UNE IMP

EHE ★QMIPS★ CWI

INR UER

ZAR UTO

Quantitative
Modeling
In
Parallel
Systems

Communications in Multiprocessor Machines
A Survey

Alain Jean-Marie and Philippe Mussi
INRIA Sophia-Antipolis
06902 Sophia Antipolis Cedex, France

Michel Syska
I3S/Université de Nice-Sophia Antipolis
06905 Sophia Antipolis Cedex, France

D W3.T2-T3.19.v1

29 September 1994

Abstract The purpose of this paper is to provide ideas on what are the most important issues to be addressed by the modeling and performance evaluation community in the area of communications for parallel and distributed systems. We first give a description of the principal components of a parallel or distributed computer, as far as communications are concerned. We then provide a descriptive list of the most prominent multiprocessor systems existing on the market. We then describe what are, in our views, some of the most challenging problems faced by engineers and researchers who try to use these machines.

The QMIPS Project: The QMIPS project is a collaborative research project supported by the CEC as ESPRIT-BRA project no 7269. It is being carried out by the following organisations: CWI (Amsterdam), EHEI (University of Paris V), Imperial College (London), INRIA (Sophia-Antipolis), University of Erlangen, University of Newcastle, University of Torino and University of Zaragoza.

Contents

Introduction

The purpose of the QMIPS project is to provide to the research community and to engineers, tools allowing to solve modeling problems arising when conceiving or using distributed systems.

In the two first years of the project, a number of formalisms and solution techniques able to answer to this kind of problems at various levels of detail and with various levels of tractability and accuracy have been identified [5, 23, 10]. Most of the third year will be devoted to testing the application of these techniques to "real" problems. Indeed, a legitimate fear is that, when the research stays too far apart from the actual source of problems, there is a risk that the models selected to test a methodology remain "toy" examples, too idealized or too small in size to be useful to practitioners. This motivates a preliminary effort in order to identify classes of problems of certified practical interest.

Performance evaluation techniques can be applied to parallel or distributed multiprocessor systems at two levels:

- at the conception level, that is, before actually building the system, to allow to decide between several architectural choices
- at the utilization level, in order to evaluate beforehand performances (throughput, response times) of programs, or find which distributed algorithm is best adapted to solve some problem on some given architecture.

It appears that a close inspection of existing machines provides models for both aspects, for the following reasons.

Parallel machines were in the beginning experimental computers offering to the potential users the hardware necessary to devise parallel programs. Different architectural choices lead to different classes of machines. This gave birth to the classical classification of multiprocessor systems : distributed memory *vs* shared memory, synchronous execution (SIMD) *vs* asynchronous execution (MIMD), network topology (hypercubes *vs* grids) etc. The users had very little software available to help them use the machine.

As the vendors of multiprocessor systems moved to the industrial market, they found themselves obliged to provide more and more tools without which programming the machine would require an investment which no industrial cares to spend.

After a big development in the early 90's, which accounts for the length of the list in the next section, the market of distributed and parallel systems seams to experience some stagnation. This is stressed by the recent closure of the activities of some prominent vendors on the market place.

The reasons for this phenomenon are not yet clear. It is probably not due to a lack of need in parallel and distributed computing in the industry. In our opinion, some important factors are that, for one part, some proposed architectures are still too "experimental" and not reliable enough to be trusted by industrialists. But the most important reason may be that at the moment, these machines are still too difficult to program efficiently. From an industrial point of view, developing codes using optimally the hardware is too large an investment, and using the "high-level" routines provided by the vendor too often results in a big loss of efficiency, compared to the theoretical one.

In any case, the result is that the number of architectures actually available is decreasing. One may risk the prediction that in the future, on the one hand, only a few significantly different architectures will be proposed (those which will be judged the best by the main machine vendors and the main machines buyers), and that on the other hand the technical solutions of these computers may be found in present day systems.

For these reasons, we chose to concentrate on the study of existing systems, and to try to identify what are the relevant questions to ask, and the principal modeling challenges which the performance evaluation community, and the QMIPS project in particular, should address in priority.

The remainder of the paper is organized as follows. In the next section (section 1), we give a short description of the principal characteristics of communications related parts of multiprocessor systems. In section 2, we provide a list of the most prominent available architectures, using the description criteria above. Finally, in section 3, we expose a number of problems which are, in our views, to be addressed when analyzing the performance of parallel systems.

1 Taxonomic Characteristics

1.1 Network Interface

We study parallel computers whose processing units are duplicated so that a single machine could consist of several thousand processing units all working on the same application. Data exchanges between nodes are realized using the links of the interconnection network. This means that each node must contain the set of units necessary for communication.

It is important to distinguish these *distributed* memory parallel machines from *shared* memory parallel machines. In the latter, the processors all use a common memory, while in distributed memory parallel machines each processor has its own memory and exchanges of data between the processors are realized by message exchanges in the interconnection network.

Today, due to the high costs and the complexity of realization (it is very difficult to manage a shared memory used by dozens of processors), many manufacturers have opted for "distributed". This choice might have to be revised if and when technological changes warranty such a move. The great supercomputer companies have made the choice in order to respond to the challenge posed by the processing speed in TFlops (10^{12} floating-point operations per second) on real applications.

Unfortunately, in order to execute a program n times faster, it is not sufficient to execute it on n processors (even assuming that the instructions of the program show a high degree of parallelism). Indeed, the processors must exchange information and the cost of communicating may be high. Thus the data exchanges will be done to the detriment of the global performance of computation. Optimizing the time it takes to move data within distributed memory parallel machines is, therefore, the key to obtaining a significant performance on these machines.

The support of these communications consists mainly of two large classes of networks:

- *Point-to-point* networks, in which each node contains a set of units as described above and can directly communicate with its neighbors (the nodes to which it is physically linked), or use other, intermediate, nodes to communicate at greater distance.

- *Multistage* networks, where it is convenient to distinguish between two types of nodes – those that contain the processing units and those containing units dedicated to communication. The latter are usually *switches.* They serve either to route messages between processors (distributed machine), or to route messages between processors and common memory (shared memory machines). In the latter case, the memory is divided into *banks* in order to avoid access conflicts.

There are other classes of networks (e.g. bus networks, optical networks), but we shall restrict ourselves to the two large classes described above.

A distributed memory parallel machine is usually shown as a cabinet containing a set of cards. On these cards we find *modules* (nodes), composed of elementary units (processor(s), external memory), and possibly a communication management component. This cabinet is often connected to a workstation called *host* or *frontal.*

Here we consider a machine with nodes of the type shown in Figure 1. In it, we find scalar and floating-point processors, some external memory, a memory interface, various related caches and several links for interconnection network access. For each link interface we have a set of independent DMA (*Direct Memory Access*) systems. Each DMA interface can access data contained in the memory or in the caches without slowing down the processing units or the other interfaces. This is done thanks to the interfaces' own buses. For example, the Inmos' T9000 [40] processor is conceived on this model.

The nodes communicate with their neighbors by exchanging messages through *channels.* A channel is a one-way point-to-point connection between two nodes connected by a physical link. Several channels can share the same physical link, in which case we speak of *multiplexing.*

Communications of processes executed on distant processors are supported by the interconnection network formed by the physical links. Each node can have a *router* whose role is to route messages between non-neighboring nodes. The routers implement – in a distributed manner – a routing algorithm which specifies the path to follow from any node x to any other node y. This algorithm is given by a *routing function.*

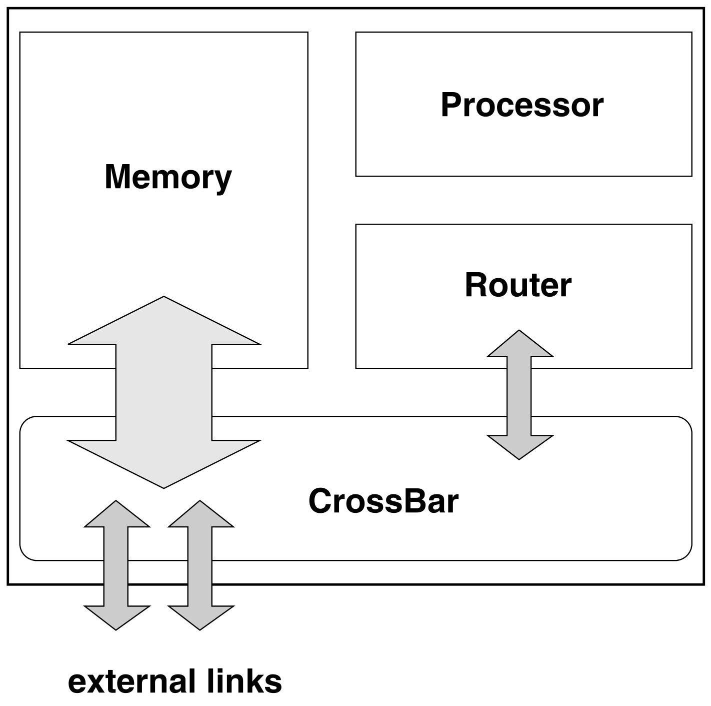

Figure 1: Architecture of a network node

Locally, each router chooses, for each message arriving on one of the channels, an appropriate output channel. For this a table might be used. This sort of method is not usually applied in multiprocessor machines (contrary to the big planetary networks, for example for routing electronic mail on Internet) which most often use a routing method based on a simple arithmetic which can be implemented in the node with programmable memory.

Distributed architectures are usually modeled by a graph $G = (V, E)$ where the set V of vertices of the graph represents the nodes and the set E of edges represents the communication links between the nodes. If a link is one-way, it is modeled by an arc; in general, however, the links are two-way (there is at least one channel in each direction).
Let us consider the case of two processors p_1 and p_2 joined by a two-way communication link.

- If only one message at a time can pass between p_1 and p_2, either from p_1 to p_2 or from p_2 to p_1, the connection is called *half-duplex*. The network is then modeled by an undirected graph.
- If two messages can pass at the same time through the same link, one from p_1 to p_2 and the other from p_2 to p_1, the connection is called *full-duplex*. This is the usual case. The network is then modeled by a symmetric directed graph (if there is an arc from a vertex x to a vertex y then there is one from y to x).

We must also characterize the communication possibilities of the interface between the memory and the communication links, for each processor.

- If, during a communication, each node can only send or receive one message on one link at a time, the communication is called 1-*port*. This will result in slow-downs on the processor level as they may not be able to send messages sufficiently fast. Intel's iPSC/1 is an example of a half-duplex 1-*port* machine.
- If, on the contrary, each node can simultaneously use all its links, communications are called Δ-*port*, where Δ refers to the maximum degree of the nodes in the network. It is now the number of links that limits communication. Δ-*port* nodes with full-duplex links are used in transputer-based systems and Intel's iWARP.

The time spent to transmit and to receive data must be minimized, but most importantly, the interface must permit simultaneous transmission and reception of several message. This is not possible in the majority of today's elementary processors. The technique used by Fujitsu in the AP1000 allows to send a message directly from cache memory to a router and this minimizes the transfer time. Indeed, the data to be transmitted is often that just processed and so is in the cache.

