

Transputer

Occam and the Transputer

The following paper was presented in March 1984 at the IFIP working group 10.3 workshop on the Hardware Implementation of Concurrent Languages and Distributed Systems, and will be published by North-Holland in the workshop proceedings.

David May

27 June 1984

OCCAM AND THE TRANSPUTER

David May and Roger Shepherd

INMOS Limited
Whitefriars
Lewins Mead
Bristol, BS1 2NP
England

The transputer is a programmable VLSI device with communication links for point-to-point connection to other transputers. Occam (*) is a language which enables a transputer system to be described as a collection of processes which operate concurrently and communicate via named channels. This paper describes how the transputer provides an efficient implementation of concurrency and message passing in a distributed system.

1 Introduction

The transputer is a programmable VLSI device. It provides a 'building block' for high-performance concurrent systems in the same way as the logic gate provides a 'building block' for today's electronic systems. Occam is a language which simplifies the design of transputer systems just as boolean algebra simplifies the design of systems built from logic gates.

An important design objective of occam and the transputer was to provide the same concurrent programming techniques both for a single transputer and for a network of transputers. Consequently, the features of occam were carefully chosen to ensure an efficient distributed implementation on transputer systems and then the concurrent processing mechanisms within the transputer were designed to match.

The result is that a program ultimately intended for a network of transputers can be compiled and executed efficiently by a single computer used for program development. Once the logical behaviour of the program has been verified, the program may be configured for execution by a single transputer (low cost), or for execution by a network of transputer (high performance).

This paper describes the main properties of the transputer and discusses the influence of occam on its design. A brief summary of the relevant aspects of occam is included (see [1] for more details, and [3] for an introduction to occam).

(*) Occam is a trademark of the INMOS Group of Companies

automatically?
or really?

2 Transputer systems

The transputer contains memory, a processor and a number of communication links which allow direct connection to other transputers [2]. A system is constructed from a collection of transputers which operate concurrently and communicate through links. Such a system can be programmed in occam, a language which enables a system to be described as a collection of processes operating concurrently and communicating through named channels [3].

2.1 Point-to-point communications

One of the most significant aspects of the transputer view of system design is the use of point-to-point communication links to connect processing elements, each of which has its own local memory. This choice has many advantages; in particular, it enables arbitrarily large systems to be constructed using local processing and local communication. Furthermore, point-to-point communication links are easy to use and can be efficiently implemented.

However, the choice of local processing and communication necessitates a significant change in programming concepts and new algorithms need to be developed [4]. Most existing languages assume the existence of a global communication system (implicit in the presence of a uniformly accessible address space). Such communication systems (such as buses) suffer from the disadvantage that their speed reduces as their size increases. A consequence is that existing languages are not suitable for distributed concurrent processing, and this alone gives rise to the need for a new language.

2.2 Simulated and Real concurrency

Existing concurrent languages have been designed to provide simulated concurrency. This is not surprising, since until recently it has not been economically feasible to build systems with a lot of real concurrency.

Unfortunately, almost any system can be simulated by a sequential computer. There is, therefore, no guarantee that a language designed for simulated concurrency will be relevant to the needs of systems with real concurrency. The choice of features in such languages has been motivated largely by the need to share one computer between many independent tasks. In contrast, the choice of features in occam has been motivated by the need to use many communicating computers to perform one single task.

3 Occam

Occam enables a system to be described as a collection of concurrent processes, which communicate with each other and with peripheral devices through channels. Occam programs are built from three primitive processes:

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

The primitive processes are combined to form constructs:

SEquential	components executed one after another
PARallel	components executed together
ALternative	component first ready is executed

A construct is itself a process, and may be used as a component of another construct.

Conventional sequential programs can be expressed with variables and assignments, combined in sequential constructs. IF and WHILE constructs are also provided.

Concurrent programs can be expressed with channels, inputs and outputs, which are combined in parallel and alternative constructs.

Each occam channel provides a communication path between two concurrent processes. Communication is synchronized and takes place when both the inputting process and the outputting process are ready. The data to be output is then copied from the outputting process to the inputting process, and both processes continue.

An alternative process may be ready for input from any one of a number of channels. In this case, the input is taken from the channel which is first used for output by another process.

The benefits of point-to-point communication have already been mentioned above. The choice of synchronized communication simplifies programming as it prevents the loss of data. The choice of unbuffered communication removes the need for any store to be associated with the channel. Copying data from the outputting process to the inputting process is clearly essential for communication between transputers, and has significant additional performance advantages. It is easy to make copying within a machine fast by use of microcode.

4 The transputer

A transputer system consists of a number of interconnected transputers, each executing an occam process and communicating with other transputers. As a process executing on a transputer may itself consist of a number of concurrent processes the transputer has to support the occam programming model internally. Within a transputer concurrent processing is implemented by sharing processor time between the concurrent processes.

4.1 Registers

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; six registers are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data paths and control logic.

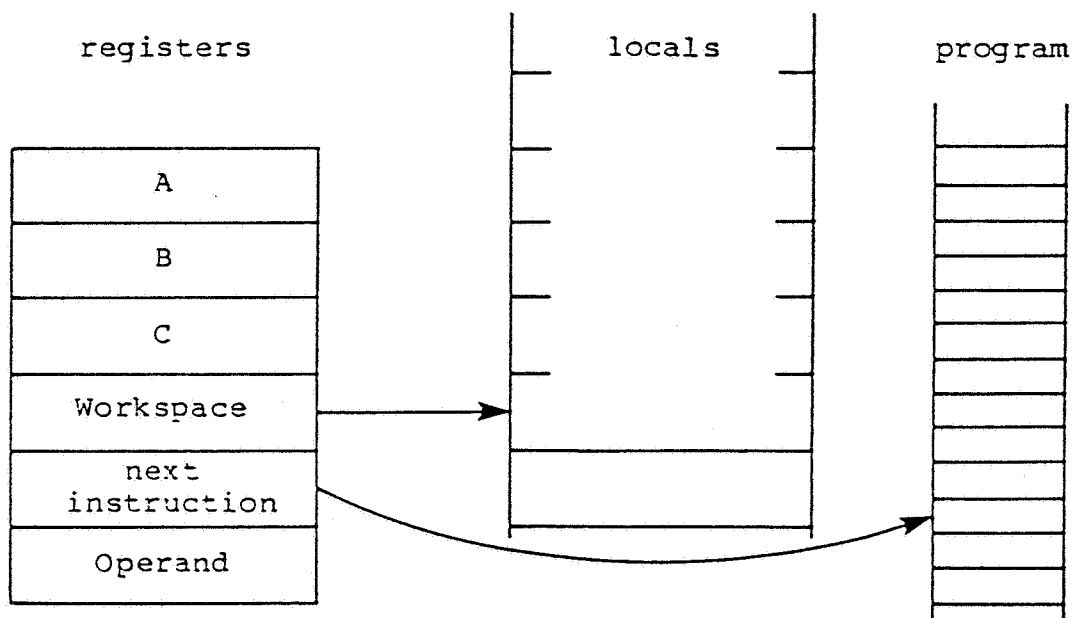
The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

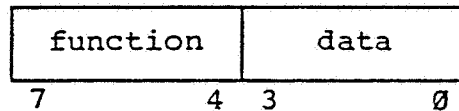
The A, B and C registers which form an evaluation stack, and are the sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A, pops B into A and C into B.



Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the 'add' instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide the optimum balance between code compactness and implementation complexity.

4.2 Instructions

It was a design decision that the transputer should be programmed in a high level language. The instruction set has, therefore, been designed for simple and efficient compilation, and to support the occam process model. It contains a relatively small number of instructions, all with the same format, chosen to give a compact representation of the operations most frequently occurring in programs. Each instruction is one byte long, and is divided into two 4 bit parts. The four most significant bits of the byte are a function code, and the four least significant bits are a data value.



4.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Thirteen of these are used to encode the most important functions performed by any computer. These include:

load constant
add constant

load local
store local
load local pointer

load non local
store non local

jump
conditional jump

call

The most common operations in a program are the loading of small literal values, and the loading and storing of one of a small number of variables. The 'load constant' instruction enables values between 0 and 15 to be loaded with a single byte instruction. The 'load local' and 'store local' instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

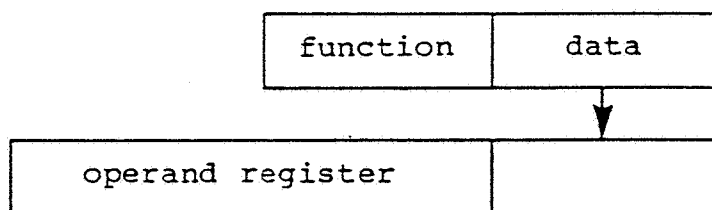
The 'load non local' and 'store non local' instructions behave similarly, except that they access locations in memory relative to the A register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of block structured programming languages.

4.2.2 Prefixing functions

Two more of the function codes are used to allow the operand of any instruction to be extended in length. These are:

prefix
negative prefix

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefixing instructions end by clearing the operand register, ready for the next instruction.



The 'prefix' instruction loads its four data bits into the operand register, and then shifts the operand register up four places. The 'negative prefix' instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefixing instructions. In particular, operands in the range -256 to 255 can be represented using one prefixing instruction.

The use of prefixing instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation, by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form which is independent of the processor wordlength.

4.2.3 Indirect functions

The remaining function code, 'operate', causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefixing instructions can be used to extend the operand of an 'operate' instruction just like any other. This allows the number of operations in the machine to be extended indefinitely.

The encoding of the indirect functions is chosen so that the most frequently occurring operations are represented without the use of a prefixing instruction. These include arithmetic and comparison operations such as

add
greater than

Less frequently occurring operations have encodings which require a single prefixing operation (the transputer instruction set is not large enough to require more than 512 operations to be encoded!).

4.2.4 Word length independence

A program which manipulates bytes, words and truth values can be translated into an instruction sequence which behaves identically whatever the wordlength of the processor executing it (apart from overflow conditions resulting from word length dependencies). This results from the fact that the instruction size is independent of wordlength, the method of representing long operands as a sequence of prefixing instructions, and the memory addressing structure.

A byte in memory is identified by a single word value called a pointer. A pointer consists of two parts: a word address and a byte selector. The byte selector contains as many bits as are needed to identify a single byte within a word and occupies the least significant bits of the pointer. For example, in a 24 bit transputer the word address would occupy the 22 most significant bits and the byte selector the 2 least significant bits.

Special instructions, such as 'load local pointer' and 'word subscript', are provided to construct and manipulate pointers. Pointer values are treated as signed integers. This enables the standard comparison functions to be used on pointer values in the same way that they are used on numerical values.

4.2.5 Efficiency of encoding

Measurements show that about 80% of executed instructions are encoded in a single byte (ie without the use of prefixing instructions). Many of these instructions, such as 'load constant' and 'add' require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions.

Short instructions also improve the effectiveness of instruction prefetch, which in turn improves processor performance. As memory is word accessed, a 32 bit transputer will receive four instructions for every fetch. There are two words of prefetch buffer so that the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be refilled.

4.2.6 Example

The following example illustrates the use of the instruction set for sequential programming. The variables v, w, and x are assumed to be in the first sixteen locations of local workspace. The compiler has allocated a local workspace location, here called StaticLink, to hold the address of the outer level workspace containing the variable y.

occam

```
x := (v + w) - (y + #24)
```

instruction sequence

```
load local v
load local w
operate add
load local StaticLink
load non local y
prefix 2
add constant 4
operate subtract
store local x
```

This requires a total of 9 bytes.

4.3 Support for concurrency

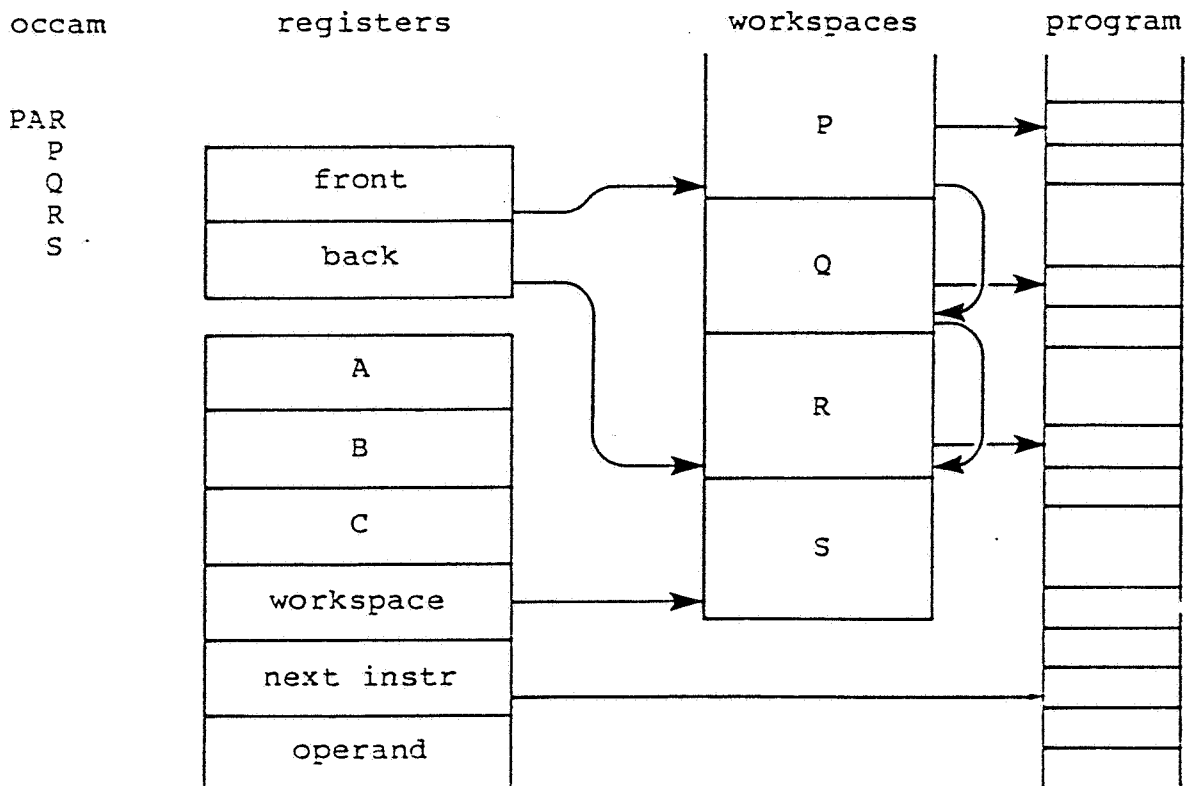
The processor provides efficient support for the occam model of concurrency and communication. It has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel. The processor does not need to support the dynamic allocation of storage as the occam compiler is able to perform the allocation of space to concurrent processes.

At any time, a concurrent process may be

- active
 - being executed
 - on a list awaiting execution
- inactive
 - ready to input
 - ready to output
 - waiting until a specified time

The active processes waiting to be executed are held on a list. This is a linked list of process workspaces, implemented using two registers, one of which points to the first process on the list, the other to the last.

In this illustration, S is executing, and P, Q, and R are active, awaiting execution.



A process is executed until it is unable to proceed because it is waiting to input or output, or waiting for the timer. Whenever a process is unable to proceed, its instruction pointer is saved in its workspace and the next process is taken from the list. Actual process switch times are very small as little state needs to be saved; it is not necessary to save the evaluation stack on rescheduling. why?

The processor provides a number of special operations to support the process model. These include

```
start process
end process
```

When a parallel construct is executed, 'start process' instructions are used to create the necessary concurrent processes. A 'start process' instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed.

The correct termination of a parallel construct is assured by use of the 'end process' instruction. This uses a workspace location as a counter of the components of the parallel construct which have still to terminate. When the components have all terminated, the counter reaches zero, and a specified process can then proceed.

4.4 Communications

Communication between processes is achieved by means of channels. A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being

input message
output message

The 'input message' and 'output message' instructions use the address of the channel to determine whether the channel is internal or external. This means that the same instruction sequence can be used for both internal and external channels, allowing a process to be written and compiled without knowledge of where its channels are connected.

As in the occam model, communication takes place when both the inputting and outputting processes are ready. Consequently, the process which first becomes ready must wait until the second one is also ready.

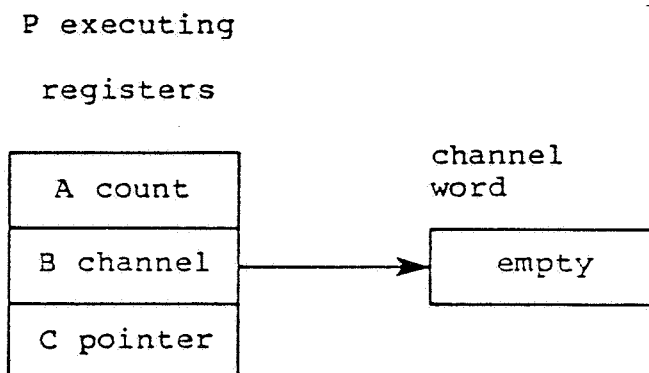
A process performs an input or an output by loading the evaluation stack with a pointer to a message, the identity of the channel, and the count of the number of bytes to be transferred, and then executing an 'input message' or an 'output message' instruction as appropriate.

4.4.1 Message passing on an internal channel

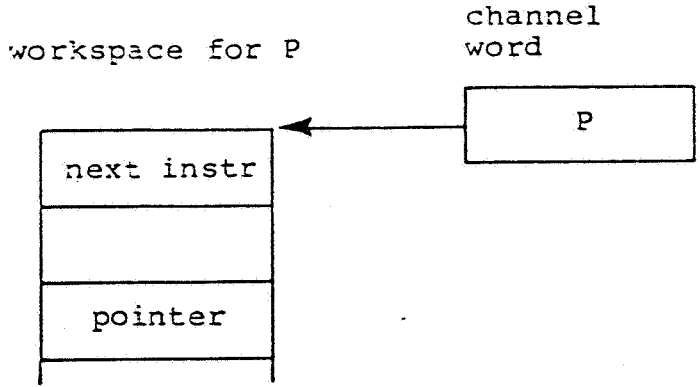
A program initializes an internal channel when it is declared. It does this by writing the value 'empty' to the channel. The value 'empty' is chosen so as not to correspond with the identity of any process.

When a process communicates using an internal channel the message passing instruction examines the contents of the channel.

In this example, the process P is about to execute an input or output message instruction on an initialized channel.

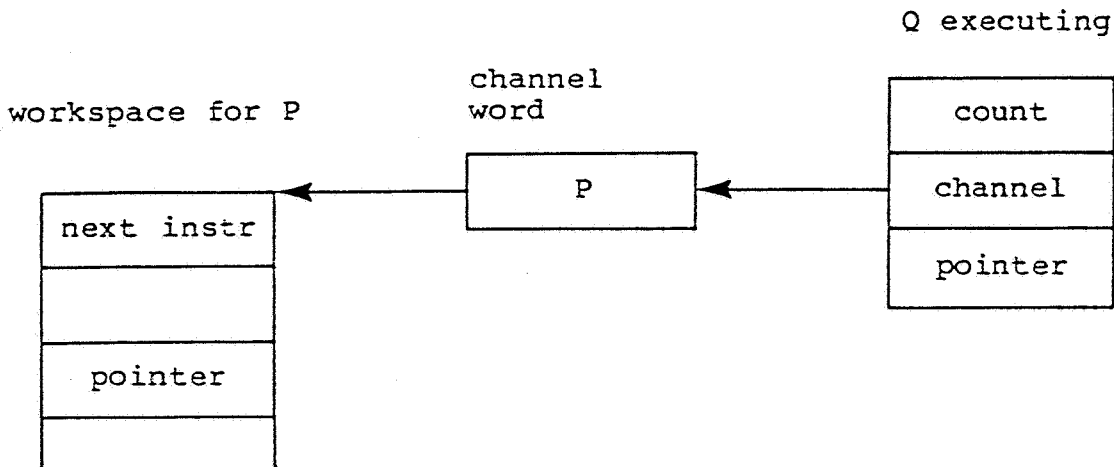


If the channel contains 'empty', then this is the first process to become ready to communicate via this channel. The identity of the process is stored in the channel, and the message pointer stored (with the instruction pointer etc) in the workspace. The processor then starts to execute the next process from the scheduling list.



how is this recognized?

If the channel contains the identity of a process, then this is the second process to become ready to communicate via this channel.



The message is copied, the first process is added to the active process list, and the channel is reset to the 'empty' state. It does not matter whether the inputting or the outputting process becomes ready first.

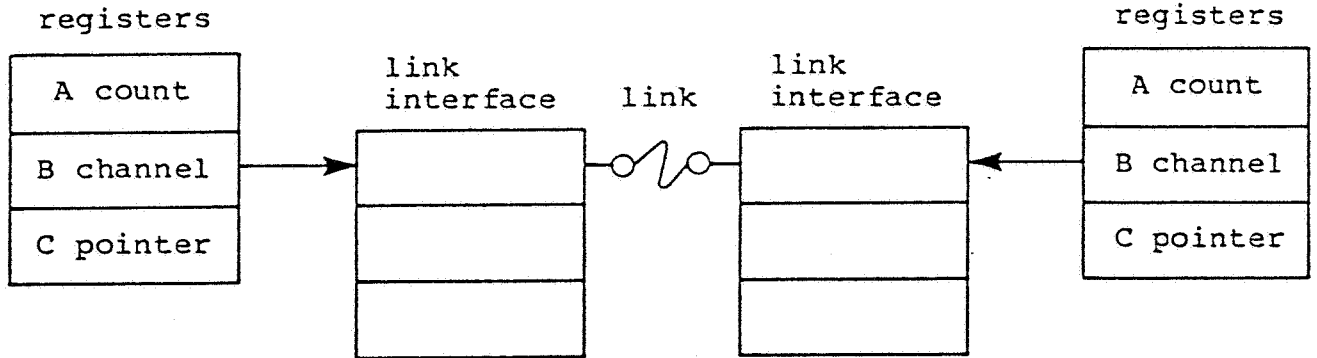
4.4.2 Message passing with point to point links

When a message is passed via an external channel the processor delegates to an autonomous link interface the job of transferring the message and deschedules the process. When the message has been transferred the link interface causes the processor to reschedule the waiting process. This allows the processor to continue the execution of other processes whilst the external message transfer is taking place.

The following diagrams show the sequence of operations when two processes, executing on separate transputers, communicate using a link connecting the two transputers.

P executing

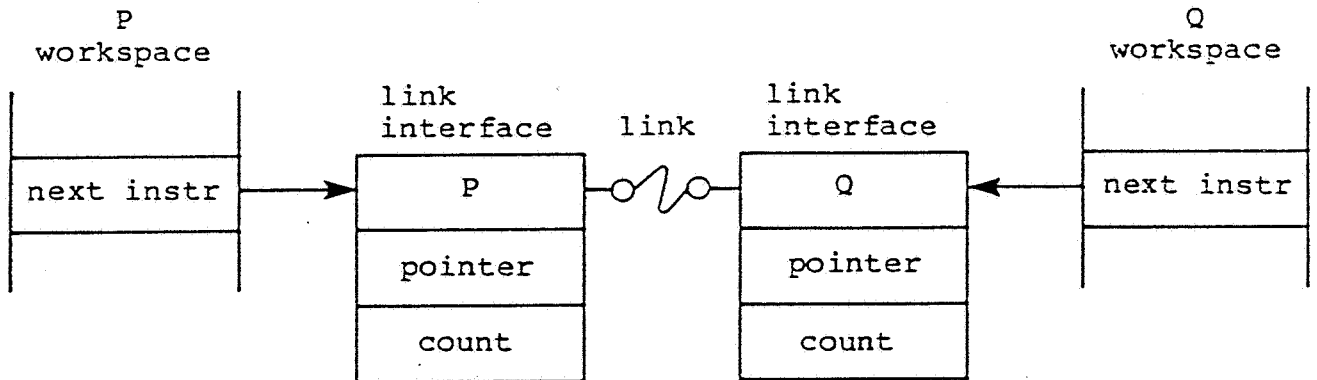
Q executing



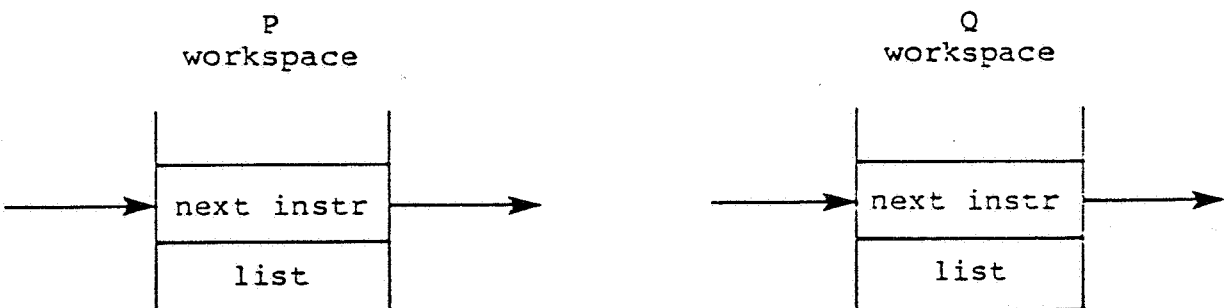
Each link interface uses three registers to hold the following information

- a pointer to the workspace of the process
- a pointer to the message
- a count of bytes to be transferred

When the 'input message' or 'output message' instruction is executed, these registers are initialized, and the instruction pointer is stored in the process workspace. The processor then executes the next process on the scheduling list.

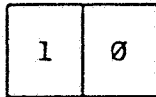


When both link interfaces have been initialized, the message is copied. Each link interface then adds the respective process to the end of the corresponding processor's active list.





Data byte



Acknowledge

Data bytes and acknowledges are multiplexed down each signal line. An acknowledge is transmitted as soon as reception of a data byte starts (if there is room to buffer another one). Consequently transmission may be continuous, with no delays between data bytes.

5 Conclusion

By taking an integrated approach to the design of a VLSI computer and a concurrent programming language it is possible to produce a new level of system 'building block' and the corresponding design formalism.

In particular, it is possible to support the use of the same concurrent programming techniques both within a single transputer and for a network of transputers. The concurrent processing features of a general purpose programming language can be efficiently implemented by a small, simple and fast processor.

