Chapter 17 - Distributed memory MIMD architectures

17.1 Introduction

17.1.1 Solution for memory and synchronisation latency

Distributed memory MIMD architectures are called message-passing computers. This name reflects the solution offered by this type of architectures to the problem of accessing remote memory and hiding both memory and synchronisation latency. The basic idea in these machines is that processes are allocated along with the needed data area in local memory and hence remote memory references are avoided. Whenever access of a remote memory location becomes necessary its content is requested by sending a message to the process owning that memory area. Meanwhile the response is produced, the requester process can be handled in two different ways:

1. The requester process is suspended and its processor is allocated for another process.

2. The requester process can continue (doing other useful computations) until a certain point when it needs the result of communication. At this point it waits for the reply message in a busy-wait way.

The first approach requires efficient process switching mechanism similar to thread switching of multithreaded systems and sufficiently large number of processes per processors to replace the suspended processes with active ones. In case of the second approach efficient software and hardware support is needed to poll pending communications.

In order to reduce the frequency of interprocessor communication message-passing computers raise two new important questions:

1. How to partitioning the parallel program into processes for minimising the interprocess communication?

2. How to map processes on processors in order to minimise interprocessor communication?

The answers for these questions are not trivial, they lead to NP-complete problems and though they stimulated many research efforts in the last decade generally accepted algorithms are not available yet. Even if satisfactory solutions could be found, the problem of load balancing immediately appears. An initial mapping, optimal from the point of view of interprocessor communication, can result in unbalanced system having

idle and overloaded processors at run time. These are the problems that turned parallel computer designers to distributed shared memory systems where these problems do not appear or more correctly they are automatically solved by the underlying hardware/software system. Nevertheless, message-passing computers are still popular and can deliver the best peak performance for problems containing large amount of parallelism.

17.1.2 Design space

Recall that in multithreaded machines the main design issue was to reduce the time and frequency of thread switches. Naturally process switching time and frequency is not negligible in message passing architectures either but they are less dominant than the following two factors:

1. frequency of communication
2. processor time spent on communication

Accordingly, the focus point in designing message passing parallel computers is the organisation of the communication subsystem, i.e. the interconnection network of processing elements and the hardware support for passing messages among nodes of the multicomputer. The frequency of communication can be reduced only by software technology using efficient partitioning, mapping and load-balancing techniques. From the point of view of hardware design the main concern is the reduction of processor time spent on communication.

A multicomputer consists of a set of nodes and a communication network connecting the nodes. Communication among nodes is typically realised via point-to-point or direct connections called **links** or **channels**. Such networks are called **direct or static networks** and can be represented by a **communication graph** in which vertices correspond to the nodes of the multicomputer and edges represent channels. The nodes of message-passing architectures consist of three main components:

1. Computation processor + private memory (PE)
2. Communication processor
3. Router (or switch unit)

The role of both the communication processor and the router is to organise communication among the nodes of the multicomputer. Since the size of messages can vary depending on the application, messages are divided into **packets**. On the sender node it is the task of the communication processor to packetise messages and similarly depacketisation is performed by the communication processor of the receiving node.

When the message is sent among nodes that are not directly connected (remote nodes), intermediate nodes should transmit the message to the next node of the **communication path** established between the source and destination nodes. Establishing such a path is the task of the communication processor of the source node or the router depending on the actual routing protocol. Transmitting messages via the communication path is always the task of the router.

Multicomputers can be classified according to:

1. how the interconnection network is organised
2. how the three components of nodes are composed and integrated
3. what is the message passing computational model supported by the multicomputer
4. what is the optimal grain of computation supported by the multicomputer

The interconnection network design contains three main considerations. Firstly, the **topology** of the communication graph is an important characteristics of the interconnection network which significantly influences the message transmission time in the multicomputer. Accordingly, one of the main design decisions in message passing architectures is the selection of a proper network topology. Secondly, a strongly related question arises: what kind of **switching technique** is to be used at the nodes of the network. Switching is the actual mechanism by which messages are transmitted from input buffers to output buffers. Finally, **routing protocols** play crucial role in finding communication paths between non-neighbour source and destination nodes.

Clearly, according to the organisation of interconnection networks and nodes three generations of multicomputers can be distinguished. In first generation message-passing computers the processing elements were directly connected, communication processors and switch units were missing (Figure 1.a). Their functions were provided by the processing element via low-level software layers (for example machines built of Transputers). In second generation multicomputers independent switch units have been introduced and separated from the processor. The generic interconnection schemes of these machines are either decentralised (for example iPSC/2, Figure 1.b) or centralised (for example Parsys Supernode, Figure 1.c). In current third generation message-passing architectures the three main components are clearly distinguishable and for each component a separate, eventually specialised processor is applied (for example Meiko CS-2). Figure 1.d depicts the node organisational concepts of third generation multicomputers.

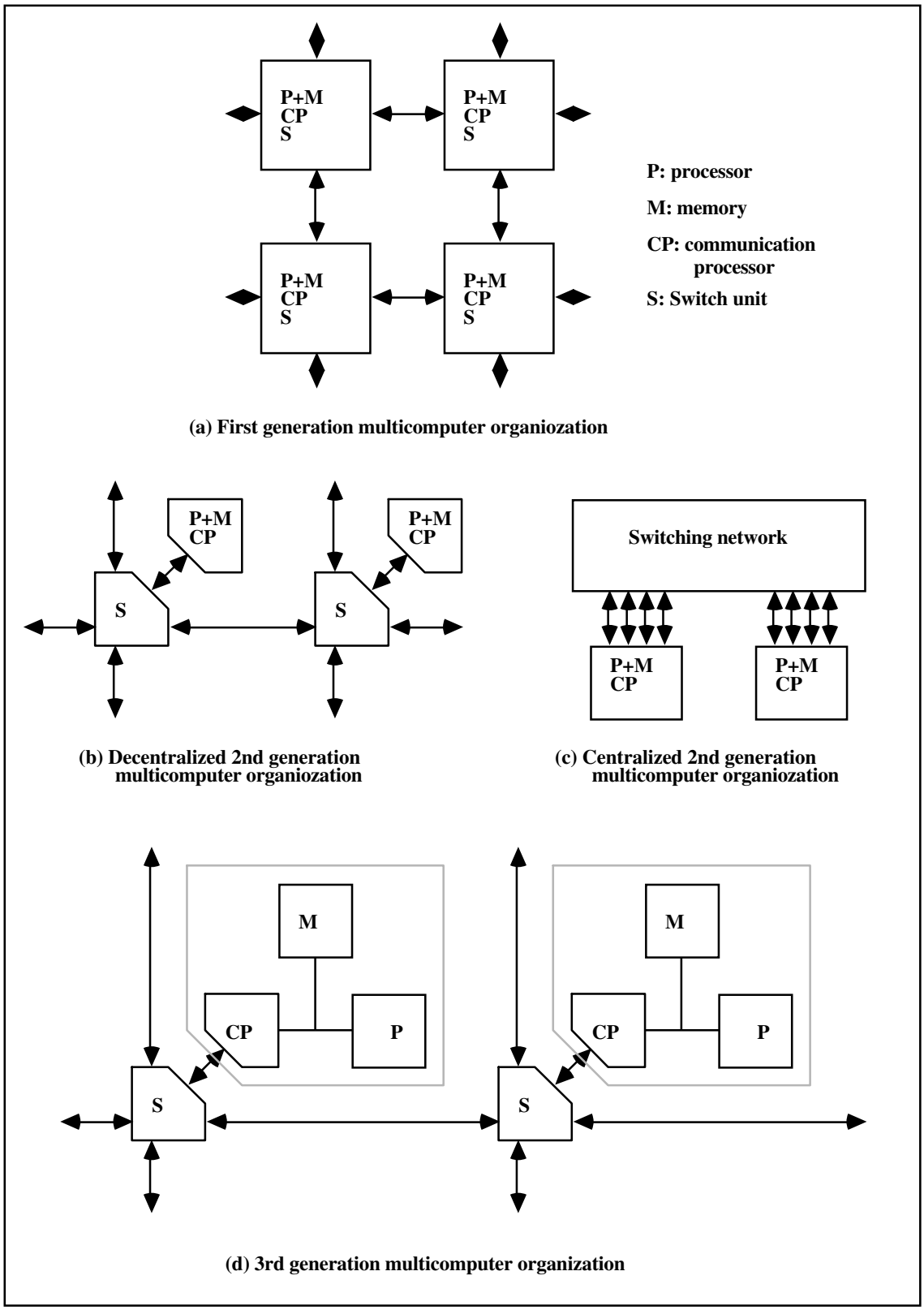

Figure 1. Node organization in different generations of multicomputers

Concerning the computation processor of the nodes it could be either a commodity microprocessor or a special custom designed one. Employing off-the-shelf microprocessors significantly reduces the design cost of multicomputers, though the

performance of such machines might be lower than those including custom designed chips. Another aspect of computation processors, if they are supported by special vector processors or not, is related with computing power, too. Interestingly, only few commercial multiprocessors were enhanced with vector processors (Intel iPSC/2, Meiko CS-2), though the lack of vector processors obviously reduces their performance comparing to other parallel architectures.

The third important issue in designing multicomputers is the computational model to be supported. Three main directions can be distinguished in this respect. The first, very pragmatic approach wants to apply conventional programming languages (like C and FORTRAN) extending them by special communication libraries. The theoretical foundation of this concept is quite weak, leading to error prone programming styles. Nevertheless, this is the most popular direction followed by most of the supercomputer vendors. The second approach has a sound theoretical foundation, called CSP (Communicating Sequential Processes) which was introduced by Hoare (Hoare, 1978, 1985) and expanded towards a practical programming language Occam by Inmos (Inmos, 1988a). Occam has also been implemented in hardware resulting in a new message-passing building block chip called the Transputer (Inmos, 1988b). Though the general judgement of the Transputer is very controversial, it is undeniable that it had a great influence on the progress and distribution of parallel computers. Its success came from the elegant parallel program constructs it supported and the hardware simplicity of directly connecting Transputers into a larger multicomputer system. All of these advantages were an obvious result of the well-founded theoretical model of CSP. The third approach followed at MIT by the Message-Driven Processor (MDP) supports two nonconventional programming models for parallel computer architectures. These two models are the dataflow and an actor based object-oriented models (Dally et al., 1987). Though many similarities can be found in the Transputer and the MDP concept, they mainly differ in the supported computational model. While the Transputer relies on a static, synchronous model of parallel programming, the MDP is based on a dynamic, asynchronous model.

The granularity of multicomputers is strongly related with the supported model. Coarse-grain architectures are sufficient to implement conventional languages on multicomputers. The CSP model and its hardware/software implementation, the Transputer/Occam system, represents medium-grain parallelism. The dataflow and actor based object-oriented models can be efficiently implemented on fine-grain systems, as it was proved by the MDP project run at MIT.

The design space of multicomputers is summarised by Figure 2. The relationship of computational models, grain size and multicomputer generations is depicted in Figure 3. This figure demonstrates that three main lines of multicomputers were developed through

three generations. The first line representing the conventional approach has two branches: one for the computers built on commodity processors and another one where custom designed processors are employed. In the current chapter the properties of the direct interconnection networks are studied first, then the three main directions of multicomputers are investigated by concentrating on their third generation representatives.

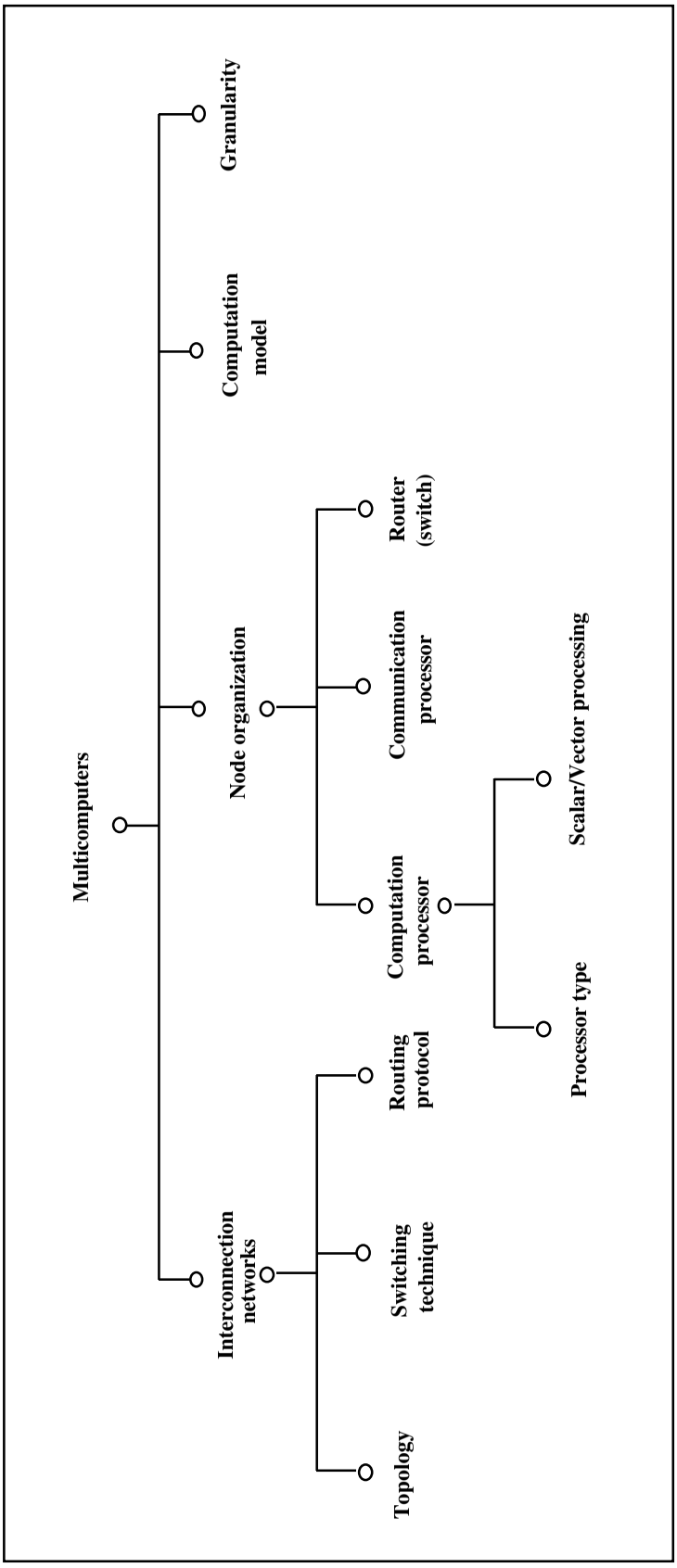

Figure 2. Design space of multicomputers

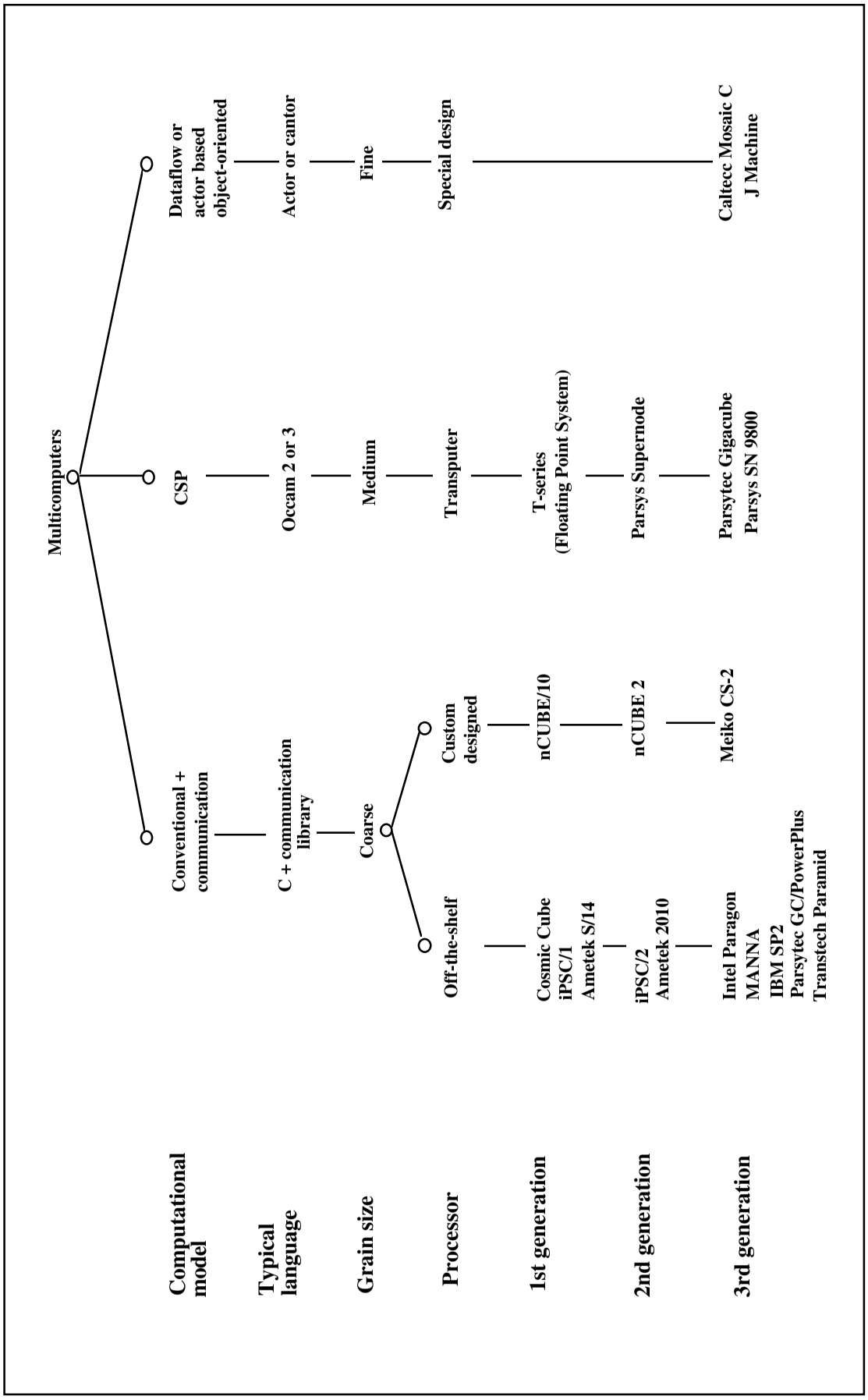

Figure 3. Classification of multicomputers

17.2 Direct interconnection networks

Direct interconnection networks are characterised by the following main measures:

1. **Network size (N)**: the number of nodes in the network.

