JOURNAL OF COMPUTATIONAL PHYSICS 90, 313–335 (1990)

A Special Purpose Computer for Molecular Dynamics Calculations

A. F. BAKKER,* G. H. GILMER, M. H. GRABOW, AND K. THOMPSON

AT&T Bell Laboratories, Murray Hill, New Jersey 07974

Received February 27, 1989; revised September 6, 1989

We have constructed a computer facility for interactive study of atomic systems, with fast turnaround between simulation runs. The computer includes a large external memory which is shared by up to 16 parallel processor boards. Each processor board contains four fast floating point chip sets, also operating in parallel. A host computer running the UNIX® operating system is used to assemble and download instructions to the processor boards and to transfer atomic coordinates to the memory. A machine with eight processor boards has a theoretical speed of 182 Mflops, and runs molecular dynmics code 30–100% faster than the in-house supercomputer. The architecture was chosen specifically for applications involving molecular dynamics code, using a new implementation of the algorithm, but it has also been programmed for finite difference calculations. In general, it should be effective for simulations of physical systems that can be subdivided into cells, such that the material of a cell is influenced only by local interactions.

1. INTRODUCTION

There has been much progress in recent years in the development of atomic scale models of materials. These models have proved to be powerful tools for the determination of the structure of interfaces, thin films, liquids, and glasses. They are also useful for simulations of the processing of materials, since they provide detailed information on atomic movements and mechanisms. Melting and solidification of silicon [1–4] and molecular beam epitaxy [5, 6] are some of the processes that have been studied recently.

The molecular dynamics (MD) method involves the determination of atomic trajectories for a group of interacting atoms. Given the initial positions and velocities of the atoms, and a set of interatomic potentials that describes the interaction between atoms, the force on each atom is calculated. The force is used to calculate the new velocity and position of each atom at a time Δt later, using classical equations of motion and an approximation scheme to extrapolate to time Δt. This process is repeated until the desired time period for the simulation has been spanned. The time step Δt used in the integration scheme must be short compared with, e.g., the vibrational period of an atom in a crystal.

* Permanent address: Applied Physics Department, Delft Technical University, 2600 G.A. Delft, The Netherlands.

0021-9991/90 $3.00

The interatomic force is of short range in many systems including semiconductors, and existing potentials extend to about 10 to 100 neighbors. Although pair interactions, in particular the Lennard–Jones potential, have been used extensively for MD simulations of close-packed materials, they are unable to stabilize the relatively open structure of the diamond–cubic lattice. The interaction between the atoms in silicon has recently been modeled using a combination of pair (2-body) and triplet (3-body) interactions [7, 8], or a more general multi-body term [9].

General purpose supercomputers are not designed for efficient MD calculations, but for solving a wide range of problems. Furthermore, such computers typically achieve large peak speeds by using vector floating point hardware or by parallel processing architecture. In either case it has not been possible to achieve efficient use of the floating point processors, because of the difficulty of writing compilers that can optimize general code for complex architectures. In most cases only a small fraction of the theoretical speed is utilized. As the trend moves towards parallel architectures with many processors, optimization may become increasingly difficult, although there has been some recent progress [10].

For applications that have straightforward algorithms and that require large amounts of computer power, one alternative is to design the hardware to suit the algorithm [11, 12]. If an algorithm is efficient for a particular architecture, it is possible to achieve high speeds at relatively low cost. Fast scalar floating point chips are now available at low cost; a $400 Weitek chip can perform up to three million floating point operations per second (3 Mflops). We have designed a processor ATOMS with a parallel architecture that can keep the floating point processors busy much of the time when running our MD code. In this sense, it is an algorithm-oriented processor. It is, however, fully programmable and is also efficient for other calculations. Because of the simplicity of the shared memory architecture, it was possible to construct and program the processor in an acceptable time period using available microprocessors, floating point chips, memory elements, and other components.

In the next section, we describe the amount of computer power needed for MD simulations, which provided the motivation for this project. Section 3 is a brief summary of the MD technique using 2- and 3-body forces. Section 4 contains an overview of architecture of ATOMS, and a more detailed description is given in Section 5. In Section 6 we discuss the specific computing requirements for our implementation of the MD technique, with some of the timing considerations that went into the design of our current version for ATOMS. Also included is a brief discussion of the efficiency of ATOMS for finite element calculations of fluid flow problems. Section 7 is a summary of our experience to date working with ATOMS.

2. Computer Requirements

Molecular dynamics methods have been applied to models for the study of the structure of liquids and radiation damage for over 30 years [13]. Initially they were

limited to problems with short time scales and were practiced only at locations with large computer facilities. Later, increases in computer power made it possible to perform useful calculations on dedicated minicomputers, and many more investigators became involved. A wide variety of systems has now been studied, including diatomic molecular liquids, water, hydrocarbons [14], large molecules, e.g. proteins [15], and solid interfaces and surfaces.

In all of these applications, the shortest relevant time scale for atomic motion is ν^{-1}, where ν is a vibrational frequency for an atom in a potential energy well. The time step Δt used in the integration scheme must be small, such that $\nu \Delta t \ll 1$, and typically $\Delta t \approx 0.05\nu^{-1}$. The simulation of several thousand atoms for a time longer than a few vibrational periods clearly requires a large number of floating point operations.

The actual time required for a simulation depends on the type of simulation being performed, the properties of the system, and the rates of any dynamical processes involved. Molecular statics simulations are usually the least computer intensive, since they involve the relaxation of an initial configuration of atoms to a local minimum in the potential energy. These calculations provide information on the zero Kelvin structures, and have been used to test out interatomic potentials on surface reconstructions and gain boundaries, to calculate the energy of defects (e.g., dislocations) and to look at the structure of a protein molecule. Since a unique point in configuration space is calculated, the energy and other properties can be obtained to high accuracy without the statistical uncertainty inherent in finite temperature simulations.

The ability to generate a number of configurations quickly is crucial to molecular statics investigations, since often the ground state must be selected from a number of relaxed states. As an example, the equilibrium density of misfit dislocations in a thin epitaxial film can be calculated by preparing a number of initial states with different dislocation arrays [16]. The initial configurations are allowed to relax with dissipative forces and the chemical potentials of the resulting states are calculated from the potential energies. A configuration with 500 atoms interacting with Stillinger–Weber forces must be simulated for about 10^4 time steps to approach the minimum potential energy. For a crystal at finite temperature, the four nearest neighbors and approximately four of the 12 next-nearest neighbors interact with the central atom, and the calculation takes about 1.5 days on a computer in the one Mflop range, such as the VAX 780; 4.5 h on an Alliant FX4; 24 min on a Cray XMP (one processor), and 18 min on ATOMS configured with eight processor boards. Therefore, the VAX 780 or Alliant must be operated in batch mode, while it is clearly feasible to do rapid exploratory work on small systems with ATOMS, and also with the Cray, if it is not being shared too heavily. We note that the times given for the general purpose computers were measured using existing Fortran code written by us and which we have used at some stage for production MD calculations. Different algorithms were used in some cases in order to more effectively utilize the computer capabilities, but we do not claim that this code is beyond improvement, of course. Furthermore, all calculations on the

Cray and VAX were made in double precision, whereas most of the force calculations on ATOMS were made in single precision. Single precision is adequate for most MD simulations.