6 References

- [1] INMOS Limited, Occam Programming Manual (Prentice-Hall International, London, 1984).
- [2] Barron, I.M. et al., The Transputer, Electronics, 17 November 1983, p 109.
- [3] May, M.D., OCCAM, ACM SIGPLAN Notices vol 18-4 (Apr 1983) pp69-79.
- [4] May, M.D. and Taylor, R.J.B., OCCAM, Microprocessors and Microsystems vol 8-2 (Mar/Apr 1984)

Tiny: An Efficient Routing Harness for the INMOS Transputer

Lyndon Clarke (exploration and routing)
Greg Wilson (monitoring and documentation)

February 8, 1990

Abstract

Message-based MIMD computers that support arbitrary connections over a small number of communications links cannot support parallel applications with complex communication patterns unless the support software includes message routing. This paper describes design issues for such software, a router for the Inmos T800 transputer called Tiny as an example. Performance figures show that Tiny achieves near optimal routing performance given the hardware limits. Modifications to Tiny to monitor message traffic are also described.

1 Introduction

In order to realize the potential of an MIMD computer for complex parallel applications, it must be possible to send messages efficiently and reliably between arbitrary processors. One popular model for such applications is Communicating Sequential Processes (CSP), developed by Hoare [HOAR] and embodied in the occam language [OCC2]. Named channels connect processes pairwise (Figure 1). This approach forces synchronization at the time of communication: message data moves through the channel from process *A* to process *B* only when *A* is ready to send and *B* ready to receive. When not communicating, processes proceed independently.

Our experience shows that CSP is a difficult paradigm within which to develop large applications. Programmers usually want messages sent *to* a destination, but occam forces them to specify sending messages *through* a channel. If a message's destination is not on the same processor as its sender (which it usually is not), then message-passing processes must be created, along with multiplexers and demultiplexers, to accommodate low connectivity between processors. It is tedious to plumb such processes together properly, and difficult to trace the plumbing in someone else's program. In practice, programming in the CSP paradigm is like programming in assembly

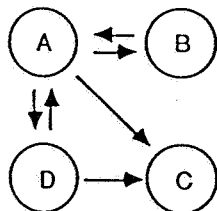


Figure 1: The CSP Model of Concurrency

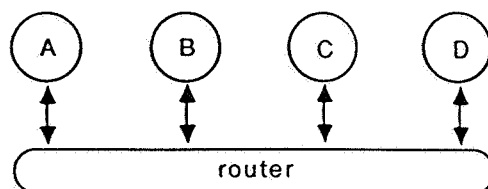


Figure 2: A More Useful Model of Concurrency

language; instead of tracing GOTOs, programmers spend a great deal of time asking, "Where does this message go to?"

The occam implementation of CSP on the transputer is also inefficient. The paradigm's insistence on the independence of processes means that when a message travels between two processes on a single processor its data must be copied from one part of memory to another. Programs which contain multiplexers and FIFO buffers to decouple applications processes may have to copy a message several times before it arrives at its destination, which inevitably degrades performance.

A more comfortable paradigm is one which allows (possibly asynchronous) communication between any two named processes. In this paradigm, *A* sends a message to *B* by giving the message to a router; *A* is then free to carry on calculating. When process *B* wants to receive a message it blocks until the router has one for it.

This paradigm is the one actually used in applications containing any but the simplest parallelism, even if only the first paradigm is directly available. Programmers build FIFOs and multiplexers to provide the illusion of global, asynchronous connectivity so frequently that it is worth doing it once, properly, to provide a package to applications programmers.

This was the motivation behind Tiny [CLA1] and TITCH (Tiny's immediate predecessor, now retired). To users, Tiny is a kernel running on each processor, connected to one or more clients by channels (Figure 2). Each process in the network has a unique user-assigned client identifier (CID). To send a message to process *P* the sender specifies *P*'s CID, the buffer containing the data, the amount of data, and the channel through which that buffer is passed to the router (see Section 4.1). To receive a message, *P* specifies which channel it is listening to, and where the incoming data should be placed. Everything else is Tiny's responsibility.

An essential feature of a general-purpose router is that it should be able to run on any topology, so that users can connect processors to suit their applications. Much of Tiny is therefore devoted to exploring the network and building up routing tables recording the shortest paths from each processor to each other. This allows applications to use randomly-connected graphs, in which mean and maximum inter-node distance and worst link loading increase only as $\log(n)$ [PRNC]. How networks are explored and routing tables built up is described in [CLA3].

Another important feature of Tiny is its efficiency. Most users of massively parallel computers are primarily interested in performance. If routers must reside on the chip, users therefore want them not only to deliver messages quickly, but also to steal as few CPU cycles from the application as possible. As Section 6 discusses, the performance of Tiny is very close to hardware limits.

Acknowledgements

Tiny was developed by Lyndon Clarke during Ph.D. research in the Department of Physics at the University of Edinburgh supported by the Science and Engineering Research Council. It has since become an important part of the Club Domain software of the Edinburgh Concurrent Supercomputer Project (ECSP). The amendments made to Tiny to monitor its internals were

made by Greg Wilson while he was supported by Meiko Limited, and as part of his Ph.D. research in the Department of Computer Science at the University of Edinburgh. We would like to thank Roy Campbell for his many helpful comments.

2 The INMOS T800 Transputer

The INMOS transputer [T800] was designed for use in both real-time control applications and multiprocessor computers. Each transputer contains a CPU, four kilobytes of on-chip memory, and four high-speed communications links, each with its own DMA controller. The T800 transputer also contains a 1 MFLOP floating-point unit.

The transputer was designed to run concurrent processes efficiently. It automatically maintains one queue each for low-priority and high-priority processes. Low-priority processes are timesliced automatically; when one is interrupted, it is placed at the tail of the low-priority queue and the next one scheduled. High-priority processes are not timesliced, and cannot be interrupted, even by other high-priority processes. A high-priority process runs until it deschedules itself or is blocked by an i/o operation.¹

The transputer's CPU contains only three general-purpose registers, called *Areg*, *Breg*, and *Creg*, arranged as a stack. All instructions implicitly manipulate particular registers — for example, the instruction *stnl* (store non-local) stores the contents of *Breg* in the location pointed to by *Areg*, popping both values.

The transputer also contains a workspace pointer register called *Wptr*. This holds the base address of the active process's workspace. All local variable accesses are offset relative to *Wptr*; an instruction to load the value at address 5, for example, actually loads the value at (*Wptr*+5). When a low-priority process is descheduled the values of *Wptr* and the instruction pointer *Iptr* are saved, but the contents of the register stack are not. Descheduling can only take place after certain instructions, so care must be taken to ensure that nothing valuable is on the stack at such times.

The instructions *in* and *out* are of particular interest in the design of routing software. These are used to implement both internal and external communications. *in* interprets *Creg* as the address of a data block, *Breg* as the address of a communications channel, and *Areg* as a byte count. When *in* is executed to carry out internal communications, the transputer checks the value pointed to by *Breg*. If this contains *mint* (the least integer the transputer can represent), then the other process taking part in the communication has not yet rendezvoused. The process doing the *in* puts its workspace pointer in the channel word and stores the base address of the data to be transferred in a reserved location in its workspace. When the other process involved executes an *out*, it finds a valid address in the channel word, so it copies its data into the first process's workspace and reschedules that process. (If the sending process reaches the communication first, the reciprocal steps are carried out.)

External communication is done in much the same way. In this case, however, the values in the register stack are given to the controller responsible for the link over which the communication is taking place. When both of the link controllers involved have been given their orders, they arrange for data to be transferred without further CPU intervention, re-scheduling the communicating processes when the transfer completes. While external communication is slower than internal communication (since link bandwidth is lower than memory bandwidth), it retards other CPU activities only slightly by stealing memory cycles for DMA.

One final important feature of the transputer is the way its hardware implements choice in communications. The occam ALT construct takes pairs of channels and Boolean guards and accepts

¹Two other queues are maintained, one each for processes of each priority level waiting on clock interrupts. This "wait until" feature is exploited by the monitor described in Section 7.

a single communication on a channel whose guard is TRUE, then executes the code nested below that communication. For example:

```
ALT
  time > startTime & connection[0] ? newTime
    process(newTime)
  time < stopTime & connection[1] ? newTime
    process(newTime)
  TRUE & interrupt ? any
    stop := TRUE
```

will accept an integer newTime on element 0 of the channel array connection if time is greater than startTime, or on element 1 of connection if time is less than stopTime, or any value on interrupt.

To implement ALT, the transputer evaluates each Boolean guard and initializes each channel. When one of the branches is triggered, the transputer goes through these operations again to disable each channel, so that if several processes tried to rendezvous with the ALTing process, only one will succeed.

It is important to note that the servicing of ALT branches is arbitrary, not random. If several branches are ready, the transputer will choose one of them; if, when the same piece of code is next executed, the same branches are ready again, the transputer will choose the same branch. This has an important effect on the efficiency and correctness of CSP-style routers, as is discussed in the next section.

3 Router Design

The most important constraint on the design of a processor-resident router like the one in Figure 2 is the low number of inter-processor connections. Since these connections are the "narrowest" points in the system, i.e. the ones with the least bandwidth, each link controller should be given its own pair of processes (one each for input and output) so that external communications can be externally driven. Each client process should similarly be connected to the router through its own input and output handlers, to decouple the interface procedures from the underlying router. These various output handlers should be sufficiently buffered with FIFO queues to avoid the head-of-line blocking problem [HLUC], in which throughput is limited because messages which could be routed to idle output points are hidden behind messages which cannot currently be routed.

This leads to the process structure shown in Figure 3. Each routing process is either a multiplexer or a demultiplexer, and is referred to as an *agent*. In Tiny, the routing logic which decides where to send messages lies in the demultiplexing agents.

In order to maximize the performance of a computer in which message routing and computation compete for CPU cycles, each processor must act as if through-routing messages were more important than doing the user's calculations. (To understand this, consider what would happen if processor P_1 gave higher priority to its own work than it did to routing messages to other processors. In this case, $N - 1$ of the computer's N processors could be waiting for new tasks while only one of its processors was doing useful work.) This means that the router's processes must run at high priority, while the application processes run at low priority. However, since high priority processes on the transputer cannot be interrupted, the router must contain mechanisms to deschedule its agents explicitly. The transputer provides such mechanisms naturally by descheduling even high-priority processes when they attempt to communicate with an unready partner. Another more explicit method used by Tiny is described in Section 5.3.

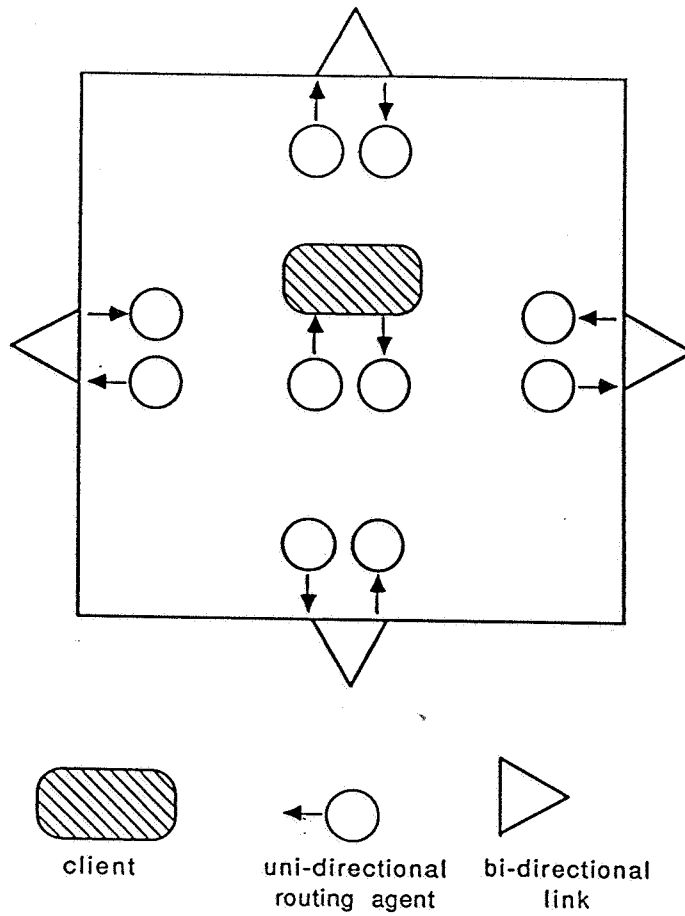


Figure 3: Structure of Processes in a Router

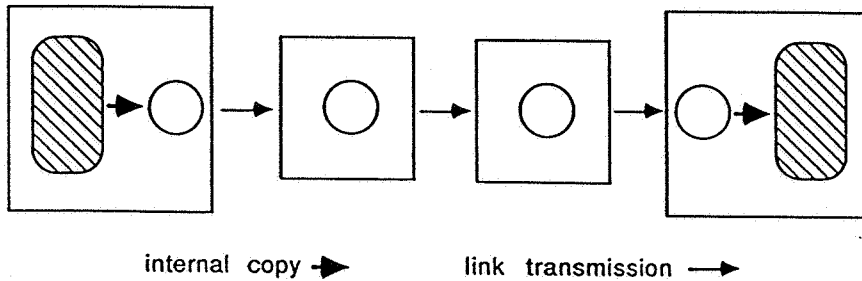


Figure 4: Time Taken to Copy a Message

3.1 Improving on CSP Routers

Tiny differs from pure CSP routers by avoiding the overheads of data copying that occur when the in and out instructions are used between agents on a single processor. The entire message is copied each time these instructions are used. If messages are large or frequent, this copying could consume the majority of the processor's time. Even ignoring the copying they initiate, in out are very expensive — the instructions Tiny uses instead (described in Section 5.3) consume a similar number of cycles, but perform much more useful work.

Instead of copying data, Tiny's agents pass pointers to buffers.² The total time spent copying each message is therefore only $M(2t_C + Lt_L)$, where M is the message size in bytes, L is the number of links the message passes through, t_C is the time to copy a single byte to or from a process's workspace, and t_L is the time to send a byte down a link (Figure 4). This overhead could be further reduced by eliminating t_C , and allowing user processes to manipulate buffer pointers directly. For the reasons given in Section 8, such a scheme is practical.

Another difference between pure CSP routers and Tiny is the elimination of ALTs to ensure both fairness and greater speed. Consider the occam procedure given by:

```
PROC mux(□CHAN OF ANY input, CHAN OF ANY output, □INT buffer)

  INT buffer.size :
  WHILE TRUE
    ALT i = 0 FOR (SIZE input)
      input[i] ? buffer.size; buffer
      output ! buffer.size; buffer
  :
```

This multiplexer repeatedly accepts input on any of its array of input channels, then outputs the message received. The problem is that in a transputer this multiplexer will always choose the same input channel if all channels are always ready. A better multiplexer is:

```
PROC fair.mux(□CHAN OF ANY input, CHAN OF ANY output, □INT buffer)
```

²A way of doing this in occam is outlined in [WILS].

```

INT buffer.size, client :
WHILE TRUE
  ALT i = client FOR (SIZE input)
    VAL chan.id IS (client \ (SIZE input)) :
      input[chan.id] ? buffer.size; buffer
    SEQ
      output ! buffer.size; buffer
      client := chan.id + 1
:

```

(The backslash \backslash is occam's remainder operator, while (SIZE input) gives the dimension of the channel array being multiplexed.) This multiplexer repeatedly re-orders its inputs so that any order-dependent favouritism is eliminated, and looks suspiciously like a queue with multiple inputs. Each client puts buffers into the queue (which in this case only has depth 1), and the multiplexer removes them in order of their arrival. The ALT-less implementation of multiple-input queues described below not only speeds up the multiplexer, it also makes the multiplexer's behaviour independent of its number of inputs.³

4 Internal Structure

Tiny is made up of several interacting agents on each processor. Each handles the input or output half of a link, or a single input or output channel connecting Tiny to a client. Each agent's workspace, or *A-page*, contains space for certain transputer instructions and an *agent handle* used directly by Tiny.

Agents manipulate *buffers*, represented by *B-pages* which contain routing and ownership information and a pointer to the message's data. As with A-pages, the part of a B-page manipulated directly by Tiny is called its handle.

4.1 Message Typing

Just as programmers assign types to variables within processes, in Tiny they may type messages by sending them on different channels. For example, in a simple grid decomposition in which processes repeatedly swap boundary conditions, each process receives two messages containing new boundary values during each iteration. If the process is connected to the router by a single channel, it must read the message, decide which boundary the data is for, and then copy the data into that boundary. If the channels connecting the process to the router have types left-bound and right-bound, the process can know *a priori* where the message data belongs. This makes for more efficient programs (less internal copying is required) and more efficient programming (message typing provides structure, which reduces programming errors).

Multi-channel interfaces are only slightly more complicated to implement than single-channel interfaces. The agents handling traffic from clients to Tiny are the same in either case. An intermediate agent called `agent_multi_man` is used; All messages to a client *C* are sent to its `agent_multi_man`, which forwards the message to the subsidiary agent responsible for the channel through which that message should be delivered.

4.2 Routing Strategies

Two routing strategies, called *adaptive* and *sequential*, are available for point-to-point message traffic in Tiny. A third strategy is used to implement broadcasts. The sequential strategy guar-

³I.E. of the size of the channel array input.

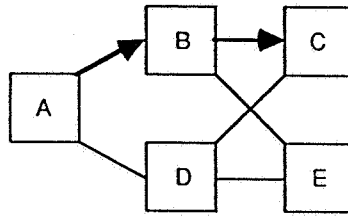


Figure 5: Sequential Routing Strategy

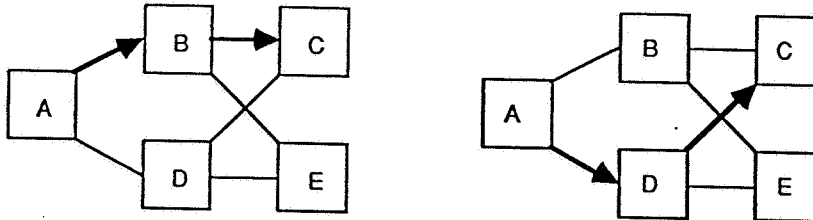


Figure 6: Adaptive Routing Strategy

antees that messages arrive in the order in which they were sent, which is useful for such things as implementing fragmentation of large messages. Tiny implements the sequential strategy by using the same shortest path⁴ from any through-routing node towards the destination. (Figure 5).

The adaptive strategy decides locally at each through-routing node which of the shortest paths from that node to the destination seems likely to be quickest. Using this strategy, Tiny examines the output queues of each of the through-routing node's appropriate links, and enqueues the message for the link with the shortest queue (Figure 6).⁵

Finally, the broadcast strategy is implemented using a broadcast tree for each processor P . The tree for each processor is determined once during initialization, and information is stored in each other processor Q to indicate Q 's position in P 's tree (Figure 7).

4.3 The Interface

Two interfaces for Tiny were initially constructed. The first of these, called the PKT interface, copies data from the user's process into one of Tiny's internal buffers during a send, and copies data back during a receive. The PTR interface, on the other hand, swaps pointers to buffers so that the user's buffer becomes one of Tiny's internal buffers and *vice versa* during either type of interaction. While faster, this second interface was found to be too fragile to support general applications (see Section 8), and has been removed.

The interface provides the user with four functions: `pktRead`, `pktWrite`, `pktSeqWrite`, and

⁴In fact, the one stored first in the routing table.

⁵An alternative way of implementing adaptive routing would be to enqueue the buffer for each possible output link, and have the link which was ready first remove the buffer from the other links' queues. This would require doubly-linked lists, whose maintenance would slow the router down in the general case. In addition, the extra instructions which would have to be executed by the link which wound up sending the message would probably negate most of the time saved by always "guessing" right.

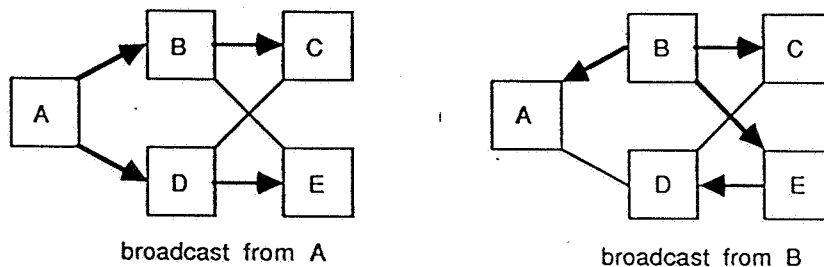


Figure 7: Broadcast Routing

`pktBroadcast`. These functions, described below, are characterized by *blocking reads* and *non-blocking writes*.

4.3.1 Receiving a Message

```
void pktRead(CHAN *in, int *src, *msg, *size)
```

This procedure blocks until a message becomes available on the channel `in`. The source CID of the message is put in `src`; the message's size is put in `size`, and the message data is put in the buffer pointed to by `msg`.

4.3.2 Sending a Message by the Quickest Route

```
void pktWrite(CHAN *out, int dst, *msg, size)
```

`pktWrite` sends a message from the caller to the client identified by `dst` using adaptive routing. As soon as the message data contained in the buffer pointed to by `msg` has been copied into one of Tiny's internal buffers this function returns control to the caller.

4.3.3 Sending a Message by a Fixed Route

```
void pktSeqWrite(CHAN *out, int dst, *msg, size)
```

`pktSeqWrite` sends a message from the caller to the client identified by `dst` using sequential routing. As soon as the message data contained in the buffer pointed to by `msg` has been copied into one of Tiny's internal buffers this function returns control to the caller.

4.3.4 Broadcasting a Mesasge

```
void pktBroadcast(CHAN *out, int *msg, size)
```

`pktBroadcast` sends a message from the caller to every other client in the network connected to Tiny by the same type of interface channel (see Section 4.1). As soon as the message data contained in the buffer pointed to by `msg` has been sent down all the necessary links, this function returns control to the caller. (An unfortunate exception to this rule is discussed in Section 8).

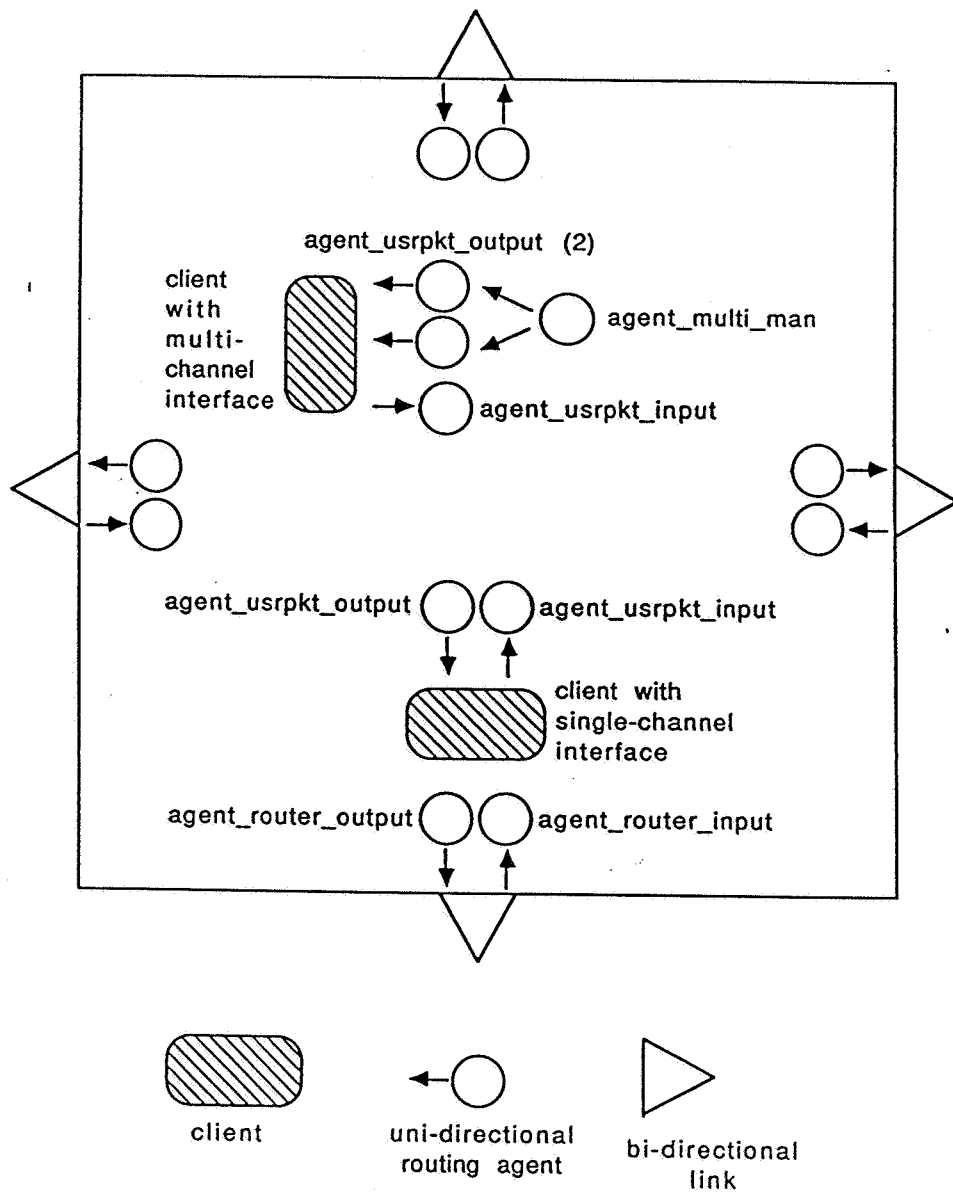


Figure 8: Arrangement of Actual Routing Agents

5 Agents and Their Internals

The five different agents used within Tiny to implement the router shown in Figure 3 are described below; a typical mix of them is shown in Figure 8. All agents are written in a mixture of C and Tcode (transputer assembler).

agent_multi_man: demultiplexes messages arriving at a client with a multi-channel interface, passing those messages to the client's input agents. Its main loop dequeues a buffer, then enqueues it for a subsidiary agent.

agent_router_input: accepts messages on the input half of a hard link and routes them. Its main loop dequeues a buffer, gets a message into the buffer, and routes the buffer, putting it in another agent's queue.

agent_router_output: handles the output queue for a hard link. Its main loop dequeues a buffer, sends its contents down the link, then decrements the extra reference count of the buffer (Section 5.4.1), and, if this is zero, returns the buffer to its owner.

agent_usrpkt_input: accepts messages from a client process and routes them. Its main loop dequeues a buffer, gets a message into the buffer, makes a header for the message, then routes the buffer, putting it in another agent's queue.

agent_usrpkt_output: gives a message to a client process and frees the buffer. This agent may be a subsidiary agent of an `agent_multi_man`, or may be used directly. Its main loop dequeues a buffer, gives the buffer's contents to the user, then decrements the extra reference count of the buffer, and, if this is now zero, returns the buffer to its owner.

5.1 Agent and Buffer Pages

An agent page, or A-page, is a workspace for a single routing agent. The fields in an A-page are given below; those from `AP_RT` to `AP_ENTRY` called the *agent handle*, are pointed to by the workspace pointer `Wptr` when the agent is running.

