Identifying 3D maximal common substructures using Transputer networks

Andrew T. Brint and Peter Willett*

Department of Information Studies, University of Sheffield, Western Bank, Sheffield S10 2TN, UK

This paper describes the use of a multiprocessor system for identifying the maximal common substructure (MCS) between pairs of three-dimensional (3D) chemical structures. The system is constructed from Transputers, 32-bit RISC microprocessors produced by Inmos Ltd., linked together in a tree network. The MCS algorithm used, developed by Crandell and Smith, identifies the MCS by a breadth-first search in which individual atoms common to the structures are extended one atom at a time until no further extension of the common substructure can be obtained. Experiments using a Pascal-based simulation package demonstrate the feasibility of using a multiprocessor system to increase the speed of MCS identification. Experiments with networks of Transputers demonstrate that substantial increases in speed can be achieved in practice if, and only if, the MCS is large.

Keywords: atom-by-atom searching, transputer networks, maximal common substructure, parallel processing

Received 3 August 1987
Accepted 26 August 1987

The increasing computational demands of applications such as protein crystallography, molecular modeling, and chemical graphics has led to interest in the use of parallel computer hardware for processing a variety of types of chemical information. Many parallel computers have been designed or built, although the way in which parallelism is manifested differs drastically from one machine type to another. Thus, pipelining and multiple functional units have been used to increase the efficiency of uniprocessors, while a few sophisticated uniprocessors can be coupled together via the sharing of global memory, as in commercial multiprocessor systems. Array processors, conversely, contain many simple processing elements, all of which execute the same instructions on their local stores. Other architectures under active investigation include associative processors and data-flow machines. Overviews of research and development in this area are presented by Zakharov[1] and by Hwang and Briggs.[2]

Most scientific applications of parallel processing have been used to increase the efficiency of numeric computations of various sorts. However, parallel-processing techniques can also be used for processing nonnumeric data. In this paper, we are interested in the use of microprocessor-based, multiprocessor networks for processing three-dimensional (3D) chemical structure information. In a multiprocessor system, each processor in the network is fully programmable and can execute its own program while communicating with others over the network; a multiprocessor system is thus an example of a multiple-instruction stream, multiple-data stream (MIMD) architecture.[3]

Recently, several types of 32-bit microprocessors have become available that can be used as building blocks for constructing multiprocessors that offer the prospect of extremely high processing rates at cost levels substantially less than those associated with conventional mainframes or supercomputers. An example of a microprocessor building block is the Transputer produced by Inmos Ltd., which has been designed specifically for concurrent processing.[4,5] A Transputer contains a high-performance RISC (reduced instruction set computer) processor, local memory, and multiple communication links, called channels, that provide direct connections to other Transputers and to external memory and backing storage. A programming language called Occam has been developed not only for the design of concurrent systems based on linked microprocessors but also for programming the individual microprocessors in the network. Thus, a program can be developed on one Transputer and implemented on a network of Transputers, linked together in the configuration that seems most appropriate for the particular application.[6] See the Appendix for details of the Transputer and of Occam.

Our interest in multiprocessor systems stems from a paper by Wipke and Rogers that considers the simulation of a star network of multiprocessors to increase the efficiency of atom-by-atom searching in chemical substructure search systems.[7] In such systems, a query substructure (i.e., a set of atoms and the bonds connecting them together) must be mapped onto each of the structures in a database to determine whether the substructure is present. A simulation approach has also been used in our department to investigate an alternative method of substructure searching called relaxation searching.[8] This involves comparing a query substructure with a structure in a database by iteratively examining correspondences between the two sets of atoms; the aim is to reduce the number of possible correspondences that must be considered in the final, atom-by-atom

*To whom all correspondence should be addressed

 0263-7855/87/040200-08 $03.00

search. In both simulations, the results suggested that the inherent parallelism in the problem could be mapped onto the processors in a multiprocessor system so as to increase substantially the efficiency of substructure searching. Similar conclusions have been obtained from a simulation study of tree-based, nearest-neighbor searching in textual databases.[9] More recently, we have tested the relaxation searching technique using networks of real Transputers, and we have demonstrated near-linear speed-ups with networks containing up to 10 Transputers.[10]

To date, studies of the use of multiprocessor systems for handling chemical-structure information have considered only two-dimensional (2D) representations of molecular structure. In this paper, we investigate the potential of Transputer networks for handling 3D chemical structures — specifically, the computationally demanding task of identifying the largest substructure common to pairs of 3D molecules.

