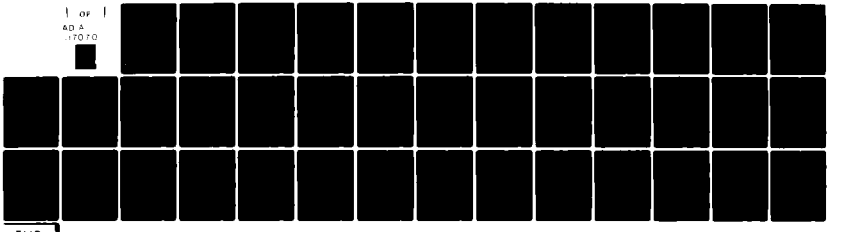
AD-A117 070 CALIFORNIA UNIV SAN DIEGO LA JOLLA DEPT OF CHEMISTRY F/G 7/4
MOLECULAR MECHANICS WITH AN ARRAY PROCESSOR.(U)
JUN 82 P H BERENS, K R WILSON N00014-78-C-0325
UNCLASSIFIED TR-8 NL



END
DATE FILMED
08-82
DTIC

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1 REPORT NUMBER: 8	2 GOVT ACCESSION NO.: AD A117070
3 RECIPIENT'S CATALOG NUMBER:	
4 TITLE (and Subtitle): MOLECULAR MECHANICS WITH AN ARRAY PROCESSOR	5 TYPE OF REPORT & PERIOD COVERED: Technical Report
	6 PERFORMING ORG. REPORT NUMBER:
7 AUTHOR(s): Peter H. Berens and Kent R. Wilson	8 CONTRACT OR GRANT NUMBER(s): ONR-N00014-78 C-0325
9 PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Chemistry, University of California, San Diego, La Jolla, CA 92093	10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11 CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Arlington, VA 22217	12 REPORT DATE: June, 1982
	13 NUMBER OF PAGES: 23 pages + figures
14 MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15 SECURITY CLASS. (of this report): Unclassified
	15a DECLASSIFICATION/DOWNGRADING SCHEDULE:

16 DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18 SUPPLEMENTARY NOTES

To be published in the Journal of Computational Chemistry.

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

molecular mechanics
computer simulation
infrared spectra
Raman spectra
electronic spectra
solutions reactions

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

OFFICE OF NAVAL RESEARCH

Contract N00014-78 C-0325

TECHNICAL REPORT NO. 8

Molecular Mechanics with an Array Processor

by

Peter H. Berens and Kent R. Wilson

Prepared for Publication

in

Journal of Computational Chemistry

University of California, San Diego
Department of Chemistry
La Jolla, CA 92093

MOLECULAR MECHANICS WITH AN ARRAY PROCESSOR

Peter H. Berens and Kent R. Wilson

Department of Chemistry
University of California, San Diego
La Jolla, California 92093

ABSTRACT

Computer simulation of the mechanics of molecular systems is a popular and powerful method for understanding chemical processes. The complexity of modeled chemical systems has advanced from hard spheres and rare gases to liquid solutions and biomolecules. Such simulations are computationally intensive and thus are limited by the speed of available computers. This paper describes the use of specialized hardware, a high speed floating point array processor (AP), to dramatically speed up molecular mechanics, in other words molecular dynamics, Monte Carlo, and energy minimization calculations. Although the array processor is a cost effective solution for computationally intensive problems in terms of hardware (full-time AP usage is equivalent to two to eight hours per day of Cray-1 time), its full speed comes at the expense of programming in a relatively difficult parallel assembly language. Since the architecture of the machine is dramatically different from conventional computers and utilizing its fast speed necessitates using this architecture on the assembly language level, the proper design and implementation of algorithms is critical. The molecular mechanics software design discussed here, consisting of 12000 lines of C and 7000 lines of AP assembly language code, is quite general and has been used to study systems ranging from rare gases to biomolecules. This implementation yields effective speeds approximately thirty-five times faster than a dedicated DEC VAX 11/780 computer with floating point accelerator and optimized VMS Fortran, thus allowing simulations to be run in a week and a half on the AP which would require a year of dedicated VAX time. The flexibility of the UNIX operating system, whose source code is accessible and can be modified to optimize performance, combined with the modern features of the C language, have made this implementation much easier, by providing a convenient and powerful environment in which to imbed the hand-coded AP assembly language modules. Applications to date range from the molecular dynamic calculation of infrared, Raman, and electronic spectra in gas and liquid solutions to the calculation of thermodynamic quantities for water and the simulation of the molecular dynamics of solution reactions and of polypeptides.

Submitted to *J. Comp. Chem.*, July 1982.

MOLECULAR MECHANICS WITH AN ARRAY PROCESSOR

Peter H. Berens and Kent R. Wilson

Department of Chemistry
University of California, San Diego
La Jolla, California 92093

I. INTRODUCTION

In recent years molecular mechanics, the computer simulation of molecular systems using molecular dynamics, Monte Carlo, and energy minimization, has emerged as a powerful tool for investigating and understanding chemical properties and processes. While straightforward in principle, these techniques can be unusually demanding computationally and thus the size and complexity of systems which can feasibly be simulated is determined by the speed and availability of computer hardware. In this paper we discuss a particular solution to the computational needs of molecular mechanics, the use of specialized hardware, a high speed array processor, in our case a Floating Point Systems, Inc. AP-120B, whose use we pioneered after purchasing serial number two. In other papers[1,2] we have also discussed an alternative solution, the division of the problem among an array of different processors operating in parallel. Yet a third solution[3] is to use a vector processing machine such as a Cray-1. We will focus here primarily on molecular dynamics within the array processor program package for molecular mechanics we have developed, called "Newton", whose conception is described in earlier papers.[1,2] In section II we examine the architecture of the AP-120B, as this is necessary to understand how to efficiently use the machine. In section III we lay out the specifics of the structure of the program package we have developed for molecular dynamics. Finally, in section IV we present the results, analyze potential array processor improvements, and point out the advantages and importance of the environment in terms of language and operating system.

Classical molecular dynamics[4] is the simulation of systems by the numerical integration of, for example, Newton's Second Law, to obtain particle trajectories, i.e. coordinates and momenta as functions of time, and then the computation of physical properties as averages over these trajectories. In the Monte Carlo technique, particles are moved artificially as the result of random number generation, rather than dynamically. The Monte Carlo technique is less general than molecular dynamics in that it is appropriate for the calculation of properties which depend upon averages over coordinates, but not for properties which are explicitly time dependent. These fields are reviewed by Alder,[5] Wood and Erpenbeck,[6] Valleau and Whittington,[7] McDonald,[8] Binder,[9] and Wood[10] and are the subject of a workshop.[11] Energy minimization[12-15] is the search for the minimum energy of the system as a function of particle coordinates.

Early liquid state molecular dynamic simulation[16] involved hard sphere atoms. Later, soft potentials, *e.g.* Lennard-Jones, were introduced.[17,18]

The authors provided phototypeset copy for this paper using TEX | LATEX | TROFF -ms on UNIX
UNIX is a trademark of Bell Laboratories.
DEC, VAX, PDP, VMS, and ULTRIX are trademarks of Digital Equipment Corporation.
FPS and AP-120B are trademarks of Floating Point Systems, Inc.

Computer simulation became a tool for testing approximate theories for atomic liquids.[19] More recently, with increasingly powerful software and faster hardware, attention has focused on more complex molecular systems, ranging from water[10] to proteins[20] and nucleic acids.

Array processors can bring much greater computer power to bear on chemical problems than was previously available.[1,2,21,22] However, their use carries the serious disadvantage that, at least up to the present, their maximum efficiency has only been gained by assembly language programming, as an efficient compiler for such unusual architecture is not yet available. Thus the development of efficient and general purpose software for array processors is very important as, in the long run, software costs will usually dominate hardware costs even in a University environment. We thus hope that the techniques discussed here which we have used in the development of Newton, our tool for molecular mechanics on an array processor, will prove useful to others using array processors for molecular mechanics as well as for other classes of problems.

II. ARRAY PROCESSOR

There are three ways to build faster computer hardware. The first is to increase the speed of the logical elements themselves, the second is to horizontally spread out the calculation among many parallel elements all working at once, and the third is to vertically spread out the steps of the calculation in time along a pipeline, so that many successive different calculations can trickle down the pipeline together. Arrays of processors[1,2] are an example of highly parallel architecture and supercomputers such as the Cray-1 and Cyber 205 make extensive use of pipelines (as well as parallelism).

The Floating Point Systems AP-120B (AP) array processor is a special purpose computer, which combines all three approaches, logic speed, parallelism, and pipelining, but pushes none of these to its extreme limit. It is designed for very fast processing of floating point arithmetic.[23] In comparison, logical instructions are much slower and more cumbersome. It is not an array of processors as many infer from the name, but a single parallel and pipelined processor whose architecture is easily exploited for array type operations. The maximum floating point rate is 12 million operations per second (12 megaflops). This speed puts the AP in the same computational class as many larger main frame computers, as indicated in Table I. Because it is a peripheral processor, the AP requires a host computer to initiate data and program transfers to and from it. The nature of the host interface as well as the host operating system are thus important factors, as will be discussed below.

Why Use an Array Processor?

The major reason that the use of an array processor is attractive is its cost effectiveness. This is shown in Table I from the work of Bucy and Senne[24] as is described in detail by Karplus and Cohen.[25] Columns one and two show computers that have been commonly used in scientific numerical applications and their theoretical speed, as measured in millions of floating point operations per second (megaflops). Columns three and four show the performance in terms of time and achieved megaflops for a sample application. Columns five and six show the motivating economic reason for using the AP-120B. It has the lowest cost per achieved megaflop and one of the lowest overall installation costs. However the last column shows where the real price is paid. The AP-120B has a substantially higher program development time, in addition to a costly learning curve as users must become familiar with its unusual and challenging parallel assembly

Table I. Comparison of Processor Cost/Performance for Demodulation Problem

Machine	Maximum Theoretical Megaflops	Time per Iteration (msec)	Achieved Megaflops	Approximate Cost (Dollars per Flop)	Installation Costs (Millions of Dollars)	Software Development Time (Man-months)
Cray-1	80-140	2.1	38.4	0.21	8	0.5
Star-100	25-50	4.9	16.8	0.48	8	2.0
Illiac IV	40-80	9.0	9.1	1.10	10	3.0
AP-120B	8-12	13.9	5.9	0.05	0.15	6.0
CDC 7600	6-15	25.0	3.3	0.91	3	1.1
IBM 370-168	2-4	94.0	0.87	2.30	2	1.0
CDC 6600	1-3	130.0	0.63	1.59	1	1.0
VAX-11/780	0.5	311.0	0.26	0.77	0.2	0.5
PDP-11/70	0.2	870.0	0.09	1.67	0.15	1.2

language.

Programming Techniques

Three methods of programming the array processor are available. The level highest in abstraction, but slowest in speed, is the Fortran compiler. Fortran is a language familiar to most computational chemists, but unfortunately no existing Fortran compiler[26-28] for the AP-120B generates efficient or compact code. Compiled code is thus lengthy (one can quickly consume more program source memory than is available for the AP-120B) and therefore necessarily slow, since the machine is synchronous. Most applications would thus require the storage of instructions in main data memory which is costly not only due to the loss of main data memory (two locations per each instruction stored) but also in the overhead of the overlaying process both in terms of the bookkeeping required and the actual transfer of the instructions to and from main data memory as the overlaying is done. The improvement in program speed and performance of Fortran in the AP-120B over a VAX 11/780 can be as little as a factor of two or three.[21] This degrades even further if many data transfers to and from the host or much loading of program source memory from main data memory is required.

The second level of usage is to vectorize the algorithm into a form in which the problem can be solved by a series of calls to canned vector routines written in assembly language by Floating Point Systems (FPS). This can be an improvement over the Fortran compiler, but since the code for the individual vector operations cannot be overlapped, the full advantage cannot be taken of the speed of the AP. Two modes of operation are possible, one in which the canned routines are called on an individual basis from the host C or Fortran program and the other where these calls are linked together in a rudimentary higher level language, know as the vector function chainer. The vector function program is then compiled into AP assembly code, and executes the set of operations as a group. It is vitally important to use the vector function chainer to string these operations together so that the overhead involved in loading and starting the AP is only paid once. However, the vector function chainer generates even worse code than the Fortran compiler, so any loops coded in the vector function program can also degrade performance. A typical increase in speed over that of a VAX 11/780 is three to twenty using this approach, as it is very dependent on how well the particular canned functions encompass the problem being solved. Molecular mechanics calculations tend to be at the low end of this range.

The third level of usage, needed to realize the full potential of the AP, is to hand-code commonly used algorithms in AP-120B assembly language. This presents a problem to most computational chemists in that it may require skills

unfamiliar to them. In order to investigate the reason for the necessity and inherent difficulty of assembly language coding, a knowledge of the architecture of the AP is helpful.

Architecture

The array processor is a parallel and pipelined machine [29] A parallel computer is one that can perform more than one operation in a single instruction The array processor consists of several independent memories and arithmetic units, as shown in Fig. 1, all of which can be addressed or initiated within a single instruction of the AP-120B There are three types of memory. Program source memory, which is 64 bits wide, is used to store assembly language instructions Data is not stored intermixed with instructions, as in most computers, but resides primarily in main data memory (38 bits wide) which is available in two speeds, known as "fast" and "slow" memory Data can also be stored in table memory which can consist of both Read Only Memory (ROM) and Random Access Memory (RAM) In our configuration we have 64 K words of main data memory, 2K words of table ROM, 4 K words of writable table memory, and 1.5 K words of program source memory. The actual hardware of the machine can support up to 4 K of program source, 4 K of writable table memory, and 512 K of main data memory (however, only one individual 64 K bank can be accessed at any one time). In addition there are two banks, DPX and DPY, of thirty two (38 bit) registers called data pads (only eight of which can be accessed at a time) for the storage of intermediate results. Addresses (pointers) and integer parameters are stored in sixteen (16 bit) integer registers, known as S pads In addition to the memories there are two separate floating point arithmetic processors, an adder and a multiplier. The floating adder is capable of a variety of operations such as addition, subtraction, logical *and*, as well as logical *or*. There is also an integer arithmetic unit which can be initiated every instruction and whose result is available for use in the instruction in which it was initiated.

