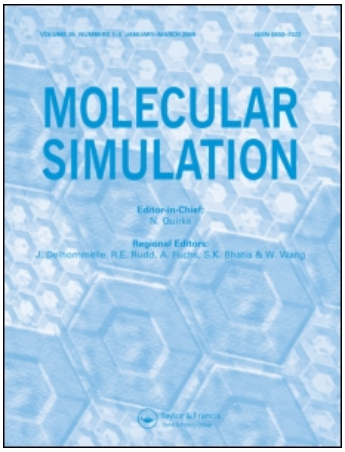

Molecular Simulation
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713644482

Molecular Design with Transparallel Supercomputers

David N. J. White[a]; J. Noel Ruddock[a]; Paul R. Edgington[a]
[a] Chemistry Department, The University, Glasgow, Scotland, U.K.

To cite this Article White, David N. J. , Ruddock, J. Noel and Edgington, Paul R.(1989) 'Molecular Design with Transparallel Supercomputers', Molecular Simulation, 3: 1, 71 — 100
To link to this Article: DOI: 10.1080/08927028908034620
URL: http://dx.doi.org/10.1080/08927028908034620

Molecular Simulation, 1989, Vol. 3, pp. 71–100
Reprints available directly from the publisher
Photocopying permitted by license only

MOLECULAR DESIGN WITH TRANSPARALLEL SUPERCOMPUTERS†

DAVID N.J. WHITE, J. NOEL RUDDOCK, and PAUL R. EDGINGTON

Chemistry Department, The University, Glasgow G12 8QQ Scotland, U.K.

(*Received January, 1988; in final form April, 1988*)

Transputer-based parallel ("Transparallel") computers are capable of dealing locally with the majority of computationally intensive tasks associated with small and macromolecular design which would otherwise require access to a remote supercomputer. In addition computations traditionally executed on a laboratory minicomputer can be made to run one to two orders of magnitude faster. However parallel computing is still in its infancy and the software tools required for effective parallel programming have an extended development period ahead of them.

This paper begins with a brief introduction to parallel computers, the transputer and occam; describes the continuing development of a transparallel supercomputer; the molecular modelling software which has been, and will be, developed to run on it; and some preliminary results from a force field development study.

In particular; a parallel molecular mechanics (MM) program has been implemented on a network of transputers. The configuration of transputer networks for scientific computation, the programming of concurrent parallel processes, and the overall performance of the hardware/software ensemble are discussed; and a program for the automatic optimization of MM force constants has been implemented on a single transputer and is in the process of being "parallelized". The algorithms used and some preliminary results are described.

KEY WORDS: Molecular modelling, molecular mechanics, force fields, biological molecules, transputers, parallel processing

PARALLEL COMPUTERS — INTRODUCTION

Whilst remote supercomputers are a suitable vehicle for some computational chemistry codes, this remoteness may preclude their effective use in situations where interactive codes are necessary. Laboratory scale parallel processing appears to offer a suitable price/performance ratio for the large scale scientific computations involved in molecular modelling and other interactive simulation techniques. However, until fairly recently technical obstacles with hardware, lack of parallel languages, and high cost ensured that parallel computers remained a development laboratory curiosity.

The first large grain parallel (as opposed to the fine grain parallelism of vector processors) computer to appear in the commerical marketplace was the ICL DAP which was based on a square array of simple processors. The machine was initially composed of a 64 × 64 array of single bit processors which simultaneously executed identical instructions on a 64 × 64 array of data. The machine was therefore SIMD (Single Instruction Multiple Data) and could be programmed either in assembler or a non-standard superset of fortran 77, which implemented features to allow parallel operations on vectors and/or arrays on data. These machines had to be hosted by an ICL mainframe computer, which made them very expensive, and there was no high

†Invited paper.

speed interface to a computer graphics display, which precluded their use for general purpose molecular modelling. The ICL machines are no longer manufactured, but AMT have produced a successor to the DAP which is compact, has an integral high speed graphics display, and is priced at the level of a medium sized minicomputer. The Physics Department at Edinburgh University have made good use of this architecture for simulation studies [1].

The first laboratory-sized (and priced!) parallel processors were MIMD (Multiple Instruction Multiple Data) and composed of multiple interconnected 16 or 32-bit microprocessor systems. These currently come in two flavours. The "straight" machines contain 16 or more processor nodes; where each node typically consists of a 16/32 bit microprocessor (CPU), often associated with a separate single chip floating point unit (FPU), 128 kbytes or more of memory, and a number of channels to communicate with other nodes; connected together in a fixed (usually) hypercube topology. The floating point performance of this variant depends on the CPU or single chip FPU. Examples of this architecture include the Intel iPSC Hypercubes, the Sequent Balance, the Alliant FX series, and the Ametek System 14 [2].

The "turbo" machines incorporate a small vector processor at every node in addition to the CPU/FPU. Machines, such as the Intel iPSC-VX series or the FPS T series, with 8 or more nodes, are obviously capable of very respectable floating point performance [3].

Although excellent in almost all respects these MIMD machines do have some shortcomings. They are physically quite large, filling 1–4, or more, five feet high 19" racks, and require some care to program effectively. Not only does the overall program need to be split into a number of independent tasks capable of running concurrently, but each task needs to be "vectorized" for efficient execution on the vector processor. This latter task is best performed manually for frequently used programs. The fixed hypercube topology can lead to processors lying idle; one can certainly embed any useful interconnect topology into a hypercube of high enough dimension, but this can leave numbers of unused processor nodes during a given calculation.

If a number of the discrete components involved in the above MIMD architecture could be integrated into a single VLSI chip the physical size of a parallel computer with a useful number of nodes could be reduced to desktop dimensions. For instance the CPU, floating point unit, and the communications channels might be placed on a single chip. Because fewer parts are then involved the mean time between failtures of a machine based on such a chip would be lower, as would the manufacturing cost. The utility of such a chip would be greatly enhanced if the node interconnexion topology were programmable. Software too would require attention. Ideally a simple language which coped naturally and effectively with concurrency is required, together with fortran 77/8X, pascal, and C for the "diehard" traditionalists. Mixed language programming should also be possible.

There are currently two ways in which one can come close to this ideal. The VLSI components incorporated in the NCUBE range of parallel processors integrate a 32-bit processor, a memory interface, and 11 1 Mbyte/sec communications links on a single chip [4]. The NCUBE chips are not individually available, the memory interface is only 16-bits wide, and the chips do not incorporate an FPU. The second option is the Inmos transputer.

THE TRANSPUTER

The Inmos T800 transputer is a 32-bit RISC microprocessor with 4096 registers (or a 4 kbyte cache memory, or 4 kbytes of 50 nS local memory, depending on one's viewpoint), a 64-bit floating point unit, 4 2 Mbyte/sec serial interconnecting links, a dynamic memory interface, and a timer all incorporated on a single chip [5]! A schematic diagram of the Inmos T800 is shown in Figure 1. Transputers are commercially available at the component level in the same way as ordinary microprocessors.

Viewed as a straightforward merchant microprocessor the T800-20 (20 Mhz) transputer is undisputably the fastest device available on the world open market – by quite a considerable margin! The T800-20 executes 4.6 million Whetstone benchmark (a mix of data processing and floating point operations representative of scientific computations) operations per second and 1.5 Mflops (millions of floating point operations/sec) sustained on loop 7 of the Livermore Loops benchmark. This performance is compared with that of other microprocessors and minicomputers in Table 1. The T800 is only one of a family of transputers and other devices intended to support parallel processing. Ths T414 is identical to the T800 in every respect save the absence of an on-chip floating point unit and only 2 kbytes of 50 ns on-chip memory rather than 4 kbytes. But as we shall see the T414 is still a formidable computing engine and modest assemblies of processors better superminis both in absolute performance and price/performance for scientific computing. Two speed variants of the T414 are available: 15 and 20 Mhz. The T212 is a 16-bit variant of the T414 which

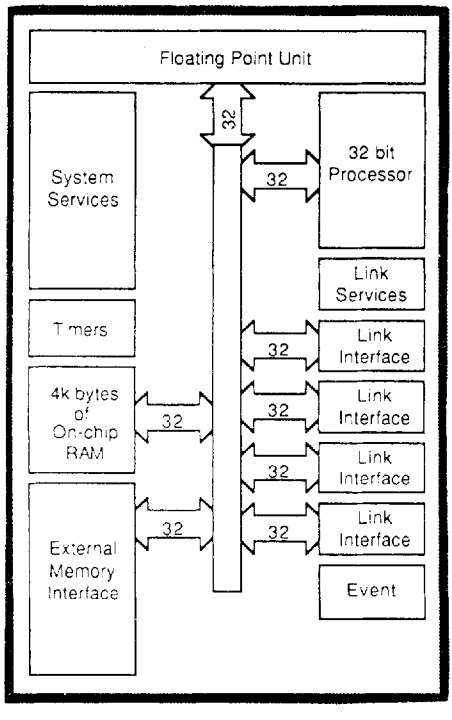

Figure 1 A schematic diagram of the Inmos T800 transputer.

Table 1 Performance of the Inmos T800-20 transputer in comparison with other microprocessors and minicomputers.

Processor	*Clock speed*	*Whetstones/sec*
IMS T800-20	20 Mhz	4600K
80386/WTL 1167	16 Mhz	3600K
Fairchild Clipper	33 Mhz	2220K
Intel 80386/80387	16 Mhz	1354K
VAX-11/780 FPU	n/a	1152K
MC 68020/68881	16.67 Mhz	1028K
ATT 32000/32100	n/a	1000K
NS 32332/32081	15 Mhz	728K
IMS T414-20	20 Mhz	663K
Intel 80286/287	8 Mhz	300K

only addresses 64 kbytes of external memory (the T800 and T414 address 4 Gbytes) and is not suitable for large scale scientific computation. The M212 is a version of the T212 with complete on-chip floppy and hard-disc interfaces, together with a simple on-chip operating system in read-only memory (actually a library of operating system disc primitives). The M212 makes implementing parallel database-searching systems very straightforward in addition to its more mundane uses. Different types of transputer may communicate transparently with each other over their links.

Transputers are not designed to be used singly, they are intended to be connected together in large numbers as the nucleus of powerful parallel processors. It is frequently desirable to alter the pattern of link interconnections (i.e. the network topology) between a group of transputers depending on the particular program that is being executed. This may be achieved by replugging a set of sockets, much in the manner of a manual telephone exchange, but this is very tedious (if cheap!) and to be avoided if possible. Inmos provide a 32 × 32 single chip crossbar switch, the C004, which operates under software control and allows up to 8 transputers to be interconnected in any arbitrary topology. If more than 8 transputers are to be interconnected then 2 or more COO4s are required. Some common network topologies are shown in Figure 2.