IDENTIFYING MAXIMAL COMMON 3D SUBSTRUCTURES

Several studies have been reported of computational techniques that can be used to identify the maximal common substructure (MCS) for a set of molecules.[11-16] MCS identification is highly demanding of computer resources. Thus, Levi shows that identifying all substructures containing K atoms that are common to two structures containing M and N atoms, respectively, requires, at most,

$$M!N!/K!(M-K)!(N-K)!$$

atom-by-atom comparisons.[11] However, the task is of considerable practical interest, since the MCS for a set of structurally disparate 3D molecules that exhibit a common activity may represent, or at least contain, the pharmacophoric pattern (i.e., the geometric pattern of atoms responsible for the observed activity).

The MCS algorithm studied here is that described by Crandell and Smith;[15] it involves taking all the common substructures of size N atoms associated with each molecule in the set of molecules that is being processed and adding an extra atom to each such substructure. These enlarged substructures are then canonically named so as to allow them to be compared rapidly with the enlarged substructures associated with all of the other molecules in the data set. If a substructure is not found in all of the other molecules' lists, it is deleted from consideration. The surviving substructures form the common substructures of size $N + 1$; these go forward to the next iteration of the algorithm.

The selection of an atom to add to a substructure in the growing step is done by consulting a distance matrix associated with each molecule. This matrix contains the distances between all of the atoms in the molecule. Negative distances in the matrix are those distances not included in the current set of common substructures, while atoms associated with positive distances are available for addition to the common substructures in the next growth phase of the algorithm. Thus, the algorithm consists of the following basic steps:

(1) Create the initial distance tables; set $N := 0$.
(2) Grow and name the substructures of size $N + 1$ atoms from the common substructures of size N atoms identified in the previous iteration.
(3) Compare the substructures to identify those that all molecules still have in common; if there are none, stop.
(4) Amend the distance tables and return to Step 2.

In more details, these stages are as follows:

Creating the distance tables

The comparison of the substructures in Step 3 is implemented efficiently by representing each of the interatomic distances in each molecule by a single integer code. This is accomplished by the following clustering procedure. A list is formed of the interatomic distances present in the molecules for each pair of atomic types present. These lists are then sorted into ascending order and the distances in them clustered together so that a distance belongs to the same cluster as its predecessor if the difference in their values is less than the tolerance value, which is taken to be 0.09 Å, as in the original paper;[15] otherwise, a new cluster is formed. The clusters are then numbered starting from one, and groups that that not contain atom pairs from every molecule have their numbers negated. A distance matrix, D, is associated with each molecule, and the element $D[I,J]$, $I <> J$ contains the group number for the interatomic distance between this molecule's Ith and Jth atoms; these group numbers are referred to subsequently as distances.

Growing and naming

Each substructure associated with a molecule is grown by enlarging its atom set by one. This is effected by numbering each atom in a structure from one up to the number of (nonhydrogen) atoms present, then adding an atom with a number greater than any of the atoms in the current set of atoms that make up a common substructure and whose distances in the distance table to these atoms are all positive. Where it is possible to add several different atoms, a new substructure is produced for each of them. In the first iteration, the grown substructures are the individual atoms in each molecule. Thus, the common structures are grown by a breadth-first tree-search procedure. Each node set (i.e., a set of atoms that make up a common substructure) produced in the growth stage is given a canonical name by taking each of the $N(N-1)/2$ atom pairs in the substructure (where N is the size of the substructure) and forming a triple consisting of the two atom types (with the larger coming first) and the relevant distance entry in the distance table. Once these $N(N-1)/2$ triples have been sorted, each triple need only be represented by its distance element, since this implicitly contains the atom type information.

Comparing

The named substructures for each of the molecules are compared with those of the other molecules. If another molecule is found that does not contain this substructure, then the substructure is deleted; hence, the N-atom substructures surviving this step are those that all molecules have in common.

Amending the distance tables

After the comparison stage, each nonnegative entry in the distance tables has its atom pair checked to see whether it still occurs in the relevant molecule's list of node sets. If it does not, the distance entry is negated so as to avoid growing substructures that contain this atom pair at some later point in the procedure.