Equilibrium MD simulations are used to study phase transitions, diffusion of atoms in liquids and on surfaces, and general equilibrium properties of materials at finite temperatures.These calculations can require sizable computer resources. First, it is essential to equilibrate the system for a time greater than a correlation period; i.e., the time required for the system to assume a structure that is only weakly correlated to that of the initial state. This time depends on the system and the parameter that is being measured, but for viscous liquids or those with boundaries between coexisting phases, it can be quite long. In some cases it is necessary to test several different initial configurations to ensure that the system is not trapped in a region of phase space with a local minimum in the free energy. Second, the measurement of time-averaged properties usually includes a large statistical uncertainty. Often one is tempted to assign physical reality to changes in the properties which are, in fact, large statistical fluctuations. The time required to obtain reliable averages is considerably longer than the correlation period.

Finally, non-equilibrium MD is used for studying transport coefficients, the dynamics of a system subjected to a sudden change in conditions, or the structures and mechanisms that occur during the processing of materials. In practice, most MD simulations are limited to extremely short observation times, when measured in terms of the simulated physical system. Models with 1000 or more particles are usually simulated for less than a nanosecond (1 ns $\approx$ 500,000 time steps). Many physical processes are simply too slow to permit direct MD simulations. For example, the relaxation of glasses, crystallization of liquids, dislocation climb, and diffusive aggregation of defects often do not advance appreciably in a nanosecond. One interesting process that does occur on this time-scale is the solidification of silicon, which has been measured experimentally to have crystal-melt interface speeds up to 15m/s. At this speed the interface moves 15Å in a nanosecond, and this is the minimum time required to obtain useful information. However, most processes involving crystal surfaces are limited by the rate of surface mass transport. The surface diffusion coefficient D_s has a maximum value near the melting point approximately equal to the diffusion coefficient in the melt; i.e., $D_s < 10^{-4}\text{cm}^2/\text{s}$. A simulation time of 0.1 ns would be required for one atom to diffuse an average distance of 5Å. Simulation of molecular beam epitaxy with, for example, a 7×7 reconstruction on a (111) face of silicon would require even longer runs, and it would perhaps be necessary to run a system with 2000 atoms for 10^6 time steps. This would provide information on the transformation of the 7×7 symmetry to that of the bulk as the surface layer is covered up by the new material arriving from the vapor. This one run would require nearly two years on a VAX 780, but only six days on an 8-board version of ATOMS. Thus, dedicated special-purpose computers operating in batch mode make it feasible to study new problems.

There are two additional considerations that go into simulating all of the applications discussed above. First, an iterative procedure is often required. For example,

to determine a transition temperature for a phase change, a simulation is initiated at the estimated temperature, and, after the run is completed the data are evaluated and used to make a new estimate for the next run. This process is repeated until the temperature is determined to the desired accuracy. Second, most MD systems are quite small, and the size dependence of the results should be measured. Again, this requires additional simulations of different systems under identical conditions. Clearly the turnaround time between runs is the limiting factor in assessing the behavior of the system.

A major goal of this project was to construct a computer with sufficient computational power that short molecular dynamics jobs and molecular statics computations could be carried out interactively, with the uninterrupted attention of the investigator. This requires enough power to simulate atomic motion for several nanosecnds per day. Currently an eight processor system running for one day can simulate a nanosecond for a configuration of 500 atoms with eight interacting neighbors, assuming a relatively simple potential such as that of Stillinger and Weber.

3. Algorithm

We will now discuss an efficient algorithm for molecular dynamics simulations when the interatomic potentials involve both 2- and 3-body interactions. The input data consist of initial positions and velocities of the atoms and a set of interatomic potentials. The MD algorithm then involves updating positions, calculating new forces, updating momenta; and, typically, the gathering of statistics (e.g., thermodynamic averages and correlation functions).

The two issues that must be addressed for each part of the algorithm are (i) the number of calculations and (ii) the amount of memory that is required. The number of calculations required to update positions and momenta scales with N, the number of atoms in the system. In principle, the force calculation scales as N^2 when only 2-body interactions are included, and as N^3 when 3-body interactions are present. However, this number is much smaller in the case of potentials that are of short range, and can be truncated after a distance r_c of several atomic radii.

Several techniques are used to reduce the required number of calculations for potentials of short range. The linked list method [17] speeds the force calculation on atom i, since it provides lists of atoms occupying cells that are subdivisions of the computational box. This results in a force calculation scheme of order N, independent of the type of interactions that are included. When memory space allows, an approximate pair list may also be calculated. This provides a more precise list of neighboring atoms than that obtained with the linked list. Our MD code includes both techniques. Associated with each atom is a list of neighbors that were within a distance r_m at the time it was compiled. The magnitude of r_m is chosen to be somewhat larger than the force cutoff, r_c. To facilitate this calculation, linked-list cells are used. The edge length of a cell is chosen to be greater than or

equal to r_m. Neighbors in the same cell and in the surrounding 26 cells are tested for inclusion in the approximate pair list. The approximate pair list is used for ten time steps, and r_m is chosen to be larger than r_c by a sufficiently large margin that atoms beyond the larger sphere will not drift into the smaller one during this period. The memory space required for such a list scales with Nn, where n is the average number of neighboring atoms in the pair list.

We now consider the force scheme when 3-body interactions are included. The energy of the system can be expressed as:

$$E = \sum_{ij;\, i<j} V^{(2)}(r_{ij}) + \sum_{ijk;\, i<j<k} V^{(3)}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k), \tag{1}$$

where $V^{(2)}$ is the 2-body interaction and r_{ij} is the distance between atoms i and j. $V^{(3)}$ is the 3-body interaction which for the Stillinger–Weber and Biswas–Hamann potentials can be written as:

$$V^{(3)}(r_i, r_j, r_k) = h(r_{ij}, r_{ik}, \theta_{jik}) + h(r_{ji}, r_{jk}, \theta_{ijk}) + h(r_{ki}, r_{kj}, \theta_{ikj}), \tag{2}$$

where θ_{jik} is the included angle between atoms j, i, k. The first term of this sum, $h(r_{ij}, r_{ik}, \theta_{jik})$, involves atom i at the apex of a triangle, with legs r_{ij} and r_{ik}. This term will contribute a force $F_j = -\partial h/\partial \mathbf{r}_j$ to atom j and a force $F_k = -\partial h/\partial \mathbf{r}_k$ to atom k, as well as a force $F_i = -\partial h/\partial \mathbf{r}_i$ to atom i. Since the sum of the forces must be 0, the force on atom i is usually evaluated as $F_i = -(F_j + F_k)$. Each 3-body term in the potential of (2) is a product of three terms,

$$h(r_{ij}, r_{ik}, \theta_{jik}) = f(r_{ij}) f(r_{ik})\, g(\cos \theta_{jik}), \tag{3}$$

in the Stillinger–Weber and Biswas–Hamann potentials. This feature simplifies the use of lookup tables for the potentials and makes possible the very efficient MD algorithm discussed below.

The algorithm for calculating forces should be considered carefully. In the case of triplet forces satisfying Eq. (3), an efficient method can reduce computation time by more than an order of magnitude compared to a simpler approach! There are two commonly used algorithms for this calculation. The method used most frequently is to calculate the total pair force on all atoms, and then the total force due to 3-body interactions. The triplet calculation does not make use of any information generated for the pairs, and the triplet terms in Eq. (3) are calculated from scatch for each triplet. This can be extremely time consuming, since each triplet requires two calculations of the exponential in $f(r_{ij})$. The second option is to consider each atom in turn and first calculate all pairs with the given atom and the associated forces and then all triplet forces with the specified atom at the apex. During the pair calculation, the $f(r_{ij})$ are also computed and saved. Only the pairs with $r_{ij} < r_c$ are passed on to be used in the identification of the triplets. This approach avoids the necessity for maintaining a complete triplet list with the

associated memory allocation and does not repeat calculations of r_{ij} and other parameters for the 3-body forces that have already been performed for the pair forces. This approach also eliminates the need for recalculating $f(r_{ij})$ for each new triplet that contains the ij pair. Since the number of triplets containing a given pair scales with n, this can have a very large effect on the efficiency. This scheme also generalizes for the Tersoff potential [9], and for any scheme where the n-body terms are simply combinations of pair terms. The MD code for ATOMS implements the second approach.