Fig 2 shows the pipeline[29] nature of the AP-120B. The floating multiplier of the AP is a three stage pipeline, and thus a multiply operation started in one instruction will take a minimum of three instructions to complete A new multiply, however, can be initiated at every instruction and each initiation causes the preceding operations to be pushed through the stages of the pipeline. If the preceding operations are not pushed, the results stay in the pipeline until pushed out later by initiating other multiplies. The floating adder is a two stage pipeline with similar properties to the multiplier. The memory units are also pipelined in the sense that a memory fetch initiated in one instruction does not complete and cannot be used until three (or two, in the case of table memory) instructions later.

Program branching can only be performed, with a few exceptions, on the output of either the floating adder or the integer arithmetic unit. For example, to branch if a particular floating number is negative, the number must first be generated in the adder, which requires two instructions On the third instruction the result is available from the adder and on the fourth, and not before, the result may be tested and branched upon. This is the principle reason that calculations which require a significant amount of logic do not perform as well on the AP-120B Branching based on integer operations is not as slow because the result is available in the instruction in which the operation is initiated and may thus be tested in the following instruction. In either case, building logic efficiently into an AP-120B assembly language program can be quite cumbersome

Another important feature of the architecture of the AP-120B is that each separate memory or register unit can have one read or write (in the case of

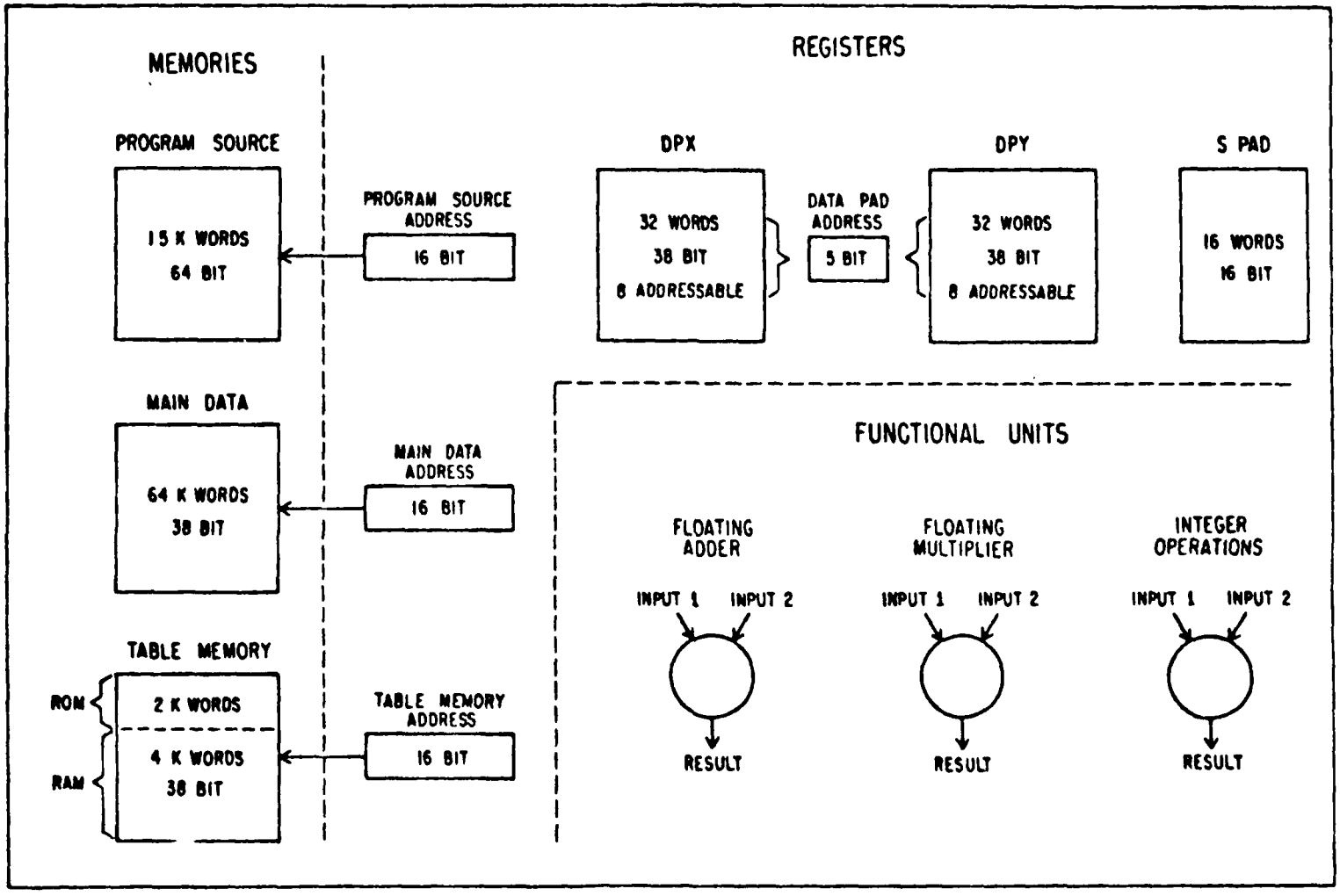


Figure 1. AP-120B multiple functional units illustrating the parallel nature of the array processor (AP). It consists of multiple memory and arithmetic units which can all be accessed or initiated in a single instruction cycle.

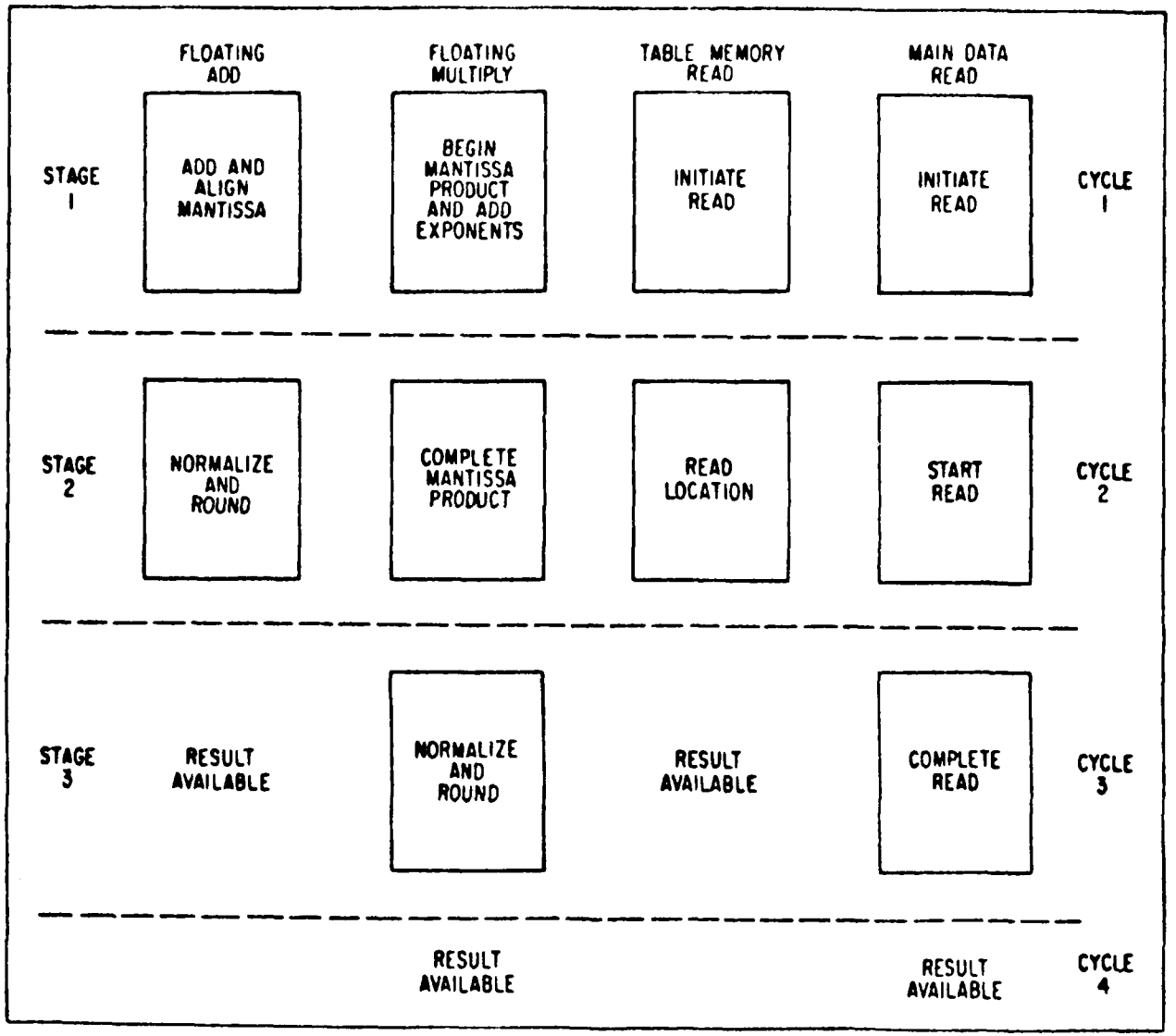


Figure 2. AP-120B functional unit pipelines. The various memory and arithmetic unit pipelines are shown. The floating multiplier and main data memory consist of three stage pipelines. The floating adder and auxiliary memory (table memory) are two stage. The arithmetic pipelines, unlike the memory pipelines, require the program to push preceding operations through the pipe each instruction cycle for the data to advance.

registers like DPX or DPY both a read and a write) initiated per instruction. Thus the placement of data among the various memories is extremely important in that the more spread out the data the more data can be accessed in an individual instruction. For this reason it is important to have writable table memory in addition to main data memory. The difficulty encountered with the use of writable table memory is that changing data stored in that memory is a multiple instruction process and very slow in comparison with writing into main data memory. Whether it is important to have the fast rather than the slow main data memory is highly dependent on the algorithm being coded. For molecular mechanics code, which is usually computation rather than memory bound, the slow memory seems adequate.

As can be seen in Fig. 3, all these units interconnect through a complicated data pad bus structure. It is contention for the data pad bus, which can contain only one value per instruction, that is the most common difficulty in AP assembly language programming. Thus efficient programming of the AP-120B involves exploiting the parallel nature of the components of the AP and the inner connectivity of the data paths to give the maximum throughput. This means that the manner in which a particular part of an algorithm is coded in the early part of the code has long reaching effects in the later stages of coding the algorithm. In many instances it is necessary to code an algorithm in more than one way and then examine which scheme can be made the most efficient. This can require a high degree of patience and persistence.

The AP-120B interfaces to the host computer (in our case a DEC VAX 11/750) through two (or possibly three) channels, as shown in Fig. 3. The virtual front panel is used by the host to control the AP-120B. It does not exist in reality but is a set of registers (in our case on the VAX UNIBUS) which can be examined and set by the host computer. The UNIX device driver for the AP uses these registers to start and stop the AP, to examine and deposit into memories and registers, and to initiate Direct Memory Accesses (DMA's) over the UNIBUS. Any data manipulation done using the front panel is necessarily very slow. Thus, data and program instructions are normally transferred between the host and AP via the DMA process which can occur at approximately a megabyte per second. Due to the difference in word lengths between the usual host floating point representation (32 bits) and the AP representation (38 bits), floating point numbers are converted "on-the-fly" during the DMA process by the AP's interface hardware. To preserve the full precision of the AP's representation requires using the much slower front panel or DMA'ing out the data in successive passes. Preserving full precision is important if numbers are to later be reloaded back into the AP and the calculation continued, as the result will not be the same as would be obtained by doing the same uninterrupted calculation in the AP if the numbers are rounded to 32 bits. The AP has a much larger dynamic range (10^{-155} to 10^{+155}) than the usual host representation (10^{-38} to 10^{+38}) and the only way in which these larger floating point numbers can be extracted from the AP-120B preserving the full 38 bits of precision is to use the front panel. However, if the floating point numbers are within range of the host's floating point representation, precision can be preserved by DMA'ing out the data in multiple passes.[30] First the actual data is DMA'ed out, undergoing convergent rounding to 32 bits. Next the data is DMA'ed back to the AP and subtracted from its original value in the AP. The result of this subtraction, the remainder, is then DMA'ed to the host. These two arrays (the rounded data and remainder) can then be used to reconstruct the original value in the AP by DMA'ing them into the AP separately and adding them together in the AP. This requires the allocation of scratch areas in the AP's memory.

