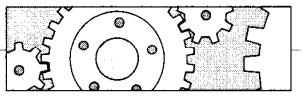


Transputers—Past, Present, and Future

High-performance enhancements in transputers signal a trend toward general-purpose computing. We present the progress, products, and results of ongoing work in transputer development.

Colin Whitby-Strevens

Inmos Limited

A transputer transformation is underway. Exploiting the multiprocessing potential of the transputer, ESPRIT projects continue to develop higher performance, lower cost parallel processing computers. In advancing the transputer toward general-purpose computing, Inmos Limited is also improving the transputer's suitability in embedded systems with the upcoming release of new products.

The first transputer emerged at a time when very large scale integration (VLSI) technology permitted a combination of a small, fast processor (20,000 transistors) and local memory and communications facilities on silicon. At the same time, the effective exploitation of VLSI technology also resulted in the development of reduced instruction-set computing (RISC) processors. Both the transputer and conventional RISC processors reevaluate architecture trade-offs in the context of VLSI capabilities. However, developers of the transputer created a design at the instruction-set level to support multiprocessing across a number of transputers and within a single transputer without the overheads usually associated with complex runtime software.

In embedded systems, the transputer application designer directly controls the transputer hardware without the need for a resident operating system or runtime kernel. Applications for transputers include office systems (fax, videophones, laser printers, terminals), digital telecommunications, military systems and handheld satellite navigation systems (see box on sample applications), industrial control systems, and music synthesizers.

Initial promotion of the transputer focused on its use as a general-purpose component for special-purpose systems. A number of applications—particularly in graphics and image processing—clearly required high-performance, floating-point operations. Inmos achieved this capability very effectively by adding a floating-point unit within the transputer chip (in contrast to the conventional, but more cumbersome, coprocessor approach). This advancement opened up the exciting prospect of constructing parallel processing machines using arrays of transputers to produce supercomputer-level performance.

Applications requirements for simulation and modeling (for example, quantum chromodynamics and fluid-flow analysis) provided the incentive to establish the ESPRIT Supernode

0272-1732/90/1200-0016$01.00 © 1990 IEEE

Two sample applications

The transputer's suitability to a wide range of embedded systems is illustrated by considering applications at both extremes. The first application—a hand-held navigational system—uses just one transputer to address such issues as high performance, low cost, and low power. The second application—a long-range, three-dimensional radar system—uses up to 4,000 transputers to provide supercomputer levels of performance, but in a real-time application.

Hand-held satellite navigation system

Some 18 satellites, in six orbits, operate as part of a Global Positioning System. Each satellite continuously transmits its position in an encoded form, and by listening to four satellites one can calculate the position (in three dimensions) of the receiving station.

The traditional approach involves the use of complex dedicated signal processing hardware. However, the Gypsy hand-held receiver, developed by Columbus Positioning Limited, performs the complex signal processing and mathematics required in one transputer, which also services a keyboard and controls a display (see Figure A).

A large, developing market for the system includes marine navigation, transport control systems (reporting delivery truck positions), and automotive applications (dashboard navigation systems).

The advantages of the transputer-based solution over the traditional approach include relatively fast acquisition time (which also saves battery power), the elimination of custom logic for signal processing, the minimal amount of glue logic required, the low specification required on radio oscillators (transputer software performs frequency compensation), and quick implementation of future product changes.

Figure A. Gypsy hand held receiver.

Long-range radar system

The Martello long-range, three-dimensional radar system under construction by Marconi Radar Systems uses up to 4,000 transputers for signal processing of the radar returns.[1] The transputers combine to form a very fast parallel computer that accepts digitized radar returns at its input and defines the range, bearing, and height of every target at its output. (Some environmental information is also added to the output.) The processing load amounts to roughly three billion operations per second through each radar channel.

The alternative to transputers—hardwiring—is inherently expensive (previous Martello systems used about 50 different board types). With radar technology now moving so fast, hardwiring is doubly expensive because boards are quickly rendered out of date. By contrast, in the transputer system, software performs the entire signal processing task, and the same basic computer remains usable throughout a whole generation of radars.