4. General Architecture

The algorithm described in Section 3 maps very well onto a parallel computer architecture, and motivated the design and construction of ATOMS. There are two key features of the algorithm. First, the time required for each part of the program scales as N (updating positions, calculating forces, and bookkeeping). Second, the most time consuming part (97%) of the algorithm is the force calculation and the associated bookkeeping. Therefore, we have concentrated on the implementation of the force calculation to determine an effective computer architecture.

There are many ways to perform independent force calculations in a multiprocessor system. They differ mainly in the method of distributing the data for processing. In a shared memory machine a current and complete set of data is kept in the shared memory, and data are sent to each board in turn, along with a data-manipulation task. Inter-processor communication is not required since each processor board accesses the data directly. In a distributed memory machine, where a some of the data are kept in local memory on each processor, inter-processor communication may limit the efficiency. It can be effective when applied to local problems that can be programmed in such a way that a processor needs to communicate only with a small group of neighboring units. However, the programming of the communications can be quite difficult.

A shared memory architecture has the restriction that the number of processors is limited by the bandwidth of the common data bus, since all processors must access this memory. For a given number of processors, it is most efficient when the algorithm involves a large number of calculations based on a relatively small data

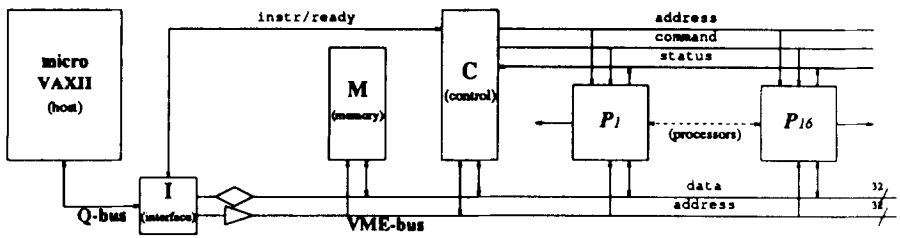
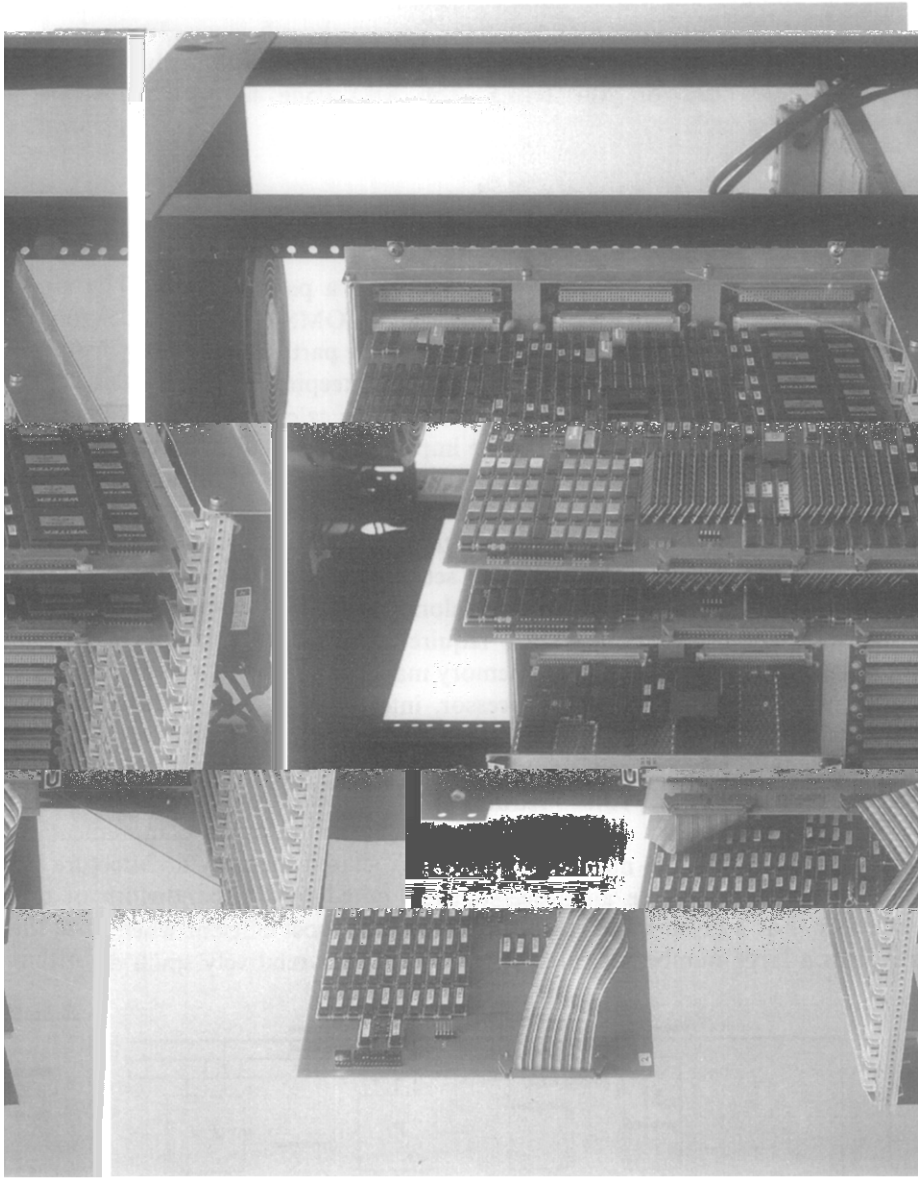


FIG. 1. Basic components of the shared memory machine, ATOMS.



set. The number of the data required for a 3-body force calculation scales as the number of neighbors n, and the number of calculations scales as n^2. For sufficiently large n, the shared memory architecture is a good solution. We find that the shared memory architecture of ATOMS is efficient for the Stillinger–Weber potential, and therefore it should be effective for most other potentials that have been developed for semiconductor systems since they involve more complex n-body calculations.

ATOMS is a multiple instruction, multiple data (MIMD) parallel computer. The basic components are shown in Fig. 1. All boards in ATOMS are interconnected through a single VME-bus, which is interfaced to the Q-bus of the host, a MicroVAX II. The system consists of an interface (I) board, one or more memory (M) boards, one control (C) board, and many identical processor (P) boards. The backplane can accommodate up to 16 boards.

The M board is the shared memory. The VME-bus is the only data channel between all boards and is controlled either by the C board or by the host through the I board. The host controls the VME-bus to download microcode for the C and P boards and accesses the M board to transfer data. The C board controls the VME-bus to transfer data between the M board and the C and P boards. As indicated in Fig. 1, there are also connections for transferring data between adjacent

The P boards perform the floating point operations. There is a second level of parallelism in the architecture of ATOMS, since each P board contains four identical floating point chip sets. However, these chip sets are controlled by one instruction flow, so this part of the architecture is single instruction, multiple data (SIMD). Each chip set contains at least one arithmetic logic unit and one multiplier, and the instruction flow is designed to keep both chips working at the same time.

A picture of the hardware with two P boards is shown in Fig. 2. The processor is mounted in a standard 19 inch rack, and the P boards are $9U \times 40$cm Euroboards.

In addition to our goal of constructing a fast, efficient processor suitable for the MD algorithm, we had some additional requirements. First, the floating point chips should have available both single and double precision arithmetic. This provides the flexibility to simulate a variety of physical systems with different requirements on the precision of the results. Second, the floating point section should be relatively simple to program, since small changes to the code are often desired (e.g., different integration scheme, different form for the interatomic potentials, or a new correlation function).