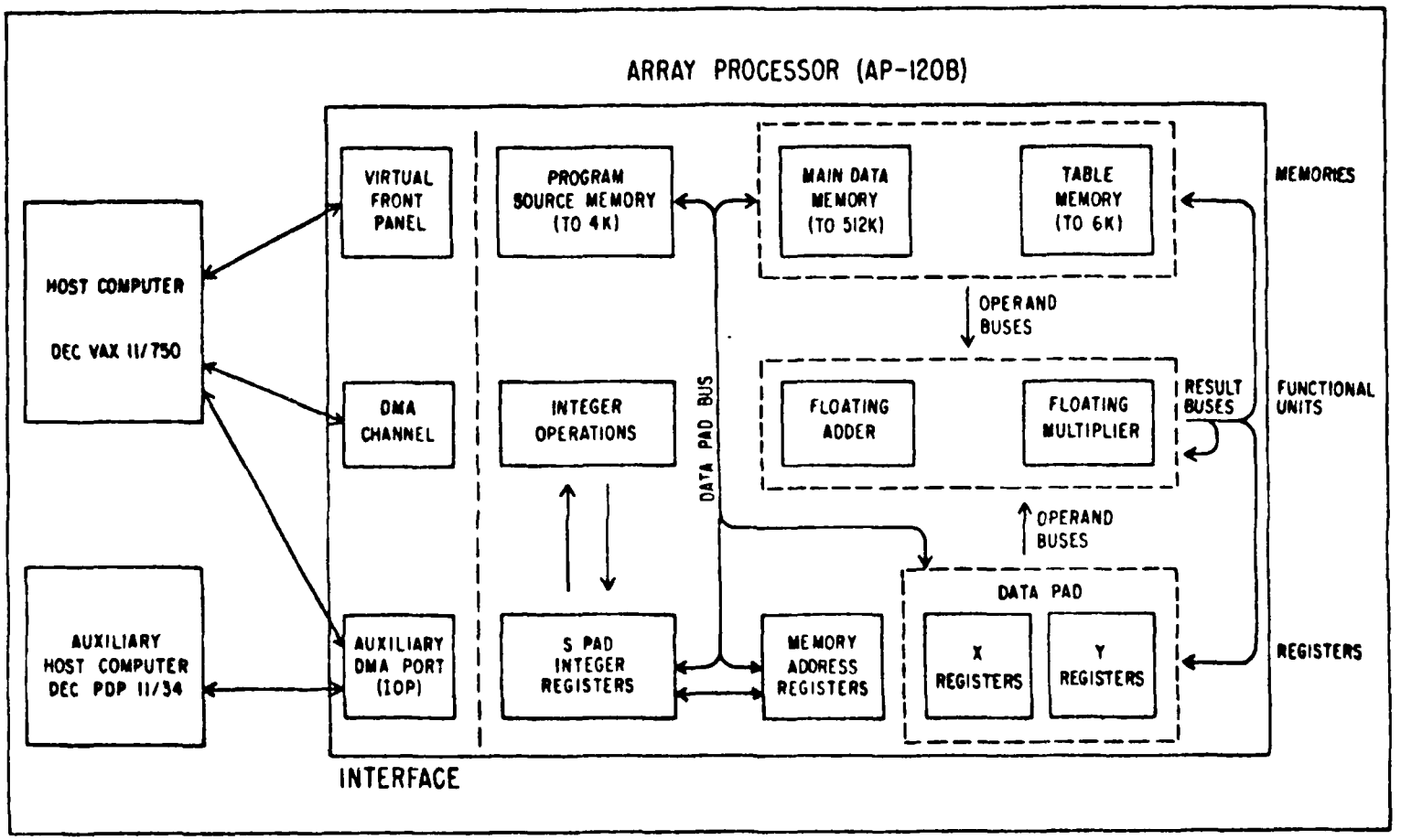


Figure 3. AP-120B host interface and internal connections. The various data paths connecting the individual functional units internal to the AP-120B are shown in the right portion of the figure. On the left are shown the data paths that interface the AP to the host computer.

The DMA channel also is a bottleneck when large volumes of data are to be transferred to and from the host computer. The AP can use approximately 80% of the bandwidth of the UNIBUS. A typical transfer rate is approximately 0.8 MB (megabytes per second). Particularly operating system dependent is the severity of overhead paid for initiating each transfer. On VAX's, the poor hardware design of the UNIBUS adapter also lowers the bandwidth. However, the throughput can be maximized by the using the asynchronous capability of the DMA channel. A third optional DMA channel, know as an IOP, operates in a manner similar to the standard DMA channel except that it has less general format conversion capabilities and lacks a sophisticated handshaking protocol which may make it inadequate for some applications.

Another important architectural feature of the AP-120B is that all main data memory locations are 38 bits and designed to accommodate a single floating point number. While it is a quite possible (and easy) to store a single 16 bit integer in each 38 bit location, in many instances it would be more efficient to pack more than one integer in each floating point word. This would provide savings not only in the amount of memory required but also in execution time by cutting down the number of memory accesses required to retrieve the data. Although the 38 bit word, as shown in Fig. 4, consists of three different fields, a 16 bit low mantissa, a 12 bit high mantissa, and a 10 bit exponent, not all of these bits are accessible to an AP-120B assembly language program. All 16 bits of the low mantissa and all 10 bits of the exponent can be placed into the S pad registers, but only 8 of the 12 bits of the high mantissa (the so-called table lookup bits) can be accessed by the AP-120B program. Furthermore, there is no way to write these integers back into the packed word from an AP program and these packed integers cannot be loaded by the DMA channel into the AP-120B, but must be loaded using the front panel which can examine and deposit all the bits of the low mantissa, high mantissa and exponent separately.

III. NEWTON

Our tool for molecular mechanics, Newton, consists of a variety of hardware processors and a generalized software package which will be described in this section. The software portion of Newton consists of 12000 lines of C code and 7000 lines of AP code. It is designed to be a modular and generalized approach, applicable to systems from rare gases and diatomics to polypeptides, proteins and nucleic acids in solution. While Newton can be used for Monte Carlo and energy minimization, its major use to date has been for molecular dynamics based calculations, and it is these we will describe in more detail. The array processor, as shown in Fig. 3, is connected to a VAX 11/750 as the host processor. The second DMA port can be connected to a PDP 11/34 which serves as the host to a three dimensional graphical display system, an Evans and Sutherland Picture System. All four of these processors can be activated simultaneously to allow the computation of molecular dynamics (and derived properties such as spectra and thermodynamics) and the real time display of the atomic motions or derived properties on the Picture System as they are calculated.

We know of four other applications of aspects of molecular mechanics which have been made using array processors. Pottle, Scheraga and co-workers[31] describe a package for energy minimization of proteins. Although the problems of force evaluation are similar, the approaches chosen are different. Newton is not confined to any one set of potential surfaces, such as the electrostatic plus Lennard-Jones potential surfaces used by Pottle, but is capable of implementing essentially arbitrary potential surfaces, as will be seen below. However, we have also paid the price for this generality. While their inner loop code operates using 80% of the theoretical maximum floating point speed of the array

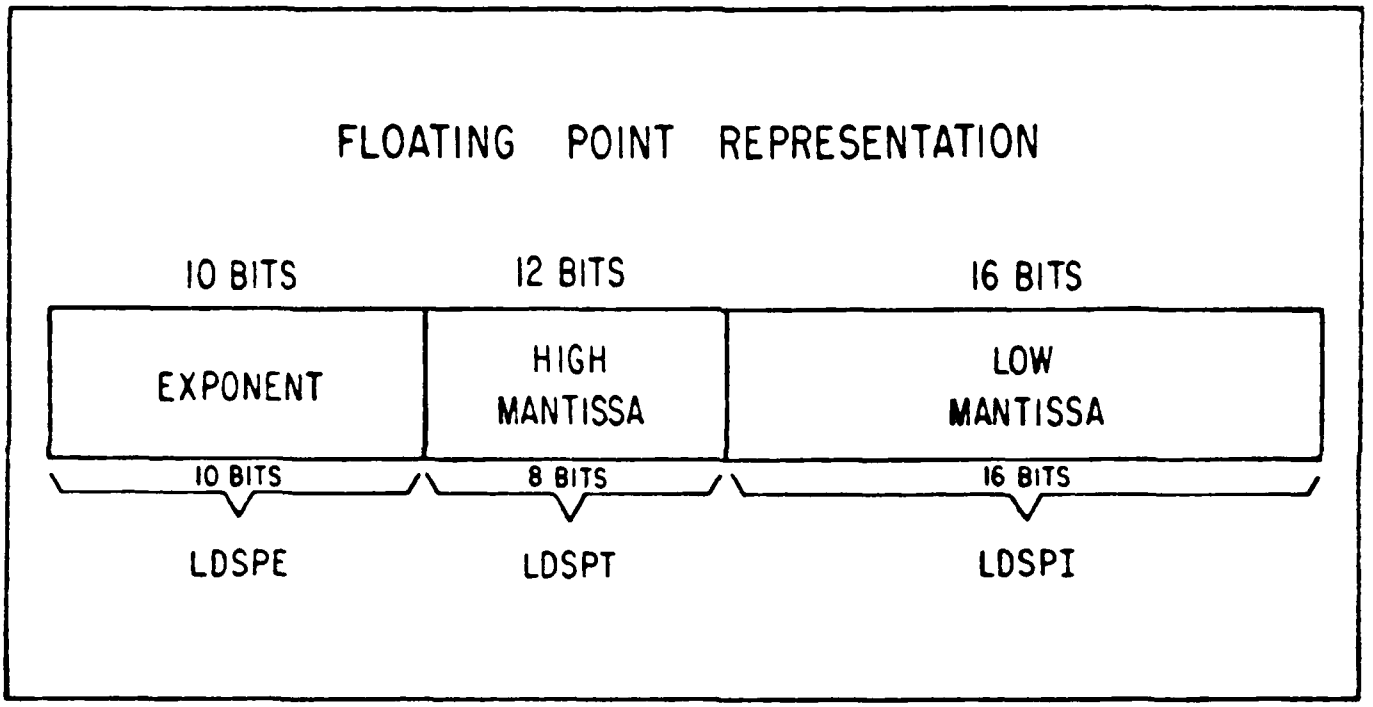


Figure 4. AP-120B floating point representation A main data memory word is divided into three parts: a 16 bit low mantissa field, a 12 bit high mantissa and a 10 bit exponent Below are shown the various AP instructions that can be used to read and treat these fields as three variable length integers Note that some of the bits are inaccessible in this manner.

processor, we average somewhat less than 50% This is primarily due to the extensive logic and indexing necessary to handle the general case Much of this code is overlapped with the actual floating point computation (since the logic consists of integer operations) With a conventional computer, the integer operations would have to be done in serial with the floating point operations This group has also implemented molecular dynamics calculations for water[32] and Monte Carlo calculations for liquid ammonia[33] and for liquid methane [34] In addition, Andersen and Swope[30,35] have implemented molecular dynamics code for water and atomic solutes in water, and Berne and co-workers[36] have carried out Monte Carlo calculations, both on an AP-120B Dammkoehler and co-workers[37] are implementing molecular mechanics code on multiple CSP,[38] array processors Also of related interest are the crystallographic work of Furey and co-workers[39] and the simulations of plasma dynamics by the UCLA group,[40] both on FPS AP-120B's

Newton contains seven separate host programs which operate cooperatively as shown in Fig. 5. The first of these is APCOM which is an interactive command interpreter It allows the user to type in various commands, validates the commands and, where appropriate, instructs the next link in the chain APRUN, to perform the operation requested APRUN is the program responsible for running the AP, and its only function is starting and stopping the AP and emptying the data collection buffers while the AP is running It uses two other programs, APLOAD to load the AP initially with constants and parameters, and APTABLES to calculate the force lookup tables for the intermolecular interactions whenever a new system is simulated DRAW is an optional process that can be used to display the dynamics on the Picture System as they are being calculated on the AP MOVIE can be used to display the previously calculated dynamics of a system using coordinates saved in a file by APRUN It also allows color movies to be made from these files. DODATA is a program which allows the computation in background mode of a series of runs to collect data over an ensemble of initial starting configurations and momenta It is interruptible to allow users to carry out program testing and development and systems tasks such as disk backup without interfering with the runs

All of the actual code involved in molecular dynamics calculation has been written in AP-120B assembly language and split into independent modules, as shown in Fig 6. Other AP modules not described here provide for other types of molecular mechanics and for the calculation of phenomena derived from molecular dynamics, such as infrared,[41,42] Raman,[43-45] and electronic[46,47] spectra and thermodynamic quantities [48] The modules are then linked together using a simple vector function chainer program that loops over the routines to perform the number of integration steps requested Data is buffered inside the AP-120B and DMA'ed out asynchronously when the buffers are filled, while the AP is running The buffering mechanism will be examined later in more detail.

The remainder of this section concentrates on our approach to molecular mechanics The implementation of molecular dynamics consists of two basic steps, force evaluation followed by numerical integration. It is these two functions which are performed by the AP modules shown in Fig 6 and the methodology behind their operation and design which will be discussed We examine the types of lists and indices needed for a general purpose molecular mechanics package We also examine various types of boundary conditions which are important in many applications, especially those with long range forces.