A characteristic of the Crandell-Smith algorithm is that it involves many largely independent computations; accordingly, it seems possible that it might be appropriate for implementation on a multiprocessor system, where the independent computations could be executed in parallel. We have investigated this hypothesis in two stages. Our initial experiments used a simulation package to simulate the operation of the algorithm using a pool of microprocessors. We undertook this simulation primarily to determine whether there was any scope for parallel processing in the Crandell-Smith algorithm. As will be seen, the results of the simulation have proved sufficiently encouraging to proceed to an actual Transputer implementation as the requisite hardware and software have become available to us.

SIMULATING THE CRANDELL-SMITH ALGORITHM

The simulation was carried out using PASSIM, a Pascal simulation system developed in the Division of Economic Studies at Sheffield University.[8,9,17] PASSIM produces Pascal code from a description of a simple queueing system in which processes move elements (i.e., computational tasks) from one queue to another. The time that the processes take can be specified to be modeled by various probability distributions, including the normal and negative exponential distributions. The simulation model was analogous to that used by Wipke and Rogers[7] and by Gillet *et al.*[8] in that the processors communicated by using a region of shared memory.

The serial version of the algorithm for two molecules is shown in Figure 1, and a corresponding parallel version is shown in Figure 2. The only stage of the algorithm that requires interaction with the other molecule's growths or distance table is the comparison stage; the processing is synchronized by ensuring that neither set of comparisons can start until the other molecule has finished the naming stage.

Each box in Figure 2 corresponds to one of the principal stages of the Crandell-Smith algorithm and represents a PASSIM process. In fact, except for the initialization box, each of the boxes was set to represent 50 identical PASSIM processes, thus allowing up to 50 processors to be working on the same stage of the algorithm at the same time. Each process takes a processor from a queue when it starts and returns it to the queue when it finishes; the initial size of this processor queue is one of the parameters of each simulation run and represents the number of processors in the multiprocessor system. In addition, each stage was split up into a user-specified number of subtasks.

The CPU time taken for each step of each generation of the algorithm was found by running a FORTRAN 77 version of the serial algorithm on a Prime 9950 computer. It was assumed that each stage was made up of a large number of small processes whose durations formed a normal distribution; sampling theory for

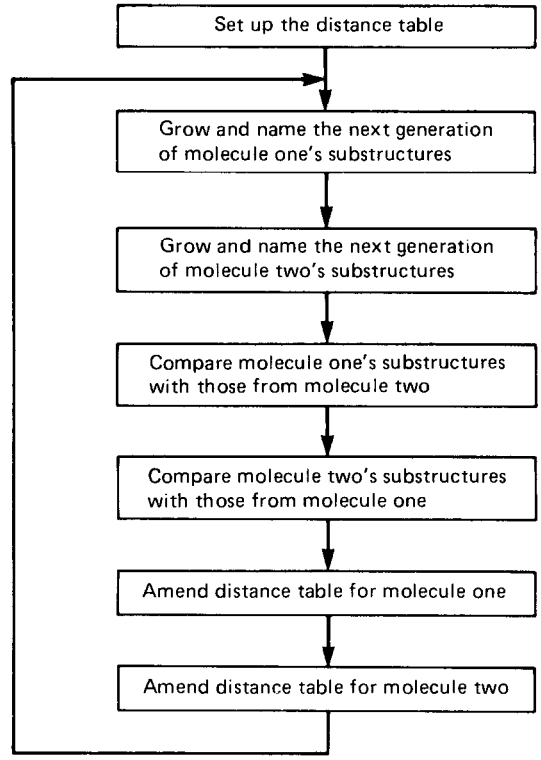

Figure 1. The serial form of the Crandell-Smith algorithm for comparing two 3D molecules

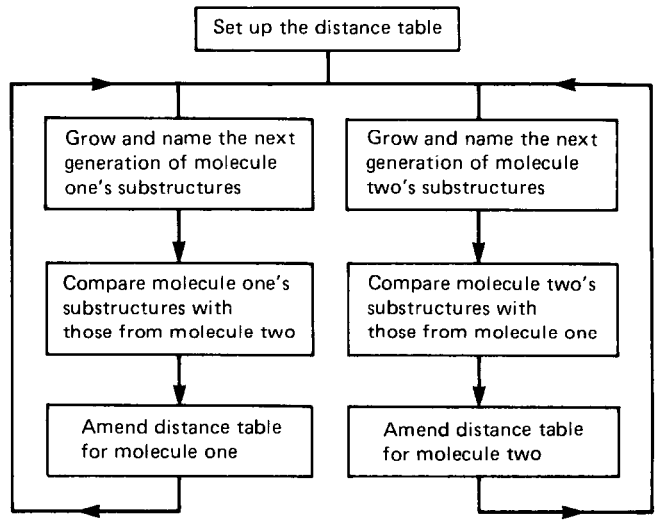

Figure 2. The parallel form of the Crandell-Smith algorithm for comparing two 3D molecules

normal distributions could then be used to obtain the times required for each of these processes. Brint discusses the limitations of such an approach;[18] however, as noted above, we undertook the simulation merely to determine whether it was possible to introduce some degree of parallelism into the identification of an MCS, and we felt that the limitations would not be such as to invalidate the results of such a qualitative approach.