The signal processing algorithms take place in real-time by using pipeline parallelism to spread the parts of the algorithms across an array, with computation overlapping communications (see Figure B on the next page). Simulation devised the optimum map, making it necessary to build only a small part of an array to prove the concept. In an array node of 50 transputers, housekeeping uses two transputers, while the rest are available for data processing.

continued on p. 18

Two sample applications

continued from p. 17

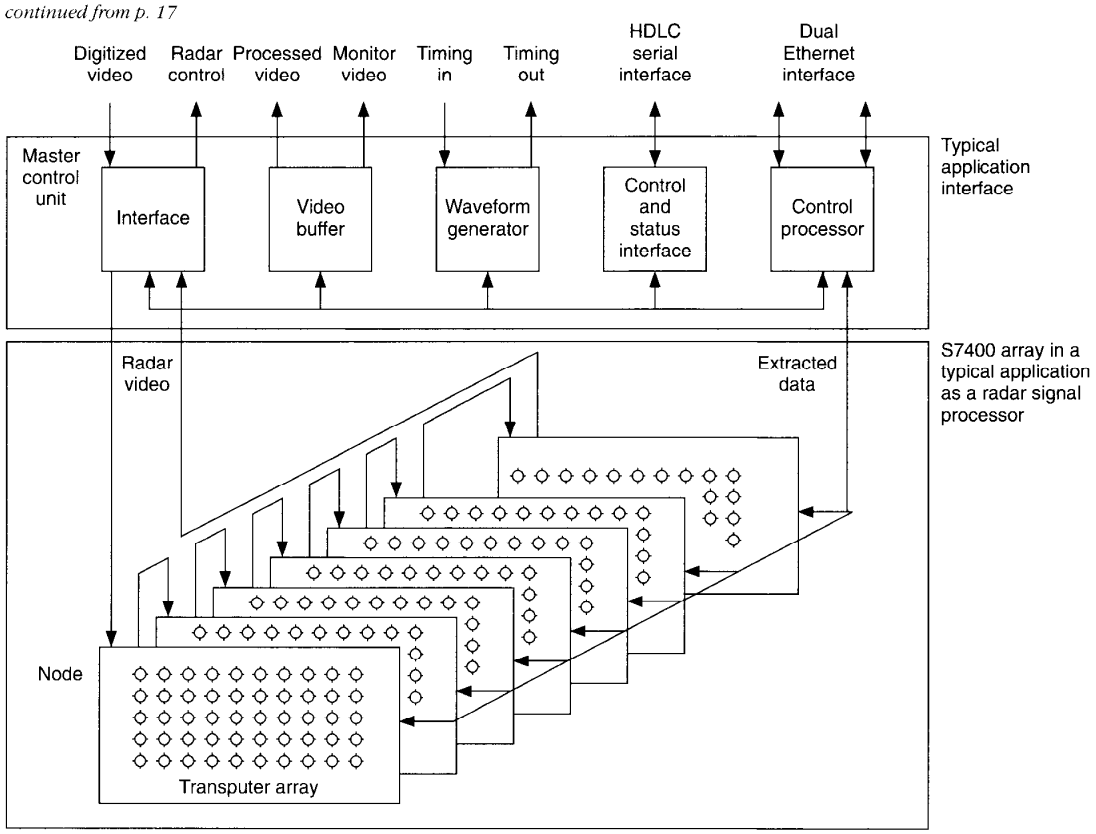


Figure B. Diagram of Martello radar system architecture.

Geometric parallelism achieves the processing rate required for the Martello radar. The radar coverage breaks into range bands, and a particular array node directs all samples from a given band. Each array node executes an identical program, but on different data. The scalable performance of the radar is proportional to the number of array nodes.

The vast number of processors in the Martello computer are16-bit T222 transputers, with some 32-bit T425 transputers. A follow-on project proposes to construct a multifunctional radar for the Italian navy using floating-point transputers. However, no fundamental change to the architecture is necessary to take advantage of new generations of transputers.

project (see ESPRIT Supernode box). This project led to the development of many new commercial hardware and software products, including the Inmos T800 transputer, the Parsys Supernode machine, the Telmat T-node machine, and the N.A. Software Limited's parallel processing libraries for Fortran numerical routines. More significantly, the project contributed greatly in establishing the need for reconfigurable systems and in creating paradigms for parallel programming.

The personal workstation also opened up another market for the transputer, since it provides the basis for workstation accelerators. More than 50 companies now market

The ESPRIT Supernode Project

Over about a four-year period (Dec. 1985-Nov. 1989), the Supernode Project aimed to create a low-cost, high-performance computer system from transputers. This project incorporated the development of the T800 floating-point transputer[2] and the Supernode RTP (Reconfigurable Transputer Processor).

One Supernode consists of 16 transputers connected via a crossbar switch (also developed during the project). Using crossbar switches, multiple Supernodes combine to provide systems of up to about 1,000 transputers with no limitations on topology (except those implied by the four links on each transputer). Constructing larger systems remains possible with minor constraints on topology.

The wide range of applications demonstrated on the Supernode included finite-element analysis, logic simulation, luminosity simulation, and real-time image analysis. Development of parallel algorithms and parallel numerical libraries also occurred.

The main results demonstrated the feasibility of the system, and the surprising ease in programming applications to take advantage of concurrency. A large number of European research projects base their development of parallel computing techniques on the now commercially exploited Supernode.