AP_LINK	A pointer to the next agent, used to chain agents together while they are being created.
AP_FUNC	An integer indicating the agent type.
AP_TSPACE	The start of 5 words reserved for transputer instructions [TREF]. These are not explicitly manipulated by Tiny.
AP_TMPO	A scratch pad used by the agent.
AP_BP	A pointer to the agent's active buffer.
AP_COUNT	A count of the number of buffers in the agent's queue, plus 1 for the buffer currently being worked on. When this is 0, the agent should be asleep.
AP_HEAD	A pointer to the first buffer in the agent's queue. If AP_COUNT is 1 or less, this value is invalid.
AP_TAIL	A pointer to the last buffer in the agent's queue. If AP_COUNT is 1 or less, this value is invalid.
AP_QOFF	The offset (in bytes) from the start of the buffers on this transputer to the word used by this agent to link its queue together (see the description of the buffer pages).
AP_CHAN	The address of the channel for which the agent is responsible.
AP_PID	The ID of the client for which the agent is responsible.
AP_DEV	The device number (i.e. channel number in a multi-channel interface) for which the agent is responsible.
AP_ROUTE	A pointer to the agent's routing table segment (Section 5.2).
AP_ENTRY	A pointer to the routing table entry currently being used.

Buffer pages, or B-pages, contain information about message buffers, and are manipulated by agents. The fields in a B-page are listed below; those from BP_LEN to BP_OWNER are referred to as the *buffer handle*.

BP_LINK	Used to link buffers together during initialization.
BP_BYTES	The maximum size of the message buffer (in bytes).
BP_LEN	The length of the message data (in bytes).
BP_SRC	The CID of the process sending the message.
BP_DST	The CID of the message's destination.
BP_MSG	A pointer to the message data associated with the buffer.
BP_XREFS	The number of extra references to the buffer. This field is used during broadcasts, when several agents might be reading from the buffer. When XREFS reaches 0, the broadcast is finished.
BP_OWNER	A pointer to the workspace of the agent which owns the buffer (see Section 5.4.2).

The remaining entries are links to other buffers, used to chain buffers together to form queues. The number of such links is the same for all buffers on any transputer, though it may differ between transputers. Each group of agents within a given transputer responsible for either a single link or a single client has a unique byte offset (stored in the AP_QOFF field of its agent page) which indicates how far from the beginning of the B-page their queue's link entry is. Multiple links are provided so that a single buffer can be in the queues of several agents during a broadcast operation.

Storage for buffers is allocated by the user during initialization. Tiny is given a block of memory and two parameters describing the number of buffers to create and the size of each

buffer. This allows users to tailor memory usage to fit their application; typically, they allocate four or five buffers for each output agent's queue, each roughly one kilobyte in size.

5.2 Routing Tables

Routing tables are built during topology exploration. Initially stored as arrays, their cross-referencing indices are converted to pointers during initialization. The structure of a routing table in array form is:

free_ptr	link indices	routing segments	tables	free space
----------	--------------	------------------	--------	------------

where:

free_ptr is an index into the *free space* used during initialization;

the *link indices* are 5 indices (one for each physical link and another for a "pseudo-link" representing the user's processes) indicating where in the *routing segments* the routing table for messages entering the processor on that link is located;

the *routing segments* contain a status word and a pair of indices into the *tables* for each process in the entire network;

the *tables* are lists of links to which messages may (or must) be sent;

free space is that part of the table which has not been used.

Users select which of Tiny's two routing algorithms it is to use at initialization. One of these provides provable freedom from deadlock [CLA4] by eliminating cycles in routing tables. This is done by routing messages for a destination through different links on a particular processor depending on the link through which that message reached the processor. To allow this, a pointer to a routing table is associated with each input link. These pointers may indicate the routing table, or different routing tables. A fifth routing table is required for routing messages which originate in the user processes on the transputer. Each link's index therefore indicates a segment of the form:

status	(<i>txmt</i> ₀ , <i>brdcst</i> ₀)	...	(<i>txmt</i> _{<i>n</i>} , <i>brdcst</i> _{<i>n</i>})
--------	---	-----	---

where *status* indicates whether the table is bad (i.e. Tiny hasn't built the table yet), in an array/index state (as described here), or has been converted to pointers. The remainder of the segment contains pairs of indices into the *tables* area. The table indicated by the first index is a negative-terminated list of the links through which messages for a particular process *may* be sent; if there are several equally-short paths to a process, the list will contain more than one entry (Section 5.4). The table indicated by the second index is a negative-terminated list of the links through which broadcast messages from other processors through this processor *must* be sent.

These negative-terminated tables consist of one or more integer entries, followed by the value -1. A typical such table would be:

3	0	-1
---	---	----

meaning that messages to or from the corresponding process could or must be sent through links 3 and 0.

5.3 Enqueueing and Dequeueing Buffers

The basic operations performed by Tiny's internal agents are to enqueue or dequeue a message buffer. Agents can do this for other agents, i.e. an agent A_0 can manipulate agent A_1 's internals to put a buffer B into A_1 's queue. At all times, each agent keeps a count of the number of buffers under its control. The buffer currently being worked on is called the *active buffer*; the agent's other $n - 1$ buffers form its queue. The three different cases are:

Number	Meaning
0	No buffers in queue, no buffer being worked on — agent is asleep
1	No buffers in queue, active buffer being worked on
> 1	Buffers in queue, active buffer being worked on

When an agent A_0 wants to enqueue a buffer for an agent A_1 it starts by resetting the workspace pointer register to point at A_1 's workspace. If A_1 is asleep, A_0 makes the buffer A_1 's active buffer and puts A_1 back on the process queue. If A_1 has an active buffer but no queue, A_0 creates a single-buffer queue for it; if A_1 already has a queue, A_0 appends the buffer to that queue.

Dequeueing buffers is simpler than enqueueing them. The agent's buffer count is decremented, and, if the count has become zero, the agent takes itself off the active process queue. If the count has become 1 the buffer which was at the head of the queue is made the active buffer. Finally, if the count after the decrement is greater than 1, the buffer at the head of the queue is made the active buffer, and the head-of-queue pointer changed to point to the next buffer in the queue.

5.4 Routing

Messages are routed as soon as they arrive in a processor. Routing relies on a 3-word *message header* which Tiny keeps in each buffer page. The three entries in this header are the *message source* and *message destination* CID's, and the *message length* (in bytes).

To access the correct routing table entry for a message, Tiny adds the CID of the message's destination to the base address of the agent's routing segment (stored locally in `AP_ROUTE`) to obtain the address at which a pointer to the routing table entry is stored (Figure 9). This extra level of indirection is needed because the routing tables used for broadcast messages may be of arbitrary size.

Tiny's first action when routing is to discover which strategy is being used by testing the two least significant bits in the destination field of the header. The `RT_FST` bit indicates that the sequential strategy is being used, while `RT_ANY` indicates adaptive routing. If `RT_FST` is set, Tiny adds the destination CID to the routing table base address in the agent handle and looks up the routing table segment starting at that address. The entry found here is the address of the workspace of the agent responsible for that link. The current agent puts the buffer in that agent's queue.

If the adaptive strategy bit is set in the message header, Tiny accesses the routing table segment as above, but then examines the lengths of the queues of the agents responsible for the links down which it could send the message. The message is enqueued for an agent with a shortest queue length (Section 5.4).

When a message is first introduced into the network, Tiny checks to see whether it is a broadcast message. If it is, then the CID of the *source* process is put into message's destination field, and bit 2 set in this field. Since the low two bits in a transputer word act as a byte selector, while the other 30 bits are a word selector, this bit being 1 guarantees an odd word address, while the bit being 0 guarantees an even address one space lower in memory. When the word-selector

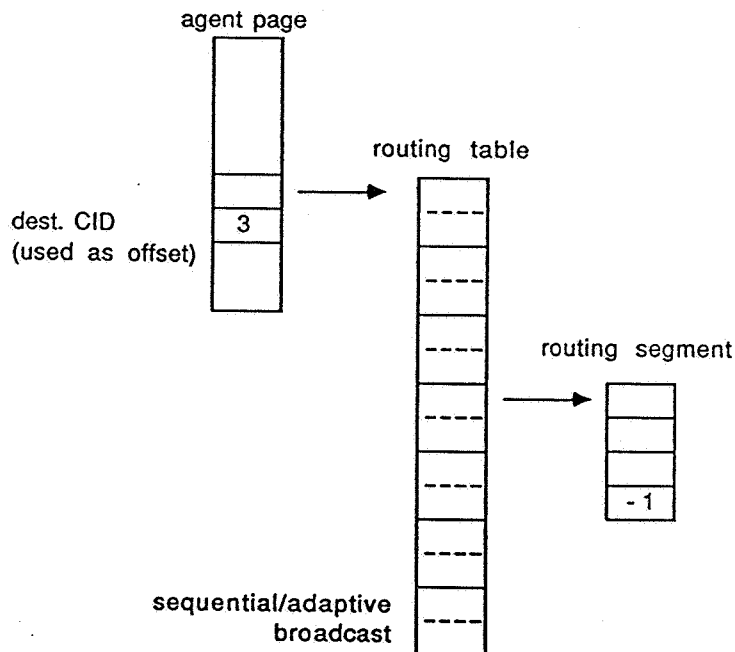


Figure 9: Accessing Routing Tables

bits of the destination field are added to the routing table base address stored in the agent page, Tiny will select the second of the two words in that routing entry. The table this points to contains the broadcast tree information for messages *from* the source process, rather than routing information for messages *to* that process.

5.4.1 Multiple References to Buffers

During broadcasts a single buffer may be queued up for output on several links. To allow for this, the B-page structure contains several fields for pointers to the "next" buffer in the queue. Each of these fields is used by a different group of agents (i.e. all those responsible for a single link or client); the offset from the beginning of the B-page to the pointer belonging to a particular group of agents is stored in the A_QOFF entry of the A-pages of agents in that group. Figure 10 shows how these overlapping queues work.

5.4.2 Buffer Ownership

Every buffer has a nominal owner, which is the agent responsible for it when it is not being used. Once the contents of a buffer have been given to a client process, that buffer must be put back in the queue of its owner. This is done by decrementing the extra reference count XREF of the buffer, and, if this has become zero, putting the buffer in its owner's queue.

One subtlety of buffer ownership is that all buffers must be owned by agents responsible for handling messages as they arrive at processors (i.e. `agent_router_input` and `agent_usrpkt_input`) and not by agents responsible for handling messages leaving processors (i.e. `agent_router_output`, `agent_usrpkt_output`, and `agent_multi_man`). The second type of agent only goes to sleep when

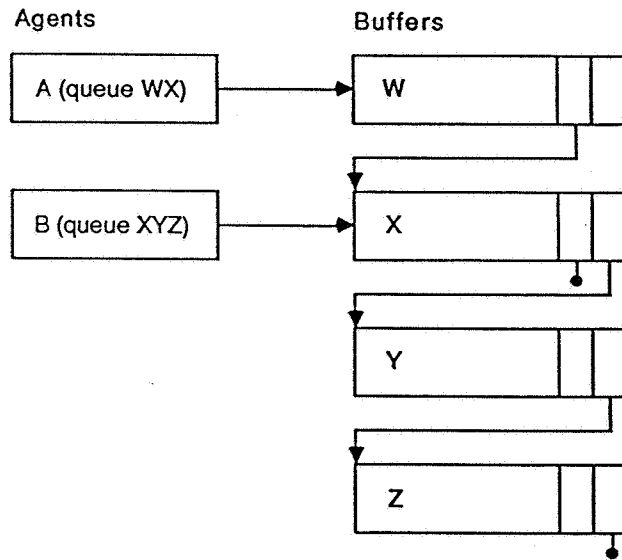


Figure 10: Implementation of Overlapping Buffer Queues

its buffer queue is empty, so as long as other agents give it buffers, it will continue to run. Input agents, on the other hand, run until there are no incoming messages to be serviced, so it is safe for them to store empty buffers in their queues.

6 Performance

Devising experiments whose results are both understandable and relevant to applications programming is a major problem in measuring the performance of routing software such as Tiny. To obtain basic performance figures, messages were echoed on a chain of processors using occam code which read and sent messages on the links directly, occam code which had buffer processes managing the links, and Tiny. The time measured at the source between sending and receiving the message should be very nearly twice the time taken for the master to send a single message to a slave at that distance in a quiet network. Figures 11- 14 show the relationship between message size, distance travelled, and time taken in microseconds. Note particularly that even for medium-sized message (256 bytes) Tiny outperforms simply buffered occam processes even over moderate distances (4 links).

The message time in this situation should vary linearly with both the number of links and the message size. Let $T(l, b)$ be the time taken for a message of b to pass over l links; we anticipate that:

$$T(l, b) = \alpha + \beta l + \gamma b + \delta b l \quad (1)$$

where α is a constant setup time, β is the overhead per link, γ is the time taken to move message data from the user to the router and back, and δ is the time to transfer a single byte through a link. Tests gave:

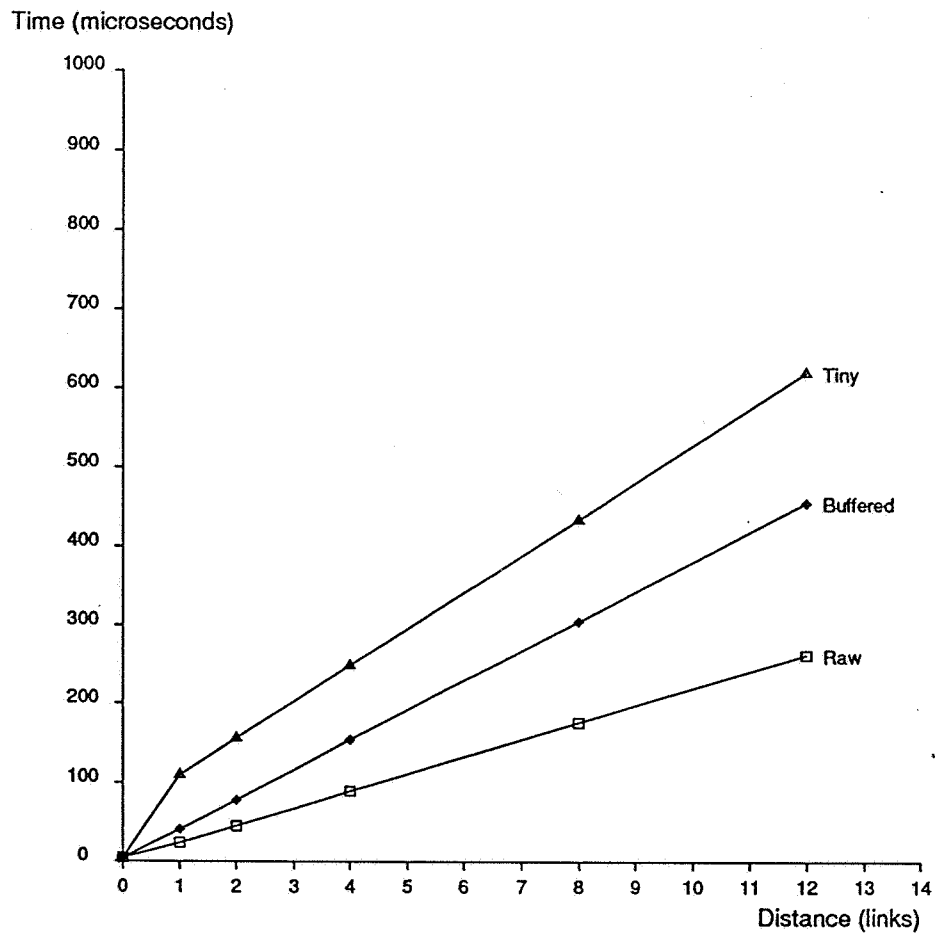


Figure 11: Travel Time vs. Distance for 4-byte Message

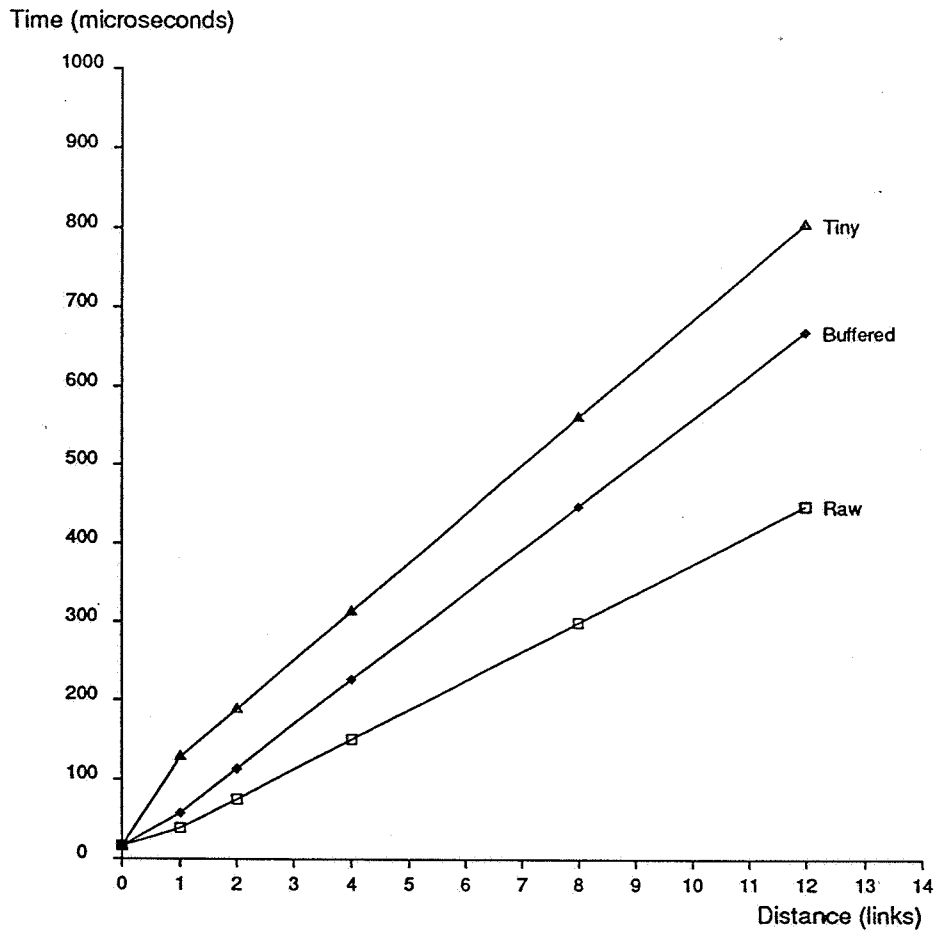


Figure 12: Travel Time vs. Distance for 16-byte Message

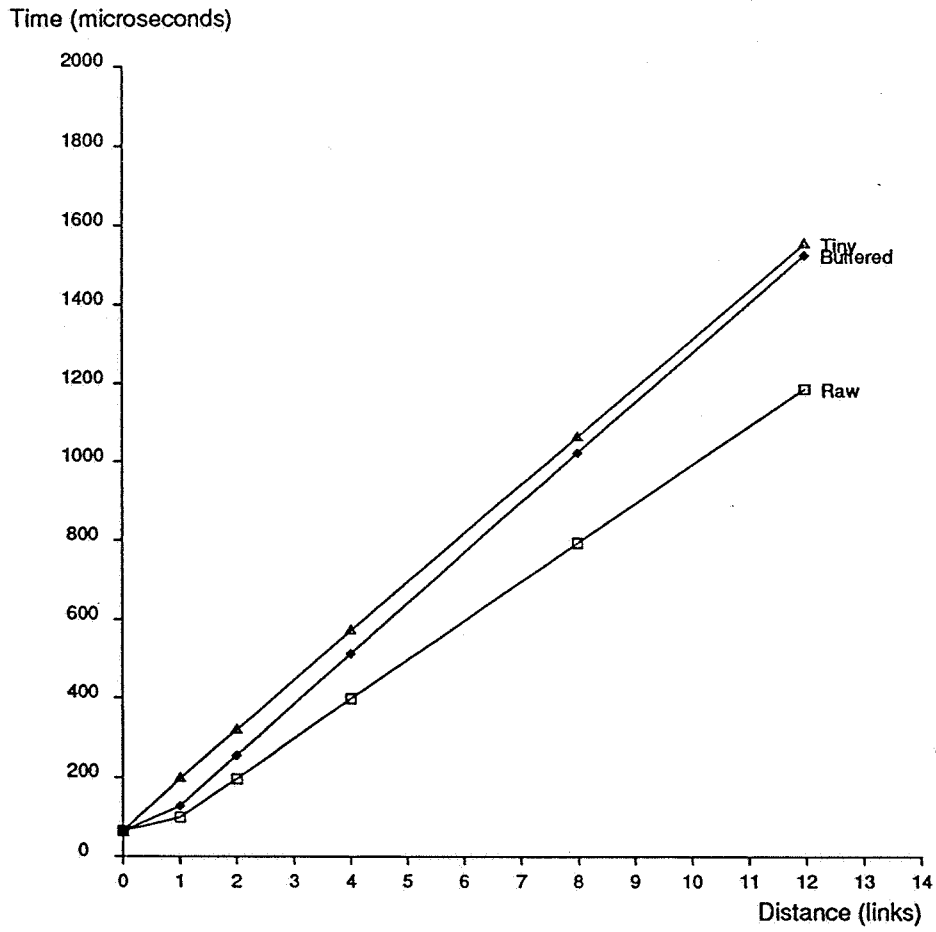


Figure 13: Travel Time vs. Distance for 64-byte Message

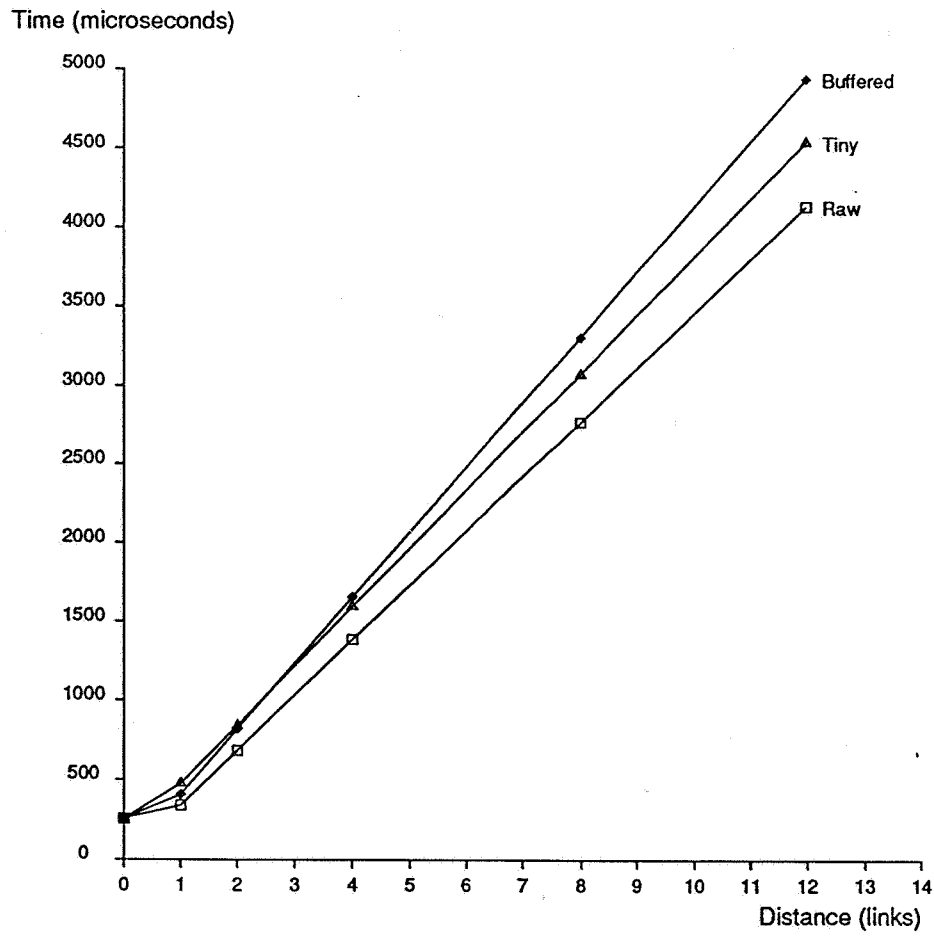


Figure 14: Travel Time vs. Distance for 256-byte Message

Method	α	β	γ	δ
Raw	1.0	16.7	0.0	1.3
Buffer	3.3	31.7	0.0	1.5
Tiny	62.4	41.0	0.1	1.3

These results may be interpreted as follows:

- δ is the same for Tiny and raw communications, implying that Tiny achieves the hardware limit.
- γ , the multiplier on message size, reflects the time taken to copy the message data from the user to the router and back, and is small (i.e. less than clock resolution) in all cases.
- The differences in β values between Tiny and raw communication shows the time taken to enqueue and dequeue buffers, and to make routing decisions. The additional CPU impact of Tiny on intermediate nodes, which is the difference between β_{raw} and β_{Tiny} , is only 24.4 μ sec/node in a quiet network.
- The parameter α is the sum of operation times which are independent of message length performed in the user processes and their agents on the source and destination nodes, including several synchronization events, protocol and header generation, etc.

To summarize these results, the only significant performance differences between Tiny and raw hardware are the time taken to communicate header information (which is simply the time taken to send the header words through the links) and the once-per-message cost of protocol generation, which affects only the source and destination processors. Both of these costs are very small, as is the through-routing time β . By comparison, the values for Version 2 of Tiny, currently under development, are:

Method	α	β	γ	δ
Tiny V2	30.5	30.0	0.0	1.25

Tiny V2 will avoid internal copies completely (reducing γ to exactly 0), and, despite extra protocol generation and checking, also reduces start-up and through-routing costs.

7 Monitoring

Tiny can route arbitrary networks so that users could tailor the connectivity of the available processors to suit their application, rather than tailoring their problem to fit (for example) a hypercube. However, this raises an interesting question: how can a user determine whether a particular topology, or a particular mapping of processes to processors, is efficient? In the absence of good theoretical models for most of the applications now being put on MIMD computers, users must approach the topology and mapping problems using experiments. Rather than just testing several different configurations (chosen at random, or based on some intuition about the normal pattern of communication within the program), a communications monitor would allow users to get their program running in some simple topology, and then observe the flow of messages in order to locate potential bottlenecks. Such a monitor has been built for Tiny, and has been used to investigate the performance of a parallelised molecular graphics package.

7.1 Monitor Design

The first problem when building a software monitor for a sparsely-connected multiprocessor is how to collect the information the monitor obtains. In the Computing Surface on which this work was done, there were three ways messages containing monitoring information could be collected:

- through links reserved for the monitor's use
- through the Computing Surface supervisor bus (a low-bandwidth bus connecting all the processors in the machine)
- through Tiny itself

The first option was rejected immediately, since reserving links for the monitor would mean that users would have to decrease the inter-connectivity of their programs in order to monitor them. This would change the message-passing behaviour of programs so much that the monitoring information obtained would be irrelevant to the original problem.

