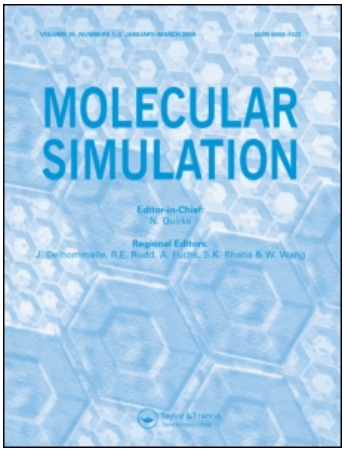

Molecular Simulation
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713644482

Grand Canonical Ensemble Monte Carlo Simulation on a Transputer Array

Stephen J. Zara[a]; David Nicholson[a]
[a] Department of Chemistry, Imperial College of Science, Technology and Medicine, University of London, London, UK

To cite this Article Zara, Stephen J. and Nicholson, David(1990) 'Grand Canonical Ensemble Monte Carlo Simulation on a Transputer Array', Molecular Simulation, 5: 3, 245 — 261
To link to this Article: DOI: 10.1080/08927029008022134
URL: http://dx.doi.org/10.1080/08927029008022134

Molecular Simulation, 1990, Vol. 5, pp. 245–261
Reprints available directly from the publisher
Photocopying permitted by license only

GRAND CANONICAL ENSEMBLE MONTE CARLO SIMULATION ON A TRANSPUTER ARRAY

STEPHEN J. ZARA and DAVID NICHOLSON†

Department of Chemistry, Imperial College of Science, Technology and Medicine, University of London, Exhibition Road, London SW7 2AY, UK

(*Received October 1988; accepted March 1990*)

Five methods are described for the distribution over a 3D Transputer array of the calculation of the pair interaction component of particle energy. The most efficient method, expressed in terms of the time to complete a simulation, depends on the size of the simulation and the Transputer array. This dependence is quantified, with emphasis on Grand Canonical Ensemble Monte Carlo simulation, and yields criteria for the optimum strategy for parallel implementation of GCEMC algorithms. The equations derived are generally applicable, and have implications for the programming of Molecular Dynamics simulations.

KEY WORDS: Monte Carlo simulation, transputers, parallel processing

1. INTRODUCTION

The Inmos T800 Transputer is a powerful microprocessor yielding 10 MIPS and 1.5 MFLOPS, with a 32 bit data bus, and 32-bit addressing; a performance equal to the VAX 8600 'supermini' computer. In addition to the high speed instruction processing and mathematical operations, each T800 processor contains four high-speed (2×10^7 baud) two-way serial links, with the appropriate communications circuits running in parallel with the instruction processors and floating point calculator. These serial links are present to simplify communication between Transputers and permit the construction of processor networks of arbitrary size. The Transputer is therefore a building block for inexpensive parallel processors.

The development of the Transputer differed from that of most current computers in that it was designed in the context of a concurrency principle proposed by Hoare [1], and embodied in the high level programming language Occam. An Occam program can execute as parallel tasks (time-sliced) on a single processor, or as concurrent processes on a network. Occam permits the distribution of a program over a network of arbitrary size without the programmer having to consider specific hardware configuration [2]. Multi-processor arrays based on Transputers are potentially sources of computing power at the supercomputer level. To access this power requires the formulation of efficient parallel algorithms.

Scientific application of Transputers has been discussed by Hey [3], and their use in computational chemistry reviewed by Fincham [4]. Molecular dynamics algorithms have been described by Smith and Fincham [5]. Aspects of these papers and articles

† All correspondence to: Dr. D. Nicholson

illustrate how approaches to parallel processing, usually applied to current single-processor supercomputer architectures, are being adapted for use in transputer arrays. An example of such an approach is pipelining, in which stages of a sequential calculation are run concurrently, forming a 'conveyor belt' arrangement. Pipelining is well suited to small arrays of processors, each with low storage capacity and very fast intercommunication. Pipelining is frequently implemented by hardware control of the independently operating sections of CPUs, and is the primary source of the speed in the CRAY machines, it requires a linear arrangement of processors. Other techniques for parallel calculation use different topologies. A common practice is to pass data around loops of processors, as in the 'Pass-the Parcel' and 'Tractor-Tread' techniques [5].

The pipeline methods and linear or loop topologies have, to date, been important in the evaluation of Transputer potential in computational chemistry. However, these techniques are only efficient in the use of resources when (1) the number of processors is small, (2) the calculation time greatly exceeds the communication time, and (3) the memory per processor is small. The potential for inefficiency can be clearly seen in many methods, as the redundancy of some of the Transputer's communication links, and the large volume of data communicated between processors. As we will demonstrate, for simple calculations on large arrays of processors, serious bottlenecks can arise.

