PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Very-High-Performance Multiple-Instruction Multiple-Data Applications [and Discussion]

C. J. Elliott and J. H. Davenport

Phil. Trans. R. Soc. Lond. A 1988 **326**, 471-479
doi: 10.1098/rsta.1988.0098

Phil. Trans. R. Soc. Lond. A **326**, 471–479 (1988) [471]
Printed in Great Britain

Very-high-performance multiple-instruction multiple-data applications

By C. J. Elliott
Smith Associates, Surrey Research Park, Guildford GU2 5YP, Surrey, U.K.

The MIMD (multiple instruction multiple data) class of multiprocessors offers, in principle, the capability to bring the resources of an arbitrary number of processors to a problem. This confers the dual advantages of very high performance and of fault-tolerance.

A number of successful applications will be presented, from which some general conclusions are drawn on the practical feasibility of the use of MIMD machines.

1. Introduction

High-performance multiple instruction multiple datastream (MIMD) computers have represented a 'holy grail' of computing research for many years. In this context, performance can be interpreted in terms of a number of attributes. Four such attributes are considered in this paper: (1) processing speed; (2) integrity; (3) verifiability; (4) flexibility. There has been a major technological development during this decade which has made the implementation of high-performance MIMD computers feasible. This is the joint development of the OCCAM language and the Inmos transputer.

The OCCAM language evolved from the concepts of Communicating Sequential Processes. Its key features are:

(1) it is very simple;
(2) parallelism is fundamental;
(3) parallel processes do not share variables;
(4) parallel processes communicate via tightly defined channels;
(5) parallel processes only synchronize when they communicate.

The Inmos transputer is a single chip computer designed to implement the OCCAM language. Its key features are:

(1) hardware is included to allow interchip communications to be handled as interprocess communications;
(2) context switch is done by simple hardware which imposes low processing overheads;
(3) the central processor unit uses a RISC-like architecture to achieve high speed and efficient execution of compiled code;
(4) transputer systems require a low chip count.

This paper presents examples of high-performance MIMD applications using OCCAM and transputers. It aims to provide an illustrated overview rather than a detailed description of each of the applications. All of the applications described are based on work by Smith Associates.

2. Applications

2.1. *Processing speed*

An example of the use of occam and the transputer to achieve high processing speed has been the work on automatic fingerprint recognition (afr) for the Scientific Research and Development Branch of the U.K. Home Office.

Manual techniques for recognizing a fingerprint use a description of the pattern of the query print in general terms, such as arch or whorl, to shortlist the set of records to be searched. afr techniques do not do this. They depend for their operation of the detailed matching of the positions of the minutiae in the print. Reliable minutiae are the ends of and forks in ridges. Other features, such as scratches or sweat pores, vary with time and are thus unreliable descriptors.

afr involves two separate activities, image encoding and database searching. Both show significant benefits, in terms of processing speed per unit cost, from the use of mimd techniques. This application will be described in some detail because of its relevance to many other problems of pattern matching and database searching.

Image encoding is necessary to transform a fingerprint image into a digital representation suitable for comparison by a computer. This takes place in four stages:

(1) image enhancement, to generate a binary image of the ridges of the print from the grey-scale source image;

(2) feature detection, to find reliable minutiae which characterize the fingerprint;

(3) false feature removal, to eliminate features arising from dirt, damage or sweat pores;

(4) image coding, to describe the relative location of the features in a manner invariant under likely geometric distortions.

Human interaction is needed to check the accuracy of this process. To provide an efficient tool for the human operator, it is necessary to achieve simultaneously high throughput and low latency. This double constraint requires that many processors cooperate on the encoding of one image.

A convenient geometry for the first two stages is that of a rectangular mesh of processors, with each processor handling the region of the image corresponding to its geometric position in the mesh. The image data is distributed as a set of rectangular patches, each slightly larger than the geometrical area of the image to be processed. This minimizes interprocessor communications by providing all of the data needed to analyse up to the edge of the geometric region on one processor.

Stage 3 is done with data largely confined to the immediate neighbourhood of the feature of interest, and thus maps with reasonable efficiency on to the geometry used for stages 1 and 2. Some techniques for false-feature removal require the formation of 'closed sets' of features to, for example, identify corresponding ridges on opposite sides of a long scratch cutting many ridges. These closed sets frequently require data on features located on different processors to be considered simultaneously.