2. **Node degree (d)**: the number of input and output links of a node. The node degree represents the cost of a node from the communication point of view and hence those network topologies are favourable where the node degree is low. Another important aspect of the node degree is related to scalability. Those networks where the node degree does not depend on the network size are scalable.

3. **Network diameter (D)**: Let S be the set of shortest paths between all two nodes of the network. D is the number of connecting arcs along the longest path of S. The network diameter is important from the point of view of latency. In order to achieve low latency the diameter should be kept as small as possible.

4. **Bisection width (B)**: the minimum number of links that should be removed to break the network into two equal halves. This parameter reflects the wiring density of a network and is strongly related to the communication bandwidth of the network if it is multiplied by the channel bandwidth:

bisection bandwidth = bisection width x channel bandwidth

where

channel bandwidth = channel width x channel rate

Here **channel width** is the number of bit wires within a link connecting two nodes and **channel rate** is the peak rate at which a single physical wire can deliver bits.

5. **Arc connectivity (A)**: the minimum number of arcs that should be removed from the network to cut it into two disconnected networks. Higher connectivity is better since it reduces the contention for links.

6. **Cost (C)**: the number of communication links required by the network.

In the design of direct interconnection networks three design choices should be made: network topology, switching technique and message routing. At first sight they seem to be largely orthogonal, however, the progress of generations of multicomputers demonstrates that switching techniques have influenced the choice of network topology.

17.2.1 Interconnection topologies

The main six parameters enumerated previously represent conflicting requirements: Scalability requires large number of nodes with small and constant degree on a low cost, i.e. with possibly small number of links. However, low latency and high bandwidth

require higher node degree and large number of links. These contradictory requirements lead to a compromise in static network design decisions and particularly in the selection of the interconnection topology.

Static interconnection topologies can be classified according to their dimensions:

1. one-dimensional topologies
2. two-dimensional topologies
3. three-dimensional topologies
4. hypercubes

The design space for direct network topologies is shown in Figure 4. and the main static interconnection topologies are depicted in Figure 5. A brief overview of the main topologies are given below.

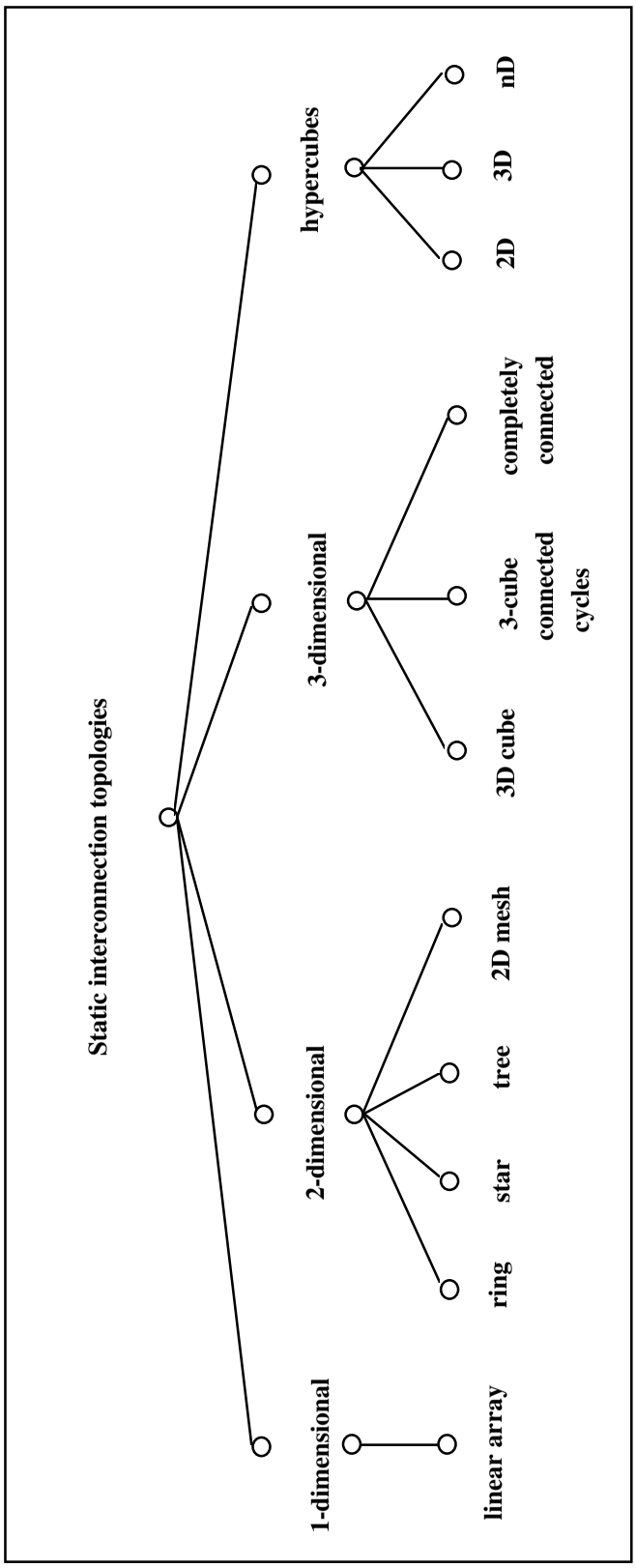

Figure 4. Design space of static interconnection topologies

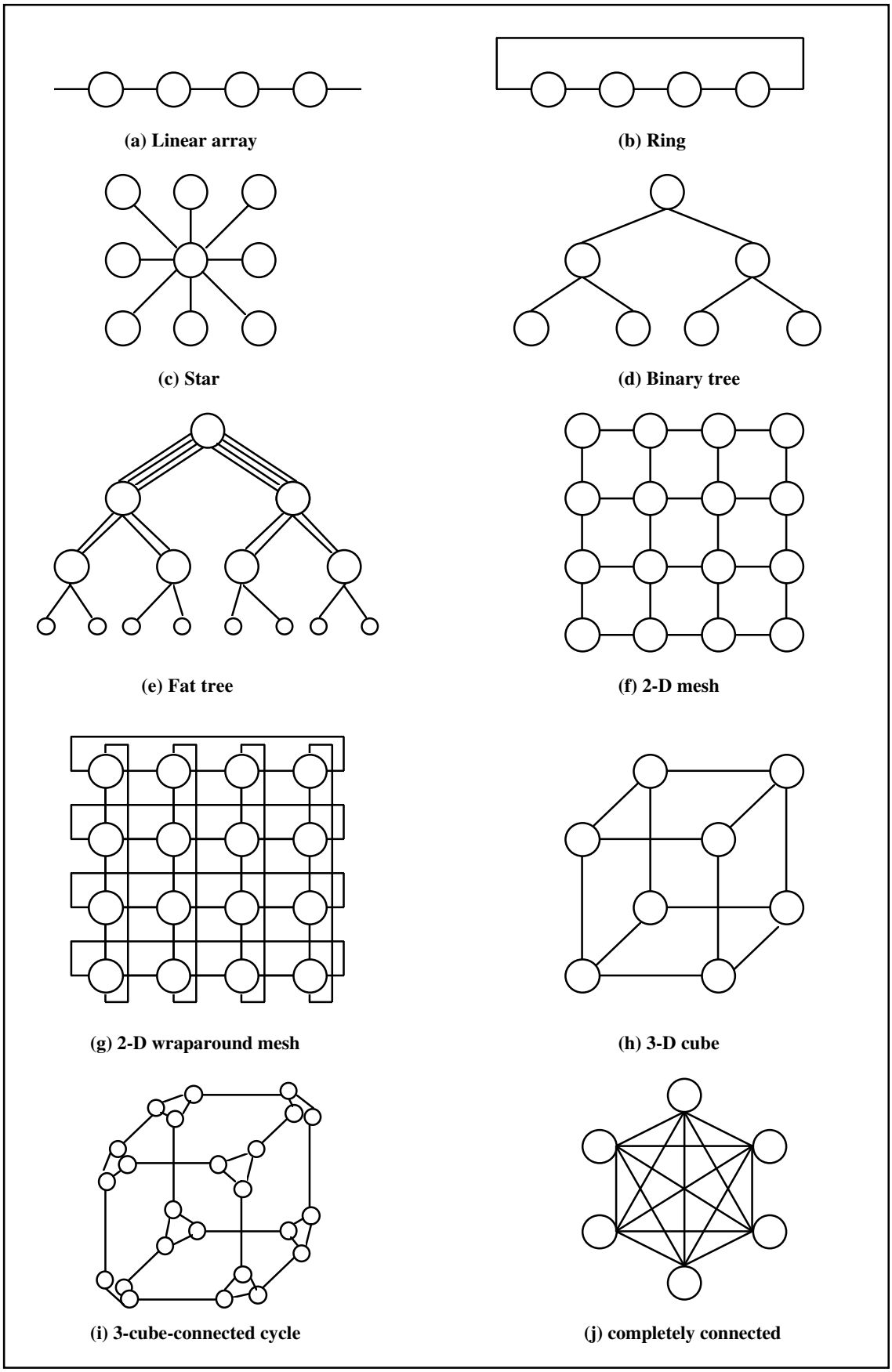

Figure 5. Main static network topologies

Linear array The simplest way to connect nodes is the linear array topology (Figure 5.a) which can be advantageously used in pipelined algorithms. This is a low cost network (C=N-1) requiring low node degree (d=2) but has many drawbacks. The diameter is the worst of all the possible topologies (D=N-1) and both the bisection width (B=1) and arc connectivity (A=1) parameters are very poor.

Ring By applying a simple wraparound connection between the processors at the ends of a linear array one can get the ring topology (Figure 5.b) which improves the poor parameters of the linear array by a factor of two: D=N/2, B=2 and A=2. Recently the ring topology has become popular, particularly at various levels of hierarchical topologies. For example, in KSR1 rings are connected in a hierarchy, in Convex Exemplar the so-called hypernodes are connected by multiple parallel rings.

Star The star connection is another simple and low cost (C=N-1) topology (Figure 5.c) which has the same poor bisection width (B=1) and arc connectivity (A=1) parameters as the linear array but the diameter is radically improved (D=2). From the point of view of both scalability and fault-tolerance the central node of the star topology represents a critical point.

Tree The **binary tree** topology (Figure 5.d) takes a significant step towards reducing the network diameter (D=2log((p+1)/2)) without sacrificing the constant node degree (d=3). Other advantage of the tree topology is that it fits well to many important parallel algorithm classes. Accordingly, the binary tree architecture gains popularity time to time. The main drawback of the binary tree network is the poor bisection width (B=1) and arc connectivity (A=1). The **fat tree** network (Figure 5.e) tries to eliminate this problem by duplicating the number of links at each level going upwards in the tree and also duplicating the number of links connecting a node to a higher level node. The price for this improvement is that the node degree becomes variable and the network cost significantly grows. Nevertheless, recently the fat tree topology has gained popularity. For example CM-5 and CS-2 employ fat tree networks.

2-D Mesh One of the most popular topology is the mesh (Figure 5.f) since it represents a good compromise among the contradictory requirements of static network parameters. It assures a shorter diameter ($D=2(N^{1/2}-1)$) than the binary tree and a significantly improved bisection width ($B=N^{1/2}$). Even the arc connectivity (A=2) is doubled comparing to the binary tree. The other side of the coin is that the node degree (d=2, 3 or 4) and network cost ($C=2(N-N^{1/2})$) are slightly higher than in the binary tree. The **wraparound 2-D mesh** or torus (Figure 5.g) further improves the most important parameters ($D=N^{1/2}$, $B=2*N^{1/2}$, A=4) on the cost of introducing $2*N^{1/2}$ more links. Examples of the 2-D mesh based parallel computers are the DAP, Caltech Mosaic C, Intel Paragon, MIT Alewife, Stanford DASH.

3-D Mesh (Cube) Another frequently used topology is the three-dimensional extension of the 2-D mesh (Figure 5.h) which further improves the critical parameters ($D=3*(N^{1/3}-1)$, $B=N^{2/3}$, $A=3$) without causing too high node degree (d=3, 4, 5 or 6) or network cost ($C=2(N-N^{2/3})$). The technological realisation of this topology is much more difficult than the 2-D mesh and hence the cube topology is introduced only in the most recent parallel computers like Cray T3D, J-Machine and Parsytec GC.

Hypercube The hypercube topology is described in detail in Section 10.2.4 and depicted in Figure 10.2.5. Here only its main parameters are overviewed. The hypercube assures an even smaller diameter ($D=\log N$) than the cube network. It has good bisection width ($B=N/2$) and arc connectivity ($A=\log N$) but naturally on a higher cost ($C=(N\log N)/2$). However, its main drawback is that the node degree is not constant, it grows proportionally with $\log N$ which renders tremendously difficult fabrication of massively parallel hypercubes. The hypercube topology was frequently used in the first and second generation multicomputers where the applied switching technique made the diameter the most crucial parameter of the network. For example, Cosmic Cube, nCUBE and iPSC employed the hypercube topology. When wormhole routing become generally accepted the diameter parameter has lost its superior significance and hypercube lost its popularity.

Notice that the 2-D mesh and 3-D cube are actually two- and three-dimensional hypercubes. Another topology that can be derived from the hypercube is the cube-connected cycle. The **3-cube-connected cycle** of Figure 5.i is obtained by replacing each node of a 3-D cube by a 3-node cycle. A notable property of the 3-cube-connected cycle is that each node in a cycle is connected to the corresponding node in another cycle.

Completely connected network This topology (Figure 5.j) is ideal from the point of view of network diameter ($D=1$) since any two nodes are directly connected. However, the cost ($C=N(N-1)/2$) and node degree ($d=N-1$) parameters are prohibitive in building massively parallel computers based on this topology. Table 1. summarises the main properties of the most important static network types.

Topology	Node degree	Diameter	Bisection width	Arc connectivity	Cost
Linear array	1 or 2	N - 1	1	1	N - 1
Ring	2	N/2	2	2	N
Star	1 or N - 1	2	1	1	N - 1
Binary tree	1, 2 or 3	2log((N+1)/2)	1	1	N - 1
2-D mesh	2, 3 or 4	$2(N^{1/2}-1)$	$N^{1/2}$	2	$2(N-N^{1/2})$
2-D wraparound mesh	4	$N^{1/2}$	$2N^{1/2}$	4	2N
3-D cube	3, 4, 5 or 6	$3(N^{1/3}-1)$	$N^{2/3}$	3	$2(N-N^{2/3})$
Hypercube	logN	logN	N/2	logN	(NlogN)/2
Completely connected	N - 1	1	$N^2/4$	N - 1	N(N - 1)/2

Table 1. Summary of static network parameters for the main topologies

17.2.2 Switching techniques

Switching is the actual mechanism by which a message is removed from the input buffer and placed in the output buffer. The applied switching technique has overwhelming effect on message latency and hence the choice of switching method is absolutely crucial in designing any distributed memory computer. The design space of switching techniques is shown in Figure 6.

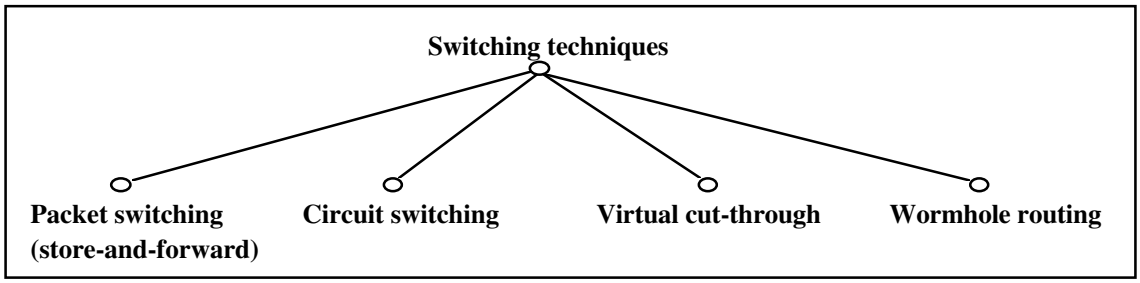

Figure 6. Design space of switching techniques

Packet switching (store-and-forward)

In the first generation of multicomputers the packet switching mechanism has been applied that was borrowed from the world of computer networks. The packet switching mechanism behaves in a store-and-forward manner similar to the mail service. A message is divided into packets which are sent independently via the communication network between the source and the destination nodes. A packet consists of a header and the data. The header contains the necessary routing information based on that the switching unit decides where to forward the packet. The unique feature of the packet switching scheme is that when a packet arrives at an intermediate node, the whole packet

is stored in a packet buffer as it is illustrated in Figure 7.a. The packet is forwarded to a neighbouring node if an empty buffer is available in that node.