In this paper we present algorithms for Grand Canonical Ensemble Monte Carlo Simulation which can be implemented on large arrays of Transputers, and compare these algorithms on the basis of communication overheads and memory requirements.

2. METHODS

2.1 The Transputer Installation

We have obtained estimates of Transputer processing speed, for calculation and communication, from our Transputer installation. This consists of a multi-user host machine, a DELL 300 (16 MHz 80386 processor, VGA graphics, 40 Mbyte hard disk) running UNIX System V (SCO XENIX). Four Transputers (20 MHz T800B, each with 2 Mbytes of memory on Microway Monoputer boards) are connected to form a single cell of the 'diamond lattice' topology (see Figure 1). We have developed driver software to provide multi-user access to the Transputers in the XENIX environment, conforming to the Inmos Alien File Server protocol, so that commercial Transputer software, such as Occam2 and 3L Fortran 77 compilers, can be run on the processor array.

2.2 Occam Code for Grand Canonical Ensemble Monte Carlo Simulation

Although the simulation procedure for GCEMC simulation has been extensively described [6], we believe that it is helpful to present the procedure in Occam source code, since this is appropriate to the discussion of Transputer programming. The code is given in Appendix 2 in the form of an Occam PROCedure, trial.step.

The Occam code illustrates the framework of the Grand Canonical Ensemble

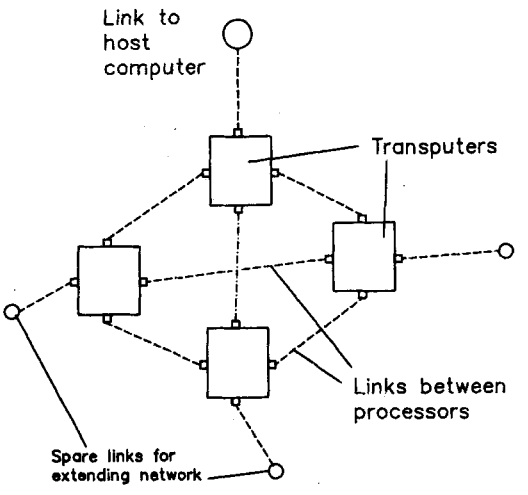

Figure 1 The installation upon which the GCEMC simulations have been implemented. The 'diamond' lattice arrangement of the Transputers is shown.

Monte Carlo Simulation method, and is part of the program that resides on a single processor — the controller. The role of the controller is to coordinate the activity of all other processors (servers) involved in the simulation. This commumication is not an important factor in determining the efficiency of simulation algorithms, as it contributes little to the total simulation time.

2.3 The Storage of Particle Data on a Processor Array

In a simple simulation in which only the singlet densities of the various species is required, the coordinates of each particle have to be stored, along with some indication of the species. This species indicator can be used as an index to tables of species-dependent properties, such as radius and charge. Storage for other data is also required, such as the singlet density histograms, but the size of these data structures is insignificant in comparison with that needed, for example, for the coordinates.

The particle coordinates can be stored to satisfactory precision in cartesian form as three 32-bit floating-point numbers (x, y and z): 12 bytes per particle. If, as is likely, there are fewer than 257 species of particle, then the species indicator can be stored as a single byte per particle. The basic storage requirement for a mixed-species simulation is, therefore, 13 bytes per particle.

A typical hardware environment for Transputers provides each processor with at least a million bytes of memory. This memory allocation may seem small compared to a typical mainframe. However, Transputers currently possess no resident operating systems and utilise a host system to provide hardware control, file manipulation, and the supply of executable code. The Transputers are therefore left as 'bare machines', with no memory overheads for a run-time operating system. In our Occam2 program for Grand Canonical Ensemble simulation of restricted primitive electrolytes, the server program, present on all but the controller processor, consists of ~ 1000 lines of source code which compile to 30 Kbytes of executable code, leaving 1970 Kbytes

free for data. Coordinate storage, for 500 structureless particles, would require only another ~ 6 K bytes. The storage available for energy data is therefore around 1960 Kbytes, or 98% of the total supplied.

It is apparent that it is feasible to provide each processor with a complete copy of coordinate data, thereby reducing the need for communication between processors.

2.4 Parallel Algorithms for the Calculation of Pair Interaction Energy

The energy of the system is given by the sums of (1) the pair interaction energies between particles, given by $\sum_{ij} u_{ij}(2)[j < i]$. (2) The particle wall interactions $\sum_i u_i^{(1)W}$. (3) The long-range corrections to the energy $\sum_i u_i^{(1)L}$.

The calculations of the particle-wall interactions and the long-range corrections to the energy are usually rapid, and will not be discussed further. The process that occupies virtually all the computing power is the calculation of the particle pair interactions.