Current ESPRIT projects

The Supernode 2 project studies software issues—particularly those concerned with advanced operating systems—in the context of the Supernode machine. Such an operating system converts the machine from one that runs one application into a more general-purpose facility.

The two-year PUMA (Parallel Universal Message passing Architecture) project explores the possibilities offered by the new virtual communications architecture (see the "The next-generation transputer" section in the main article). Besides studying the performance offered by various topologies and the implications for computer architecture, the project examines high-level models of parallelism and their implementation using a virtual message-passing system.

The three-year GPMIMD (General Purpose, Multiple-Instruction, Multiple-Data) project aims to develop a standard European MIMD architecture based on H1 transputers and virtual communications. Four main European transputer-supercomputer manufacturers—Meiko, Parsys, Parsytec, and Telmat—currently work as key collaborators on the project, which is led by Inmos.

The OMI MAP (Open Microsystems Initiative Microprocessor Architecture Project) forms the core of the European-led Open Microsystems Initiative. It aims to define the architecture of a new microprocessor family designed to exploit the VLSI capabilities anticipated in the latter half of the 1990s. Collaborators include Bull, Inmos, Olivetti, Siemens, and Thomson.

Several other projects exploit the transputer architecture for specific application areas. For example, Padmavati (Parallel Associative Development Machine as a Vehicle for Artificial Intelligence) uses transputers and associative memory for symbolic processing.

accelerator cards, offering a range of hardware and software capabilities. Initially these cards targeted the developing transputer embedded systems market. However, as application software for the transputer grew, the marketplace widened—first to engineering design workstations and more recently to commercial and financial workstations. Several workstations based entirely on transputers are now available.

ESPRIT projects continue to attack the difficult problems of providing very high performance systems (see box for current projects). The original issues concerned the feasibility of constructing such systems (possibly containing thousands of processors) and determining the methods of programming individual applications. The Supernode project developed a machine that could effectively execute a wide range of applications, and in many cases nearly achieved the theoretical maximum performance from the machine.

Current ESPRIT project issues involve "scalability," "portability," and programming ease. Scalability refers to the ease of achieving increased performance from an application by using more processors. Portability relates to the ease of transferring programs between parallel machines of different architectures. The term "general-purpose" summarizes these issues of parallel computing. A general-purpose parallel computer executes a range of applications without concern for the number of processors employed or the topology in which they are connected. Indeed, whether the underlying hardware architecture is based on message passing (such as transputer-based and hypercube-style systems) or shared memory (such as Sequent's Balance system or Cedar systems) is of no concern to the programmer.

The expertise derived from ESPRIT and Supernode projects continues to spin off other applications. Migration of processing techniques and applications from general-purpose computers into special-purpose systems signifies a definite computing trend. For example, high-performance laser printers incorporating Postscript processing and intelligent terminals

continued on p. 76

Transputers

continued from p. 19

supporting windowing systems enjoy increasing popularity. In both cases, the workstation formerly carried out the corresponding processing. Similarly, one expects techniques for high-performance systems developed on machines such as Supernode to migrate into next-generation embedded systems.

Transputer architecture

The development of transputer architecture involved four main objectives:

- it created a commercial product range that set new standards in ease of programming and engineering;
- it provided maximum performance to the user;
- it allowed the exploitation of future developments in VLSI technology within a compatible family; and
- it created a programmable component for building systems with large numbers of concurrent computing components.

A transputer contains a processor, memory, and a number of standard point-to-point communications links—all integrated into one silicon chip (as shown in Figure 1). An external memory interface extends the on-chip memory. When appropriate, transputers also incorporate special-purpose processing and/or interfacing capabilities. Separating the external memory interface (for local memory) from the communications optimizes performance and minimizes contention.

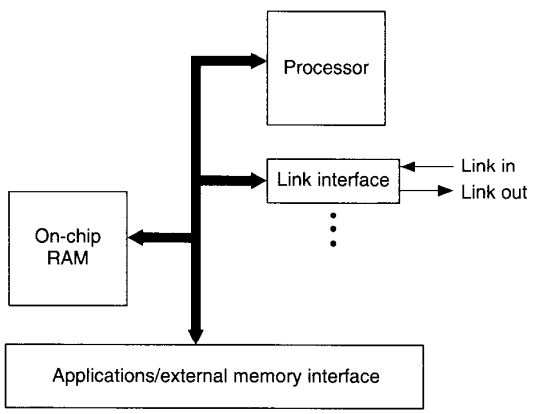


Figure 1. Processing and interfaces in transputer architecture.

A system is constructed from one or more transputers operating concurrently and communicating through standard links. The programming language Occam (see box) formalizes the computational model. Occam describes a system as a collection of processes and communications that operate concurrently and communicate through channels.