1.2 Memory

1.2.1 Memory Hierarchy

The influence of memory hierarchy on computational performance is not particular to multi-processor systems. However, recent MIMD machines, often based on standard RISC processors, tend to incorporate more complex memory architectures. A typical example is the Convex MPP machine (see Figure 8), whose storage system is made of 6 different stages:

- processors registers (Access time: 1cycle)
- separate on-chip data and instruction caches (1 cycle)
- external private caches (10 ns)
- local memories (500 ns)
- access to other processors local memories (500 to 1200 ns)
- permanent storage (disks), with access times in milliseconds.

Even when access conflicts to shared memories are not taken into account, those highly heterogeneous access times to storage, for both data and instruction streams, must be carefully considered when modeling the behavior of parallel applications on such architectures.

1.2.2 Memory Sharing

In addition to cache misses at various levels, access conflicts to share memory locations must be taken into account when modeling or observing parallel programs behavior.

Memory locations may be shared by means of several mechanisms:

- *central shared memory*: this approach, although reaching a wider audience by appearing in general–purpose systems (file-servers, workstations), is gradually declining in the field of high performance computing, probably due to the limitations it puts on scalability, particularly on the number of processors that can be connected efficiently.
- *distributed shared memory*: distributed memories, generally attached and prioritarily owned by elementary processing units, are accessible to all the processors via specific pieces of hardware, in charge of coherency control and access arbitration (see sections 2.2 and 2.9).
- *virtual shared memory* [31, 12]: distributed memory units are multiply accessed through software. This mechanism may be seen as an extension to the classical *virtual memory* mechanism used in modern operating systems.

1.3 Synchronism

The various technological choices led to a great diversity in parallel machines [1, 2, 3, 18, 25, 26, 51]. Flynn introduced a classification [20], still authoritative if slightly dated. The classification is based on only two criteria: the type of *instruction flow* and the type of *data flow* treated by elementary processors. The flows are either *simple* or *multiple*. It is difficult to classify some machines in this scheme (for example systolic machines) or to distinguish distributed memory parallel machines from shared memory parallel machines.

The parallel computers of concern in this report are, in Flynn's classification, of type SIMD (*Single Instruction, Multiple Data flow*) or MIMD (*Multiple Instructions, Multiple Data flow*), in either case, with distributed memory. The SISD (*Single Instruction, Single Data flow*) machines correspond so sequential computers (von Neumann model) and the MISD (*Multiple Instructions, Single Data flow*) machines can be seen as pipeline units since such a unit executes several parallel instructions on the same data flow. Below we describe the SIMD and MIMD modes.

1.3.1 Type SIMD

This is a hardware concept: the processors have no sequencers and the program is executed on a station called *frontal*, which distributes instructions to the processors. The functioning is *synchronous*.
During the execution of a program, the instructions that manipulate parallel data are broadcast to all the processors so that they can be applied to the data distributed in their local memories. Hence, all the processors execute the same instructions, albeit on different data. It is, fortunately, possible to mask, on some processors, the writing of the result of an instruction. This allows for *conditioning* (that is, the possibility of testing).

The main architectural advantage of the SIMD approach is that only the processing units need to be duplicated. Indeed, a single control unit is sufficient to decode the instructions for all the processors. This sort of machine favors regular and local data movements, as, for example, the shifts of lines in a mesh, but also sometimes allows access to fast communication procedures such as broadcasting of simple data from one processor to all the others.

Algorithms developed for these machines are of type *parallel data.* Thus, these machines are well adapted to computations using vectors and matrices. On the other hand, solving "irregular" problems on this sort of architecture is not easy and it is not clear if this type of machine will survive in the future, except for specific applications.

Examples of these machines are CM-2 (Thinking Machines Corporation), MP-x (MasPar), DAP (AMT). They all have a common feature in that each connects many simple processors (64K 1-bit processors on the CM-2, 16K 4-bit processors on the MP-1).

1.3.2 Type MIMD

This is another hardware concept: each processor has its own sequencer and so executes its own program on local data (when the machine has distributed memory) or on data located in the global memory (when the machine has shared memory). Usually, MIMD machines don't have a global clock and operate in *asynchronous* mode. Given the clock problems, the execution of a program on an MIMD machine is nondeterministic.
The interaction processors/processors or processors/local memories is realized through message exchanges in the interconnection network.

The class of algorithms that can be implemented on this type of machine is wider than that of algorithms implementable on SIMD machines. Fine-tuning, however, is much more complex.

Examples of these machines are Paragon (Intel), Inmos' transputer-based machines and the T3D (Cray). These machines have a smaller number of processors than the SIMD machines but are much more powerful (up to 4096 64-bit processors with two processors per node on Intel's Paragon).

1.3.3 SPMD mode

This term covers two concepts - a programming mode (*weak* SPMD) and a semantic concept (*strong* SPMD).

In the first case, SPMD simply specifies that programming on an MIMD machine is done by loading the same code on the processors. Of course, each processor works on its own data.

From a semantic point of view, SPMD refers to the specification of communication in parallel programming. Thus, in SPMD, communication is regarded as nothing but a rearrangement or redistribution of data. Local instructions of type `send` or `receive` don't come into the picture; all the processors participate in the rearrangement of the data. Consequently, the program is decomposed into distinct processing and communication phases. An SPMD program is easily compiled on an SIMD machine where synchronization is intrinsic. But compiling an SPMD program on an MIMD machine requires access to synchronization tools.

Up to a few years ago, such synchronization could only be done by software, thus limiting applications of this semantic concept. In the last few years new MIMD machines appeared, with fast synchronization tools in the hardware (e.g. TMC's CM-5q).

1.4 Topology

We list the dominant network topologies in use in parallel computers, with their principal properties. The interested readers may find more details in [32, 4, 50].

1.4.1 Mesh

The *n-dimensional mesh*, denoted by $M(p_1, p_2, \ldots, p_n)$, is the Cartesian product of n paths on p_i vertices, with $i = 1, 2, \ldots, n$ and $p_i \geq 2$.

This graph has maximum degree $\Delta = 2n$, minimum degree $\delta = n$, the number of vertices is $N = \prod_{i=1}^{n} p_i$, the number of edges is $\sum_{i=1}^{n} \frac{(p_i - 1)N}{p_i}$ and the diameter is $\sum_{i=1}^{n} (p_i - 1)$.

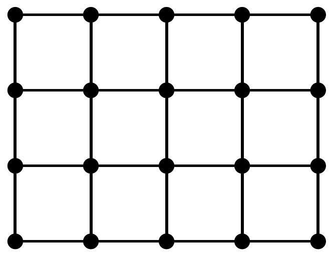
Figure 2: Mesh $M(4,5)$

This topology is used by Intel for the Paragon machine described in the next section.

1.4.2 Toroidal mesh

Here we add wraparounds to the Mesh, and the new parameters follow:

The *toroidal mesh of dimension n*, denoted by $TM(p_1, p_2, \ldots, p_n)$, is the Cartesian product of n cycles C_{p_i} with $i = 1, 2, \ldots, n$. Degree: $2n$, number of vertices: $N = \prod_{i=1}^{n} p_i$, number of edges: $n \times N$, diameter: $\sum_{i=1}^{n} \lfloor \frac{p_i}{2} \rfloor$.

The toroidal mesh topology is used in the Fujitsu AP1000 (dimension 2) and Cray T3D (dimension 3) multiprocessors.

1.4.3 Hypercube

The *hypercube of dimension n*, denoted by $H(n)$, is a graph whose vertices are all words of length n over the two-letter alphabet $\{0, 1\}$. A vertex of the hypercube is denoted $x_1 x_2 \cdots x_i \cdots x_n$ and is joined to the vertices $x_1 x_2 \cdots \overline{x_i} \cdots x_n$ with $i = 1, 2, \ldots, n$. Degree: n, number of vertices: $N = 2^n$, number of edges: $n 2^{n-1}$, and diameter is $n = \log_2 N$

This was a popular topology used in the CM-1 and CM-2, nCUBE series and Intel's iPSCs.

1.4.4 Butterfly network

The *butterfly* multistage network of dimension n is a network formed by $n + 1$ stages of 2^n $(2, 2)$-switches each. Thus, it has 2^{n+1} input links and 2^{n+1} output links. The stages are generally labeled by the integers from 0 to n. The associated graph is of order $(n + 1)2^n$ and has $n 2^{n+1}$ edges. Its vertices are pairs (l, x) where l is the number of the *level* or *stage*, $0 \leq l \leq n$, and x is the number of the switch's row in n-bit binary (a binary word of length n, denoted by $x_0 x_1 \cdots x_{n-1}$). Two vertices (l, x) and (l', x'), $l \leq l' \leq n$, are adjacent if and only if $l' = l + 1$ and if one of the two following conditions is satisfied:

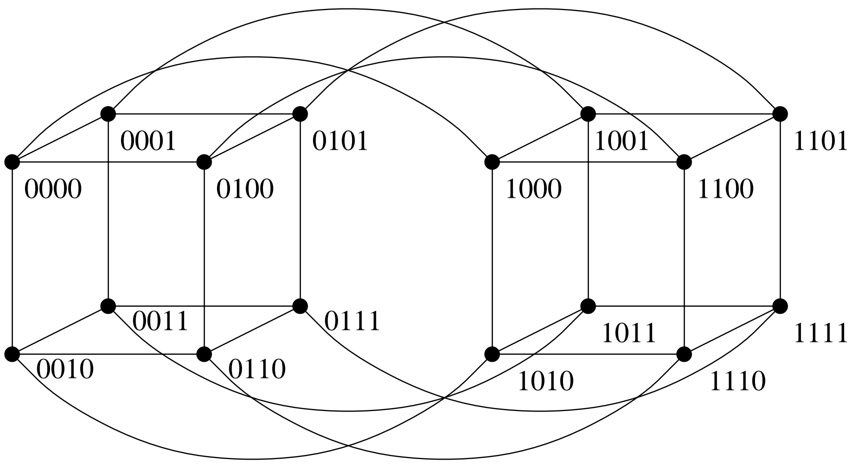

Figure 3: Hypercube of dimension 4

- x and x' are identical
- x and x' differ in the l-th bit.

If x and x' are identical, the edge is a *straight edge*. Otherwise, it is a *cross edge*.
Figure 4 shows a butterfly network of dimension 3 and its associated graph. We number the inputs and the outputs of the network from 0 to $2^{n+1} - 1$ or by the corresponding binary representation. This comes down to adding a 0 or a 1 after the binary representation of the switch's row number.