In the trial step method, presented here as Occam code, the change in system energy between two configurations results from the change in the energy of a single particle i undergoing a move, a creation, or a destruction. The pair interactions that change are $u_{ij}^{(1)}[j = 1..n, j > i]$. The simplest approach to evaluating this change is to perform the calculation of the $u_{ij}s$ for the existing particle state, and for the trial particle state, in each trial step. The problem with this approach is that it is wasteful of time. For move and destroy steps, the pair interactions for the existing state will already have been calculated at some previous step in the simulation. The alternative to re-calculating these interactions is to maintain a table of pair interactions. The table is arranged so that element $[i,j]$ is the pair interaction $u_{ij}^{(2)}$, and the total of the interactions for a particle i is the sum of the elements of row i (or column i: the table is symmetrical). For systems containing large numbers of particles the storage of pair interactions is not feasible, as the size of the table grows with N^2, and the re-calculation method must be adopted.

One matter that has not been mentioned is the time taken for the transmission of information between the controller processor and the server processors. As will be seen later, the existence of a single destination process or for messages from the servers will yield a communication bottleneck. This will occur when the partial sums of the pair energies are transmitted to the controller. However, the volume of data involved in this communication is insignificant, especially when compared to the transmission of pair energies between server processors in methods C-E below, and can safely be neglected.

Now that the two general approaches to the calculation of pair interactions have been described, specific algorithms (A-E below) will be discussed in detail, in the context of concurrent calculation using Transputers.

A No energy storage

The procedure in which the table of $u^{(2)}$ values is not stored to the entire data set. For each energy calculation the controller processor sends the data for the test particle to each server process. Each of the P servers sums the pair interactions between the test particle and a distinct subset (of size N/P) of the particles in the system, and transmits the sum to the controller.

B Storage of particle energy totals

The benefit of the full-storage approach compared to recalculation is that the current $u^{(2)}$ value for the test particle is available at the start of each trial configuration. An intermediate approach is to store these sums in a one-dimensional array with N elements. These elements need to be recalculated only after a successful trial; in most MC simulations these only constitute half of the move steps and (at least in dense systems) a considerably smaller fraction of the creation and destruction trials. This approach yields a saving of at least 50% in time over method A.

C Storage of all pair interactions, each processor has a complete copy of the interaction table

The storage of all particle pair interactions can be implemented for systems of moderate size. However, the division of the pair interaction table between processors in a parallel array is not a simple task. One reason for this is that communication with the server processor is necessary when pair interactions are stored. This can be illustrated with the simplest storage method, which also happens to place the highest demand on communication.

The storage of the interaction table on each processor requires $4N^2$ bytes for N particles and is only feasible therefore for simulation systems of moderate size. In addition to the storage requirement, this method makes very intensive use of communication — see Figure 2. When the pair interactions for a particle i are being calculated, the elements of both column i and row i must be updated. If a division of labour is arranged between processors, such that each of the P processors calculates a separate group of N/P elements, then each processor will have to broadcast these elements to all other servers, and will have to receive the values of the other $N\text{-}N/P$ elements.

D Storage of all pair interactions; each processor holds a group of N/P rows of the interaction table

In this method, the interaction table is fragmented into groups of rows, each group located on a separate processor. This arrangement overcomes some of the communications requirement of method C as illustrated in Figure 3, in addition to reducing the storage per processor from $4\,N^2$ to $4N^2/P$. For the calculation of the pair interactions for a particle i, each processor calculates the elements of column i for the rows in its possession, and broadcasts these elements to the single processor containing row i. However, even though the volume of communication is considerably reduced by the presence of only a single destination processor for the column data, the limited number of communication links on each processor (4 for each Transputer) can result in a bottleneck for large processor arrays (see below). All of the $N\text{-}N/P$ column elements transmitted to the processor holding row i have to pass through these communication links.

E Storage of all pair interactions; each processor holds a matrix of N/P rows by N/P columns

The communication bottleneck described in method D can be overcome by dividing both rows and columns between processors as in Figure 4. This means that rather than a single destination processor holding the row data for the selected particle, each of the P processors will hold N/P elements of this row. The volume of communication