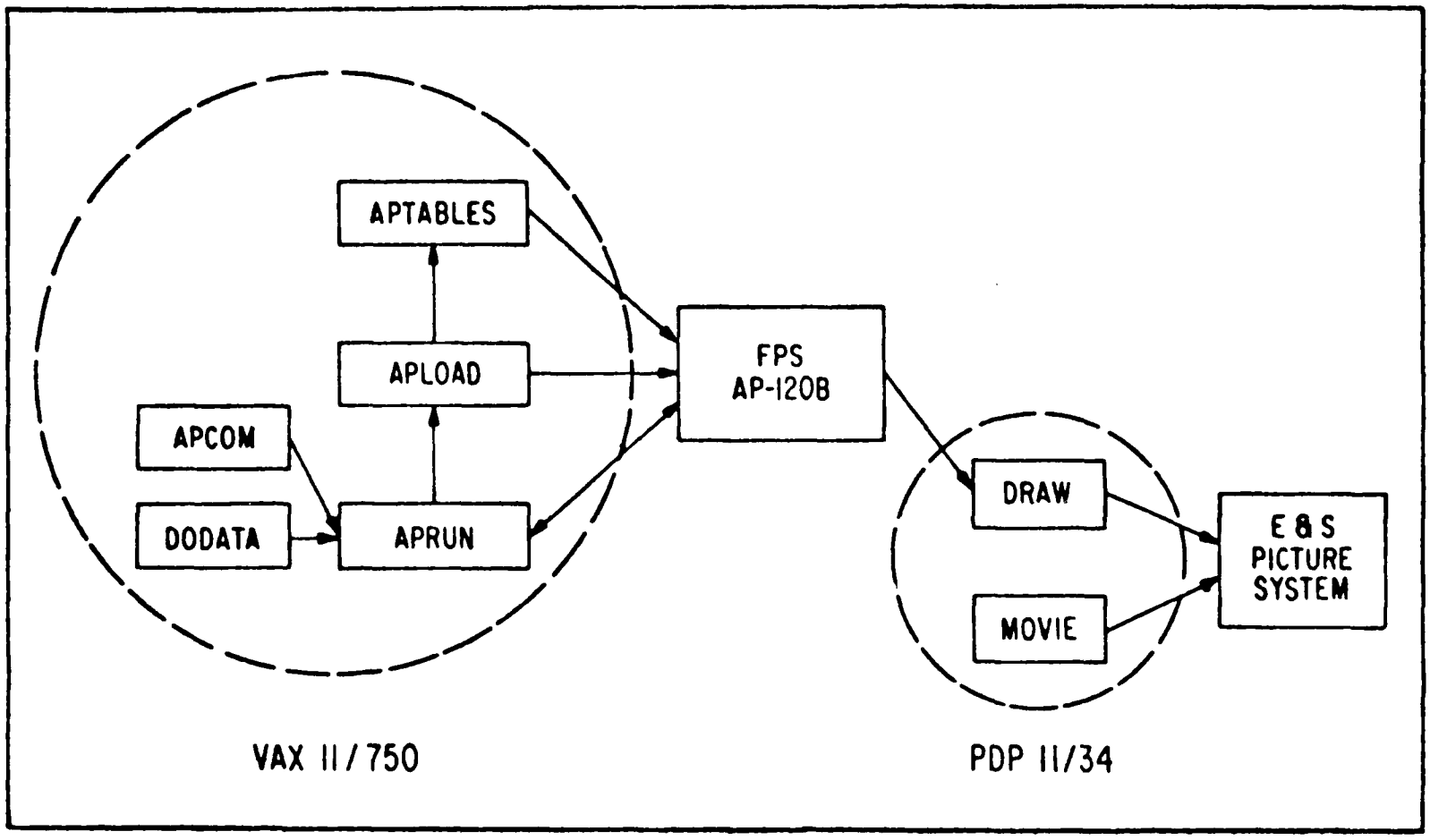


Figure 6. Overview of the molecular mechanics software in Newton. The modules on the left are C program packages that execute cooperatively on the VAX host computer. Their intercommunication and interface to the AP are shown. On the right is an optional interface from the AP to another host computer, a DEC PDP 11/34, to which an Evans and Sutherland Picture System is attached. The modules on the right are C program packages that run on the PDP 11/34.

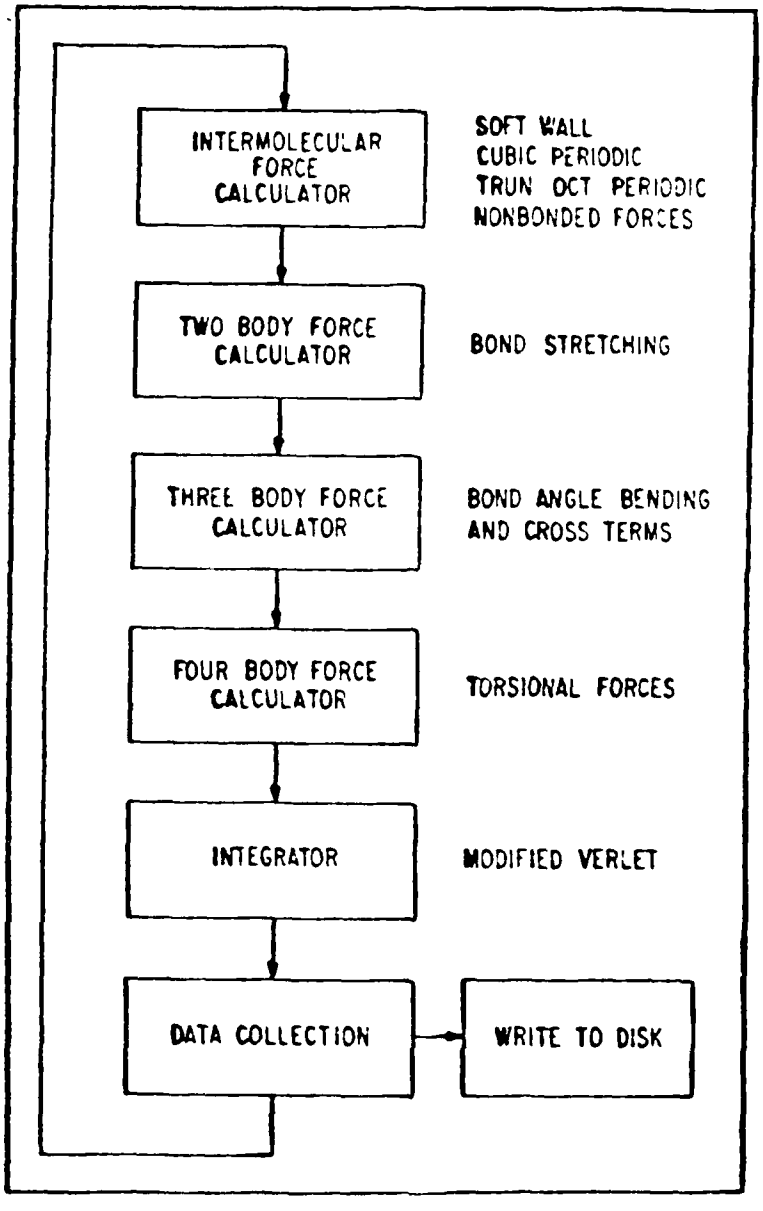


Figure 6. Newton AP modules for molecular dynamics. The various hand-coded AP-120B assembly language modules are shown in the loop in which they are executed. The entire loop is executed in the AP itself with no intervention required from the host. Brief descriptions of the modules are given on the right.

How to Describe an Atom

In addition to initial coordinates and momenta, other information is needed to create the wide variety of lists necessary to look up masses, positions, and forces among atoms. These can all be derived from a basic set of information.

```
struct  {
        char        flags,
        char        type,
        int         parent,
        int         param,
        } parts,
```

where the above is the C language[49] template for a structure describing the atom *parts*. The first item in the structure is the atom *flags*. These flags can be used to enable or disable particular features. For example, one flag bit can be set to fix the atom in space so that it cannot move (although it will still exert forces on neighboring atoms). Another is used to specify that the atom is the start of a molecule. If this flag is set then the *param* part contains the total number of atoms that follow which are in the same molecule. Another flag bit can be set to indicate that the atom is part of a ring structure. In this case the *param* element contains the other *parent* of this atom necessary to complete the ring structure. Another flag is used to specify if the atom is to be drawn in the picture display or not. Currently these four flags are the only ones used, although 8 such flags are possible for future uses.

Another part of the structure is the *type* of the atom, a number between 0 and 255. For example for water the atom types are hydrogen and oxygen. In organic molecules or ionic solutions it is often necessary to distinguish between different types of multivalent species (such as carbon) or different ionic states and thus each different chemical state of a particular elements will have a different *type* number.

The *parent* element specifies to which atom the present atom is connected. Using this tree linking mechanism the entire bond structure of any non-ring molecule can be determined. With the addition of the ring flag and the extra link provided by the *param* word, any reasonable chemical structure can be handled.

The atom *flag*, *type* and *parent* are stored as the exponent, high mantissa and low mantissa, respectively, of an AP-120B writable table memory location. The atom *param* word occupies the low mantissa of a main data memory word whose other fields are currently unused.

Solely from this simple information all the other lists needed by the AP-120B modules described below can be generated. The atom *parts*, atomic coordinates, and momenta are the only pieces of information kept in Newton fill files which are used to start or reload a Newton run.

Boundary Conditions

Two of the AP modules (the intermolecular force evaluater and the integrator) depend on the type of boundary conditions being used. Currently we have programs that allow the use of four types of boundary conditions: soft walls, hard walls, minimum image cubic periodic, and minimum image truncated octahedral periodic.[50] Other possible boundary schemes[7] include spherical and periodic boundary conditions using Ewald sums.

Cubic soft walls are the simplest. No imaging is done, and thus particles feel only the forces of the other particles in the cube. When a particle approaches a wall, a soft spring force pushes the particle back into the cube

This type of boundary condition is useful for studying small clusters or droplets of particles The disadvantage is that a high fraction of the particles can be on the surface of the droplet or cluster and, in many situations, these surface effects can be important Consequently, a larger number of particles are needed to study bulk phenomena In addition, collisions with a wall can give a molecule a large, artificial, angular momentum as it is shoved back toward the cluster

Hard walls are specularly reflecting and cause problems with integration as the velocity of the particle normal to the wall reverses itself in one integration time step Such a discontinuous change can cause integration algorithms to "blow up". This can be avoided by altering the algorithm so as to alter the past time history of the particle (as far back as necessitated by the algorithm) when it strikes the wall to become that of a particle having entered the box with the reversed normal velocity. The problems with surface effects still remain

Periodic boundary conditions[7] are commonly used to reduce surface effects for the simulation of bulk matter The simplest is a cubic minimum image In this scheme the system of particles resides in a central cube which is surrounded by exact replicas of this central cube on all sides, edges, and corners Particles interact only with the closest image of any other particle In all cubic periodic boundary algorithms, when a particle leaves the central box it is replaced by one of its images entering from the opposite side

A truncated octahedral boundary condition[50] is similar except that the unit cell is a truncated octahedron which more closely resembles a sphere This is important for minimum image boundary conditions as the forces must be smoothly feathered to zero at the radius of the inscribed sphere of the unit cell to avoid abrupt changes in force and loss of energy conservation. For a cube, 48% of the volume of the cube lies outside the inscribed sphere in the corners of the cube and particles in this volume don't contribute to the forces on the test atom For the truncated octahedral geometry, only 4.5% of the unit cell volume lies outside the inscribed sphere, and thus more of the dynamics calculation is effectively used. In addition, the excluded volume is more evenly distributed in angle than for the cube, and the isotropy of space is thus less distorted There exists an easy way[50] to code algorithms to implement the truncated octahedron The number of possible space filling solid tessellations is small Out of the regular and Archimedean polyhedra there are only five which are space filling: the cube, triangular prism, hexagonal prism, rhombic dodecahedron, and truncated octahedron,[2,51] and thus the natural alternative to the truncated octahedron would be the rhombic dodecahedron

Intermolecular Force Evaluation

The first module of AP code is the intermolecular force evaluater, used to compute nonbonded forces, i e those between atoms on different molecules, or separated by so many bonds in a single molecule as to be considered independent As pointed out above, it comes at present in four flavors soft and hard walls, cubic periodic, and truncated octahedral periodic boundary conditions. We approximate all intermolecular forces as purely pairwise additive, thus

$$V = \sum_{i<j}^{\text{nonbonded}} V(r_{ij}) , \qquad (1)$$

in which r_{ij} is the distance between the i th and j th atoms The potential function $V(r_{ij})$ depends solely on the chemical type of the atoms involved Currently, intermolecular forces are evaluated by looping over all the possible pairs of atoms in the system simultaneously calculating the force for both

members of the pair. This requires logic that allows the intermolecular force evaluater to skip over all the pairs of atoms whose forces are to be calculated by one of the other bonded force routines.

To skip over the appropriate bonded interactions the *parent* (and, if the ring flag is set, the *param* word) is used to determine bonding up to and including four body interactions which causes the intermolecular force evaluater to skip over these interactions. While this involves extensive integer arithmetic and logic, the overhead is inconsequential as it is overlapped completely with the code to do the minimum imaging for the periodic boundary conditions. Earlier experimental versions of the intermolecular module did not carry out this logic and instead calculated intermolecular forces for all pairs of atoms whether bonded or not. The bonded force evaluaters then simply subtracted out these erroneous intermolecular forces when they calculated the intramolecular forces. This addition and subtraction caused disastrous results due to numerical round off caused by adding the relatively small intramolecular forces to the large erroneous intermolecular forces that had not yet been subtracted out.

Andersen[30] has pointed out that when using a smoothing function for potential energy, the correct force evaluation involves calculating both the force and potential energy, as can be seen from,

$$V_s(r) = V(r)\,S(r) \tag{2}$$

$$F = -\frac{dV_s(r)}{dr} = -\left[\frac{dV(r)}{dr}S(r) + V(r)\,\frac{dS(r)}{dr}\right], \tag{3}$$

where $V_s(r)$ is the $V(r)$ potential smoothed to zero by the smoothing function $S(r)$.

Currently all intermolecular forces are calculated by table look up. This involves allocating most of main data memory to the force look up tables. Linear interpolation of these grid points is used to give the actual forces. At least in principle, as has been pointed out by Andersen and co-workers,[52] this scheme has a flaw when used with the Verlet integration algorithm due to the infinite second derivative of the force at the boundary between the linear segments. Andersen uses a better scheme employing a polynomial to fit fixed length segments of the potential curve with each polynomial joining smoothly and continuously out to several derivatives at the end points of the adjoining segments. This involves less data storage to calculate the intermolecular forces as only the polynomial coefficients need be stored. In addition, since polynomials are used, the potential energy as well as the forces can be calculated with little extra effort from the same set of coefficients.

For large systems, most of the computational time is spent in intermolecular force evaluation, since when done on a pairwise basis for the entire system of N atoms it becomes a calculation proportional to N^2, while the bonded, intramolecular calculations only scale as N. A possible alternative is to use neighbor lists[8,53] which are only updated after several time steps so that only the atoms which are close to the atom in question are scanned to calculate the intermolecular forces. However, neighbor lists create a considerable storage problem in that each atom must have a list of all the other atoms which are near it. For a relatively large system this list could easily exceed the amount of main data memory available. For small systems, the effort involved in updating and indexing the interactions from such a list may exceed the effort to do all the pair-wise calculations.