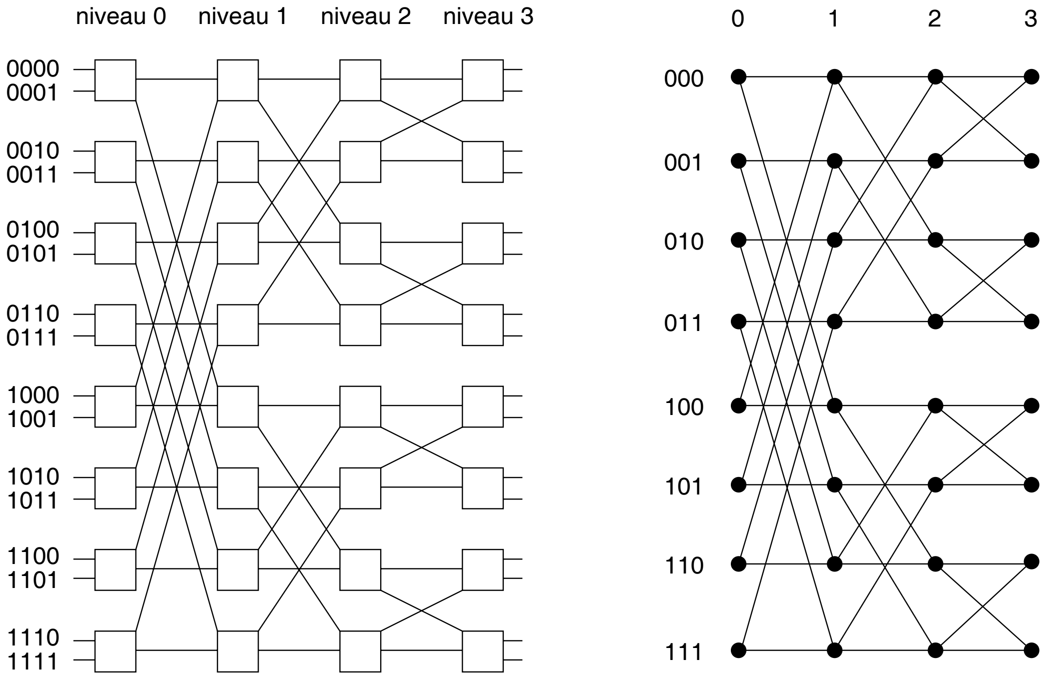

Figure 4: Butterfly network of dimension 3 and its graph

The butterfly network has recursive properties. The butterfly network of dimension n contains, as subnetworks, two butterfly networks of dimension $n - 1$ (simply remove the vertices of stage 0 and the edges incident with them).

Another important property of the butterfly network is simplicity of its routing. The input link $x_0x_1\dots x_n$ of stage 0 is joined to the output link $x'_0x'_1\dots x'_n$ of stage n by exactly one path of length n. This path traverses each stage exactly once, using the cross edge between the l and $l+1$ if and only if $x_l \neq x'_l$.

1.4.5 Ω network

The Ω network of dimension n is formed by $n+1$ stages of 2^n $(2,2)$-switches. The outputs of the first 2^{n-1} switches of a stage are joined to the top inputs of the switches of the following stage, in the same order. More formally, the graph associated with the an Ω network of dimension n has $2^n(n+1)$ vertices, denoted by (l,x), with $0 \le l \le n$ and $x \in \{0,1\}^n$. For each l such that $0 \le l \le n-1$ there is an edge between the vertices (l,x) and $(l+1,x')$ if and only if one of the two following conditions holds:

- x' is a left cyclic shift of x
- x' is a left cyclic shift of x followed by an exchange of the last bit.

For example, in the graph associated with an Ω network of dimension 2, the vertex $(l,01)$ is joined to the vertices $(l+1,10)$ and $(l+1,11)$. Figure 5 shows an Ω network of dimension 2 and its graph.

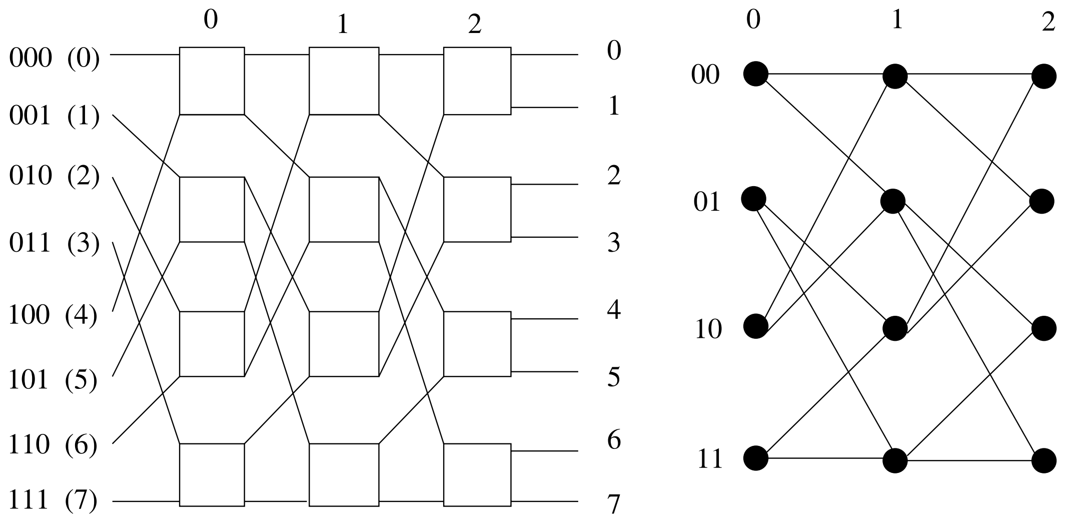

Figure 5: Ω network of dimension 2

Note that, by definition, placement of the edges is the same between two consecutive stages. The form what is called a *perfect shuffle.*
In fact, the graph associated with the Ω network is isomorphic to that associated with the butterfly network. It suffices to map the the vertex (l,x) of the Ω graph to the vertex $(l,\pi_l(x))$ of the butterfly graph, where π_l is the l times iterated right shift of x. Then, for example, the vertex $(2,100)$ of the Ω network of dimension 3 becomes the vertex $(2,001)$ of the butterfly network of the same dimension. When the graphs associated with two networks are isomorphic, we say that the networks are *topologically equivalent.*

The Ω network has 2^{n+1} input links and as many output links. We number them from 0 to $2^{n+1}-1$, in binary. The input links $0x_1\cdots x_n$ and $1x_1\cdots x_n$ are joined to the switch $x_1\cdots x_n$, that is, stage 0 of the network is also preceded by a perfect shuffle on the input links. Let M_j be the memory bank which the processor P_k wants to reach and let $m_0\cdots m_n$ be the binary representation of j. The switch of the stage i of the Ω network uses the bit m_i of the destination address to determine the link to establish. For any input of this switch, if $m_i=0$, the connection passes through the top output of the switch, otherwise it goes through the bottom output.

Suppose that in the Ω network of dimension 2 we would like to connect the processor P_6 to the memory bank M_3. The address of M_3 is 011. Thus, $m_0=0$, $m_1=1$ and $m_2=1$. We make the following connections.

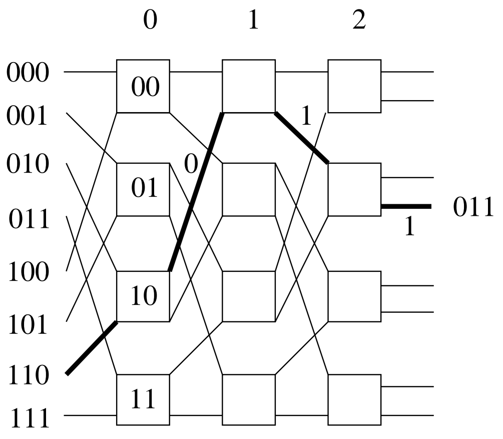

Figure 6: State of an Ω network

We leave stage 0 (where we arrived from the processor P_6) from the top and the stages 1 and 2 by the bottom, as shown in Figure 6.

Since the Ω network is equivalent to the Butterfly network, it is not rearrangeable and not all permutation of the inputs can be realized.

1.5 Commutation

When a message is transmitted between processors that are not directly linked the message must be routed through intermediate nodes and this routing is done with the help of routers. A router will be characterized by its VLSI area, which we want as small as possible, and by its *switching time*, also called *latency*. Switching in a router is a physical phenomenon consisting of receipt of a destination address, decoding of the address in order to determine the appropriate output channel, and sending of the address through this channel. Depending on the protocols used, switching can also include physical connection of the input link with the output link determined by the router. Its latency can be only a few tens of nanoseconds in case of hardware routing but can go beyond a microsecond in the case of software routing.

The various usual switching techniques are described in Kermani and Kleinrock in [30].

1.5.1 Circuit-switching

This is the principle of a telephone: we first establish a connection (this means reserving a sequence of channels) and the conversation begins after. The making of the connection is made by propagation of the header h of the message containing the destination address, return of a receipt, and then transmission of the message in one go. Other communications wanting to use a part of the connection (circuit) thus established are blocked until all of the connection becomes free.

The *direct-connect* suggested by Intel on the hypercubes of the iPSC/2 and iPSC/860 series is routing by circuit-switching.

1.5.2 Store-and-forward

Messages move through the network to their destination by passing through intermediate vertices. At each stage, the channel used is immediately freed. This technique is known under the name *store-and-forward* in the context of distributed machines. One flaw of this technique is the necessity of a large register for storing the message on the intermediate processors. In fact, such messages are usually stored in memory. Memory access times, however, being proportional to the size of the messages, then slow down communication dramatically.

Meiko's Computing Surface, as well as other transputer based machines use store-and-forward routing.

1.5.3 Packet-switching

This mode is an adaptation of the preceding one. Messages are cut into packets of fixed size and pipeline effect can be exploited. The packets are routed independently one from another since each has a header containing he destination address; they can even take different routes.

Let us observe that this implies an overhead in communication, generated by the supplementary information added to each packet, in comparison to store-and-forward mode. Furthermore, the recomposition of a message from the various packets also requires supplementary information to be transported in each packet. On the other hand, the advantage of this mode of switching is the use of small registers resident in the router. The storage capacity of a node is limited to the size of a packet per channel.

1.5.4 Wormhole routing

In the most recent distributed memory machines, the store-and-forward routing mode was abandoned in favor of *wormhole* routing.
Contrary to the store-and-forward mode, in which messages (or packets) are entirely stored in the memory of a processor before being transmitted to the next processor, in the wormhole routing mode the messages proceed through the processor network flit by flit (a *flit — flow control digit* — is the size of the buffer of a channel), with the first flit containing the destination address. The header, that is, the first flit, advances by a channel each time it is possible. The rest of the message follows, freeing the last channel which contains the end of the message. The last channel then becomes available for another message.

It is important to distinguish this routing mode from routing by packet-switching. In the latter, each packet contains the destination address in its header and can be routed independently. In wormhole routing mode, only the first flit contains the destination address.

In the wormhole routing mode the intermediate stages between the source and the destination consist in establishing a *virtual circuit*. The movement of a message in this mode can be likened to that of a worm moving through earth. A message can begin to be received before its sending has been completed. Similarly, if the message is sufficiently short, the source is freed before the reception of the first flit at the destination. Note that once a flit has been assigned to a channel, the channel cannot transmit any flit of any other message until the original message has passed through it (one cannot "cut a worm"). Only one flit is stored in a node's buffer and there is no memory access (no intermediate storage of the message, costly in time [48]). Further, small flits allow for the use of small buffers, which makes easier VLSI implementation of a router. If the header is blocked, that is, if all output channels are used by other messages, the movement of the message stops and the flits remain stored in the buffers of the channels they occupy.

This technique is well adapted to processor networks in which *physical distance* (measured in metric units) between neighboring nodes makes transmission errors negligible. The internal flits containing the body of the message are sent without waiting for a receipt and pass directly from channel to channel following the connections made by the header flit. On the other hand, if the physical distance between neighboring nodes is over a certain threshold, transmission errors can occur and thus make necessary a protocol with receipts issued for the flits. This will hurt the transmission speed but does not invalidate the protocol.