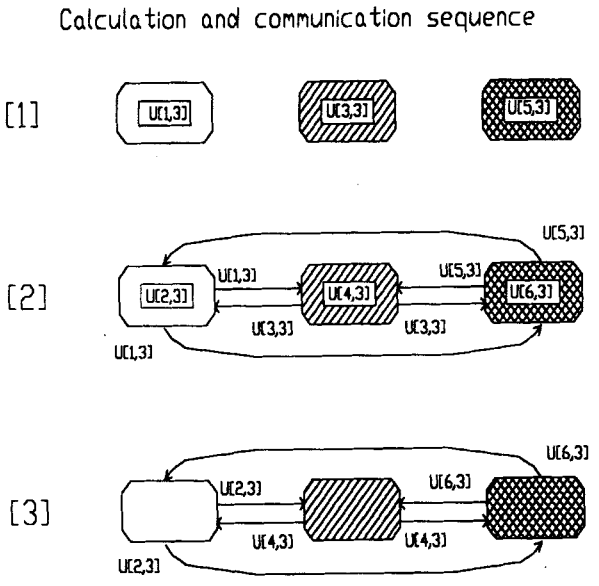

Figure 2 The calculation and communication sequence for method C. In this example, six particles are present in the system and a calculation of the pair interactions for the third particle is being performed. A table (U) of pair interactions is present in its entirety on each of three transputers. Large boxes represent transputers. A square within a transputer represents a calculation in progress; the table element being calculated is shown. An arrow indicates the communication of table elements between processors, or between different elements within a single processor. The element being transmitted is indicated at the tail of the arrow. The processors are indicated by different shadings: open – processor 1; diagonally shaded – processor 2; cross-hatched – processor 3. The time progression is indicated by the bracketed numbers to the left of the figure.

will not be reduced in comparison to with method D, as there are still N-N/P elements to be transmitted, but these elements will not be funnelled onto one processor.

2.5 Storage of particle-neighbour pair interactions

For systems which interact primarily by dipole-dipole interactions or multipole interactions of higher order, or for which dispersion interactions dominate, only interactions between 'neighbouring' particle are important. The definition of 'neighbouring' depends on the interaction, but for a dipole-dominated system, such as water, only the nearest few hundred particles need to be considered. This limitation on the interactions is important in terms of the energy storage requirement, as the pair interaction table becomes a sparse array. If the insignificant interactions are assumed to be zero, then the non-zero elements of the table will be randomly located on one side of the diagonal, with a mirror image on the other side. Thus, in addition to storage for the non-zero pair interactions, lists of the locations of these elements in the table will be required. However, as the close-neighbour limitation on pair interactions (to the nearest C particles) cuts down the storage dependence from N^2 to CN, the storage saving is still very significant.

The allocation of elements between processors can be performed as in methods C, D & E, with only the non-zero elements actually requiring storage space in each case.

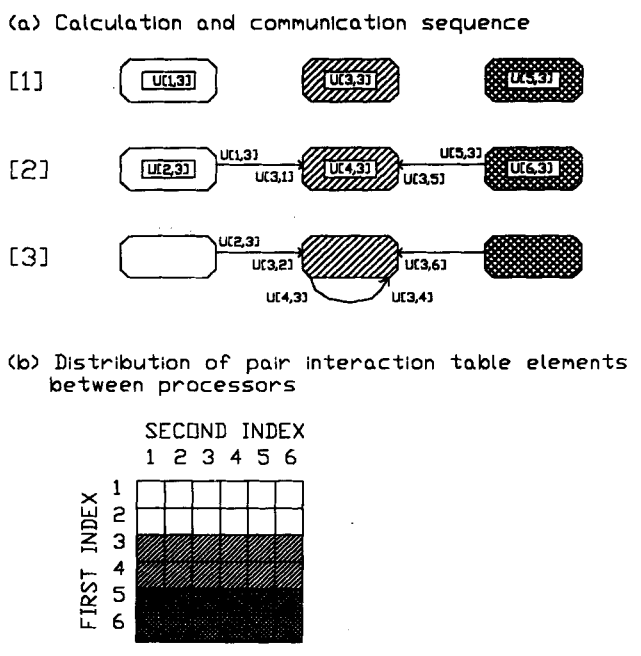

Figure 3 (a) The calculation and communication sequence for method D. See figure 2 for details. Note the reduction in the volume of communication as compared with Figure 2 (although the total communication time is unchanged – see Table 2).

(b) The pair interaction table (U) is shown, with the elements shaded to indicate the transputer in which they are held.

2.6 Quantifying Calculation and Communication Times for a Transputer Array

The calculation times, communication times, and energy storage statistics are shown, for each of the five parallel algorithms, in Table 1. It should be noted that communication and calculation can occur in parallel, as the four links and CPU of a Transputer form five independent units which operate concurrently. The equations for the total time taken, T_{tot}, are approximations, which assume that both communication and calculation start simultaneously. However, this is obviously not the case, as the first communication can only commence when the first pair interaction has been calculated [3], and no calculation will be in progress during the final communication. However, the error resulting from this assumption is negligible when each processor calculates many pair interactions, which is usually the case. Also, the time taken for a pair interaction to be transmitted through the processor array, from source to destination, has been neglected. The time taken for each processor has been taken as the time for one processor to complete a transmission to its neighbour, T_c. In practice this is a reasonable assumption since, for the algorithms we are considering, communications between the transmitting processor and final destination are not synchronised. Once a value has been communicated to the first processor on the route to the destination, the transmitter processor considers the operation complete, and can immediately