OPERATING SYSTEMS & COMPILERS

Until fairly recently all transputer based systems, whether using a single or multiple transputers, required a host computer, primarily to provide filestore facilities. The IBM pc/xt or /at running msdos is by far and away the most popular host computer, followed by the Dec vax range running vms, the Sun workstation range running unix, and any Stride computer running the p-system. Some versions of the Dec and Stride systems relied on an rs-232 serial connection between the host computer and transputer(s) and these were necessarily somewhat slow; in all other cases the host/transputer connection is a fast parallel interface.

Stand-alone host-based systems

Host-based systems can be used in two ways; either with "stand-alone" language processors, or with the Inmos Transputer Development System (TDS or tedious if you say it quickly).

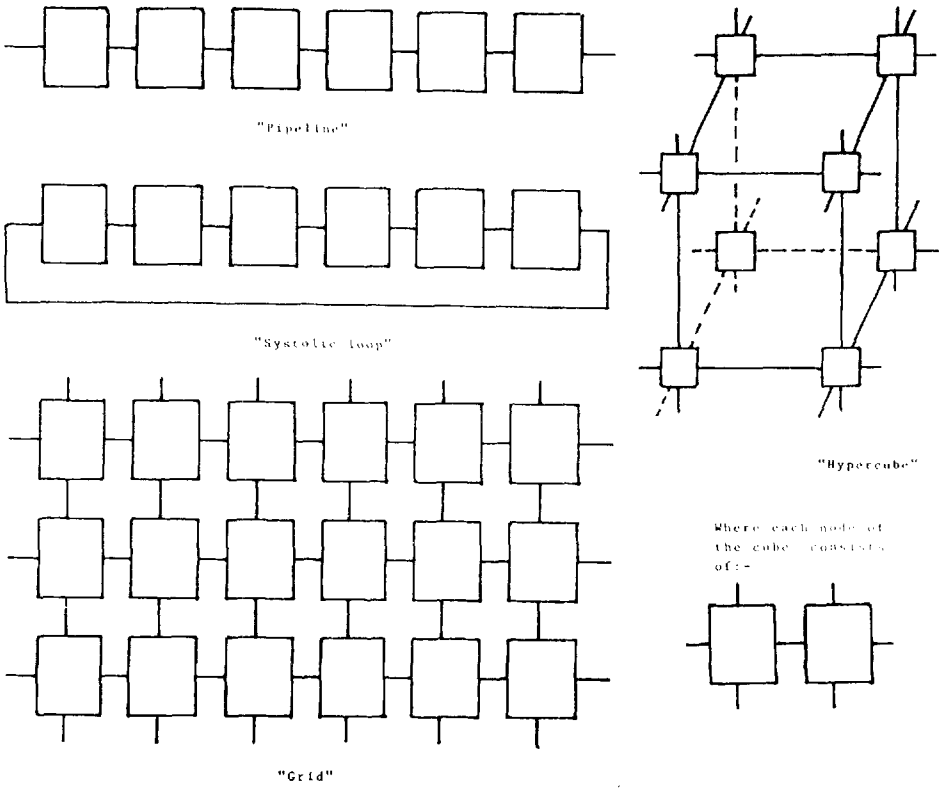

Figure 2 Some common network topologies.

There are stand-alone compilers for fortran-77, pascal, c, and occam; these are used in exactly the same way as the native compilers on the host machine in an edit-link-run cycle — so that your favourite editor and other software tools are still available. The drawback to stand-alone mode, is that in the absence of a configurer the compiled code can only run on a single transputer (this is not strictly true, a little ingenuity and a C004 can circumvent the problem).

Despite the fact that advertisments in scientific and other journals would suggest a plethora of sources for transputer compilers these are all ultimately traceable back to three sources: at present Inmos of Bristol are the only source of occam compilers; 3L of Livingston produce fortran-77, pascal, and c compilers which are distributed by Inmos and a large number of transputer systems manufacturers worldwide; and Topexpress of Cambridge produce the fortran-77 compiler which is distributed (exclusively at present) by Meiko. The 3L c and Topexpress f-77 transputer compilers are syntactically almost identical to the compilers that those same companies produced for the Acorn Cambridge workstation (an opportunity to "try-before-you-buy" if you have a Cambridge).

As most scientific codes are written in fortran the f-77 compilers are obviously of most interest. There appears to be little to choose between the two alternatives as both of them incorporate language extensions to make porting code from a vax very

straightforward, and the efficiency of the generated code is comparable (the last time the author made a comparison, code generated by the 3L compiler ran about 10% faster than the Topexpress code. Interestingly enough the 3L code also ran marginally faster than the same algorithm coded and optimised in occam and then run through the Inmos compiler. However all of these compilers are still being actively developed and the situation may well be different by the time that this paper is published).

Parallel host-based systems

Very recently 3L have introduced parallel c, fortran-77, and pascal compilers. The parallel processes are written entirely in the language of choice and the TDS is not required to construct multi-transputer programs. The compilers support concurrent tasks, multi-threading, semaphores, transputer channel I/O with timeouts, and an object file disassembler. As with occam under the TDS, an application can be developed on a single transputer and then run on a transputer network for increased performance.

There are two configurer utility programs. The first creates an executable file from independent tasks to be run in parallel, whilst giving the programmer complete control over placement of the tasks on physical processors. Changes to the network topology only requires updating of a simple configuration file — not recompilation and relinking as is required with the TDS.

The second configurer works out the topology of the transputer network for itself and then automatically floodfills the whole network with copies of a single flask. This is the so called "processor farm" approach where identical copies of the single task each operate upon a separate data set (e.g. a number of energy minimization calculations each on different molecules).

This approach allows very straightforward parallel implementation of already existing sequential fortran programs and will therefore by very popular.

THE TDS

The TDS may be used in order to develop parallel programs, which will run distributed across a network of transputers, on a host-based system. The TDS provides a complete environment for transputer program development and access to all facilities is through a full screen structured editor [6]. Programs may be written entirely in occam, or processes written in "alien languages" such as fortran, pascal, or c may be embedded in an occam harness.

The editor is based around a heirarchical collection of constructs called folds which may be nested to any depth. Folds may contain either text, program source, or executable code. Folding is analogous to taking a document with page headers written across the creases of fanfold paper and then folding it so that the text is hidden but the page headers are still visible.

A fold is rather like a macro in that a single line fold can contain subfolds each of which may contain other folds. In the same way as macros (text) folds may be expanded for editing and then collapsed to make the overall structure clearer. Programs can then be written by first laying out the major procedures as a series of empty folds to get the overall structure correct. A series of empty subfolds may then be written into each of the major folds, and so on, until the program is broken down into

sufficiently small fragments for real code to be written into the bottom level folds. This is illustrated in the following simple example: a series of collapsed folds might look like this:-

```
... Livermore loop 6
... Livermore loop 7
... Livermore loop 8
```

If the Livermore loop 7 fold (a fold marker is a string of five characters the first three of which are ...) is entered it might contain the following occam code:-

```
{{{ Livermore loop 7
  SEQ k = 0 for n
    x[k] :=          u[k]    + (( (r*(z[k]       + (r*y[k])))            +
             (t*((u[k + 3] +    (r*(u[k + 2] + (r*u[k + 1])))))  ) +
             (t*((u[k + 6] +    (r*(u[k + 5] + (r*u[k + 4])))))  )
}}}
```

An open fold is delimited by {{{ }}} and the comment on the closed fold line is inserted immediately after the opening {{{ on the same line. Folds frequently contain complete procedures and their associated declarations, abbreviations, and constants, but this is not a mandatory requirement. Notice that folds are not a feature of occam, only the Inmos TDS implementation of it.

All of the system utilities of the TDS; such as the occam syntax checker, occam compiler, linker, and configurer; are contained within executable folds and accessed in a similar fashion to program source folds (although they cannot be edited, only invoked).

The TDS is fine in theory but in practise there are some irritations. Folds are accessed by moving the screen cursor with keyboard cursor direction keys and then pressing function keys to operate on the folds. This is very tedious and the whole process would, in the authors opinion, be much more convenient if operated by a mouse rather than the keyboard. The other drawback to the TDS is that there is no way of getting from fold to fold by specifying a path. If the user is at the bottom level of a large branch and wishes to get to the bottom level of another large branch, the only option is repeated application of the "exit fold" function key until the bough or trunk is reached followed by repeated application of the "enter fold" function until the bottom level of the new branch is reached. Overall the TDS is quite a good system once the user has come to terms with its unfamiliarity.

NATIVE OPERATING SYSTEMS

At the time of writing (Jan '88) two native transputer operating systems (OS) have just appeared. Microport Systems have developed a distributed, concurrent version of Unix System V to run on an array of transputers: where the unix pipes are mapped onto transputer links; and tasks or processes instead of being timeshared on a single processor are farmed out so that one transputer executes one task in parallel with all of the others. Few other details are available at present.

The second native OS promises to have a much greater impact because it is targetted at the mass market by a major manufacturer of personal computers. The entry level Atari Abaq consists of a T800 transputer, 4 Mbytes of main mercury, a 1280 × 960 resolution colour graphics display, a 40 Mbyte hard disc, a floppy disc,

68000 I/O processor, keyboard, mouse, three internal expansion slots, and the helios operating system from Perihelion Software [7].

One of the helios (the name will probably be changed as it has been previously registered as a trademark) user interfaces is almost a facsimile of the unix c-shell, although helios is not a version of unix. The alternative user interface is x-windows vii a "point-and-push" graphical interface using pull-down menus and a mouse pointing device. Helios supports three models of parallelism: the running of a number of completely independent user programs on different transputers at the same time; the running of a parent and one or more cooperating child processes, written in conventional sequential languages such as f-77, pascal, or c, located on different transputers and interconnected by pipes; and the running of programs written entirely in parallel programming languages (currently only occam). Perihelion claim to support the f-77, c, pascal, lisp, bcpl, and occam programming languages. Probably the most important feature of helios is that it will not be restricted to Atari machines, but will be widely available for a range of machines at modest cost (currently 500 pounds including a c compiler).

OCCAM

Occam is a language with a small number of primitive features which may be combined in an infinity of diabolically cunning permutations [8]. Becoming an occam hacker is simple, becoming a virtuoso is very difficult for the average physical scientist. Occam has a lot in common with pascal, and to a lesser extent c, but is very different from the scientific lingua franca — fortran-77.