FIG. 2. Photograph of 2-board machine. Boards visible here are, from left, the C board, the I board (edge visible with two cables connected to it), the M board, and the two P boards. The Weitek floating point chips are in the bottom row on the P board, the dual port SM chips are in the row above, the microprocessor XP is the square chip near the center, and the LM printed circuit boards are to the left of center and bent at an angle to the P board.

5. Detailed Architecture

5.1. *M Board*

The shared memory M is a commercially available dynamic ram board for VME-bus applications, and can be obtained in 4, 8, and 16 Mbyte sizes. Multiple M boards can also be accommodated to extend the capacity further, with each occupying one of the 21 slots on the backplane.

5.2. *C Board*

The C board is a custom multiwire board, with commercially available chips. It has its own microprogrammable control, with a 100 ns cycle time. The C board contains a 64k × 32 bit fast static memory XM and a 16-bit processor XP, the Advanced Micro Devices AM29117. They are interconnected by a 16-bit address channel, the A-bus, and a 32 bit data channel, the X-bus, as shown in the diagram in Fig. 3.

XM is addressed by a 16-bit address counter/register *xmac*, which can be loaded by the A-bus and incremented or decremented. A bidirectional latch *xml* interfaces the 32-bit data with the X-bus. One cycle is required for a read or write operation, including the transfer over the X-bus to its destination.

The external memory M on the VME-bus is addressed by the C board. This is accomplished by a 24-bit address counter/register *lmac*, which can be loaded by the A-bus in two portions, and incremented or decremented. The address is fed to the VME address bus via address register *lmar*. Three cycles are required for a read or write operation including the transfer over the VME-bus.

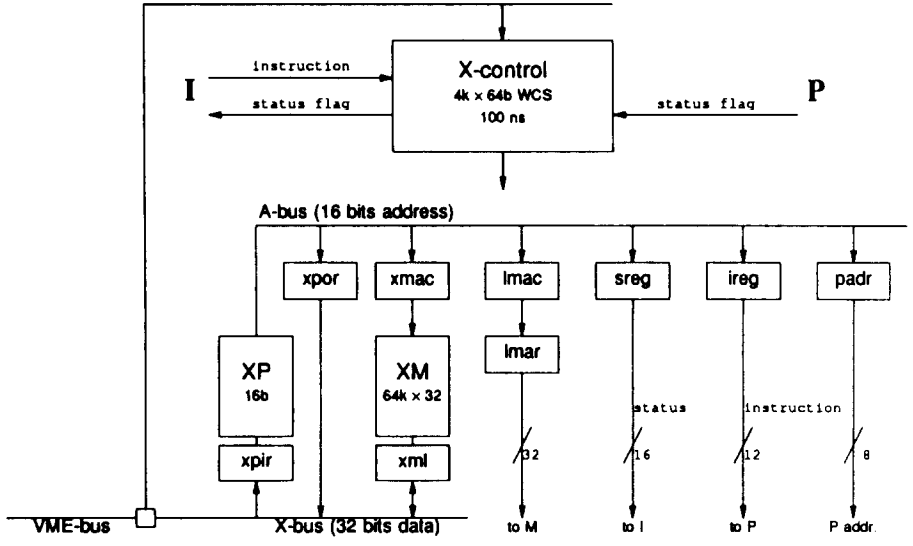


FIG. 3. Components of the control board (C).

The XP microprocessor is used for address generation, loop counting, and other operations involving integer arithmetic. The X-bus supplies XP with input data via its 32-bit input register xpir. The most or least significant part of xpir can be selected as input for the processor XP. The output of XP is the A-bus, which feeds xmac and lmac and a 32-bit output register xpor. This register can be loaded in two 16-bit portions, and its destination is the X-bus.

Additional A-bus destination registers are present: sreg supplies I with status, ireg supplies P with microprogram start addresses, and padr selects each P board based on its unique board address. When a P board is selected it can accept an instruction or is able to use the VME-bus for data transfer under control of the C board.

A microprogrammable instruction flow runs the entire C board. A $4k \times 64$ bit writable control store (WCS) contains the microprogram, which is downloaded from the host. An Advanced Micro Devices AM2910 sequencer is used to generate the microprogram addresses, the contents of which are stored in a microprogram instruction register. An instruction stream is fed to the C board at 10 MHz; this allows nested subroutines, conditional jumps, and other powerful commands.

5.3. *P board*

The P board consists of two sections, an X section that is very similar to the C board, and a floating point section F. The X and F sections each have their own microprogrammable control, as shown in Fig. 4. X has a 100 ns cycle time, but the floating point part F has a 50 ns cycle time. X stores and controls the transfer of data whereas F executes the floating point operations.

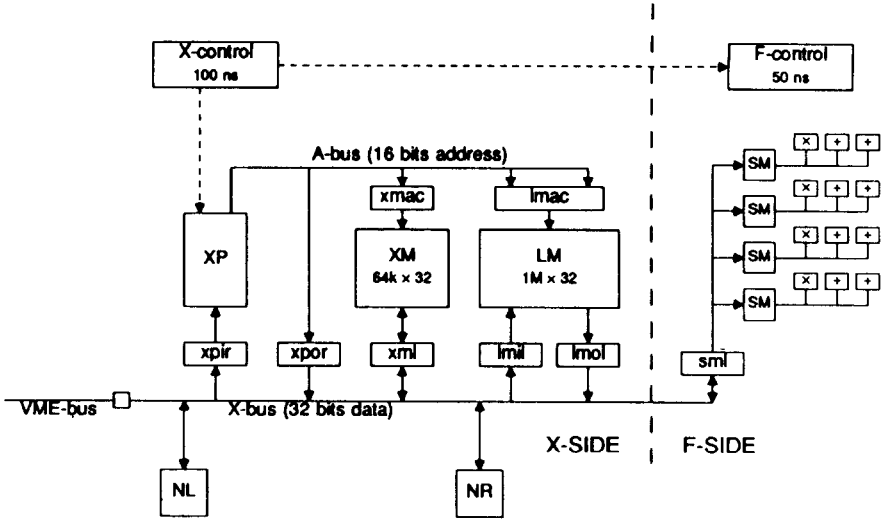


FIG. 4. Components of the professor board (P).

We first consider the X part of the P board. As in the C board, the data path of X contains a 64k × 32 bit fast static memory XM and a 16-bit processor XP, the AM29117. A major difference is the presence of a 1M × 32-bit dynamic memory LM. This memory is composed of 16 printed circuit boards with nine 256K-bit chips on each board. We have also tested the circuit with 1M-bit chips, which increase the capacity of LM to 4M × 32 bits. Access to LM is similar to that of the M dynamic memory from the C board. XP, XM, and LM are interconnected by the 16-bit address channel A-bus and a 32-bit data channel, the X-bus.

As in the C board, XP is supplied with input data from the X-bus by way of xpir, and the output is the A-bus. The XM memory is addressed by xmac and is interfaced to the X-bus by xml in exactly the same way as described for the C board. Similarly, xpor is used to transfer output from XP to the X-bus.

LM is addressed by a 22-bit adress counter/register lmac, which can be loaded by the A-bus in two portions and incremented or decremented. Latches lmil and lmol interface the 32-bit data with the X-bus. Three cycles are required for a read or write operation, including the transfer over the X-bus.

Other connections to the X-bus are the interface section for the VME-bus (not shown), and a bidirectional latch sml to the F part. Also included is interface hardware for direct data transfer with two neighboring P boards; NL and NR are bidirectional latches with connectors for cables going to the "left" and "right" P boards, respectively. This hardware provides the option for running closely coupled processors in a linear array. This feature has been useful in finite difference applications, where extensive data transfer between boards is essential.