Transputer processing

The transputer directly implements the Occam model of concurrency. A hardware scheduler allows any number of Occam processes to share a single processor, and transputer instructions implement Occam message passing. An application designer can configure a collection of processes ready for execution on a network of transputers. Each transputer executes a component process and transputer links implement Occam channels.

Both internal and external communications use the same instructions, allowing for Occam program reconfiguration (such as using a different processes-to-processors allocation) without recompilation. In particular, an application designer can configure an Occam program to execute on a small number of transputers for low cost or on a larger number of transputers for high performance.

The transputer processor supports fast interrupt response by providing two levels of priority. (Typically, the interrupt response is less than 1 microsecond on a 20-MHz clock transputer; worst case, it is less than 4 microseconds). Using the ALT construct (a key word in Occam meaning alternative), a high priority process waits for the first of several inputs to become ready. It then executes the specific piece of code to respond to the particular interrupt.

The processor treats access to a timer as an input. In a delayed input, the process waits until the timer reaches an appropriate value. The processor supports an arbitrary number of timer inputs. A programmer can also use a timer input within an ALT construct as a time-out on a communication.

Developing the transputer instruction set involved five design objectives:

- to implement Occam effectively, so that high-level language usage results in the effective use of silicon capability, and that highly concurrent programs execute with minimum overheads;
- to implement Occam simply and directly to faciliate easy, straightforward program compilation, and to ensure that lower level programming is unnecessary;
- to provide word-length independence, so that a program executes using processors of different word lengths without recompilation;
- to provide position independence, so that programs and workspaces are allocated anywhere in memory after recompilation; and

continued on p. 78

Basic Occam concepts

The Occam language[3] programs concurrent, distributed systems. The word *distributed* emphasizes the unsuitability of previous languages in this area. Occam describes a system as a collection of concurrent processes that communicate with each other and with peripheral devices through channels. Concurrent processes do not communicate via shared variables; thus Occam is particularly suitable for programming systems with no memory sharing between processors.

Occam provides 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

Occam provides constructs that combine primitive processes:

SEQ	Components execute one after another (sequential)
PAR	Components execute together (parallel)
ALT	First ready component executes (alternative)

The language also provides IF and WHILE constructs.

A construct is itself a process and it may be used as a component of another construct. Occam syntax uses indentation to indicate program structure.

A programmer writes parallel programs by using channels, inputs, and outputs combined in parallel and alternative constructs. Each Occam channel provides a communication path between two processes. Communication is synchronized and takes place when both the inputting and the outputting processes are ready. Data to be communicated is copied from the outputting process to the inputting process, and both processes continue.

An ALT process waits for input from any one of a number of channels. ALT takes input from the first to be used for output by another process.

Occam provides a replicated constructor. For example

```
SEQ i = base FOR count
  a[i] := i
```

implements as a loop and is equivalent to

```
SEQ
  a[base]                 := base
  a[base + 1]             := base + 1
  . . . . . .
  a[base + count - 1]     := base + count - 1
```

Replication used with PAR provides arrays of similar processes. ALT and IF can also be replicated.

Using a construct as a component in another construct makes it possible to design a system as a set of nested processes (for example, by using PAR within PAR as shown in Figure C). The messages input and output on the channels of a process fully specify the process, and this completely hides its internal structure from the outside world.

Internally, the programmer can structure the process itself as a set of nested processes. At any level of design, the designer works with a small and manageable set of processes.

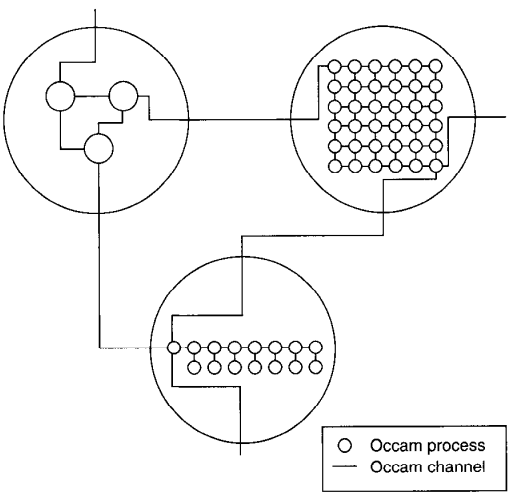


Figure C. Design of nested Occam language processes.

- to provide low-latency response to communications with external devices.

The resulting design[4] uses a simple linear address space, six functional registers for sequential programming (see Figure 2), and additional registers as queue pointers to support concurrency.