The second option would have relied on a global bus which is connected to all of the transputers. This bus is normally used as a secure channel for resetting and querying processors, and for printing debugging messages. However, experience indicated that contention for this (slow) bus would change programs' behaviour significantly.

The final option was to send monitoring information through Tiny using the same links and agents as the application. This is in many ways the simplest choice to implement — since Tiny already makes provision for multiple clients on a single transputer, the monitoring process could appear to it as just another client.

7.2 Collecting Information

Having decided to run a monitoring process as a Tiny client, the next problem was how to collect information from the agents. One option briefly considered was to connect each agent to the monitor through a channel, and have each agent periodically send a description of its current state. This was rejected for two reasons: it would require some sort of ALT in the monitor (which clearly needs to be as efficient as possible), and it would mean that one agent trying to communicate with the monitor could be blocked by other agents. If the agents had to queue up to talk to the monitor frequently, they would spend no time through-routing.

The method actually used takes advantage of the fact that agents are linked together by their AP_LINK fields during initialization. Given a pointer to the first agent in the linked list, the monitor can step through the agents, collecting what information it needed directly from their A-pages. In order to do this correctly, the monitor must run at high priority. (If it did not, it could be interrupted by an agent whose workspace it was examining, which could leave it with an inconsistent picture of that agent's state.)

Two ways of putting the monitor to sleep so that the router and the application get some CPU time have been implemented. In the first, the monitor is given a fixed dormancy time T . It puts itself on the high-priority clock queue, from which it is rescheduled after (at least) T clock ticks have passed.⁶ It then collects information from the agents and sends a message to a collecting process, which is a normal (low-priority) Tiny client.

The second method used to control the monitor is to have it collect information only when requested to. In this case, the monitor dequeues itself waiting on communication, then becomes active when a request arrives. This second mode of operation was found to be the most useful. If every transputer in a large network is running a monitor, the volume of information being sent to

⁶If any agents are active when T ticks have passed, the monitor is queued after them, which may increase its effective dormancy period.

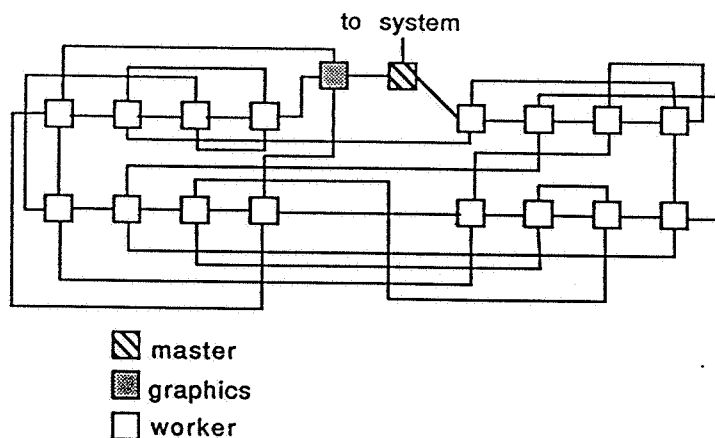


Figure 15: Processor Network for Molecular Graphics Program

the collector is so high that either much of it must be discarded or statistics must be calculated for display. Since sending information only to have it thrown away is counter-productive, and in most cases only statistical properties of the information are of interest, it is better to have monitors collect and send statistics in the first place.

7.3 Recording Agent Information

The snapshots provided by having the monitor examine each agent's workspace turned out to contain too little information to be useful. Two more fields were therefore added to the A-page in order to obtain the statistics desired, and modifications made in the Tcode responsible for enqueueing and dequeueing in order to update their values. These fields are:

```

AP_MSGNM  number of messages dequeued
AP_MSGSZ  total size of messages dequeued (in bytes)
  
```

7.4 Using the Monitor

The monitored version of Tiny was first used to measure traffic in a FORTRAN molecular graphics package called BALL. This package distributes a description of a molecule containing several thousand atoms (represented as spheres) among the available worker processors, along with lighting, shading model, and viewpoint information. Workers are assigned tiles of the image to draw, and generate rasters and send them to the graphics board.

The processor network initially used in this program contained a single master processor, a graphics processor, and 16 workers connected in a dense random graph. Since the graphics processor is an obvious potential bottleneck during the display phase of this program, the first thing examined was the message traffic on its links. The results were:

graphics link	# msgs	# bytes
0	360	40699
1	303	30488
2	617	61802
3	290	30177

The differences in link loadings were unexpected, and could explain why the program's performance was not as good as expected. Even in a dense random graph of 64 worker processors, message traffic was still unbalanced:

graphics link	# msgs	# bytes
0	347	33073
1	461	46266
2	497	51201
3	319	32727

(Total traffic in these two configurations differ because workers each send a synchronisation message to the graphics board when they finish drawing their tiles of the image.) The monitor's information is currently being used to help look for more evenly balanced topologies.

7.5 Monitor Impact

The Heisenberg Principle is as important in monitoring computer operation as it is in quantum mechanics. If CPU cycles are being stolen to collect monitoring information (i.e. to increment `AP_MSGNM` and `AP_MSGSZ` every time a message comes through), the behaviour of the router will necessarily be distorted. If this distortion is great, or cannot be estimated, the value of the monitor information is greatly reduced.

We found that the eight instructions added to each agent to update its monitoring information fields had a negligible effect on Tiny's performance. Figure 16 shows the time taken to echo a message down a chain 1, 2, or 3 processors long with and without monitoring. As can be seen, the run-time degradation introduced by monitoring is almost unnoticeable.

8 Problems with the Current Implementation

There are several problems in the current implementation of Tiny. First and foremost is its interface. While users have found its send and receive functions comfortable, configuring Tiny is difficult and error-prone. The next generation will hide many of these details in order to simplify setup.

A second problem is that during broadcasts, multiple references to a single buffer are queued for several links. This makes the fast pointer-swapping interface unreliable, since the application has no way of knowing when it is safe to re-use the buffer. The pointer-swapping interface is being removed from the next version of Tiny. As partial compensation, Version 2 will reduce message copying by reading data from a link directly into a client's workspace whenever possible.

Finally, the use of multiple references to a single buffer can have subtle side effects during broadcast even when the normal PKT interface is being used. If one of the clients receiving the broadcast message is on the same processor as the sender, the buffer belonging to the sender's

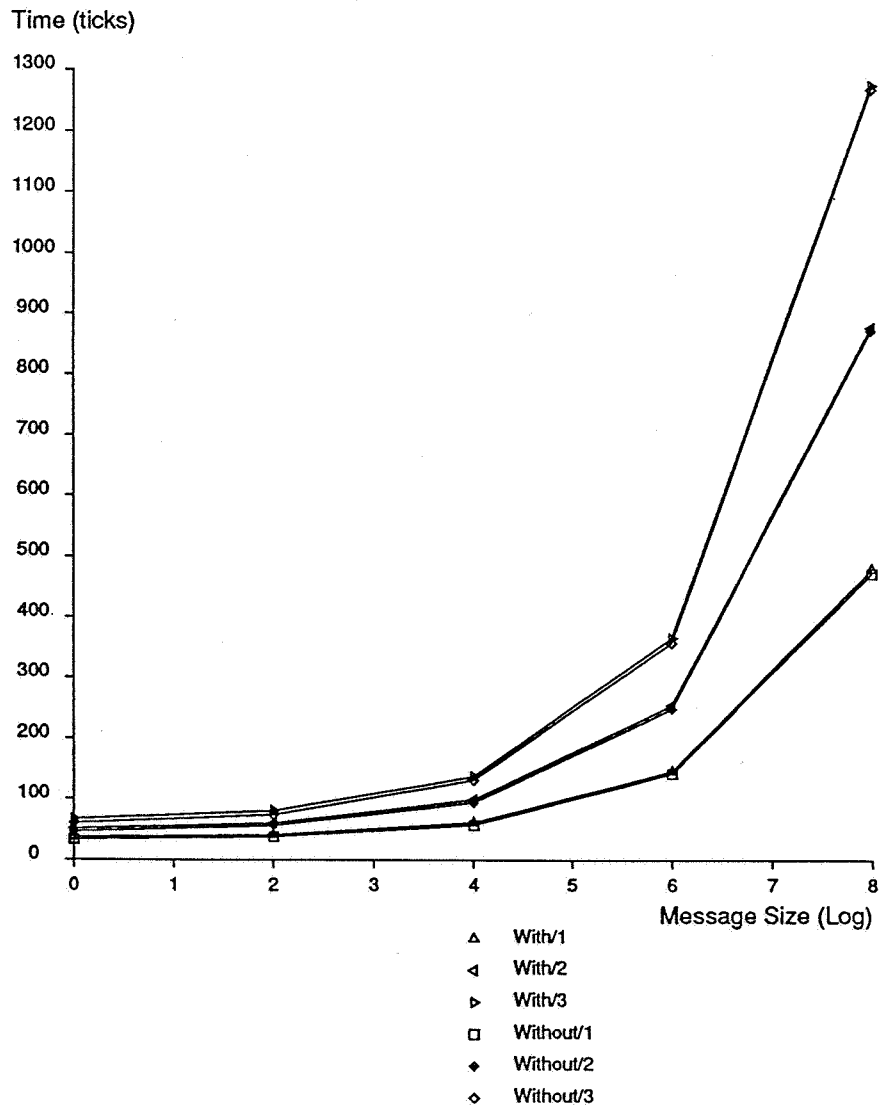


Figure 16: Timings With and Without Monitoring

`agent_usrpkt_output` is queued for that destination. This agent cannot proceed until the destination has read the message and freed the buffer, which forces a partial synchronization between the sender and that receiver. In general, Tiny assumes that clients will never refuse to read on a particular channel indefinitely. Even with cycle-free routing, applications containing clients that do this can deadlock.

9 Conclusions

Two conclusions can be drawn from this work. First, it is possible to build fast, efficient message routers in software for the transputer which are capable of handling arbitrary topologies. The keys to doing this are to avoid redundant copying by taking advantage of the fact that some processes are running on the same transputer as others, and to avoid choices and searches through lists (i.e. not to use ALTs or similar constructs).

Second, it is possible to monitor message traffic in applications built on top of such routers, again using software alone, in a way which does not significantly disturb the behaviour of those applications. Furthermore, the information provided by such monitoring is a useful tool for analysing and improving programs.

References

- [CLA1] L.J. Clarke — “Tiny Discussion and User Guide”, ECSP User Guide # 9, 1989.
- [CLA2] L.J. Clarke — “Communications in Networks of Parallel Processors”, Edinburgh Concurrent Supercomputer Newsletter #7, April, 1989.
- [CLA3] L.J. Clarke — “Efficiently Exploring Random Graphs”, ECSP Technical Note, forthcoming.
- [CLA4] L.J. Clarke — *A Thesis*, Department of Physics, University of Edinburgh, 1990.
- [HLUC] Michael G. Hluchyj, Mark J. Karol — “Queueing in High-Performance Packet Switching”, IEEE J. Selected Areas of Communication, Vol. 6, No. 9, December 1988.
- [PRNC] D.M.N. Prior, N.J. Radcliffe, M.G. Norman, and L.J. Clarke — “What Price Regularity?”, in *Concurrency: Practice and Experience*, forthcoming.
- [HOAR] C.A.R. Hoare — *Communicating Sequential Processes*, Prentice Hall International, London, 1985, 0-13-153289-8.
- [OCC2] INMOS Limited — *occam2 Reference Manual*, Prentice Hall International, London, 1988, 0-13-629312-3.
- [T800] INMOS Limited — *Transputer Reference Manual*, Prentice Hall International, London, 1988, 0-13-929001-X
- [TREF] INMOS Limited — *Transputer Instruction Set: A Compiler Writer's Guide*, Prentice Hall International, London, 1988, 0-13-929100-8.
- [MILN] R. Milner — *Communication and Concurrency*, Prentice Hall International, London, 1989, 0-13-115007-3.
- [WILS] G.V. Wilson — “Examples of Occam Programming 3: A Buffering Library”, ECSP Technical Note # 4, 1988.

Occam Evaluation Kit

Features Statement-level concurrency
Integrated compiler/editor
Portable to all UCSD (Version IV) hosts

Overview The occam evaluation kit is an integrated collection of software and documentation. It provides the capability to experiment with concurrency in system design and programming through the use of the language occam.

Written in UCSD Pascal, it is portable to any host running UCSD p-System version IV. It is available in tailored form for a number of specific hosts, and in Softech distribution format for all others.

Occam is a simple language whose most innovative feature is its handling of concurrency. Providing parallelism right down to the statement level, occam acts as both a programming language and as a design/description language.

The kit is based on an integrated compiler/editor which offers a full screen oriented inserting editor married to an occam compiler. This close coupling allows simple and convenient creation, compilation and correction of occam programs. The compiler generates a standard p-Code object file, for execution by standard p-System commands.

Contents	Software:	Documentation:
	Occam compiler/editor	Occam programmer's manual
	Runtime system	Occam evaluation kit user guide
	Example programs	Getting started

Use The occam evaluation kit provides an introduction to occam as a programming language and as a system design tool. The process oriented view of the world that occam provides is the key to a building block approach to systems design, where each component is a separate process, linked by channels. The connection pattern of processes and channels can exactly mirror the shape of the system which it describes. Design steps of increasing refinement may be applied to the processes defined at the top level to lead from a system description down to a detailed implementation.

This integrated approach to design description and implementation is of vital interest to the creation of future super-complex systems, and will provide invaluable training opportunities for all designers.

Included with the evaluation kit is a collection of example occam programs in source form. Extensively annotated, these programs provide a flying start to learning occam, and are intended to be extended and modified by the user to deepen understanding.

Description The compiler/editor integrates an easy to use screen editor with an occam compiler. The user sees two windows on the screen; one holds the text of the program under construction and the other is used for commands to the editor. The editor has an understanding of the language syntax, simplifying the creation and editing of occam programs. To speed compilation performance, text is lexically analysed as it is typed in.

The compiler signals syntax errors by error messages in the command window while an indication of the error is shown by placing the cursor at the offending program text in the edit window, so that the system is ready to correct the error.

Commands provide occam block editing facilities, full cursor movement around the screen with appropriate scrolling of the text vertically and—for systems with limited linelengths—horizontally.

The compiler may be asked to check the program, to identify any syntax errors, or to generate code from a syntactically correct program. This approach, together with the parallel lexing on input, allows the compiler to be used incrementally throughout the construction of a program, giving feedback on program errors whilst the intentions are still clear in the mind of the designer.

Runtime System

The runtime system includes utilities to allow the user to access system facilities like screen, keyboard, serial links and files. These are all represented as p-System files, and the runtime utilities provide a method for constructing occam channels between the user program and as many named files as are desired.

Thus occam programs can interact with the user, manipulate files and even communicate with occam programs in other hosts via a serial link, allowing true multiprocessor systems to be investigated.

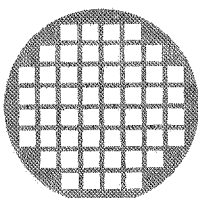
To aid the investigation of the behaviour of occam, debug software provides simple logging facilities to trace the behaviour of programs.

Occam Evaluation Kit Package

The occam evaluation kit is shipped as a number of floppy disks together with documentation. The number and format of the disks is dependent on the target host; the tailored kits use the disk densities, sizes and formats of their hosts, while the uncommitted version uses Softech's 8 inch single sided single density format.

Occam, **inmos** and  are trademarks of the INMOS group.
UCSD p-System is a trademark of the Regents of the University of California.

INMOS reserves the right to make changes in specifications at any time and without notice. The information furnished by INMOS in this publication is believed to be accurate; however, no responsibility is assumed for its use; nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents, trademarks, or other rights of INMOS group.



inmos

INMOS Limited Whitefriars Lewins Mead Bristol BS1 2NP UK Telephone (0272) 290861 Telex 444723

INMOS Corporation PO Box 16000 Colorado Springs Colorado 80935 US Telephone 303 630 4000 TWX 910 920 4904

occam

The choice of what is to be omitted from a new language is in practice much more critical than the choice of what is to be added
Niklaus Wirth

Niklaus Wirth is renowned as the designer of Pascal. Together with Dijkstra and Hoare, he has been influential in establishing the principles of good programming practice. Ada, the most ambitious language development ever attempted, is largely based on Pascal. Wirth's quote is from the original 'Green' submission to DoD for the Ada contest. Green won, but Wirth's comment was omitted from the final version.

Programmers are always surrounded by complexity; we cannot avoid it. If our basic tool, the language in which we design and code our programs, is also complicated, the language itself becomes part of the problem rather than part of its solution
CAR Hoare

Professor Hoare, Director of the Programming Research Group at Oxford University, is well known for his concerns over the unnecessary elaboration of languages. He fears that systems programmed in complex languages may pose dangers in the real world. He received the Turing Award—the ACM's highest honor for technical contributions to the computing community—for his "fundamental contributions to the definition and design of programming languages" with work "characterised by an unusual combination of insight, originality, elegance and impact."

Tony Hoare has been closely involved in the design of occam, the new language developed at INMOS by David May to provide a better tool for programming microprocessors and future systems. The precursors of occam are well structured languages like Algol 60 and Pascal, system languages like BCPL and C and experimental languages like Hoare's CSP—which established the communication primitives subsequently elaborated in Ada.

Sequential systems will not be adequate for the future. There are an additional four orders of magnitude in computational capability available through concurrent systems
Carver Mead

Carver Mead is the foremost advocate of structured VLSI design. He holds the chair at Caltech endowed by Gordon Moore, Chairman of Intel. Carver Mead was joint winner (with Lynn Conway, manager of VLSI system design at Xerox PARC) of Electronics 1981 Achievement Award. He is said to have significantly influenced the design of the Motorola 68000 and Intel iAPX 432.

Concurrency is clearly the key to higher performance systems. Occam is designed to unlock the potential of VLSI by providing the concepts for describing and programming systems containing many interconnected processing elements—the fifth generation systems of the future.

Entia non sunt multiplicanda praeter necessitatem
William of Occam

Occam's Razor—that entities are not to be multiplied beyond necessity—has been the guiding principle in designing this new language. William of Occam was a fourteenth century Oxford philosopher, whose teaching was condemned by the Pope. He is now recognised as having anticipated a basic principle of modern scientific method.

Occam is the new programming language.

Occam is based on the concepts of concurrency and communication. These concepts enable today's applications of microprocessors and computers to be implemented more effectively. They are essential for tomorrow's systems built from multiple interconnected transputers.

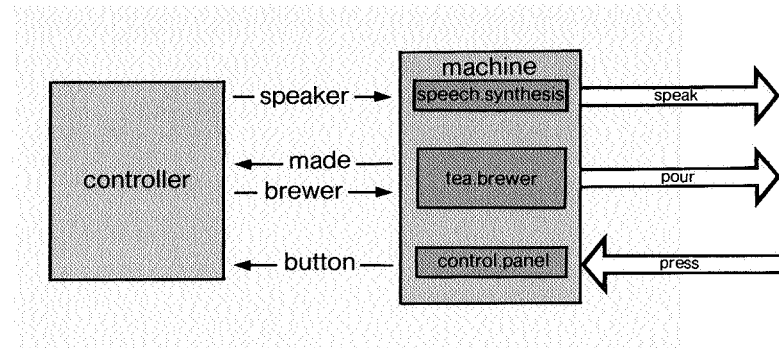
Occam is designed for the professional programmer. The language is oriented to interactive use. It enables complex systems to be programmed in a concise and readable form. As a result, programmer productivity is enhanced.

Occam has a formal basis and uses the minimum of concepts. It is easy to understand and easy to compile for a wide variety of microprocessors and computers.

The use of occam is illustrated by the updating of an old fashioned tea maker. This wakes you up in the morning with a traditional message and offers a hot cup of tea. It is also a clock and will make tea at any other time, on request.

The tea maker has a number of units which interact with each other: the tea brewer which makes and pours the tea, a speech synthesiser for saying 'good morning' and telling the time, request buttons and an overall controller.

These units can be represented as a network:



In occam, each of the units is described by a process and each connection by a channel. The processes communicate by sending messages via the channels. A process can be constructed from smaller processes, as in the case of this machine which has a number of parts. Indeed the collection of processes is itself a process in occam, and could be part of some larger system.

This network is represented by defining the channels and the processes. CHAN introduces the channels through which the processes communicate, and the PAR construct causes the various processes to operate concurrently:

```

CHAN speaker, made, brewer, button
PAR -- tea maker
... -- controller
PAR -- machine
... -- speech synthesiser
... -- tea brewer
... -- control panel
  
```

The controller may do one of three things. Firstly, it may receive a message from the request buttons asking it to make tea, or tell the time:

```

button ? request
IF
  (request = tea.please) AND NOT brewing
  PAR
    brewer ! make.tea
    brewing := TRUE
  request = time.please
  speaker ! say.time, NOW
  
```

This inputs a request from the button channel, and uses IF to determine whether it is a request for tea, or a request for the time. If it is a request for tea, a message is output to the brewer channel telling the tea brewer to make the tea, and the boolean variable brewing is set to prevent further attempts to initiate tea making. If the request is for the current time, a message is output to the speaker channel requesting the speech synthesiser to tell the time, which is the value NOW.

Secondly, the controller may receive a message from the tea brewer, telling it that the tea is made:

```
made ?
  SEQ
    speaker ! say.message, tea.made
    brewer ! pour.tea
    brewing := FALSE
```

This uses SEQ to stop the tea being poured until the tea maker has said 'tea is made'. Finally, at daily intervals, the tea maker may say 'good morning' and make the tea:

```
WAIT AFTER alarm.time
  SEQ
    alarm.time := alarm.time + one.day
    speaker ! say.message, good.morning
  IF
    NOT brewing
  PAR
    brewer ! make.tea
    brewing := TRUE
```

These individual program sections, each of which is a process, are combined into the complete controller process by declaring the local variables, and by using WHILE and ALT to enable the controller to perform whichever alternative is required:

```
VAR alarm.time, brewing := 0, FALSE
WHILE TRUE
  ALT
    buttons ? request
    IF
      (request = tea.please) AND NOT brewing
      PAR
        brewer ! make.tea
        brewing := TRUE
      request = time.please
      speaker ! say.time, NOW
    made ?
    SEQ
      speaker ! say.message, tea.made
      brewer ! pour.tea
      brewing := FALSE
  WAIT AFTER alarm.time
  SEQ
    alarm.time := alarm.time + one.day
    speaker ! say.message, good.morning
  IF
    NOT brewing
  PAR
    brewer ! make.tea
    brewing := TRUE
```

Entia

Model Programs are expressed in terms of concurrent processes, which communicate using channels. An obvious implementation of an occam program is a network of microcomputers, each executing one of the concurrent processes. However, the same occam program can also be executed by a single computer sharing its time between the concurrent processes.

Values The basic data type is a word, which may be used to represent numbers, characters, truth values or bit patterns. Vectors and subscript operations, including record access, are provided. There is a wide range of logical and arithmetic operators for use in expressions.

Structure Programs are constructed from a small number of primitive processes: assignment, input, output and wait. Processes are combined using the constructors sequence, parallel, conditional and alternative.

Assignment An assignment may be used to set the value of a variable to the value of an expression.

Communication A channel provides communication between two concurrent processes. The communication is synchronised, and takes place only when both the input process and the output process are ready; the values being copied from the output process to the input process.

Time Execution of a process may be related to the passage of time. A wait process may be used to delay execution until a specified time is reached.

Sequence The component processes are executed one after the other. A sequence construct terminates after the last of its components has terminated.

Parallel The component processes are executed concurrently. Each component process operates on its own variables, communicating with other concurrent processes using channels. A parallel construct terminates only after all of its components have terminated.

Conditional The component processes are tested in sequence. If one is ready, it is executed. At most, one of the component processes is executed.

Alternative One of the component processes is chosen and executed. The alternative constructor chooses the first component process which is ready to be executed.

Repetition A while construct causes its component process to be executed repeatedly until the result of evaluating a condition is false.

Abstraction In the construction of a process, a name may be used in place of a component process which is to be used or defined elsewhere in the program. A process definition is used to associate such a name with a process. A completely self-contained form of abstraction is provided. This is an independent unit of compilation, and may be transmitted around a system, or loaded when a system is initialised.

Syntax Each primitive process and each constructor is represented by a single line of program. The component processes combined by a constructor follow it on successive lines. This makes interactive editors and compilers simple and efficient.

Semantics The design of occam is based on a formal model which facilitates reasoning about the properties of the language constructs, and the behaviour of specific programs. Each process can be described by an assertion in the predicate calculus, and the composition of processes into networks can be described by the logical conjunction of the assertions describing each process.

Occam Evaluation Kit

This is a complete portable software kit to provide programmers with the opportunity to experiment with occam.

The kit is inexpensive and comprises a compiler and editor together with tutorial examples on a floppy disk. It includes manuals for occam and the compiler itself.

The kit is based on the UCSD p-System version IV and compiles occam into p-code. It can be used on a wide range of computers, from the Apple II to the VAX.

Occam Programming Station

This is a personal interactive work station with comprehensive facilities to develop applications written in occam for microprocessors in the Intel iAPX 86 and Motorola 68000 ranges.

It employs a multiple window interface to give the programmer full control over the display of relevant information.

Programs are created and edited with a language directed editor which uses keystroke lexing to provide instantaneous feedback on syntax errors. An incremental compiler is used to provide the effect of instantaneous compilation after a program change.


When debugging programs, all interactions are expressed using occam, so that the programmer need not be concerned with machine level representations such as assembler and hex. The structure of occam has been exploited to provide significant new debugging facilities including action replay and channel monitoring, while conventional debugging facilities such as trace and breakpoint are also provided.

The entry level station contains a microcomputer system with a bit mapped graphics screen, 256K bytes of memory and dual high density floppy disks.

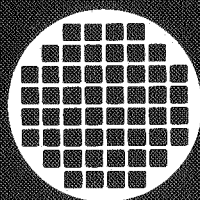
Occam Environment

This is a software package enabling suitably configured Intel and Motorola development systems to be used to develop applications written in occam for microprocessors in the Intel iAPX 86 and Motorola 68000 ranges.

The environment provides a reduced version of the facilities in the Occam Programming Station, determined by the capabilities of the host development system.