The microprogrammable instruction flow for the X side is identical to that of the C board. A $4k \times 64$ bit writable control store (WCS) contains the microprogram, which is downloaded from the host. The C board supplies X with instructions in the form of microprogram start addresses, so that C determines which instruction flow will be executed.

We now consider the F part of the P board, shown on the right-hand side of Fig. 4. The F part consists of four identical floating point chip sets. The interface of each of these chip sets to the X section is provided by dual port memories labeled SM in Fig. 4. These memories are 32 × 32 bit register chips (Weitek 1066), two in each chip set. Each chip set contains a 32-bit bus which is connected to the three floating point chips. One of the floating point chips is an arithmetic logic unit (ALU), which performs additions, subtractions, divisions, and certain other operations (Weitek WTL1165) and the second chip is a multiplier (WTL1164). The ALU and multiplier operate at 20 MHz, and 7 cycles are required to perform a single precision multiply or add, for 2.86 Mflop each. It is possible for the ALU and multiplier to perform operations concurrently, as long as the result from the ALU is not needed for the next multiply operation. A socket for a third chip is included; it can accommodate either an ALU or a multiplier, or remain empty. This feature was included so that the hardware can be easily adjusted for code which has an unbalanced number of ALU operations relative to multiply operations. Including a second ALU was found to increase the speed of the MD code with the

Stillinger–Weber potential by about 10%, but these extra chips increase the cost of a P board by about 16%.

The two SM memories in each chip set are used in a "ping-pong" mode. The thick lines in Fig. 5 indicate active data paths at one instant in time. The X side (XM in this case) is transferring data to SM-A of chip set 2, while all four chip sets are performing calculations on data in SM-B. Therefore, data transfer with X can proceed in parallel with the floating point calculations.

Although the four chip sets of the F section are generally executing the same instruction in SIMD, there is the capability of modifying the instruction that each set is executing. Each of the four sets has four addressable two bit registers that can be used for op-code modification, so that the actual op-code that one set executes can be add, subtract, or copy depending on the code in its register. The address of the register is the same in all four floating point chip sets. This is used for handling periodic boundary conditions in the MD code. Each pair in the approximate pair list has an associated eight-bit function code that determines whether the interaction is direct or is the result of a periodic repetition of the system in the x, y, and/or z direction. When the pair parameters are processed in a floating point chip set, the two-bit registers are loaded from SM with the op-code associated with that pair. The calculation of the difference in the x coordinates, for example, incorporates this feature to add or subtract the length of the computational box in that direction, if necessary.

Each floating point set has a five-bit status register associated with it to report any floating point exceptions and the result of comparisons. The exception bit is set

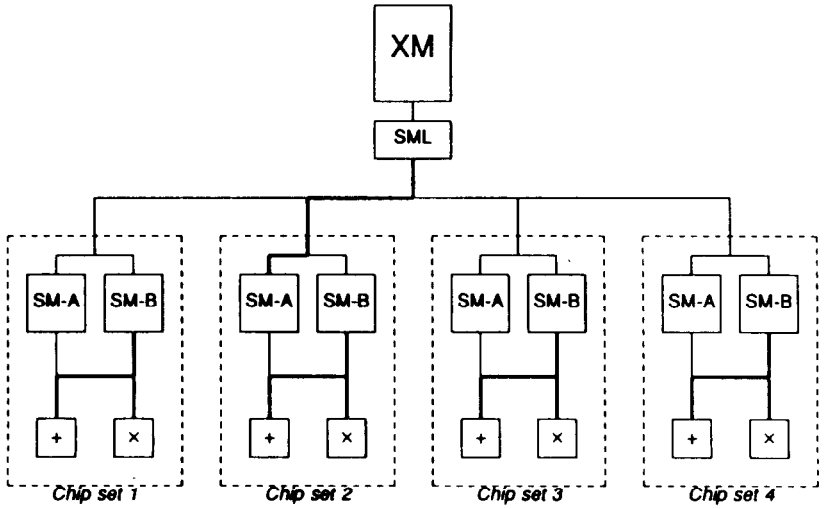


FIG. 5. Data paths between XM and the F side of the P board. In ping-pong mode, transfer data from the X side to one of the SM-A, while floating point chips are performing calculations on data in all SM-B chips.

when one or more exceptions occurs during operations that are specified in the code. It can be read by the X side of the P board, and it is cleared by a separate command. Three bits are addressable and can be used to report the result of comparisons to the X side. The fifth bit is a flag internal to the F side, and can be used to change a write to SM-A or SM-B into a "no-op." This does not change the timing of one floating point set relative to the others and permits certain steps in the code to be modified if, for example, the distance r_{ij} exceeds the cutoff r_c.

There is a separate microprogrammable instruction flow for the F section. A $4k \times 32$-bit WCS contains the microprogram, which can be downloaded from the host. Here a binary counter is used to generate the microprogram addresses in order to guarantee an instruction stream at 20 MHz. Microprogram start addresses are received from X.

The ability to have the ALU and multiplier (and optional third chip) of each chip set running concurrently is implemented by an optimizing assembler. The assembler takes the user-written code, a sequence of additions, multiplications, and divisions, and schedules the floating point chips for the seven cycles of each operation. The seven cycles are used for loading the chip with two operands from SM, performing the operation, and unloading the result to SM. As shown in Fig. 5, there is only one data path from SM to the two (or three) floating point chips, so only one chip can be loaded from SM at any time. The assembler allows an arbitrary mix of ALU and multiplier chips, so different combinations of chips can be tested with the code.

5.4. *Other Boards*

The host interface I consists of two boards, one at the host side and one at the ATOMS side, and they are interconnected by a flat cable. Its primary function is to connect the ATOMS VME-bus to the host system Q-bus. A separate instruction/status bus directly connects the I board to the C board to provide a means for the host to supervise ATOMS. In addition, a four-bit code from the host determines the connection of the host bus with the VME bus (on or off), and the mode of ATOMS for download (reset or start).

Finally, we note that all memory units in the machine are accessed by two different sections, as though they were dual ported. Particle data is put into the shared memory M by the host computer. This data is transferred by a program running in the C board to the different P boards and stored in XM. From XM the particle coordinates are transferred to the four chip sets by means of the dual-ported registers SM-A and SM-B. These registers are then used to supply the floating point chips, where the calculations are processed.

6. Current Applications

6.1. *Molecular Dynamics*

We have programmed the MD algorithm described in Section 3 for atoms interacting via the Stillinger–Weber type potential. We can currently handle configura-

size and/or strength of interaction between different types of atoms). The host MicroVAX compiles the MD code and downloads the code for the C board and for the X and F sections of the P boards. Identical code is distributed to all P boards. The force lookup tables are distributed to the LM memories, and an initial configuration is loaded into the M board. Execution is begun when an instruction start address is sent to the C board. After this point ATOMS is essentially autonomous, although it can be instructed to periodically transfer intermediate results to the host. This is accomplished by having the C board send a flag to the host indicating that it is can take control of the VME-bus in order to access the M board, where the results are stored. The host returns control to the C board when it has finished reading the data.