False-feature removal is done by the processor on which the feature of interest is located, according to the geometric mapping defined for stages 1 and 2. This results in some inefficiency, because of the non-uniform distribution of features over processors and the need to communicate information between processors. Process mappings which minimize this, by a more even distribution of features and by permitting concurrent communication, have been

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

devised but the additional complexity is not justified by the small improvement in speed that they offer.

Stage 4 involves consideration of the relative location of features which may be widely separated in the image, and thus suffers the same efficiency penalty that was described for stage 3. It is also possible to devise process mappings for this stage which minimize this effect, but again, the additional complexity is not justified.

The encoding software was designed to use an arbitrarily sized rectangular array of processors. The available array was defined at compilation. This parametrized approach proves to be of great value during the development of a system and during subsequent extensions. This aspect is discussed in more detail in §2.4.

Database searching requires that an encoded query image be compared with many stored encoded images. The existing algorithms compute a score for the similarity between the query and each of the stored images. This process is not sufficiently precise to be sure that the image achieving the highest score corresponds to the correct match. In practice, it is necessary to examine manually the prints corresponding to a small number of the images which achieve the highest scores.

It is necessary to achieve high throughput, but, in most cases, latency is not critical. There would be little advantage in a turnaround of better than 24 hours for all but the most serious of crimes.

Under these circumstances, there is no need to arrange for many processors to cooperate on scoring one comparison. It is simpler to score one query against one member of the database at a time and to order the scores to produce a shortlist for manual searching. Important considerations in the design are:

(1) the time taken to compute the score for a particular print and a particular query can vary between 10 ms and 1s. The average time is approximately 200 ms;

(2) database storage requirements can be reduced by storing the encoded print in a compacted form, but unpacking the print data takes typically 200 ms;

(3) 'bottlenecks' can occur if too many processors require simultaneous access to one point in the database.

There are many options for the configuration of a MIMD computer for this task. Four examples are the following.

Option 1

The processors are arranged as physical pipelines, with a connection to the database at one end. Each processor has a different member of the batch of queries. Unpacking is done by the first processor in the pipeline. Database members are passed along the pipeline and each processor computes the score for its query and the database member. The scores are also passed along the pipeline and ordered at the end of the pipeline.

Option 2

The processors are all connected to a single point, to which the database is also connected. Each processor has a different query. It requests a database member, unpacks it, computes the score, orders the score and requests another database member.

Option 3

As option 1, except that each processor in the pipeline has every member of the batch of queries. Database members are passed along the pipeline but each one is only compared with the queries by one processor.

Option 4

As option 1, except that each processor has a different batch of queries instead of only one query.

Option 1 amortizes the overhead of unpacking each print data over all of the comparisons in the pipeline, but the speed of the pipeline is limited by the slowest comparison being conducted. Consideration 1 above shows that this may be very inefficient.

Option 2 can be achieved by constructing option 1 with 'bypass' software to allow work to be served on demand. It is efficient but demands a high bandwidth of the database (consideration 3 above).

Option 3 provides a useful compromise between these extremes. It relies on the mix of queries providing a range of comparison times and hence a lower probability that one processor completes its workload substantially before the others.

Option 4 combines the benefits of option 3 with a very low bandwidth being demanded of the database, but at the cost of requiring a large batch of queries to be accumulated.

Each option has different merits in terms of throughput, latency, database bandwidth and minimum query batch size. Table 1 compares these qualitatively.

TABLE 1

(H (high), M (medium) and L (low) refer to the relative merit of the four options in terms of the four measures listed. They do not indicate any absolute measures of performance.)

option	1	2	3	4
measure				
throughput	M	H	H	H
latency	M	L	M	H
database bandwidth	M	H	M	L
minimum query batch size	M	L	M	H

It is possible to construct the physical processor array and database storage media such that the choice between these options can be made at run time. This allows a number of different modes of operation to be used. For example, the system can be configured to provide a routine service for investigating local crime during normal office hours, an overnight service with a low-bandwidth tape database for non-local crime or a very low latency service for emergency use.