Apart from variations in the number of processors in the pool, it was also possible to model variations in the data-transfer rate between the shared memory and the processors' local memories, in the overhead incurred every time a task is allocated to a processor, and in the number of jobs corresponding to each of the boxes in Figure 2. Full details of the simulation are presented by Brint.[18]

Table 1. The relationship between the speed-up and the number of processors in the simulation experiments

Number of processors	Common substructure size		
	14	12	9
1	1.00	1.00	1.00
2	1.68	1.45	1.41
3	2.28	1.90	1.88
4	2.55	2.22	2.17
5	2.89	2.51	2.43
10	3.99	3.33	3.06
20	5.72	5.37	4.34
30	7.05	6.51	5.00
40	8.09	7.46	5.32
50	8.41	8.05	5.69

The effect of variations in the number of processors on system efficiency was measured by the speed-up. If the execution time of a program using a network of p processors is $T(p)$, then the speed-up for p processors, $S(p)$, is given by

$$S(p) = T(1)/T(p)$$

The algorithm was tested using molecules taken from the Cambridge Crystallographic Data Bank. A pair of structurally related molecules was obtained by selecting a single molecule and then reordering the atoms and distorting some of the interatomic distances to obtain a pair of molecules with a substantial, and controllable, degree of structural overlap.

Table 1 lists the speed-ups obtained using pairs of molecules containing 14, 19, and 14 nonhydrogen atoms, with MCSs of 14 (i.e., structural identity), 12, and 9 atoms, respectively. The figures listed are those obtained assuming zero-time overheads for allocating processors to processes. It is clear that the speed-up increases in a sublinear manner with an increase in the number of processors available; the maximum degree of speed-up obtained decreases in line with the size of the MCS that is to be identified (i.e., in line with the computation required for the identification). Thus, the efficiency of the multiprocessor system is maximized when the greatest amount of computation is involved in the processing (as opposed to interprocessor communication, initialization overheads, etc.). The speed-up also increases with the degree of parallelism; thus, identifying a size-11 MCS in a set of three 14-atom molecules resulted in a 50-processor speed-up of 12.41, which is substantially larger than the speed-ups obtained when only two molecules were compared.

The large number of parameters that needed to be specified or estimated meant that the simulation was unlikely to yield precise figures for the efficiency of an actual multiprocessor implementation; however, the results we obtained sufficiently encouraged us to proceed to an implementation of the Crandell-Smith algorithm on a network of real Transputers.

IMPLEMENTING THE TRANSPUTER NETWORK

The implementation of the Crandell-Smith algorithm used a tree-shaped network of Transputers. The various

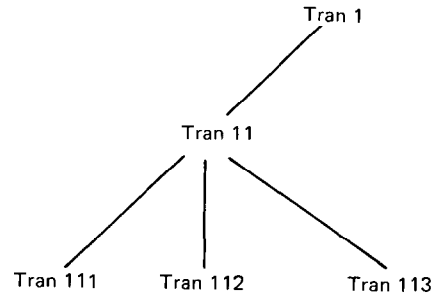

Figure 3. A five-Transputer branch of the network

stages of the algorithm were broken up into jobs in an analogous way to the simulation, and these jobs were then distributed to the individual Transputers in the tree. We chose a tree structure, as opposed to other configurations such as a linear chain or a ring, since this topology minimizes the number of intermediate nodes that a message from the root to a node in the network must pass through before reaching the target node (and similarly for returning a result from a node to the root). A discussion of the communication characteristics of various types of Transputer networks is given by Lynch *et al.*[10]

Figure 3 shows five Transputers making up a branch of the tree. Assuming that each job consists of reading a number from an array on Tran1, sending it to a Transputer, undertaking a calculation using this number and returning the result to Tran1, then the Occam executed by Tran11 can be split up into the following three processes (which all execute in parallel).