The MD code distributes the force calculation among the P boards on an atom by atom basis, as shown in Fig. 6. The P boards are polled sequentially by the C board to collect results and distribute new data for processing. The C board transfers the coordinates of a central atom and its n neighbors (listed in the approximate pair list) from the shared memory, M, to one of the P boards. If the P board has results from a previous central atom, the forces are accumulated in the M board for all affected atoms. If the system is operating efficiently, the C board will select a P board for data transfer before it has completed its calculation. That is, no time is lost because the P board must wait for a new set of data. In order to achieve this condition, the data transfer time with N_P P boards must be shorter than the time t_{calc} for one P board to transfer its data and perform its calculations; i.e.,

$$N_P t_{trans} < t_{calc} + t_{trans}, \tag{4}$$

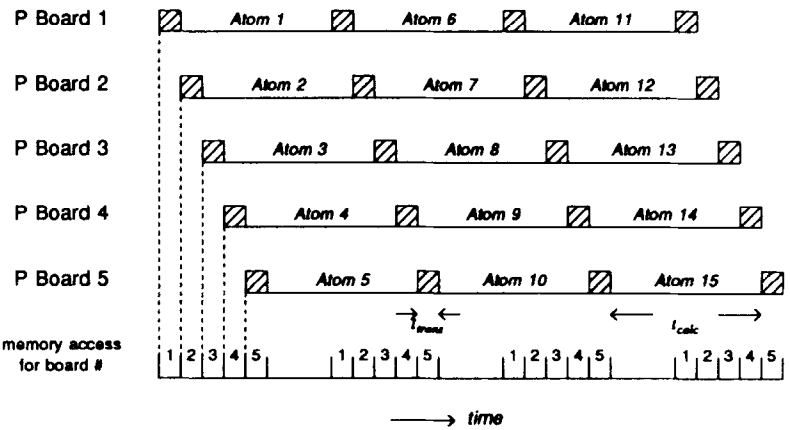


FIG. 6. Activity as a function of time for force calculation of 20 atoms for a 5-board machine; top row is activity of board 1, next row board 2, etc., and bottom row is the activity of the M board.

where t_{trans} is the data transfer time for one board. Furthermore, since the number of neighbors is larger for some atoms than for others, some P boards may finish before others that were selected earlier in the sequence, which can cause a delay for these boards. This is most pronounced for inhomogeneous systems, although even in this case the fraction of atoms with low n values is usually small, and the reduction in efficiency is not important, as will be demonstrated below. If this were a problem, a more sophisticated polling scheme could be implemented by a software change for the C board.

The force calculation is performed in two phases. First, force parameters for all pairs with the central atom are calculated, including those that will be needed for the 3-body calculation. The 3-body parameters are stored in XM for later use. The functions that determine the 2- and 3-body forces reside in LM as lookup tables which are evaluated by quadratic interpolation. An important consideration with such a powerful floating point section is that of supplying data fast enough to keep the chips busy with calculations. The dual ported memories SM-A and SM-B provide a fast channel between the floating point chips and the XM memory, but it should be noted that the microprocessor XP must supervise the transfer of this data and can only load one SM memory at a time. Because of the necessity for numerous data transfers with both XM and the slower LM, the calculation of the pair parameters does not utilize the floating point chips efficiently. However, the triplet calculation can then be easily arranged. All parameters in the 3-body forces are associated with pairs, as discussed in Section 3, and the forces are obtained from all unique combinations of pairs taken two at a time. (Since all pairs include the central atom, two of these pairs constitute a triplet.) The large number of computations necessary for the 3-body forces can be accomplished efficiently using more than one chip set, since the SM memories can be loaded with the necessary data directly from XM, as shown in Fig. 5.

The initial design of ATOMS was based on the characteristics of the 3-body force calculation, since this involves the largest number of floating point operations. That is, N_P and the number of chip sets per board N_C were chosen so as to optimize this part. An increase V_P and N_C would increase the theoretical speed, but this is cost-effective only if the data transfer is fast enough to keep the floating point chips supplied. In the following paragraphs we discuss the estimates that went into choosing the values of N_C and N_P to implement in hardware, and we also discuss how this has worked out in practice. For studying thin film systems, e.g., Ge films on Si substrates, atoms in the strained film have 16 neighbors completely within the cutoff, r_c. The timing estimates given below are for a perfect crystal with $n = 16$.

Consider the 3-body force calculation for one triplet with the central atom i; i.e., the forces resulting from one of the terms $h(r_{ij}, r_{ik}, \theta_{jik})$ in Eq. (2). This calculation is performed in one of the N_C chip sets. The following data associated with pair ij must be transferred from XM to SM: r_{ij}^{-1}, dx_{ij}/r_{ij}, dy_{ij}/r_{ij}, dz_{ij}/r_{ij}, $f(r_{ij})$, a force term based on $f(r_{ij})$, and the three components of the force on atom j that were accumulated in previous calculations. Thus, each pair has nine input parameters associated with it, for a total of 18. The results of the calculation are the potential

energy h of the triplet and the new totals of the force components on all three atoms. Only the forces on atoms j and k are written to XM; since the central atom is involved in all triplets, its force components and the potential energy h are accumulated in SM. This is six output parameters, or a total of 24 data transfers between SM and XM. The time for these transfers is estimated as one cycle each, i.e., $t = 24\text{cycles}/10\ \text{MHz} = 2.4\mu\text{s}$. Since the chip sets must be loaded sequentially, the transfer of data for four chip sets takes a total of $9.6\mu\text{s}$. In fact, the data transfer is slower than the above estimate because of the overhead required for XP to calculate the memory addresses. The activities of the SM memories during the triplet calculations are illustrated in Fig. 7.

Once the data is in the SM memory, the calculations for the forces and potential energy requires 25 additions and 25 multiplications for the Stillinger–Weber model. The optimum calculation time is therefore $t = 25/2.86\text{Mflop} = 8.74\mu\text{s}$, assuming that the two operations can run concurrently. Although both estimates are on the low side, the data transfer and calculations should be approximately equal for $N_C = 4$, and increasing N_C beyond this point would result in a data transfer bottleneck. That is, there is almost enough time to unload results and load new data with the SM-A memories of the four chip sets while the floating point chips are performing the calculations on data in the SM-B memories. The total time for performing one triplet calculation is the larger of the two times; in the case of our estimates, it is the triplet transfer time of $t_t = 9.6\mu\text{s}$.

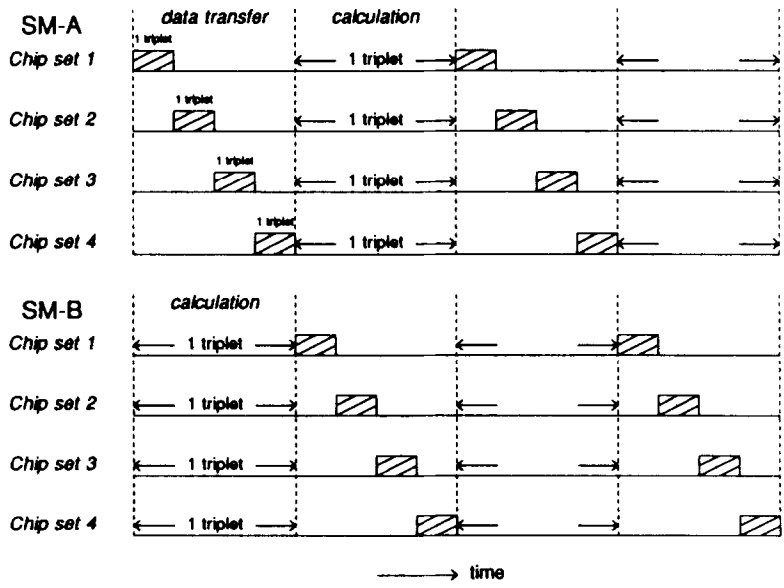


FIG. 7. SM activity as a function of time for triplet calculations. Here the data transfer for four chip sets can take place during the calculation time.