Occam, inmos,  are trademarks of the INMOS group.
Apple II is a trademark of Apple Computer Inc.
VAX is a trademark of Digital Equipment Corporation
UCSD p-System is a trademark of the Regents of the University of California
iAPX is a trademark of Intel Corporation
Ada is a trademark of the US Department of Defense

INMOS reserves the right to make changes in specifications at any time and without notice. The information furnished by INMOS in this publication is believed to be accurate; however, no responsibility is assumed for its use; nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents, trademarks, or other rights of INMOS group



inmos

INMOS Limited Whitefriars Lewins Mead Bristol BS1 2NP UK Telephone (0272) 290861 Telex 444723

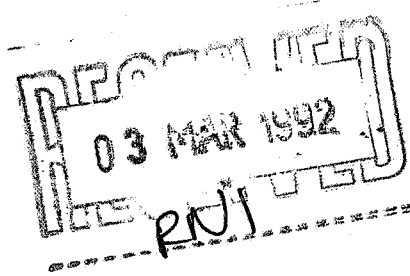
INMOS Corporation PO Box 16000 Colorado Springs Colorado 80935 US Telephone 303 630 4000 TWX 910 920 4904

3 1 MAR 1992

SERC

press notice

Date: 30 March 1992



PN No. 11:92

92085PRO0058

SEMINAR MARKS COMPLETION OF TRANSPUTER INITIATIVE

To mark the completion of the Science and Engineering Research Council/Department of Trade and Industry Initiative in the Engineering Applications of Transputers, a two-day Symposium and Exhibition starts today (30 March) at the University of Reading.

The Initiative was a £2.9 million venture between the SERC and DTI to promote the development of transputer applications within the UK.

Entitled "Transputer Applications - Progress and Prospects", the Symposium considers some of the successful industrial applications developed at the Initiatives regional Centres and goes on to explore the future prospects for the transputer approach to parallel processing by looking forward to the technology of the next generations of transputers, both announced and unannounced. The presentations will be given by leading figures from both industry and academe. The Symposium will point the way forward to the year 2000 and beyond. Over 200 delegates have registered with many coming from outside the UK.

Many of the most well known suppliers of transputer hardware and software are present in the accompanying Exhibition which is the only major exhibition of transputer equipment in the UK this year.

On public view for the first time will be the newly built KAPPA signature verification demonstrator which will be of immense interest to any organisation where transactions are authorised on the basis of signatures.

MORE/...

Science and Engineering Research Council

Press Office
Polaris House
North Star Avenue
Swindon
SN2 1ET

Telephone (0793) 411256/57
Facsimile (0793) 411468
Telex 449466
JANET: PRU @ UK.AC.RLIB

Also on public view for the first time is the Eddy Current NDT Instrumentation Demonstrator which will be of interest to any organisation concerned with crack detection in metals and the new video from the Transputer Initiative 'Transputers In Use' which looks at some of the current industrial uses of Transputers.

The Hexapodal Robot demonstrator which is currently featured in the opening credits to Tomorrows World will also be on display.

Admission to the Exhibition is free.

- ENDS -

Further information can be obtained from:

Terry Mawby, SERC/DTI Transputer Initiative, Rutherford Appleton Laboratory, Chilton, Didcot, Oxon., OX11 0QX. Tel: 0235 44 5787 Fax: 0235 44 5893
email: tpm@uk.ac.rl.inf

NOTES FOR EDITORS

1. PRESS ADMISSION : Free press admission to the Symposium is available. Please contact Terry Mawby at the address below or at the Registration desk on the day.
2. The KAPPA signature verification demonstrator was built by the Electronic Engineering Laboratory at the University of Kent for the Initiatives 'Image Processing Transputer Applications Community Club' (IPTACC) and represents the state of the art in this area.

MORE/...

Its high degree of sophistication allows it to even detect some fraudulent signatures that are geometrically identical to the genuine signature based on dynamic information such as the speed at which the characters of a genuine signature should be written.

3. The Eddy Current Non-Destructive Testing (NDT) Demonstrator was built at the University of Salford under an award from the Initiative's 'Control Transputer Applications Community Club' (CTACC).

Eddy Current NDT is a standard method for detecting cracks in metals; the novel aspect is that instead of the electromagnetic field being displayed on a screen and being interpolated by a human operator, a transputer-based instrument gathers and interprets the data. The instrument examines the field far more quickly and far more accurately than a human operator can, resulting in not only greater efficiency but higher levels of quality, production and safety.

4. A comprehensive review of the work of the Initiative is contained within the Proceedings of the Symposium. Complimentary Press copies will be available from the address below.

5. The major transputer exhibition, the International Transputer Exhibition takes place this year in Barcelona alongside the 4th International Conference on the Applications of Transputers (PACTA92). Both events were founded by the Initiative.

6. Transputers allow parallel processing systems to be built for a fraction of their former cost, permitting such systems to cost effectively address applications where their use was formerly cost prohibitive or very expensive. Additionally their small size has opened up the embedded controller market to parallel processing for the first time. They can also be used as add-in boards for existing computers or as stand alone computers in their own right, such is their versatility.

MORE/...

6. Transputers are made by Inmos Ltd. of Bristol, originally a UK Government start -up company. Inmos is now a member of SGS-Thomson Microelectronics NV, a Franco-Italian company in which Thorn-EMI, Inmos' previous owners, have a 10% share.

The transputer

Colin Whitby-Stevens
INMOS Limited,
Whitefriars, Lewins Mead,
BRISTOL, BS1 2NP, UK

Abstract

The transputer is a programmable VLSI component with communication links for point-to-point connection to other transputers. Occam () is a language that enables a multi-transputer system to be described as a collection of processes that operate concurrently and communicate using message passing via named channels.*

The INMOS transputer architecture is standardized at the level of the definition of occam (rather than at the level of the definition of an instruction set). The implementation of the first commercially available transputers is illustrated by describing the implementation of occam.

The paper concludes with outline examples of some applications.

1 Introduction

The transputer architecture has been developed to fulfil four main objectives:

To create a commercial product range that sets new standards in ease of programming and ease of engineering.

To provide the maximum performance to the user.

To exploit future developments in VLSI technology within a compatible family.

To create a programmable component that can be used to build systems with large numbers of concurrent computing elements.

VLSI currently permits 5-10 MIP processors to be manufactured in volume for low prices. There is therefore no economic barrier to the construction of very powerful computer systems containing many processing elements. The challenge is a technical one: how to engineer a system with, say, 1000 processors so as to make the inherent concurrency usable, and how to support the design of applications to take advantage of this amount of concurrency.

(*) occam is a trade mark of the INMOS Group of Companies

In the transputer architecture, the exploitation of a high degree of concurrency is made possible through a decentralized model of computation, in which local computation takes place on local data, and concurrent processes communicate by passing messages on point to point channels. The localized communications architecture also has substantial engineering advantages, described below.

An important design objective of occam and the transputer was to provide the same concurrent programming techniques both for a single transputer and for a network of transputers. Consequently, the features of occam were chosen to ensure an efficient distributed implementation on transputer systems. The concurrent processing mechanisms within the transputer were then designed to match.

The result is that a program ultimately intended for a network of transputers can be compiled and executed efficiently by a single computer used for program development. Once the logical behaviour of the program has been verified, the program may be configured for execution by a single transputer (low cost), or for execution by a network of transputers (high performance), or for a configuration representing a trade-off between these two extremes.

The choice of local processing and communications necessitates a significant change in programming concepts, and new algorithms need to be developed [4]. The study of various applications from this point of view is showing encouraging results ([15], [16], [17], [18], [19], [20], [21], [22]) and illustrative applications are given at the end of this paper.

2 Transputer architecture

2.1 Overview

The architecture of the transputer is defined by reference to occam. Occam provides the model of concurrency and communication for all transputer systems. Defining the architecture at this level leaves open the option of using different processor designs in different transputer products. This allows implementations which are optimized for different purposes. It also allows implementations to evolve with changes in technology, without compromising the standards established by the architecture.

A transputer contains memory, a processor and a number of standard point-to-point communication links which allow direct connection to other transputers. The processing capability may be general purpose, or may be optimized to a specific purpose. The on-chip memory may be extended off chip by a suitable interface.

A transputer may also have special purpose interfaces for connection to specific types of hardware. The separation of

the transputer system interface from other interfaces (eg the memory interface) means that it is possible to optimize the various interfaces individually, simplifying their use and improving their performance.

A system is constructed from a collection of transputers which operate concurrently and communicate through the standard links. Occam formalizes the computational model. It enables such a system to be described as a collection of processes operating concurrently and communicating through named channels.

Transputers directly implement the occam model of a process. Internally, an individual transputer can behave like any occam process within its capability; in particular, it can implement internal concurrency by timesharing processes. Externally, a collection of processes may be configured for a network of transputers. Each transputer executes a component process, and occam channels are allocated to links, which directly implement occam message-passing.

2.2 Occam

Occam [1, 3, 4] enables a system to be described as a collection of concurrent processes, which communicate with each other and with peripheral devices through channels. Occam programs are built from three primitive processes:

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

The primitive processes are combined to form constructs. Each construct is introduced by a keyword, followed by a list of the component processes:

SEQuential components executed one after another
PARallel components executed together
ALTernative component first ready is executed

A construct is itself a process, and may be used as a component of another construct.

Conventional sequential programs can be expressed with variables and assignments, combined in sequential constructs. **IF** and **WHILE** constructs are also provided.

Concurrent programs can be expressed with channels, inputs and outputs, which are combined in parallel and alternative constructs.

Each occam channel provides a communication path between two concurrent processes. Communication is synchronized and takes place when both the inputting process and the outputting process are ready. The data to be output is then copied from the outputting process to the inputting process, and both processes continue.

An alternative process may be ready for input from any one of a number of channels. In this case, the input is taken from the channel which is first used for output by another process.

The choice of synchronized communication prevents the loss of data. The choice of unbuffered communication removes the need for any store to be associated with the channel. Copying data from the outputting process to the inputting process is clearly essential for communication between transputers, and it is easy to make copying within a machine fast by use of microcode.

2.2.1 Design correctness

It is necessary to ensure that systems built from transputers, possibly involving hundreds or thousands of concurrent devices, can be designed and programmed effectively.

The design of occam and the transputer architecture has followed two principles to help the designer increase his confidence that his design is correct: simplicity and formality.

Occam has been kept simple, with the aim of making it easy to learn, and easy to use [3].

Formal techniques become much more important when concurrency is involved, as techniques based on exhaustive testing are impracticable. Occam has been designed to have a formal semantics. The way that this was achieved was to define a set of formal properties that the language should possess. These take the form of a number of behaviour-preserving transformations that should be applicable to any occam program [12]. Many semantic issues in the design and development of the language were resolved by reference to this set of properties. Enforcing this discipline has enabled a formal semantics for the language to be developed [13], and has laid the basis for software engineering tools ranging from formal validation to program transformation.

Practical and immediate benefits have been that the language is very self-consistent (which makes life easier for the compiler writer and user alike), that the equivalence of concurrent algorithms can be studied, and that programs can be transformed to have greater or less decentralisation without changing their logical behaviour[4].

2.2.2 Real time

On an individual transputer, a parallel construct may be configured to prioritize its components, and an alternative construct may be configured to prioritize its inputs. A higher priority process always proceeds in preference to a lower priority one.

The equivalent of an interrupt (a high priority process being scheduled in order to respond to an external stimulus) is designed entirely in occam, as all input and output is formalized as channel communication. A high priority process may wait for the first of several different inputs to become ready by using the **ALT** construct.

A high priority process proceeds until it terminates or has to wait for a communication. A system can thus be designed to meet real-time constraints by designing each high priority process so that the amount of processor time it requires over a given period is bounded, thus placing a bound on the total time that a high priority process may have to wait for the cpu. In many cases, it may be possible to reason that two or more high priority processes will never conflict, and that the latency reduces to the time required to switch from a low priority process to a high priority process. Each transputer implementation places a bound on this time.

A global synchronized sense of time is not practicable, and not representative of real-world situations. There is therefore a local concept of time, each timer being implemented as an incrementing clock.

Logically, access to a timer is treated as an input. A delayed input may be used, which waits until the value of the clock reaches an appropriate value. A timer input may be used in an alternative construct. This can be used to provide timeout on a communication.

2.3 Inter-transputer links

A link between two transputers provides a pair of occam channels, one in each direction. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer by two one-directional signal lines. Each signal line carries data and control information.

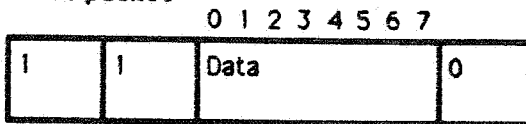
Communication through a link involves a simple protocol, which provides the synchronized communication of occam. The protocol provides for the transmission of an arbitrary sequence of bytes, which allows transputers of different wordlength to be connected.

Each message is transmitted as a sequence of single byte communications, requiring only the presence of a single byte buffer in the receiving transputer to ensure that no information is lost.

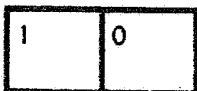
Each byte is transmitted as a start bit followed by a one bit followed by the eight data bits followed by a stop bit. After transmitting a data byte, the sender waits until an acknowledge is received; this consists of a start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged byte, and that the receiving link is able to receive another byte. The sending process may proceed only after the acknowledge for the final byte of the message has been received.

Figure 1 Link protocol

Data packet



Acknowledge packet



Data bytes and acknowledges are multiplexed down each signal line. An acknowledge is transmitted as soon as reception of a data byte starts (if there is a process waiting for it, and if there is room to buffer another one). Consequently transmission may be continuous, with no delays between data bytes.

Using point to point serial communications, rather than busses has a number of advantages:

Board layout is much simplified.

Communications bandwidth is increased, as many links in a system can operate concurrently.

Devices of different word lengths and performance can be easily interconnected.

Transputers with different word lengths and performance will all interwork together, as will all future products, ensuring that systems can be readily upgraded as the technology advances. It is not necessary to downgrade the performance of a connected set of components to that of the slowest.

2.3.1 Electrical properties of links

The signals are TTL compatible and their range can be extended by inserting industry standard line drivers and receivers. The standard transmission rate is 10MHz, providing a maximum performance of about 1MByte/sec in each direction on each link.

The links are designed to make the engineering of transputer systems as easy as possible. Irrespective of internal performance, all transputers use a reference clock of 5MHz, and this is required only for approximate frequency information and not for phase. All future transputers will also use this same frequency. The low frequency was chosen to simplify the distribution of the clock in a large system and it is not necessary for all transputers to be on the same clock, enabling interworking between independently designed systems. Thus, transputers can be interconnected just as easily as TTL gates - indeed, the constraint on the designer is just the same - he must not exceed the maximum capacitance.

3 Implementation

3.1 Instruction set requirements and overview

The first transputer product is the T424, a general purpose 32 bit machine with 4K bytes of on-chip memory (which can be extended with off chip memory) and four bi-directional communications links, which provide a total of 8Mbytes per second of communications bandwidth. This will shortly be followed by the T222, a 16 bit machine providing similar facilities.

The design objectives of the II instruction set and the processor for these first transputers were as follows:-

To provide an efficient implementation of occam, so that the use of high level languages results in efficient use of silicon capability, and that highly concurrent programs execute with minimum overheads.

To provide a simple and direct implementation of occam so that programs can be compiled simply and straightforwardly, and to ensure that there is no need to consider programming at a lower level than that defined architecturally.

To provide word length independence, so that a program can be executed using processors of different word lengths without recompilation.

To provide position independence, so that program and workspaces may be allocated anywhere in memory after compilation.

To provide low latency response to communications with external devices.

The lowest level of programming transputers is to use occam (occam is equivalent in effectiveness to a conventional microprocessor's assembler). The instruction set, and the use of occam as its programming language, is therefore illustrated by describing the main usage of the various registers in the machine, and by giving typical instruction sequences for simple occam constructs. Note that it is not common practice to abbreviate the names of the instructions, or to use mnemonics. Transputer system designers have no general need to write down instruction sequences, and using full names aids readability of the examples.

3.2 The II instruction set

3.2.1 Performance note

Two important performance measures are the number of bytes to hold the program, and the speed of execution provided by an implementation. It should be realized that the speed of execution of individual instructions is less important than the speed with which key system functions are performed, bearing in mind the intended uses of the machine.

The II instruction set is designed specifically with a view to efficient and fast VLSI implementation, although various trade-offs of performance versus silicon area are still possible. On the first transputers, each instruction is executed in one or more processor cycles using one level microcode. The figures given in this paper assume that program and data are stored on chip. Extra cycles may be required if program and/or data are stored off chip, though the significance of this can be reduced to a low level with careful organisation of the application. Full details are given in [14].

It should be noted that although all transputers have an external clock cycling at 5 MHz, the internal speed is set as part of the manufacturing process. It is expected that the range of speeds of the first transputers will provide internal processor cycle rates of up to 20MHz.

The design of the first transputers carefully balances the costs of memory access and alu operation, and contains sufficient overlap to ensure a high degree of efficiency. Many of the instructions execute in a single cycle, and typical sequences of commonly used instructions can deliver a 15 MIPS execution rate.

3.2.2 Memory organization

The memory address space comprises a signed linear address space. The instruction architecture does not differentiate between on-chip and off-chip memory. This allows the application designer to have complete control over the placement of code and data to take advantage of the performance benefits of on-chip memory.

A byte in memory is identified by a single word value called a pointer. A pointer consists of two parts: a word address and a byte selector. The byte selector contains as many bits as are needed to identify a single byte within a word and occupies the least significant bits of the pointer. For example, in a 24 bit transputer the word address would occupy the 22 most significant bits and the byte selector the 2 least significant bits.

Special instructions, such as *load local pointer* and *word subscript*, are provided to construct and manipulate pointers. Pointer values are treated as signed integers, starting from the most negative integer and continuing, through zero, to the most positive integer. This enables the standard comparison functions to be used on pointer values in the same way that they are used on numerical values.

The addressing instructions provide access to items in data structures, using short sequences of single byte instructions, allowing the representation of data structure access to be independent of the word length of the processor.

3.2.3 Registers

The design of the transputer processor exploits the availability of fast-on-chip memory by having only a small number of

registers; six registers are used in the execution of a sequential process. In the internal organization of the processor, all internal registers and data paths are the wordlength number of bits wide. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data paths and control logic.

The six registers are:

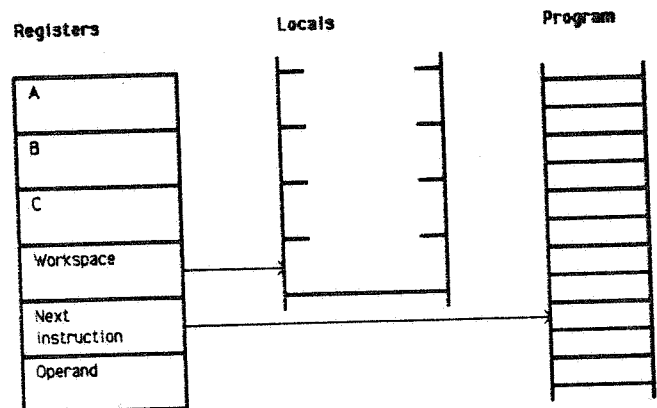
The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The A, B and C registers which form an evaluation stack. The evaluation stack is used for expression evaluation, to hold the operands of scheduling and communication instructions, and to hold parameters of procedure calls.

Figure 2 Registers for sequential programming



The evaluation stack removes the need for instructions to specify registers explicitly. Consequently, most of the executed operations (typically 80%) are encoded in a single byte. The II instruction set saves on time and area through not having to decode secondary control fields or register fields.

3.2.4 Support for concurrency

The processor provides efficient support for the occam model of concurrency and communication. It has a scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel. The processor does not need to support the dynamic allocation of storage as the occam compiler is able to perform the allocation of space to concurrent processes. There is also no need for the hardware to perform access checking on every memory reference, resulting in an overall improvement in performance.

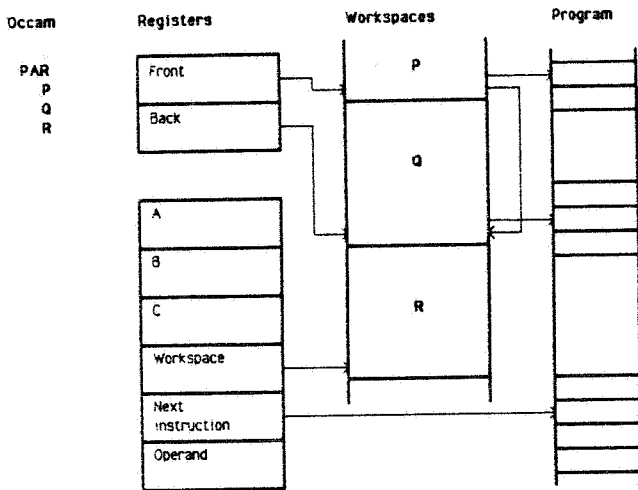
At any time, a concurrent process may be

active - being executed
 - on a list awaiting execution

inactive - ready to input
 - ready to output
 - waiting until a specified time

The active processes waiting to be executed are held on a list. This is a linked list of process workspaces, implemented using two registers, one of which points to the first process on the list, the other to the last.

Figure 3 Concurrent processes



A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the components of the parallel construct which have still to terminate. When the components have all terminated, the counter reaches zero, and a specified process can then proceed.

The processor supports two priority levels, implemented using two lists as described above. A switch from a priority 1 process (low priority) to priority 0 process (high priority), or vice versa, may occur when a process stops, when a channel becomes ready, or when a communication completes and causes a priority 0 process to become ready.

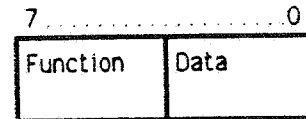
To allow a maximum latency figure to be calculated, the instructions which may take a long time to execute have been implemented to allow a switch during execution. Consequently, the maximum time taken to switch from priority 1 to priority 0 is 58 cycles (less than three microseconds with a 50ns processor cycle time). The switch from priority 0 to priority 1 only takes place when there is no priority 0 work available. The time taken for the switch is 17 cycles.

A context switch between processes, both executing at priority 1, occurs only at times when the evaluation stack has no useful contents, and therefore affects only the instruction pointer and the workspace pointer. With the need to save and restore registers at a minimum, the implementation of concurrency is very efficient.

3.2.5 Instruction format

All instructions have the same format. Each is one byte long, and is divided into two 4 bit parts. The four most significant bits of the byte are a function code, and the four least significant bits are a data value.

Figure 4 Instruction format



The use of a single instruction format requires only a simple decode mechanism in the processor, which reduces area and increases speed. The use of single byte instructions decouples the instruction format from the wordlength of the machine. In particular it avoids the commonly found problems concerned with aligning instructions on word boundaries.

Short instructions also improve the effectiveness of the instruction fetch mechanism, which in turn improves processor performance. The processor uses otherwise spare memory cycles to fetch instructions. As memory is word accessed, a 32 bit transputer will receive four instructions for every fetch. There are two words of instruction fetch buffer so that the processor rarely has to wait for an instruction fetch before proceeding (only on transfers of control if on-chip memory is used). Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be filled.

There is no instruction cache, as only rarely would such a cache reduce the number of processor cycles required. An on-chip cache incurs a significant cost in terms of chip area, as a cache requires several times the area of a simple memory to store the same amount of information. An off-chip cache complicates the external interface. Both require extra logic, even when aided by software (as in the IBM 801 [7]), which would be likely to slow down the overall speed of operation and use up even more chip area. The view is taken that the chip area is better spent on providing memory for the application.

3.2.6 Direct functions

The representation provides for sixteen functions, each encoded as a value in the range 0 to 15. Thirteen of these values are used to encode the most important functions performed by any computer. These include:

- | | |
|---------------------------|-------------------------|
| <i>load constant</i> | <i>load non local</i> |
| <i>add constant</i> | <i>store non local</i> |
| <i>load local</i> | <i>jump</i> |
| <i>store local</i> | <i>conditional jump</i> |
| <i>load local pointer</i> | <i>call</i> |

The most common operations in a program are the loading of small literal values, and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded onto the evaluation stack with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non local* and *store non local* instructions behave similarly, except that they access locations in memory relative to the A register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of block structured programming languages. This eliminates the need for complicated and difficult-to-use addressing modes.

In the following examples, *x* and *y* are assumed to be local variables allocated to offsets *x* and *y* respectively in the first sixteen words of workspace.

occam	instruction sequence	bytes	cycles
<i>x := 0</i>	<i>load constant 0</i>	1	1
	<i>store local x</i>	1	1
<i>x := y</i>	<i>load local y</i>	1	2
	<i>store local x</i>	1	1

In this example, *z* is assumed to have been declared externally to the PROC which contains this assignment statement. The compiler allocates a local workspace location, at offset *staticlink*, to hold the address of the workspace that contains the variable *z*.

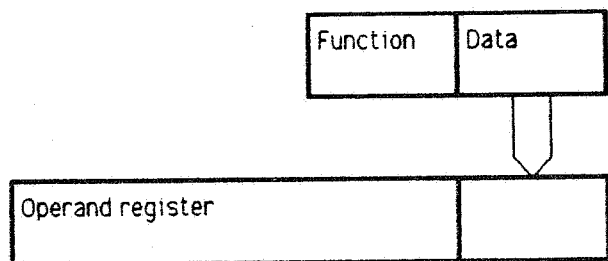
occam	instruction sequence	bytes	cycles
<i>z := 1</i>	<i>load constant 1</i>	1	1
	<i>load local staticlink</i>	1	2
	<i>store non local z</i>	1	2

3.2.7 Prefixing functions

Two more of the function codes, *prefix* and *negative prefix*, are used to allow the operand of any instruction to be extended in length.

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefixing instructions end by clearing the operand register, ready for the next instruction.

Figure 5 Use of operand register



The *prefix* instruction loads its four data bits into the operand register, and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefixing instructions. In particular, operands in the range -256 to 255 can be represented using one prefixing instruction.

The following example shows the instruction sequence for loading the hexadecimal constant #754 into the A register, and gives the contents of the O register and the A register after executing each instruction

	O register	A register
<i>prefix #7</i>	#7	?
<i>prefix #5</i>	#75	?
<i>load constant #4</i>	0	#754

The use of prefixing instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation, by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form which is independent of the processor wordlength.

Each prefixing instruction occupies one byte and takes one cycle to execute.

3.2.8 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. For example, the plus operation adds the values of the A and B registers. The result is left in the A register, and C is copied into the B register.

The *operate* instruction allows up to 16 such operations to be encoded in a single byte instruction. However, the prefixing instructions can be used to extend the operand of an *operate* instruction just like any other.

The encoding of the indirect functions is chosen so that the most frequently occurring operations are represented without the use of a prefixing instruction. These include arithmetic, logical and comparison operations, together with the most frequently used control functions and register manipulation functions.

Less frequently occurring operations have encodings which require a single prefixing operation (the transputer instruction set is not large enough to require more than 512 operations to be encoded!).

3.2.9 Expression evaluation

Loading a value onto the evaluation stack pushes B into C, and A into B, before loading A. Storing a value from A, pops B into A and C into B.

The A, B and C registers are the sources and destinations for arithmetic and logical operations. For example, the *add* instruction adds the A and B registers, places the result in the A register, and copies C into B.