The wormhole mode was chosen by Inmos for its new series of transputers T9000 and crossbars C104. TMC's CM-5 also uses this type of routing.

1.5.5 Virtual-cut-through

A technique similar to that of wormhole routing is the *virtual-cut-through*, studied by Kermani and Kleinrock [30]. Its routing mode is identical to that of wormhole routing except when the progress of the first flit of a message is impossible thanks to all output channels being already used to route other messages.

Let us recall that in wormhole routing the flits remain stored locally in the buffers of the channels they occupy, that is, "along the way". In contrast, in virtual-cut-through routing, the flits containing the body of the message continue to move and are all stored on the node where the first flit is blocked. The router must, therefore, have arbitrarily large buffers, which means that it cannot easily be integrated to the node. As far as we know, this technique has not been implemented while wormhole routing has been studied and, as mentioned earlier, implemented.

1.5.6 Others modes

Two routing features that we find interesting have not yet been implemented in general–purpose machines.

- A *broadcasting* mode, which would allow the router to transmit on all of its output channels a message received on one entry channel. The routing would then be on trees rather than paths. This would make easier broadcasting procedures of a message from one node to another.
- A *transparent* mode, in which a processing unit could read the flits passing through the router to which it is attached. This would also facilitate broadcasting procedures. As it is, for example, in wormhole routing, when a message is sent from a node x to a destination z and when the routed message passes through a node y, the message is not stored in y's memory but only in the router's buffer. So y has no knowledge of it.

1.6 Routing

We have seen that the routers realize in a distributed manner a routing algorithm which specifies the path to follow in the network in order to go from a node x to a node y. This routing algorithm is described by a *routing function*, as opposed to *centralized routing* in which a master node manages all routes.

The interconnection network may be modeled by a directed graph $G = (V, E)$ where

- $V = P \cup M$, where P models the routers related to the nodes and M the nodes' local memories;
- $E = C \cup I \cup O$, where C represents the communication channels between the routers, I the channels from the local memory to the router, and O the channels from the router to the local memory. More precisely, each vertex x of P is connected to a corresponding vertex x' of M by an input channel i_x of I (inputs) from x' to x, and by an output channel o_x of O (outputs) from x to x'. These channels model the exchanges between the router and the processor memory.

In order to simplify exposition, we will often use only a simplified representation, in which we identify P and M and we no longer refer to the input (I) and output (O) channels. The simplified graph is $G = (V, E) = (P, C)$ and the routing function is a function from $E \times V$ in E.

A routing is called *shortest path* if for every pair of vertices x and y the routing path from x to y, defined by the routing function, is a shortest path from x to y.

The routing function assigns a unique path to each pair (*source*, *destination*). Such a routing is called *static* or *deterministic.* Such a routing can cause congestion problems when the traffic is not homogeneous and when several messages coming from different sources are routed through the same link. This is the same principle as when many cars coming from different streets attempt to turn into the same street. In the following section we look at some solutions to this problem.

Congestion problems and adaptive routing

To solve congestion problems, an *adaptive* routing function can be used, which constructs the paths dynamically, depending on traffic. In this case, the routing function R in its simplified representation, is no longer from $E \times V$ into E but from $E \times V$ into E^{Δ}, where Δ refers to the maximum degree of the nodes in the network. More precisely, the function R assigns, to an input channel c of a vertex v and a destination d, a set of output channels leaving v.

There are many possible strategies for choosing a channel among those suggested.

- **Greedy routing**: wait until a channel which would bring the message closer to its destination becomes free and use it
- **Random routing**: choose a direction randomly and take it if possible; otherwise repeat the operation. This homogenizes the flow of data in the whole network.
- **Universal routing**: go to a randomly chosen destination and follow by using static routing from this new source to the destination [53]. Such a routing no longer uses local properties of the network. This possibility is allowed by Inmos' C104 [40].

- **Forced routing**: this technique is described in [22] in relation to the Mega machine. The goal is never to block a message. In the case where a delivered message cannot be accepted immediately, it is re-routed until it is accepted later. Similar techniques are used in the CM-2.

1.6.1 Deadlocks in static routing

Deadlock is often an unpleasant surprise when a first parallel program is written. It is the result of cyclic dependence in the management of communication between processes which block each other. We shall show how to construct deadlock-free routings, especially using *virtual channels* [14] or *virtual networks* [39].

These only ensure that there will be no deadlock during the execution of a program due to communication routines.

We will now study deadlock problems in wormhole mode using static (deterministic , see [14]) routings.

1.6.1.1 An example of deadlock Figure 7 shows an example of a deadlock. Suppose that the nodes

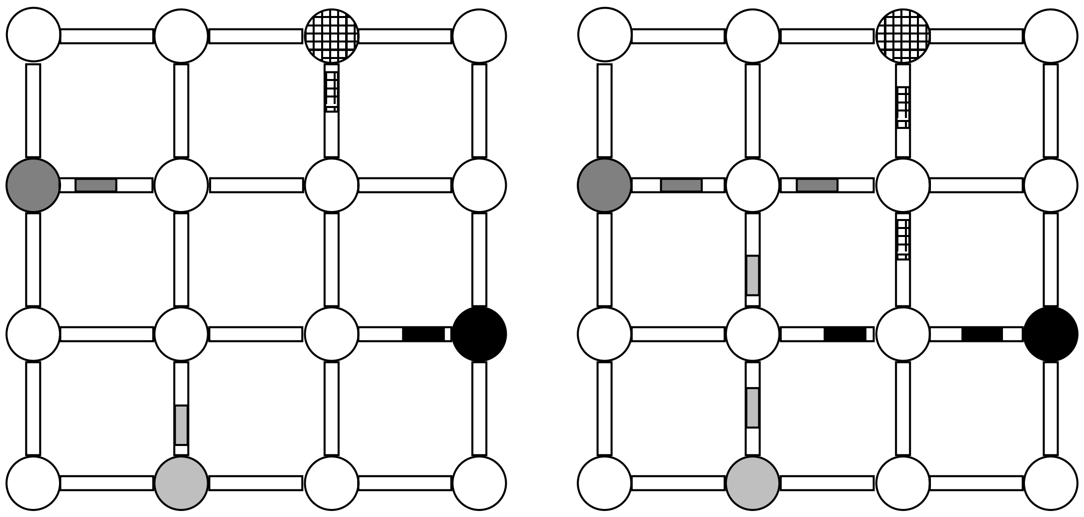

Figure 7: Deadlock in the mesh

10, 02, 23 and 31 must each send a message to, respectively, 22, 21, 11 and 12. Suppose further that the routing function gives the following four paths:

$$10 \rightarrow 11 \rightarrow 12 \rightarrow 22$$

$$02 \rightarrow 12 \rightarrow 22 \rightarrow 21$$

$$23 \rightarrow 22 \rightarrow 21 \rightarrow 11$$

$$31 \rightarrow 21 \rightarrow 11 \rightarrow 12$$

At step 1, the first flit of each message – containing the destination address – is deposited in the queue of the appropriate output channel. At step 2, the first flits have moved towards their destinations by a channel and the second flit of each message has been sent by each of the sources. Here we suppose – without loss of generality – that the transmissions are simultaneous. After the second step we find ourselves in a deadlock: all the queues that the flits would like to use are occupied by other flits which cannot move for the same reason.

There is a cyclic dependence among the queues of the channels in this instance of the routing. To study this dependence we use the *channel dependence graph* discussed below.

1.6.1.2 Channel dependence graph

In order to observe the properties of the routing function R we define the *channel dependence graph*, denoted by $\mathcal{D}(G,R)$, associated with R.

Given a graph G and a routing function R on G, the channel dependence graph $\mathcal{D}(G,R) = (E,U)$ is a graph whose vertices are the channels of G and the set of arcs U is defined by

$$U = \{(c,c') \in E \times E \mid \exists\ x \in V, c' = R(c,x)\}$$

The notation $\mathcal{D}(G,R)$ recalls that the dependence graph is strongly related to R. $\mathcal{D}(G,R)$ is in fact a subgraph of the line graph of G.

With the channel dependence graph we can characterize the presence or the absence of deadlocks. The following theorem gives a sufficient condition for the absence of deadlocks.

Theorem 1.1 (Dally and Seitz [14]) *Let R be a routing function on the graph graph G. If $D(G,R)$ is acyclic then R is deadlock-free.*

The problem to solve now is "Given a graph G and a routing function R on G, what can be done if $\mathcal{D}(G,R)$ contains circuits?". One of the tools used to solve deadlock problems are *virtual channels.*

1.6.1.3 Virtual channels

A physical link can "let through" only one flit at a time. If this flit belongs to a message whose progress is blocked somewhere ahead, no other message can use this link, which usually means that many, if not all, messages are stopped.

In order to allow the messages not implicated in the "traffic jam" to follow their course, new resources can be allocated in the form of additional queues, *virtual channels.* This will allow the sharing the physical link by as many independent channels. We speak of virtual channels since the physical link supports several channels. Translated into the language of interconnection networks, this means adding arcs to C in order to obtain a directed multi graph C'.
The general technique used in [14] is the following.

1. A channel corresponds to several queues rather than just one. Let k be the number of queues at each channel. A physical channel is equivalent to k virtual channels, that is, each physical channel is multiplexed into k virtual channels.

2. One tries to define a new routing function so that the associated channel dependence graph is acyclic. For reasons of homogeneity of the network, we multiplex each link in the same way, even though some virtual channels might be useless in some cases, as we have already seen.

Point 1 depends on physical conception of the machine, while point 2 depends on software. However, once the routing function is determined, it will be physically implemented in the router. In any case, nothing depends on the programmer.

To get a circuit-free dependence graph we label the channels so that the paths corresponding to a possible routing in the graph give rise to an increasing sequence of labels.

1.6.1.4 Routing function of common graphs

In this section, when we speak of k channels we mean that a physical link has been multiplexed into at most k channels. The graphs that model these networks are assumed to be symmetric (full duplex).

- Hypercube

  The routing algorithm which consists in successively changing dimension in the hypercube, in an order defined in advance, is deadlock-free [14]. Usually, the natural increasing dimension order is used, but any permutation of the dimensions is valid. If we allow virtual channels, we can choose a great number of paths by using adaptive routings.

- Oriented ring

  Simply take two virtual channels per link, as in the famous example in [14]. Note that not all channels are used.

- Ring

  Superposing the routing channels of the oriented case would mean using two virtual channels in each direction. A more detailed analysis, however, of the dependence graphs leads to using at most one channel in one direction and two virtual channels in the other.