Two Body Module

The two body module calculates all simple bonded forces in diatomics such as *CO* and N_2. The force between two bonded atoms i and j with coordinates $\mathbf{r}_i$ and $\mathbf{r}_j$, respectively, is

$$\mathbf{F}_i = -\mathbf{F}_j = -\nabla_i V = \nabla_j V = -\frac{dV(r_{ij})}{dr_{ij}} \nabla_i r_{ij} \,. \tag{4}$$

in which $\mathbf{F}_i$ and $\mathbf{F}_j$ are the vector forces on the i th and j th atoms, respectively, which are separated by the distance $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$. ∇_i is the gradient with respect to the Cartesian coordinates of atom i as expressed in Eq. (A1) of the Appendix

The two body program uses a list as shown in Fig. 7. The low mantissa and exponent of the first word in the list are used to index the two atoms involved The atom number of the first atom (stored in the low mantissa) is subtracted from the atom number of the second before it is stored in the exponent field This allows us to use the fields, such as the exponent, to index an atom number which in principle can be much larger in magnitude than the bit field could normally handle. Since atoms that are bonded to each other tend also to be close to one another, the atom numbers are very close in magnitude, and thus the smaller bit fields are big enough to allow this relative indexing of atoms Code common to all two body force evaluations is used to fetch the atomic coordinates and calculate the internuclear vector. The potential index, or switch parameter, is then used to pick among a variety of force calculators (such as harmonic, Morse, *etc.*) that will calculate, given the internuclear vector, the scalar force along the bond The low mantissa of the second word in the list is the address of the force constants for the force evaluator to use. In this way, for example, one routine can be used to evaluate all harmonic forces. The common main line code then decomposes the force along the space fixed x, y, and z axes. This module is only used for diatomics. For more complex molecules it is more efficient to have the three body module also calculate the two body forces

Three Body Module

Three body interactions are those whose calculation depends on the coordinates of three particles, $\mathbf{r}_i$, $\mathbf{r}_j$, and $\mathbf{r}_k$. An example of this is a force due to bond angle bending which requires the three atoms involved to be specified in order to calculate the bond angle, ϑ. The three body module uses a list similar to the two body module, as shown in Fig. 7. The low mantissa of the first word contains the index to the middle atom in the three body interaction. The exponents of the first and second words contain the atom numbers of the other two atoms after subtracting the atom number of the middle atom. The high mantissa of the first word selects which three body force evaluation routine is to be used Currently we have two such evaluaters, a complex one[54] for water molecules and a simpler harmonic one,

$$V(\delta r_a, \delta r_b, \delta\vartheta) = k_0(\delta r_a)^2 + k_1(\delta r_b)^2 + k_2(\delta\vartheta)^2 \,, \tag{5}$$

which is written in terms of the bond vectors $\mathbf{r}_a$ and $\mathbf{r}_b$ where

$$\delta r_a = |\mathbf{r}_i - \mathbf{r}_j| - r_a^\circ = |\mathbf{r}_a| - r_a^\circ \tag{6}$$

$$\delta r_b = |\mathbf{r}_k - \mathbf{r}_j| - r_b^\circ = |\mathbf{r}_b| - r_b^\circ \,, \tag{7}$$

and $\delta\vartheta = \vartheta - \vartheta^\circ$, in which r_a°, r_b° and ϑ° are the equilibrium bond distances and angles, respectively, of the potential V. The two body part of the potential is shown in the above expression as it is also calculated in this module as explained above. The water[54] force evaluator has various higher order terms among δr_a,

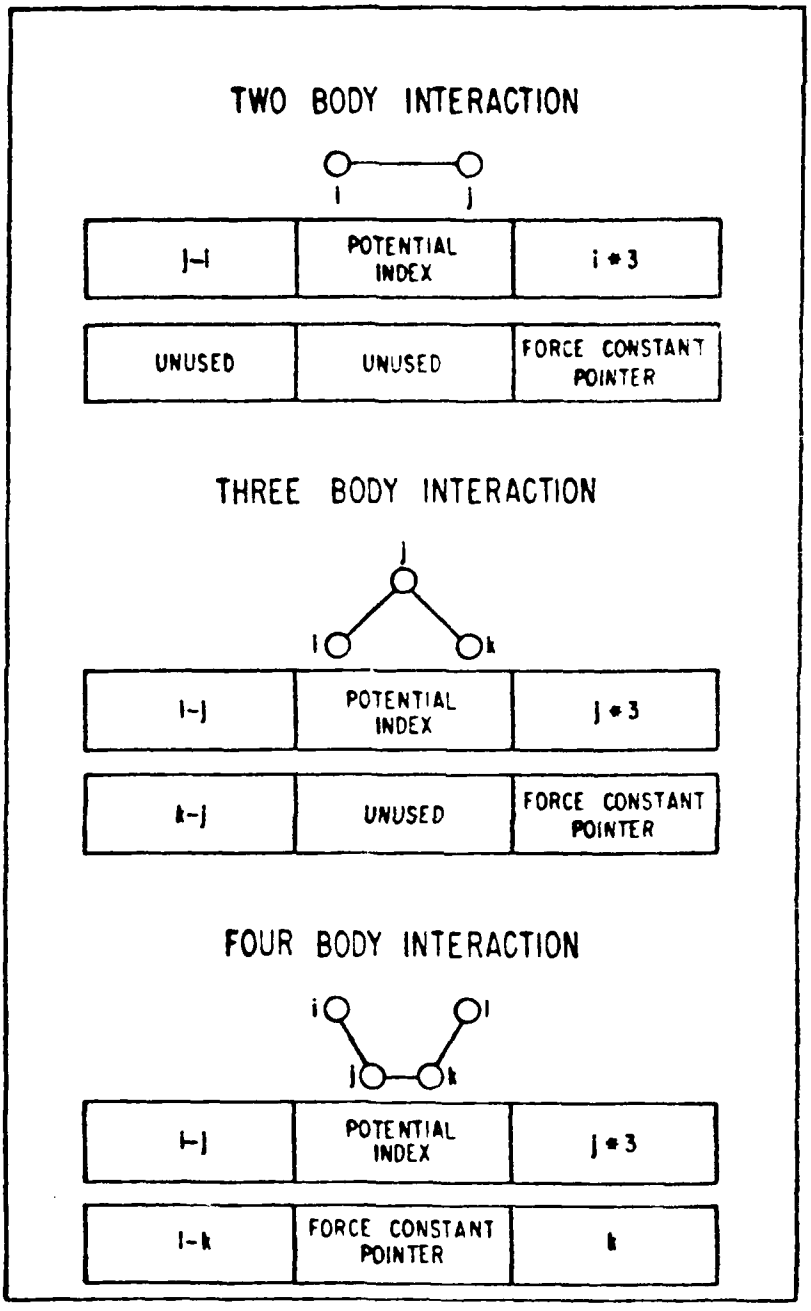


Figure 7. Format of interaction lists. The top panel shows the format of the two body interaction list as stored in AP main data memory. The low mantissa of the first word holds the *i* th atom number times three (for faster indexing into the three dimensional arrays). The high mantissa is an 8 bit integer specifying which force calculator to use (harmonic, Morse, *etc.*). The exponent field contains the difference between the *j* th and *i* th atom numbers. This is done to allow more dynamic range in the 10 bit field. The second word contains the address of the force constant. The switch parameter or potential index is used to specify which force evaluation routine is to be used for this interaction. The three body list, in the middle panel, is similar except that the middle atom is used to index the other two. The four body list, as shown in the bottom panel, is also similar except for an constant offset which is subtraced from the force constant pointer to allow more dynamic range.

δr_b, and $\delta\vartheta$ in addition to the terms shown in Eq. (5). The low mantissa of the second word contains the address of the force constants (k_0, k_1, and k_2) used by the force evaluators

Common code is provided by the three body module to calculate the two internuclear vectors and the bond angle before calling the appropriate three body force evaluator. The evaluator returns scalar forces along the two internuclear vectors and a force associated with the bond angle, and the three body module resolves these forces into Cartesian forces on the three atoms involved The forces on the atoms are given by

$$\mathbf{F}_i = -\nabla_i V = -\left[\nabla_a V \nabla_i \mathbf{r}_a + \frac{dV}{d\vartheta}\nabla_i\vartheta\right] = -\left[\nabla_a V + \frac{dV}{d\vartheta}\nabla_i\vartheta\right] \tag{8}$$

$$\mathbf{F}_j = -\nabla_j V = -\left[\nabla_a V \nabla_j \mathbf{r}_a + \nabla_b V \nabla_j \mathbf{r}_b + \frac{dV}{d\vartheta}\nabla_j\vartheta\right] = \left[\nabla_a V + \nabla_b V - \frac{dV}{d\vartheta}\nabla_j\vartheta\right] \tag{9}$$

$$\mathbf{F}_k = -\nabla_k V = -\left[\nabla_b V \nabla_k \mathbf{r}_b + \frac{dV}{d\vartheta}\nabla_k\vartheta\right] = -\left[\nabla_b V + \frac{dV}{d\vartheta}\nabla_k\vartheta\right], \tag{10}$$

where we have used Eqs. (A9) and (A10) of the Appendix to evaluate the chain rule gradients for the tensor vector product appearing in Eqs. (8)-(10). Similarly, using the results of the Appendix, the gradients involving the bond angle ϑ in Eqs (8)-(10) can be expanded in terms of gradients involving the bond vectors $\mathbf{r}_a$ and $\mathbf{r}_b$.

$$\nabla_i\vartheta = \nabla_a\vartheta \nabla_i \mathbf{r}_a = \nabla_a\vartheta \tag{11}$$

$$\nabla_j\vartheta = \nabla_a\vartheta \nabla_j \mathbf{r}_a + \nabla_b\vartheta \nabla_j \mathbf{r}_b = -\nabla_a\vartheta - \nabla_b\vartheta \tag{12}$$

$$\nabla_k\vartheta = \nabla_b\vartheta \nabla_k \mathbf{r}_b = \nabla_b\vartheta . \tag{13}$$

Since the bond angle can be written in terms of the dot product of the two bond vectors,

$$\cos\vartheta = \frac{\mathbf{r}_a \cdot \mathbf{r}_b}{r_a r_b}, \tag{14}$$

the terms in Eqs. (11)-(13) can be evaluated as

$$\nabla_a\vartheta = \frac{-1}{\sin\vartheta}\nabla_a\cos\vartheta . \tag{15}$$

Using Eqs (14) and (15),

$$\nabla_a\cos\vartheta = \frac{r_a r_b \nabla_a(\mathbf{r}_a \cdot \mathbf{r}_b) - \mathbf{r}_a \cdot \mathbf{r}_b \nabla_a(r_a r_b)}{r_a^2 r_b^2}, \tag{16}$$

where

$$\nabla_a(\mathbf{r}_a \cdot \mathbf{r}_b) = \mathbf{r}_b \tag{17}$$

and

$$\nabla_a(r_a r_b) = \frac{r_b}{r_a}\mathbf{r}_a . \tag{18}$$

Substituting Eqs. (17) and (18) into Eq. (16) yields

$$\nabla_a\cos\vartheta = \frac{r_a r_b \mathbf{r}_b - \frac{r_b}{r_a}(\mathbf{r}_a \cdot \mathbf{r}_b)\mathbf{r}_a}{r_a^2 r_b^2}. \tag{19}$$

Now, by using Eq. (14) with Eq (19) we have

$$\nabla_a \cos\vartheta = \frac{\mathbf{r}_b}{r_a r_b} - \frac{\mathbf{r}_a \cos\vartheta}{r_a^2} \tag{20}$$

A similar derivation for the gradient with respect to the other bond vector $\mathbf{r}_b$ yields

$$\nabla_b \cos\vartheta = \frac{\mathbf{r}_a}{r_a r_b} - \frac{\mathbf{r}_b \cos\vartheta}{r_b^2} \tag{21}$$

Finally, substituting Eqs (20) and (21) into Eq (15) we have

$$\nabla_a \vartheta = \frac{-1}{\sin\vartheta}\left[\frac{\mathbf{r}_b}{r_a r_b} - \frac{\mathbf{r}_a \cos\vartheta}{r_a^2}\right] \tag{22}$$

$$\nabla_b \vartheta = \frac{-1}{\sin\vartheta}\left[\frac{\mathbf{r}_a}{r_a r_b} - \frac{\mathbf{r}_b \cos\vartheta}{r_b^2}\right] \tag{23}$$

Thus, using Eqs (8)-(13) and Eqs (22) and (23) the forces can be appropriately resolved onto the three atoms involved. These results are the same as those arrived at using the Eliashevich and Wilson s-vector method[55,56] to evaluate the elements of the B matrix used in normal mode vibrational analysis to relate the internal and Cartesian coordinates through a Taylor's expansion.

Four Body Module

A four body interaction requires the knowledge of the positions of four particles to calculate the force. The two most common examples are torsional forces and out-of-plane bending forces.[55,56] The four body module for torsional forces uses a list as shown in Fig. 7. The atom numbers of the two inner atoms are stored in the low mantissa field of the first and second words, with the j th atom number multiplied by three for indexing convenience. The inner atom index numbers are first subtracted from the closer outer atom number and then stored in the exponent field of the two words. The high mantissa of the first word is used to store the potential index value to select a particular torsional force evaluator with the force constants being indexed by the value in the high mantissa field of the second word. Since a complete memory address cannot be stored here, the value in the high mantissa of the second word is an offset to the base address of the torsional force constants. Currently there are three types of torsional force evaluaters which handle single, double, and triple bonds.