The CHECKING process is used to access a three-element array containing information on whether Tran111, Tran112, and Tran113 are waiting for another data element to be sent, are executing, or have closed down as all the data has been processed. The array is stored in a process so that it is protected from simultaneous accesses from processes executing in parallel; these processes have to communicate with CHECKING via channels (as described in the Appendix).

The RECEIVE_FROM_TRAN1 process receives data from Tran1, asks the CHECKING process for the name of a free Transputer, and then sends the data to this Transputer.

The RECEIVE_FROM_TRAN11* process receives data from Tran111, Tran112, or Tran113, informs CHECKING that this Transputer is waiting for further data, and then passes the data back to Tran1. Tran111, Tran112, and Tran113 receive just data elements, perform the calculation, and then transmit the result, while Tran1 receives data, stores it, and sends out the new data. The code avoids deadlocks because data is sent only to Tran11 if one of Tran111, Tran112, or Tran113 can receive it. Hence, RECEIVE_FROM_TRAN1 never has to wait to input or output (apart from temporarily waiting its turn for CHECKING). Consequently, no circle of processes halted in midexecution can occur, so there cannot be a deadlock. RECEIVE_FROM_TRAN11* informs CHECKING that a Transputer has finished before sending the data to Tran1; this eliminates the possibility of Tran1 sending the next piece of data to this Transputer before its flag has been set and then being reset by RECEIVE_FROM_TRAN1. Once Tran1 has sent out all of the data that are to be processed, the tree is closed down

by sending out a flag, instead of a fresh data element, every time that Tran1 receives a result.

The basic structure shown in Figure 3 can be extended by adding further Transputers and then having Tran1 and Tran11 perform the same processing functions as Tran111, Tran112, and Tran113 in parallel with their administrative activities; in our experiments, trees that contained up to 11 Transputers were used. The addition of further Transputers involves changing the arrays in CHECKING so that they indicate the number of free Transputers down each of the branches (and making corresponding changes in the RECEIVE processes so that these elements are incremented/decremented instead of being set or reset).

The Compare and Amend stages in the main loop of the Crandell-Smith algorithm were split into x and y (where x and y could be varied), separate jobs that were then distributed over the tree. The initial Grow/Name stage was divided into only two jobs. The major reason for this is that growing a common substructure can lead to the generation of many candidate substructures. This causes a problem, since extra storage has to be allocated to take account of the worst possible case, and the resulting memory requirements can be a substantial constraint on the largest MCS that can be identified using the Crandell-Smith algorithm; this point is discussed by Brint and Willett.[16] The situation is exacerbated by having several processes executing in parallel on one Transputer, since this necessitates the storage of multiple copies of the large data structures used by the algorithm. In retrospect, the experimental results show that this stage might have been further subdivided, especially when the MCS is small.

In addition to distributing these jobs (and receiving their output), the Transputer at the root of the tree also sends out global data in between the stages to keep the other Transputers informed of the progress of the algorithm. The first of these stages is to send the distance table to the other Transputer involved in the growing and naming stage. The other two distributions are the sending out of the named growths (i.e., the $N(N-1)/2$ interatomic distances for generation N and the corresponding sets of n atoms) and the sending out of the compared growth sets. These data are distributed to all of the Transputers in the network. These three types of data distribution are referred to as Send1, Send2, and Send3 in Table 2.

The program was developed in Occam on a single Transputer connected to an RM Nimbus microcomputer. Once the program was fully debugged, it was extended to execute on the network, with various numbers of Transputers connected to the Nimbus.

The algorithm was run using a distorted molecule of size 14 (C_9N_4O) with a common substructure of size 12, an undistorted molecule of size 11 (C_5NO_5), an undistorted molecule of size 8 (C_6Br_2) and the same molecule distorted to have a common substructure of size 7. The number of Transputers was varied for each structure and x and y, and the numbers of jobs into which the Compare and Amend stages were split were also varied so as to try to produce the lowest possible times. The results are given in Tables 2 (a)–(d), where the entries are the lowest obtained for the given structures using the stated number of transputers (i.e., the Compare and Amend entries might have been obtained using different values for the number of jobs the algorithm was split into).

The relevant values for the overall speed-up are listed in Table 3. Owing to the use of the optimal x and y values, it is sometimes possible to obtain speed-up values $S(p)$ such that $S(p) > p$. In addition, the speed-up for the comparison stage can be greater than linear, depending upon the exact order in which the comparisons are undertaken. This factor also affects the processor-use figures in Table 4.