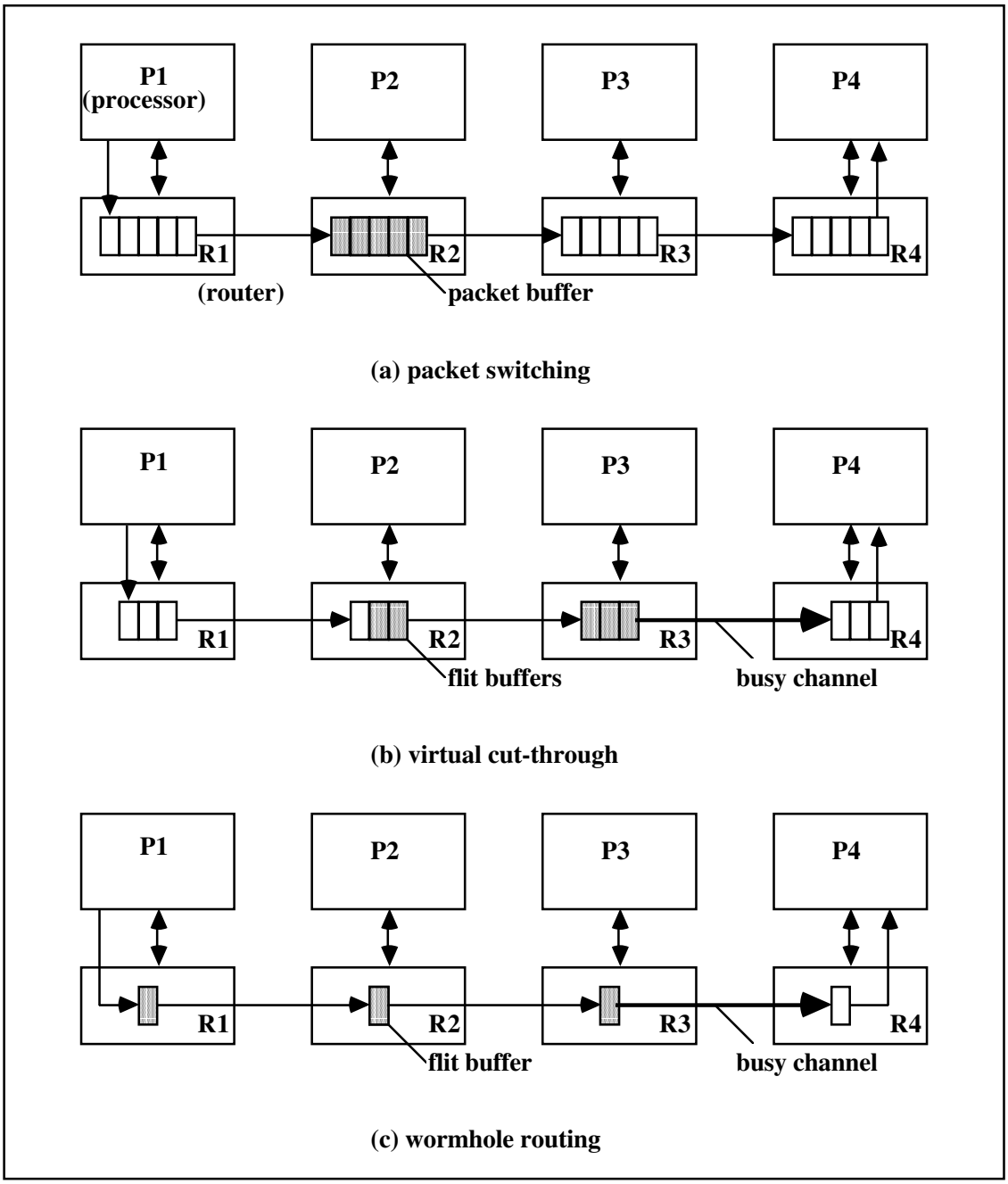

Figure 7. Comparing three switching techniques

The network latency (L) in case of packet switching is:

$$L = \frac{P}{B} * D$$

where P is the packet length, B is the channel bandwidth and D is the distance (number of nodes) between the source and the destination nodes. The timing diagram of the store-and-forward technique shown in Figure 8.a. and the latency expression above both reflect

the main drawback of packet switching: latency is proportional to the message path length. This was the reason why the diameter was the most important parameter in the first generation of multicomputers and why the hypercube topology was so popular. Although the node degree and cost parameters are not the best for the hypercube, its diameter is one of the smallest among the topologies surveyed in Table 1. The other drawback of packet switching is that it consumes significant memory space for buffering every incoming packet. Despite of these drawbacks packet switching was used in every first generation multicomputer (Cosmic Cube, iPSC-1, Ncube 1, Ametek 14, FPS T-series and other Transputer based machines) because of its simplicity.

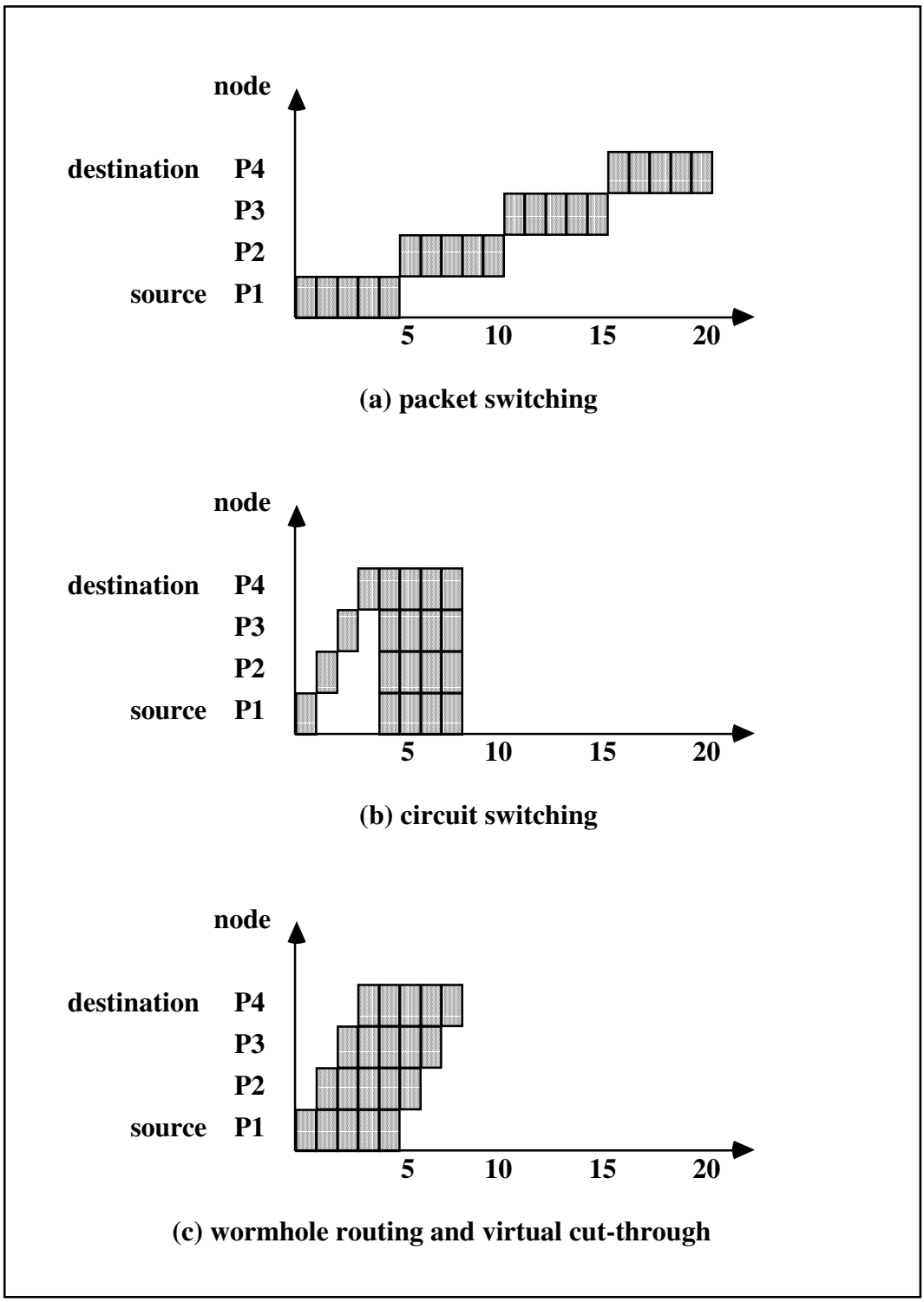

Figure 8. Comparing network latency

Circuit switching

Circuit switching methods behave analogous to telephone systems where a path between the source and destination is initially built up and the circuit is held until the

entire message is transmitted. The circuit switching mechanism is realised in three phases:

1. circuit establishment phase
2. transmission phase
3. termination phase

In the first phase the communication circuit is built up by sending a special short message (called the **probe**) through the network. The probe plays a similar role to the header of packets in packet switching schemes. When the communication circuit is established, the second phase can be started in which the actual message is sent via the communication circuit. During this phase, the channels constituting the circuit are reserved exclusively, no other messages can be transmitted by them. In the last phase the circuit is torn down either by the tail of the transmitted message or by an acknowledgement message returned by the destination node. If a desired channel is used by another circuit in the circuit establishment phase, the partially built up circuit may be torn down. In this policy, called the **loss mode**, the establishment phase is attempted later again.

The circuit switching technique has several advantages over the packet switching method. First of all it does not need packetising. No matter what the message size is, the entire message can be transmitted in the transmission phase on the communication circuit. There is no need for buffering and hence the memory consumption is greatly reduced comparing to the packet switching method. The routing decision is supported by hardware mechanism. This reduces switching time and releases software resources for calculation purposes solving one of the major bottlenecks of multicomputers: the problem of passing messages to remote nodes without degrading node processor performance. The most important benefit of circuit switching becomes apparent if network latency is considered:

$$L = \frac{P}{B} * D + \frac{M}{B}$$

where P is the length of the probe and M is the length of the message. If P << M, the first component of latency becomes negligible and consequently, the network latency becomes independent of the communication distance as it is shown in the timing diagram of Figure 8.

After recognising the advantages of circuit switching over packet switching, Intel applied the circuit switching technique in the second generation iPSC-2 machine. This resulted in a ten times bandwidth improvement over the iPSC-1 machine. In iPSC-2

sending of a message between the two most distant nodes requires only about 10% longer time than sending it between neighbour nodes. Accordingly, mapping algorithms have become much simpler since they do not have to handle communication requirements, only computational load requirements should be considered.

Virtual cut-through

Virtual cut-through attempts to combine the benefits of packet switching and circuit switching. The message is divided into small units called **flow control digits**, or **flits**. As long as the required channels are free, the message is forwarded flit by flit among nodes in a pipeline fashion. If a required channel is busy, flits are buffered at intermediate nodes. Depending on the size of the available buffers virtual cut-through can behave in different ways. If the buffers are large enough, the entire message is buffered at the blocked intermediate node resulting a behaviour similar to packet switching. If the buffers are not large enough, the message will be buffered across several nodes, holding the links between them as shown in Figure 7.b.

The network latency is:

$$L = \frac{HF}{B} * D + \frac{M}{B}$$

where HF is the length of the header flit. If HF << M, the second component of latency will dominate and consequently, the network latency becomes independent of the communication distant like in circuit switching.

Wormhole routing

Wormhole routing is a special case of virtual cut-through, where the buffers at the intermediate nodes have the size of a flit. The network latency is the same as in virtual cut-through and therefore independent of the communication distant. The functional principles of the wormhole routing are shown in Figure 7.c, while its pipelining mechanism is illustrated by Figure 8.c. Packet switching, virtual cut-through and wormhole routing can be compared by Figure 7. showing that virtual cut-through is a compromise between the other two switching techniques. Figure 8. compares the communication latency of the four switching techniques in a contention-free network where the behaviour of virtual cut-through is the same as that of the wormhole routing.

Comparing circuit switching and wormhole routing a significant difference between them is that meanwhile wormhole routing is able for **packet replication**, i.e. to send copies of flits on several output channels, circuit switching is unable to do it. Packet replication is useful in implementing broadcast and multicast communication.

The advantage of wormhole routing over circuit switching and virtual cut-through appears in networks where contention is not negligible. In circuit switching, once a channel is assigned to a message, it cannot be used by other messages until the channel becomes free. In virtual cut-through blocked messages should be stored in the buffers of intermediate nodes. In case of wormhole routing, channels can be shared by multiple messages after introducing the **virtual channel** concept.

Virtual channels make it possible to use the same physical channel by several independent messages by providing multiple buffers for each channel in the network. Virtual channels decouple allocation of buffers from allocation of physical channels. A virtual channel consists of a buffer for holding one or more flits. Several virtual channels share the bandwidth of a single unidirectional physical channel through a pair of multiplexer-demultiplexer units as shown in Figure 9. for three virtual channels A, B and C. For each virtual channel a bidirectional control line is employed with a handshaking protocol in order to assure pipeline usage of virtual channels. The low value represents an empty buffer flit on the receiver side. The multiplexer switch can choose any of the virtual channels whose control line is low and the sender flit buffer is full. After selecting such a channel the corresponding control line is raised indicating that the channel is ready to send a flit. When the flit arrives, the control line remains high as long as the flit buffer on the receiver is not emptied.

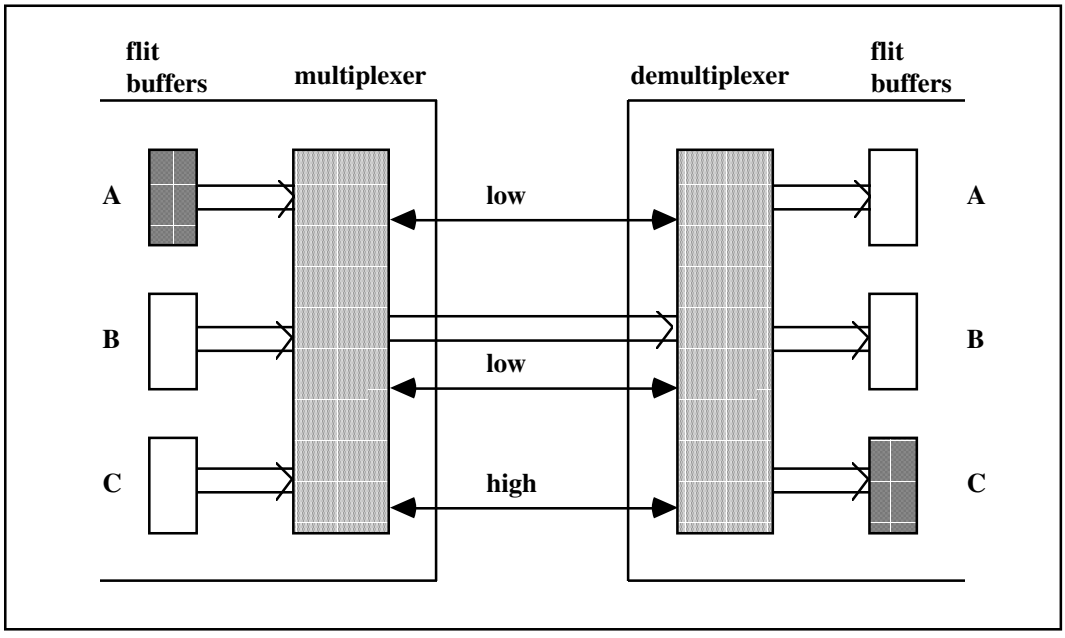

Figure 9. Organization of virtual channels

The introduction of virtual channels entails the following advantages:

1. Virtual channels increase network throughput by reducing physical channel idle time.

2. Virtual channels can be used for deadlock avoidance.

3. Virtual channels facilitate the mapping of logical topology of communicating processes onto a particular physical topology.

4. Virtual channels can guarantee communication bandwidth to certain system-related functions, such as debugging and monitoring.

17.2.3 Routing

The third crucial decision point in designing message passing architectures is the selection of the routing scheme. The task of routing is to determine the path between the source and the destination nodes of a message. Routing has great influence on the performance of the direct network and hence it plays a crucial role. Routing algorithms on the nodes of the multicomputer can be realised either by software or hardware. Obviously, those routing algorithms are preferable which are easy to implement in hardware.

Routing algorithms are basically divided into two classes according to their capability of modifying the routing paths based on dynamic network conditions. In **deterministic routing** the path is completely determined by the source and destination nodes. Intermediate nodes are unable to redirect messages even in case of network congestion. On the contrary, in **adaptive routing** intermediate nodes can take the actual network conditions such as the presence of failures or bottlenecks into account and are able to determine accordingly which neighbour should be sent the message to.

Deterministic routing can be further classified according to the node position where the deterministic path is selected. In case of **source routing** it is the source node that selects the complete path between the source and destination nodes. **Distributed routing** gives the freedom to each intermediate node to independently determine the next node of the path where the message should be sent to. Notice that adaptive routing is always based on distributed routing.

All of these routing schemes can be divided into **minimal** and **nonminimal routing**. A routing algorithm is minimal if the path selected by the algorithm is one of the shortest paths between the source and the destination nodes. Otherwise, the algorithm is called nonminimal routing. Notice that in minimal routing the message is getting closer to the destination in every hop, while this is not true for the nonminimal routing. Though theoretically deterministic routing schemes can be nonminimal, in practice they are always minimal. The categorisation of routing protocols is summarised by Figure 10. Obviously for each routing algorithm a strict requirement is to guarantee deadlock- and livelock-freeness.

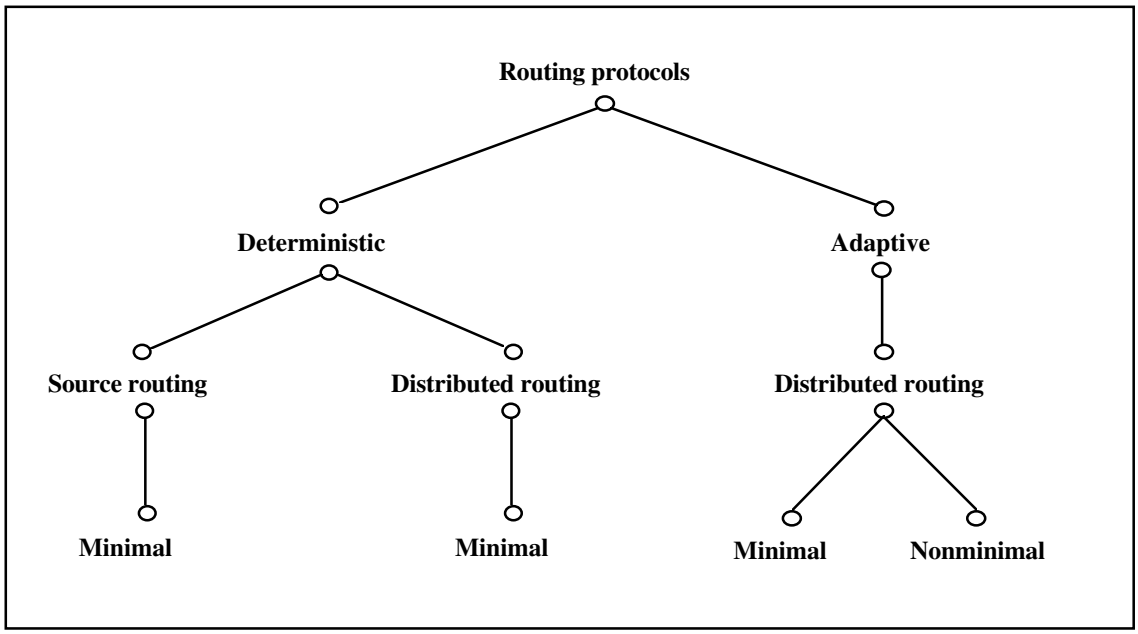

Figure 10. Categorization of routing protocols

Deterministic routing

Three deterministic routing schemes are applied in practice:

1. Street-sign routing

2. Dimension-ordered routing

3. Table-lookup routing

Street-sign routing belongs to the class of source routing and accordingly the message header should carry the complete path information. Since the length of the message influences the performance of the system, the header information should be kept as small as possible. In the street-sign routing this requirement is fulfilled by introducing the concept of default direction. The message header contains routing information for those intermediate nodes only where the message should turn. Any node receiving a header flit compares the node address with its own address. In case of mismatching the message should follow the default direction. In case of matching the second information of the header flit describes the operation to be done by the node. This can be either a turn of the message or the arrival to the destination. In case of turn action the node and action identifiers are removed from the header which is forwarded to the specified neighbour node. The street-sign routing scheme is used in the Intel iWarp machine.