Occam programs are built from three primitive processes:-

```
v := e   assign expression e to variable v
c !  e   output expression e to channel c
c ?  v   input from channel c to variable v
```

The primitive processes are combined to form constructs:-

```
SEQ components executed sequentially
PAR components executed in parallel
ALT component ready first is executed
```

A construct is itself a process and may be used as a component of another construct. All keywords in occam are reserved words and must always be typed in upper case. identation is mandatory in occam and makes statements like END DO and END IF redundant. This is illustrated by the following program fragment which consists of two parallel processes:-

```
PAR
  c1 ? x
  SEQ
    c2  ?  y
    y  := y + 1
    c3  !  y
```

The first process inputs from channel c1 to variable x whilst the second process executes simultaneously. The second process is a construct consisting of three primitive processes which will be executed sequentially. The second process inputs from

channel c2 to variable y, adds one to it, and outputs the result on channel c3. Conditional execution is provided by the IF statement which performs a number of tests in sequence, and executes only the first one which is true:-

```
IF
    x > 0
      z := z + 2
    x = 0
      z := z - 1
    x < 0
      SKIP
```

This example will have the following effect: if x is greater than zero then two is added to z, otherwise if x equals zero one is subtracted from z, but if x is less than zero do nothing. An alternative construct illustrates another possibility:-

```
IF
    x >= 0
        c1 ? v
    TRUE
        c2 ? v
```

In this case if x is greater than or equal to zero channel c1 inputs into variable v otherwise channel c2 inputs into variable v. Occam supports multidimensional arrays, the following example illustrates the use of a singly dimensioned array:-

```
[20] INT x :
PAR
  c1 ? [ x FROM  0 FOR 10]
  c2 ? [ x FROM 10 FOR 10]
```

x is declared as a 20 element array of integers and then channel c1 inputs into x[0] to x[9] and whilst channel c2 simultaneously inputs into x[10] to x[19]. Occam also allows abbreviations to represent array segments (amongst other things):-

```
a IS [ x FROM 10 FOR 10]:
```

this example identifies a[0] with x[10] through a[9] with x[19]. This is more than mere convenience, because if x is a global variable, a can be made local to the process incorporating above abbreviation. The transputer operates on local data much more quickly than on global data. The replicated constructor should also be illustrated here:-

```
PAR i = 0 FOR 3
  pass.along(in[i],out[i])
```

This invokes three instances of the process pass.along in parallel. The first uses the channels in[0], out[0]; the next in[1], out[1]; and the last in[2], out[2]. All three processes execute simultaneously.

Concurrent or parallel processes communicate only through occam channels and synchronization occurs as a consequence of the requirement that both the outputting and inputting process must be ready before a transfer takes place. If the PAR construct is used then "parallel" processors are timesliced on the same transputer and

computer via memory mailboxes which simulate transputer links. If on the other hand the PLACED PAR construct is used in conjunction with the PLACE channel AT link. address construct the processes will execute on different transputers and communicate through hardware links. The following example illustrates this point:-

```
... SC first.process
... SC second.process
DEF link0.out=0,link0.in=4,link1.out=1,link1.in=5:
CHAN c1,c2:
PLACED PAR
  PROCESSOR 0 T8
    PLACE c1 AT link0.out:
    PLACE c2 AT link0.in:
    first.process(c1,c2)
  PROCESSOR 1 T8
    PLACE c1 AT link1.in:
    PLACE c2 AT link1.out:
    second.process(c1,c2)
```

Here we have two T800 transputers with transputer0 running first.process and transputer1 running second.process. The link0 output on transputer0 is joined to the link1 input of transputer1, and the link0 input of transputer0 is joined to the link1 output of transputer1 enabling the two processors to communicate.

Occam code is transputer independent, assuming enough memory, and may be transferred between the T212, T400, and T800 merely by changing the PROCESSOR declaration at the head of the program from T2, to T4, or T8. Furthermore large occam programs designed to run on large transputer arrays or networks may be written, debugged, and tested on a single transputer. The hardware links of the real transputer array are simulated by software channels, and the computational processes timeshared on the single transputer rather than distributed across the array. The single transputer development program behaves exactly the same as the "real thing" and may be transferred onto the network merely by changing a few lines in the configuration section of the program.

This means that the large transputer array can be reserved entirely for production work with program development taking place on multiple host computers (or a single timesharing host) where each user has permanent access to a single transputer, associated memory, and the software development tools. Any workstation may be software switched into the transputer array as desired, assuming that the array is free, or the array may be software configured to give each user exclusive access to a number of transputers in the array.

HARDWARE

The host computers used for all of the work desribed in this paper are either IBM pc/xts or a pc/at clone using the Intel 80386 processor. The xts consist of an Intel 8088 with 640 kbytes of memory, a 20 mbyte winchester disc, and a colour graphics adapter running pcdos v3.2. The at clone contains a 32-bit 80386 with 1 Mbyte of memory, a fast 40 Mbyte hard disc, a 40 Mbyte tape cartridge, and a monochrome screen running pcdos v3.3.

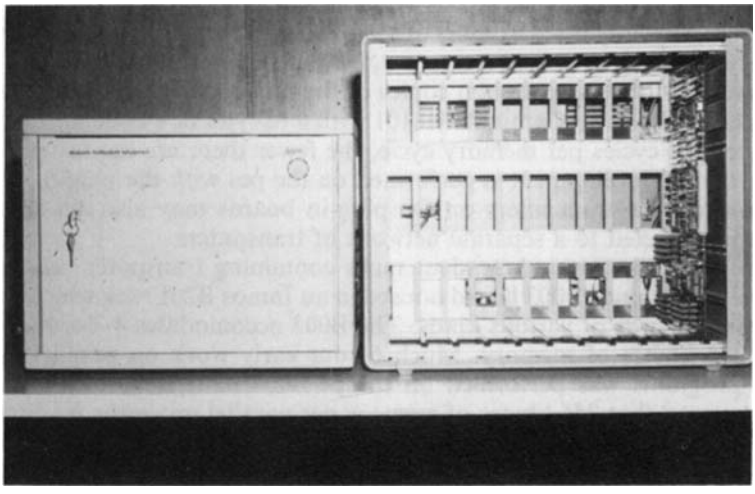

Figure 3 The triple-eurocard 19" rack contains a power supply and can house up to ten 220 × 366 mm circuit cards.

The pcs and pcdos act as a fileserver for the Inmos transputer development system (TDS) running the occam 2 compiler with its folding editor. The files produced by the TDS are not directly readable by the pcdos system because of the extra information required and inserted by the folding editor, but Inmos provide utilities for converting source code back and forth between pcdos and TDS format. The pcs also run stand-alone versions of the Lattice Logic C and f-77 compilers.

A plug-in board in each of the pcs contains a single T414 or T800 transputer with varying amounts of memory and these actually host the TDS plus compilers. The

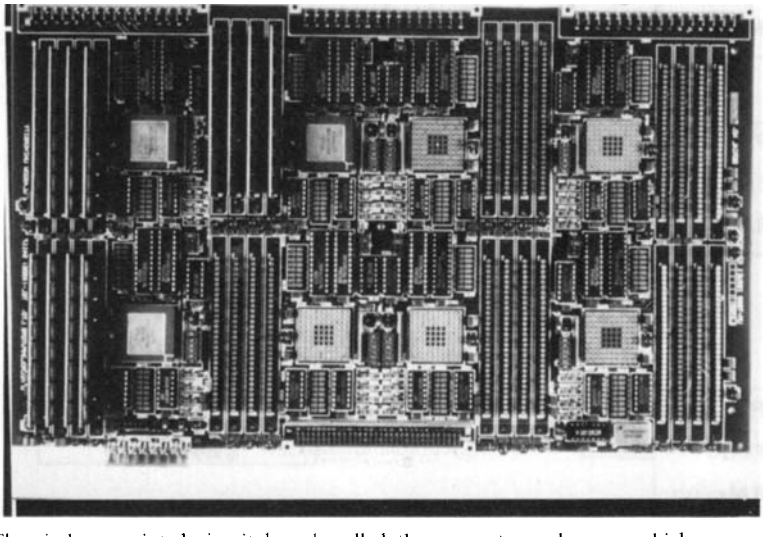

Figure 4 The six-layer printed circuit board, called the compute card, upon which are mounted 8 transputers each having 4 Mbytes of 4 cycle dynamic memory.

transputers are connected to the pc bus via a transputer link and Inmos link adaptor chip. There are currently three flavours of plug-in board in use; the Inmos Board B004 with 2 Mbytes of 5 cycle dynamic memory, a MicroWay monoputer with 2 Mbytes of 4 cycle memory (these are carbon copies of the non-copyrighted B004 design at a slightly lower price), and a Gemini GM8101 with 8 Mbytes of 4 cycle memory (cycles refer to processor cycles per memory cycle, the fewer there are the faster programs run). All program development is performed on the pcs with the plug-in transputer boards. However the transputers on the plug-in boards may also act as a master device when connected to a separate network of transputers.