The figures in Table 2 show the huge computational requirements of the Compare stage of the algorithm when the MCS is large. As noted previously,[16] this means there is substantial scope for increasing the efficiency of MCS identification, and the degree of speed-up for this stage varies in an almost linear manner with the number of Transputers available. However, with smaller MCSs, the Compare stage becomes less important relative to the other stages of the algorithm.

The efficiency of the Set Up stage, where the distance tables are created, is independent of the number of processors, since it is always carried out on the Transputer at the root of the tree. The Grow/Name stage, however, was split into only two jobs; there is thus an approximately twofold speed-up for this stage for all networks containing more than a single processor. For the smaller MCSs, this stage takes a nontrivial fraction of the total computer time; in retrospect, it might have been worth splitting this stage of the processing into more than two subtasks.

The communication costs Send2 and Send3, which represent the distribution of the growth sets over the network, increase with the number of processors; however, the fraction of the total time is less than 18 per cent, even in the case of the size-7 MCS with the full set of 11 Transputers.

The conflicting effects of the various stages of the algorithm are summarized in the overall speed-ups listed in Table 3, where it is clear that the efficiency of the network falls off very rapidly as the size of the MCS decreases. When the MCS is large and when the computational demands of the Compare stage dominate the other stages of the algorithm, it is possible to achieve substantial increases in system performance. Thus, the results obtained here are in accordance with the general observation[19] that the efficiency of a multiprocessor system is largely determined by the *granularity* of the problem that is being studied (i.e., the relative amounts of computation and communication required).

It is hard to compare these results with those obtained from the PASSIM simulation described previously; thus, the simulation adopted the simplifying assumption of a shared memory multiprocessor system, the clock times used for the length of the PASSIM activities were taken from a Prime 9950, and fairly crude estimates had to be made for the probability distributions of the various steps of the algorithm. While the simulation clearly underestimates to some extent the increase in speed-up that can be achieved if there is a large common substructure (e.g., predicting a speed-up of 3.33 for a 12-atom MCS with 10 Transputers, as opposed to an observed speed-up of 7.72), the two sets of results are in fair agreement in predicting the largest degree of speed-up when a large MCS needs to be identified.

An alternate way to measure the performance of a

Table 2. The relationship between processor requirement (in units of 16 milliseconds) and the number of processors for MCSs of varying sizes of atoms

	Number of Transputers										
	1	2	3	4	5	6	7	8	9	10	11
	12 atoms										
Set Up	11	11	11	11	11	11	11	11	11	11	11
Grow/Name	1379	724	724	724	724	724	724	724	724	724	724
Amend	722	380	349	228	242	213	223	216	214	213	216
Compare	25477	11279	7067	5286	4254	3594	3142	2882	2588	2280	2175
Send1		3	3	3	3	3	3	3	3	3	3
Send2		119	144	157	235	235	259	259	260	260	260
Send3		32	39	43	63	63	63	66	66	66	69
Total	27590	12656	8417	6511	5538	4844	4449	4125	3881	3575	3457
	11 atoms										
Set Up	7	7	7	7	7	7	7	7	7	7	7
Grow/Name	590	312	313	313	313	312	312	312	312	312	312
Amend	148	114	91	75	70	69	71	67	56	57	51
Compare	5490	2472	1580	1184	998	878	803	718	648	622	601
Send1		2	2	2	2	2	2	2	2	2	2
Send2		50	62	68	99	99	108	105	110	110	110
Send3		14	18	19	28	28	28	30	30	30	31
Total	6235	2985	2080	1666	1512	1397	1333	1240	1165	1140	1116
	8 atoms										
Set Up	4	4	4	4	4	4	4	4	4	4	4
Grow/Name	40	24	24	24	24	24	24	24	24	24	24
Amend	17	14	12	11	11	12	13	12	12	12	12
Compare	78	44	34	27	26	26	27	24	22	21	20
Send1		1	1	1	1	1	1	1	1	1	1
Send2		4	6	6	8	8	8	9	9	9	9
Send3		2	2	2	3	3	3	3	3	3	3
Total	139	92	83	75	77	77	79	77	75	74	73
	7 atoms										
Set Up	4	4	4	4	4	4	4	4	4	4	4
Grow/Name	16	11	11	11	11	11	11	11	11	11	11
Amend	11	10	9	8	7	8	8	9	9	8	9
Compare	23	16	12	10	10	10	10	9	9	9	9
Send1		1	1	1	1	1	1	1	1	1	1
Send2		2	3	3	4	4	4	4	4	4	4
Send3		1	1	1	1	1	1	2	2	2	2
Total	54	43	40	38	38	39	39	40	38	38	39