Dimension-ordered routing is a distributed deterministic routing scheme applied in n-dimensional meshes. The main idea is that messages travel along a certain dimension until reaching the proper co-ordinate of that dimension. At this node they proceed along

the next dimension and so on. Deadlock-free routing is guaranteed if the dimensions are strictly ordered.

Table-lookup routing, another distributed deterministic routing scheme, can be used in any network topology. At each node a routing table contains the identifier of the neighbour node where the message should be forwarded in case of each destination node. Table-lookup routing is very advantageous if realised by software but introduces unnecessary upper limit for the size of network if it is realised by hardware. Furthermore, large lookup table reduces the available chip area. Accordingly, the main design concern in table-lookup routing is how to reduce the size of the lookup table. One possible solution, called **interval labelling**, associates each output channel of the node with an interval. If the destination address carried in the header belongs to this interval, the header should be forwarded on the corresponding output channel. This scheme is employed and realised by hardware in the C104 routing chip of the Inmos IMS T9000 Transputer system.

Adaptive routing

Adaptive routing schemes are much more favourable than deterministic routing since, as their name suggests, they are able to take routing decisions in the presence of faulty and blocked channels. Accordingly, large number of various adaptive routing schemes have been proposed which can be further classified. The categorisation of adaptive routing protocols is depicted in Figure 11.

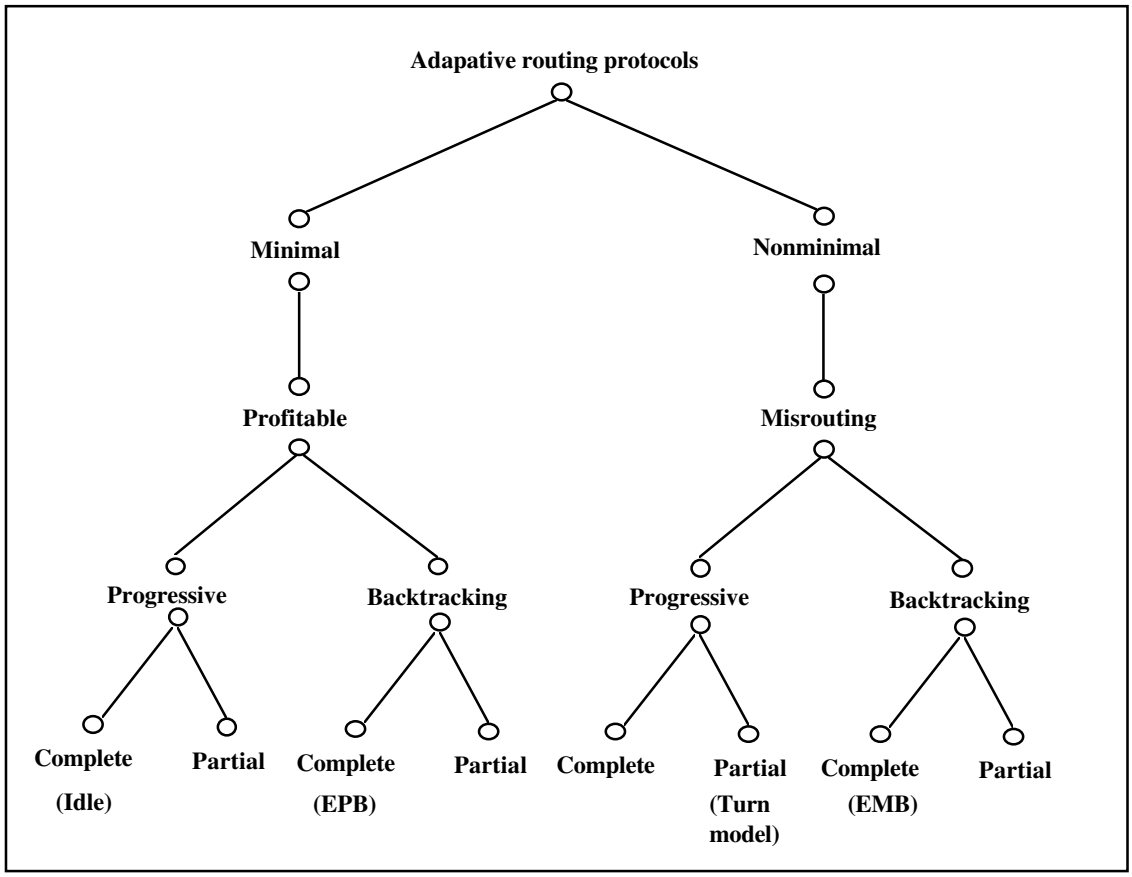

Figure 11. Categorization of adaptive routing protocols

Adaptive routing schemes can be either **profitable** or **misrouting** according to the selection of the output channel. In case of profitable routing only those channels are candidates for selection that are known to be guaranteed to move closer to the destination. Profitable routing represents a conservative view. Misrouting protocols rely on an optimistic view and can use both profitable and nonprofitable channels for establishing a path between the source and the destination. Selecting a nonprofitable channel optimistically assumes that it will lead the message to a free set of profitable channels allowing further advance to the destination. The advantages of profitable protocols are:

1. They result in a minimum-length path.

2. They are free from livelock.

3. They make it easier to prove deadlock freedom.

Misrouting protocols are advantageous when faulty channels are present in the network. Under such conditions misrouting protocols have higher chance to find an appropriate path.

Adaptive routing schemes can be further divided into **progressive** or **backtracking** protocols. In progressive routing, messages cannot step back on the path they have

already followed. Conversely, in the backtracking scheme, messages are able to backtrack and systematically explore all possible paths between the source and the destination nodes. The message header should contain some status information to avoid repetitive search of the same path and hence livelock freedom is assured. Backtracking schemes are also deadlock-free since they do not block holding resources. Notice that in wormhole routing the backtracking protocol cannot be used since the pipelined nature of wormhole routing contradicts to backtrack.

In order to prevent repeated search of the same path, backtracking protocols store history information in the message header (or probe). Since the search space can be very large, especially in misrouting protocols, the header becomes too long which significantly increases the latency time. In case of circuit switching a hardware support on each node can significantly reduce the header size. Each channel on each node of the multicomputer is supplied with a history bit vector which contains as many bits as many channels are available on the node. If a probe enters the node on channel K, the bits of history bit vector K contain information on the corresponding output channels. If an output channel has been tried the corresponding bit is high, otherwise it is low. An additional history bit vector is used on each node for administrating the probes that were created by the node itself.

If there is no free profitable channel at an intermediate node there are several alternative strategies to follow:

1. Progressive profitable routing waits until a profitable channel becomes free.

2. Progressive misrouting protocol tries a nonprofitable free channel.

3. Backtracking routing steps back and starts over at a previous node.

All the protocols mentioned above can be **completely** or **partially adaptive** ones. A completely adaptive protocol can choose any channels from its class. For example, a completely adaptive backtracking protocol realises an exhaustive search for the path. Partially adaptive protocols are used when certain restrictions are introduced in order to guarantee deadlock freedom in the network.

From the many available adaptive routing protocols only a typical one called the **turn model** (misrouting, progressive, partially adaptive) is described here as case study. Glass and Ni recognised that deadlock occurs because the message paths contain turns that form a cycle. They proposed to restrict the possible turns by disallowing certain combinations of turns and as a result cyclic dependencies are prevented (Glass and Ni, 1992). Based on this principle several protocols can be defined. All are misrouting, progressive and partially adaptive. One of them is the **west-first routing** algorithm applied in 2D meshes. According to the main idea from the eight possible turns those two

to the west are prohibited. This creates the rule: first route the message west, if necessary, and then turn adaptively to other directions.

17.3 Fine-grain systems

In the eighties there was a general expectation that massively parallel systems should be built on new computational models that require fine-grain or at least medium-grain parallelism. Many research projects and commercial parallel computers appeared based on this concept. In Europe the most influential result was the Transputer which was built on a new theoretical concept, called the CSP (Communicating Sequential Processes). In the USA one of the main representatives of this direction was the J-Machine project at the MIT. The Transputer represents a static, synchronous model realised at medium-grain parallelism, while the J-Machine project represents a dynamic, asynchronous model realised at fine-grain parallelism.

By the nineties it became clear that big parallel computer manufacturers are not interested in revolutionary new concepts. Rather they follow a cautious evolutionary approach which means that they are more interested in connecting high-performance off-the-shelf microprocessors into a loosely connected network. This approach obviously leads to coarse-grain parallelism. The research on fine-grain and medium-grain systems gradually lost ground. Initially fine-grain commercial products like the Connection Machine recently become coarse-grain as well. After all, there remained only one significant fine-grain research project, the J-Machine project that is worth describing in the current book.

17.3.1 Message-Driven Processor and the J-Machine

The J-Machine was designed at MIT with two primary goals. Firstly, they wanted to create a single chip processing node in which the communication and synchronisation mechanisms can efficiently support several parallel programming models like dataflow and actor based object-oriented models. The chip is called Message-Driven Processor (MDP) since, it processes in response to messages without having any receive instructions. Secondly, they wanted to prove that connecting plenty of inexpensive nodes like the MDP will result in a powerful, cost-efficient, fine-grain multicomputer.

The MDP design shows similarities with several other parallel computer projects. Like the Inmos Transputer, the Caltech Mosaic C, and the Intel iWARP, the MDP is a single chip processing element, integrating a processor, memory, and a communication unit. It supports a global virtual address space like the BBN Butterfly and the IBM RP3.

The novelties of MDP come from its efficient primitive mechanisms for communication, synchronisation and naming.

The J-machine is an experimental prototype multicomputer to study the possible exploitation of VLSI technology in constructing massively parallel fine-grain multicomputers. Each processing node of the J-Machine consists of an MDP and 1 Mbyte of DRAM memory (Dally et al., 1992). The nodes are connected into a *3-D mesh network* which uses *deterministic, dimension-ordered, wormhole routing*. A 1024 node J-Machine is constructed and used at MIT.

17.3.2 The communication subsystem of the J-Machine

The pin layout and inner structure of the MDP is shown in Figure 12.a and b, respectively. The MDP provides 6 bidirectional bit-parallel channels, each consisting of 9 data and 6 control lines. The six network ports correspond to the six cardinal directions (+X, -X, +Y, -Y, +Z, -Z) and make possible the direct wiring of MDP chips by connecting opposite direction ports. These connections result in a 3D mesh topology in which deterministic dimension order routing is realised by routing messages first in dimension X, then Y and finally Z.

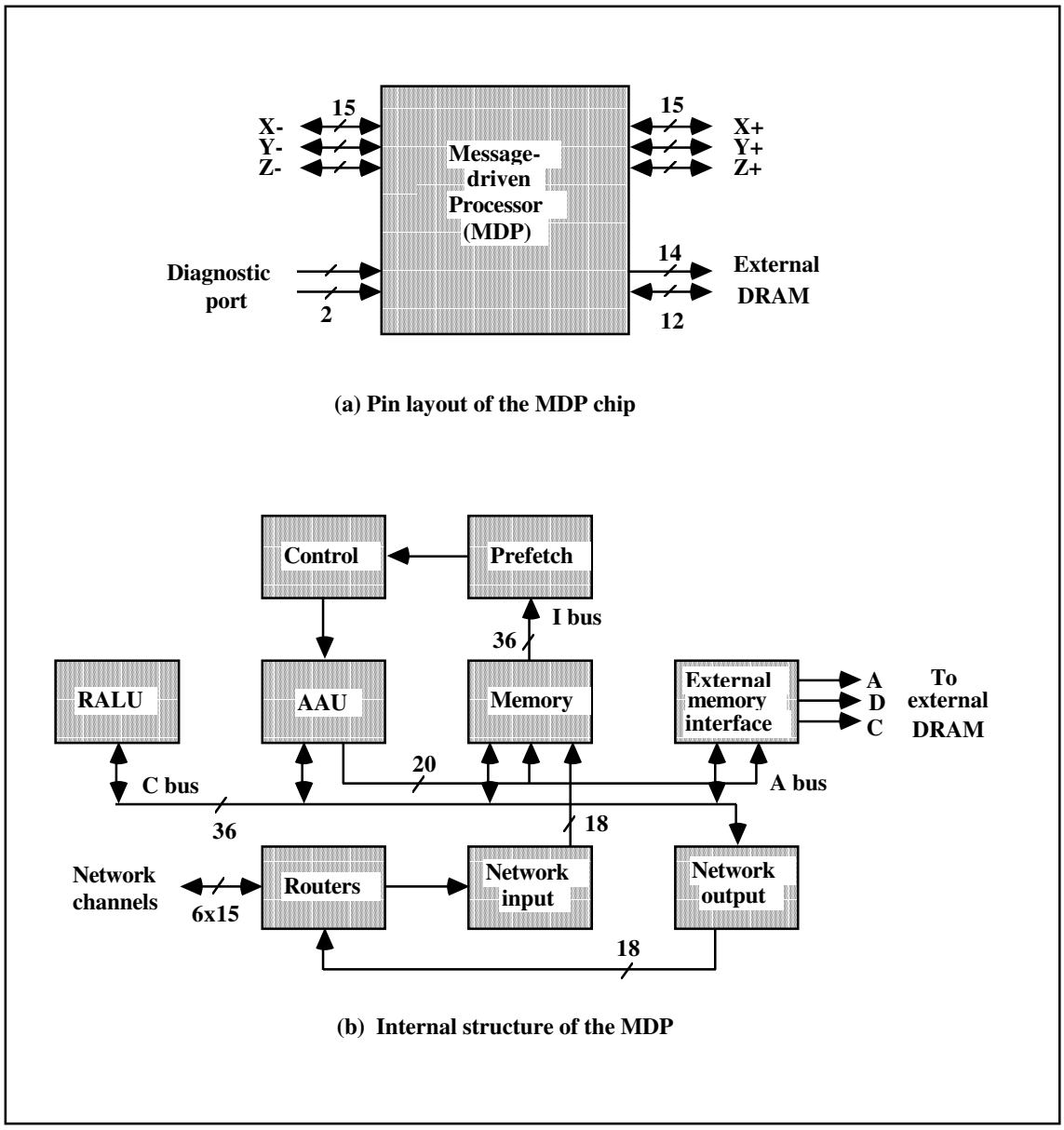

Figure 12. The Message Driven Processor (Dally & al. 1992)

Messages are divided into flits. The first three flits contain the X, Y and Z addresses of the destination node. When a message arrives to a node along dimension D, the router unit compares the D address of the node with the first flit of the message. If the two values do not match, the message is forwarded along the same dimension D. If the two values match, the node strips the lead flit of the message and routes the rest along the next dimension. The part of the routing unit, being responsible for the dimension order routing, is depicted in Figure 13. The MDP contains three such independent routers for each of the three dimensions. Notice that messages entering the new dimension compete with messages continuing in that dimension at a two-to-one switch. The granted message will block the other competing one for its duration. The switching scheme used in the J-machine is the wormhole technique, i.e. once the head flit establishes the communication path, subsequent flits follow directly behind it.

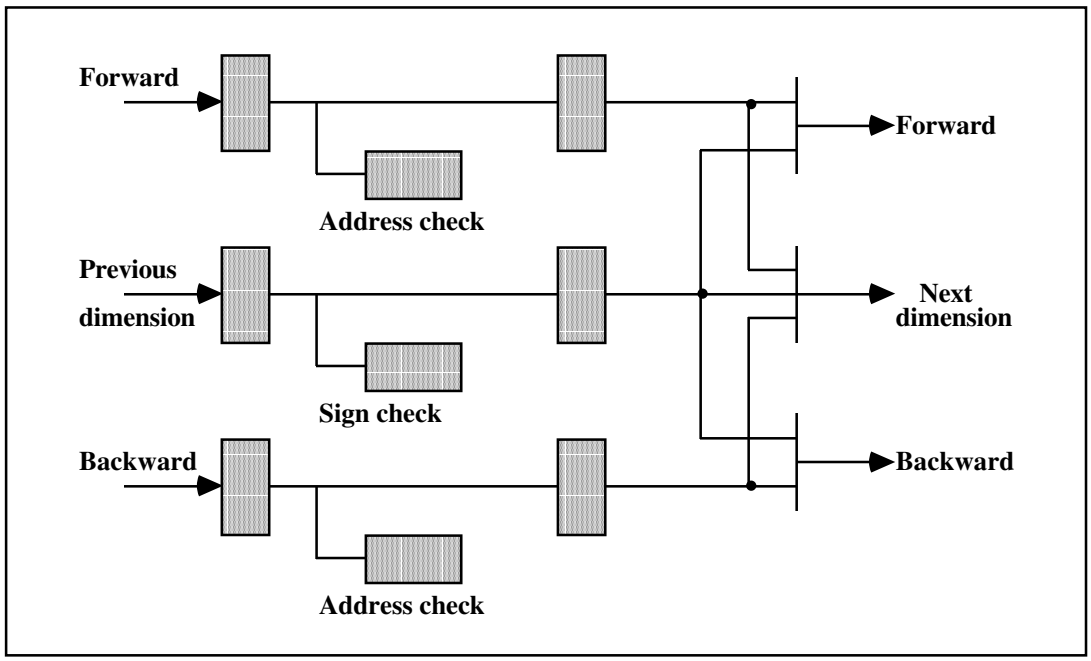

Figure 13. Structure of the routing unit for one direction (Dally & al. 1992)

In case of contentions mentioned above the J-machine network applies blocking flow control. The blocked message is compressed in the routers along the message path, occupying a two-flit buffer at each node as shown in Figure 14. When the necessary path is cleared up the message is uncompressed and advanced towards its destination node. Messages are divided into two priority classes which have separate buffers and routing logic on each node but they share the physical wires. Low priority messages can not block high priority ones, but the high priority network can preempt the wires even if the low priority network is congested.