  We consider two types of channels - the *out*-channels, directed clockwise, and the *in*-channels, directed counterclockwise. Each channel is divided into two virtual channels a and b.
  The dependence graph is then formed by an *in* component and an *out* component, each of which is a path (without circuit) as follows:

  $$in^a_{N-1},\ in^a_{N-2},\ \ldots,\ in^a_0,\ \ldots,\ in^b_{N-1},\ \ldots,\ in^b_{N-\lfloor\frac{N}{2}\rfloor+2}$$

  and

  $$out^a_0,\ out^a_1,\ \ldots,\ out^a_{N-1},\ out^b_0,\ \ldots,\ out^b_{\lfloor\frac{N}{2}\rfloor-2}$$

  Thus, we use the channels of type b

  $$out^b_0,\ \ldots,\ out^b_{\lfloor\frac{N}{2}\rfloor-2}\text{et}in^b_{N-1},\ \ldots,\ in^b_{N-\lfloor\frac{N}{2}\rfloor+2}$$

  Since $N-\lfloor\frac{N}{2}\rfloor+2$ is strictly greater than $\lfloor\frac{N}{2}\rfloor-2$, these two partitions are disjoint and we only need one additional queue to implement these virtual channels.

- General graphs

  If there is a concept of dimension in$G$, it is enough to solve the problem in each dimension, order the channels by dimension and use a routing function which respects this orders. Otherwise the first problem to solve is "Given a graph G and a shortest-path routing R for G, what is the minimum number of M of virtual channels needed in each link so that $\mathcal{D}(G,R)$ is acyclic?". The following theorem [7] gives an upper bound on this number. It adapts the buffer-pool technique used in store-and-forward [47].

Theorem 1.2 *For any network of diameter D there is a shortest-path routing function such that the channel dependence graph is acyclic if D virtual channels are used.*

In fact, as we shall now show, in an arbitrary network we can avoid deadlocks by using only two virtual channels in place of each physical link if we do not insist on shortest-path routings and if we ignore the load of nodes and channels. The constraints on the routing are relaxed in order to obtain a small maximum number of channels independent of the size of G (see [7]).

Theorem 1.3 *For each graph G there is a routing of G such that if we replace each channel by two virtual channels, the corresponding dependence graph has no circuits and each path is of length at most $2D$.*

In some cases we can do without virtual channels - it is enough that the two trees (or the anti-arborescence and the arborescence) have no common channels.

The routing can also be improved by not going all the way to the root if an intermediate vertex in phase 1 is on a branch to the destination p. Note, however, a disadvantage of this method – the root is a bottleneck.

Finally, several roots can be used, and in order to use more paths, we can look for directed acyclic graphs (DAG's) rather than arborescences.

1.6.2 Multi-casting and deadlocks

It happens frequently during the execution of a parallel program that communication does not simply consist of successive permutations of data among the processors (type one-to-one) but that the same message is sent from a processor to several others (type one-to-many). We call such a scheme *multi-casting*. *Broadcasting*, in which one processor sends the same message to all the others, is a special case of multi-casting which deserves a separate study.

Given a routing function R, static or adaptive, in a graph G, a multi-cast can be realized by a sequence of successive transmissions of the message to different destinations. If R is deadlock-free, several multi-casts can happen without deadlock. Such a protocol, however, will be slow – not only must each message be copied but the quantity of data circulating in the network becomes substantial.

Lin and Ni [37, 38] have suggested a different approach. It assumes two physical modifications.

- each router is assumed to be able to transmit and to copy into local memory the flits received
- each message can have several headers, the first containing the address of the first destination, the second of the second, etc.

Thus, if a processor wants to send a message M to k destinations numbered d_1, d_2, …, d_k, it puts in the header of M the addresses of the destinations in some order σ so that M is routed from the source to $d_{\sigma(1)}$, from $d_{\sigma(1)}$ to $d_{\sigma(2)}$, etc.

Several problems come up. For example, the choice σ is crucial for a fast multi-cast (an order of the destinations must be fast so that the path followed is the shortest possible). Furthermore, this kind of multi-cast introduces new channels dependencies and care must be taken so that these do not lead to deadlocks.

We can also mix this type of multi-cast with copying of messages and simultaneous transmission of several copies to several subsets of destinations. This problem has been studied in [19, 37, 38].

1.6.3 Fault tolerance

The number of components of a distributed memory parallel machine obviously grows with the number of processors. Thus, massively parallel machines will be likely to fail frequently if there are no provisions, software or hardware, to allow these machines to function even in case of faults in some of their components.

In this short section we only discuss some aspects of the area. We consider briefly the *vulnerability* and the *survival graph*. In either case, only structural aspects of the graph representing the interconnection network of the machine is taken into account. For other references the reader can look at the special issue on fault tolerance of *IEEE Transaction on Computers*, April 1990, and at [21] for references on communication with faults.

1.6.3.1 Vulnerability A first approach to the determination of the capacity of a network to tolerate faults is the study of the changes in the diameter of the graph when edges or vertices are deleted. We can consider what can be called a *Menger* property, that is, the existence of disjoint paths of length at most a given value between two vertices of the graph, so that, in case of a processor fault, we can get around it by a path which is not too long.

Let us call $f(t, D)$, the maximum possible diameter of an undirected graph obtained from a $(t+1)$-edge-connected graph of diameter D by eliminating t edges. Chung and Garey [13] proved the following:

Theorem 1.4 *For $D \geq 4$ we have*

$$(t+1)(D-2) \;\leq\; f(t,D) \;\leq\; (t+1)D+t$$

The result was extended by Peyrat [45] in the following theorem:

Theorem 1.5 *With the same notation we have*

$$f(t,2) \;=\; 4$$
and
$$3\sqrt{2t}-3 \;\leq\; f(t,3) \;\leq\; 3\sqrt{2t}+4$$

For some graphs we can do better, for example for hypercubes, de Bruijn graphs and Kautz graphs.

1.6.3.2 Survival graph Another concept of fault tolerance was introduced by Dolev, Halpern, Simons and Strong [17] for a network G endowed with a routing. The *survival graph* is a graph whose vertices are the faultless ones in G, two of which are joined by an edge if the routing path between them is untouched by a (vertex or edge) fault. The diameter of this graph then suggests the number of re-routings that a messages might go through in case of a breakdown. As a first approximation, the transmission time of a message will be multiplied by this diameter.

1.7 Operating Systems

A classical hypothesis made in performance evaluation of multi-processors is that elementary processing resources are dedicated to only one parallel program.

This assumption may fail in many modern high performance architecture, as:

- processors and/or memories may be dynamically allocated to jobs and may even have to be shared by several users.
- complex operating system kernels (schedulers, routers, memory managers, I/O handlers, etc.) may compete with user tasks for resources.

2 Some Existing Machines

In this section, we provide an insight of architectural choices made by the designers of some representative existing multi-processor machines, and particularly of their communication devices.

Let us emphasize that this list is by no means intended to be exhaustive. Its sole purpose is to give the reader an overview of the variety of the architectures currently available.

2.1 BBN Butterfly

GP1000

The Butterfly GP1000 is a shared memory multiprocessor housing up to 256 Motorola 68020s. Any processor can access any memory location through a multi-stage Butterfly-type switch, for a total memory bandwidth of up to 1024 MB/s and a memory access time under 4 microseconds.

TC2000

The BBN TC2000 incorporates up to 504 Motorola's 88000s and increases memory bandwidth up to 2.56 GB/s.

2.2 Convex Exemplar

The Exemplar series (SPP1000) machines from Convex Computer Corp. are made up of *hyper-nodes* connected by means of 4 SCI buses. Each hyper-node incorporates 8 HP/PA 7x00 RISC processors.

The memory architecture of this machine is of particular interest:

- each processor has both internal and external caches
- processors in a block of 2 share a local memory
- local memories are connected inside an hyper-node by cache coherency units and a crossbar
- crossbars are connected by the SCI buses.
- the programmer's view of the memory is that of a global shared memory.

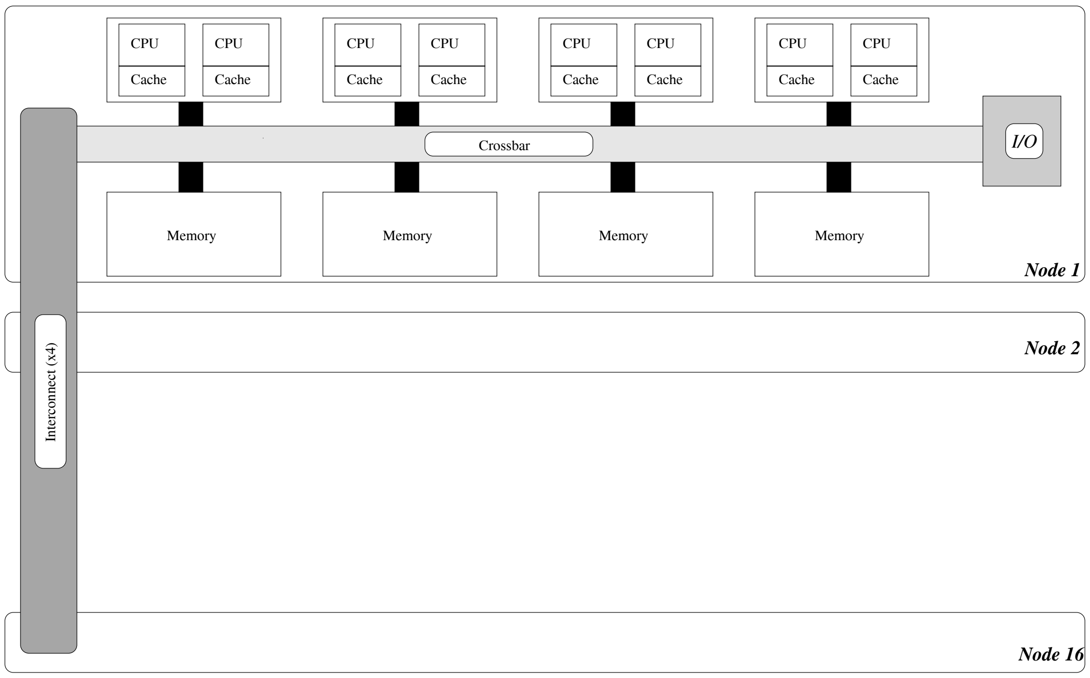

Figure 8: The Exemplar Architecture

2.3 Cray T3D

Cray Research Inc. [41] introduced the T3D. It has an MIMD architecture with distributed memory which, however, is globally accessible by all the processors. Synchronization methods allow the programmer to use a SPMD mode. Each node of the network consists of two DEC Alpha processors. The communication network is a three-dimensional toroidal mesh. Interprocessor rates are up to 300 MB/s, giving a bisection bandwidth up to 76.8 GB/s.

2.4 DAP (*Distributed Array of Processors*)

The DAP is an SIMD machine made by the English manufacturer AMT. It consists of 4096 1-bit processors connected in a 64×64 mesh. These processors have clock frequency of 10 MHz. The DAP with 4096 processors has a power of 560 MFlops. Communication is along the rows and the columns of the mesh. The total bandwidth of the machine is 5.2 GB/s. Further, AMT has developed communication and computation libraries which make the DAP particularly well suited for signal and image processing.

2.5 Fujitsu AP1000

This MIMD machine consists of up to 1024 nodes. A program is made up of one part which runs on the host (Sun workstation, which manages all the input and output) and of tasks which are loaded on the nodes. Each node is equipped with a Sparc processor and is connected to three distinct communication networks.

- The *T-net* is a torus which assembles the nodes. It can communicate at 25 MB/s and has links 16 bits wide. Routing between nodes is of wormhole type.