The time required to generate the pair forces on the central atom is the sum of the data transfer and calculation times, since in this case we did not distribute the calculation over the chip sets and only SM-A was used. Twelve parameters are needed from LM (force and potential energy values for both 2- and 3-body interactions, three numbers are required for a quadratic interpolation of each); each LM to SM transfer takes four cycles. The XM memory is accessed for the six coordinates, and eight results are written to XM at the end. The calculation requires 69 multiplications, and somewhat fewer additions. The total time to generate the results for one pair is $t_p = 69/2.86\ \text{Mflop} + 62/10\ \text{MHz} = 30.3\ \mu\text{s}$.

The total time required for the P board to process a central atom is $t_{\text{calc}} \approx nt_p + n(n-1)\, t_t/2N_C = 773\ \mu\text{s}$, where we have assumed that $n = 16$ and $N_C = 4$. The time to load and unload the board can also be estimated. The coordinates of the central atom and its neighbors in the approximate pair list are transferred from M to the P board at the start of the calculation. Each transfer takes four cycles at 10 MHz, and there are $3(n+1)$ transfers, so the input stage takes 20.4 μs. At the end of the force calculation the M board is accessed to read current values for the energy, the force on the central atom, and the forces on the n neighbors; these are incremented by the values in the XM memory and the updated values are written back to the M board. This involves $m_{\text{end}} = 3(n+1) + 1$ transfers from the M board to SM, m_{end} transfers from XM to SM, m_{end} additions, and m_{end} transfers from SM to the M board. The additions are performed in three chip sets, and the total time for read/modify/write is approximately 53 μs. Therefore the total transfer time for input and output is $t_{\text{trans}} = 73.4\ \mu\text{s}$. Thus, from Eq. (4), the total number of P boards that can be used without degrading performance is $N_P = (773 + 73.4)/73.4 = 11.5$.

We now discuss the actual speed achieved with the current version of the MD code. On one board, the theoretical speed is 22.8 Mflops, assuming all four chip sets are used, and both the ALU and multiplier are performing operations at all times (two operations every seven cycles). Because of data dependencies, the ALU and multiplier cannot perform concurrently for all of the 25 additions and 25 multiplications involved in the triplet calculation. The triplet code actually produces an average of one operation every 5.9 cycles, instead of one every 3.5 cycles for perfect concurrency. The time for one triplet calculation in a chip set is 14.75 μs. Thus, the maximum speed of the triplet calculation is 13.6 Mflops per board, assuming that all four chip sets are fully loaded with data at all times. The data transfer time for four triplet calculations varies from 16 to 34 μs, depending on whether the new triplets can use the data for one of the pairs already stored in SM. Both the data transfer time and the calculation time are somewhat larger than our estimates. The calculation of all triplets associated with a pair utilized 6.6 Mflops of the 13.6 Mflops possible. It is possible to significantly decrease the data transfer time and to approach closer to 13.6 Mflops limit, but it is shown below that other parts of the algorithm would severely limit the overall gain in speed.

The average speed for the entire force calculation is 4.7 Mflops per board. This is somewhat smaller than the triplet speed because of the necessity to access data

from the slower memory LM during the pair calculation. For one central atom, the force calculation, t_{calc} is 1.57ms and the data transfer time, t_{trans} is 0.20ms. The ratio of the total time a board is occupied with a central atom to the data transfer time, $1.77/0.20 = 8.85$, shows that we can keep nine boards busy for the force part of the calculation. To do the force subroutine on a one-board system, the data transfer and the calculation are done sequentially, so the actual speed we achieve is 4.1 Mflops. In Fig. 8, the crosses (+) indicate the speedup achieved for the force calculation as a function of the number of boards. The performance factor is normalized by the one-board system, where the data transfer between P and M occurs without delay. It agrees very well, with a 7.8-fold speedup with eight boards. Therefore, the 8-board system achieves a speed of 32.0 Mflops for the force part of the calculation. As mentioned above, we can reduce the time it takes to do the triplet calculation (currently limited by data transfer between XM and SM). However, with an 8-board system we would then become limited by data transfer between the M board and the P board, so unless we can decrease that time, we will not actually see a speedup.

We have also evaluated the overall processor speed versus N_P, and this data is represented by the squares (□) in Fig. 8. Because parts of the code, such as updating positions and velocities, have not been optimized to use more than one board, they start to take a larger fraction of the time, up to 15% for the 8-board system. Therefore, the actual speedup for the 8-board system is a factor of 6.8. The 8-board version of ATOMS is twice as fast as the Cray XMP for $n = 16$, and is 30% faster for $n = 8$.

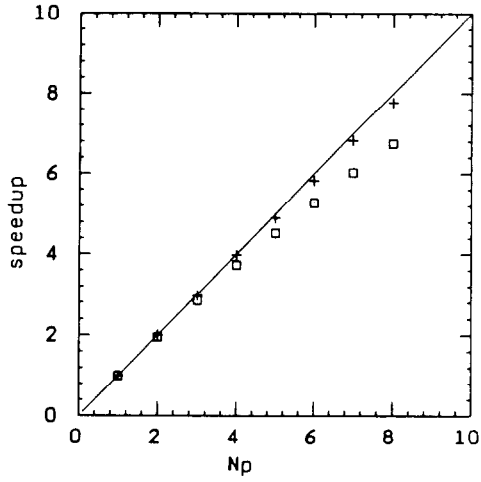


FIG. 8. Speedup as a function of N_P, the number of P boards (speedup ≡ time for MD calculation with one P board/time for MD calculation with N_P P boards). The squares (□) show the overall speed for the entire calculation, whereas the crosses (+) show the speed of the force calculation only.

6.2. *Finite Difference Equations for Fluid Flow*

A program to simulate convection and other phenomena in a fluid confined to a cavity has also been implemented on ATOMS. Fluid flow is induced by a non-uniform temperature imposed at the boundaries of the cavity. Studies of conditions leading to laminar or turbulent flow in two dimensions have been performed. Navier–Stokes equations were solved using a SIMPLE finite difference algorithm [18]. A 90×90 grid was employed in a simulation of a square cell. A 3-board ATOMS system was used for the calculations, with the P boards running as closely coupled processors (connected to their neighbors with NL and NR). The problem was divided among the processors by assigning each one a domain of the cell consisting of 30 rows of grid points.

The parameters describing the system properties at each grid point are: (i) components of the flow velocity u_x and u_y, (ii) the pressure p, (iii), the turbulent kinetic energy k, (iv) the energy dissipation ε, and (v) the enthalpy h. Current data for the three domains are maintained in LM on each processor board; the data in the M board are not updated at the end of each iteration, in contrast to the MD code discussed above. All calculations for the evolution of the system parameters at the points in a given domain are copied from LM to XM. New parameters for the center row of the three can then be calculated with transfers between XM and SM only. Parameters at the boundary adjacent to another domain are required for the calculations in that domain and are transferred to the corresponding P board using the NL and NR interfaces.

A typical example is the calculation of one of the four coefficients needed for a new value of u_x. This routine uses all four chip sets, operating in SIMD mode, and each one works with data from a different grid point. At the beginning, all four SM-A and SM-B memories are initialized with common parameters required for all grid point calculations. These parameters are stored in eight of the 32 locations in SM. Then the four SM-A memories are loaded with data from XM for the first four grid points in the row, the X side is switched to SM-B, and the floating point program is started on the job of processing the data in SM-A. Data for the next four grid points are loaded into SM-B at the same time that the floating point program is running. All calculations are performed in double precision. Eight parameters are sent to each of the SM-B memories, and they occupy 16 of the 32 locations. The transport of one parameter to each of the four SM-B memories requires 11 cycles (10 MHz clock), three cycles overhead at the start and eight more for the eight 32-bit words. Thus, the time to load all eight parameters is 8.8μs. The optimum calculation time for the 25 double precision operations performed in this example is 6.88μs (11 20-MHz cycles per operation, both chips working in parallel). At the end of the routine, three parameters are transferred from SM to XM, and a pointer to the location of a new set of data in XM is updated; this process requires 3×11 10-MHz cycles, or 3.3μs for all chip sets. Data transport between XM and SM probably limits the speed of the routine, since the optimum calculation time of 6.8μs is much less than the time required for transport and pointer

updating. The total time for the routine is expected to be 12.1μs per grid point. As shown by this example, the efficiency of utilization of the floating point chips depends critically on the amount of data transport and pointer manipulation involved in each routine. The efficiency ranges from 20 to 60% for the routines used in the finite difference algorithm. The overall performance of the program has an average efficiency of about 30%.