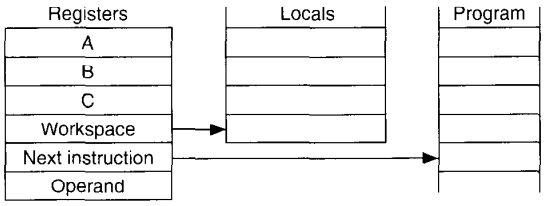


Figure 2. Function of transputer registers for sequential programming.

The six registers for sequential programming are the

- workspace pointer that points to a storage area containing local variables,
- instruction pointer that points to the next instruction to be executed,
- operand register used to form instruction operands, and
- A, B, and C registers that form an evaluation stack. This stack holds the operands and intermediate results for expression evaluation.

The hardware scheduler allows for the combined execution of any number of processes through the sharing of processor time. At any time, a concurrent process is active (either currently executing or on a list awaiting execution) or inactive (either ready to input, ready to output, or waiting until a specified time).

A list holds active processes awaiting execution. This linked list of process workspaces uses two registers in implementation—one that points to the first process on the list and one that points to the last process. The hardware scheduler maintains two such lists—one for high-priority processes, the other for low-priority processes.

The implementation of the instruction set uses a single level of microcode. Many instructions execute in one cycle (50 ns on a 20-MHz transputer); many of the rest execute in two cycles. Some complex functions (such as block move) take an arbitrary number of cycles. These instructions still provide higher performance than possible with software.

To limit the latency figure for switching between low and high priority, time-consuming instructions allow a switch during execution. Consequently, the processor never takes more than 4 microseconds to switch between low priority and high priority.

A context switch between processes executing at low priority occurs only when the evaluation stack contains no useful contents. With minimal need to save and restore registers, the processor implements concurrency very efficiently.

The instruction format uses very compact encoding based on 1-byte instructions. Prefixing instructions are used to form long operands. The instruction size is independent of the word length. In general, a program requires much less storage to hold it than an equivalent program in a conventional or RISC microprocessor. Since a program requires less storage to represent it, fetching instructions use less memory bandwidth. As the transputer accesses memory one word at a time, the processor receives several instructions for every fetch (depending upon the number of bytes in a word).

In addition to Occam, high-performance compilers for C, Fortran, Pascal, and Ada have been implemented for the transputer.

Transputer communications

A link between two transputers implements a pair of Occam channels, one in each direction. Two one-directional signal lines connect a link interface on one transputer to a link interface on the other transputer. Each signal line carries data and control information.

Communication through a link involves a simple protocol, which supports the synchronized communication of Occam. The protocol provides for the transmission of an arbitrary sequence of bytes, which allows transputers of different word lengths to communicate.

Each byte transmits as a start bit, followed by a one bit, 8 data bits, and a stop bit (see Figure 3a). After transmitting a data byte, the sender waits until receiving an acknowledgment, which consists of a start bit followed by a zero bit (see Figure 3b). The acknowledgment signifies both that a process received the acknowledged byte, and that the receiving link is ready to receive another byte. The sending process proceeds only after receiving acknowledgment for the final byte.

Data bytes and acknowledgments multiplex down each signal line. An acknowledgment transmits as soon as recep-

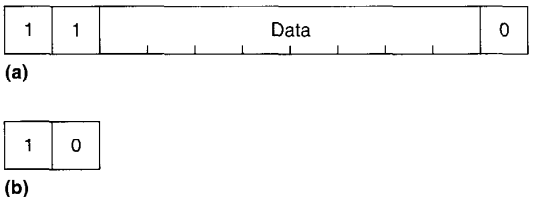


Figure 3. Formats of data links (a) and acknowledgment (b).

tion of a data byte begins (provided a process is waiting for it and room to buffer another data byte is available). Consequently, transmission may continue without delays between data bytes.

As the transputer uses transistor-transistor-logic-compatible (TTL) signals, the applications engineer can extend the links by inserting industry-standard line drivers and receivers.

The design of the links makes engineering of transputer systems easy. Irrespective of internal performance, all transputers use a 5-MHz clock for frequency reference. The low frequency simplifies the clock distribution in large transputer systems. The communications system does not require a phase reference. Therefore, it is not necessary for all transputers to operate on the same clock. The flexibility to use a number of clocks enables interworking between independently designed (sub)systems.

The use of point-to-point serial communications, instead of buses, offers the following advantages:

- simplified board layout and backplane design;
- increased communications bandwidth, as many links in a system operate concurrently; and
- easy interconnection of devices with different word lengths and performance.

Transputers with different word lengths and performance all interwork together, ensuring the easy upgrading of systems as the technology advances. It is not necessary to downgrade a connected set of components to the performance of the slowest component!

Each transputer contains a separate communications engine, allowing communications to proceed in parallel with the execution of processor instructions. Indeed, many applications completely overlap communications and processing, maximizing overall system throughput.