Common code is used to calculate all the internuclear vectors and the torsional angle. A scalar force is returned which is solely dependent on the torsional angle. This force is then decomposed onto the atoms as follows. If we take a four body interaction as shown in top half of Fig. 8, each of the four atoms i, j, k, and l has coordinates represented by the vectors $\mathbf{r}_i$, $\mathbf{r}_j$, $\mathbf{r}_k$, and $\mathbf{r}_l$. If we now define the bond vectors,

$$\mathbf{r}_1 = \mathbf{r}_i - \mathbf{r}_j, \quad \mathbf{r}_2 = \mathbf{r}_k - \mathbf{r}_j, \quad \mathbf{r}_3 = \mathbf{r}_l - \mathbf{r}_k, \tag{24}$$

and look down the $j-k$ bond, then we have the representation shown in bottom half of Fig. 8 where $\mathbf{r}_a$ and $\mathbf{r}_b$ are the projection of $\mathbf{r}_1$ and $\mathbf{r}_3$ into a plane and the φ torsional angle is formed between them. We can now write

$$\mathbf{r}_a = \mathbf{r}_1 \times \mathbf{r}_2 \qquad r_a = |\mathbf{r}_a| \,. \tag{25}$$

$$\mathbf{r}_b = \mathbf{r}_3 \times \mathbf{r}_2 \qquad r_b = |\mathbf{r}_b| \,. \tag{26}$$

and

$$\cos\varphi = \frac{(\mathbf{r}_1 \times \mathbf{r}_2)\cdot(\mathbf{r}_3 \times \mathbf{r}_2)}{|\mathbf{r}_1 \times \mathbf{r}_2|\;|\mathbf{r}_3 \times \mathbf{r}_2|} = \frac{\mathbf{r}_a \cdot \mathbf{r}_b}{r_a r_b} \,. \tag{27}$$

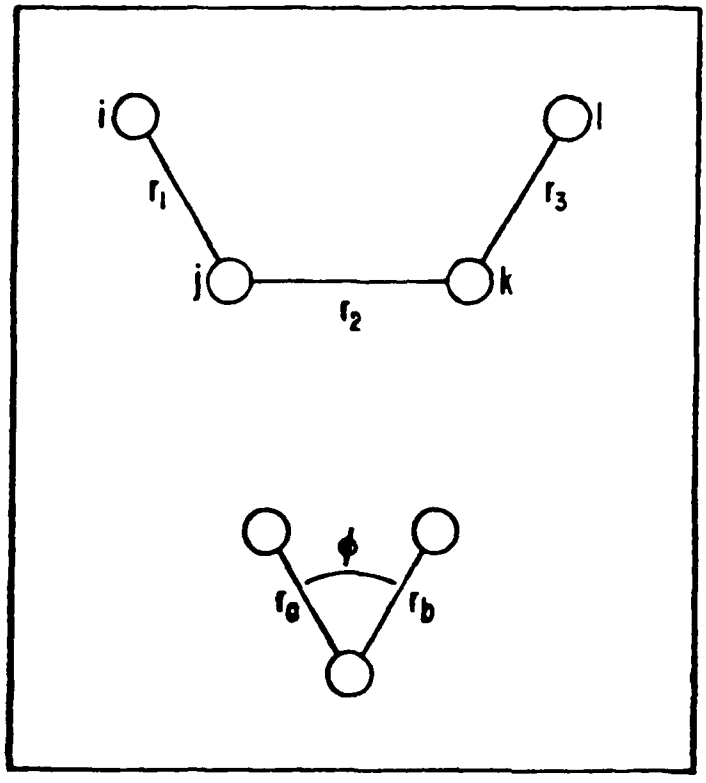


Figure 8. Calculation of torsional angle. The top half of the figure shows the atoms and vector conventions used in the calculation of the torsional angle. The bottom half of the figure shows the projection of the bonds in a plane when viewed down the center bond.

The force on each atom, using Eqs. (A9) and (A10) from the Appendix, is given by

$$-\nabla_1 V = -\nabla_1 V \nabla_1 r_1 = -\nabla_1 V \tag{28}$$

$$-\nabla_j V = -\nabla_1 V \nabla_j r_1 - \nabla_2 V \nabla_j r_2 = \nabla_1 V + \nabla_2 V \tag{29}$$

$$-\nabla_k V = -\nabla_2 V \nabla_k r_2 - \nabla_3 V \nabla_k r_3 = -\nabla_2 V + \nabla_3 V \tag{30}$$

$$-\nabla_l V = -\nabla_3 V \nabla_l r_3 = -\nabla_3 V . \tag{31}$$

Since the potentials used are solely functions of the torsional angle,

$$V = V(\cos\varphi) , \tag{32}$$

each of the terms in Eqs. (28)-(31) can be evaluated as

$$\nabla_\xi V = \frac{dV(\cos\varphi)}{d\cos\varphi} \nabla_\xi \cos\varphi , \qquad \xi = 1, 2, 3 . \tag{33}$$

However, following the format of Eqs. (A4)-(A7) of the Appendix, we can expand the gradients of $\cos\varphi$ in terms of the projections $\mathbf{r}_a$ and $\mathbf{r}_b$ to give

$$\nabla_\xi \cos\varphi = \nabla_a \cos\varphi \nabla_\xi r_a + \nabla_b \cos\varphi \nabla_\xi r_b , \qquad \xi = 1, 2, 3 . \tag{34}$$

From Eqs. (20) and (21) we know the gradients of the torsional angle are given by

$$\nabla_a \cos\varphi = \frac{\mathbf{r}_b}{r_a r_b} - \frac{\mathbf{r}_a \cos\varphi}{r_a^2} \tag{35}$$

$$\nabla_b \cos\varphi = \frac{\mathbf{r}_a}{r_a r_b} - \frac{\mathbf{r}_b \cos\varphi}{r_b^2} . \tag{36}$$

Using Eqs. (35) and (36) with the vector tensor products of Eq. (34) which are given by Eqs. (A13)-(A15) of the Appendix, we have

$$\nabla_1 V = \mathbf{r}_2 \times \left[\frac{\mathbf{r}_b}{r_a r_b} - \frac{\mathbf{r}_a \cos\varphi}{r_a^2} \right] \tag{37}$$

$$\nabla_2 V = \left[\frac{\mathbf{r}_b}{r_a r_b} - \frac{\mathbf{r}_a \cos\varphi}{r_a^2} \right] \times \mathbf{r}_1 + \left[\frac{\mathbf{r}_a}{r_a r_b} - \frac{\mathbf{r}_b \cos\varphi}{r_b^2} \right] \times \mathbf{r}_3 \tag{38}$$

$$\nabla_3 V = \mathbf{r}_2 \times \left[\frac{\mathbf{r}_a}{r_a r_b} - \frac{\mathbf{r}_b \cos\varphi}{r_b^2} \right] . \tag{39}$$

Using Eqs. (37)-(39) with Eqs. (28)-(31) the forces on all four atoms can be resolved.

Integration Module

The second step in molecular dynamics is the numerical integration of Newton's Second Law for each atom i,

$$\frac{\mathbf{F}_i}{m_i} = \mathbf{a}_i = \frac{d^2 \mathbf{r}_i}{dt^2} , \tag{40}$$

where m_i is the mass of the i th particle, $\mathbf{a}_i$ its acceleration, $\mathbf{r}_i$ its position, and t time. As pointed out above, the integration module depends on the type of boundary condition in use, as this module also applies any position changes necessary to keep the atoms in the unit cell (in the case of periodic boundaries) or applies any restoring forces necessary (in the case of soft walls). The integration algorithm we use in each of these modules is the same, a version of the Verlet algorithm as discussed by Beeman[57] with further modifications by Andersen.[36] In our implementation, the vector difference in positions $\mathbf{d}_i(t + 1)$ at time

step $t + 1$ is calculated from the previous difference $\mathbf{d}_i(t)$ and the force.

$$\mathbf{d}_i(t+1) = \mathbf{d}_i(t) + \frac{1}{h^2}\frac{\mathbf{F}_i}{m_i}, \tag{41}$$

in which t indicated the time step t, and h is the size of the time step. Next the new positions are calculated from the new difference in positions

$$\mathbf{r}_i(t+1) = \mathbf{r}_i(t) + \mathbf{d}_i(t+1). \tag{42}$$

The improvement made by Andersen is that now only the difference in position is stored rather than the velocity. Therefore there is less round off error as the numbers being added in each stage of the calculation are closer in magnitude. A crude forward difference velocity can easily be obtained by dividing the difference in position by the time step, or, if desired, more accurate velocities can be calculated.[57]

Beeman[57] shows that higher order integration techniques tend not to be as stable as the simple Verlet algorithm when larger time steps are used. He also shows that the Verlet algorithm conserves energy as well as other integrators tested in the large time step limit. The advantage of using this simple technique with the array processor is that only a minimal amount of memory is set aside for the storage of positions and the past time history of the system as compared with higher order integration methods.

Data Collection Modules

The data collection routine used for a particular application is often very specialized and thus to analyze different properties different AP modules must be written. However, MDOUT, the mechanism for buffering the data out to be the host computer, is common to all routines once the property that is to collected is calculated. In many cases this property can be calculated in a vector function program.

Various data collection modules exist for use with Newton. An example is the module that calculates and collects dipole moments and polarizabilities for the system as a function of time, in order to compute by linear response theory the infrared[41,42] and Raman[43-45] spectra. These modules operate in a very similar manner to the force evaluation modules, in that for the most part they use a list to index the atoms involved in the data calculation and the relevant parameters (such as static and derivative dipole moments for molecules). MDOUT saves the data in two internal buffers in main data memory and then DMA's the data out under interrupt control using a double buffering technique described in the next section. In general, the actual data is all that is saved from an individual Newton run. Normally, trajectories are not saved in that it is easier and faster to do the run over rather than saving and storing the details of the run.

IV. RESULTS AND ANALYSIS

As pointed out in section II, the main advantage of an array processor is high computational speed at a relatively low price. In the way of benchmarks, we have obtained the following results presented in Table II. Column one shows the various computers on which the benchmarks are taken. Columns two, three, and four compare the speed of the AP-120B for three different molecular mechanics packages. The one presented in column two is a direct C translation of our AP molecular dynamics package run on a VAX 11/780 and a VAX 11/750 without floating point accelerators (FPA's). Column three compares our AP version against an optimized Fortran version due to Hagler[58] and run on a VAX 11/780 with a floating point accelerator. Column four compares some speeds obtained for Monte Carlo

calculations from Chester, *et al* [59]

Table II. Comparison of Simulation Times on Various Computers

Computer	Molecular Dynamics (a) C w/ FPA on VAX	(a) Fortran w/ FPA on VAX	Monte Carlo
AP-120B	1	1	1
VAX 11/780	80	35	
VAX 11/750	150		
PDP 11	300		
Prime 400			850
IBM 370/168			1.5
CDC 7600			0.43

As can be seen from the table our molecular mechanics package is approximately thirty-five times faster than a VAX 11/780 with a floating point accelerator and an optimized Fortran compiler. Thus a simulation that can be run in a week and a half on the AP-120B would take a year on a VAX even if the VAX were totally dedicated to that calculation.

Although the AP-120B has proven to be a very fast and economical machine for molecular mechanics and allows us to simulate systems which otherwise would not be feasible, it is far from ideal. One concern is the word length of the floating point numbers. For example, for many quantum mechanical calculations 32 bit floating point representations are inadequate whereas 64 bit precision suffices. The question arises as to the adequacy of the 38 bits of the AP and if particular sections of algorithms can be painstakingly coded in the forced double precision possible on the AP to give adequate results. While most molecular mechanics is certainly adequately handled with 38 bits, one might do the integration module for molecular dynamics in double precision and leave the more computationally intensive force evaluation in single precision.

Another feature missing in the AP-120B is the ability to DMA data out of main data memory to the host processor preserving the full 38 bits of precision. This impacts the ability to do three things. The first is that we would like to be able to rapidly store intermediate results on disk so that the runs could be stopped and started at a later time without loss of precision in the data. A rapid method of writing such intermediate files would also facilitate periodic file dumps for restart capability if the host system were to crash. The second impacted area is the ability to rapidly load and retrieve integers packed into floating point words. The amount of time spent in loading the AP through the virtual front panel with long packed lists, for example, which would be necessary if neighbor lists were used could be extraordinarily burdensome. Thirdly, this limitation interferes with making the AP a rapidly sharable machine rather than an exclusive use device as it is now. To make it truly sharable all of the machine's memories and internal registers would have to be rapidly DMA'ed out to disk (swapped) and this is not possible with the present architecture.

In future versions of array processors we would also like to see a separate integer memory that could be accessed faster. Lacking this, an instruction should be added that would allow the access of all the bits of an integer packed into a main data word. Since molecular dynamics is not usually memory bound and most code is parallel for each of the three coordinate axes, more than one floating point adder and multiplier might be useful. In this way calculations for each of the three axes could be carried out in parallel rather than serially. The serial approach, however, is often convenient, especially given the fact that the multiplier is a three stage operation, but since the adder is only two stages it often disrupts any attempts to build the physical symmetry of the problem into the code. If such improvements were to be incorporated, speed enhancements

by utilizing fast main data memory could more easily be realized

Another obvious misfeature is the absence of integer multiplication which makes the addressing of multidimensional arrays difficult Additional memory address registers so that more than one item could be fetched from different banks of main data memory in the same instruction would replace the necessity of using the writable table memory for selective data storage, thus allowing all data to be accessible through the DMA channel, freeing table memory for seldom changed constants

There is currently available from Floating Point Systems a 64 bit array processor, known as the FPS-164 which solves the problem of numerical accuracy, at least for most problems of chemical interest It is, however, no faster (in fact somewhat slower) than the AP-120B and quite expensive It also differs in that program memory and data memory occupy the same space, and thus the Fortran compiler approach, although no faster in speed, is more feasible due to the abundance of memory now available for its bulky code This offers some advantages, for example, for quantum applications as large previously developed program packages can be run in the AP using its Fortran compiler with only the inner loops being optimized as hand coded AP routines Studies of usual quantum packages indicate that only 500 to 1000 lines of code take up most of the execution time.[60]

Another improvement that would facilitate the use of array processors for computational chemists, is a good higher level language compiler. We would suggest the language[49] C For many reasons C is more appropriate than Fortran for compiling into efficient AP-120B assembly language For example, C allows the use of pointers to arrays as an alternative to subscripts Incrementing and decrementing a pointer and stepping along memory can be much more efficiently handled in AP-120B assembly language than adding a subscript offset (which may need to be decremented if it does not start at 0, such as is the case with Fortran) to the array base address. In addition, C allows the declaration of variables as registers, thus allowing the programmer to warn the compiler that a particular piece of data needs to be kept in data pad registers rather than written and reread from memory. We are optimistic that a good C compiler can be written that will produce AP-120B assembly language code good enough to obsolete the desire to program in assembly language except for extremely critical loops which are executed too many times to tolerate any inefficiencies. The only such loop of this type in our molecular dynamics code is the intermolecular force evaluation, which is less than ten percent of the total AP code.