- The *B-net* is a network dedicated to broadcasting of messages from a node to all the others. It connects the nodes, by packets of 32, to a ring. Each packet has the structure of a tree whose root is a node of the ring. The B-net has a bandwidth of 50 MB/s thanks to which it can be considered as a shared bus, in spite of its multi-layer structure.

- The *S-net* establishes synchronization points between network nodes. It is a tree whose edges are signal-carrying lines and whose nodes are logical "AND" ports. Each node writes its signal to the S-net and receives in return an "AND" of all the nodes.

This machine efficiently uses the caches of the RISC processors by directly exchanging processor data with the router, without using DMA access [49].

2.6 IBM SP1

Having developed networks of RS/6000 workstations connected by fiber-optic connections, IBM produces a parallel machine, the SP1 (*Scalable Parallel*). The machine can have between 8 and 64 processors and a maximal power of 8 GFlops. Each processor is a Power processor with a clock frequency of 62,5 MHz and 125 MFlops of peak power. The SP1 has two communication networks. The first one connects the processors via Ethernet at 10 MB/s, or via FDDI (*Fiber Distributed Data Interface*) at 100 MB/s. The second one is a multistage Ω network. The links of this network work at 40 MB/s (full-duplex). The SP1 with 64 processors is made up of four clusters each containing 16 Power processors (grey rectangles in Figure 9) and two switch stages (small white rectangles in Figure 9). From each cluster go sixteen links: twelve to the three other clusters and four to external connections (disks, for example).

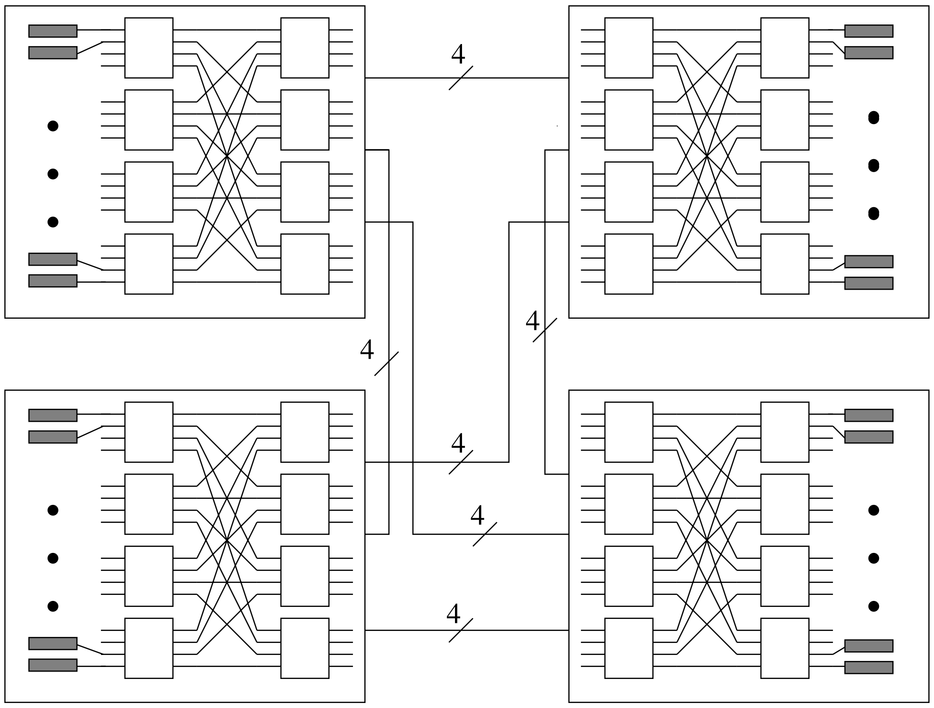

Figure 9: Interconnection network of the SP1 with 64 processors

2.7 Intel Paragon

This machine is a commercial version of the *Delta-Touchstone* project undertaken by Intel. It has between 56 and 4096 nodes physically connected in a two-dimensional mesh (see Figure 10) and delivers peak power of 4 to 300 GFlops. It is a distributed memory MIMD machine [27]. Each node consists of a computing processor i860 XP, a memory of up to 128 MB, and a second i860 XP in charge of message preparation at reception or emission. The second i860 frees the computation processor from this task. Each node also contains a PMRC (*Paragon Mesh Routing Chip*), with the set of the PMRCs making up the vertices of the mesh. The PMRCs route information in the network. Routing is done in wormhole mode. Each PMRC has four bidirectional links with its four neighbors and each link can handle 200 MB/s in full-duplex.

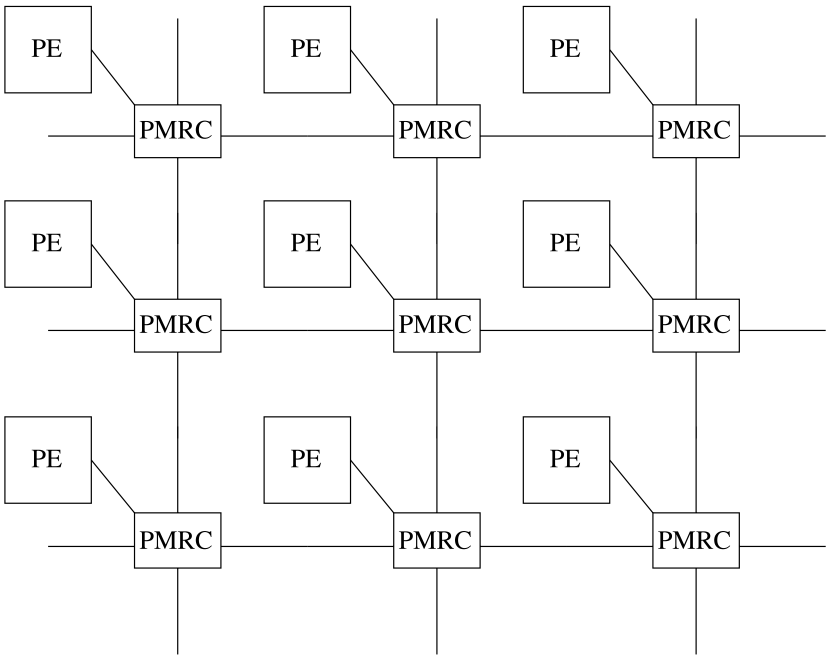

Figure 10: Network of the Paragon

2.8 Intel/CMU's iWarp

This is a machine for intensive real-time applications (signal processing and image processing). It is made up of 8 to 1024 iWarp cells, connected in a two-dimensional torus. Each cell is a VLIW processor (20 MIPS and 20 MFlops in single precision) which can execute up to nine operations in parallel. A communication module with four parallel bidirectional links gives the cell a total bandwidth of 320 MB/s ($4 \times 2 \times 40$ MB/s) [6]. Twenty virtual channels, multiplexed on the physical ports, can be managed in parallel. Routing is of wormhole type.

Parallel links (transmitting bytes) are difficult to use in intensive systems, as opposed to serial links (transmitting by bits) on the T800 (Inmos). There is a systolic communication mode which is very efficient when the algorithm lends itself to it and when the messages are well ordered.

2.9 KSR

The KSR made by Kendall Square Research is a machine that can have between 32 and 1088 processors and which delivers a total power of 43 GFlops. The processor network is a ring of rings (see Figure 11). On the machine with 1088 processors there is a first level made up of 32 processors, and a second level made up of 34 controllers, themselves connected in a ring. To each controller is connected a ring of the first level. It is an MIMD machine with a memory architecture called *All-Cache*, i.e., distributed on the network, but virtually shared by all the processors. In fact, each processor has a local cache memory, denoted by LC in Figure 11. The processors, made by Sharp, have a 64-bit RISC architecture and each is coupled to three co-processors, one floating point computation unit (FPU), one unit for integer and logical computation, and one co-processor to manage input and output. The processor has a power of 40 MFlops and 20 MIPS. Communication is done by access to memory sub-pages of size 128 KB [29]. When a processor wants to get data it does not have, it sends a request on its ring at level 1. If it is not successful, it passes to level 2. In return, the requesting processor gets its page.

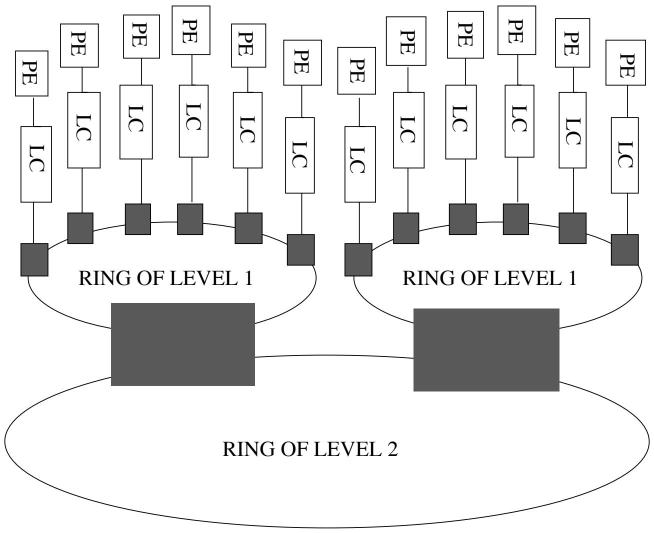

Figure 11: Ring of rings of the KSR

2.10 MasPar MP-1 and MP-2

The MasPar MP-1 is an SIMD machine whose processors form an octagonal toroidal mesh [8]. Each processor is connected to eight other processors in compass directions. The elementary processors are 4-bit, with base clock frequency of 12.5 MHz and 16 KB of memory each. The MP-1 interconnects between 1024 and 16384 processors. The MP-1216, i.e., the MasPar with 16384 processors, has a power of $1,5$ GFlops. Each processor of the mesh is connected to a sequencer ((or an ACU, *Array Control Unit*). This sequencer is a 4-bit processor with its own registers and its own memory for data and instructions. It is the sequencer that receives the executable code and that is responsible, during execution, for sending the code and the data to the processors in the mesh.

There are three types of communication on the MP-1.

- *X-Net* communication for all the active elementary processors to send a message to all the processors at a given distance, in the eight directions defined by the links.
- Router communication so that any two processors can communicate with one another. The routing is realized thanks to three levels of crossbars.
- "proc"-type communication to establish communication between the elementary processors and the ACU.

The MasPar MP-2 is an evolution of the MP-1 and has between 1024 and 16384 32-bit processors, each capable of 133 MIPS and 12.5 MFlops following the same principles. The MP-2 has a 40 MB/s communication bandwidth.

2.11 Meiko CS-2

The CS-2 is the last from the PCI consortium. It can interconnect up to 1024 processors. It is a distributed memory MIMD machine. The communication network used is the Ω network (see Figure 12). The switches that make up the network are $(8, 8)$ crossbars called Elite, developed by PCI. This component resembles the Inmos' C104 in its operation.

An Elite is connected to four nodes. Each node is formed by four processors - two Fujitsu vector processors of 200 MFlops each, a SuperSparc processor giving 150 SPECmark and 30 MFlops LINPACK, and a routing component called Elan, made by PCI. This processor has a RISC architecture and has the same instruction set as the SuperSparc.

