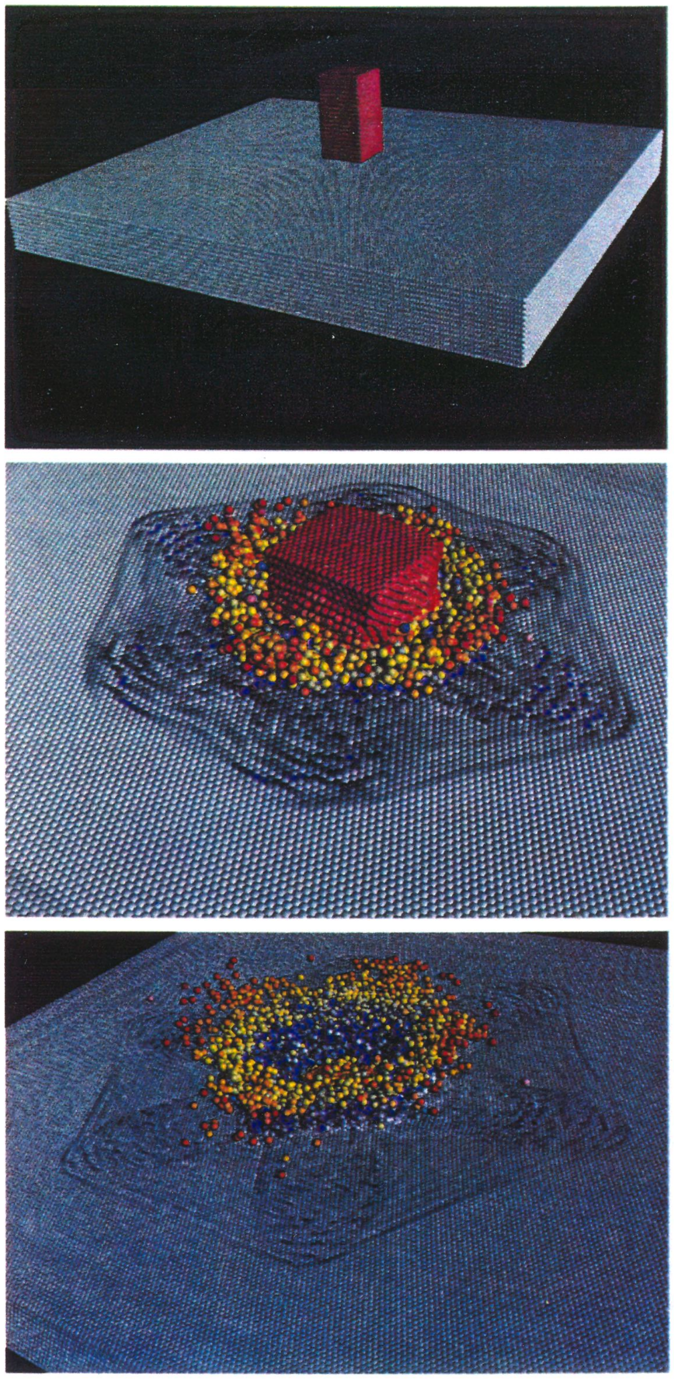

Figure 6. An impact simulation with 1014000 particles. Colors indicate kinetic energy with grey/blue indicating low energy and yellow/red high energy.