The actual generation of such a compiler is a difficult task Since the AP is not of the von Neumann architecture,[25] there is little expertise in this area of software design Ken Wilson[61] has suggested a Monte Carlo method of code generation, whereby given certain rules and constraints the AP itself would try to optimize its generated assembly code using a Monte Carlo technique, varying the code while preserving its logical outcome In this case code, not particles, would be randomly moved with the overall length of the code being minimized His attempt at implementing such an optimizer also starts with C language source code.

All of the support code for Newton run on the host computer is written in C. This has allowed us to maintain a single set of source code files which are shared by many programmers and used to simulate systems dramatically different in nature C is much more structured than Fortran and self documenting in many cases, as it has superior readability. It discourages the use of "go to" statements which have been described as a marvelous way to write impossible-to-understand programs Although it is a higher level language, it allows the

programmer the freedom and degrees of manipulation of data found in most typical assembly languages. It is portable since there is no built-in I/O and system dependencies are only in word lengths. Although C is very closely tied to the UNIX operating system, there are C compilers running on VAX's under VMS, on IBM 360's, on and various other machines. There is a portable C compiler that can be bought up on most machines with just a few months of work. The C compiler itself is written in C.

The advantages of the UNIX operating system[62,63] should not be overlooked. The reason UNIX and C are so related is that all of the UNIX utilities and over 90 percent of the actual UNIX kernel are written in C. Thus the UNIX operating system itself is portable. It is becoming a standard operating system for a wide variety of computers, so that we can (and have) moved both the UNIX operating system and Newton from one type of processor to another. Its debugging, editing, and friendliness to the user are superior, enhancing programmer productivity and the ease of making and debugging changes to Newton.

The development of Newton would have been very difficult without the UNIX operating system environment. The operating system kernel and device drivers for UNIX are written in C, and thus are easily changed. We have hand tailored the AP-120B driver to meet our needs. It allows the host computer and the AP-120B to operate asynchronously, coordinating efforts via interrupts. As the AP-120B fills up its data buffers it sends an interrupt to the host which causes the host to empty the buffer from AP-120B memory to disk while the AP-120B continues the calculation, filling up a second buffer. This means that there is no lost computation time by the AP-120B waiting for the host to empty the buffer and restart the calculation. Furthermore this procedure allows the host program to be inactivated (and even swapped) without having to loop just to check on the state of the AP. This change has increased our data throughput by over a factor of ten and allows the use of the AP-120B on a timesharing system with minimal impact to other users.

UNIX, a third generation operating system, is easy to learn. Our research group consists entirely of chemists, most of whom have little previous training in computer science and most have had little difficulty in picking up the necessary skills to use the operating system on a sophisticated level. Since most have little or no experience in traditional computer languages such as Fortran, it is interesting to note that they can begin writing complicated C programs in much less time that it would have taken to gain the equivalent abilities in Fortran.

V. CONCLUSION

As computational chemists search for more computer power, others will surely turn to array processors as we have, as they provide at the moment by far the most computational power per hardware dollar, particularly since the cost is low enough that they can be dedicated full-time to a particular task or class of tasks. While running on a supercomputer such as a Cray-1 will result in more computation per hour of processor use, it is unlikely to result in as much computation per year. The reason is that the equivalent to 24 hours per day of dedicated AP-120B time is, for example,[3,24,25,59] two to eight hours per day of Cray-1 time, a usage rate which few, if any, research groups are able to afford over the long run. Even if a group's budget were large enough to annually purchase this much supercomputer time, for the same cost several array processors could be purchased each year.

While array processor use is very appealing and the reward can be high, we believe our effort in bringing up a general purpose program package for molecular mechanics has also uncovered many of the pitfalls. That we can run in ten

days problems which would require a year of dedicated VAX 11/780 time allows us to handle problems in solution reaction and biomolecular dynamics which would not otherwise be feasible. However, the price we have paid is substantial. While molecular mechanics is straightforward in nature, it has taken over six man-years to develop efficient AP code to carry out the task.

An important feature of our code is its modularity. Since reprogramming is expensive, we have attempted to isolate the individual aspects of the calculation into individual AP modules. The generality of the program package allows us to simulate a wide variety of systems using essentially the same code. Past work includes the calculation from molecular dynamics and linear response theory of infrared,[41,42] Raman[43-45] electronic[46,47] spectra in the gas phase and in liquid solution. In addition, we have computed the dynamics and rotational and vibrational spectra of alkanes (such as methane, ethane, cyclohexane and their solutions), water (in both the gas and liquid phase as well as various N-mers of water molecules), and ions and microcrystals dissolved in water. We have computed the transient Raman and electronic absorption spectra during the course of a chemical reaction by computing the dynamics for the photodissociation of iodine in a solution of liquid xenon.[44-47] Other applications involve the computation from molecular dynamics of thermodynamic quantities and their quantum correction through spectral analysis of atomic velocity time histories.[48] Newton also incorporates a general set of protein potentials for biomolecules and is currently being applied to the molecular mechanics of polypeptides and membranes in collaboration with A. Hagler.

ACKNOWLEDGEMENTS

We thank the National Science Foundation, Chemistry, the Office of Naval Research, Chemistry and the National Institutes of Health, Division of Research Resources, for the support which has made this work possible and Don Mackay for his substantial work in obtaining many of the benchmarks presented here.

APPENDIX: CHAIN RULE FOR GRADIENTS

In the calculation and decomposition of forces it is often convenient to switch from Cartesian to internal coordinate systems. This presents difficulties as the gradients must also undergo this transformation. In this appendix we present formula for converting gradients in one frame of reference to another.

The gradient with respect to the Cartesian coordinates of particle i, $\mathbf{r}_i = x_i\hat{\mathbf{i}} + y_i\hat{\mathbf{j}} + z_i\hat{\mathbf{k}}$, is the vector operator given by

$$\nabla_i = \hat{\mathbf{i}}\frac{\partial}{\partial x_i} + \hat{\mathbf{j}}\frac{\partial}{\partial y_i} + \hat{\mathbf{k}}\frac{\partial}{\partial z_i}. \tag{A1}$$

The force on the i th particle $\mathbf{F}_i$ can be expressed using the operator in Eq (A1) as

$$\mathbf{F}_i = -\nabla_i V, \tag{A2}$$

where V is the potential energy. Commonly, however, the potential V is more easily expressed as a function of some internal coordinate $\mathbf{r}_a$, where the internal coordinate is a function of the Cartesian coordinates, $\mathbf{r}_a = \mathbf{r}_a(\mathbf{r}_i)$. We would therefore like to convert the gradient in Eq (A2) into a gradient with respect to the internal coordinate $\mathbf{r}_a$. Using the chain rule, the following terms result

$$\nabla_i V(r_a) = \left[\frac{\partial V}{\partial x_a}\frac{\partial x_a}{\partial x_i} + \frac{\partial V}{\partial y_a}\frac{\partial y_a}{\partial x_i} + \frac{\partial V}{\partial z_a}\frac{\partial z_a}{\partial x_i}\right]\hat{\mathbf{i}}$$
$$+ \left[\frac{\partial V}{\partial x_a}\frac{\partial x_a}{\partial y_i} + \frac{\partial V}{\partial y_a}\frac{\partial y_a}{\partial y_i} + \frac{\partial V}{\partial z_a}\frac{\partial z_a}{\partial y_i}\right]\hat{\mathbf{j}}$$

$$+ \left[\frac{\partial V}{\partial x_a} \frac{\partial x_a}{\partial z_i} + \frac{\partial V}{\partial y_a} \frac{\partial y_a}{\partial z_i} + \frac{\partial V}{\partial z_a} \frac{\partial z_a}{\partial z_i} \right] \mathbf{k} \tag{A3}$$

Equation (A3) can be written in a more compact form as

$$\nabla_i V(r_a) = \nabla_a V(r_a) \nabla_i r_a \tag{A4}$$

where $\nabla_a V(r_a)$ is a vector with components

$$\nabla_a V(r_a) = \left[\frac{\partial V}{\partial x_a} \quad \frac{\partial V}{\partial y_a} \quad \frac{\partial V}{\partial z_a} \right] \tag{A5}$$

and $\nabla_i r_a$ is a tensor[64] with components

$$\nabla_i r_a = \begin{bmatrix} \frac{\partial x_a}{\partial x_i} & \frac{\partial x_a}{\partial y_i} & \frac{\partial x_a}{\partial z_i} \\ \frac{\partial y_a}{\partial x_i} & \frac{\partial y_a}{\partial y_i} & \frac{\partial y_a}{\partial z_i} \\ \frac{\partial z_a}{\partial x_i} & \frac{\partial z_a}{\partial y_i} & \frac{\partial z_a}{\partial z_i} \end{bmatrix}. \tag{A6}$$

Thus by Eqs. (A5) and (A6), the expression in Eq. (A4) is actually the vector matrix product[65]

$$\nabla_a V(r_a) \nabla_i r_a = \left[\frac{\partial V}{\partial x_a} \quad \frac{\partial V}{\partial y_a} \quad \frac{\partial V}{\partial z_a} \right] \begin{bmatrix} \frac{\partial x_a}{\partial x_i} & \frac{\partial x_a}{\partial y_i} & \frac{\partial x_a}{\partial z_i} \\ \frac{\partial y_a}{\partial x_i} & \frac{\partial y_a}{\partial y_i} & \frac{\partial y_a}{\partial z_i} \\ \frac{\partial z_a}{\partial x_i} & \frac{\partial z_a}{\partial y_i} & \frac{\partial z_a}{\partial z_i} \end{bmatrix} \tag{A7}$$

which when expanded gives Eq. (A3).

We will now apply this technique to the examples presented in the text. In the case of the three body module, where the internal coordinate vectors $\mathbf{r}_a$ and $\mathbf{r}_b$ as in Eqs. (6) and (7) of the text, respectively, are given by the difference in position of the i th particle with respect to the j th particle, *viz.*,

$$\mathbf{r}_a = \mathbf{r}_i - \mathbf{r}_j \tag{A8}$$

it can be easily seen by evaluating the partial derivatives in Eq (A6) using Eq (A8) that the relevant tensors are given by

$$\nabla_i \mathbf{r}_a = \mathbf{I} \tag{A9}$$

$$\nabla_j \mathbf{r}_a = -\mathbf{I}, \tag{A10}$$

where $\mathbf{I}$ is the unit tensor.

However, in the four body force decomposition the internal coordinates $\mathbf{r}_a$ and $\mathbf{r}_b$ are vector cross products of the bond vectors $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$, *viz.*,

$$\mathbf{r}_a = \mathbf{r}_1 \times \mathbf{r}_2 = \left[y_1 z_2 - z_1 y_2, \quad z_1 x_2 - x_1 z_2, \quad x_1 y_2 - y_1 x_2 \right]. \tag{A11}$$

For this case we must evaluate the partial derivatives in Eq. (A6) using Eq. (A11) except with respect to the Cartesian vector $\mathbf{r}_1$ instead of $\mathbf{r}_i$. This gives for the gradient expressed in Eq. (34) of the text the following result

$$\nabla_1 \cos\varphi = \left[\frac{\partial \cos\varphi}{\partial x_a} \quad \frac{\partial \cos\varphi}{\partial y_a} \quad \frac{\partial \cos\varphi}{\partial z_a} \right] \begin{bmatrix} 0 & z_2 & -y_2 \\ -z_2 & 0 & x_2 \\ y_2 & -x_2 & 0 \end{bmatrix}. \tag{A12}$$

When Eq (A12) is expanded it becomes obvious that it is equivalent to

$$\nabla_1 \cos\varphi = \mathbf{r}_2 \times \nabla_a \cos\varphi \,. \tag{A13}$$

Similarly, the following results can be obtained for the other gradients

$$\nabla_2 \cos\varphi = \nabla_a \cos\varphi \times \mathbf{r}_1 + \nabla_b \cos\varphi \times \mathbf{r}_3 \tag{A14}$$

$$\nabla_3 \cos\varphi = \mathbf{r}_2 \times \nabla_b \cos\varphi \,. \tag{A15}$$

References

1. K R Wilson, in *Computer Networking and Chemistry*, ed P Lykos American Chemical Society, Washington, 1975, p 17.
2. K R Wilson, in *Minicomputers and Large Scale Computations*, ed P Lykos American Chemical Society, Washington, 1977, p 147.
3. D Fincham and B J Ralston, *Comput Phys Comm*, **23**, 127 (1981)
4. D Fincham, *Comput Phys Comm*, **21**, 247 (1980).
5. B J Alder, *Ann Rev Phys Chem*, **24**, 325 (1973).
6. W W Wood and J J Erpenbeck, *Ann Rev Phys Chem*, **27**, 319 (1976).
7. J P Valleau and S G Whittington, in *Statistical Mechanics Part A Equilibrium Techniques*, ed B J Berne, Plenum, New York, 1977
8. I R McDonald, in *Microscopic Structure and Dynamics of Liquids*, ed J Dupuy and A J Dianoux, Plenum, New York, 1977.
9. K Binder, *Monte Carlo Methods in Statistical Physics*, Springer-Verlag, Berlin, 1979.
10. D W Wood, in *Water, A Comprehensive Treatise, Vol 6, Recent Advances*, ed F Franks, Plenum, New York, 1979, p 279.
11. D Ceperely and J Tully, *Proceedings of the Workshop on Stochastic Molecular Dynamics*, National Resource for Computation in Chemistry, Berkeley, 1979
12. M Levitt and A Warshel, *Nature*, **253**, 694 (1975).
13. J A McCammon, B R Gelin, and M Karplus, *Nature*, **267**, 585 (1977).
14. A Warshel, in *Semi-Empirical Methods of Electronic Structure Theory, Part A*, ed G Segal, Plenum Press, New York, 1977, p 133.
15. A T Hagler, F Naider, P S Stern, and R Sharon, *J Am Chem Soc*, **101**, 6842 (1979).
16. B J Alder and T E Wainwright, *J Chem Phys*, **31**, 459 (1969)
17. A Rahman, *Phys Rev*, **136A**, 405 (1964).
18. L Verlet, *Phys Rev*, **159**, 98 (1967).
19. P Schofield, *Com Phys Comm*, **5**, 17 (1973).
20. J A McCammon and M Karplus, *Ann Rev Phys Chem*, **31**, 29 (1980)
21. N S Ostlund, in *Report of a Minicomputer Workshop*, National Resource for Computation in Chemistry, Berkeley, 1978.
22. N S Ostlund, *Attached Scientific Processors for Chemical Computations: A Report to the Chemistry Community*, National Resource for Computation in Chemistry, Lawrence Berkeley Laboratory (LBL-10409), Berkeley, 1980, also available from the National Technical Information Service.