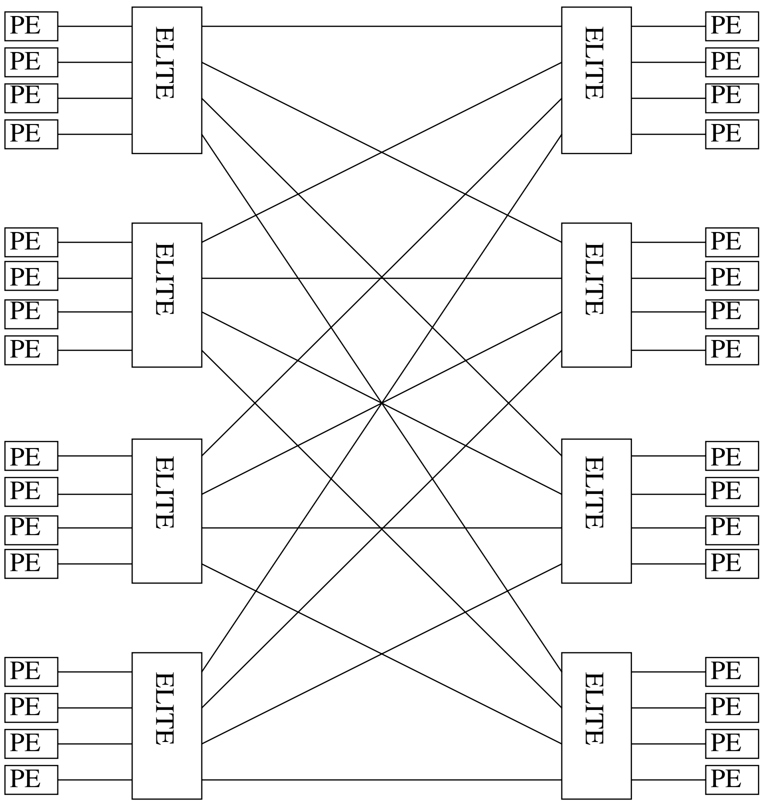

Figure 12: Interconnection network of the CS-2 with 32 processors

Communication mode is wormhole. As on the CM-5, broadcasting is done by going up to the top in the Elite component levels, before spreading throughout the whole network. Between an Elan and an Elite the throughput is 50 MB/s.

2.12 nCUBE

- **nCUBE 2**

  This machine can interconnect up to 8192 processors assembled in a hypercube (13-cube) and delivers a maximal power of 27 GFlops. It is a distributed memory MIMD machine. The elementary processors are made by nCUBE and have a peak power of 2.4 MFlops at clock frequency of 20 MHz. Each elementary processor can have up to 32 MB of memory. Each link can handle 2.22 MB/s [43].

  The nCUBE 2 has two special features. The first is that each elementary processor has a link directly connected to the outside, which facilitates input and output. The second is the existence of a global clock. Indeed, few MIMD machines have this and we can therefore introduce real synchronization points into programs.

  Routing is wormhole, hardware implemented.

- **nCUBE 3**

  The nCUBE 3 is the latest parallel computer of nCUBE. The machine is of MIMD type and interconnects up to 65536 64-bit processors. As in its predecessor, the processors are connected in a hypercube (16-cube), but this time nCUBE uses processors made by Hewlett-Packard. These processors run at 50 MHz and deliver in the order of 100 MFlops and 3 GIPS. Each processor has eighteen communication channels and 1 GB of local memory. The machine in its complete version has a theoretical peak power of 6.5 TFlops.

2.13 Networks of Workstations

DEC made the choice of a workstation cluster based on Alpha AXP chip. Regular stations are interconnected by Ethernet, FDDI or a FDDI-based switch, the *GIGAswitch*, that can operate at 3.6 GB/s. This high-speed crossbar has up to 22 FDDI ports. Several Gigaswitch units may be interconnected by ATM links.

2.14 Connection Machines

CM-2

The Connection Machine 2 [24] was manufactured by Thinking Machines Corporation (TMC). It consists of identical cards with 16 modules of 32 1-bit elementary processors running at 10 MHz. In its maximal configuration it has 65536 elementary processors. One module contains two circuits made up of 16 elementary processors and a communication unit, to which are added local memory and a floating point unit. The modules are connected to the host computer through a sequencer, by an instruction bus and an address bus. The circuits are interconnected by a hypercube-type network (of degree 12 for the maximal configuration). The internal connections of a group of 16 elementary processors are also realized by an interconnection network - a four-stage Butterfly [16]. Since most applications tested on the CM-2 are scientific, a floating point unit was added in order to speed up floating point number computations. On the latest versions of the machine (CM-200) it is possible to use processors working in double precision (64 bits) to improve the precision of computations. The performance gain thanks to the floating point unit is of the order of 20 % when compared with serial (bitwise addition) computation on the elementary processors.

Possible communications are of three kinds:

- General communication, which allows to send messages from one processor to another. It uses an internal communication unit (a router on the CM-2) and induces a substantial performance loss.
- *NEWS* (North, East, West, South) communication for sending messages to a neighboring processor in the chosen topology. Since the processors execute the same instructions, there is no link conflict. Such communication is, therefore, very fast.
- Structured communication for communication between processors in the same dimension of the chosen topology. They combine communication and binary composition operations on the messages. Such communication uses the hypercube links connecting all the circuits (group of 16 processors).

CM-5

This is the latest machine from Thinking Machines Corporation (TMC) [52]. It can interconnect between 32 and 2048 processors. It is an MIMD machine with distributed memory. In its current configuration, it has a maximum power of 262 GFlops.

The elementary processors consist of a Sparc processor, 32 MB of memory, and four floating point units of 32 MFlops each.

One of the main originalities of the CM-5 lies in the existence of three networks. The Data Network manages point-to-point communication, the Control Network takes care of global operations (such as broadcasting, reductions, synchronizations, etc.) by pipelining the messages. Finally, the Diagnostic Network transmits error messages.

The topology adopted for the data network is a *fat tree* (see Figure 13), defined by Leiserson in [33]. The leaves are the processors, and the intermediate nodes are the controllers/routers. Such a topology gives the machine good fault tolerance properties by duplicating each link and each controller/router.

Each cell of the routing has four children and two or four parents, depending on the number of levels of the network (or, more precisely, depending on the number of processors). A link of the controller has a throughput of 20 MB/s and the four links can function in parallel.

The throughput of the network depends on the proximity of the elementary processors, that is, on the number of controllers through which the message must pass. For one controller, the throughput is 20 MB/s ; for two controllers, it is 10 MB/s ; for more than two it falls to 5 MB/s. For point-to-point communication,

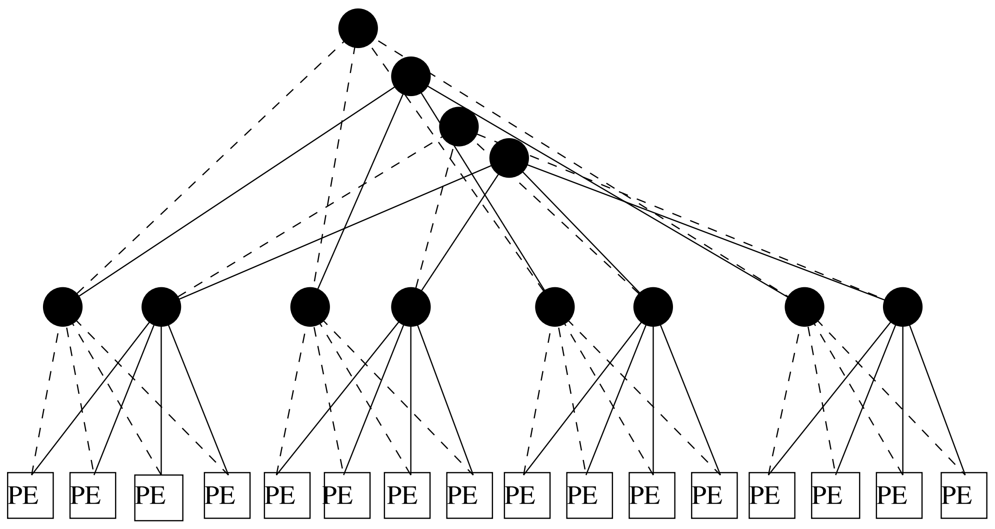

Figure 13: Fat tree of the data network in a CM-5 with 16 processors

a message uses the following technique. It goes up through the network of controllers/routers randomly until it finds one from which it can descend to its destination deterministically. The global bandwidth is 0.5 GB/s.

The topology of the control network is much simpler. It is a complete binary tree [34] whose leaves are the processors and whose other nodes are the controllers. The network realizes all global operations (broadcasting, synchronization, logical "OR", "XOR", etc.). The algorithms used are very simple. For broadcasting, for example, the message goes up to the root controller before descending along all the branches.

2.15 Transputers

T9000

The T9000 is the processor that replaces the T800 in the Inmos Transputer family. As the T800, it is intended as a building block for the construction of MIMD machines. Each of its four bidirectional links can be connected to a specific routing component, the C104. The T9000 is a 32-bit processor with a 64-bit floating point unit and its claimed performance is 200 MIPS and 25 MFlops. Its architecture is super-scalar and includes a pipeline. An instruction grouper schedules waiting instructions [46] in order to execute some of them in parallel, but without modifying the order in the instruction flow.

As in the T800, there is an integrated scheduler thanks to which the context switching overhead is modest (in the order of a microsecond). Furthermore, inactive processes do not consume CPU time. This allows for efficient parallel programs, in OCCAM or in C [11, 35].

The main developments in comparison with the T800 are the following:

- clock frequency goes from 33 to 50 MHz ;
- the throughput of each communication link goes from 2.5 MB/s to 12.5 MB/s.
- virtual channel management which facilitates programming. An internal unit, the *Virtual Channel Processor*, manages the multiplexing of the physical links and the sending of messages, locally or via the C104. The messages are put in a queue so as not to block the CPU. The VCP manages the four links of the T9000 and their DMA controllers.

C104

The C104 is a routing component containing a non-blocking 32×32 crossbar for interconnecting the links of the T9000 or the links of the C104. Messages transmitted on these links are split into packets of 32 bytes and are routed in wormhole mode in the C104 [36]. Such packet routing facilitates the management of virtual channels allowed by the T9000, Thus, messages can be multiplexed on the links by the Virtual Channel Processor.

The concept of *wormhole routing* used here is somewhat different from that used in point-to-point networks. There are no links in the C104 and no intermediate nodes to pass. It differs from circuit switching in that we do not wait for a receipt in this switching mode; data is transmitted at the same time as the route is established.

The C104 communication protocol helps to avoid deadlocks. It assigns an interval to each output link so as to select a link depending on the destination address of the message. The labeling algorithm does not always lead to an optimal route, but it performs well on current topologies. To avoid bottlenecks in the network, we can also use a universal routing mode [53]. These two functions are provided by internal components of the C104, an *Interval Selector* and a *Random Header Generator.*

The programmable crossbars can be connected either to T9000s or to other C104s in order to construct bigger networks. Simulations by Inmos give average transmission delays between two nodes of 27 to 64 μs, depending on the size of the networks (hypercubes of 64 to 16384 nodes).