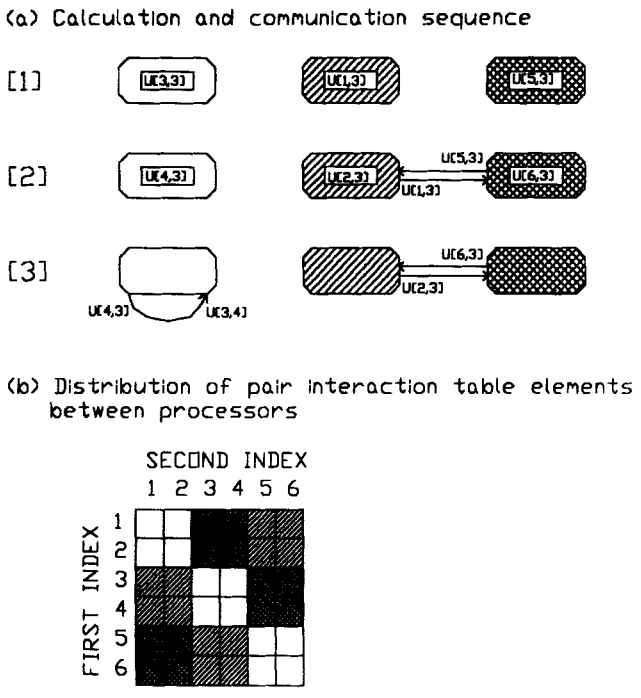

Figure 4 (a) The calculation and communication sequence for method E. See Figure 2 for details. The total volume of communication is unchanged as compared with method D (Figure 3), but the distribution of the destination row (number 3) over all processors reduces the total communication time (see Table 2).

(b) The pair interaction table (U) is shown, with the elements shaded to indicate the transputer in which they are held.

Table 1 Equations giving the storage requirements and total communication time (T_{cT}) are given for the parallel algorithms A-E. P is the number of processors, and L is the number of communication links per processor which can operate concurrently with each other, and with the calculation. The total calculation time is the same for each method (see section 2.6); $T_{uT} = NT_u / P$. The units of time are those in which the communication times and calculation times for a single pair interaction (T_c and T_u respectively) are measured. The equations given below are derived in section 2.6.

Method	*Total Communication Time*	*Storage Required (Bytes)*
A	0	0
B	0	$4N$
C	$N(1-1/P)T_c/L$	$2N^2$
D	$N(1-1/P)T_c/L$	$4N^2/P$
E	$N(1-1/P)T_c/LP$	$4N^2/P$

initiate another transmission: the delay between values for the transmitter is only T_c. Similarly, for the destination processor, the delay between values received is only T_c. Therefore, communication pipelines are established, with transmissions and recep-

tions proceeding without interruption, and a small (compared to the total communication time) delay resulting between the final calculation and the end of the whole procedure while the pipeline empties. As the lengths of the largest pipelines (equal to the maximum dimension of a processor array) are small for the 'diamond lattice' topology, this assumption is not a serious problem.

The calculation workload is divided evenly between processors; each calculates N/P pair interactions. Therefore the total time spent in calculation, $T_{uT} = T_u N/P$, where T_u is the time taken for one processor to calculate one pair interaction. This calculation time applies for each of the five algorithms; the primary difference between these algorithms is therefore the total communication time, T_{cT}. As communication and calculation proceed concurrently (but see above), and both must be completed to obtain the total pair interaction energy of a particle, the total time T is simply $max\ (T_{uT}, T_{cT})$ in each case.

A. For this method, no communication is required between processors and therefore $T_{cT} = 0$. The storage per processor for energies is, by definition, zero.

B. Again, the communication time is zero, but the energy storage for each of the N processors is one word per particle, or $4N$ bytes in total.

C. In this case, the pair interaction table occupies N^2 words on each processor. However, space can be saved as the table is symmetrical and only $N^2/2$ elements need to be retained. As each processor calculates N/P pair interactions, each will have to broadcast these values to all other processors, and receive the remaining $N(1\text{-}1/P)$ elements from the other processors. As these transmissions occur in parallel for all the processors, only the communications for one processor need to be counted to determine the time taken. Since the reception and transmission can occur concurrently on each processor, and there are four links, the communication time is given $N(1-1/P)T_c/4$.

D. In this method, there are N^2/P elements per processor. Note that the table cannot be stored in triangular form (as in C) when distributed between processors. The discussion about communications for C applies for D also, although only one processor holds a copy of the row to receive the calculated pair interactions. Because of the parallel operations of processors described for C, this makes no difference to the communication time; $T_{cT} = N(1-1/P)T_c/4$.