23 W R Wittmayer, *Computer Design*, 93 (March 1978)

24 R S Bucy and K D Senne, *Comp Math Applications*, **6**, 317 (1980)

25 W J Karplus and D Cohen, *Computer*, 11 (Sept 1981)

26 D Bergmark, in *Proceedings of the 1978 ARRAY Conference*, Floating Point Systems, Inc., Portland, 1978

27 *Array Processor Fortran*, Floating Point Systems, Inc., Portland, 1978

28 *Toast Fortran Development System*, System Software Factors, Reading-Berkshire, England, 1981

29 A E Charlesworth, *Computer*, 18 (Sept 1981)

30 H C Andersen, private communication

31 C Pottle, M S Pottle, R W Tuttle, R J Kinch, and H A Scheraga, *J Comp Chem*, **1**, 46 (1980)

32 D C Rapaport and H A Scheraga, *Chem Phys Lett*, **78**, 491 (1981)

33 R H Kincaid and H A Scheraga, *J Phys Chem*, **86**, 833 (1982)

34 R H Kincaid and H A Scheraga, *J Phys Chem*, **85** 838 (1982)

35 W C Swope, H C Andersen, P H Berens, and K R Wilson, *J Chem Phys*, **76**, 637 (1982)

36 B J Berne, private communication

37 R Dammkoehler, private communication

38 P Alexander, *Indust Res Devel*, 111 (May 1980)

39 W Furey, Jr., B C Wang, and M Sax, *J Appl Crystallogr*, **15**, 160 (1982)

40 J M Dawson, R W Huff, and C Wu, *AFIPS National Computer Conference Proceedings*, **47**, 395 (1978)

41 P H Berens and K R Wilson, in *Picosecond Phenomena II*, ed R Hochstrasser, W Kaiser, and C V Shank, Springer-Verlag, Berlin, 1980, p 246

42 P H Berens and K R Wilson, *J Chem Phys*, **74**, 4872 (1981)

43 P H Berens, S R White, and K R Wilson, *J Chem Phys*, **75**, 515 (1981)

44 P Bado, P H Berens, and K R Wilson, in *Picosecond Lasers and Applications*, ed L S Goldberg, Proc Soc Photo-Optic Engin, Bellingham, WA, 1982, p 230

45 P H Berens, J P Bergsma, and K R Wilson, "Molecular dynamics and spectra III Transient Raman spectra for a chemical reaction," to be submitted

46 P Bado, P H Berens, J P Bergsma, S B Wilson, K R Wilson, and E J Heller, in *Picosecond Phenomena III*, ed K Eisenthal, R Hochstasser, W Kaiser, and A Laubereau, Springer-Verlag, Berlin, 1982, in press

47 P H Berens, J P Bergsma, and K R Wilson, to be submitted

48 P H Berens, D H J Mackay, G M White, and K R Wilson, "Molecular dynamics, thermodynamics, and quantum corrections for liquid water," to be submitted

49 B W Kernighan and D M Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978

50 D J Adams, in *The Problem of Long-Range Forces in the Computer Simulation of Condensed Media*, ed D Ceperely, National Resource for Computation in Chemistry, Berkeley, 1980, p 13

51. H. M. Cundy and A. P. Rollet, *Mathematical Models*, Oxford University Press London, 1961

52 T. Andrea, W. C. Swope, and H. C. Andersen, to be submitted.

53. B. Quentrec and C. Brot, *J. Comp. Phys.*, **13**, 430 (1973).

54. R. O. Watts, *Chem. Phys.*, **26**, 367 (1977).

55. E. B. Wilson, Jr., J. C. Decius, and P. C. Cross, *Molecular Vibrations*, McGraw-Hill, New York, 1955, Chapter 4.

56. S. Califano, *Vibrational States*, Wiley, London, 1976, Chapter 4.

57. D. Beeman, *J. Comput. Phys.*, **20**, 130 (1976).

58 A. T. Hagler, private communication

59 G. Chester, R. Gann, R. Gallagher, and A. Grimison, in *Computer Modeling of Matter*, ed. P. Lykos, American Chemical Society, Washington, 1980, p. 111.

60 A. Komornicki, private communication

61. D. Jacobs, J. Prins, and K. Wilson, in *Proceedings of the 1982 ARRAY Conference*, Floating Point Systems, Inc., Portland, 1982.

62 D. M. Ritchie and K. Thompson, *Bell Sys. Tech. J.*, **57**, 1905 (1978).

63. R. Thomas and J. Yates, *A User Guide to the UNIX System*, Osborne, Berkeley, 1982.

64. P. I. Richards, *Manual of Mathematical Physics*, Pergamon, London, 1959, p. 299.

65 R. E. Williamson, R. H. Crowell, and H. F. Trotter, *Calculus of Vector Functions*, Prentice-Hall, Englewood Cliffs, 1968, p. 167.

SP472-3/A1

472:GAN:716:enj
78u472-608

TECHNICAL REPORT DISTRIBUTION LIST, GEN

	No. Copies
Office of Naval Research Attn: Code 472 800 North Quincy Street Arlington, Virginia 22217	2
ONR Western Regional Office Attn: Dr. R. J. Marcus 1030 East Green Street Pasadena, California 91106	1
ONR Eastern Regional Office Attn: Dr. L. H. Peebles Building 114, Section D 666 Summer Street Boston, Massachusetts 02210	1
Director, Naval Research Laboratory Attn: Code 6100 Washington, D.C. 20390	1
The Assistant Secretary of the Navy (RE&S) Department of the Navy Room 4E736, Pentagon Washington, D.C. 20350	1
Commander, Naval Air Systems Command Attn: Code 310C (H. Rosenwasser) Department of the Navy Washington, D.C. 20360	1
Defense Technical Information Center Building 5, Cameron Station Alexandria, Virginia 22314	12
Dr. Fred Saalfeld Chemistry Division, Code 6100 Naval Research Laboratory Washington, D.C. 20375	1
U.S. Army Research Office Attn: CRD-AA-IP P.O. Box 1211 Research Triangle Park, N.C. 27709	1
Naval Ocean Systems Center Attn: Mr. Joe McCartney San Diego, California 92152	1
Naval Weapons Center Attn: Dr. A. B. Amster, Chemistry Division China Lake, California 93555	1
Naval Civil Engineering Laboratory Attn: Dr. R. W. Drisko Port Hueneme, California 93401	1
Department of Physics & Chemistry Naval Postgraduate School Monterey, California 93940	1
Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1
Naval Ship Research and Development Center Attn: Dr. G. Bosmajian, Applied Chemistry Division Annapolis, Maryland 21401	1
Naval Ocean Systems Center Attn: Dr. S. Yamamoto, Marine Sciences Division San Diego, California 91232	1
Mr. John Boyle Materials Branch Naval Ship Engineering Center Philadelphia, Pennsylvania 19112	1

TECHNICAL REPORT DISTRIBUTION LIST, GEN

	No. Copies
Mr. James Kelley DTNSRDC Code 2803 Annapolis, Maryland 21402	1
Mr. A. M. Anzalone Administrative Librarian PLASTEC/ARRADCOM Bldg 3401 Dover, New Jersey 07801	1

472:CAN:716:tam
78u472-608

TECHNICAL REPORT DISTRIBUTION LIST, 051A

	No. Copies
Dr. M. A. El-Sayed University of California, Los Angeles Department of Chemistry Los Angeles, California 90024	1
Dr. M. W. Windsor Washington State University Department of Chemistry Pullman, Washington 99163	1
Dr. E. R. Bernstein Colorado State University Department of Chemistry Fort Collins, Colorado 80521	1
Dr. C. A. Heller Naval Weapons Center Code 6059 China Lake, California 93555	1
Dr. J. R. MacDonald Naval Research Laboratory Chemistry Division Code 6110 Washington, D.C. 20375	1
Dr. G. B. Schuster University of Illinois Chemistry Department Urbana, Illinois 61801	1
Dr. E. M. Eyring University of Utah Department of Chemistry Salt Lake City, Utah 84112	1
Dr. A. Adamson University of Southern California Department of Chemistry Los Angeles, California 90007	1
Dr. M. S. Wrighton Massachusetts Institute of Technology Department of Chemistry Cambridge, Massachusetts 02139	1
Dr. M. Rauhut American Cyanamid Company Chemical Research Division Bound Brook, New Jersey 08805	1
Dr. J. I. Zink University of California, Los Angeles Department of Chemistry Los Angeles, California 90024	1
Dr. B. Schechtman IBM San Jose Research Center 5600 Cottle Road San Jose, California 95143	1
Dr. John Cooper Code 6130 Naval Research Laboratory Washington, D.C. 20375	1

TECHNICAL REPORT DISTRIBUTION LIST, 051B

	No. Copies
Professor K. Wilson Department of Chemistry, B-014 University of California, San Diego La Jolla, California 92093	1
Professor C. A. Angell Department of Chemistry Purdue University West Lafayette, Indiana 47907	1
Professor P. Meijer Department of Physics Catholic University of America Washington, D.C. 20064	1
Dr. S. Greer Chemistry Department University of Maryland College Park, Maryland 20742	1
Professor P. Delahay New York University 100 Washington Square East New York, New York 10003	1
Dr. T. Ashworth Department of Physics South Dakota School of Mines & Technology Rapid City, South Dakota 57701	1
Dr. G. Gross New Mexico Institute of Mining & Technology Socorro, New Mexico 87801	1
Dr. J. Kassner Space Science Research Center University of Missouri - Rolla Rolla, Missouri 65401	1
Dr. J. Telford University of Nevada System Desert Research Institute Lab of Atmospheric Physics Reno, Nevada 89507	1
Dr. B. Vonnegut State University of New York Earth Sciences Building 1400 Washington Avenue Albany, New York 12203	1
Dr. Hank Loos Laguna Research Laboratory 21421 Stans Lane Laguna Beach, California 92651	1
Dr. John Latham University of Manchester Institute of Science & Technology P.O. Box 88 Manchester, England M601QD	1

SP472-3/B1

472:GAN:716:lab
78u472-608

TECHNICAL REPORT DISTRIBUTION LIST, 051C

	No. Copies
Dr. M. B. Denton Department of Chemistry University of Arizona Tucson, Arizona 85721	1
Dr. R. A. Osteryoung Department of Chemistry State University of New York at Buffalo Buffalo, New York 14214	1
Dr. B. R. Kowalski Department of Chemistry University of Washington Seattle, Washington 98105	1
Dr. S. P. Perone Department of Chemistry Purdue University Lafayette, Indiana 47907	1
Dr. D. L. Venezky Naval Research Laboratory Code 6130 Washington, D.C. 20375	1
Dr. H. Freiser Department of Chemistry University of Arizona Tuscon, Arizona 85721	
Dr. Fred Saalfeld Naval Research Laboratory Code 6110 Washington, D.C. 20375	1
Dr. H. Chernoff Department of Mathematics Massachusetts Institute of Technology Cambridge, Massachusetts 02139	1
Dr. K. Wilson Department of Chemistry University of California, San Diego La Jolla, California	1
Dr. A. Zirino Naval Undersea Center San Diego, California 92132	1
Dr. John Duffin United States Naval Postgraduate School Monterey, California 93940	1
Dr. G. M. Hieftje Department of Chemistry Indiana University Bloomington, Indiana 47401	1
Dr. Victor L. Rehn Naval Weapons Center Code 3813 China Lake, California 93555	1
Dr. Christie G. Enke Michigan State University Department of Chemistry East Lansing, Michigan 48824	1
Dr. Kent Eisentraut, MBT Air Force Materials Laboratory Wright-Patterson AFB, Ohio 45433	1
Walter G. Cox, Code 3632 Naval Underwater Systems Center Building 148 Newport, Rhode Island 02840	1
Professor Isiah M. Warner Texas A&M University Department of Chemistry College Station, Texas 77840	1
Professor George H. Morrison Cornell University Department of Chemisty Ithaca, New York 14853	1
Professor J. Janata Department of Bioengineering University of Utah Salt Lake City, Utah 84112	1
Dr. Carl Heller Naval Weapons Center China Lake, California 93555	1

TECHNICAL REPORT DISTRIBUTION LIST, 051C

	No. Copies
Dr. L. Jarvis Code 6100 Naval Research Laboratory Washington, D.C. 20375	1


ATE
LMED