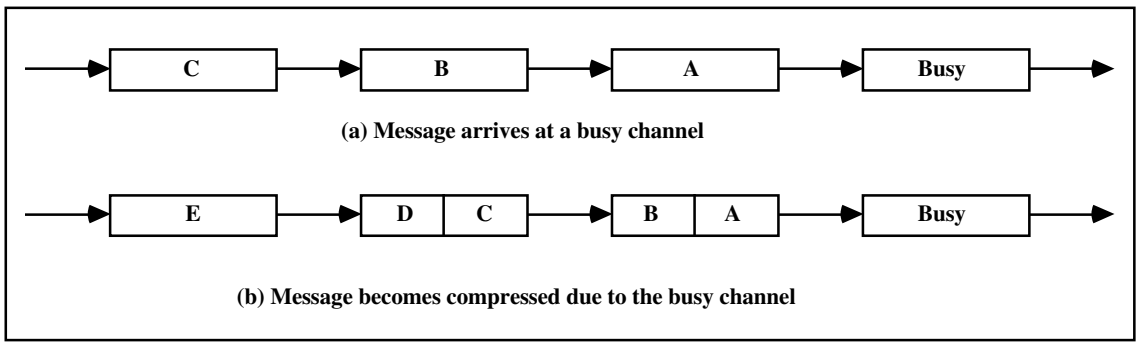

Figure 14. Message compression due to busy channels (Dally & al. 1992)

Additionally to the router, a network output and network input unit belong to the communication subsystem of the MDP. The network output unit contains an 8-word FIFO buffer where the MDP transfers one or two words on each SEND instruction. If a message is complete or the FIFO is full the network output unit launches the message to the network. Similarly, the network input unit provides a 4-word buffer where flits from

the network are collected. When the buffer is full, the network input unit writes its contents to the on-chip memory in one cycle.

17.3.3 Architecture of the Message-Driven Processor

The MDP was designed to support fine-grain concurrent programs by:

1. reducing the overhead and latency associated with receiving a message

2. reducing context switch time

3. providing hardware support for object-oriented programming.

There are three execution levels in the MDP: background, priority 0 (low priority), and priority 1 (high priority). When no message requires service the MDP works at the background level. For each level a separate register set is maintained consisting of the following registers:

(a) R0-R3: general purpose data registers
(b) A0-A3: address registers
(c) ID0-ID3: ID registers
(d) instruction pointer

Two sets of ID registers (ID0-ID3) and two sets of queue registers are available for the priority levels. Each queue register set contains a base/limit register and a head/tail register for administrating the two message queues. Additionally, a translation buffer base/mask register and a status register are shared between the different execution levels. The use of separated register sets for different execution levels assures extremely fast switch among them. Also, the small size of each register set facilitates quick task switching within one execution level.

For making software more efficient and robust MDP employs hardware support for type checking and synchronisation. Each 36-bit word contains a 4-bit tag. Two tags, Fut and Cfut, are used for interprocess synchronisation. Fut serves for global synchronisation and Cfut for the local one. Initially an empty location is supplied with the tag Cfut. If a read attempts to access such a location, the reading process is suspended by the Cfut fault handler. A write access substitutes the Cfut tag with the tag of the final value. Notice that the synchronisation scheme applied in the MDP is equivalent with the full/empty bit synchronisation of the multithreaded machines (Chapter 16).

A series of network instructions (SEND, SENDE, SEND2, SEND2E) are used to compose and send messages between processors. They can inject one or two words into the network either at priority 0 or 1. A typical code segment for sending a message at priority 1 is the following:

```
SEND      R0,1          ; send destination address
SEND2     R1,R2,1       ; header and receiver
SEND2E    R3,[2,A3],1   ; two words and end of msg
```

The first instruction sends the X, Y, and Z co-ordinates of the destination processor from register R0. The second instruction enqueues two words from registers R1 and R2 for transmission. The third instruction takes a word from R3 and another one from the memory and finally, creates an end of message mark.

The MDP contains a Message Unit (MU) besides the conventional Instruction Unit (IU). When a message arrives, it is the task of the MU to decide if the message should be immediately executed by preempting the IU or to be queued. Queuing takes place without interrupting the processor, by stealing memory cycles. The logical structure of a message is as follows:

<priority>, <opcode>, <arg>, ... ,<arg>

If the incoming message has higher priority than the task currently executed by the IU, its processing is immediately started and the current task is suspended. The message opcode is a physical address to the routine that implements the function required by the message. These routines are stored in a small ROM and realise the following message types:

1. READ, WRITE, READ_FIELD, WRITE_FIELD: These messages are used for reading and writing memory locations.

2. DEREFERENCE reads the entire contents of an object.

3. NEW creates a new object and returns an identifier.

4. REPLY, FORWARD, COMBINE, GC are used to implement "futures", multicast, fetch-and-op combining, and garbage collection, respectively.

5. CALL and SEND cause a method to be executed. In case of CALL the method is defined by the <method-id> parameter of the message. In case of SEND the method is determined at run time based on the type of the receiver.

An innovative unique feature of the MDP is that a dispatch mechanism directly processes messages requiring low latency (for example, FORWARD and COMBINE). When a message reaches the head of the highest priority nonempty message queue, the dispatch mechanism transforms it into a task. The opcode of the message is written into the instruction pointer and the message length (part of the message header) together with the queue head are used to create a message segment descriptor.

17.3.4 Assessment of the J-Machine

Experimental work on the J-Machine prototype at MIT proved that the initial design goals were fulfilled, i.e., combining fast message injection, automatic message buffering, and fast message dispatching resulted in efficient communication and synchronisation on a fine grain level where task sizes are less than 150 instructions. Table 2. compares the asynchronous send and receive overheads (without including network latency) of the J-Machine to those of several other current multicomputers constructed from off-the-shelf microprocessors (Noakes et al., 1993). The first two rows are based on the vendor message libraries, while the third and fourth lines report results gained by the Active Message system (Eicken & al. 1992) implemented on the same hardware. The table clearly shows that the J-Machine reduced the overhead per message by up to two orders of magnitude and the overhead per byte by almost an order of magnitude. Further measures showed that threads achieve processor utilisation of over 50% with task length between 25 and 50 cycles. 90% of the total network bandwidth can be achieved by messages of 8-word length. A barrier synchronisation library has been elaborated at MIT that exceeds the similar libraries of other multicomputers with one to two orders of magnitude.

However, the experimental measures have also revealed some limitations of the J-Machine. The routing protocol is not fair in the sense that messages locked out by a contention can be delayed for an arbitrary time. The register set proved to be too small and hence led to much more memory references than necessary. The external memory bandwidth of the MDP is three times less than the network would require. The external memory latency also limits the sequential performance of the nodes. The naming mechanism is not efficient enough to support the global name space concept of the J-Machine. Finally, the lack of floating point hardware strictly limits the achievable performance on numerical applications.

Considering both the benefits and limits of the J-machine architecture one can conclude that marrying the J-Machine communication and synchronisation scheme with a contemporary high performance RISC processor and 4 Mbytes of high-bandwidth DRAM memory would result in an outstanding multicomputer system.

Machine	T1: us/msg	T2: us/byte	cycles/msg	cycles/byte
nCUBE/2 (Vendor)	160.0	0.45	3200	9
CM-5 (Vendor)	86.0	0.12	2838	4
nCUBE/2 (Active)	23.0	0.45	460	9
CM-5 (Active)	3.0	0.12	109	4
J-Machine	0.9	0.04	11	0.5

Table 2. One-way message overhead
T1 is the sum of the fixed overheads of send and receive
T2 is the injection overhead per byte (Noakes & al. 1993)

17.4 Medium-grain systems

The Transputer approach is another attempt to realise multicomputers by directly connecting processor chips. A Transputer is a complete microcomputer integrating a central processor, a floating point unit (except for the first generation T212 and T414 chips), static RAM, interface for external memory, and a number of communication links into a single chip. The communication links enable Transputers to be connected directly to each other and to communicate. The appearance of the Transputer in the mid-eighties led to a breakthrough in the spreading and acceptance of parallel processing. The Transputer offered a relatively cheap and very modular tool to build parallel computers from a very small size up to massively parallel systems based on the same theoretical and practical platforms.

In contrast to the Message-Driven Processor approach, Transputers were designed to implement a static, synchronous model of parallel programming based on Hoare's CSP concept (Hoare, 1978 and 1985). CSP (Communicating Sequential Processes) had a solid theoretical background which resulted in a well-understood concurrent programming paradigm. CSP was further developed and turned into a concurrent programming language, called Occam-2, that contains clear programming constructs and concepts (PAR and ALT constructs, channel concept, etc.) to handle processes, communication and synchronisation.

Transputers and Occam have two generations. The first generation reflects the requirements of the originally intended application areas, like signal processing and real-time systems. For such applications relatively small size systems with fast communication links, basically neighbourhood oriented communication, and fast context switching seemed to be sufficient. However, it turned out that the Transputer was more than the building block of special purpose parallel systems. General purpose

multicomputers built from Transputers were comparable in performance with other first generation multicomputers. Notable Transputer chips of the first generation are the T212, T414, and T805 Transputers.

However, when the 2nd generation multicomputers appeared, it became clear that the first generation Transputers cannot compete with these new machines. A thorough redesign was necessary which improved both the sequential performance and the communication concept of the Transputer. In the new concept the Transputer is supported by a router chip and the processing node structure has become like the one shown in Figure 1.b. The first generation store-and-forward switching technique is replaced by the wormhole routing like in other 2nd and 3rd generation multicomputers. Another important improvement was the introduction of the virtual channel concept which significantly reduced the mapping problem. The 2nd generation Transputer is called the T9000 and the router is realised by the C104 routing chip.

Both the first and second generation Transputers have been designed for supporting advanced communicating process architectures. Fulfilling this objective was so successful that both generations of Transputers are much stronger in the field of communication than in the field of computing. The first generation was revolutionary comparing to other microprocessors in supporting direct connection of Transputers without using complicated multibus systems. On the other hand, the numerical performance of the Transputer was not better than other contemporary microprocessors. Even, the very first Transputers (T414 and T212) have been delivered without floating point units. Similarly, the second generation Transputer family implements a very advanced communication system but, concerning the computational speed, either in MIPS or Mflops, the T9000 is left far behind the other modern superscalar chips like PowerPC, Pentium, etc. Therefore, the main significance of the Transputer concept appears in its supportive mechanisms to create multicomputers and hence, in the next sections these features of the second generation Transputer family will be explained in detail.

17.4.1 Internal structure of the T9000 Transputer

The internal structure of the T9000 Transputer is depicted in Figure 15. The main components of the T9000 are the 32-bit-integer processor, a 64-bit floating point unit, 16 kilobytes of CPU cache memory, four serial-communication-link engines, a virtual channel processor (VCP) and a programmable memory interface (PMI). Synchronisation with the external world is supported by four pairs of event channels and two control links. All the subunits are connected by an internal crossbar switch that controls four 32-

bit data paths which connect the CPU, the VCP, the PMI, and the process scheduler to the four banks of the main cache.

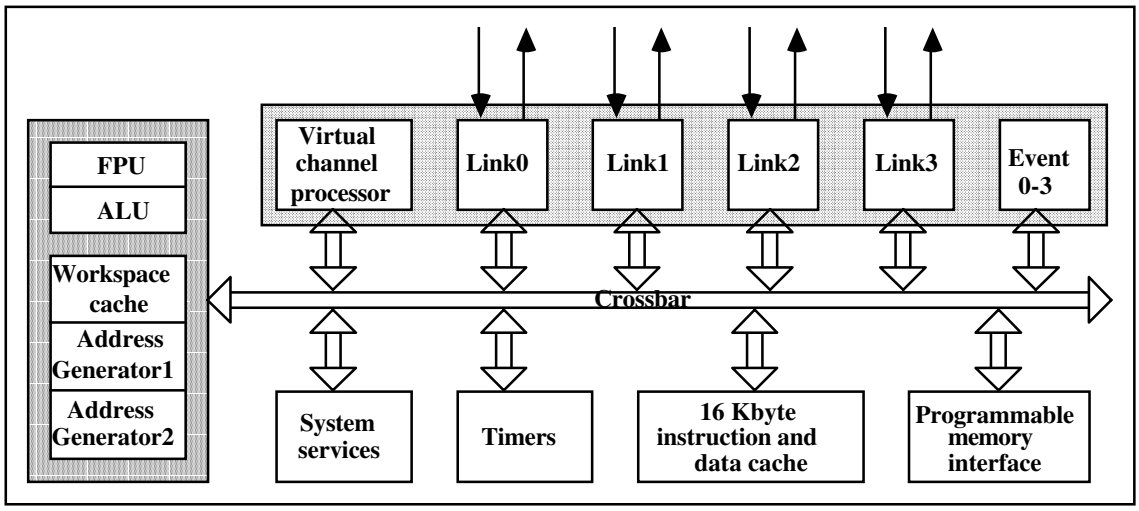

Figure 15. Structure of the T9000

17.4.2 Process management in Transputers

One of the most remarkable features of Transputers is the employment of a very small register set and, as a result, a very fast context switching scheme. Each Transputer has only six registers to accommodate the active process. The first three ones referred as A, B, and C compose a small size stack for holding operands of arithmetic operations. Register A is the top of the stack. The fourth register (W) points to the workspace area of the active process. The fifth register plays the role of a program counter (PC). Finally, the sixth register, called the operand register (O), serves for elaborating operands and instructions longer than 4 bits. In those Transputers where floating point unit is included in the chip, registers FA, FB, and FC are added to the register set to support floating point operations. The register structure of the Transputers is shown in Figure 16.

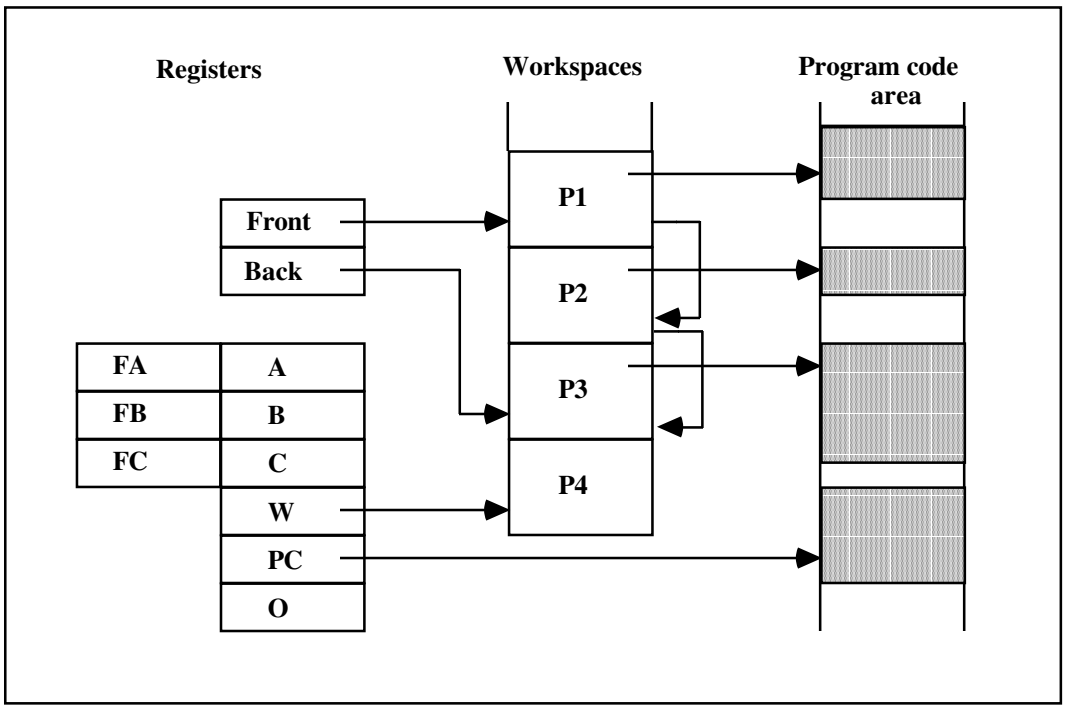

Figure 16. Register and Ready List structure of transputers

Processes can be in one of the three states:

1. Active

2. Ready

3. Inactive

The Active and Ready states do not differ logically and are called Active in the Transputer Manual, but we distinguish them from didactic reason; the Active process is currently executed by the Transputer, while the Ready processes are waiting for the availability of the Transputer in the Ready List. Inactive processes are waiting for a condition like

1. execution of a channel operation

2. reaching a specified time

3. access of an open semaphore.

A process is represented by its program counter and workspace. For the Active process the program counter is loaded into register PC and the workspace is pointed by register W of the Transputer. The program counters of Ready and Inactive processes are stored in their workspaces. The workspaces of Ready processes are queued in the Ready List. Two kernel registers are employed in the Transputer to support the hardware maintenance of the Ready List. The first one, called Front, points to the workspace of the first Ready process in the queue, while the second one, called Back, points to the last

Ready process as illustrated in Figure 16. Actually two Ready Lists are introduced in the Transputer; one for the low priority processes and another for the high priority processes. Inactive processes are placed in different queues depending on the reason of their suspension. The forthcoming sections will explain how the processes waiting for channel operation are blocked.

Process switch is an extremely fast operation. It requires only the following steps:

1. Saving the PC of the Active process to its workspace.

2. Adding the workspace of the active process to a waiting queue.

3. Restoring register PC from the workspace of the first process in the Ready List.

4. Loading Register W with the value of register Front and set register Front to point to the second process of the Ready List.

Notice that the other registers (A, B, C, and O) are not needed to be saved since the Transputer permits process switch only at instructions where the value of these registers are irrelevant.

17.4.3 Channel communication inside a Transputer

Channels always connect two processes: the sender and receiver ones. Communication on the channel is synchronous, i.e. neither the sender nor the receiver can continue after executing a channel instruction until the partner process is not ready to perform the transmission. As a result, the first process trying a channel communication should be suspended till the partner arrives to the channel operation.

A genuine method is applied in every Transputer chip to implement the synchronous channel communication mechanism. The channel itself is realised as a single memory word which is initialised to a special value 'NotProcess' before using the channel. When a process (Pi) wants to perform a channel operation three data (length, channel, pointer) should be placed in the register stack as it is shown in Figure 17.a. The contents of register B identifies the channel to be used in the communication. Register C contains the address of the message in memory, while A defines the length of it. The 'NotProcess' value of channel C means that the partner process (Pj) has not arrived yet to its communication point and hence, Pi should be suspended. It is simply done by overwriting the contents of channel C with the address of the workspace of Pi and saving the registers PC and C into the workspace (Figure 17.b).