2.2. *Integrity*

Recent work has shown the use of a MIMD architecture for achieving high integrity (Armstrong *et al.* 1987). The work was done on behalf of the European Space Agency (ESA) and concerned the use of transputers for processing on-board spacecraft. It exploits MIMD in an unusual way, and is only feasible because of the low chip count and power consumption of a transputer array.

On-board processors are constrained by the need to be highly reliable, because of the difficulty of maintenance, and to resist the space environment, particularly the effects of radiation.

There are two significant modes of damage by radiation for chips such as the transputer, known as total dose failure and single event upset (SEU).

Total dose failure occurs when charge induced by the action of radiation accumulates in the non-conducting oxide layers. This accumulates until the changes in bias level and conductivity cause hard, i.e. irreversible, failure.

An SEU is the effect of a charged particle causing a local change to the bias on a gate and hence changing the state of a bit of data, either in a memory or a register. This is a soft error which can be recovered by rewriting the correct data to that location.

Tests on the effect of radiation done at ESA and elsewhere (Thomlinson *et al.* 1987) showed that the transputer has a high resistance to total dose effects but, like all CMOS devices, is susceptible to SEU. It was estimated that a single-bit error would occur in a transputer between once per month in low Earth orbit and twice per day in interplanetary space.

MIMD has been used to provide high integrity in the presence of SEU. Four transputers are used, each of which is connected to each other. The connection takes the form of a communications link and a 'reset' signal in each direction. Voting logic at the receive end of the reset signals will reset a transputer if at least two reset inputs are present at once.

Each processor executes the same function, although different code may be used for each to avoid systematic coding errors. At regular intervals, and before any output, the processors carry out self-test and, if healthy, exchange state vectors. If one of the state vectors differs from at least two others, the majority processors will reset the faulty device. The code for that processor and the state vector are then re-booted from one of the correct processors.

Only three processors are needed for this sequence to obtain. However, use of the fourth processor ensures that the self-repair of soft errors can continue in the presence of any one hard error. A more detailed analysis of other failure and recovery modes of this architecture is given in Armstrong *et al.* (1987).

2.3. *Verifiability*

This aspect of MIMD application has received the least attention to date but may represent one of the most important areas to which MIMD can contribute. The verification in question is the proof that a particular hardware and software combination implements identically the requirements defined by the specification.

A prerequisite of a verifiable system is a specification which is complete, i.e. one which defines the action required of the system in all circumstances. Given a complete specification, it is possible, in principle, to transform a program in a high-level language to show that it is identical to the specification. A further exercise, to verify that the compiler meets its specification, is also possible in principle, so that it is eventually possible to verify that the machine instructions generated by the compiler are identical to the specification. Even then, it is also necessary to verify that the processor executes its instructions in identically the manner for which the compiler was defined.

In practice, complete formal verification of any other than the smallest system has not proved feasible. Even the first step, the definition of a complete specification, is not feasible for large systems. Similarly, the definition of most high-level languages is too loose to allow transformation rules to be applied.

The OCCAM language has a formal basis and is subject to transformation rules which make it feasible to verify that an OCCAM program is identical to its specification. The simplicity of the language, together with the highly regular, RISC-like structure of the transputer, make it feasible to consider verifying a compiler. However, to the best of my knowledge, this has not yet been done.

A different approach is to use the MIMD properties to reduce the error rate of a system and to bound the damage caused by those errors. Given that processes A and B can only communicate via a channel, it is only necessary to control the effect of the channel on process A to control the effect of process B on process A. This has been used to good effect in an extension of the high-integrity system described in §2.2.

The four processors described above form a high-integrity kernel. Their operation can be tested extensively and the OCCAM code could in principle be verified. It is necessary to introduce a further layer of software to execute an application, and this must be done in such a way as to ensure that this application software cannot interfere with the operation of the kernel.

In principle, OCCAM provides for this by ensuring that parallel processors cannot share memory and only communicate via channels. Provided that the application programmer restricts himself solely to valid OCCAM constructs, and always leaves array-bound checking in operation, OCCAM may provide adequate separation between parallel processes running on a single transputer.

However, these conditions cannot be enforced reliably and the transputer has no hardware check that the physical memory address generated by a process lies within the region allocated to that process.