There are currently two independent racks containing transputer networks. The first consists of an Inmos B003 board housed in an Inmos B201 rack which can house up to ten circuit cards of various kinds. The B003 accomodates 4 T414 transputers each with 256 kbytes of memory. Much of our early work on parallel molecular mechanics programs was performed on the pc/B004/B201/B003 ensemble but we rapidly discovered that 256 kbytes of memory per parallel processor was inadequate for molecules containing more than 100–150 atoms. Unfortunately at that time (end of 1986) none of the commercially available transputer boards or machines intended for concatenation into large networks accommodated more than 1 Mbyte of memory

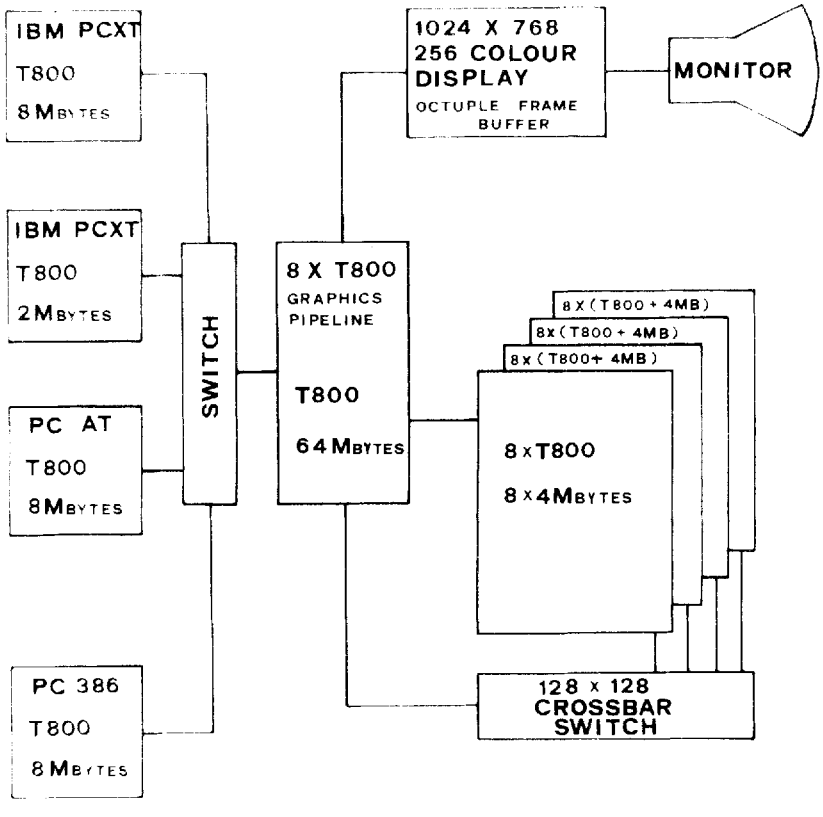

Figure 5 A schematic diagram of the planned transparallel supercomputer.

per node, and this was insufficient for the work that we envisaged. (We could have used multiple B004s but these were extremely expensive then and the pc bus interface cfor concatenation into large networks accommodated more than 1 Mbyte of memory per node, and this was insufficient for the work that we envisaged. (We could have used multiple B004s but these were extremely expensive then and the pc bus interface circuitry would have been redundant on all but the first board.) Encouraged by Inmos and financed by Ciba-Geigy Industrial Chemicals we decided to design our own transparallel supercomputer system.

The second transputer network in use is therefore to our own design. The triple-eurocard 19" rack contains a power supply and can house up to ten 220 × 366 mm circuit cards (Figure 3). At the present there is one type of circuit card operational. This is a six-layer printed circuit board, called the compute card, upon which mounted 8 transputers (T800s, T414s, or a mixture) each having 4 Mbytes of 4 cycle dynamic memory (Figure 4). The rack currently houses one compute card. A workstation card designed to fit into this rack will accomodate a single master T800 with 64 Mbytes of 3 cycle memory plus an octuply-buffered 1024 × 1024 resolution × 256 colour graphics display driven from the master T800 via a graphics transformation pipeline of 8 T800s. When complete by the end of 1988 the transparallel supercomputer will be as shown in Figure 5.

MOLECULAR MECHANICS

Molecular mechanics calculations are a prime candidate for implementation on transparallel processors because of the very long program run times associated with computations on macromolecules, ensembles of large numbers of small molecules, and multiple independent small molecules. The question here however is whether it is possible to implement efficient parallel codes for MM calculations. As we shall subsequently show the answer is affirmative, but first a brief discussion of some of our applications of MM calculations and the need for good general purpose force-fields.

Molecular mechanics calculations play an important role in our protein model building studies [9] where they are used to anneal sections of the polypeptide chain after insertions or deletions. It would also be desireable to energy minimize the whole molecule after model building to iron out non-local changes in structure between the starting and target proteins. However the latter minimizations may do more harm than good because of deficiencies in currently available force-fields (energy minimization of a known protein crystal structure frequently results in significant differences between starting and minimized structures — how much of this is due to imperfections in the molecular mechanics calculations and how much due to inaccuracies in the crystal structure has never been properly investigated). MM calculations on macromolecules and large assemblies of smaller molecules have traditionally been performed on a laboratory minicomputer, where they are extremely time consuming.

Artificial enzyme substrates have a much greater diversity of functional groups than polypeptides, proteins, and nucleic acids but we want our molecular mechanics calculations to deal adequately with all of these; as well as coping with diverse polymers, water treatment chemicals, corrosion inhibitors, dyestuffs, etc. and the interactions of these molecules with other molecules. In the normal course of events this would mean constructing a force-field composed of thousands of parameters — an almost impossible task.

There are therefore two major problems extant; the first is to reduce the run time

of large MM calculations to acceptable proportions which can be achieved by parallel processing with transputers; the second is to develop a good general purpose force-field with the minimum of parameters.

The majority of currently popular MM force-fields are developed on a trial-and-error basis [10] but this is extremely time consuming and acts as a deterrent to improving force-fields. The alternative force-constant optimization procedure such as that employed by Lifson and coworkers in the development of the consistent force-field (CFF) [11]. The drawback to computational optimization is the tremendous amount of computer time required, typically 30 days of VAX-11/750 time per new functional group. Fortunately computational optimization parallizes very well and is a natural candidate for execution on a transparallel supercomputer. The way in which the number of force-field parameters is reduced will become clear during the description of the potential functions and force-constants.

POTENTIAL FUNCTIONS

The molecular potential energy, or steric energy, may be written as:-

$$V_s = V_l + V_\theta + V_\omega + V_\chi + V_r + V_q$$

where V_l represents the molecular potential energy due to bond stretching, and V_θ, V_ω, V_χ, V_r, V_q are the corresponding terms for angle bending, bond torsion, out-of-plane bending, nonbonded interactions, and coulombic interactions respectively.

The potential function for bond stretching is very straightforward and assumes that the bonds exhibit simple harmonic restoring forces on perturbation:-

$$2V_l = \sum_l k_l(l - l_0)^2$$

where k_l is the stretching force constant and l_0 is the reference bond length.

The angle bending potential function is a little more complex, primarily because the force constants involved are sufficiently small as to allow large deviations from the reference bond angles. This in turn means that the simple harmonic approximation is no longer valid, and an anharmonicity correction term has to be added to the potential function. The correction term is usually cubic in $\Delta\theta$, but a slightly more complex variant is employed here:-

$$2V_\theta = \sum_\theta k_\theta(\Delta\theta^2 - k'_\theta(|\Delta\theta^3| - 0.0004|\Delta\theta^5|))$$

$$\Delta\theta = \theta - \theta_0$$

where k_θ is the angle bending force constant and θ_0 is the reference bond angle. If only a simple cubic correction term is employed it can dominate the quadratic term for very large bond angle deviations from the reference value, so that further increases in deviation lead to a reduction in energy! The minimizer will then further increase the bond angle deviation from the reference value as this leads to a reduction in energy!. This situation could occur when attempting to start a MM calculation from a low-resolution protein x-ray structure or when "three-dimensionalizing" a sketched molecular structure. In order to remove this problem a term in $\Delta\theta^5$ has been added. The reason for choosing this particular extra term, other than the fact that it works, is that $\Delta\theta^5$ is efficiently calculated from $\Delta\theta^2$ and $\Delta\theta^3$, an important consideration when there are thousands of bond angles.

For the torsional potential energy a short Fourier series is used:-

$$2V_\omega = \sum_\omega [V_1(1 + s\cos\omega) + V_n(1 + s\cos n\omega)]$$

where the summation is over all torsion angles around all bonds, V_1 and V_n are the one-fold and n-fold components of the barrier to free rotation, ω is the torsion angle, n is its periodicity, and $s = +1$ for a staggered torsional energy minimum (e.g. ethane) and $s = -1$ for an eclipsed minimum (e.g. ethylene). V_1 is zero in most instances; but non-zero for the C-C-C-C torsion angle in an n-butane fragment, or the O=C'-N-H torsion angle of an amide linkage, for example.

In order to account for the increase in potential energy due to pyramidalization of trigonal planar systems such as $R_1R_2C{=}O$ or C'NHC a simple quadratic function of the following form is used:-

$$2V_\chi = \sum_\chi k_\chi(180 - \chi)^2$$

where k_χ is the force constant for out-of-plane bending and χ is an improper torsion angle in degrees. For example if we consider a carbonyl group and its substituents $R_1R_2C = O$, the improper torsion angle would be $R_1 - C = O..R_2$ and this is obviously 180 degrees for a planar system. If the system is pyramidalized by displacing the central Csp^2 atom either above or below the plane then the potential energy of the system will increase and χ will deviate from 180 degrees by an amount depending on the extent of distortion.

A Lennard-Jones potential of the kind:-

$$V_r = \sum_r [Ar^{-12} - Br^{-6}]$$

is used for the non-bonded interactions where the summation is over all 1,4 and higher unique pairwise non-bonded distances. There are different pairs of the constants A and B for interactions between pairs of different atom types (e.g. Csp...Csp, Csp^3...H, H...H, etc).

The Lennard-Jones potential is used in preference to the Buckingham potential for the following reasons. The Buckingham potential turns over at very short non-bonded distances and the energy of interaction decreases with decreasing distance — in a similar fashion to the quadratic plus cubic angle bending potential. Using the Buckingham potential on poor starting models can give problems unless appropriate computational fixups are made for short non-bonded distances — fixups are expensive in computer time when there are hundreds of thousands of non-bonded distances. The Lennard-Jones potential is free of this vice. The Buckingham potential contains an exponential term which takes around 20 times longer to compute than a floating point multiply on most computers. On the other hand the terms in the Lennard-Jones potential can be computed very economically:- the calculation of non-bonded distances gives r^2 directly, this can be squared to give r^4 and multiplied by r^2 to give r^6 for the B term, and then r^6 can be squared to give r^{12} for the A term. Use of the Buckingham potential would seriously slow down MM calculations on proteins relative to the same calculation performed with a Lennard-Jones potential.

Finally the coulombic interactions are accounted for by a pairwise summation over all of the non-bonded atomic monopole... monopole interaction:-

$$V = 332\sum_r qiqj/Dr$$

where qi and qj are the charges, in units of electronic charge, on the atoms i & j separated by a distance r. D is the dielectric constant and 332 converts the energy into

units of kcal per mole. There are two choices for D, one can either use a fixed value somewhere between 1 and 5, or one can use a distance dependent dielectric where D = r. The latter choice is to be preferred because the r squared term in the denominator obviates the need for a computationally expensive square root function call to get r, and also because there is some physical justification for a distance dependent dielectric. The value of D appropriate to a system consisting of two separated point charges in a vacuum is 1, and as matter is interspersed between the charges the appropriate value of D becomes greater than 1 — the greater the separation the greater the chance of interspersed matter and the higher the value of D. Alternatively it can be argued that a distance dependent dielectric goes some way towards accounting for solvation effects. The distance dependent dielectric corresponds to the tendency of the field lines connecting interior atoms in a molecule to spread into the high dielectric solvent region as the atoms are drawn apart.