Two link adapters and a link switch add to the flexibility and use of communications. The link adapters provide an interface between a link and a byte-wide port. The Inmos C004 link switch provides a crossbar switch between 32 links, controlled by a separate configuration link. The C004 is cascadable, allowing for the construction of arbitrary networks of transputers (limited only by the number of links on each transputer; most transputers contain four links).

Programming paradigms

While designing transputer-based hardware to perform at any desired level of performance is easy, one must ensure that the software configuration exploits the hardware architecture. Software structured as a sequential program with conventional compilation operates no faster on 10 transputers than on one!

For embedded systems, the design of software architecture and hardware architecture optimally occurs hand-in-hand, resulting in essentially a system design activity.[5]

To configure a program for a network of transputers, the applications designer identifies the parallelism. The designer subsequently maps the parallel processes onto the transputer network to optimize the system according to the design criteria (such as maximized performance and minimized latency).

Experience gained to date with parallel systems—particularly with ESPRIT projects—identified a number of programming paradigms that help to structure systems designs.[6] These paradigms can be written in languages such as Ada,[7] or can employ parallel extensions or libraries in C or Fortran. However, Occam[3] describes these paradigms most conveniently.

Descriptions of the paradigms for algorithmic parallelism, geometric parallelism, and farming appear below:

1) *Algorithmic parallelism.* The designer splits the application into functional units. In simple cases, these units can form a pipeline; in general, they can form more complex structures such as feedback loops. The various stages potentially operate in parallel. With a modest amount of buffering, the communication of data or partially computed results between stages will, in many cases, completely overlap processing (eliminating communications overhead). The structure maps onto one or more transputers, up to the number of components in the structure, with the communications performed internally or via transputer links, as appropriate. For a pipeline structure, its slowest stage limits the maximum performance.
2) *Geometric parallelism.* Designers partition the design on a regular basis. For example, they can divide a screen image into quadrants, or a matrix operation into submatrices. A separate process performs the computation for each partition. The processes often operate independently or interact only with immediate neighbors. A particularly beneficial use of geometric parallelism involves scaling up the size of a problem to be solved (such as performing weather forecasting on a finer mesh).

   Designers usually implement geometric parallelism by allocating processes straightforwardly onto a physical regular network. Geometric parallelism usually attains maximum performance by considering granularity issues. The processes communicate very efficiently with their immediate neighbors when executing on the same processor. The processes communicate less efficiently with immediate neighbors when executing on separate processors. Where one process is allocated to each processor (see Figure 4a on the next page), communication costs dominate performance in most instances. With all processes on a single processor, computation time dominates performance.

   Computation and communication achieve balance at an intermediate level of granularity (see Figure 4b). This

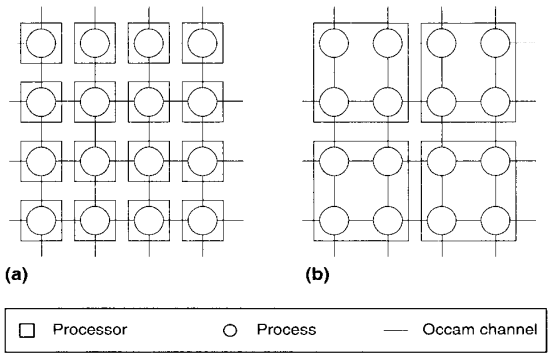


Figure 4. A simple allocation of geometric parallelism granularity (a); a balanced allocation of geometric parallelism (b).

balance results from a boundary-to-area effect (for two-dimensional grids) in which the amount of communication at the boundary varies linearly with the grain size, and the area of computation varies as the square of the grain size. In comparing the two figures, the boundary-to-area effect in Figure 4b results in four times the amount of computation, but only twice the communication as found in Figure 4a. A similar surface-to-volume effect occurs for three-dimensional grids.

3) *Farming*. Designers divide the application into small, similar pieces (for example, when needed to process a large number of similar data items). Each transputer in a network provides a server process, and a master process allocates (or "farms out") work to the servers as they become free. Farming provides two major benefits. First, it automatically balances the load—a server completing one piece of work immediately proceeds to the next. Second, it functions relatively independent of the topology—any reasonably linked network works well. The limit to performance is the rate of dispensing work and handling results.

The Occam model of parallelism offers a significant benefit: Messages sent and received completely define a process. The designer can structure a process internally as a set of processes, thereby using any desired level of nesting. A very powerful technique, therefore, combines the above paradigms in one application, such as a farm of geometric-array servers functioning with pipeline components. (The earlier Occam box diagrams the Occam program that encapsulates all three paradigms within a simple top-level structure.)

The designer can use these models to easily structure applications to define a large amount of parallelism. The narrow and easily defined interface specifications allow for easy reasoning about an application's correctness.