When Pj reaches its communication point, registers A, B, and C contain the message length, channel identifier and message area address, respectively (Figure 17.c). Since channel C contains the identifier of Pi, both partner processes are ready to execute the

communication which immediately takes place. When the message transmission is completed, Pi is moved to the Ready List of the Transputer and Pj can continue its progress (Figure 17.d). In the T9000 there is a possibility to transfer variable length messages, too. In that case the message length parameters of Pi and Pj could be different and hence it should be saved into the workspace together with the message area address.

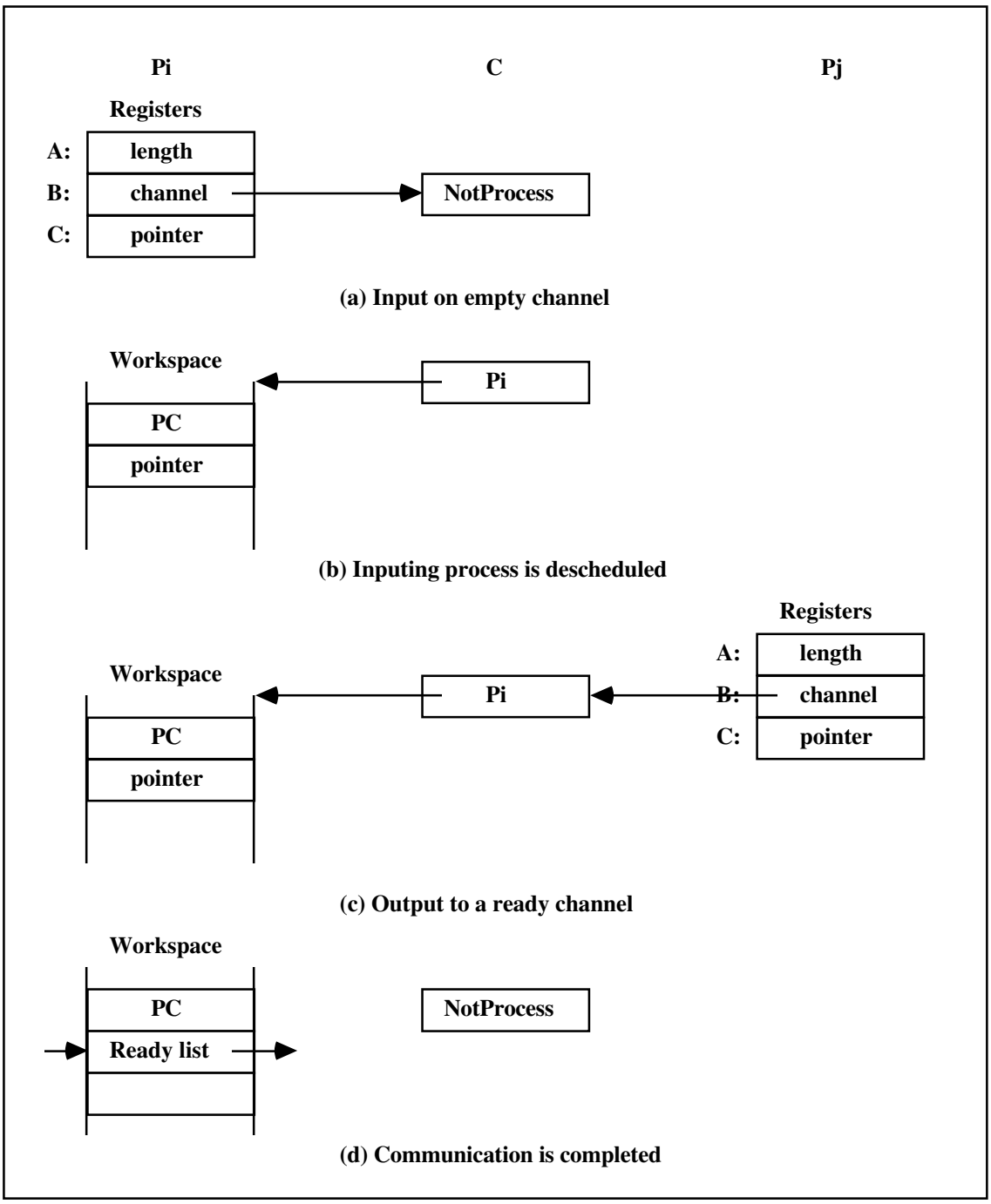

Figure 17. Internal channel communication in transputers

Notice that neither a message queue nor an extra process queue was necessary to realise the synchronous communication based on the channel concept. This is an advantageous consequence of the Transputer communication concept.

17.4.4 Communication between neighbour Transputers

First generation Transputers employ serial communication links, called the 'OS-Links', as an integral part of the processor chip. Each OS-Link comprises two wires: one for an input channel and another one for an output channel. It provides a full-duplex, asynchronous, bit-serial, point-to-point connection between two neighbour Transputers by connecting a link interface of one Transputer to a link interface of the other Transputer. Data is sent byte-by-byte. Every byte of data sent on a link is acknowledged on the input of the same link. The quiescent state of a link output is low. The serial bit protocol for a data byte on the link is as follows:

< high, high, <data>$_8$, low >

The ACK signal is:

< high, low >

The OS-Link has an exceptionally low implementation cost, but by today's standards its speed of 20 Mbits/s is relatively low. In the first generation Transputers a *limited number* (typically 2 or 4) *of unshareable physical links* have been used to support direct communication among adjacent Transputers. This limited number of links resulted in a serious limitation in the Transputer implementation of the CSP concept. One of the cardinal principles of CSP is the **scheduling invariance** which means that a correct CSP program can be executed on any Transputer system, no matter how the processes are actually mapped to the processor space. The scheduling invariance forms the basis of developing Occam programs on a single Transputer environment and then porting the correct program to any multi-Transputer system. However, this principle is hurt when processes on a Transputer should communicate to more than four processes mapped on other Transputers. Such a problematic case is illustrated in Figure 18.

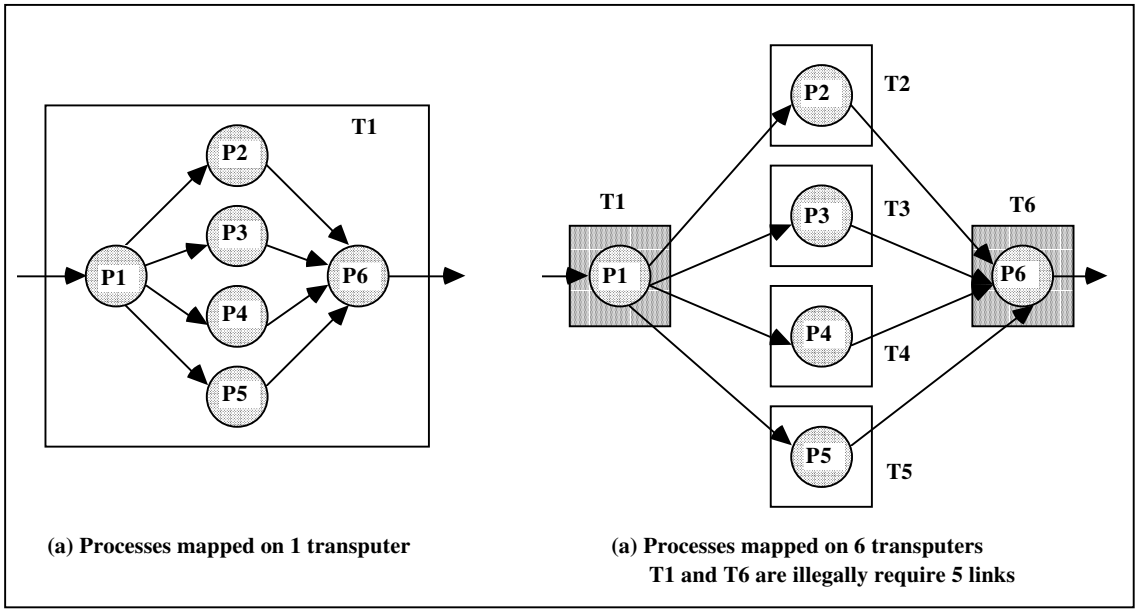

Figure 18. Problem with scheduling invariance

There are several possibilities to tackle the problem. One obvious solution could be to increase the number of physical links on the Transputer chips. However, it would only defer the emergence of the problem but could not really solve it. Another possible remedy could be to write multiplexer/demultiplexer processes for each physical channel to be able to serve for several "user" processes. Though it would mean a general solution for the problem, it would be quite tedious for the Occam programmer to write all of these extra processes and moreover, the program would become significantly slower.

In the 2nd generation Transputer family the scheduling invariance was retained by hardware implemented multiplexer/demultiplexer processes. The number of links remained four in the T9000 Transputer chip but a new hardware unit called the Virtual Channel Processor (VCP) has been introduced. It allows an *arbitrary* number of processes to use each link. These links that are now transparently shared by processes are called as **virtual links**. Each virtual link provides two channels, one in each direction. The VCP unit of the T9000 multiplexes virtual links onto the four physical ones. In order to realise virtual links, several layers of link protocols have been introduced in the 2nd generation Transputer family:

1. Bit level protocol
2. Token level protocol
3. Packet level protocol
4. Message-passing protocol

Bit level protocol. The OS-Links of the first generation Transputers are replaced with DS-Links that have four wires, a data and a 'strobe' line for both the input and

output directions. The strobe line changes state each time when two consecutive bits are identical on the data line. The introduction of the extra strobe line led to several advantageous features. The only limit for the transmission rate is the speed of the receiver. The achievable transmission speed over the DS-Link (200 Mbits/s) is an order of magnitude higher than over the OS-Link. DS-Links also simplify the clock distribution within a system, since the exact phase and frequency of the clock on a pair of communicating devices is not critical.

Token level protocol. There are two types of communication tokens: data and control tokens. The protocol for data tokens is as follows:

< P, 0, <data>$_8$ >

where P represents the parity bit. The second bit is the control bit (C) used to distinguish data (C=0) and control tokens (C=1). The structure of control tokens is:

< P, 1, <control>$_2$ >

The control can be:

00: Flow control token (FCT)
01: End of packet (EOP)
10: End of message (EOM)
11: Escape token (ESC)

The FCT token is used to prevent a sender from overrunning the input buffer (at least 8-token long) of a receiving link. The receiver sends an FCT token to the sender when its input buffer has room to receive another 8 tokens. After sending an 8-token dose, the sender waits until a new FCT token arrives.

Packet level protocol. The VCP unit of the T9000 splits up messages of arbitrary length into packets. Each data packet consists of a header followed by the data field and closed by a terminator. The header identifies the virtual channel and specifies the receiver process. The data field can contain up to 32 bytes. The terminator is either an EOP or an EOM token.

Acknowledge messages are used to implement synchronous communication on a virtual channel. After sending a packet, the sender VCP should wait for an acknowledge message. When the acknowledgement of the last packet has arrived the sender process can resume. After receiving the first packet the receiver VCP sends back the acknowledge message if the receiver process is ready to input from the channel. As a consequence of this strict acknowledgement protocol, sequence numbers of packets are not necessary, even when the network applies adaptive routing scheme.

The acknowledge message is sent as a zero length packet consisting of only a header field and an EOM token. Since virtual links are always built from a pair of virtual

channels, it is not necessary to include a return address in packets. The acknowledgements are simply sent back along the other channel of the virtual link.

Message-passing protocol. Implementation of the virtual link concept in the T9000 processor requires that each physical link should be associated with a list of virtual links waiting to use it. Virtual links are represented by a pair of virtual link control blocks (VLCBs), one on each Transputer. The organisation of virtual link lists is identical to that of the Ready List (Figure 16). When a process executes a send or receive message operation, the computation processor of the T9000 delegates the job of transferring the message to the VCP and deschedules the process. The identifier of the process is saved in the control block which is added to the queue of the associated physical link. After completing the message transfer, the VCP reschedules the waiting process. This mechanism allows that the T9000 computation processor can run another process while the VCP performs the external message-passing protocol.

The series of figures in Figure 19. illustrates the activities of the VCP to implement the external message-passing protocol. Assume that Pi is the sender and Pj is the receiver process on virtual channel C which is associated with physical link L. When either Pi or Pj starts the message operation, registers A, B, and C contain the same information as in case of the internal message passing. The only difference comes from the channel representation which is a VLCB instead of a single word (Figure 19.a). The VCP copies the pointer and length values of the message to the VLCB. The process is descheduled and its workspace is connected to the VLCB by storing the address of its workspace in the VLCB (Figure 19.b). The VLCB is added to the virtual link list of the physical link L.

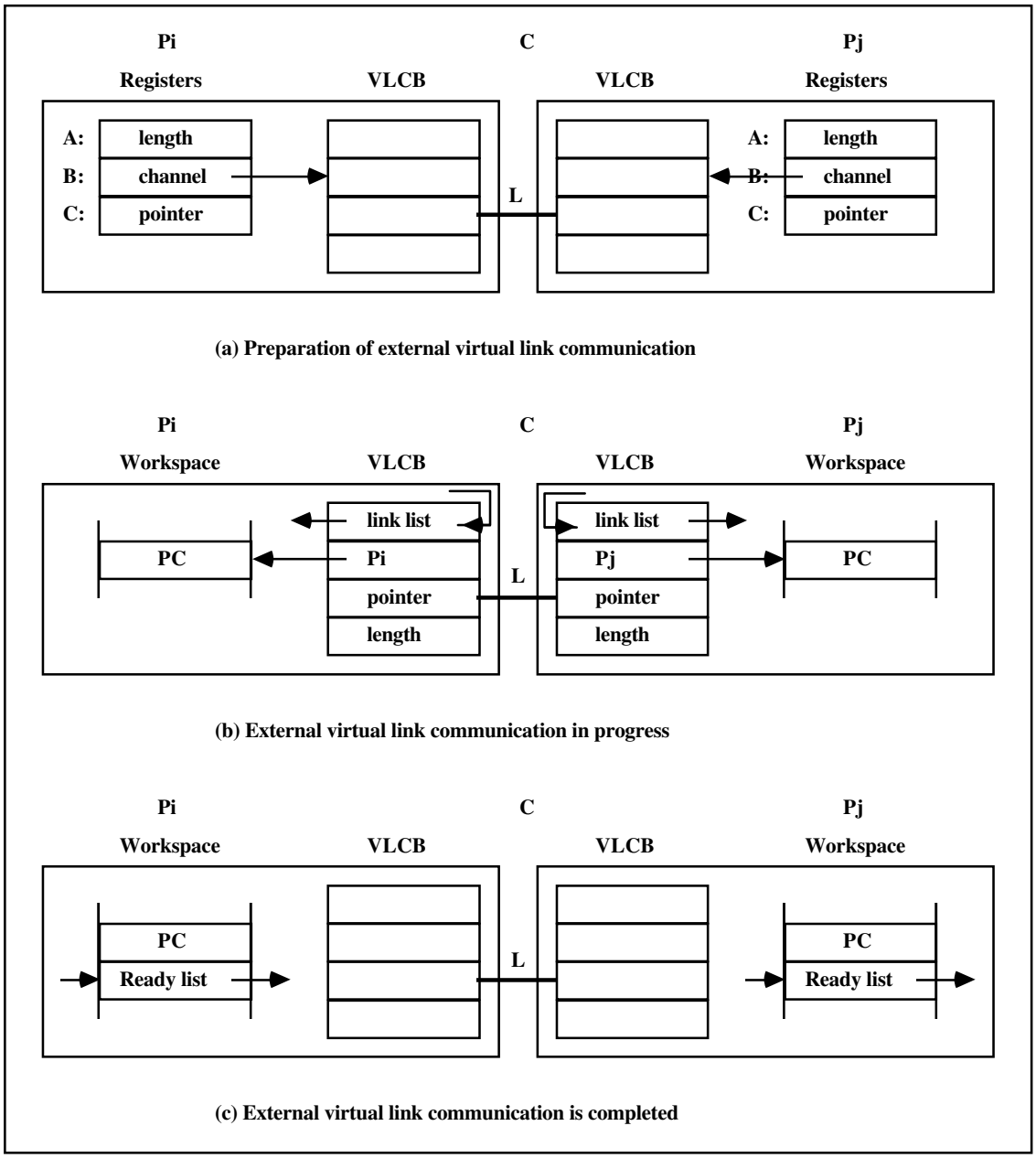

Figure 19. External channel communication between transputers

On the sender side, the first packet of the message is sent via link L when the VLCB becomes the first in the waiting list. Any further packet is sent when the acknowledge message for the previously sent packet has arrived. On the receiver side, there are two possibilities depending on the relative speed of the two processes. If the first packet arrives when the receiver is not ready yet to input message from channel C, the VCP temporarily stores the packet but does not send back the acknowledge message. If Pj is ready for the channel operation, the message is stored to the memory location the pointer of which is stored in the VLCB of Pj and an acknowledge message is generated. When the message transfer is completed (last packet has arrived at the receiver side, final acknowledgement has arrived at the sender side), the VCPs reschedule the corresponding

processes at both sides as shown in Figure 19.c. Notice that the process switching mechanism applied for handling interprocessor communication among T9000 processors has many similarities with the scheme used in coarse-grain multithreaded machines and hence, the T9000 could be considered as a coarse-grain multithreaded processor, too.

It should be mentioned here, that in the T9000 there are two other important communication mechanisms supported by hardware:

1. handling of alternative inputs (implementation of ALT)

2. operations on shared Channels and resources.

However, these topics are not covered here due to the lack of space. The interested reader can find the description of these mechanisms in (May, Thompson and Welch, 1993).

17.4.5 Communication between remote Transputers

The protocol layers introduced for the DS-Links support virtual links through a communication network between remote Transputers that are not directly connected by physical links. In such cases the header of the packets contain the routing information. In the first generation Transputer family, Transputers were connected directly to each other or through the programmable C004 crossbar switch. In both cases the store-and-forward switching technique had to be used. In the 2nd generation Transputer family, the C104 packet routing switch is introduced to connect remote Transputers.

The logical structure of the C104 chip is depicted in Figure 20. It can be considered as a generalised version of the C004 chip since both include a full 32 x 32 non-blocking crossbar switch enabling to connect up to 32 Transputers that are not connected directly to each other. However, the C104 is more than a crossbar switch, it implements a complete routing algorithm in hardware. Like the routing chips of other 2nd and 3rd generation multicomputers, the C104 routing switch realises wormhole routing in order to reduce network latency. It also applies a deterministic, distributed, minimal routing protocol called **interval labelling**.