multiprocessor system is by means of the *processor utilization*, which represents the extent to which the available processors execute useful work. If the speed-up for p processors, $S(p)$, is as defined previously, the corresponding processor utilization, $U(p)$, is defined as

$$U(p) = S(p)/p.$$

Table 4 lists the processor utilization figures corresponding to the speed-ups listed in Table 3. Note that a few of the calculated values are greater than unity, because, as noted previously, some of the $S(p)$ values are greater than p. The figures in Table 4 reinforce the conclusions that can be drawn from Table 3 in that the multiprocessor system achieves a high throughput of computational work only when the MCS is large; if this is not the case, the available processors are significantly underutilized.

CONCLUSIONS

We have considered the use of a tree-shaped multiprocessor system to increase the efficiency with which 3D MCSs can be identified using the Crandell-Smith algorithm. The experimental results confirm initial simulation experiments by demonstrating that considerable increases in speed can be obtained as additional Transputers are added into the network. However, this is likely to be worth doing only if the MCS is large when the

Table 3. The relationship between the speed-up and the number of processors in the Transputer experiments

Number of Transputers	Common substructure size			
	12	11	8	7
1	1.00	1.00	1.00	1.00
2	2.18	2.09	1.51	1.26
3	3.28	3.00	1.67	1.35
4	4.24	3.74	1.85	1.42
5	4.98	4.12	1.81	1.42
6	5.70	4.46	1.81	1.38
7	6.20	4.68	1.76	1.38
8	6.69	5.03	1.81	1.35
9	7.11	5.35	1.85	1.42
10	7.72	5.47	1.88	1.42
11	7.98	5.59	1.90	1.38

Table 4. The relationship between processor use and the number of processors in the Transputer experiments

Number of Transputers	Common substructure size			
	12	11	8	7
1	1.00	1.00	1.00	1.00
2	1.09	1.05	0.76	0.63
3	1.09	1.00	0.56	0.45
4	1.06	0.94	0.46	0.36
5	1.00	0.82	0.36	0.28
6	0.95	0.74	0.30	0.23
7	0.89	0.67	0.25	0.20
8	0.84	0.63	0.23	0.17
9	0.79	0.59	0.21	0.16
10	0.77	0.55	0.19	0.14
11	0.73	0.51	0.17	0.13

comparison stage of the algorithm dominates the computation; for small MCSs, the other, less demanding stages of the algorithm result in only slight increases in the overall efficiency of the system as the number of Transputers is increased. Thus, the results support the use of multiprocessor systems only when the matching operations are extremely demanding of computational resources.

ACKNOWLEDGEMENT

We thank Val Gillet for assistance with PASSIM, Gordon Manson and George Wilson for assistance with the Transputer system and the Science and Engineering Research Council for the award of a Research Studentship to A.T.B.