E. In this method, the storage requirement is the same as in D. However, the distribution of the destination row between all the processors means that there are $4P$ communication links through which the calculated pair interactions can pass to this row, rather than just 4. This increased bandwidth reduces the communication time to $N(1-1/P)T_c/4P$.

Another matter of importance is the 'start-up' time, which results from the time taken for a Transputer to switch between tasks and initiate a communication. We have obtained an estimate of this time to be about 10 μs (see Appendix 1). This is a significant delay compared with a single energy or coordinate communication, particularly as 'start-up' cannot be overlapped with calculation. However, the start-up time can be reduced to a small fraction of the average T_c by transmitting several values (coordinates or energies) in one message. In this way, communication and calculation will always overlap except at the beginning and end of each particle energy calculation since, after the first communication is started, calculation and accumulation of values for the next communication proceed concurrently. Therefore, in the algorithms we discuss, start-up time is an insignificant factor and will not be considered further.

2.7 *Selection of the Best Method for a Given Processor Array*

The values of T_{tot}, the total time taken to obtain the sum of the pair interactions for a particular particle, have been calculated for different values of N and P, and are shown in Figure 5. There are two points which arise from this analysis; firstly, T_{tot} has a linear dependence on the number of particles, N, so that the following discussion of processor dependence applies to systems of all sizes. Secondly, non-integral results have been rounded up to the next integer: a non-integral number of calculations or communications is not valid.

Figure 5 shows an important, and perhaps unexpected phenomenon; for calculation methods C and D, there is an optimum number of processors. Adding more Transputers to the processor array is not merely wasteful of computing resources, but actually slows down the calculation! This result is a consequence of the severe communication bottleneck that is produced when all procesors attempt to transmit to a single destination so that all data have to pass through only four Transputer links. This problem can be quantified using the formulae for T presented in Table 2. The size of the processor array at which communication time exceeds calculation time is given by

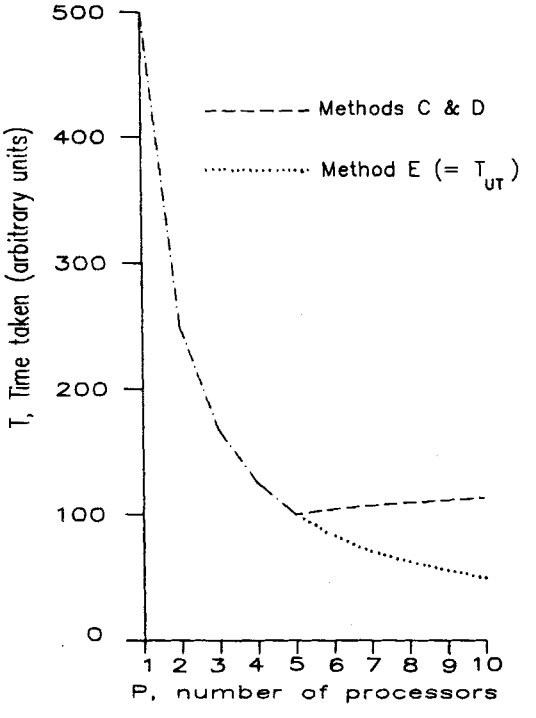

Figure 5 The total time to determine the pair interactions for a particle is shown against the number of Transputers (P) for a system of 500 particles, using the values $T_c = T_u = 1$. The time is shown for methods C, D and E, the latter also equaling the total calculation time alone (T_{uT}). Note that (1) the time for methods C and D increases after $P = 5$, and (2) for these T_c and T_u values the time for method E to complete the pair interaction sum is always equal to T_{uT}.

$$T_c N(1-1/P)/4 > T_u N/P \tag{1}$$

which solves for P to give

$$P > 1 + 4T_u/T_c \tag{2}$$

For roughly equal communication and calculation times, method C becomes wasteful of processor power on arrays of more than 5 Transputers.

The distribution of the destination row between all processors in the network, as in method E, gives a quite different dependence on array size. The size of the processor array at which communication time exceeds calculation time is given by

$$T_c N(1-1/P)/4P > T_u N/P \tag{3}$$

The solution for P is

$$P < 1/(1 - 4T_u/T_c) \tag{4}$$

If $4T_u \geqslant T_c$, no valid solution exists, hence the communication bottleneck does not arise. The communication bottleneck is only present if $T_c > 4T_u$. This is a most unlikely circumstance, since the communication of an energy value and associated communication protocol information, along with an index to the destination processor will be more than 10 bytes giving a transmission time of around 5 μs. For the bottleneck, the calculation time would have to be less than 1.25 μs. Estimates of T800 speed average 1.5 MFLOPS, so the calculation would have to involve less than 2 floating point operations for this to occur.

Clearly, method E is a dramatic improvement over methods C and D, and should be the preferred technique when the storage capacity is available, a matter which will be discussed in the next section.