3 Some Modeling Challenges

3.1 Elementary Communications Modeling

In order to get performance evaluations of parallel programs, we must define a model of elementary communications between any pair of nodes in the network. This will give basic communication cost parameters to be incorporated into a general model of a parallel application.

3.1.1 Constant time model

In this model we assume that a communication between two processors “costs” one time unit and is independent of the length of the transmitted message.

Messages can be cut and reassembled without affecting transfer time. For example, if a node received two messages, it can combine them into one and the cost of sending it will still be 1 even if it has become huge. Conversely, a message can be cut into several packets but the transmission of each of them will cost 1 even though they can be much smaller than the initial message. We do not take into account the time of memory manipulation.

3.1.2 Linear time model

The transmission time of a message of length L between two neighboring processors is given by the sum of

- *start-up* (or initialization) time, denoted by β, which is the time it takes to initialize the memory registers or the time of receipt procedures, and
- propagation time $L\tau$, directly proportional to the length L of the message.

Hence the expression $T(L) = \beta + L\tau$, where the parameter τ is the propagation time of one unit of the message (often a bit or a byte). The bandwidth of a link is then $\frac{1}{\tau}$.

The linear time model better approximates the behavior of a distributed machine than the constant time model.

3.1.3 Communication with nodes at distance d

In certain routing modes, for example circuit-switching or wormhole, one can send a message directly to a node at distance d greater than 1 without repeating d times the process of neighbor-to-neighbor communication. A possible model is ([9, 44]) $T(L, d) = \alpha + d\delta + L\tau$ where L is the length of the message and $\frac{1}{\tau}$ is the bandwidth of the links. The parameter α is the start-up time of the sending processes. The delay δ is the time related to the router switching at each intermediate node.

In order to compare the last two models, observe that a distributed machine with circuit-switching or wormhole routing can also communicate neighbor-to-neighbor. Hence $\beta = \delta + \alpha$.

3.2 Global Communications

In addition to point-to-point communications, parallel applications exhibit several kinds of *communication patterns*:

- **Broadcasting** (*one to all*): an operation of sending a message from a single source to all processors.
- **Gossiping**(*all to all* or *total exchange*): an operation of broadcasting from all the processors simultaneously.
- **Scattering** (*personalized one to all* or *distributing*): an operation of sending distinct messages to all the other processors from a single source. This differs from broadcasting in that the messages sent are not the same throughout the communication process.

  One should also consider the inverse operation, *gathering*, which allows one processor to retrieve various data from the others.
- **Multiscattering** (*personalized all to all* or *complete exchange*): an operation of scattering simultaneously from each processor; each processor sends different messages to all the others.

Clearly, there are other kinds of global communication, for example *one-to-many* or *multicasting*, where a processor sends the same message to a group of processors. In particular, this happens when a processor sends a message to all its neighbours.

3.3 Communications Modeling and Benchmarking

One of the major concerns in the modern approach of modeling communications is the choice of proper hypotheses for the incoming traffic. In the classical way of modeling communication systems using queuing networks, it is customary to assume that the "customers" of the network arrive according to Poisson processes, or maybe more general processes described by simple laws on the "inter-arrivals". Recent studies tend to show that such models are not suited at all for modeling certain aspects of wide areas of tomorrow, in particular the loss probabilities of packets in packet switching networks.

The issue is then to find out what is a correct way to model incoming traffic of packets, with however the constraint that the resulting model should keep some mathematical and numerical tractability. Some models using classes of so-called "Markov modulated" processes have already been studied. It remains to assess the relevance of these studies in the field of networking for parallel applications.

Indeed, a point is that, in most of the architectures described in section 2, the network hardware is very different of that of usual communication networks. Another point is that the traffic generated by distributed programs on a multiprocessor system may well be completely different from that of wide area, broadband ISDN networks.

Another important issue, often neglected, is to devise techniques to extract the parameters of the models from a particular problem.

Acknowledgments

The authors would like to thank the members of the working group RUMEUR for their work on communications problems in processors networks and for their outstanding survey books [28, 15] in this field.

References

[1] S. G. Akl. *The Design and Analysis of Parallel Computers.* Prentice-Hall, 1989.

[2] G. Almasi and A. Gottlieb. *Highly Parallel Computing.* Benjamin Cummings Publishing Co., 1989.

[3] G. S. Almasi. Overview of parallel processing. *Parallel Computing*, 2:191–203, 1985.

[4] M. Auguin and F. Boeri. Réseaux d'interconnexion et leurs commandes asynchrones. *Parallélisme, communication et synchronisation, Editions du CNRS*, 1985.

[5] F. Baccelli, editor. *Proceedings of the QMIPS Workshop on Stochastic Petri Nets, Sophia–Antipolis*, November 1992.

[6] B. Baxter and B. Greer. Apply: a parallel compiler on iWarp for image-processing applications. In IEEE Computer Society Press, editor, *Proceedings of The 6th Distributed Memory Computing Conference*, pages 186–193, 1991.

[7] J-C. Bermond and M. Syska. Routage wormhole et canaux virtuels. In M. Cosnard, M. Nivat, and Y. Robert, editors, *Algorithmique Parallèle*, pages 149–158. Masson, 1992.

[8] T. Blank. The MasPar MP-1. *IEEE Transactions on Computers*, pages 20–24, 1991.

[9] R. Boppana and C. Raghavendra. All-to-all personalized communication on circuit-switched hypercubes. Technical report, Dept EE-Systems, USC, Los-Angeles, 1990.

[10] O. J. Boxma and G. M. Koole, editors. *Proceedings of the QMIPS Workshop on Solution Techniques, Turin*, number 105 & 106 in CWI Tracts, September 1993.

[11] A. Burns. *Programming in OCCAM-2.* Addison Wesley, 1988.

[12] Gilbert Cabillic, Thierry Priol, and Isabelle Puaut. MYOAN : an implementation of the KOAN shared virtual memory on the INTEL PARAGON. RR 2258, INRIA, April 1994.

[13] F.R.K. Chung and M.R. Garey. Diameter bounds for altered graphs. *Journal of graph theory*, 8:511–534, 1984.

[14] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[15] Jean de RUMEUR. *Communications dans les réseaux de processeurs.* Masson, Paris, 1994.

[16] D. Delesalle, D. Trystram, and D. Wenzek. *Tout ce que vous voulez savoir sur la Connection Machine.* Laboratoire de modélisation et de calcul, Grenoble (France), 1991.

[17] D. Dolev, J. Halpern, B. Simons, and R. Strong. A new look at fault-tolerant network routings. *Proc. ACM*, 16:526–535, 1984.

[18] J. Dongarra and I. S. Duff. Advanced architecture computers. Technical report, The University of Tennessee, (USA), 1990.

[19] J. Duato. A new theory of deadlock-free adaptative multicast routing in wormhole routing. Technical report, University of Valencia, Spain, 1993.

[20] M. J. Flynn. Very high speed computing systems. *Proc. IEEE*, 54:1901–1909, 1966.

[21] P. Fraigniaud. Asymptotically optimal broadcasting and gossiping in faulty hypercubes multicomputers. *IEEE Transactions on Computers*, 41(11):1410–1419, 1992.

[22] C. Germain-Renaud. *Etude des mécanismes de communication pour une machine massivement parallèle : MEGA.* PhD thesis, Université de Paris-sud, Centre d'Orsay, 1989.

[23] N. Götz, U. Herzog, and M. Rettelbach, editors. *Proceedings of the QMIPS workshop on Formalisms — Principles and State-of-the-Art*, volume 26 of *Arbeitsberichte des IMMD.* University of Erlangen, September 1993.

[24] W. D. Hillis. *The Connection Machine.* MIT Press, 1985.

[25] R. W. Hockney and C. R. Jesshope. *Parallel Computers 2, Architecture, Programming and Algorithms.* IOP Publishing ltd, 1988.

[26] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, 1984.

[27] Intel Corporation. *PARAGON XP/S, product overview*, 1991.

[28] editor J. C. Bermond. *Interconnection Networks*, volume 37,38. Discrete Applied Mathematics, 1992.

[29] Kendal Square Research. *Technical summary*, 1992.

[30] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communication switching technique. *Computers Networks*, 3:267–286, 1979.

[31] Zakaria Lahjomri and Thierry Priol. KOAN: a shared virtual memory for the IPSC-2 hypercube. RR 1504, INRIA, September 1991.

[32] F. T. Leighton. *Introduction to parallel algorithms and architectures.* Morgan Kaufmann, 1992.

[33] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.

[34] C. E. Leiserson. The network of the CM-5. In ACM, editor, *SPAA 92*, 1992.

[35] Inmos Limited. *OCCAM2 : Manuel de référence.* Masson, 1989.

[36] Inmos Limited. Special T9000. *La Lettre du Transputer (hors série)*, 1991.

[37] X. Lin and L. Ni. Multicast communication in multicomputers networks. In *International Conference on Parallel Processing'90*, pages III–114–III–118, 1990.

[38] X. Lin and L. Ni. Deadlock-free multicast wormhole routing in multicomputers networks. In *18th Annual Internatinal Symposium on Computers Architecture*, pages 116–125. ACM, 1991.

[39] D. H. Linder and J. C. Harden. An adaptative and fault tolerant wormhole routing strategy for $k-$ary $n-$cubes. *IEEE Transactions on Computers*, 40(1):2–12, 1991.

[40] INMOS Ltd. *The T9000 Transputer Products Overview Manual.* SGS-Thomson Microelectronics, 1991.

[41] T. McDonald. *The Cray research MPP FORTRAN programming model.* Cray Research Inc., 1992.

[42] P. Merlin and P. Schweitzer. Deadlock avoidance in store-and-forward networks-I : store-and-forward deadlock. *IEEE Transactions on Communication*, 28:325, 1980.

[43] nCUBE. *Technical overview system - nCUBE 2*, 1990.

[44] J. G. Peters and M. Syska. Circuit-switched broadcasting in torus networks. Technical Report CMPT TR 93-04, Simon Fraser University, 1993.

[45] C. Peyrat. Diameter vulnerability of graphs. *Discrete applied math.*, 9:245–250, 1984.

[46] D. Pountain. Virtual channels: The next generation of Transputers. *Byte*, April 1990.

[47] E. Raubold and J. Haenle. A method of deadlock-free resource allocation and flow control in packet networks. In *Proceedings of ICCC 1976, Toronto, Canada*, page 483, 1976.

[48] C. L. Seitz. Concurrent architectures. In R. Suaya and G. Birtwist, editors, *VLSI and Parallel Computation*, pages 1–84. Morgan Kaufmann, 1990.

[49] T. Shimizu, T. Horie, and H. Ishihata. Low-latency message communication support for the AP1000. In ACM SIGARCH, editor, *The 19th International Symposium on Computer Architecture*, pages 288–297, 1992.

[50] H. J. Siegel. *Interconnection networks for large-scale parallel processing.* Lexington books, 1985.

[51] H. S. Stone. *High-Performance Computer Architecture.* Addison-Wesley, 1987.

[52] Thinking Machines Corporation. *CM-5*, 1992.

[53] L. G. Valliant. General purpose parallel architectures. Technical Report 07-89, Harvard University, (USA), 1989.