One iteration involves updating all 30×90 grid points on a given board. This includes 600 floating point operations for each grid point, or 1.62×10^6 operations for all 30 rows. Data transport to neighboring boards does not slow the processors significantly. Even if the P boards were to operate at 100% efficiency, the calculation of the 600 operations on all of the grid points in the domain would take $600 \times 90 \times 30 \times 11$ cycles/(8 processors) $\times 2 \times 10^7$Hz $= 0.11$s. The time to transfer data for a row of 90 grid points to the neighboring board is relatively small; ten double precision values per grid point gives 900 double precision words, each requiring 40 cycles at 10 MHz, or a total of 0.0036s. The time for transport to two adjacent boards is therefore less than 2% of the computation time. It should be noted, however, that this data transport cannot proceed in parallel with other operations on the P board, and this time is added to that for the computation.

7. Conclusions

We have designed and constructed an algorithm-oriented parallel processor for molecular dynamics calculations that is based on a shared memory architecture. We have implemented a system that operates as a MIMD parallel processor with up to 16 P boards. In addition, each P board has four floating point chip sets which operate in a SIMD mode. Each chip set includes a 2.86 Mflop adder and multiplier and two dual-port registers for data transfer between the local memory on the P board and the floating point chips.

The current MD code for multicomponent systems uses all four chip sets for the computationally intensive triplet calculation of the Stillinger–Weber potential, and a single board is 50% faster than an Alliant FX4 for the same calculation. The 8-board system is more than six times the speed of a single P board and is faster than the Cray XMP for the same problem. The processor will be even more efficient for 4-body and higher interactions. With 4-body interactions the number of calculations scales as n^3 but data transfer scales as n, so higher order interactions can keep more P boards busy without bus contention. By similar reasoning, it is also more efficient for long-range interactions.

In addition to using ATOMS as a shared memeory machine, we have taken advantage of the direct linear connections between P boards for finite difference calculations. We have found that the machine is efficient for these types of problems, provided that appropriate "local" algorithms are employed.

The total cost of a system with ten P boards is $115,000. This includes $18,000

for the MicroVAX II host, $9000 for each P board, $3000 for C and I boards, $2000 for the card cage, bus connectors, and fan rack, and $2000 for a four Mbyte M board. We have included the cost of manufacturing the multiwire P and C boards, but the labor-intensive socket and chip insertion and debugging operations were performed by us and are not included. The cost of the P boards is based on the inclusion of one WTL1164 and one WTL1165 chip per chip set, and $4300 of this is associated with these chips together with the WTL1066 memories.

We have constructed and tested 21P boards and four C boards. We operate two "production" systems with eight boards each and running 24h a day; and use two smaller systems for finite element calculations and testing. This is one of the most powerful systems available for molecular dynamics investigations of materials. The production systems are currently being used to study silicon solidification and nucleation, silicon/germanium epitaxy, and related problems. We have repeatedly demonstrated the effectiveness of ATOMS for interactive investigations of small systems and for studying large systems and slow processes in the batch mode.

ACKNOWLEDGMENTS

We acknowledge Joseph H. Condon for getting this project started. Essential to the success of the project were the suggestions and technical expertise of a large number of people. Accordingly, we would like to acknowledge T. D. S. Duff, E. H. Grosse, P. Hillner, A. G. Hume, B. W. Kernighan, T. J. Kowalski, W. Moy, D. L. Presotto, E. J. Sitar, F. H. Stillinger and N. T. Wilson. One of us (A. F. B.) also thanks J. W. de Bruin, D. A. van Delft, and F. F. van der Vlugt.

REFERENCES

1. F. F. ABRAHAM AND J. Q. BROUGHTON, *Phys. Rev. Lett.* **56**, 734 (1986).
2. M. D. KLUGE, J. R. RAY AND A. RAHMAN, *J. Chem. Phys.* **87**, 2336 (1987).
3. W. D. LUEDTKE, U. LANDMAN, M. W. RIBARSKY, R. N. BARNETT, AND C. L. CLEVELAND, *Phys. Rev. B* **37**, 4647 (1988).
4. G. H. GILMER AND M. H. GRABOW, AND A. F. BAKKER, *Mat. Sci. Eng. B* **6**, 101 (1990).
5. M. SCHEIDER, I. K. SCHULLER, AND A. RAHMAN, *Phys. Rev. B* **36**, 1340 (1987).
6. R. BISWAS, G. S. GREST AND C. M. SOUKOULIS, *Phys. Rev. B* **38**, 8154 (1988).
7. F. H. STILLINGER AND T. WEBER, *Phys. Rev. B* **31**, 5262 (1985).
8. D. HAMANN AND R. BISWAS, *Phys. Rev. Lett.* **55**, 2001 (1985).
9. J. TERSOFF, *Phys. Rev. Lett.* **56**, 632 (1986); *Phys. Rev. B* **37**, 6991 (1988); M. I. BASKES, *Phys. Rev. Lett.* **59**, 2666 (1987).
10. J. L. GUSTAFSON, G. R. MONTRY, AND R. E. BENNER, *SIAM J. Sci. Stat. Comput.* **9**, 609 (1988).
11. A. F. BAKKER, C. BRUIN, F. VAN DIEREN AND H. J. HILHORST, *Phys. Lett. A* **93**, 67 (1982); A. F. BAKKER, in *Computer-based microscopic description of the structure and properties of materials, Proceedings*, Vol. 63, (Materials Research Society, Pittsburg, PA, 1986), p. 181 and see references therein.
12. D. J. AUERBACH, W. PAUL, A. F. BAKKER, C. LUTZ, W. E. RUDGE AND F. F. ABRAHAM, *J. Phys. Chem.* **91**, 4881 (1988).
13. A. RAHMAN, *Phys. Rev. A* **136**, 405 (1964).
14. B. P. VAN EIJCK AND J. KROON, *J. Mol. Structure* **195**, 133 (1989).

15. M. KARPLUS, A. T. BRUNGER, R. ELBER AND J. KURIYAN, *Cold Spring Harbor Symp. Quant. Biol.* **52**, 381 (1987).
16. M. H. GRABOW AND G. H. GILMER, in *Semiconductor-Based Heterostructures: Interfacial Structure and Stability*, edited by M. L. GREEN, J. E. E. BAGLIN, G. Y. CHIN, H. W. DEKMAN, W. MAYO AND D. NARASINHAM, (Metallurgical Soc., Warrendale, PA, 1986), p. 3; G. H. GILMER AND M. H. GRABOW, *J. Metals* **39**, 19 (1987).
17. R. W. HOCKNEY AND J. W. EASTWOOD, in *Computer Simulation Using Particles* (McGraw–Hill, New York, 1981).
18. The SIMPLE algorithm was implemented in order to distribute local regions of the chamber to different P boards; for a description of the method see S. V. PATANKAR, *Numerical Heat Transfer and Fluid Flow* (McGraw–Hill, New York, 1980).