2.8 Storage Requirements

Of the techniques considered here, method E clearly makes the most effective use of a Transputer array. However, the memory requirement imposes a limit to the size of the system that can be modelled using this technique. As shown in Table 2, the storage for a system containing N particles, using calculation method E is N^2/P multiplied by the storage per pair interaction. For 32-bit precision (4 bytes per number) the minimum processor array size required to model a system of N particles, given a fixed memory allocation per Transputer, can be specified. Therefore, method E become inoperable when N is too large. When this occurs, the preferred algorithm is method B. This is no more than 50% worse (in terms of calculation time) than method E.

3. CONCLUSIONS

Five parallel algorithms have been presented for the calculation of particle-particle pair interactions in Monte Carlo simulation. These algorithms are of varying efficiency in their use of the Transputer array. The question of how efficiency is determined is worth some discussion. Fox and Otto [7] define a term of the form

$$E = T_{uT}/(T_{uT} + T_{cT}),$$

where E is the efficiency, and the other terms are defined above. However, this expression does not take into account the practical economies of processor array installation, and differences between algorithms. The expression measures how well a particular calculation method distributes over an array. What is required is an absolute, not relative, measure of the speed of an algorithm. This measure should yield a function $T(N,P)$, which returns the time taken to complete a calculation, where N is a measure of the size of the calculation (in our work, the number of particles in the simulation), and P is the number of processors used. The function T is algorithm-dependent, and can be used to compare different computation techniques for different size problems on different processor array sizes. We have, in effect, devised such T functions in the form of T_{tot}.

Economic factors are becoming increasingly important; a re-calculation technique such as method B, with minimal storage requirement, uses 50% more time than a storage method such as E. However, as long as the cost of computer memory remains relatively high, method B is preferable to method E even on small processor arrays.

A point we wish to emphasise is that optimising a calculation on a processor array is not a simple task; an algorithm that is efficient, keeping all processors busy, and with minimal communication, may not be the fastest method of getting the results. An algorithm may seem wasteful of time for a particular processor array specification, yet may be the best technique at a different array size. Much of the use of Transputers in computational chemistry so far has been in the evaluation of new products, or in customised and experimental hardware configurations. Such investigations have often dealt with small processor arrays (< 10 processors), in which (as we have shown), communication bottlenecks rarely appear. However, Transputer programming is moving increasingly from evaluation into large-scale data production. As larger Transputer arrays are becoming available commercially in relatively inexpensive installations the development of robust and easily implemented parallel algorithms with general applicability becomes important.

We are confident that, when the appropriate algorithm is installed on a parallel processor array, the methods we describe approach the limit of efficiency. Our argument is that (1) if all processors are involved in a calculation, and (2) all processors are busy most of the time, and (3) there is no duplication of effort (i.e. the decomposition of the problem between procesors involves no overlap), then the method used must approach optimal speed for a given numerical technique. There are a very large number of possible approaches to encoding parallel processing of the GCE Monte Carlo method, and many of these may also approach optimal calculation speed. However, for the methods we describe, we have provided simple criteria for quantifying efficiency for particular hardware configurations.

Under these new circumstances, the pipeline and systolic-loop methods are, in principle, inefficient utilisers of communication and memory bandwidths. The systolic-loop method is particularly sensitive to an imbalance of loading, and is problematic for Monte Carlo, and especialy for the Grand Ensemble method. We have presented algorithms (in particular, methods B & E), that, we believe, can make better use of the improved hardware and are therefore, in principle, faster. When current hardware is superseded (as will be the case with the new 30 MHz and 8-link Transputers) a further development of parallel algorithms will undoubtedly be required, although we feel that the equations and arguments we present in the paper will remain valid.

Acknowledgements

We acknowledge the support of Unilever PLC. S.J. Zara has been supported by a joint Unilever/SERC grant. We wish to thank Microway (Europe) Limited for advice and technical support.

APPENDIX 1

The determination of T_c and setup time

T_c was measured using the Transputer set up shown in Figure 1. The controller processor (linked to the host computer) received concurrently messages totalling 10,000 bytes in length from each of the three server processsors. The length of individual communications was varied from 1 byte to 10,000 bytes, to determine overheads associated with each message. An Occam2 TIMER process provided the measure of the communication time. The experiment was repeated with the communications running in parallel with a numerical calculation (a series of square roots) on the controller processor, to determine the non-overlapped communication overhead (setup time) associated with each message.

The determination of T_u

We have developed a Grand Canonical Ensemble Monte Carlo Simulation program constructed using method B, which is in use for routine data production. Our requirements for the simulation include the x, y and z components of the virials in addition to the energy. For the determination of T_u the viral determinations were disabled and a TIMER process provided the measurement of T_u.