The next-generation transputer

For the past two years, a team at Inmos's Bristol, England, design center has been working to enhance the transputer's performance and suitability for embedded systems. From this work, a new product family—based around a new processor code-named H1—will be launched in Spring 1991.[8]

The team's design goal called for establishing a new standard in single-processor performance while enhancing the transputer family's position as the premier multiprocessing microprocessor. It also required maintaining upward compatibility with existing transputer products.

To meet these goals, the team developed a new microarchitecture that implements the same instruction set as the existing Inmos T805 transputer. The H1 provides an order-of-magnitude increase in performance, combined with enhanced capabilities to support the software standards emerging in the embedded systems marketplace.

The H1 architecture includes such key features as a pipelined, superscalar processor combined with on-chip cache RAM, and improved communications that provide a new degree of freedom in multiprocessor programming.

To complement the H1 transputer, Inmos is now designing a range of network communication products based on a new 100-Mbit/s link protocol. The protocol supports the dynamic routing of messages between processors.

H1 performance

The H1 provides a peak performance in excess of 150 MIPS (million instructions per second) and 20 Mflops (million floating point operations per second) and a sustained performance exceeding 60 MIPS and 10 Mflops. It maintains instruction-set compatibility with the T805.

A number of design features contribute to the achievement of these performance levels. The processor itself uses a pipelined, superscalar architecture, which executes up to eight instructions on each clock cycle and operates at a clock speed of 50 MHz. The number of cycles required to execute many instructions—such as integer and floating-point multiply, and logical shift—decreases significantly.

Unlike other superscalar machines, the H1 architecture does not require an advanced compiler to schedule the different functional units in the processor. Hardware controls the flow of multiple instructions through the pipeline. It is not necessary to modify existing compilers or recompile source code.

An advanced submicron, CMOS (complementary metal-oxide semiconductor) process allows a high transistor count and high clock frequency operations. This process enables the implementation of a sophisticated processor and provides 16 Kbytes of on-chip cache memory.

The move to a cached architecture is a radical develop-

ment. The 16-Kbyte cache is sufficiently large to achieve high hit rates for most applications. The H1 still allows for directly addressed, on-chip RAM for applications containing only small amounts of memory, or ones intolerant of indeterminate performance caused by cache-line misses.

The design team took great care to ensure that the H1 transputer will provide high performance levels in low-component-count systems. For example, the H1 will provide a programmable memory interface with a 64-bit data bus sustaining high data transfer rates for cache-line refill. The interface supports four independent banks of external memory, and the timing for each bank is configured independently from software. For example, an application designer can choose to fill two banks with dynamic RAM—one bank with virtual RAM, the other with peripherals. Such a system frequently requires no external support logic.

H1 error-handling and user-mode processes

The H1 transputer hardware supports the same scheduling algorithms used by current-generation transputers, such as the T805. In addition, on the H1 transputer each process may use a second process (known as a trap handler). When an error—such as an integer overflow or a floating-point error—occurs, control transfers to the trap handler. The trap handler copes with the error in software in all cases before it (in most instances) returns control to the process in which the error occurred.

The H1 also supports a separate user mode. This mode prevents privileged instructions (including communications and scheduling instructions) from executing. It checks and translates all memory accesses from a logical to the physical address space.

H1 transputer enhancements allow programmers to write more efficient real-time kernels.

Memory protection and address-translation mechanisms specifically support secure programming and debugging in embedded systems. For dedicated (single-user) systems, the protection aids the detection of programming errors. For multiuser, general-purpose computing systems, it protects users and the operating system from erroneous programs.

The protection and translation mechanisms are optimized for the requirements of embedded systems. These mechanisms allow the processor to execute code in protected (user) mode at the same speed as normal processes without the performance overhead involved in supporting page-based, virtual memory.

In contrast, but in keeping with its intended market, additional H1 transputer enhancements allow programmers to write more efficient real-time kernels. These enhancements access and control the state of the machine, the process and timer queues, and time slicing and interruptability mechanisms.

New freedom

A limitation in exploiting existing transputer networks is the need to match the parallel structure of the algorithms used to the interconnectivity provided. In the worst case, a specific machine possesses a fixed topology. In the best case (a totally reconfigurable architecture such as the Supernode), the limitation of four links per transputer restricts mapping. This limitation can result in poor software portability, nonoptimum design, and scalability problems.

The H1 product family largely eliminates this problem by providing hardware to allow transputer connections via a low-latency communication network. It supports communication channels between any two processes anywhere in the network.