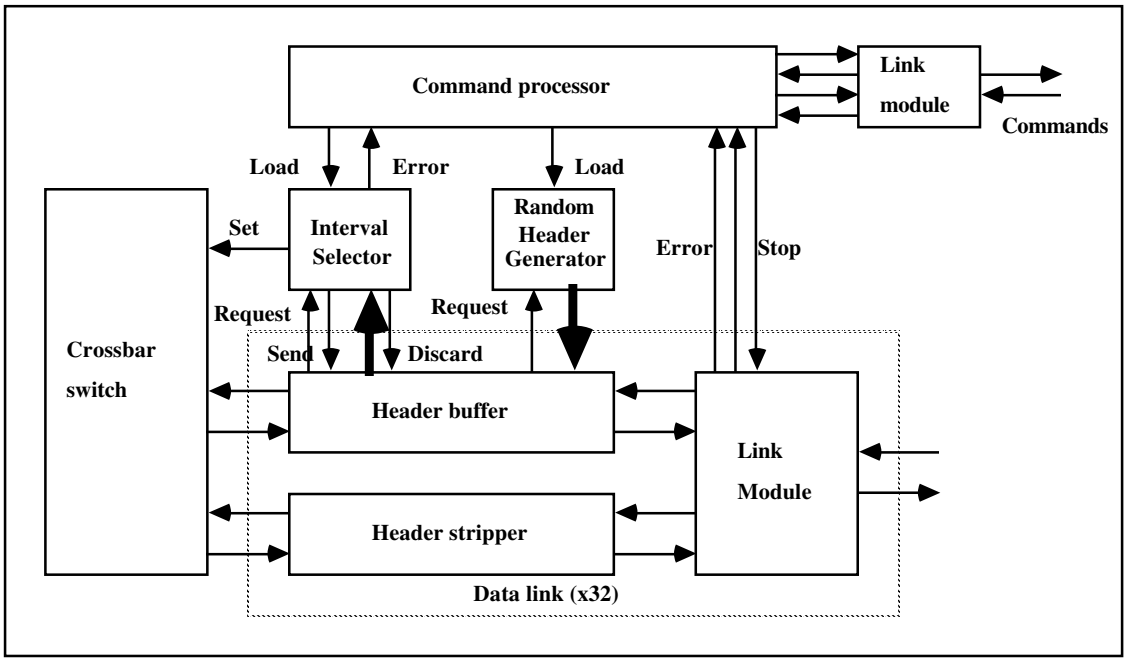

Figure 20. Structure of the C104 packet routing switch

An interval labelling scheme supplies each Transputer with a unique label: (0, 1, ... ,N-1) for an N-Transputer system. Each output link of each router is associated with one or more ranges of labels, called intervals. An interval consists of a set of consecutive labels. The intervals on a router are non-overlapping and each label belongs to exactly one interval. When a packet arrives to the router it will be transmitted on the output link the interval of which contains the destination address (label). The interval labelling scheme requires very simple hardware support; it can be implemented by a single comparison operation. Hence, the C104 chip contains little more than a pair of comparators and interval registers for each output link.

However, a non-trivial question is raised how to assign labels and intervals in an arbitrary network. Fortunately, there exists algorithm that produces deadlock-free interval labelling scheme for any network, though it is not optimal. For regular network topologies, such as rings, trees, hypercubes, and arrays even optimal, deadlock-free interval labelling schemes can be found, i.e. each packet takes one of the shortest paths to its destination. Figure 21. shows an example for a 2-D mesh topology with interval labelling. The interval notation [0,4) means that a header value must be greater than or equal to 0 and less than 4 to transmit the packet on the corresponding output link.

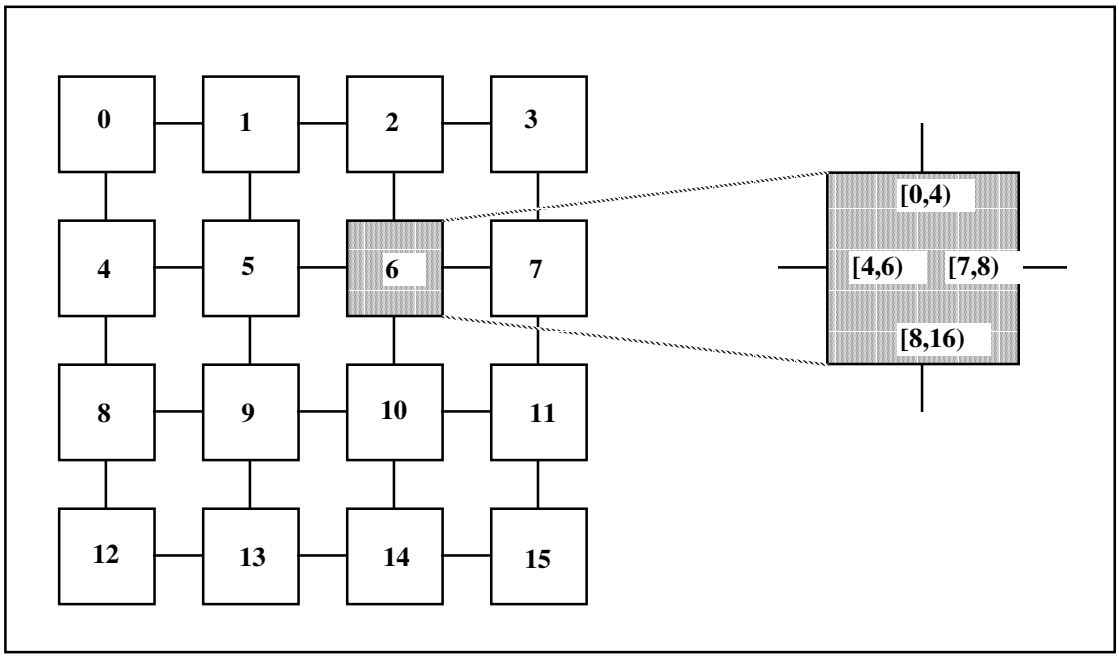

Figure 21. Interval labeling for an array

The interval routing mechanism is implemented in the C104 by the **interval selector unit** that performs the routing decision for each packet. It consists of 35 base and limit comparators which are connected to a pair of registers as shown in Figure 22. The values of the base and limit registers are loaded via the command link of the C104 (Figure 20) according to the interval labelling algorithm. Such base and limit register values are shown in Figure 21. for a 2-D mesh topology. Notice that any interval can be assigned to any link and even more than one interval can be assigned to any link. The SelectLinkn register contains the identifier of the link associated with the interval. A comparator enables a SelectLinkn register if the header value is equal to or greater than the value of the connected base register and less than the contents of the limit register.

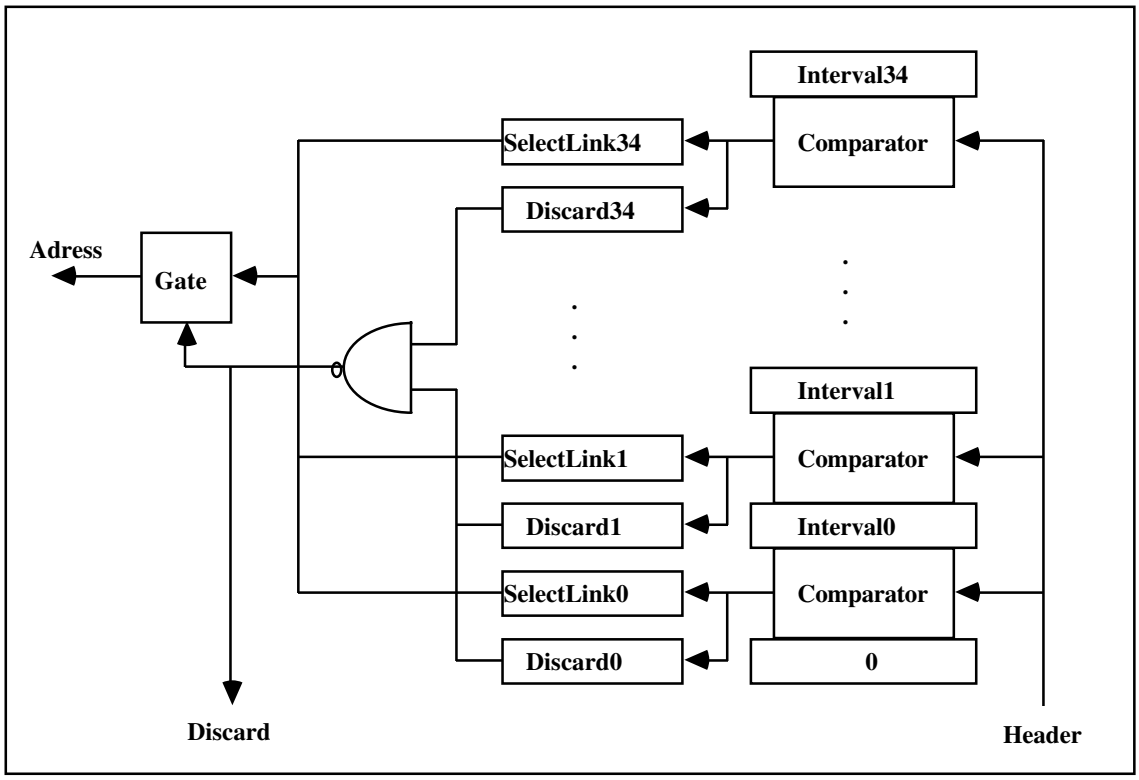

Figure 22. Interval selector unit of the C104

When a packet arrives to the C104 router, the header is passed to the interval selector unit. It selects one of the output links based on the header and the pre-programmed base and limit registers. Once the crossbar switch is set up according to the header, the data tokens are passed through directly until a terminator token arrives.

The application of interval labelling does not make it possible to route messages on an arbitrary path or to route messages via a hierarchy of networks. The introduction of **header deletion** can solve both of these problems. Header deletion mode is where any output links of the C104 router can be set to remove the header of the packet. As a result, the first data byte of the packet will behave as a header in the next node of the message path. In case of header deletion the use of two headers can separate addressing a node in the network from addressing a virtual channel on the T9000. The first header serves to route the message in the network and it is removed when the message reaches the destination node. The second header selects the target virtual channel residing on the destination node. The second typical application of header deletion is related with the hierarchical composition of networks. For example, when a single Transputer node is substituted by a multi-Transputer node, the first header directs the message to the node, whilst the second header defines one of the Transputers within the multi-Transputer node. Header deletion in the C104 router chip is implemented by setting the DeleteHeader0-31 flag for a given link.

Another important problem to be solved in a network is the hot spot avoidance. When too many messages are routed through the same node or link, it results in the drastic reduction of throughput since, most arriving packets will be delayed for an unpredictable length of time. Such a node or link is called **hot spot**. A simple method to avoid the occurrence of hot spots in a network is to realise a two-phase routing in which the first phase realises a random routing sending the message to a randomly selected node and in the second phase the message is routed from this node to the original destination node. This scheme referred to as **universal routing** was designed to minimise delay in heavily loaded networks. Though it increases latency and reduces maximum throughput, it was proven by both simulation and theory that universal routing guarantees that the worst-case performance is not far below the maximum performance, whereas without using the universal routing method the worst-case performance can be several orders of magnitude worse than the highest performance.

The C104 chip supports universal routing by two hardware mechanisms. Firstly, any input link can be programmed to randomly generate a new header by the Random Header Generator (see Figure 20). Afterwards the original header will behave as the first data token of the packet. Secondly, the Discard registers in the Interval Selector Unit support the recognition of the end of the first phase of the universal routing and discard the randomly generated header (Figure 22). As a result, the original header becomes again the header of the packet determining the destination node of the network.

17.4.6 T9000 based machines

The second generation Transputer family is a good candidate to build even massively parallel computers due to the following features of the T9000:

1. The T9000 is a complete 32/64 bit computer on a single chip offering 25 Mflops floating point arithmetic and 200 MIPS peak performance.

2. The 16 Kbyte fast on chip RAM can be used as an internal cache memory increasing the performance of the memory interface.

3. A very fast hardware scheduler supports process switching on two priority levels.

4. A high speed communication subsystem provides four bidirectional data links with a bandwidth of 20 Mbyte/s each. Unlimited number of virtual links can be established over the physical links. A dedicated communication processor enables simultaneous computation and communication in a well-balanced way.

Building a communication network for a massively parallel multicomputer is further supported by the C104 router chip. This chip can connect 32 links to each other via a 32 x 32 nonblocking crossbar switch with sub-microsecond latency. Recognising these

advantages of the 2nd generation Transputer family, two companies Parsytec and Parsys have decided to build parallel computers based on these chips.

Parsytec GC

The Parsytec GC (GigaCube) machine has a 3-D cube interconnection topology which can connect up to 16,384 nodes. These nodes are organised in clusters and gigacubes which are connected by three networks:

1. Data network (D-network)
2. Control network (C-network)
3. I/O network

A cluster contains 17 T9000 Transputers which are connected to each other and to the neighbour clusters by 4 C104 routing chips. From the 17 Transputers only 16 are visible to the user, one is reserved as redundancy to provide fault-tolerance. The structure of the cluster is shown in Figure 23. The four links of each Transputer are connected to the four C104 routers. Each router provides two links to all the six neighbour clusters and to the I/O subsystem. It results in a total of 8 connecting links between any two neighbour clusters with a 160 Mbyte/s bidirectional bandwidth.

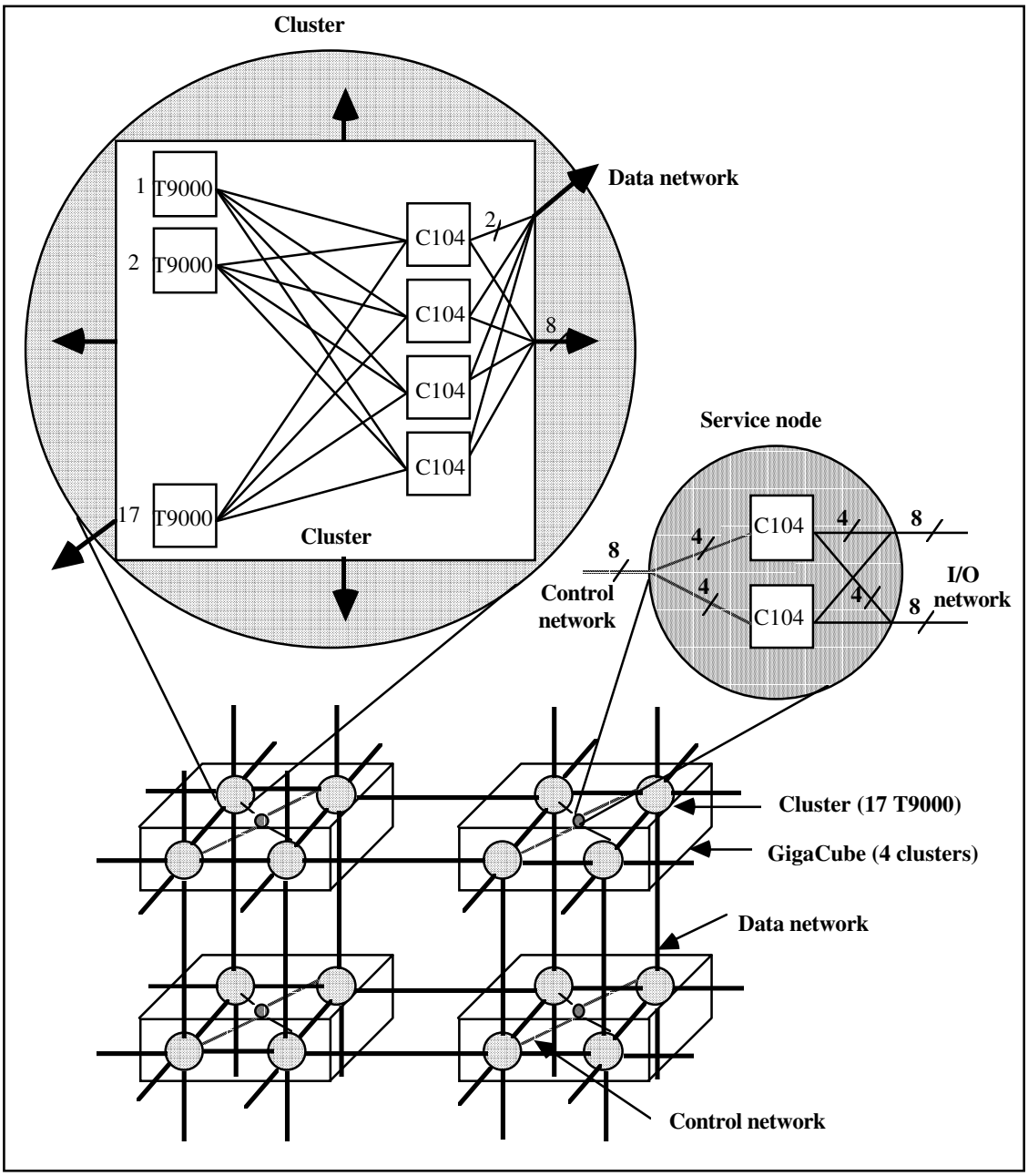

Figure 23. Structure of the Parsytec GC machine

The smallest increment in the Parsytec GC machine is the GigaCube which consists of four connected clusters and a service node. It contains 64 processors organised in a 4x4x4 cube with a peak performance more than 1 Gflop. The service node is integrated into the control network and is used for booting, controlling and monitoring the GigaCube. The service nodes of the different gigacubes are also connected to each other within the C-Network that has a 3-D cube topology similarly to the D-Network. The service node collects the 8 I/O links from each connected cluster by two C104s and provides overall 32 I/O links with 640 MByte/s bandwidth between the GigaCube and the outside world.

SN 9800 and TN300

A collaborative project among Parsys, Telmat, Meiko and Parsytec under the ESPRIT programme GPMIMD (General Purpose MIMD) led to the design of a T9000 based massively parallel computer architecture that was commercialised by Parsys and Telmat with the SN 9800 and the TN300 products, respectively. The main design issue was how to construct a C104 based communication network that maximises communication performance even under *global* communication patterns in large machines containing up to 2048 processors.