An alternative is to use physical isolation, whereby the application code runs on a different transputer from the kernel. Each of the four transputers in the kernel has one spare communications link, to which further transputers can be connected. The application program can be run on the external transputer, communicating via the tightly controlled route of the link with the kernel. Many options now exist for providing high integrity for the application, by using several external transputers and voting on their output in the kernel.

As an aside, this approach is only viable for use on board spacecraft because the extra penalty of additional transputers is very small. The kernel and four additional transputers and their memory can be accommodated within the mass and power consumption of a single conventional space-borne computer.

The example presented above illustrates the general principle of the use of MIMD to achieve verifiability. By dividing the problem into smaller parts and closely defining the communications between those parts, it is possible to reduce greatly the complexity of the task of verification. Similarly, testing complexity is also reduced.

The logical extension of this is described in Hoare (1986) whereby the processes of program design and coding are replaced by a stepwise refinement of the division of the task into smaller subtasks.

2.4. *Flexibility*

An example of an application in which flexibility is of prime importance is the design of a processor for the generation of images from a spaceborne synthetic aperture radar (SAR). It demonstrates principles that have been common to most of the parallel processing projects that I have undertaken.

Spaceborne SAR data require extensive processing to generate a two-dimensional image of the scene viewed by the radar. The processing reduces to discrete Fourier transforms in which

the dataset is constructed from interpolation of the measured data, and the interpolated points are defined by the geometry of the radar's location and the Earth's surface. It is because the geometry is complicated by the spherical earth that spaceborne SAR processing is much more computationally intensive than airborne SAR processing, for which a flat Earth may be assumed.

A detailed design study was conducted of the implementation of the algorithm on a transputer array. It was found that real-time implementation would require a total of about 4000 T414 transputers, with a form of block floating point rather than a full floating point calculation.

Two observations follow from this conclusion. The first concerns the feasibility of a design that uses 4000 transputers and the second concerns the development cycle by which this could be achieved.

A useful order-of-magnitude rule for costing a large transputer installation is that the hardware will cost about £1000 per installed transputer. Thus, the total cost of the real-time processor would be about £4M. This is similar to the estimated cost of developing the same functionality using dedicated DFT hardware processors.

It is not unreasonable to assume that 16 transputers, together with their support devices and memory, could be accommodated on a typical circuit card. The complete real-time processor would thus occupy about 250 circuit cards. These in turn would occupy two standard 19 in × 6 ft (48 cm × 183 cm) rack units. This also is similar to the likely physical size of the dedicated hardware system.

It is thus concluded that it is no less feasible to build a real-time spaceborne SAR with transputers than to build it with dedicated hardware. Since carrying out the study, the T800 Floating Point Transputer has become available. It is estimated that this would reduce the number of transputers needed, and hence all of the other estimates above, by a factor of four.

The major difference between the approaches becomes apparent when one considers the second observation, the development cycle to be followed. The transputer approach uses many repeats of a basic structure. Both the hardware and software can be developed for a small system, unable to meet the real-time requirement, and tested extensively. The extension to faster operation is achieved by the replication of the hardware units and minor changes to the configuration statements of the software.

It is thus possible to minimize the development risk by adopting a strategy of testing the full operation of the system by using limited hardware before purchasing the full equipment. This also allows an intermediate position to be adopted in which, for example, a processor achieving one tenth of real time is used until the demand for processed data grows to justify further extensions. It is also possible to offer processors of reduced speed to users who require the same algorithm as the real-time facility but cannot afford or justify the real-time version.

This flexibility arises from the MIMD nature of the solution, together with a freedom to define the mapping of *processes* onto *processors*. It leads to a general observation about the approach to be taken in developing a system for a new application.

The general approach can be summarized as follows:

(1) divide the problem into the smallest convenient independent units;

(2) develop OCCAM code for each unit on the assumption that a separate transputer will be used for each unit;

(3) measure the run time for each unit running on a transputer;

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

(4) group processes onto transputers such that the worst-case run time for one transputer is less than the required system run time;

(5) configure the transputer network to provide the necessary communication paths.

It is important to recognize that this may not result in the most efficient use of transputer hardware. There may be overheads associated with unnecessary interprocess communications which could be eliminated if the code for each transputer were optimized for that configuration. However, it is argued that the benefits of this approach outweigh this disadvantage.