The hardware simplifies programming because designers do not need to consider how to allocate processes to the transputer network until after completion of the program writing. They can use different allocations on different machines and they can change the allocation to optimize performance. In addition, it is possible—at least in principle—to let the compiler make this allocation, effectively removing all configuration details from the program. Pountain[9] discusses the communications capabilities of the H1 product family in greater detail. The following paragraphs sum up these capabilities.

The H1 transputer itself contains a separate communications processor, which multiplexes a large number of logical communication links (virtual links) along each of its physical links. Each virtual link supports one Occam channel in each direction.

The communications processor transmits messages as a sequence of packets, which all contain 32 bytes of data except the last packet. Each message packet starts with a header, which routes the packet through the communication network and identifies the destination virtual link on the remote transputer.

Designers construct a separate communications network using the Inmos C104 routing device and can use one C104 to connect small numbers of H1 transputers. In larger systems, designers can use C104 connections to form a hypercube, a multidimensional grid, or a tree network.

Each C104 provides 32 bidirectional links. The header of each packet arriving on a link input determines the link on which to output the packet. As soon as the link output is free, the whole packet transmits through it.

An algorithm known as interval labeling decides through which link to send a packet. In interval labeling, each output

link is associated with a continuous set of header values (an interval). The header of an incoming packet lies within only one range, and the packet transmits to the associated link. Optimum, deadlock-free labeling schemes exist for each of the common network topologies.

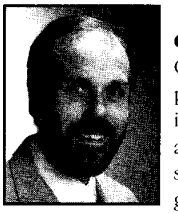
The C104 provides additional facilities to connect networks together and reduce the impact of message congestion on worst-case latency and bandwidth in heavily loaded networks.

THE TRANSPUTER IS WELL ESTABLISHED as a highly cost-effective processor, particularly in embedded applications. It provides unique advantages for applications that require more than one processor, and serves as the basis for many research programs in parallel computing.

Inmos developed the current generation of transputers as general-purpose components for special-purpose machines. The introduction of a higher performance transputer—supporting virtual communications, memory protection, and other advances—represents a significant step toward the development of general-purpose, multiprocessing computing systems. Research continues on the architecture to effectively exploit the full capabilities of VLSI technology during the 1990s. Meanwhile, the H1 provides highly efficient implementations of conventional operating systems and real-time kernels. It greatly reduces the cost of porting existing software and upgrading existing applications to take advantage of the transputer's capabilities.

References

1. A. Baker, "A Signal Achievement," *Parallelogram International*, Vol. 2, No. 29, Aug. 1990, pp. 10-11.
2. M. Homewood et al., "The IMS T800 Transputer," *IEEE Micro*, Vol. 7, No. 5, Oct. 1987, pp. 10-26.
3. Inmos Limited, *Occam 2 Reference Manual*, Prentice-Hall, London, 1988.
4. Inmos Limited, *Transputer Instruction Set—A Compiler Writer's Guide*, Prentice-Hall, 1988.
5. Inmos Limited, *Communicating Process Architecture*, Prentice-Hall, 1988.
6. A.J.G. Hey, D.J. Pritchard, and C. Whitby-Strevens, "Multiparadigm Parallel Programming," *Proc. Hawaii Int'l Conf. System Sciences*, IEEE, New York, 1989, pp. 716-725.
7. J. Barnes and C. Whitby-Strevens, "High Performance Ada Using Transputers," *Defense Computing*, Vol. 1, No. 5, Sept./Oct. 1988, pp. 45-49.
8. C. Dyson, "Inmos H1 Architecture Revealed," *New Electronics*, Vol. 23, No. 8, Sept. 1990, pp. 21-24.
9. D. Pountain, "Virtual Channels: The Next Generation of Transputers," *Byte* (European and World Edition), Vol. 15, No. 4, Apr. 1990, pp. 3-17.



Colin Whitby-Strevens manages the Central Technology Group in the Microprocessor Design Division of Inmos Limited. His interests include microprocessor architecture, parallel programming and system design, and programming languages.

He recently chaired a series of Commission of European Countries workshops—involving more than 70 companies and organizations—to establish the Open Microsystems Initiative. Prior to joining Inmos, he was a lecturer in computer science at the University of Warwick, where he developed a multiprocessor operating system for the Modular One computer. He subsequently established the Warwick Distributed Computing Research Project, which developed some of the key concepts subsequently exploited in the Inmos transputer architecture.

Whitby-Strevens holds a BSc degree in mathematics from Hull University, and the Diploma in computer science and PhD from Cambridge University.

He authored 30 technical papers and coauthored *BCPL—The Language and Its Compiler*. He is an affiliate of the IEEE Computer Society, a member of the Association of Computing Machinery, and an associate member of the British Computer Society.

Address questions concerning this article to Colin Whitby-Strevens, Inmos Limited, 1000, Aztec West, Almondsbury, Bristol, BS12 4SQ, United Kingdom.