If there is insufficient room to evaluate an expression on the stack, then the compiler introduces the necessary temporary variables in the local workspace. However, expressions of such complexity are, in practice, rarely encountered. Three registers provide a good balance between code compactness and implementation complexity.

Single length signed and single length modulo arithmetic is directly supported. In addition, a quick unchecked multiply is provided, in which the time taken is proportional to the logarithm of the second operand. The performance of these instruction sequences compares favourably, in both space and time, to that achieved by more complex instruction sets. Where a more complex instruction set cannot achieve the same effect in a single instruction, the performance gain is significant.

occam	instruction sequence	bytes	cycles
x + 2	load local x	1	2
	add constant 2	1	1
(v + w) * (y + z)	load local v	1	2
	load local w	1	2
	add	1	1
	load local y	1	2
	load local z	1	2
	add	1	1
	multiply	2	7+wordlength

3.2.10 Input and output

A channel provides a communication path between two processes. Channels between processes executing on the same transputer are implemented by single words in memory (internal channels); channels between processes executing on different transputers are implemented by point-to-point links (external channels).

As in the occam model, communication takes place when both the inputting and outputting processes are ready. Consequently, the process which first becomes ready must wait until the second one is also ready.

A process prepares for an input or an output by loading the evaluation stack with a pointer to a buffer, the identity of the channel, and the count of the number of bytes to be transferred. It then executes an *input message* or an *output message* instruction as appropriate.

The *input message* and *output message* instructions use the address of a channel to determine whether the channel is internal or external. This means that the same instruction sequence can be used for both internal and external channels, allowing a process to be written and compiled without knowledge of where its channels are connected. In particular, either an internal or an external channel can be used as the actual parameter for a channel parameter of a named process.

A communication primitive communicating a block of size n bytes requires only one byte of program, and on average the maximum of $(24, 21+(8*n/\text{wordlength}))$ cycles (including the scheduling overhead).

Instructions for enabling and disabling channels provide support for an implementation of alternative input without the use of polling.

3.3 Discussion

The requirements of the transputer indicate that a transputer processor should have a simple design. A transputer has a substantial amount of area given over to memory and communications, indeed a transputer can be thought of as a memory chip with a processor in one corner. In fact, the processor on the first transputers occupies about 25% of the available area.

It was clear that a simple processor could be constructed which would leave the majority of a chip area available for other purposes. The early RISC experiences [6, 7, 8, 9] lent further support to the evaluation that performance resulting from using a simple processor need not suffer.

Various projects, for example the IBM 801 [7] and MIPS [8], are willing to pay a price of software complexity in order

to achieve implementation efficiency. However, the evidence of interpretive schemes for high level languages was that a simple instruction set could be designed which would lead to a better hardware/software relationship, and hence simplify the software as well. This would probably mean rejecting the strategy of compiling to a level best considered as microcode.

The justification for the use of multiple cycle instructions must be that the instructions well match the software requirements. In the transputer processor for the II, repetitive operations, such as multiply, and block move, are implemented by microcode (with hardware assistance). The alternative RISC implementation [9] is to provide, for example, a single cycle multiply step, and for the software to compile the appropriate loop. The efficiency, in both code space and execution speed, resulting from the microcoded solution outweighs the cost of area and capacitance in the microcode ROM.

The II instruction set achieves word length independence, in that a program which manipulates bytes, words and truth values can be translated into an instruction sequence which behaves identically whatever the wordlength of the processor executing it (apart from overflow conditions resulting from word length dependencies). This results from the fact that the instruction size is independent of wordlength, the method of representing long operands as a sequence of prefixing instructions, and the memory addressing structure.

Workspaces are held in addressable memory, which the designer can choose to allocate on chip or off chip. Holding workspaces on chip forms a very effective alternative to the use of cache memory [11], the cost of which has already been discussed. A further advantage is that, unlike cache memory, rarely accessed data need not be brought on chip.

In general, a program needs much less store to hold it than an equivalent program in a conventional microprocessor. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. As memory is word accessed, the processor will receive several instructions for every fetch (depending upon the number of bytes in a word).

The overall effect is thus that both compactness and speed have been achieved, together with economical use of silicon.

4 The transputer as a family

The T424 32 bit transputer is the first of a range of transputer products [14]. The next products will be a 16 bit transputer offering similar facilities to the T424, a high performance disk controller and a high performance graphics controller.

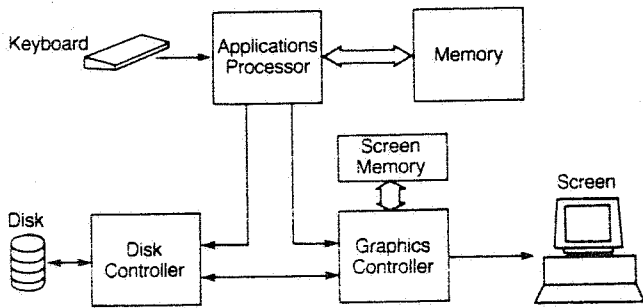
A transputer family device controller has the same organisation as a transputer, with the addition of special high speed control logic and interfaces. Device controllers are programmable, in occam, in the same way as transputers. This allows a designer to tailor the controller's function to his particular application.

4.1 A personal workstation

This section explores the design possibilities provided by the transputer architecture. The first step is the outline design of a personal workstation, which can be designed and built using functionally distributed transputers. One transputer, the applications processor, accepts the user's commands and carries out the appropriate processing, calling on two other transputers, which look after a disk system and a graphics display system respectively. Each of the latter two transputers

and associated hardware can be replaced by transputer based device controllers as they become available.

Figure 6 Personal computer workstation



The transputers are connected together using the standard transputer communications links. The resulting system can be engineered onto a single card.

The architecture permits a number of variations on the implementation of the workstation to be made without major redesign.

For example, the disk controller can double as the applications processor, and the applications transputer removed completely. Alternatively, more processors can be added, and the occam processes redistributed to take advantage of the additional concurrency. Vastly more than 1 Mbyte of memory could be attached.

4.2 Transputer without external memory

This second example explores the design and use of a large amount of processing power based on a transputer with only link interfaces in, say, a 28 pin chip carrier.

Figure 7 Single board transputer system

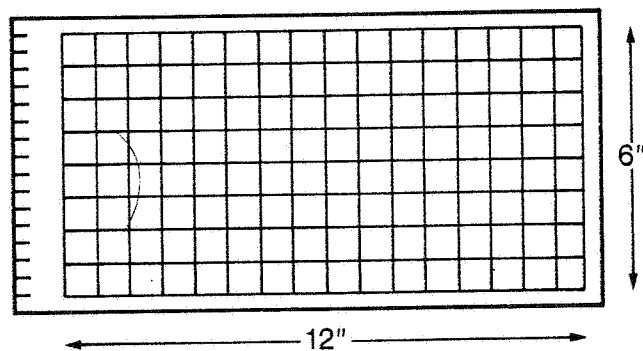
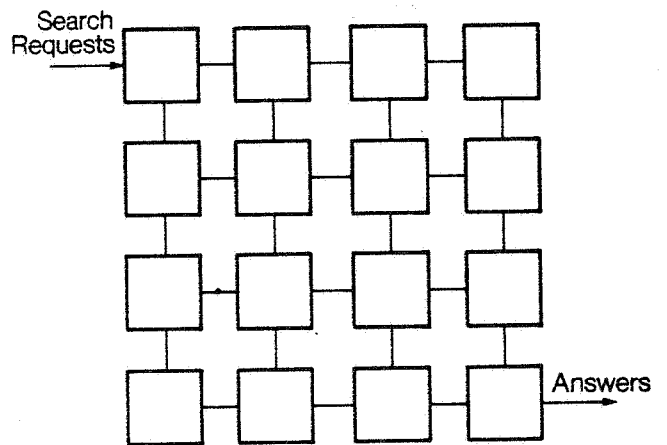


Figure 7 shows 128 transputers on a single printed circuit board. The board has 1/2Mbyte of fast static RAM and up to 1 GIPS (Giga Instruction Per Second) of processing power.

In this application, the board is used to provide high performance database searching. We assume that the database is partitioned, so that the most commonly accessed parts of a database can be placed in the transputer array.

The concept is shown in a simplified form in figure 8.

Figure 8 Concurrent database search



Here 16 transputers are connected into a square array with search requests input at one corner of the array, and answers being output from the other corner. Each transputer keeps a small part of the database in its local memory.

A small program in each transputer does the search. It can receive two sorts of input. A search request is forwarded to any connected transputer which has not yet received the request and simultaneously a search is made through the local data. The other sort of input is an answer from a transputer which has just searched its own local memory. This answer is merged with the answer generated from the local data and forwarded.

A simple performance analysis indicates the latency and throughput of this application on the 128 transputer board. Assume that each record is 16 bytes long, and that a search key is four bytes long. Each transputer can hold 200 records and the whole system can hold 25,000 records. For each transputer to search its own records against a request will take less than a millisecond.

The time taken to transmit a search request to each transputer in the array is proportional to the longest path across the system, in this case 24 links.

It takes about 6 microseconds to send a 4 byte message from one transputer to another. It will thus take about 150 microseconds to transmit a search request to the whole array, and about another 150 microseconds to transmit the answer. The whole search of 25,000 records will take less than 1.3 milliseconds.

However just as an individual transputer can be performing input, output and processing at the same time, so can the array. Requests can be pipelined through the system with a further request being input before the previous one has come out.

The size of the database partition can be increased by adding more boards. The search throughput is not adversely affected by this.

5 Conclusions

By taking an integrated approach to the design of a VLSI computer and a concurrent programming language it is possible to produce a new level of system building block which provides a very efficient implementation of the corresponding design formalism.

In particular, it is possible to support the use of the same concurrent programming techniques both within a single transputer and for a network of transputers. The concurrent processing features of a general purpose programming language can be efficiently implemented by a small, simple and fast processor.

The resulting transputer provides the unique concept of a programmable component enabling highly concurrent systems to be implemented within a formal design framework.

The architecture also provides a straight forward technology upgrade path. Future transputers can integrate more memory and more processors. The system architecture means that current and future products will be fully compatible and capable of interworking.

6 Acknowledgements

A large number of people have made invaluable contributions to the development of the transputer architecture and family of products, and these contributions are hereby collectively acknowledged. In particular, the original concept, and the drive to give it commercial reality, comes from Iann Barron, one of the founders of INMOS. David May designed occam, and led the team which developed the instruction set of the first products. Prof Tony Hoare, of Oxford University, has advised INMOS both generally on architecture and particularly on the basis for providing occam with a formal semantics.

7 References

- [1] INMOS Limited, *Occam Programming Manual*, Prentice-Hall International, London, 1984.
- [2] Barron, I.M. et al., *The Transputer*, Electronics, 17th Nov 1983, p 109.
- [3] May, M.D., *OCCAM*, ACM SIGPLAN Notices vol 18-4 (Apr 1983) pp69-79.
- [4] May, M.D. and Taylor, R.J.B., *OCCAM*, Microprocessors and Microsystems vol 8-2 (Mar/Apr 1984)
- [5] May, M.D. and Shepherd, R., *Occam and the transputer*, IFIP WG10.3 workshop on Hardware Implementation of Concurrent Languages and Distributed Systems, North Holland (1984)
- [6] Patterson, D.A and Sequin, C.H., *RISC I: A Reduced Instruction Set VLSI Computer*, Proc 8th International Symposium on Computer Architecture.
- [7] Radin, G, *The 801 Minicomputer*, IBM Journal of Research and Development, Vol 27, No 3, pp237-246 (May 1983)
- [8] Hennessy, J et al, *The MIPS machine*, Proceedings CompCon Spring 1982, IEEE, (February 1982)
- [9] Colwell, R.P. et al, *Peering Through the RISC/CISC Fog: An Outline of Research*, Computer Architecture News, Vol 11, No 1 (March 1983)
- [10] Patterson, D.A., *RISC Watch*, Computer Architecture News, Vol 12, No 1 (March 1984)
- [11] Patterson, D.A. et al, *Architecture of a VLSI Instruction Cache for a RISC*, Proc 10th International Symposium on Computer Architecture, pp108-116, ACM (1983)
- [12] Hoare, C.A.R. and Roscoe, A.W., *Programs as Executable Predicates*, Proc 1st Intl Conf on Fifth Generation Computer Systems, ICOT, 1984
- [13] Roscoe, A.W., *Denotational Semantics for occam*, Proc NSF/SERC Workshop on Concurrency, Springer LNCS, 1984
- [14] -, *IMS T424 transputer data sheet*, INMOS Limited, Bristol, England
- [15] Schindler, M. *Real-time languages speak to control applications*, Electronic design, July 21, 1983, pp105-120.
- [16] Fay, D. *Working with occam: a program for generating display images*, Microprocessors and Microsystems, Vol 8. No 1, Jan/Feb 1984
- [17] Curry, B. Jane, *Language based architecture eases system design*, Computer Design, Jan 1984, pp127-136
- [18] Taylor, R., *Graphics with the transputer*, Computer Graphics 84, 1984
- [19] Pountain, R., *The transputer and its special language. occam*, Byte, Vol 9, No 8, Aug 1984
- [20] Kerridge, J.M. and Simpson, D., *Three solutions for a robot arm controller using Pascal-Plus, occam, and Edison*, Software Practice and Experience, Vol 14, No 1, Jan 1984
- [21] Harp, J.G. et al, *Signal processing with transputers (traps)*, Computer Physics Communications (in press)
- [22] Broomhead, D.S. et al, *A practical comparison of the systolic and wavefront array processing architectures*, 2nd Proc IEEE Conf on Acoustics, Speech and Signal Processing (March 1985).



IT WORLD

CONSULTANTS IN INFORMATION TECHNOLOGY

MARKET SURVEY

SERC/DTI TRANSPUTER INITIATIVE

Meeting the Needs of UK Industry

Final Report

**IT World Ltd
London
October 1989**

E1.4.42

EXECUTIVE SUMMARY

This report describes work done in the period July - September 1989 on a survey of industrial perceptions of the SERC/DTI Transputer Initiative and of the needs of industry from the Initiative. It concludes that the existing awareness objectives of the Initiative should be retained, but that there could be a number of changes in the marketing field which, will, if implemented, enable the Initiative to play a larger part in increasing industrial awareness of advanced information technology in the United Kingdom. Particular importance attaches to recommendations on the positioning of the Initiative in relation to other awareness activities and on the training of Initiative staff.

CONTENTS

Executive Summary

	Paras
Background	1 - 3
Methodology	4 - 10
Perceptions of the Initiative	
Overall Industrial Awareness	11 - 14
Perceptions of those Already Aware	15 - 20
Industrial Needs	
Introduction	21
Awareness	22 - 23
Information	24 - 28
Training (At Centres)	29 - 30
Product Quality and Availability	31 - 34
Technical Advice and Consultancy	35 - 36
Marketing Operations	37 - 44
Conclusions	
Recommendations	

BACKGROUND

1. The work covered by this report has been done by IT World Ltd under a contract placed on behalf of the Science and Engineering Research Council by the Atomic Energy Authority. The contract was dated 3 July, with an initial duration of 10 weeks. An extension of time to 31 October was subsequently agreed to by the Initiative Coordination.

2. IT World have proceeded generally in accordance with the Terms of Reference provided when proposals were originally requested. The defined objectives of the survey were:

To look at the current perceptions in Industry of the Initiative and the six Regional Centres.

To define the future needs of industry from the Initiative, and the Centres in particular, with respect to the balance of the facilities and services the Centres offer and to the most effective ways of spending uncommitted funds through the remainder of the project.

3. The survey was commissioned at a time when relatively little freedom of action to make changes remained: only £120,000, available over 2 years, remained uncommitted. We were not asked to contain our recommendations within this sum, but we have borne in mind that improvements in effectiveness rather than radical change were being sought.

METHODOLOGY

4. Sequence We have done the work in 3 phases:-

Familiarisation with the Initiative and Questionnaire design

Questionnaire administration and analysis

Interviews with selected companies

Development of conclusions and recommendations

At the beginning of the project and at the end of each phase, a meeting was held with the Coordination team. A preliminary report was compiled at the end of September 1989 and submitted to the Coordinator. IT World was invited to present this report to the SERC/DTI committee which provides guidance to the Coordinator on 5 October 1989. This final report contains a small amount of clarification and additional comment relating to issues raised during the discussion on that occasion, but no substantial changes have been made.

5. Initial Familiarisation After an initial meeting with the Deputy Coordinator, we made contact with the Regional Centre managers or administrators, either by telephone or by personal visits. Meetings were subsequently held with two centre directors and with other members of the coordination team. Meanwhile, members of the study team spent a small amount of time reading periodicals and papers and reviewing literature from the Centres and from Suppliers.

6. Questionnaire Design A set of three questionnaires was designed by the study team, and the wording of all the questions was then reviewed by an independent consultant to avoid biased or leading questions. The forms were designed so that separate questions could be addressed to:-

1. General managers responsible for investment decisions
2. Functional managers (eg. IT Director or Technical Director) responsible for computer system implementation
3. Technologists

A note setting out the background to the survey was attached to each form and a covering letter was forwarded with each set. Specimens of these documents are at Annex A.

7. Questionnaire Distribution It was felt to be important that a random sample of companies, including the prescribed numbers of smaller companies, should be approached. The following numbers of companies are included in the terms of reference:-

	Engineering Sector	Non Engineering Service Sector
Large	6	4
Medium	12	8
Small	12	-

We considered that no useful purpose would be served by mailing questionnaires to companies with only limited use of computers, eg those confining their use to word processing or standard accounting packages, but that we should not seek to locate experience in parallel processing technology. We therefore decided to use a standard commercial directory (Kompass UK Volume III Company Information) and a random-number generator program to identify random page numbers.

We then started at the top of each identified page, and telephoned companies in the sequence listed on each page to seek the agreement of an appropriate manager to participation.

As the number of respondents built-up, companies not conforming to the size and manufacturing/service industry profile required were ignored. A total of 138 companies were contacted, and 42 agreed to participate. A list of these companies is at Annex B.

8. Questionnaire Response Questionnaires were sent out to companies who had agreed to participate in mid-August. Initially response was quite rapid, but the total number of respondents has proved disappointing. A summary of all the responses received is at Annex C.
9. Interviews Interviews were arranged with a variety of companies known to be involved with advanced computing. The Terms of Reference envisaged a total of 20 interviews with companies, not more than half of whom were to have been involved with the questionnaires. In the event, taking into account the poor response to questionnaires and the need to find more parallel processing users, a total of 26 companies were interviewed. A summary of the responses is at Annex D. Copies of individual interview reports are being made available separately.
10. Other inputs This report does not include any conclusions or recommendations which did not arise from at least one respondent (questionnaire or interview). However, a relatively small number of respondents has any detailed familiarity with the technology and we have taken into account inputs from opinion-formers, user, consultants, academics and others in weighting our conclusions. We make no claim for statistical validity.

PERCEPTIONS OF THE INITIATIVE

OVERALL INDUSTRIAL AWARENESS

11. There is little awareness of developments in Information Technology in industry generally. A recent study for the DTI of awareness of various aspects of advanced IT topics suggested that in only 8% of companies are their directors aware of parallel processing, with an additional 2% being aware of the commercial problems of INMOS. The Transputer Initiative has to be judged against this background.

12. In the course of the study, IT World have, by questionnaires and interviews, collected information from a group of potential users of increased computing power. This group is smaller than, but comparable with, the group surveyed in the study referred to above. Those companies that participated in the survey recognised their needs for better communications between computers and for more speed, more memory and "better" software, in varying degrees. In this group, awareness of the Transputer Initiative is lower than might have been expected, considering that only businesses expressing an interest in developments in IT were included. An analysis of the entire group of businesses with whom IT World have had contact during this study is:-

	Total Contacted	Total Questionnaire (Responses as at 25/9/89) and Interview Reports	Aware of Initiative	Used Centres
Engineering and Manufacturing Industry Users	111	26	10	3
Service Industry Users	36	12	2	1
Total Users	147	38	12	4
Suppliers	10	10		
Total	157	48		

Awareness in large engineering companies, who do not generally rely on support from initiatives of this type, is higher than amongst the general businesses surveyed.

13. Businesses that have received information on the initiative will be aware of the existence of the regional centres. Utilisation of the centres is perceived to be low by the centres themselves, but against the national background this is not surprising. Factors which affect the use of the centres are discussed below.
14. We were not asked to consider the perceptions of the Initiative amongst Universities and Polytechnics. However many of the research students working in these institutions will be going into industrial jobs after they have obtained higher degrees and they will contribute to industrial awareness.

Awareness of parallel processing, and of the transputer, is widespread in tertiary education and the subject is appearing in science and engineering curricula. Many research administrators and educators believe that more credit should be given to the Initiative, at the least for an increase to the range and depth of research projects, but also in some cases for promotion of involvement of industrial collaborators in the hardware and software industries, for support of technology transfer and for dissemination of information (eg in the Mailshot and in Parallelogram). However, transfer of technology "on the hoof" is more effective at post-research level than in graduate recruitment, and cannot be relied upon to influence R&D programmes (cf the slow industrial take up of UNIX despite its widespread use in universities).

PERCEPTIONS OF THE INITIATIVE BY THOSE ALREADY AWARE

15. The Initiative and the Centres are naturally perceived as being academic, in that they are staffed by SERC or academics and staff of most of the centres work on research projects in academic environments. An additional perception, in some cases, is that the Centre staff are seen as computer scientists with academic objectives rather than being aware of current (post-research) industrial need.
16. The Initiative has not been positioned in relation to other DTI awareness initiatives. The lack of a visible overall programme structure causes confusion, especially in view of a current increase in the range of DTI IT-related awareness activities.
17. Many people in industry would prefer to see money spent on direct support for innovation rather than on promotion or awareness. Government support of innovation inside individual companies has a long history and broad expectations of the role of government are slow to change.

18. IT-suppliers are in general, well aware of parallel processing and the Initiative but still have some reservations. In particular some software suppliers think that the Centres, in developing software solutions, may be competing with them, when they could usefully promote what is already available.

There seems to be some doubt about the provision of proprietary demonstration software for the Centres. We believe that the Initiative coordination team should continually seek new demonstration packages, and arrange if possible for these to be loaned to as many Centres as can use each demonstration. However, we believe that the Centre managers should have wide discretion to borrow, rent or buy (funds permitting) additional demonstration material. In general, we believe that many benefits would flow from closer relations between the Centres and companies in the software industry. Not only would suspicions be dispelled, and some duplication of products be avoided but the smaller software suppliers could benefit from exposure to user views available through the Centres. The Centres could, no doubt, provide some guidance on market needs but more important in the short term, they could influence the quality and product support standards especially in the smaller emerging companies.

19. Success in the activities already going on, which would be enhanced by some of the measures now proposed, will itself lead to the dispelling of doubts about the commercial viability of the centres after 1991/1992 and the influence of the centres will thus be further enhanced.
20. The approach to the titles of the Centres, to the use of special titles for promotional purposes and to corporate image matters is seen as fragmented and confusing.

Factors aggravating this confusion are the coexistence of other DTI-supported centres which are involved in parallel processing and the use of the title "Transputer Centre" by centres outside the Initiative.

INDUSTRIAL NEEDS

INTRODUCTION

21. In the course of this study we have not found any marked mismatch between the activities of the Initiative and the needs of industry. Many of our recommendations are therefore directed to improving the effectiveness of the Initiative as a whole in meeting its original objectives. However, some of the Centres' activities are directed to meeting needs which can be met outside the Initiative and we are suggesting some changes in emphasis.

Industrial needs in relation to a developing area of technology can be characterised as follows:

Awareness

Information on the technology

Specific information on products, including availability and compliance with standards

Opportunities to try out products and assess their performance

Technical Advice and Consultancy

Training for managers and technical staff

We deal with these, individually, below.

The desire of industry for information on, and demonstrations of, "real" applications has repeatedly been articulated during the survey.

The wish for reality embraces a variety of aspects of technology transfer, but information on availability and conformance to standards, and effective product support (including good manuals) are of more and more importance. Interesting and well documented demonstrations are increasingly required.

It is also necessary to engender confidence: there is still a tendency to describe parallel processing as Advanced Technology and there is a fairly widespread feeling that "advanced" means either "unproven" or "very complex" (or both).

This is especially true of small and medium-sized companies, who can often recognise the relevance of a technology to their own business if they see it applied, but who may not have the resources for adequate investigation of a technology that may prove to be inapplicable.

AWARENESS

22. As noted above, general awareness of IT issues amongst industrial managers is low. Responsibility for improving this situation cannot be undertaken by an Initiative which covers a specific technology. A number of government initiatives are being undertaken on IT-related issues. The effectiveness of this group of initiatives would be improved if the structure of an overall programme were published, so that industrial managers can select appropriate events for staff at different levels. Furthermore, awareness of the Transputer Initiative could, given better coordination, be more widely promoted in other initiatives.

23. At the technical management level, except in large companies, overall awareness of the Initiative is also low. We believe that it could be improved by a number of relatively simple measures which would not require large changes in resource allocation. Such measures are discussed in more detail below.

INFORMATION

24. Technical managers involved in parallel processing will be happy to rely on the Initiative for information on products, for diaries of events and for guidance on sources of data that they need.

We believe that the Mailshot has a valuable role to play and that it should be continued in its present low-cost form: some minor improvements in sub-division, pagination etc could be included at very little extra cost. A diary of events would be a popular addition, which could well improve coordination with other awareness activities and reduce clashes of dates eg between events at Centres and Institution events.

25. The inclusion of more reportage and some case study material in the Mailshot is, we understand, already planned and will be well worth while.
26. The Centres have opportunities to distribute product literature to visitors and this activity should be encouraged, provided that care is taken to avoid any impression that products are thereby being endorsed.
27. The Initiative staff take part in exhibitions, and provide a presence at conferences where they distribute information. This activity is appreciated and, in our opinion, is effective, especially if stands can be manned by people who can answer questions.

28. The Initiative takes a leading role in organising an annual international conference and exhibition. We visited the exhibition at Liverpool in September and were impressed by the enthusiasm of most of the exhibitors and the appreciation of the role of the Initiative by the academically-based exhibitors in particular. The range of commercial products is now rapidly increasing and it will be important at future events to recognise that maturing products need rather more elaborate presentation if the impression that they are still in the research phase is to be dispelled.

TRAINING (AT CENTRES)

29. We do not believe that the Centres can usefully be involved in general IT awareness activities which are covered by wider initiatives of the DTI and by the Learned Societies and Professional Institutions. Such general activities will, increasingly, promote awareness of parallel processing amongst general management and corporate planners. We accordingly suggest that the Centres' awareness courses should be specifically targetted at technical management in companies who have technical awareness needs.
30. The need for technical training is not yet very large and we doubt whether many standard courses will be useful. The continued local availability of courses in software methods and languages (eg. Petri-Nets, occam) is obviously important, but each centre should try to adapt the course profile it offers to take into account the known interest of local companies. Courses on Transputer hardware for software providers are likely to be unique to the Initiative, and should certainly be continued.