FORCE CONSTANTS

It is obvious from the equation for V_s that a valence force-field (FF) is a cumbersome beast because of the large numbers of parameters (force-constants) required. For the purposes of MM calculations it is not sufficient to divide the atoms up into types H, C, N, O, F, P, etc., but rather H(aliphatic), H(aromatic), Csp, Csp^2, C(aromatic), Csp^3, Nsp^2, N(aromatic), N(nitro), Nsp^3, etc., so that around 30–40 different atom types are required to cope with a reasonable cross section of organic molecules. If there are 30 different atom types then approximately 300 values of k_l, l_0, A and B; plus around 2000 values of k_θ, k'_θ, and θ_0; plus some 10,000 values of V_n; are required for complete generality — a formidable and in fact impossible requirement. The force constants and reference geometrical parameters all need to be derived empirically from experimental data either by trial-and-error or by a computationally optimized fit. Even assuming that the necessary experimental measurements have been made (they haven't) the task of trial-and-error or (say) least-squares fitting to some 40,000 parameters is both computationally and practically impossible. Suppose that the impossible were achieved; then consider the absurdity of calculating (say) the structure and steric energy of ethane to define six quantities (two lengths, two angles, one torsion angle, and the energy) in terms of 40,000 parameters! The most general versions of commonly used valence FFs may not yet contain tens of thousands of parameters, but they certainly contain thousands.

Some attempts have been made to reduce the number of force-constants required but procedures employed to date have been generally unsatisfactory. The first and most obvious way to reduce the size of the FF is to employ fixed bond lengths so that no k_l or l_0 are required. This is quite a reasonable simplification in the majority of cases as bond lengths are hard variables (i.e. it requires a considerable input of energy to effect a small change in a bond length) and only alter significantly in situations of extreme steric strain. A half-way compromise is to allow the bond lengths to vary and assign appropriate values to l_0 but use the same value of k_l for all bond types. However neither of these strategies are of much benefit as the k_l and l_0 account for only a small proportion of the FF parameters.

Another approach which is frequently employed for MM calculations on polypeptides and polynucleotides is to use both fixed bond lengths and fixed bond angles; which both reduces both the number of FF parameters required and the number of

molecular geometric variables to be adjusted (which in turn reduces computation time). Fixing bond lengths, as we have seen, does not usually lead to problems. In the case of bond angles omission of the k_θ, k'_θ and θ_0 makes a significant reduction in the size of the FF but has the unfortunate side-effect of rendering the subsequent MM calculations unrealistic. This is because the strain due to short, nonbonded intra and/or intermolecular contacts can be partially relieved by changes in the soft variables, which includes bond angles, involving the atoms concerned in the steric compression. If bond lengths and angles are fixed, then the only way to alleviate repulsive non-bonded interactions is by changes in torsion angles. This gives rise to artificial molecular conformations. Torsion angles cannot, of course, be fixed to save on FF parameters; which is unfortunate because the associated force-constants constitute by far and away the largest portion of the FF.

The most satisfactory way of reducing FF size so far employed has been described independently by a number of workers [12] and employs one or moe of the following features. The bond angle and torsional FF parameters are divided into two classes, generic and special. For example all valency angles of the type R_1-Csp^3-R_2 would use a single set of generic k_θ, k'_θ, and θ_0 or all torsion angles of the type R_1-Csp^3-Osp^3-R_2 would use a generic set of V_1, V_n, s, and n. However for key parameters special force constraints would override the generic versions (e.g. for Csp^3-Csp3-H special k_θ, k'_θ, and θ_0 values would override the generic R_1-Csp^3-R_2 values). This approximation applied to valency angles and torsion angles can bring a generally applicable valence FF down to a manageable size without the problems associated with fixing bond lengths and angles. In the case of bond lengths an alternative approach to that mentioned previously is to use a full set of l_0 and a limited selection of k_l. The remaining k_l are calculated by linear interpolation on the basis of differences in bond length by assuming that k_l is proportional to $1/l_0$. A similar interpolation can be used to calculate values for V_n, although somewhat less satisfactorily. The barrier to free rotation around a bond is approximately inversely proportional to its length in many cases so that a set of reference lengths and key barriers can form the basis of a crude interpolation scheme.

The penalty paid with this overall approach is loss of precision in the calculated geometries and energies — an acceptable compromise in a number of cases.

Computation of Force-Constants

In the previous section the use of linear interpolation over a limited range of l_0 was discussed as a means of estimating values of $k_l(ij)$ (i.e. k_l for a bond between an atom of type i and another of type j) by assuming that $k_l(ij) \propto 1/l_0(ij)$. However if all of the published $k_l(ij)$ from current force-fields are plotted against $1/l_0^2(ij)$ for all bonds not involving hydrogen atoms; then, perhaps not surprisingly, the result is a very acceptable straight line. The same holds true for bonds involving a hydrogen atom, when considered separately. This observation suggests the possibility of using simple equations which are global in scope to provide values of $k_l(ij)$ from a table of $l_0(ij)$ for MM calculations. Satisfactory results are obtained from the two equations shown below:-

$$\tfrac{1}{2}k_l(ij) = [c1/l_0^2(ij)] + [c2/(l_0^2(ij) - 1)] + [c3/l_0(ij)]$$

$$\tfrac{1}{2}k_l(iH) = c4/l_0^2(iH)$$

The first equation applies when both atom types i and j are non-hydrogen, and the second when either atom types i and j (or both) are hydrogen. c1, c2, c3, and c4 are

constants whose values are obtained by adjustment to give the optimum fit between observed and calculated structures containing a wide diversity of bond types. Data on reference bond lengths $l_0(ij)$ are readily available and no difficulty was encountered in constructing a table which spanned all atom types of interest. These equations were tested by incorporating them into an existing force-field, the WBFF [13], in place of tables of individual $k_l(ij)$ values where they performed satisfactorily with no less of accuracy in the structures and energies calculated therefrom.

The application of a similar kind of approach to calculating the values of $k_\theta(ij)$ and $\theta_0(ijk)$ required to describe the contribution of angle bending energy to the total steric energy was a little more difficult and the resulting equations and rules are entirely empirical. Inspection of published k_θ values for valency angles composed of atom types i-j-k revealed two significant trends. The first of these is that $k_\theta(ijk)$ is highly correlated with $l_0(ij)$ and $l_0(jk)$ so that the shorter the distances the higher the force constant. The second trend is that $k_\theta(ijk)$ correlates with the differences in electronegativity $\Delta X(ij)$ and $\Delta X(jk)$. Again it proved possible to derive an equation which predicted force constants with sufficient accuracy that tables of explicit $k_\theta(ijk)$ could be dispensed with. The general form of the equation is:-

$$\begin{aligned}
\tfrac{1}{2}k_\theta(ijk) &= c5\{\, c6 + c7(|\Delta X(ij)| \\
&\quad + |\Delta X(jk)|)\,\}/l_0(ij)l_0(jk) \\
\tfrac{1}{2}k_\theta(ijH) &= 0.45*c5\{\, c6 + c7(|X(ij)| \\
&\quad + |\Delta X(jH)|)\,\}/l_0(ij)l_0(jH) \\
\tfrac{1}{2}k_\theta(HjH) &= 0.20*c5\{\, c6 + c7(|\Delta X(Hj)| \\
&\quad + |\Delta X(jH)|)\,\}/l_0(Hj)l_0(jH)
\end{aligned}$$

where c5, c6, and c7 are constants derived in the same way as c1–c4. In the case of bond angle bending values are required for $\theta_0(ijk)$ as well as $k_\theta(ijk)$. Again examination of published values almost always shows the pattern $\theta_0(ijk) > \theta_0(ijH) \approx \theta_0(Hjk) > \theta_0(HjH)$ and this forms the basis for the following rules. A table of values for $\theta_0(j)(max)$, containing as many entries as there are atom types, is constructed, and

$$\begin{aligned}
\theta_0(ijk) &= \theta_0(j)(max) \\
\theta_0(ijH) &= 0.98 * \theta_0(j)(max) \\
\theta_0(HjH) &= 0.95 * \theta_0(j)(max)
\end{aligned}$$

Preliminary tests again showed that acceptable accuracy in calculated structures and energies could be obtained from the computed $k_\theta(ijk)$ and $\theta_0(ijk)$s.

For the non-bonded interactions a simple variation of a pre-existing technique is used. For each atom type i recognized by the MM program, values are assigned to A(ii) and B(ii), the values of A(ij) and B(ij) for interactions involving atoms of different types are derived from the geometric mean of the appropriate A(ii) and B(ii):-

$$\begin{aligned}
A(ij) &= (A(ii)A(ji))^{1/2}; & B(ij) &= (B(ii)B(ji))^{1/2} \\
A(iH) &= c8(A(ii)A(HH))^{1/2}; & B(iH) &= c8(B(ii)B(HH))
\end{aligned}$$

Initially c8 was set to 1 so that simple geometric means were used throughout; but whilst it then proved possible for the MM program to calculate either structures or

energies to the required accuracy it proved impossible to get both correct. Optimization of c8 allowed this difficulty to be overcome.

It is fortunate that changes in the barrier to free rotation around a type j — type k central bond consequent upon changes in the type i and type l atoms joined to the central bond can usually be accounted for by the non-bonded, coulombic and angle bending potentials. Thus far it has proved possible to assign values for $V_n(ijkl)$ depending only on the nature of j and k, although it is recognized that more sophisticated algorithms may be necessary as force-field development proceeds.

There are so few out-of-plane bending force constants that these are individually stored. As regards the partial atomic charges these are usually calculated by empirical [14] or MO [15] methods, but in the case of polypeptides charge templates for each amino acid are stored by the program and incorporated into the polypeptide as required.

All of these ideas have been incorporated into the transputer MM program to be described and the program for optimizing force-constants, also to be described, operates on the adjustable parameters described above.

OPTIMIZATION

Two optimization techniques are used for energy minimization; either pattern search [16] which will not be described further because it is the less frequently used; or the block diagonal Newton-Raphson (NR) method.