APPENDIX 2

The algorithm for trial steps in Grand Canonical Ensemble Monte Carlo Simulation. The procedures 'obtain.old.energy' and 'obtain.new.energy' (both show underlined in the Occam code) will contain the implementation of the parallel algorithms discussed in this paper. In the case of method A (re-calculation), these procedures will be identical. In the case of methods C, D & E, obtain.old.energy will look up the row (or column) totals of the pair interaction table and obtain.new.energy will determine new values for the elements of the selected particle's row and column in the tables. These values will be placed into the table if the move is successful. For method B, obtain.old.energy looks up the current particle energy, and obtain.new.energy calculates the new (trial) energy for the particle. If the trial move is successful then obtain.new.energy will be called by procedure 'adjust.system.energy', with the pre-trial coordinates of the particle. For method B, the energy calculation routines must deliver the contributions of individual pair interactions, so that, after a successful step, the energy totals of all particles in the simulation can be adjusted.

```
PROC trial.step

 BOOL decision,          -- accept or reject?
      ready :            -- particle selected for step

 INT particle,           -- which particle
   species,             -- which species
   step.type :          -- what type of step

REAL32 old.particle.energy,     -- energy before step
     new.particle.energy,   -- energy after step
     system.energy.change : -- the difference

VAL BOOL accepted IS TRUE, rejected IS FALSE :
VAL move IS 1, create IS 2, destroy IS 3 :

SEQ

  -- choose step type.  If move, there must be a particle
  -- present in the system to be moved

  ready := FALSE
  WHILE NOT ready
    SEQ
      select.step.type (step.type)
      select.particle.species (species)
      ready := (number [species] > 0) OR
               (step.type <> move)

      -- perform the selected step

IF
   step.type = move
     SEQ
       particle := select.particle (species)
       old.particle.energy := obtain.old.energy (particle)
```

```
-- store the old coords

save.particle.data (particle)

-- new coordinates

generate.new.coords (particle)
new.particle.energy := obtain.new.energy (particle)
system.energy.change := new.particle.energy -
          old.particle.energy
decision := decide (system.energy.change,
         step.type, species)

IF
 decision = accepted
  SEQ
    adjust.system.energy (system.energy.change)
    count.successful.move
 decision = rejected
  SEQ
    restore.particle.data (particle)
    count.unsuccessful.move

step.type = create
 SEQ
  -- as creation can adjust particle numbers, store
  -- current numbers for retrieval if failed step

  save.particle.numbers
  particle := new.particle (species)
  new.particle.energy := obtain.new.energy (particle)
  system.energy.change := new.particle.energy
  decision := decide (system.energy.change, step.type,
                  species)
  IF
   decision = accepted
```

```
        SEQ
          adjust.system.energy (system.energy.change)
          count.successful.creation
      decision = accepted
        SEQ
              restore.particle.numbers
              count.unsuccessful.creation

      step.type = destroy
        SEQ
          -- note: destructions that fail due to the absence of
          -- particles of the species selected are valid steps
          -- in the Markov Chain, and must be counted
          IF
            number [species] > 0
              SEQ
                -- there ARE particles of this species:
                -- attempt to remove the particle
                particle := select.particle (species)
                old.particle.energy :=
                        obtain.old.energy (particle)
                system.energy.change := - old.particle.energy
                decision := decide (system.energy.change,
                             step.tye, species)
                IF
                  decision = accepted
                    SEQ
                      adjust.system.energy (system.energy.change)
                      remove.particle (particle)
                      count.successful.destruction
                  decision = rejected
                    SEQ
                      count.unsuccessful.destruction

accumulate.system.statistics
```

References

[1] Hoare, C.A.R. "Communicating sequential processes", *Communs ACM*, **21**, 666 (1986).
[2] Pountain, D. and May, D. in A Tutorial Introduction of Occam Programming. BSP Professional Books, London. (1987)
[3] Hey, A.J.G. "Reconfigurable Transputer Networks: practical concurrent computation", *Phil. Trans. Roy. Soc. Lond.*, **326**, 395 (1988).
[4] Fincham, D. "Parallel Computers and Molecular Simulation", *Molecular Simulation* **1**, 1 (1987).
[5] Smith, W. and Fincham, D. "Parallel Molecular Dynamics Algorithms on the Daresbury Meiko M10", Information Quarterly for Computer Simulation of Condensed Phases. (Un-refereed newsletter of SERC Daresbury Laboratory). No. 27, 56 (1988)
[6] Nicholson, D. and Parsonage, N.G. in "Computer Simulation and the Statistical Mechanics of Adsorption" Academic Press, London. p151 (1982).
[7] Fox, G.C. and Otto, S.W. in Proc. Knoxville Hypercube Conference, Aug 1985 (ed M. Heath) Siam publ. (1985)