PRODUCT QUALITY AND AVAILABILITY

31. We assume that the Initiative will maintain a special relationship with Inmos. We would suppose that a prominent central activity will be the continued supply to the Centres of information on Inmos's products and plans.

32. Many users are concerned about the quality of, and product support for, transputer related software products.

We believe that a valuable contribution to the improvement of both these characteristics of products from the smaller suppliers can be made by the Centres. The general interest of industry will be served by encouraging staff of the Centres to comment on these issues - of which they have a unique opportunity to gain experience - as they affect local suppliers and to volunteer advice to suppliers with whom they have local relationships.

33. A particular issue in relation to the take-up of Transputer-based parallel processing is the complexity of the issues facing a company which has a major investment in single-processor versions of numerical analysis software. Many such users would like to be able to "port" them to a Transputer-based machine, so as to take advantage of the increased speed available. DTI is supporting an "Applications Portability" awareness activity which, at least peripherally, addresses this problem. The portability problem may also be addressed elsewhere but the mutual exchange of specific information, especially if case-study material were available, would be advantageous and should be supported by the Initiative.

34. Industrial users definitely need good up-to-date information, especially from INMOS but also from software suppliers, about new product specifications, release dates and distribution arrangements. We have no doubt that the Mailshot can provide a very useful addition to the normal channel of promotion, both by inclusion of copies of announcements and in special cases, by editorial comment.

TECHNICAL ADVICE AND CONSULTANCY, "HANDS-ON" TRIAL FACILITIES

35. Very few companies use these services of the Centres. This may be because larger companies like to be autonomous whilst smaller companies are not yet investing. The Centres do each have a range of specific expertise which can form the basis of a useful consultancy service, provided that:-

35.1 The services available (range and depth) are clearly identified and specifically promoted
and

35.2 The Centres make clear to potential clients whether, or not, relevant applications area knowledge is available.

The small staffs of Centres cannot be expected to have knowledge of all applications of local interest, but they should know how it can conveniently be obtained.

36. There is some concern amongst Initiative staff, not yet widely reflected in industry, that the hardware and software available at the Centres is obsolescent, on account of the rate of technical change. In our opinion, this is an issue that should be closely watched by the coordination team, but we are inclined to the view that very impressive demonstrations can be given on the systems at present available, and that it is not essential that the latest version of machines should be available.

There will be some exceptions on account of the need to accommodate new software. However, it is important that the facilities provided by the available software should be fully representative of current developments. It is to be hoped that suppliers will be willing to update software on view at the Centres without charge, in their own interests. There will, however, probably be some instances where new procurement will be needed from suppliers who are loath to provide free copies of new products. We believe that information on software new products and the revisions should be collated by the coordination team as well as forming a subject of continuing dialogue between Centres and local suppliers, and that the Initiative should not hesitate to use its influence to obtain preferential terms. The staffing of the coordination team should reflect the importance of this area of work.

MARKETING OPERATIONS

37. The Initiative as a whole, and the Centres in particular, are faced, in dealing with industry, with problems of "technology transfer" ie the transfer of knowledge about relevant applications and the transfer of confidence and motivation to put this knowledge to effective use. The means of doing this will be closely comparable with industrial marketing, albeit with less financial content. In what follows we have used the term "marketing" to embrace all aspects of the technology transfer activities of the Initiative other than those which are purely technical.
38. The management of marketing operations, like all other operations, needs to be clearly structured, so that everyone involved understands which decisions are taken by whom. The head of the Initiative is known as the Coordinator; it is clear that he cannot issue direction to the Centres (other than that at RAL) except on matters in which he has total control of funds and resources.

Such instances are limited: and the basic concept is therefore one of management by consent. Such management can be effective, but its practice requires more sensitivity than does a hierarchical structure. Management by consent only works well where there are very good communications and simple rules and procedures accepted by everyone. When some of the responsibility is undertaken on a non-executive or part time basis as with centre directors, even more importance attaches to communications.

39. We received the impression that the division of responsibility for detailed marketing issues, as between the Initiative coordination team and the Centres, is not clear to all staff. Some activities are undertaken centrally (eg the Mailshot, participation in national exhibitions, the support to "community clubs"). Similarly, Centres naturally deal with detailed arrangements for their own courses. However, there are a number of topics on which economy and effectiveness could be improved if actions were taken centrally on the Centres' behalf and others on which there is scope for a steady flow of non-mandatory guidance and information from the coordination team. A marketing survey is not an adequate basis for a set of proposals about divisions of responsibility, but we make some comments below, and have made a small number of specific recommendations.

40. Staff Training

All staff involved in the Initiative have general training needs and these should not be neglected. However, most staff have little knowledge of marketing operations when they join the Initiative and several of the scientific staff are acutely aware of this.

We believe that all staff should attend short courses on the structure and management of industrial business, the problems of small and medium sized firms and the particular problems of introducing new products.

No funds exist for such training of the University-based staff and we believe that their training on these issues, as well as the training of Initiative funded staff, will have to be funded by the Initiative. We have no doubt however, that this expenditure will be well worthwhile. Centre managers and administrators should receive detailed briefing on the mechanics of promotion, including the principles of writing literature, means of printing, sources of address lists and "image" matters generally. It may be that such courses have to be specially arranged, although we believe that much could be done locally if appropriate advice were available. All staff should be briefed on the position of the Initiative within the span of DTI awareness programmes.

41. Identification of the Individuals in the "Target Audience"

This is a key activity for all the Centres. We believe that there should be a standardised approach to an IBM-PC based address list data base, which would be built up locally, but onto which would regularly be copied mailing list information provided by the Initiative central staff in the form of suitable coded files, derived from the Mailshot distribution list and other information provided by SERC. There can be no doubt that correctly addressed personalised mailings are of great importance in awareness campaigns.

42. Corporate Image

We believe that all Initiative promotional material should be clearly recognisable as such.

This would entail; as a minimum, the distribution of standard layout "grids": the economics of central printing of some formats could be examined.

43. Press Relations

The Initiative is well placed to become aware of significant developments. We believe that, occasionally, press briefings should be arranged in London, to attract press coverage of significant events and achievements, ranging from conference reports to project milestones. Recent coverage of university computing in New Scientist or of the NCC 'Impact' initiative in the Financial Times, is an example of the type of coverage we think should be sought. It may be that the Initiative would need to get help from specialist staff or consultants to achieve a higher profile: we believe that this would be worth while.

44. Internal Communications

There can be no doubt that the Initiative staff, centrally and in the Centres, are well placed to judge external awareness and to form opinions on possible improvements in effectiveness. We suggest that feedback from individuals should be encouraged and:-

44.1 An internal Initiative newsletter, of an informal nature, should be produced.

44.2 Meetings of all Centre Directors with their Managers should be held 2 - 3 times per year.

44.3 An overt policy of briefing staff on the commercial prospects of the centres should be adopted.

44.4 Centre managers should be encouraged to have regular meetings of full and part time staff, which Directors should sometimes attend, feedback from which should be passed back to the Coordinator.

CONCLUSIONS

1. Industrial awareness of the Initiative is low, but not lower than can be expected after considering the general awareness of Information Technology issues in UK industry
(paras 11-13, 22,23).
2. Academic awareness of the Initiative is good (para 14).
3. Industrial perception of the relevance of the Regional Centres is (despite improvements in relations between universities and industry) affected by their academic staffing and location. This perception may be modified by improvements in promotion but the basis for it cannot realistically now be changed (para 13, 15-20).
4. Industrial acceptance of the Initiative would be wider if it were clearly positioned within a cohesive awareness programme structure (para 16).
5. There is some suspicion of the Centres amongst suppliers.
(para 18).
6. A uniform corporate image for the Initiative is needed
(paras 19-20, 42).
7. The Mailshot is widely appreciated (paras 24-25).
8. Centres have an important role in the distribution of product and general information (paras 26-27, 34).
9. Participation in exhibitions by the Coordinating Unit is valuable (paras 27-28).
10. The International Conference is widely recognised but this event and the associated exhibition are still seen as research oriented (para 28).

11. Awareness courses are needed for Technical Management. General management awareness is being handled in other initiatives (paras 29-30).
12. Technical training and the provision of consultancy services should continue on the present lines, but better targetting of promotion is essential (paras 30, 41).
13. There is scope for closer relationships between the Centres and software suppliers, especially in bringing influence to bear on standards of quality and product support (paras 31-32).
14. There is some danger that the hardware and software at the Centres will be unacceptably obsolescent. This issue can be monitored by the Coordinating Unit: some expediture may be needed (para 36).
15. Improvements in communications and clarity of procedures throughout the Initiative are needed. There are some weaknesses in staff training at the Centres. Central funding will be needed for some of the training and other improvements needed (paras 37-40).
16. A standard database format for address lists is needed by the Centres, which could be fed with information on floppy discs by the Coordination Unit (para 41).
17. A higher profile is needed in the press especially with the computing correspondents of papers read by managers (para 43).

RECOMMENDATIONS

General

1. The original objectives should be retained.
2. The positioning of the Initiative within the AIT programme of DTI should be clarified and mutual reinforcement sought.
3. Coordination with relevant DTI awareness activities (eg on Applications Portability) should be sought.
4. The corporate image of the Initiative should be strengthened and confusing "brand names" should be avoided.

Activities

5. There should be increased emphasis on awareness courses for Technical Management.
6. The Mailshot should be continued, and its technical content enhanced. Every opportunity should be taken to expand the mailing list.
7. Centres should distribute as much relevant product information as they can.
8. The Coordination Unit should continue to participate in exhibitions, with manned stands whenever possible.
9. The International Conference and Exhibition should be planned to attract industrial attendance in 1990 and thereafter.

10. The Centres should give more emphasis to relationships with suppliers, and especially to influencing quality and product support.
11. The Coordination Unit should monitor technical obsolescence at the Centres, collate and disseminate new product information and negotiate preferential terms with suppliers when appropriate. The complement should reflect the importance of this activity.
12. A higher profile should be sought in the general and semi-technical press.

Training and Operations

13. Steps should be taken to provide some centrally funded basic training on relevant aspects of marketing (eg address location, structure of industry, literature design, event management) for the staff of Centres at all levels.
14. More attention should be given to communication within the Initiative. (An internal newsletter and the integration of meetings with Directors and with Managers are examples of apparently realistic measures).



THE

9000

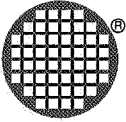
TRANSPUTER

APPLICATIONS

=



INMOS is a member of the SGS-THOMSON Microelectronics Group



Imaging

printers, x-terminals, scanners, image processing systems

Computing

application accelerators, supercomputers, disk arrays, robotics, industrial control

Communications

networking, switching, mobile and satellite communications

For all these applications the T9000 transputer delivers

- exceptional single processor performance
- scalable multiple processor performance
- real-time responsiveness
- ease of use
- low system cost
- fast time to market
- easy upgradability
- extensive software support
- compatibility

The INMOS T9000

The T9000 transputer integrates a 32 bit integer processor, a 64 bit floating point processor, 16 KBytes of cache memory, a dedicated communications processor and four high speed serial communications links on a single chip. It is the first member of a new generation of exceptionally high performance transputers and is binary compatible with the first generation, giving easy upgradability and access to the installed base of transputer applications software.

More power, more performance

The ability to provide more power and more performance while still retaining flexibility and ease of use is the goal of systems designers, OEMs and component manufacturers. Almost all applications have these requirements, and all rely ultimately on the microprocessor to provide them. Some microprocessors force the designer to opt for performance at the expense of ease of use; some provide a low cost solution but are not upgradable; some offer software support but not device compatibility.

The T9000 transputer overcomes all these.

Specifically designed for the embedded systems markets of imaging, communications and computing, the T9000, like first generation transputers, gives not only superlative uniprocessor performance but also easily adaptable multiprocessor capability

It "future proofs" systems by enabling easy upgrades and performance increases by the addition of more transputers.

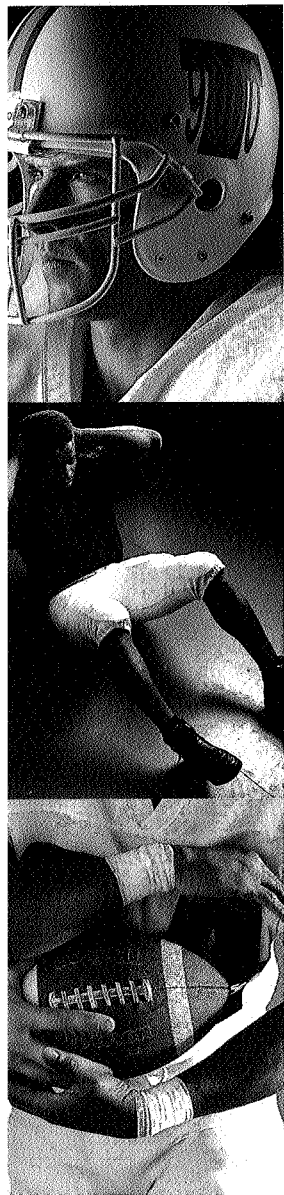
It has immediate access to the well established industry standard software base of the existing transputer range, as well as new software platforms.

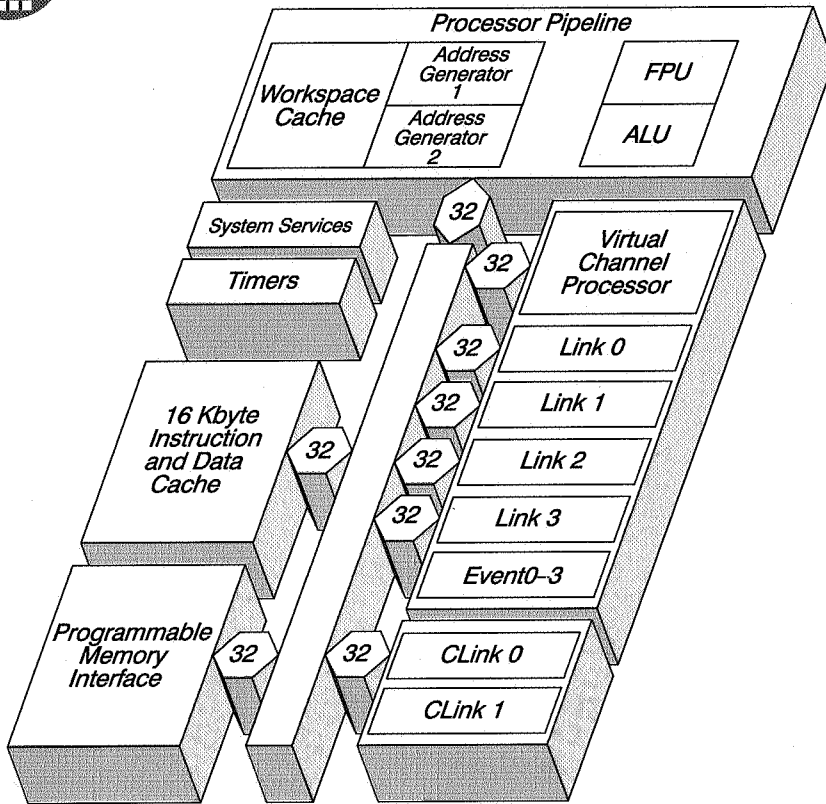
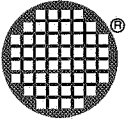
It is easy to use, giving the advantages of fast time to market.

Its high level of integration means low component count, leading to low system cost.

Multiprocessing made simple

As a single processor the T9000 is an exceptional machine for all embedded applications. As the requirements grow the T9000 can also be scaled to a multiprocessor solution using the new high speed links. This is achieved with very little software modification and gives systems designers the flexibility to design scalable products with the confidence that software development costs for upgrades are kept to a bare minimum.





Embedded applications

The T9000 has been specifically designed for embedded applications, which make special demands on a processor in terms of its ability to switch context efficiently when responding to system interrupts or timeslices between tasks.

The T9000 provides direct hardware support for context switching and process scheduling with sub-microsecond response time for multiple level interrupts. Furthermore, its unique virtual channel communications model provides direct hardware support for message passing. The ability to handle a single task error such as overflow without resetting the system is an exclusive feature of the T9000. No other microprocessor manufacturer has been able to achieve these attributes in a single design.

Application areas

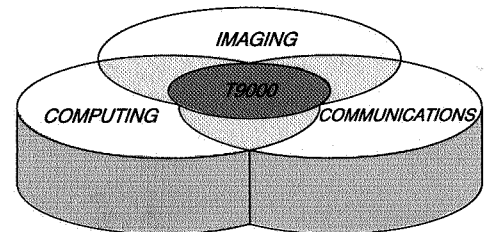
The T9000 has been specifically developed for three main application areas.

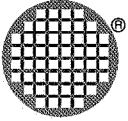
Its 200 MIPS peak execution rate, coupled with its 25 MFLOPS peak performance, make it ideally suited for imaging applications.

The ability to easily build multiprocessor systems enables thousands of MIPS and hundreds of MFLOPS to be employed to achieve massive computational power.

Networking and communications problems are efficiently solved by the T9000's communications capabilities.

- 200 MIPS peak / >70 MIPS sustained
- 25 MFLOPS peak/ >15 MFLOPS sustained
- 80 MBytes/s total bidirectional link bandwidth
- 200 MBytes/s memory interface bandwidth
- Submicrosecond interrupt response
- Per process error handling
- Memory protection
- Virtual channel processor support for message passing
- 5 MHz clock input (50 MHz internal operation)
- Programmable Memory Interface
- Fast single cycle bit and byte manipulation
- Two 32 bit timers on-board





Imaging

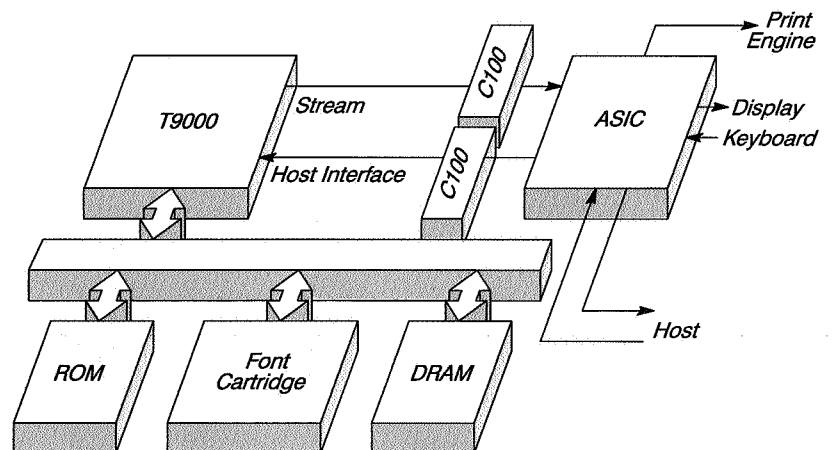
The imaging market covers a broad spectrum of applications ranging from printers to image processing systems. These applications require the generation, manipulation and transmission of image data, high performance processing for tasks such as image compression, and Postscript interpretation. These applications demand efficient communication links for I/O and a high level of system integration for minimal system size and cost. The current range of transputers effectively supports imaging applications with a range of performance options and multiprocessing capabilities. The T9000 offers, in addition to its high performance, single cycle bit and byte manipulation for fast image data handling and a fast 2-D block move function for image movement.

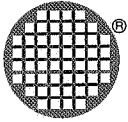
INMOS, together with SGS-THOMSON, is the standard supplier of PC graphics components such as the G171 Colour Look Up Table for IBM PS/2 VGA PCs and is in an ideal position to supply all products for a complete solution. Such imaging products include the G3XX family of Colour Video Controllers (a graphics subsystem on a chip) and the SGS-THOMSON range of image compression and manipulation products: the STIXXXX family of digital signal processors.

Page printers

System cost and software support are key elements in any high volume laser printer design and consequently the choice of the CPU is the focus for laser printer designers. The T9000 has all the necessary characteristics for mid-range and high-performance laser printers not only as a uniprocessing solution but also as a truly scalable element within a multiprocessing system. It provides the raw processing power essential for image manipulation, and has the added benefit of an efficient Postscript interpreter.

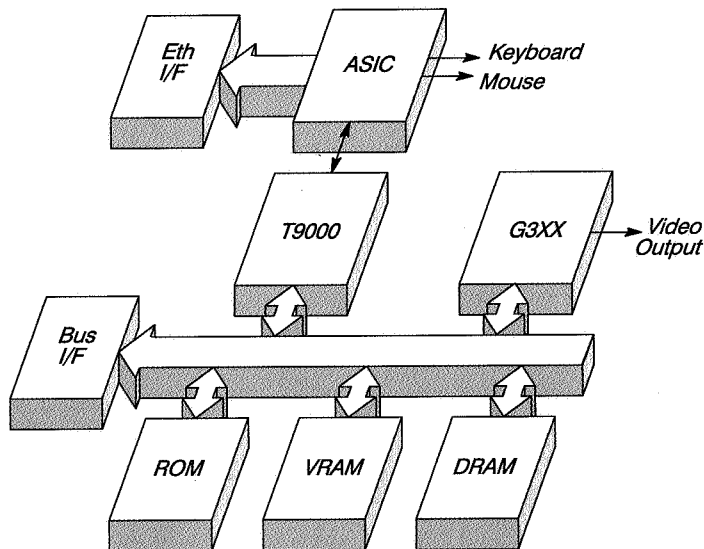
- Postscript support available
- C-Exec real time kernel available
- 200 MIPS peak integer performance
- 25 MFLOPS peak floating point performance
- 10X raster image performance of current single processor laser solutions
- High speed direct memory access (DMA) link technology working concurrently with CPU for image data transfer to main memory, print engine and host interfaces
- Low memory interface costs, direct interface to 8 Mbytes DRAM (including on-chip refresh logic)
- Demonstrator and Application Note available from INMOS





X-Terminals

Having a workstation dedicated to a single user can be an expensive solution. X-Terminals bring the features of a workstation to the desktop: local intelligence, processing power and communications at the price of a standard personal computer. The transputer family offers a variety of solutions from low end monochrome systems using the T4 family to high-end, high-resolution colour systems using the T9000.



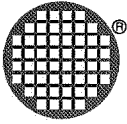
- Industry standard software including full port of X11R4 and TCP/IP
- Scalable performance
- System level solution – T9000 and INMOS IMS G3XX Colour Video Controller
- Fast single cycle bit manipulation for graphics operations
- 2-D block move instruction
- Hardware concurrency to support communications, local computation and display management simultaneously
- Direct support for 8 Mbytes of DRAM (on-chip refresh logic)
- Demonstrator and Application Note available from INMOS

Scanners

Similar in application to laser printers, but with an emphasis on data stream manipulation, scanners present designers with several problems: high speed data stream routing, image compression and optical character recognition. The T9000, unlike most microprocessors, has the processing power to cope with all these areas.

- High performance for optical character recognition and image compression algorithms
- Link concurrency with CPU for simultaneous image manipulation and data transfer
- High integration for low cost and size
- 80 MBytes/s serial data transmission capability



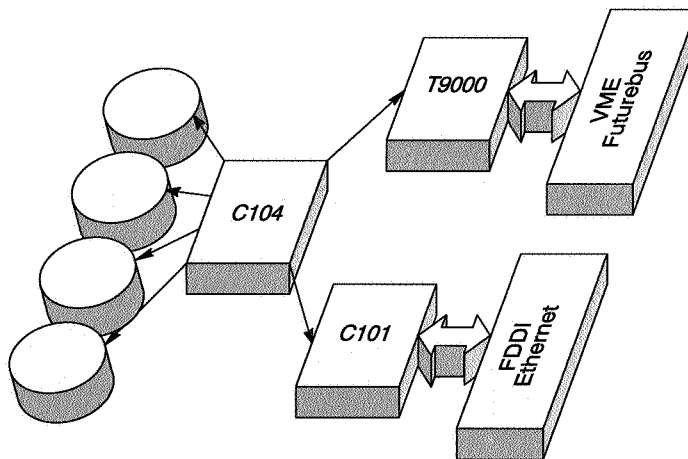


Computing

The T9000 offers many ways of increasing system performance. The T9000's 200 MIPS peak execution rate and 25 MFLOPS peak floating point performance, coupled with the ability to build multiprocessor systems, means that thousands of MIPS and hundreds of MFLOPS can be employed to achieve almost unlimited real-time performance. Alternatively, system performance can be increased by employing the T9000 in subsystems such as disk arrays, application accelerators etc.

Disk arrays

Whatever the power of the central processor in a mainframe or a workstation, a major bottleneck is in the data storage subsystem. System down-time in many cases can be narrowed down to failure of a hard disk drive within the infrastructure. The way ahead lies with an array of disks rather than one single large capacity drive. The speed of data retrieval has become a major issue in disk system design and a fault tolerant architecture has become a necessity. The T9000, along with the rest of the transputer family, is suitable for disk array design in mainframe, workstation or PC.

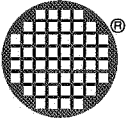


- Link technology operating at 5 times the speed of SCSI
- Fault tolerant design capability without external arbitration logic
- Scalable architecture allows any configuration
- CRC error checking algorithms can be executed as a concurrent process

Application accelerators

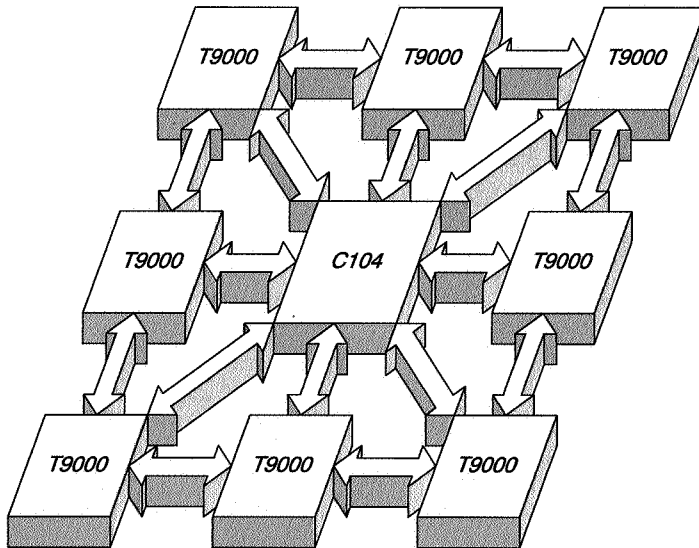
Used as an embedded processor in application accelerators, the transputer provides transparent processing power to overcome the problems of increased performance requirements. From PCs to mainframes there will always be a need for more processing power, regardless of the application running or the number of users. Transputer-based application accelerators speed up dedicated software such as financial modelling packages, relational databases and complex numerical algorithms, leaving the host CPU to concentrate on the system tasks for which it was intended.