The basic NR iteration is:-

$$\mathbf{x}\,(k + 1) = \mathbf{x}\,(k) - \alpha F^{+} \nabla V_s(x)$$

where $\mathbf{x}$ is the 3N (N = number of atoms) long vector of cartesian coordinates, α is the step length, F^{+} is the generalized inverse [18] of the Hessian:-

$$F = (\partial^2 V_2/\partial x_i \partial x_j)\ ;\ i = 1, 3N,\ j = 1, 3N$$

and:-

$$\nabla V_s(x) = (\partial V_s/\partial x_j)\ ;\ j = 1, 3N$$

For large molecules computation of the Hessian is an extremely lengthy procedure and the full 3N × 3N matrix is never used above 1 — 200 atoms, although some sparse approximations to the full matrix, containing more elements than the block diagonal approximation, are currently popular [19]. Another problem with the full matrix procedure, when used with a step length of one, is that its radius of convergence is very small and pre-minimization with a more robust technique (which will undoubtedly have a poorer rate of convergence) is always required.

The block diagonal approximation is computationally fast, has a good radius of convergence, and an adequate rate of convergence. The approximation to F is given by:-

$$F = (\partial^2 V_s/\partial x_i \partial x_j); \qquad i = 3m + 1, 3m + 3,$$
$$j = 3m + 1, 3m + 3, \qquad m = 0, N - 1$$

Each 3 × 3 block along the leading diagonal contains first and second partial derivatives of the steric energy with respect to the coordinates of only one atom. Accordingly

the partial derivatives may be calculated on an atom by atom basis, each symmetric block inverted separately and directly (i.e. no need to use the generalized inverse), and each atom moved to its corrected position before the calculations for the next atom is started. This enables each atom to be moved on the basis of the best structure known at that stage of the calculation.

The first derivatives of the steric energy change much more quickly than the second; and it is possible to save on computer time by using the same set of second derivatives for 3–4 iterations of block diagonal NR, recalculating only the first derivatives every iteration.

PARALLEL SOFTWARE FOR MOLECULAR MODELLING

Various classes of parallelism were mentioned when discussing the helios operating system, these are now defined a little more precisely:-

1) TASK PARALLELISM, where each processor is executing a different program in complete isolation from the others.
2) PROGRAM PARALLELISM, where each processor is executing the same program, with different data, in complete isolation from the others.
3) GEOMETRIC PARALLELISM, where each processor executes the same program on a different subregion of a data structure and communicates with neighbouring processors to exchange information about data on the subregion boundaries
4) ALGORITHMIC PARALLELISM, where each processor executes code corresponding to independent subalgorithms which together comprise the complete program.

Task, program, geometric, and algorithmic parallelism all have applications in molecular modelling; but in the absence of large, regularly disposed, multidimensional datasets pure geometric parallelism is uncommon (pure geometric parallelism is useful for fluid flow and lattice simulations).

Task parallelism could be used to run a MM calculation, a MO calculation, a crystallographic least-squares program, a molecular dynamics program, etc. simultaneously; each on a different transputer. This is usually handled by having a master transputer "farm-out" programs to a series of slave transputers and then collect their results. Such a collection of slaves is often called a processor farm. Task parallelism is obviously very simple to implement, and is also very efficient because it is a straightforward matter to ensure that every processor is always 100% busy. However task parallelism is of no use for speeding up the execution of individual programs.

Program parallelism is the most straightforward way of speeding up individual programs. Suppose, for example, that it is necessary to find all of the low-energy conformations of a potential pharmaceutical which has ten variable torsion angles - a frequent requirement in the rational drug design process. In the normal course of events this problem is solved by a grid search procedure. Each torsion angle is stepped around in 10–30 degree increments until conformations corresponding to all combinations of all values of each torsion angle have been generated. At each point various tests are applied to see if the generated conformation is reasonable, and if so its energy is calculated and stored [20]. This is obviously a very time consuming

process. Notice however that each conformation generation and evaluation is completely independent of all of the others.

Suppose therefore that we were going to use 30 degree torsion angle steps and 12 slave transputers on the 10 torsion angle molecule. Twelve starting conformations of the molecule could be generated where the value of the first torsion angle was 0, 30, 60, ..., 330 degrees and values of the other nine were arbitrary. Twelve copies of the conformation generation and evaluation program could then be placed, one on each slave transputer, and set to run with the first torsion angle fixed at its assigned value and only the other nine allowed to vary. Once again all processors could easily be kept 100% busy and the program would run in one twelfth of the time taken on a single processor.

Because no explicit interslave communication is involved is in task or program parallelism the interprocessor network topology is of little consequence. Usually the master transputer is linked to a linear array, or pipeline, of slaves each running a simple operating system kernel. The master transputer sends programs and associated data down the pipeline; and the slave OS kernels either pass these on to the next transputer down the pipeline if its particular transputer is already running a program, or load and run the received program plus data if its transputer free. Although task and program parallelism are very useful they cannot be applied to make individual MM, molecular dynamics (MD) or MO calculations run faster; for which a combination of algorithmic and geometric parallelism is usually required.

PARALLEL MOLECULAR MECHANICS

In order to make effective use of a parallel computer it must be possible to split the program up into a number of noninterdependent blocks capable of executing concurrently. Fortunately it is possible to do this with a MM program in a number of different ways. The process of "parallelizing" a program is in many ways analogous to vectorizing a program for execution on machines such as the Cray or Convex series. However parallelization is in some respects simpler because there is no need to worry about conditional statements inhibiting parallelization — IF statements execute in parallel just as well as any other. This latter statement illustrates the most important advantage of parallel computers; it is not only the floating-point arithmetic that is speeded up (as in a vector processor) but any arbitrary mix of instructions capable of being executed concurrently.

When we initiated our various transputer projects in September 1986 the only language available for use with the transputer was occam. A beta test 3L f-77 compiler was also available but this did not generate code for the T800, and the run times of code generated for the T414 were somewhat long. As a first step towards a parallel molecular mechanics program a 3000 line energy minimization program was converted from fortran-77 into purely sequential occam. The whole process took seven weeks from start to a working, but not fully optimized program. More recently we have been using a beta two (i.e. the "very nearly there" version) 3L f-77 which is now very good, and generates T800 code which runs marginally faster than our best translation into occam.

The program run times for the MM program, in seconds per iteration, are compared with the run times obtained from conventional minicomputers, using the same trial molecule in each case, below:-

T800-20	3L f-77	1.200
T800-20	TDS, occam	1·215
VAX-11/785/FPA VMS, f-77		1.759
VAX-11/780/FPA VMS, f-77		2.124
VAX-11/750/FPA VMS, f-77		2.906
T414-20	TDS, occam	8.482
T414-15	TDS, occam	10.245

Given the very low cost of a host IBM pc plus an Inmos B004, Microway monoputer, or Gemini GM8101 when compared with a laboratory minicomputer, transputer systems are remarkably cost-effective even for ordinary sequential computing.

Once the MM program was running in sequential occam the next problem was to produce a parallel version. A pseudocode for our BDNR MM program is given below:-

```
begin
read in atomic coordinates
calculate initial geometry
set up bond lengths table
set up bond angles table
set up torsion angles table
set up non-bonded interactions table
set up coulombic interactions table
assign force constants
calculate initial energy
print out initial geometry and energy
do until convergence
  do for each atom
    calculate V'_s(x), V'_s(y), V'_s(z)
    calculate V''_s(xx), V''_s(xy), V''_s(xz)
              V''_s(yy), V''_s(yz), V''_s(zz)
    calculate correction vector = -grad(V_s)/Fij
    add correction to previous atom position vector
  end do
end do
store final coordinates on disc
print out final geometry and energy
end
```

Although there is some parallelism which could be exploited in the setting up of the preliminary tables it is hardly worth the effort because the calculations are executed extremely quickly even on a single transputer.

At the beginning of this project we only had five transputers at our disposal (B004 and B003) and limited experience in parallel processing with transputers, so we looked for the most straightforward method of parallelizing our MM program. For any one atom the calculation of the three first and six second partial derivatives of the steric energy with respect to the atomic coordinates can take place in parallel as each calculation is completely independent of the others.

quired per first partial derivative calculated by central differences; no function evaluations extra to those for the first derivatives are required to calculate the $V''_s(ii)$ second

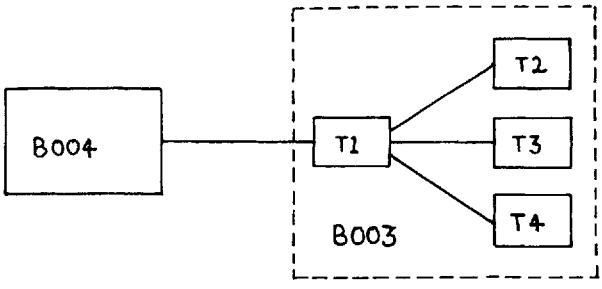

Figure 6 A schematic diagram of the linking of the B004 transputer to one of the four B003 transputers which in turn is linked to the other three.

derivatives by forward followed by reverse differences; and one extra function evaluation is required per $V_s''(ij)$ (i.ne.j) second derivative calculated by forward differences twice. In all then ten function evaluations are required: $V_s(x, y, z)$, $V_s(x + \delta x)$, $V_s(y + \delta y)$, $V_s(z + \delta z)$, $V_s(x - \delta x)$, $V_s(y - \delta y)$, $V_s(z - \delta z)$, $V_s(x + \delta x, y + \delta y)$, $V_s(x + \delta x, z + \delta z)$, $V_s(y + \delta y, z + \delta z)$.

One transputer has to set up the various tables and run the main control section of the energy minimization program. The natural choice for this is the transputer on the B004 because it has the most memory and because it is also directly linked to the main system i/o channel. This leaves the four transputers on the B003 to run copies of the steric energy evaluation procedure. The B004 transputer was linked to one of the B003 transputers which was in turn linked to the other three, as shown in Figure 6. Three of the B003 transputers are given three function evaluations each to perform, and the B003 transputer at the head of the ternary tree performs one function evaluation as well as handling communication of results to the master B004.

This configuration is reasonably efficient, although capable of improvement because the load balancing is not ideal. The transputer at the head of the B003 ternary tree has less to do than the others and therefore sits idle for a proportion of its time. However, lack of scalability is the major objection to this scheme of computation. With eleven transputers one could be the master on the B004, and the others could (say) be joined to it in a pipeline. Each pipeline transputer would run two parallel processes; a function evaluation process and a communications process to pass results and data up and down the pipeline. But there is no easy way to utilize the extra transputers when there are more transputers than function evaluations required. Ideally one needs an algorithm which does not need to be modified as more transputers become available.