The first benefit is that of project control. The development project is divided into logical structures that are related to discrete functions. This makes it easier to manage parallel development and testing activities. The final software structure resembles a 'block diagram' of the problem.

The second benefit is the flexibility for expansion. In the event that it is necessary to increase the speed of operation, it is only necessary to add more hardware and reconfigure the software. The existing, tested code for each unit remains valid.

The only disadvantage is the cost of the 'wasted' hardware. The counter to this is to examine the relative contributions of hardware and software costs to the total cost of a major project. Bell *et al.* (1987) presents the commonly held view that the software element of a major project represents up to 90% of the cost, and that it is not uncommon for managers to underestimate the cost of software by up to 50%. Under these circumstances, it does not seem unreasonable to be extravagant with the hardware to reduce difficulties with the software.

3. Observations

This section attempts to draw together the themes running through the applications presented in §2 and to extract some general observations about high-performance MIMD processors and, in particular, systems that use OCCAM and transputers. Although the applications were presented in terms of four different attributes, the qualities exhibited are similar.

A common theme is the division of the problem. This occurs to allow many processors to work on one data block for low latency and high speed and also for high integrity. The division of the problem is presented as a basic method of verifying the correct function and of achieving flexible levels of performance.

Developing further on the theme of division, it is also essential to make the divisions on logical grounds, related to the physical or algorithmic boundaries of the problem. This includes geometric division of image processing, division by query for database searching, division by the need for independent checking for high integrity and independent proof for verifiability and division into the smallest logical units for flexibility.

Underlying the need to select an appropriate division of the problem is the need to define convenient communications paths between those elements which need to exchange data. This leads to the single conclusion of the paper, which is the need to select the MIMD architecture to suit the problem in hand. In practice, the architecture may be defined by electrically configuring a fixed network of processors and routing switches. The physical configuration thus does not need to be changed between different applications.

A network of n distinguishable processors, each of which possesses m bidirectional communications paths, can be configured in $(nm)!/(2(nm-2)!)$ different ways. For example,

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

16 transputers, each with 4 links, can be configured in 2016 different ways. Each transputer may have a different external memory size or even be a different type of transputer, and hence it is appropriate to regard them as distinguishable.

Many of these configurations are apparently redundant, in that they involve two or more connections between a pair of transputers. However, this provides twice the communications bandwidth which may be necessary for some problems. Similarly, the cases in which a transputer is connected to itself can be thought of as configurations in which not all of the links are used.

It is of interest to compare the flexibility offered by this array of 16 transputers with the rigid structure of the hypercube. A fourth-order hypercube would have the same number of processors and communications links but represents only some of the possible configurations.

The final observation is that the combination of OCCAM and the transputer has made it possible to construct efficient networks of high complexity.

REFERENCES

Armstrong, R. H., Elliott, C. J., Mercer, I. C. & Wyss, G. 1987 The final report on a programme to evaluate OCCAM and the transputer for use in orbit. Smith Associates Tech. Rep. TR-87/161.

Bell, D., Morrey, I. & Pugh, J. 1987 *Software engineering, a programming approach*. London: Prentice Hall.

Hoare, C. A. R. 1986 The mathematics of programming. *Byte* **11**, 115–128.

Thomlinson, J., Adams, L. & Harboe-Sørensen, R. 1987 SEU and total dose response of the Inmos transputer. *IEEE Trans.* **NS-34**, 1803–1807.

Discussion

J. H. DAVENPORT (*School of Mathematical Sciences, University of Bath*). Dr Elliott has said, correctly in my opinion, that he wishes to mould the computer to the problem, not the problem to the computer. Yet there seems to be one aspect in which he is moulding the problems to the computer: he insists on a fixed decomposition of the problem into a fixed number of processes, themselves mapped in a fixed way onto the processors. Does he think that this restriction is necessary?

C. J. ELLIOTT. The number of processes into which the problem is decomposed may be varied at compile time, as in the example in my paper of the image encoding stage of automatic fingerprint recognition (AFR).

The mapping of processes on to processors does remain fixed. However, the distribution of datasets to processors remains flexible. The example in my paper of the different options for AFR database searching illustrates this clearly.