- Scalable solutions for performance upgrades to main frames/PCs/workstations etc
- Links to parallel interface chips
- Standard range of motherboards and modules for many hosts
- Driver software available from INMOS



Supercomputers

Natural, scientific, engineering and commercial tasks are inherently parallel. Sequential processors are often unable to perform complex calculations such as finite element analysis, computational fluid dynamics and geographical survey analysis efficiently. The transputer offers almost limitless performance through its modular, scalable architecture and parallel and multiprocessing capabilities. The virtual channel concept of the T9000 and the dynamic configurability of the C104 packet routing switch allow supercomputer designers to incorporate various network topologies — N-cube, hypercube, tree, mesh, ring, toroid etc — within the same system. A key requirement for any large parallel computer is the ability for each individual node to communicate with others. In T9000/C104 networks, wormhole routing techniques ensure that message latency is minimised and transmission is continuous, allowing the construction of large systems with full node connectivity.



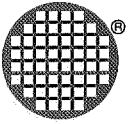
- Exceptional performance — 200 MIPS, 25 MFLOPS (peak)
- Minimal link latency.
- Dynamically configurable network topologies using C104 routing switch
- Modular, scalable, parallel architecture
- Virtual channel message passing capability in hardware

Robotics

From spot-welders to automotive production line installations, real time control of machinery is vital to efficient production. Communications capability is essential in a control system since processing nodes in remote locations often need to relay information about their status to a central control unit. The numerous axes of a robotic arm can now communicate and relay position and attitude data to each other for accurate and fast positioning through the use of INMOS' link technology. The ability to build fault tolerant systems with ease is also important; sub-microsecond response time is crucial for safety-critical systems.

The transputer family provides the essential communications and multiprocessing capability for large and complex control systems. The T9000's unique ability to pre-emptively schedule and reschedule interrupts to the pipeline, and to use the interrupts as outputs, gives flexible real-time responsiveness.

- Multitasking kernels available for real-time control — VRTX and C-Exec
- Sub microsecond pipelined interrupt scheduling
- Integrated Floating Point Unit for accurate positional control
- Ability to mix current and T9000 family members for optimum system efficiency
- Pre-emptive scheduling of interrupts



Communications

The worldwide telecommunications market is going through a period of tremendous change, with new communication standards and technologies creating new growth markets. Digital computer networking and network interconnection, the digitisation of telephone networks, and new markets like cellular radio all place greater reliance on processing performance and software than previous analogue systems.

Internetworking

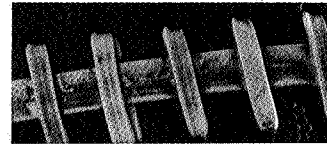
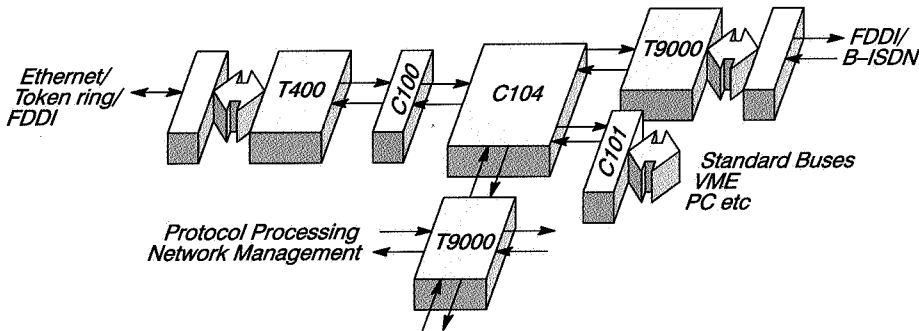
The market for the equipment that interconnects different networks together, bridges, routers, gateways, etc., has experienced rapid growth as corporations seek to unify their networking operations. Increasingly sophisticated protocols and higher data rates, particularly with the 100 Mbit/s FDDI standard, raise the processing performance required significantly. The shared-bus architecture adopted in many of these multi-processing systems becomes a significant bottleneck as data rates increase, resulting in a reduction in performance and network throughput.

Using the T9000, with its high-speed communications capability, for the interface card controllers to token-ring, ethernet, FDDI, etc. and the low latency C104 packet routing switch, bridges, routers and gateways can be constructed using the dynamic message routing architecture which naturally fits the application.

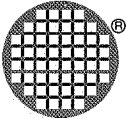
Switching

Telephone switches, from small digital PABXs to large Central Office exchanges, are classic examples of real-time, distributed multiprocessing systems. The message-passing architecture of the transputer family provides an ideal foundation for switch control systems, offering a close match to the message-passing requirements of these applications. A choice of interfacing techniques, via INMOS links or bus-based cards like VME, offers a smooth migration path for the large systems that, of necessity, must evolve. At the same time, the latest generation of transputers and routing chips offers new opportunities for smaller systems to avoid the performance bottlenecks of conventional bus-based systems.

- High speed protocol processing
- Sub micro second message routing latency using wormhole algorithms in hardware
- 32 way virtual channel Packet Routing Switch C104 allows non-directly connected processors to communicate with minimal latency
- High speed interface between exchange and networks using the new 100 MBaud T9000 links



- Single CPU performance up to 200 MIPS for fast, efficient protocol processing
- Scalable, multi-processing architecture supports flexible, modular equipment design
- Serial link speeds up to 100 Mbits/s provide fast, flexible interprocessor communications
- Sophisticated message-passing architecture avoids limitations of bus-based systems

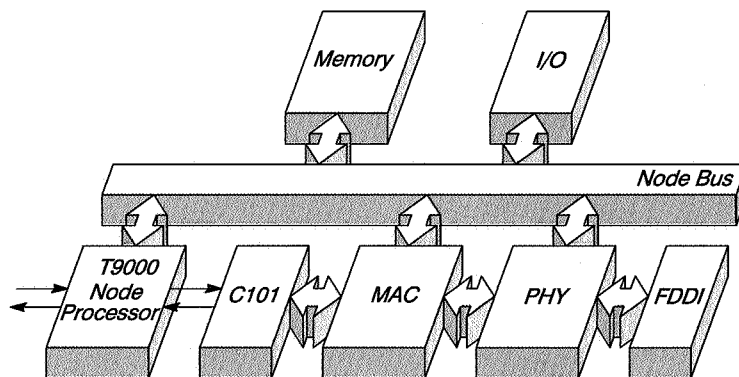


Network interfacing

Building cost-effective interfaces to today's high performance networks makes exceptional demands on the microprocessor at the heart of the interfacing system.

Higher data rates and increasingly sophisticated protocols require considerable processing performance, for today's networks and for emerging communications standards. At the same time, user demand for lower prices places greater pressure on the interface designer to minimise system cost.

Whatever the system, from low speed 64 kbit/sec ISDN through Ethernet to FDDI and Broadband ISDN interfaces, from single processor add-in cards to distributed multiprocessing systems such as telephone switches, the transputer family satisfies all communications systems requirements.



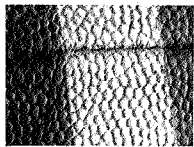
- High performance interfacing to high speed digital networks, e.g. ISDN, FDDI, Ethernet/token ring
- High integration for low system cost
- 200 MBytes/s Programmable Memory Interface to all types of memory with minimal external logic. (No external circuitry for up to 8 Mbyte DRAM)
- VRTX and Chorus (distributed UNIX) software available

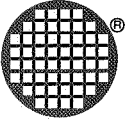
Mobile communications

Cellular radio has seen spectacular success since its introduction in the early 1980s. New digital cellular standards like the Pan-European 'GSM', new systems such as Personal Communications Network (PCN), and the next-generation 'Digital European Cordless Telephones' (DECT) set the scene for further growth.

The massive scale of these new networks and the speed with which they have to be installed make it even more important to keep infrastructure investment as low as possible. With base station and handset processing requirements rapidly increasing as the networks go all-digital, the transputer family offers high performance, and a flexible upgrade path at the lowest possible price.

- Simple multiprocessing for flexible, modular base station design
- Low component cost + high integration = low system cost
- Low cost control of 'RF line cards'
- Base station control and management functions
- High performance protocol processing for call set-up/clearance

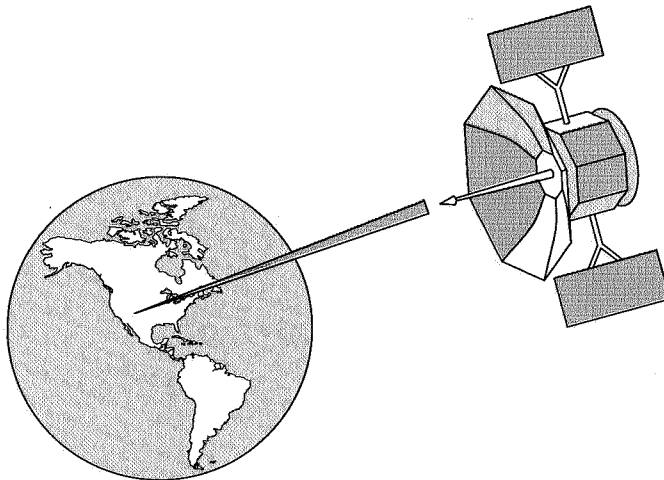




Satellite communications

There is no more demanding environment for any type of system than Space, but here, too, designers are finding the transputer ideal for the on-board processing systems of orbital vehicles and probes. INMOS is working with the European Space Agency (ESA) to produce transputers for use in space.

- High integration/low system cost means space saving designs and low mass
- Good intrinsic radiation tolerance
- Multiprocessing capability means easy to build multiple-redundant fault tolerant systems



Global positioning

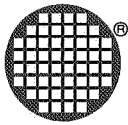
The T9000's fast, autonomous serial links, high performance CPU and low-cost package provide all the elements necessary for building a complete GPS receiver. All signal processing can be performed in software by the CPU, eliminating a costly ASIC, whilst interfacing to the RF front-end is simplified enormously through the use of the serial links.

The range of devices in the transputer family means the designer can easily make the right cost/performance decision. T4/T8 transputers provide an ideal performance level for the commercial GPS 'P'-code products, whereas the T9000 can satisfy the demanding requirements for 'Y'-code systems.

- Economical single processor solution
- Low system cost
- Fast acquisition time
- Low power requirements
- No custom hardware required
- Global positioning software available from INMOS
- High accuracy positional determination
- Powerful enough for both 'P' and 'Y' code implementations

Military

Imaging, communications and control are all applicable to the military market. INMOS' ability to provide high quality Mil-Std-883C compliant products for all these applications is well proven with the current generation of transputers, and the T9000 brings a new level of processing capability to the military segment. The high speed core of the T9000 allows the use of a general purpose processor in applications as diverse as phased array radar and fault tolerant control systems. Its integer and floating point capability mean that it is an ideal vehicle for signal analysis applications such as image processing and data acquisition. The high I/O capability enables the T9000's use in control applications for fast reaction to transducer input for ultimate real-time response. The T9000 is also supported by verified software in the form of ANSI C and the military standard ADA.



Development

System support for the T9000 reflects the requirements of the embedded systems market place. A range of industry standard software support is available for the T9000: C-Executive and VRTX real time kernels, and Chorus, a real-time distributed UNIX operating system.

The T9000 is supported by compilers for all the major industry standard programming languages, including ANSI C, C++, Fortran, ADA and occam.

A complete range of development tools is available, building on INMOS' experience in providing software tools for multiprocessor systems. These include system configuration, loading and interactive windowing debugging capabilities. All run on a wide range of industry standard hosts including IBM and NEC PC, VAX/VMS and SUN 3/4. Full binary compatibility with earlier transputers means that the existing base of transputer applications software can run on the T9000.

This comprehensive range of operating systems, compilers and development tools enables T9000 users to get systems to market in the shortest possible time.

Standard hardware platforms

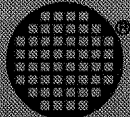
With the previous generation of transputers, INMOS defined the 16 pin TRAnsputer Module (TRAM) format, an ingenious method of interconnecting hardware modules together onto motherboards. The T9000 will be supplied in the original TRAM standard and also in a newly defined TRAM standard to accommodate the dynamic routing capability and also the new higher speed links.

TRAMs allow transputer systems to interface to standard systems such as VAX, SUN 3/4 and a variety of personal computer platforms, enabling systems integrators to build large and complex systems completely from a standard "off the shelf" product portfolio.

The INMOS T9000 transputer excels in real-time embedded applications, providing:

- **superlative performance**
- **real time responsiveness**
- **low system cost**
- **ease of design**
- **fast time to market**
- **upgradability**
- **standard software availability**
- **compatibility**

INMOS is a member of the SGS-THOMSON Microelectronics Group, and supplies high performance transputers, systems products and colour graphics devices worldwide. The company has sales offices throughout the world, and a network of experienced Field Applications Engineers to assist with design-in of the T9000. For further details please contact your local SGS-THOMSON Microelectronics sales office.



SALES OFFICES

EUROPE

Denmark

2730 HERLEV
Herlev Torv, 4
Tel. (45-42) 64.85.33
Telex: 35411
Telefax: (45-42) 948694

Finland

LOHJA SF-06150
Karttulankatu, 2
Tel. 12.155.11
Telefax: 12.155.66

France

94253 GENTILLY Cedex
7, Avenue Gallieni - BP 93
Tel. (33-1) 47.40.75.75
Telex: 632570 STMHQ
Telefax: (33-1) 47.40.79.10

67000 STRASBOURG

20, Place des Halles
Tel. (33) 89.75.50.66
Telex: 870001P
Telefax: (33) 88.22.29.32

Germany

6000 FRANKFURT
Guldenstrasse, 322
Tel. (49-69) 237492
Telex: 176957 889
Telefax: (49-69) 231957
Telefax: 6997689-STVBP

6011 GRASBRUNN

Britanischer Ring, 4
Neukönig Technopark
Tel. (49-69) 46005-0
Telex: 528211
Telefax: (49-69) 4605454
Telefax: 697107-STDISTR

3800 HANNOVER 1

Eckenerstrasse, 5
Tel. (49-511) 634191
Telex: 175118418
Telefax: (49-511) 633552
Telefax: 5118418 csf/bh

65000 NÜRNBERG 20

Erlangerstrasse, 72
Tel. (49-911) 59893-0
Telex: 626243
Telefax: (49-911) 5980701

5200 SIEGBURG

Frankfurter Str. 22a
Tel. (49-2241) 660 84-86
Telex: 889510
Telefax: (49-2241) 67584

7000 STUTTGART

Obere Kirchhaldenweg, 135
Tel. (49-711) 692041
Telex: 721718
Telefax: (49-711) 691408

Italy

20090 ASSAGO (MI)
V.le Milanotti - Strada 4 -
Palazzo A4/A
Tel. (39-2) 89213.1 (10 lines)
Telex: 330131 - 330141 SGSAGR
Telefax: (39-2) 8250449

40033 CASALECCHIO DI RENO (BO)

Via R. Fucini, 12
Tel. (39-51) 591914
Telex: 512442
Telefax: (39-51) 591305

00161 ROMA

Via A. Torlonia, 15
Tel. (39-6) 8449341
Telex: 020653 SGSATE I
Telefax: (39-6) 8444474

Netherlands

5652 AR EINDHOVEN

Meenenakkerweg, 1
Tel. (31-40) 550015
Telex: 51186
Telefax: (31-40) 526835

Spain

08021 BARCELONA
Calle Platan, 6 4th Floor, 5th Door
Tel. (34-3) 4143300 - 4143361
Telefax: (34-3) 2021461

28027 MADRID

Calle Albacete, 5
Tel. (34-1) 4051615
Telex: 27060 TCCEE
Telefax: (34-1) 4031134

Sweden

S-16421 KISTA
Borgenjordsgatan, 13 - Box 1094
Tel. (46-8) 7939220
Telex: 12076 THSWS
Telefax: (46-8) 7504950

Switzerland

1218 GRAND-SACONNEX (GENEVA)

Chemin Francois-Lehmann 18/A
Tel. (41-22) 7966462
Telex: 415493 STM CH
Telefax: (41-22) 7994889

United Kingdom and Eire

MARLOW, BUCKS SL7 1YL

Plantar House, Parkway
Globe Park
Tel. (44-628) 890900
Telex: 847458
Telefax: (44-628) 890391

AMERICAS

Brazil

054135 SAO PAULO
R. Henrique Schaumann 286-CJ33
Tel. (55-11) 883-5455
Telex: (391) 11-37988 "UMBR BR"
Telefax: 11-551-128-22367

Canada

NEPEAN, ONTARIO

301, Moodie Drive
Suite 307
Tel. (1)-(613) 8299944
Telefax: (1)-(613) 8298694

USA

NORTH & SOUTH AMERICAN

MARKETING HEADQUARTERS
1000, East Bell Road
Phoenix, AZ 85622
(1)-(602) 867-6100

SALES COVERAGE BY STATE

ALABAMA

Huntsville - (205) 533-5995

ARIZONA

Phoenix - (602) 867-6340

CALIFORNIA

Santa Ana - (714) 957-9018
San Jose - (408) 452-8585

COLORADO

Boulder (303) 449-9000

ILLINOIS

Schaumburg - (708) 517-1890

INDIANA

Kokomo - (317) 459-4700

MASSACHUSETTS

Lincoln - (617) 259-0300

NEW JERSEY

Voortees - (609) 772-8222

NEW YORK

Poughkeepsie - (914) 454-8813

NORTH CAROLINA

Raleigh - (919) 787-8555

TEXAS

Carrollton - (214) 460-8844

ASIA/PACIFIC

Australia

NGW 2027 EDGECLIFF
Suite 211, Edgecliff Centre
203-233, New South Head Road
Tel. (61-2) 327.39.22
Telex: 071 125911 TCAUS
Telefax: (61-2) 327.61.76

Hong Kong

WANCHAI

22nd Floor - Hopewell Centre
183, Queen's Road East
Tel. (852-3) 8615788
Telex: 60555 ESGIES HX
Telefax: (852-5) 8656589

India

NEW DELHI 110001

Liaison Office
82, Upper Ground Floor
World Trade Centre
Barakhamba Lane
Tel. 3715191
Telex: 031-66816 STMI IN
Telefax: 3715192

Korea

SEOUL 121

8th Floor Shinwon Building
623-14, Yulsan-Dong
Kang-Nam-Gu
Tel. (82-2) 553-0399
Telex: SGSKOR K29988
Telefax: (82-2) 552-1051

Malaysia

PULAU PINANG 10400

4th Floor, Suite 4-03
Bangunan FOP, 1230 Jalan Anson
Tel. (04) 379735
Telefax: (04) 379818

Singapore

SINGAPORE 2056

28 Ang Mo Kio - Industrial Park, 2
Tel. (65) 48214 11
Telex: RS 55201 ESGIES
Telefax: (65) 4820240

Taiwan

TAIPEI

12th Floor
571, Tun Hua South Road
Tel. (886-2) 755-4111
Telex: 19310 ESGIE TW
Telefax: (886-2) 755-4008

JAPAN

TOKYO 108

Nisseki Takanawa Bld. 4F
2-16-10 Takanawa
Minato-ku
Tel. (81-3) 3280-4125
Telefax: (81-3) 3280-4131

Copyright © INMOS Limited 1991

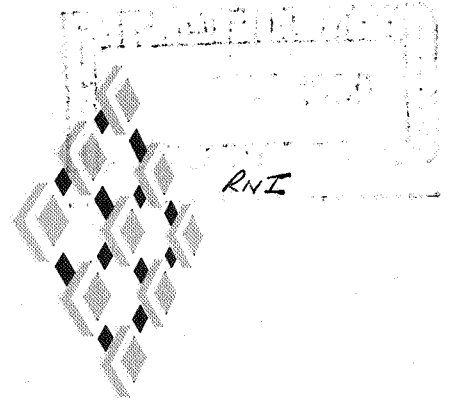
INMOS reserves the right to make changes in specifications at any time and without notice. The information furnished by INMOS in this publication is believed to be accurate; however no responsibility is assumed for its use, nor for any infringement of patents or other rights of third parties resulting from its use. No licence is granted under any patents, trademarks or other rights of INMOS.

INMOS, IMS and OCCAM are trademarks of INMOS Limited.

SGS-THOMSON
MICROELECTRONICS

INMOS is a member of the SGS-THOMSON Microelectronics Group.

March 1991



Transtech Devices Ltd.
Unit 17,
Wye Industrial Estate,
London Road,
High Wycombe,
Bucks.
HP11 1LH
Tel: (0494) 464303
Fax: (0494) 463686
Telex: 838844

Ref: BS/KJV 378Q

2nd August 1990

Mr David Mercer
Commercial Director
EPCC
Edinburgh University Computing Centre
James Clerk Maxwell Building
The King's Buildings
Mayfield Road
Edinburgh
EH9 3JZ

Commercial - In Confidence

Dear David

I would like to thank you for inviting Transtech to present its technology to your organisation. We came back with a warmer feeling and have now taken steps in defining a number of task packages.

The implementation on the Silicon graphics of GENESYS is currently in progress, however for your requirement we have a very high performance implementation in place which will derive Occam programme on the transputer communicating to Fortran-77 or 'C' programs on the IRIS. I will be contacting Mike Norman with reference to defining a schedule for the perceived project.

I have included two quotations for the 64 node i860 requirement that you currently have. Both configurations offer 16 MBytes of memory on each of the nodes, the GENESYS operating system, the i860 Fortran-77 compiler and standalone server module with SunOS and NFS. The first quote is based on the High Speed Link (HSL) product

Cont../2

and the other is based on the i860 TRAM module. Both meet your requirements however the HSL based configuration caters for the future by providing the 200 MByte/s point to point communication.

If you wish to discuss these quotes further then please do not hesitate to contact me.

Yours faithfully

Bluepinder P. Sipl

Bee Singh
AREA SALES MANAGER

cc: Professor David Wallace
Professor Roland Ibbett
Mike Norman
Richard Kenway
R Tremayne-Smith

QUOTE 1 - 64 i860 MBYTE NODES WITH HSL

<u>QTY</u>	<u>PART NO</u>	<u>DESCRIPTION</u>	<u>£</u> <u>UNIT</u> <u>COST</u>	<u>£</u> <u>TOTAL</u> <u>COST</u>
2	TRANS-HSL-40	40 slot HSL module chassis	10,000	20,000
1	TRANS-20	20 slot chassis, 16 MIPS SPARC, 1GByte disk storage, Ethernet, NFS, SunOS, RS232, 1/4" tape	25,000	25,000
64	THSL16-40	40MHz i860, 16MByte HSL node	12,890	824,960
1	MCP1000	Motherboard	4,500	4,500
1	MCS0320-1	FORTTRAN-77 for i860	2,500	2,500
1	MCS0300-1	GENESYS II	3,750	3,750
1	MCS0380-1	TDB Parallel debugger	1,000	1,000
1	Del & Inst	UK Delivery and Installation	2,000	2,000
		PRICE (EXCL VAT)	883,710	
		DISCOUNT (30%)	265,113	
		<u>TOTAL DISCOUNTED PRICE</u>	<u>618,597</u>	

QUOTE 2- 64 i860 MBYTE NODES IN TRAM FORMAT

<u>QTY</u>	<u>PART NO</u>	<u>DESCRIPTION</u>	<u>£</u> <u>UNIT</u> <u>COST</u>	<u>£</u> <u>TOTAL</u> <u>COST</u>
1	TRANS-20	20 slot chassis, 16 MIPS SPARC, 1GByte disk storage, Ethernet, NFS, SunOS, RS232, 1/4" tape	25,000	25,000
16	MCP1000	32 TRAM slot motherboard	4,500	72,000
64	TTM100-16-1	40MHz i860, 16MByte	9,995	639,680
1	MCS0320-1	FORTTRAN-77 for i860	2,500	2,500
1	MCS0300-1	GENESYS II	3,750	3,750
1	MCS0380-1	TDB Parallel debugger	1,000	1,000
1	Del & Inst	UK Delivery and Installation	2,000	2,000
		PRICE (EXCL VAT)	745,930	
		DISCOUNT (30%)	223,779	
		<u>TOTAL DISCOUNTED PRICE</u>	<u>522,151</u>	

W → MAN, RM, ROK, DBM. From OUG Newsletter July 90
only this is now public domain...

Nº 13 July 1990

83

TRANSTECH NEW PRODUCTS

TTM100 Intel i860 TRAM

- ▷ Intel i860 40 MHz 64-bit Microprocessor
- ▷ IMS T805 floating point transputer
- ▷ 5 to 20 Mbytes of fast DRAM
- ▷ Sub-system control of reset, analyse and error
- ▷ Communicates via 4 transputer serial links
- ▷ Industry standard size 6 TRAM format
- ▷ Software drivers and maths library support

The Transtech TTM100 contains an INTEL i860 64-bit microprocessor, an IMS T805 floating point transputer, and 5 to 20 Mbytes of fast DRAM. The T805 is configured with 1 or 4 Mbytes of local memory, and shares the other 4 or 16 Mbytes with the i860.

The interface between the i860 and the shared memory system has been optimised to give the i860 zero wait state access for page coherent memory cycles, such as cache fill and cache flush operations. The i860 busLock function is supported for operating system and other special non-divisible memory cycles.

The inclusion of local RAM for the transputer allows both the i860 and transputer to operate concurrently. A busLock mechanism is included in the transputer shared memory interface to optimise block move operations between local and shared memory.

Synchronisation of the transputer and i860 is achieved by a dual event mechanism. This allows either processor to interrupt the other. In this way, either processor can assume the rôle of system master.

The TTM100 returns the performance of up to twenty-four 30 MHz IMS T805's, using the i860 which is capable of 80 MFLOPS (peak single precision), 60 MFLOPS (peak double precision) and 85K Dhrystones.

The TTM100 is supplied with drivers for use with the Occam TDS, Toolset and 3L compilers, together with an array of over 200 vector library routines optimised to take full advantage of the power of the i860.

TTM32/34/38 high performance T801 TRAMs

- ▷ IMS T801 floating point transputer
- ▷ 2, 4 or 8 Mbytes of fast 2 cycle page mode DRAM
- ▷ 32 kbytes of 2 cycle SRAM
- ▷ Sub-system control of reset, analyse and error
- ▷ Communicates via 4 transputer serial links
- ▷ Industry standard size 2 TRAM format

The Transtech TTM32, 34 and 38 consist of an IMS T801 floating point transputer, 2, 4 or 8 Mbytes of fast 2 cycle page mode DRAM and 32 kbytes of 2 cycle SRAM. They have a sub-system port to control reset, error and analyse.

Fast access to the RAM is obtained by doing a very quick address comparison on the present DRAM row address and the last DRAM row address, if they are the