A better approach therefore is to give each transputer the task of calculating the various partial derivatives and corrected positions for for a "slice" of the total number of atoms — paying particular attention to load balancing requirements. For example if there are five processors available and 17 atoms in the molecule to be energy minimized; then distributing the atoms into slices of 4, 4, 4, 4, and 1 atoms would be a bad strategy because the processor with only one atom to handle would lie idle for threequarters of its time. The optimum slice sizes in this case would be 4, 3, 3, 3, and 4 atoms.

Having decided on the basic approach there are then two ways of implementing the BDNR minimizer. In the first, all of the first and all of the second partial derivatives of the steric energy with respect to the coordinates, and all of the atom position

correction vectors, are calculated by the various slice processors; then all of the atoms moved to their corrected positions simultaneously. In the second implementation the slice processor actually moves each atom to its new position after its correction vector has been calculated.

It has already been pointed out in the optimization section of this paper that BDNR programs converge more rapidly when one atom is moved at a time, rather than all together, so that a new atomic position is always calculated on the basis of the most up-to-date forces acting on each atom. Ideally, this means that at any time each atom has to know the latest position of all of the others — a situation which is clearly not amenable to parallel processing!

The compromise adopted here has convergence properties very little different to the ideal algorithm. The atoms are allocated to the transputers in slices, and within each slice the atom positions are updated by BDNR optimization one at a time. When all of the slices have been processed, that is when one iteration is complete, there is a global update step which lets each transputer have the new atomic positions calculated by all of the others. This is necessary because, for example, although transputer n may be responsible for updating the position of atom i it needs to know (say) the most recent positions of atoms p & q to evaluate contributions to the atom i derivative from bending of the valency angle i-p-q where the p & q atomic positions may be updated by processors j & k.

Although each processor needs to store a list of all the atomic positions it only needs to store that subset of the bond lengths, bond angles, torsion angles, out-of-plane bonding, and non-bonded interactions tables relevant to the atoms in its slice. The memory space saved thereby can be used to allow the slice processor to handle more atoms if necessary.

Shutting down a parallel program is a little different to the implicit termination of a sequential program when the last instruction has been executed. Each slave transputer in the parallel minimizer executes identical code and will continue to process slices of atoms indefinitely unless explicitly told to stop. In the absence of proper termination the parallel minimizer will run until all slices have been processed for every iteration; and then the slaves will all be waiting for more slices to process and the master will be sending nothing because the calculation is finished — deadlock! The master must pass a message to each of the slaves telling them to terminate all running processes, and then the master can store the results of the calculation away to a disc drive before shutting itself down and returning control to the operating system. Unfortunately deadlock can, and sometimes does, appear unintentionally in the middle of a running program where is can be very difficult to track down.

We encountered the deadlock problem with the first version of our parallel minimizer using the scheme outline above. The program would run quite happily with anything up to nine transputers, but would deadlock some way into the program when using ten or more. The problem was tracked down by writing a simulator for our program where the computational procedures were replaced with "do-nothing" dummies but all of the communications procedures were left intact. When running the simulator program sets up as many windows on the console screen as there are transputers, and each transputer writes a progress report into its own window which logs the execution of its tasks. When the simulator deadlocks there is a snapshot of each transputer's activity on the screen at the moment of deadlock. This enabled the problem to be easily located.

The performance of the parallel minimizer is quite good, as indicated by the times

given below for seconds per iteration of the sequential and parallel minimizers with identical trial molecules:-

1 × T414-15 TDS, occam	10.245
9 × T414-15 TDS, occam	0.855

The hardware configuration was realized with our own B004 and B003 plus an extra B003 kindly loaned to us by a colleague in the computing Science Department. At first sight the reduction in run time seems remarkable because the 9 × T414 time is less than the 1 × T414 time divided by 9, indicating an efficiency of more than 100%! (percentage efficiency is given by 100t(1)/Nt(n) where N is the number of processors, t(1) is the time taken to run the program on one processor, and t(n) the time taken on N processors). The t(1) time in this example is artificially long because it was obtained by using the transputer on a B004 card which is always timeslicing TDS tasks in addition to running user programs, the t(9) time was obtained by using transputers on B003 cards which are not encumbered with operating system tasks. Inmos kindly allowed us some time on their in-house facilities to evaluate our parallel minimizer with more than nine transputers and we were able to show that the efficiency of the MM program is between 80 and 90% for up to 25 or so transputers.

The MM program described above will handle molecules containing up to 12,000 or so atoms with 4 Mbytes of memory per transputer. With a molecule, or molecules, containing this number of atoms it would obviously be advantageous to use as many transputers as possible but the efficiency of the algorithm described above starts to fall off after 25 or so transputers. This is because of the communication time taken up at the end of every iteration when the corrected coordinates calculated by every slave are passed back to the master transputer, assembled into a complete list of new coordinates, and passed back to each slave transputer minus those in its own slice. The routing is not direct in most cases, and a corrected slice of atoms must pass through several other slave transputers, all of which timeslice a communications process in addition to their computation, before reaching the master. The time taken for this communication becomes more and more significant as molecular size and/or the number of molecules in the MM calculation rises.

The only way to maintain high efficiency after the first turnover point is reached is to divide up the calculations performed by the slave transputers into a series of subtask which will execute on transputers connected to the slaves by short communications pathways (This illustrates another good parallel processing dictum "keep the communications as local as possible"). For example one subslave could calculate the contribution to the various first and second derivatives of the atoms in the slice from bond stretching, angle bending, bond torsion, and out-of-plane bending; whilst one or more subslaves did the same thing for coulombic interactions; and another group of subslaves dealt with non-bonded interactions. This kind of subdivision would result in reasonable load balancing.

We are still experimenting with algorithmic decomposition and network topology (which doesn't appear to be quite so important) and are currently at the point where up to 128 transputers could be reasonbly efficiently used for MM calculations. The fact that we will probably never be able to raise the funds for a 128 processor machine is offset by the realization that we currently have funding for 32, and in our hands 32 T800-20s will give a program deliverable compute performance in the same class as the Cray-1S!

FORCE CONSTANT OPTIMIZATION

In this last section of the paper we present a preliminary description of our force constant optimization (FCO) work. A sequential f-77 version of the FCO program has been running on a VAX-11/780-equivalent MC 68020/68881 system for some four months or so. In order to save on programming effort the FCO is performed by a modified pure diagonal NR iteration which takes some 60 iterations or so to converge from a "guesstimated" starting FF. The program is run overnight and at weekends; managing 3–4 iterations per night or 16–20 iterations per weekend! Unfortunately several trial runs are required per functional group to get the weighting scheme and details of the potential functions correct. The whole process is quite time consuming, but not as time consuming as manual trial-and-error methods. At the end of this section our first set of results, for the saturated and unsaturated hydrocarbon kernel of the FF, are presented — they took some 3–4 months to obtain, including writing the program. The equivalent WBFF referred to earlier [13], was also developed in our laboratory but by manual trial-and-error methods over a period of three years!

A pseudocode for the sequential FCO program is given below:-

```
begin
read in data on molecules
calculate geometries & energies
read in experimental geometries & energies
define FF parameters to be optimised
assign weights to parameters (1/σ(p) if possible)
do until convergence
  calculate M = Σ w(obs-calc) for geometries & energies
  do for each parameter (p)
    p = p + δp
    energy minimize all molecules dependent on p
    use new structures & energies to calculate M(+)
    p = p − δp
    energy minimize all molecules dependent on p
    use new structures & energies to calculate M(−)
    reset p to old value
    calculate ∂M/∂p = {M(+) − M(−)}/2δp
    calculate ∂²M/∂p² = {M(+) + M(−) − 2M}/δpδp
    calculate parameter correction = − {∂M/∂p}/∂²M/∂p²
    p(new) = p(old) + correction
    energy minimize all molecules dependent on p
    calculate M(new)
    if M(new) > M(+) or M(−) then
      set p(new) to either p + δp or p − δp
      set M(new) to either M(+) ro M(−)
    endif
  end do
end do
store new parameters on disc
print out tables of p(obs-calc)
end
```

Table 2 Results

Molecule	*Quantity*	*Calc*	*Obsvd*	*Diff*	*Xptal Err*
Cyclohexane	Dist C–C	1.531	1.528	0.003	0.004
	Dist C–H	1.111	1.100	0.011	0.004
	Angl C–C–C	111.1	111.3	0.2	0.5
	Angl H–C–H	105.8	110.0	4.2	2.5
	Tors C–C–C–C	55.7	55.2	0.5	0.5
cis-But-2-ene	Dist C=C	1.345	1.346	0.001	0.003
	Dist C–C=	1.505	1.506	0.001	0.002
	Angl C–C=C	126.2	125.4	0.8	0.4
trans-But-2-ene	Dist C=C	1.345	1.347	0.002	0.003
	Dist C–C=	1.505	1.508	0.003	0.002
trans-But-2-ene	Dist C=C	1.345	1.347	0.002	0.003
	Angl C=C–C	124.7	123.8	0.9	0.4
Cyclohexene	Dist C=C	1.338	1.335	0.003	0.003
	Dist C–C=	1.499	1.504	0.005	0.006
	Dist C–C	1.531	1.515	0.016	0.020
	Dist =C–H	1.099	1.093	0.006	0.015
	Angl C=C–C	123.3	123.5	0.2	0.5
	Angl =C–C–C	112.2	112.1	0.1	0.5
	Angl C–C–C	110.4	110.0	0.4	0.5
	Tors C=C–C–C	15.0	15.2	0.2	2.0
	Tors =C–C–C–C	−45.4	−44.9	0.5	2.0
	Tors C–C–C–C	61.7	60.2	1.5	2.0
Ethylene	Dist C=C	1.336	1.335	0.001	0.003
	Dist C–H	1.098	1.090	0.008	0.003
	Angl C=C–H	121.8	121.7	0.1	0.4
Isobutene	Angl C=C–C	122.1	122.1	0.0	0.3
2, 3-Dimethyl But-2-ene	Dist C=C	1.357	1.353	0.004	0.004
	Dist C–C=	1.513	1.511	0.002	0.002
	Angl C=C–C	122.6	123.9	1.3	0.5
Cyclodeca-1,6 -diene	Dist C=C	1.345	1.326	0.019	0.004
	Dist C–C=	1.508	1.506	0.002	0.006
	Dist C–C	1.536	1.534	0.002	0.006
	Dist C–H	1.113	1.112	0.001	0.004
	Angl C=C–C	128.0	128.2	0.2	0.3
	Angl C–C–C=	112.4	112.8	0.4	0.3
	Angl C–C–C=	112.4	112.8	0.4	0.3
	Angl C–C–C	113.6	114.1	0.5	0.5
	Angl C=C–H	118.1	116.6	1.5	1.0
	Angl H–C–H	104.6	105.6	1.0	1.0
Cyclohex-1,4 -diene	Dist C=C	1.336	1.334	0.002	0.002
	Dist C–C=	1.497	1.496	0.001	0.001
	Dist =C–C	1.099	.103	0.004	0.003
	Dist C–H	1.110	1.114	0.004	0.003
	Angl C=C–C	123.4	123.4	0.0	
				0.2	
	Angl =C–C–C=	113.2	113.2	0.1	0.3
	Angl C=C–H	120.3	123.4	3.1	2.7
	Angl C–C(=)–H	116.3	113.2	3.1	2.7
	Angl =C–C–H	109.4	110.0	0.6	0.4
	Angl H–C–H	105.9	103.0	2.9	2.0
Propene	Dist C=C	1.340	1.341	0.001	0.001
	Dist C–C=	1.505	1.504	0.001	0.001
	Angl C=C–C	124.7	124.8	0.1	0.1

Table 2 (cont.)