REFERENCES

1 Zakharov, V. Parallelism and array processing. *IEEE Trans. Computers,* 1984, **C-33,** 45–78
2 Hwang, K., and Briggs, F.A. *Computer Architecture and Parallel Processing.* McGraw-Hill, New York, 1984
3 Flynn, M. Some computer organizations and their effectiveness. *IEEE Trans. Computers,* 1972, **C-21,** 948–960
4 Walker, P. The Transputer. *Byte,* 1985, **10(5),** 219–235
5 Barron, I. M. The Transputer and Occam, in Kugler, H. J. (ed.), *Information Processing 86.* Elsevier Science Publishing B.V., Amsterdam, 1986
6 May, D., and Taylor, R. Occam — an overview. *Microprocessors and Microsystems,* 1984, **8,** 73–79
7 Wipke, W. T. and Rogers, D. Rapid subgraph search using parallelism. *J. Chem. Information and Computer Sciences,* 1984, **24,** 255–262
8 Gillet, V. J. *et al.* Computer storage and retrieval of generic chemical structures in patents. 7. Parallel simulation of a relaxation algorithm for chemical substructure search. *J. Chem. Information and Computer Sciences,* 1986, **26,** 118–126
9 Stewart, M. and Willett, P. Nearest neighbour searching in binary search trees: simulation of a multiprocessor system. *J. Documentation,* 1987, **43,** 93–111
10 Lynch, M. F. *et al. The Application of Reconfigurable Microprocessors to Information Retrieval Problems.* Report to the British Library Research and Development Department, London, 1987
11 Levi, G. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo,* 1972, **9,** 341–352
12 McGregor, J. J., and Willett, P. Use of a maximal common subgraph algorithm in the automatic identification of the ostensible bond changes occurring in chemical reactions. *J. Chem. Information and Computer Sciences,* 1981, **21,** 137–140
13 Golender, V., and Rozenblit, A. *Logical and Combinatorial Algorithms for Drug Design.* Research Studies Press, Letchworth, 1983
14 Kuhl, F. S. *et al.* A combinatorial algorithm for calculating ligand binding. *J. Computational Chem.,* 1984, **5,** 24–34
15 Crandell, C. W., and Smith, D. H. Computer-assisted examination of compounds for common three-dimensional substructures. *J. Chem. Information and Computer Sciences,* 1983, **23,** 186–197
16 Brint, A. T., and Willett, P. Algorithms for the identification of three-dimensional maximal common substructures. *J. Chem. Information and Computer Sciences* (in press)
17 Shearn, D. C. S. *PASSIM — a PASCAL Simulation System.* Division of Economic Studies, University of Sheffield, 1982
18 Brint, A. T. Matching algorithms for handling three-dimensional molecular coordinate data. PhD thesis, University of Sheffield, 1987
19 Kung, H. T. The structure of parallel algorithms. *Advances in Computers,* 1980, **19,** 65–112

APPENDIX

Transputers are microprocessors that can communicate with their neighbors by using channels, thus allowing a powerful multiprocessor to be built by linking individual Transputers together in a network. In more detail, a Transputer, specifically the Inmos model T414A, consists of:

- A 32-bit RISC microprocessor

- Four links that each have an input and an output channel for connection to another Transputer
- 2Kbytes of on-chip RAM, with a maximum data-transfer rate of 80 megabytes per second
- A memory interface to off-chip RAM with a maximum data-transfer rate of 25 megabytes per second
- Various extra pins for use in booting the system, error tracing, and so on.

These components are connected using a high-speed bus.

Transputers were designed so that programs written for a particular configuration of Transputers can be executed on any other network of Transputers. This is achieved by allowing a Transputer to carry out two or more Occam processes in parallel by time slicing; the time slice for the T414A is approximately 800 microseconds.

Occam is a high-level assembler that is similar to Pascal but does not have some of the more elaborate data structures, such as records and pointers. Although the language was developed principally for the Transputer and is pitched at a level just above assembly language for ease of compilation, it is also intended to be used on other concurrent systems. Blocks of code (i.e., the equivalent of code between the delimiters BEGIN and END in Pascal) are labeled as processes in Occam, and a named process is analogous to a procedure in Pascal. Several of Occam's facilities were designed specifically to support parallel processing.

In conventional languages, such as Pascal, a code segment of the form

```
BEGIN
   ProcA ;
   ProcB
END ;
```

(where ProcA and ProcB are two procedures) means "execute ProcA, and when it has finished, execute ProcB" (i.e., sequential execution of the two procedures). While allowing sequential execution, Occam also lets you execute the two processes simultaneously. This is done by using the PAR (or parallel) construct, as in

```
PAR
   ProcA
   ProcB
```

Sequential execution is achieved by replacing PAR by SEQ.

A process is not allowed to change variables that are being used by processes operating in parallel with it, so as to avoid competition for variables and hence for nondeterministic execution. Communication between parallel processes has to be carried out using channels; they appear as a line containing the channel name, information as to whether the data is being sent or received and the name of the data element. On reaching such a line, the execution of a process is suspended until the process that is being communicated with reaches its corresponding communication line.

Whereas the PAR construct allows two or more processes to execute in parallel, the ALT construct allows input to be selected from one of several channels. For example, the code

```
ALT
  ChannelA ? DataA
    SEQ
      BodyOfCodeForA
  ChannelB ? DataB
    SEQ
      BodyOfCodeForB
```

will receive code from ChannelA and then execute the associated code if the data is waiting on ChannelA when the ALT is first met. If no data is waiting on ChannelA but data is waiting on ChannelB, it is received and the relevant body of code is executed; otherwise, the process waits at the ALT until data is available on one of the channels.