Molecule	*Quantity*	*Calc*	*Obsvd*	*Diff*	*Xptal Err*
Manxane	Dist H...H	2.294	2.200	0.094	0.400
cis-But-1-ene	Angl C=C-C	125.7	126.7	1.0	0.5
	Angl =C-C-C	116.1	114.8	1.3	0.5
skew-But-1-ene	Angl C=C-C	124.8	125.4	0.6	0.5
	Angl =C-C-C	113.0	112.1	0.9	0.5
2-methyl-skew-	Tors C-C(=)-C-C	64.9	72.7	7.8	
But-1-ene				5.0	
Cyclopentene	Tors Flap angle	157.3	156.7	0.6	2.0
Norbornadiene	Dist C=C	1.335	1.345	0.008	0.003
	Dist C-C=	1.504	1.535	0.031	0.007
	Dist <C-C>	1.545	1.544	0.001	0.004
	Angl C-C-C	91.9	94.1	2.2	2.0
	Tors Roof angle	112.6	115.6	3.0	2.0
Bicyclo[2.2.2.]					
-Oct-2-ene	Tors Roof angle	120.8	121.2	0.4	2.0
	Tors C-C=C-C	0.0	0.0	0.0	2.0
Bicyclo[2.2.]					
-Oct-2-ene	Tors Roof angle	120.8	121.2	0.4	2.0
	Tors C-C=C-C	0.0	0.0	0.0	2.0
Bicyclo[2.2.]					
-octa-2,5-diene	Tors Roof angle	121.8	123.4	2.6	2.0
	Tors C C-C-C	0.0	0.0	0.0	2.0
Pin-2-ene	Dist C=C	1.343	1.340	0.003	0.010
	Angl C=C-C(exo)	124.8	126.0	1.2	3.0
	Angl C=C-C(endo)	116.0	118.0	2.0	3.0
	Angl C=C-C	119.9	118.0	1.9	3.0
	Angl =C-C-C	109.1	112.0	2.9	3.0
	Tors 4R Fold	140.9	146.0	5.1	8.0
Adamantane	Dist <C-C>	1.530	1.534	0.004	0.004
	Angl C(B)-C-C(B)	109.7	110.0	0.3	0.5
	Angl C-C(B)-C	109.1	109.2	0.1	0.5
t,t,t-Cyclodeca					
-1,5,9-triene	Dist C=C	1.342	1.320	0.022	0.010
	Dist C-C=	1.506	1.490	0.016	0.010
	Dist C-C	1.531	1.540	0.009	0.010
	Angl C=C-C	124.5	124.1	0.4	0.5
	Angl C-C-C=	111.8	111.1	0.7	0.5
	Tors C-C=C-C	−178.1	178.0	3.9	2.0
	Tors C-C-C=C	−113.8	−116.5	2.7	2.0
	Tors C-C-C-C=	57.0	63.4	6.4	2.0
gauche/anti					
Butane	Energy diff	0.934	0.980	0.046	0.047
cis/trans					
But-2-ene	Energy diff	1.381	1.130	0.251	0.380

Simply recompiling the MC 68020/68881 f-77 for a single T800-20 made the program run some 3–4 times faster and reduces the FF development time to approximately one month per functional group.

The FCO program is a natural for parallelization because most of the time is taken up by the "energy minimize all molecules dependent on p" lines of the pseudocode. The two sets of minimizations required to evaluate M(+) and M(−) can be executed in parallel, as can all the minimizations within each set. The set of minimizations required to obtain M(new) have to be executed after the first two sets, but again each

minimization within the set can proceed in parallel with the others. The parallelization approach being considered here is a master control transputer farming out minimizations to a collection of slave transputers each running a sequential version of the MM program.

RESULTS

Table 2 shows a comparison of experimental geometrical and energetic quantities for a reference set of saturated and unsaturated hydrocarbons with the values calculated by our MM program using a FF calculated by the FCO program. The accuracy of the calculated structures and energies compare well with the precision of the WBFF [13].

Acknowledgements

The authors are grateful to SERC for the provision of our initial transputer system and for further funds to extend this to 32 T800s; to Ciba-Geigy Industrial Chemicals for providing funds which allowed us to develop the 8 T800, 32 Mbyte memory Compute Card; to Gemini Computer Systems Ltd for the gifts of a T800, 8 Mbyte GM8101; and to U-Microcomputers Ltd for using their PCB CAD facilities to convert our Compute Card prototype into a multilayer PCB.

GLOSSARY

AMT	— Active Memory Technology (company)
B003	— PCB with 4x (T414 + 256 kbytes memory)
B004	— PCB with T414 plus 2 Mbytes memory
CFF	— Consistent Force Field
CPU	— Central Processing Unit
C004	— Crossbar switch for Inmos Links
DAP	— Distributed Array Processor
FPU	— Floating Point Unit
FPS	— Floating Point Systems (company)
I/O	— Input/Output
iPSC	— Intel Personal Super Computer
i80386	— Intel 32-bit microprocessor
MD	— Molecular Dynamics
MIMD	— Multiple Instruction Multiple Data
MM	— Molecular Mechanics
M212	— 16-bit disc controller transputer
PC	— Personal Computer
PCB	— Printed Circuit Board
RISC	— Reduced Instruction Set Computer
RS232	— Standard for serial data transfer
SIMD	— Single Instruction Multiple Data

TDS	— Transputer Development System
TRANSPARALLEL COMPUTER	— Transputer-based parallel computer
T212	— 16-bit integer transputer
T414	— 32-bit integer transputer
T800	— 32-bit floating point transputer

References

[1] I.S. Reid and J.K. Reid (Eds.), "Vector and Parallel Processors in Computational Science", North-Holland, Amsterdam, (1985). ISBN 0-444-86794-3.

[2] K.C. Bowler, R.D. Kenway, G.S. Pawley and D. Roweth, "An Introduction to Occam 2 Programming", Chartwell-Bratt, Lund, (1987). ISBN 0-86238-137-1.

[3] J.L. Gustafson, S. Hawkinson and K. Scott, "The Architecture of a Homogenous Vector Supercomputer", *J. Parallel & Distributed Computing*, **3**, 297 (1986).

[4] G.C. Fox and P.C. Messina, "Advanced Computer Architectures", *Scientific American*, **257(4)**, 45 (1987).

[5] M. Homewood, D. May, D. Shepherd and R. Shepherd, "The IMS T800 Transputer", *IEEE Micro*, **7(5)**, 1 (1987).

[6] D. Cormie, "Getting Started with the TDS", Inmos Technical Note 3, Document No. 72-TCH-003-00, Inmos, Bristol

[7] N. Garnett, J. King and B. Veer, "Helios Technical Manual", Perihelion Software, Shepton Mallet, Somerset, (1987).

[8] D. Pountain and D. May, "A Tutorial Introduction to Occam Programming", BSP Professional Books, Oxford, (1987), ISBN 0-632-01847-X.

[9] D.N.J. White, "Computer Methods for Molecular Design", 67 (1986) in D.M. Blow, A.R. Fersht, & G. Winter (Eds.), "Design Construction and Properties of Novel Protein Molecules", The Royal Society, London, (1986). ISBN 0-85403-271-1.

[10] S.J. Weiner, P.A. Kollman, D.T. Nguyen and D.A. Case, "An All Atom Force Field for Simulations of Proteins and Nucleic Acids", *J. Computational Chem.*, **7**, 230 (1986).

[11] S.R. Niketic and K. Rasmussen, "The Consistent Force Field — A Documentation" Lecture Notes in Chemistry, Springer Verlag, Berlin, (1977). ISBN 3-540-08344-8.

[12] S.J. Weiner, P.A. Kollman, D.A. Case, U.C. Singh, C. Ghio, G. Alagona, S. Profeta and P. Weiner, "A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins", *J. Am. Chem. Soc.*, **106**, 765 (1984).

[13] D.N.J. White and M.J. Bovill, "Molecular Mechanics Calculations on Alkenes and Non-Conjugated Alkenes", *J. Chem. Soc. Perkin* **II**, 1610 (1977).

[14] R.J. Abraham, L. Griffiths and P. Loftus, "Approaches to Charge Calculations in Molecular Mechanics", *J. Computational Chem.*, **3**, 407 (1982).

[15] J.A. Pople and M. Gordon, "Molecular Orbital Theory of the Electronic Structure of Organic Compounds. I. Substitutent Effects and Dipole Moments", *J. Am. Chem. Soc.*, **89**, 4253 (1967).

[16] G.R. Walsh, "Methods of Optimization", 76 (Hooke & Jeeves Method), J. Wiley & Sons, London, (1975). ISBN 0-471-91922-5.

[17] *ibid*, 108 (Newton-Raphson Method).

[18] D.N.J. White, "Further Notes on the Generalized Inverse", *Acta Cryst.*, **A33**, 1010 (1977).

[19] T. Schlick and M. Overton, "A Powerful Truncated Newton Method for Potential Energy Minimization", *J. Computational Chem.*, **8**, 1025 (1987).

[20] D.N.J. White and C. Morrow, "Cyclic Tetrapeptides I: The Calculated Potential Energy Minima of Cyclic Tetrapeptides Composed of Small Amino-Acid Residues", Computers & Chemistry, **3**, 33 (1979).

Note added in proof: We now (5/10/88) have a parallel version of our FCO program running in occam, and two complete molecular modelling packages (GEMM & CHEMMOD) running in parallel fortran.