A **multi-layer switch architecture** has been developed with an orthogonal board structure. A two-layer system is shown in Figure 24. to illustrate the organisation scheme of the interconnection network within a single subtrack. At the first layer 8 or 16 T9000 processors are connected to 4 C104 switches on each of the 16 node boards resulting in a 128 or 256 node system. At the second layer 8 switch cards are used with 4 C104 switches on each to connect processors of different node boards. This architecture can be expanded up to 2048 nodes by the employment of a third layer of switches in an additional subtrack connecting multiple subtracks.

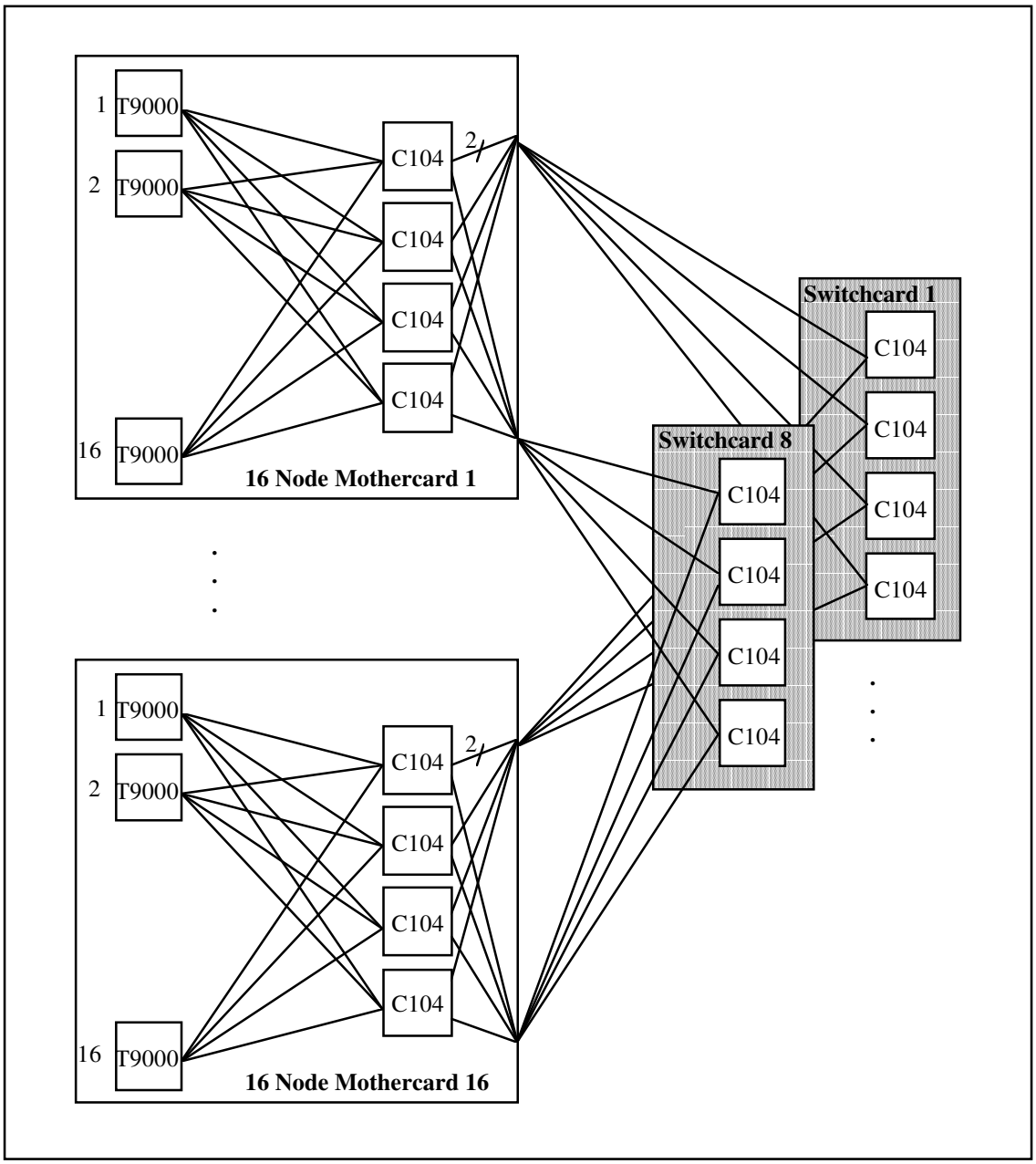

Figure 24. Two-layer interconnection scheme in Parsys SN 9800

17.5 Coarse-grain multicomputers

In the third generation coarse-grain multicomputers, the three main components of message-passing parallel computers (computation processor, communication processor and router) are clearly separated and each of them is realised as a powerful, specialised component. Accordingly, they employ commodity superscalar RISC processors as computation processors since these can deliver the highest achievable Mflops. The difference among them appears in the choice of the communication processor. Some of them use the same RISC processor for organising communication as the employed computation processor. The advantage of this architecture is on one hand the

homogeneity and, on the other hand, when communication needs are not demanding, the communication processor can support the computation processor by doubling the achievable Mflops per node. From now on we call these coarse-grain multicomputers as **homogeneous multicomputers**. Notable representatives of this group are the Intel Paragon and machines designed in the framework of GENESIS and MANNA projects.

The second group employs different processor types for computation and communication purposes. This type of message-passing computers will be called as **hybrid multicomputers**. Computers in this group are the Parsytec GC PowerPlus, Transtech Paramid, IBM SP1 and SP2. In the former machines the T805 Transputer is applied as communication processor. Though it turned out that the Transputer is not powerful enough as a computation processor, its capability of efficiently handling 4 independent asynchronous links nominates it as a capable communication processor.

In the third type of multicomputers custom designed communication processors are employed in order to achieve the highest possible performance. A representative of this type is the Meiko CS-2 computer which indeed delivers the best performance of all the current message-passing computers. The performance as distinguished design issue in the CS-2 is demonstrated by the fact that CS-2 is the only third generation coarse-grain multicomputer where vector processor units can be employed in the processing node. Although the application of vector processor units was considered in the GENESIS project too, it was finally rejected due to the time requirements of developing such a unit. It should be mentioned that the vector unit of CS-2 was not developed by Meiko, rather they employ Fujitsu uPV.

Third generation coarse-grain multicomputers differ in the structure and implementation of the communication switch unit (or router), too. Basically two concepts are popular. Either they apply custom designed communication switch unit (for example in Paragon, CS-2, MANNA, SP2) or use the C004 or C104 communication switch units of the Transputer-family.

Naturally, a crucial design factor is the choice of the interconnection network topology. From this point of view there is no clear trend what the preferred topology of future multicomputers would be. One obvious change to the previous generations is that hypercube topology has lost its significance due to the application of wormhole routing where the communication distance among nodes is not a crucial feature anymore. Consequently, those topologies are favourable where the implementation cost is radically less than in the hypercube topology and yet they assure good scalability. These topologies are mainly the mesh, cube and fat tree which appear respectively in the Paragon, GC PowerPlus and CS-2 machines. The design space of coarse-grain

multicomputers is shown in Figure 25. In the rest of this section the main representatives of the three groups are reviewed.

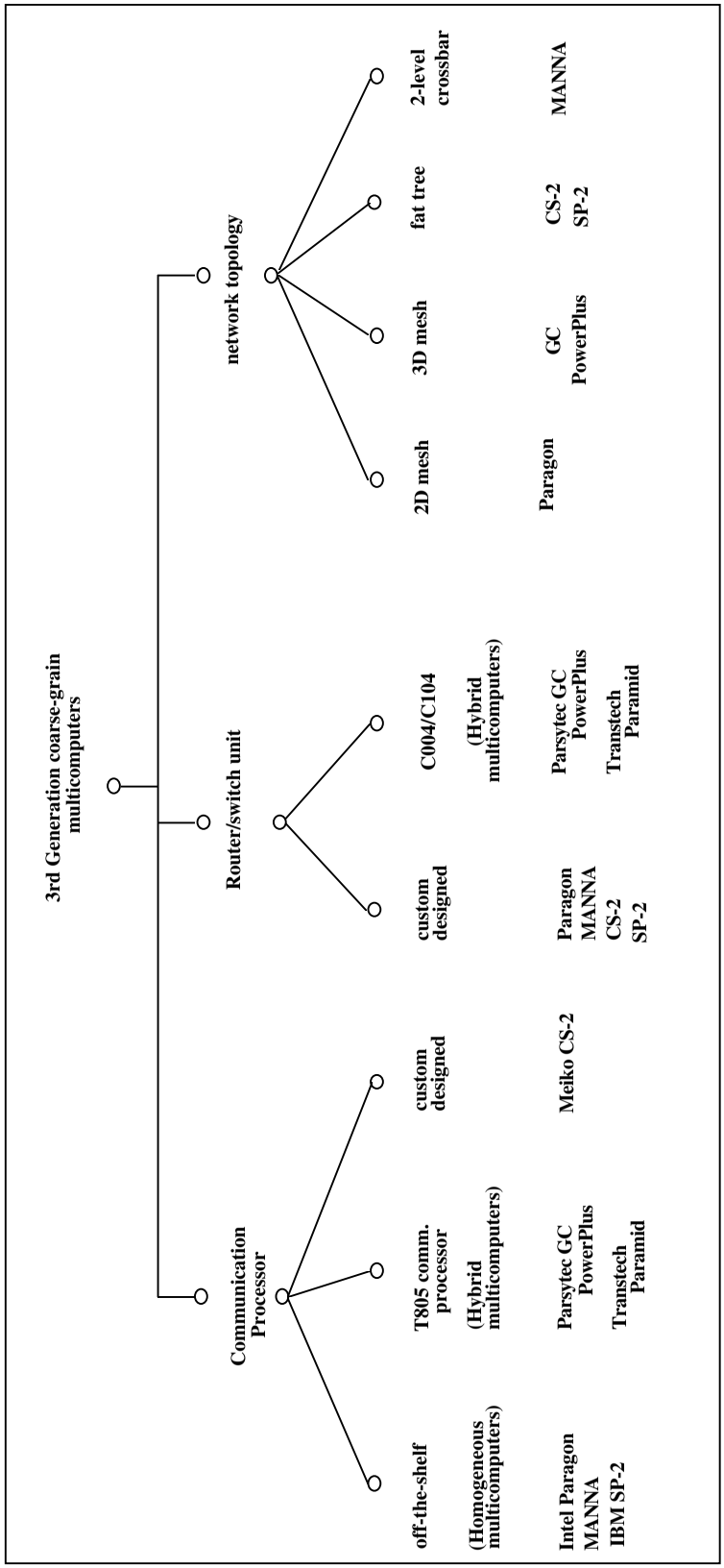

Figure 25. Design space of 3rd generation coarse-grain multicomputers

17.5.1 Homogeneous architectures

Intel Paragon

The Intel Paragon machine is radically different from the previous Intel multicomputers. All of those machines (iPSC/1, iPSC/2 and iPSC/860) were based on the hypercube topology and none of them applied the wormhole routing technique. In the Paragon machine, nodes are connected in a 2-D mesh topology and communication is based on wormhole routing. Unlike the other machines, the Paragon contains three different node types:

- compute nodes
- service nodes
- I/O nodes

The various nodes are organised into several types of partitions inside the mesh as it is shown in Figure 26. The largest partition is the Compute Partition where compute nodes are employed to perform numeric computation. The service and I/O nodes are realised as General-purpose (GP) nodes, while compute nodes appear as Multiprocessor (MP) nodes.

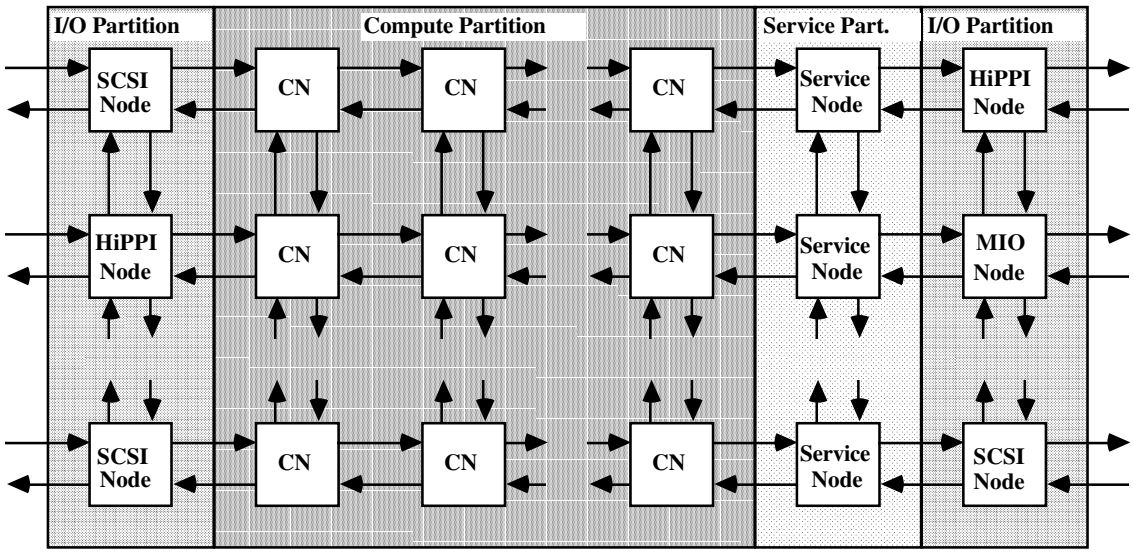

Figure 26. Structure of the Paragon machine

In the GP node two i860 processors are employed, one of them is the application processor and the other is dedicated to message processing. GP nodes contain an expansion port for adding an I/O interface such as HiPPI or SCSI. I/O nodes can support input/output connectivity, while service nodes provide interactive use for several users.

The MP node is organised as a small size shared memory multiprocessor in which four i860 application processors work together in a symmetric way. The node memory is

shared among the four processors and each of them is supplied with a 256KB second-level cache. (Inside the i860 processor chip a 16KB instruction and a 16KB data cache are available as first-level caches.) Beside the application processors MP includes an additional i860 chip as a message processor for improving latency and throughput of message passing. The structure of MP and GP nodes are shown in Figure 27.a and 27.b, respectively.

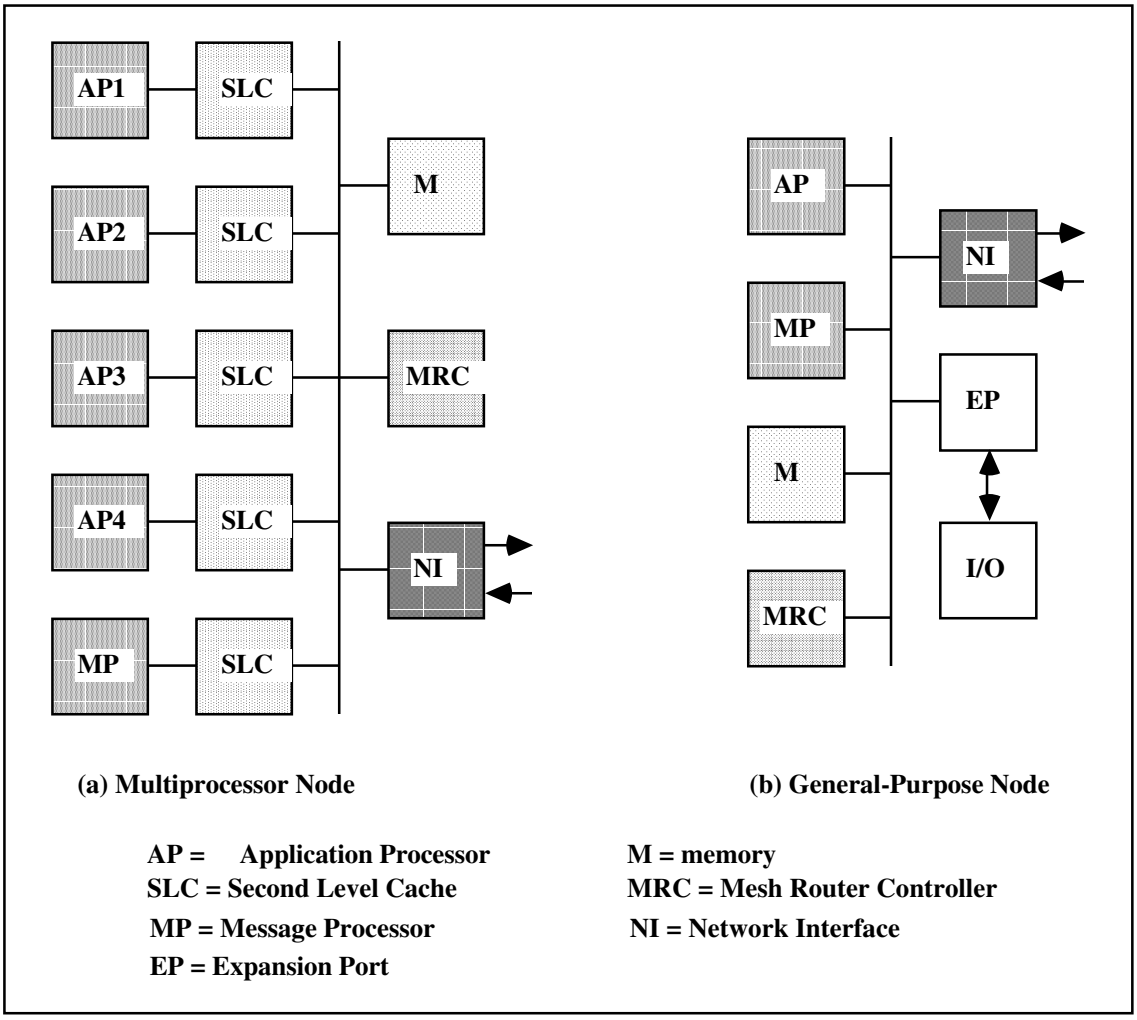

Figure 27. Structure of nodes in the Paragon

Message passing is initiated by the application processor but it is actually performed by the message processor which handles message-protocol processing. Message handling software is executed from the internal cache of the message processor. The advantage of this organisation is that application processing and communication are overlapped freeing the application processor of time-consuming context switches. Furthermore, the message processor realises such global functions as broadcasting, synchronisation and global reduction calculations (e.g., global sum).

Message routing is realised by the Mesh Router Controller (MRC) of the node. MRC is a custom-designed chip which realises the third component of multicomputers, the

communication switch unit (or router). In the Paragon, MRCs are connected in a two-dimensional mesh and they form an independent routing system for the actual transmission of messages. MRCs route messages between any two nodes at speeds of 200 MB/sec full duplex. The internal structure of the MRC is a 5 x 5 crossbar switch supplied with flit buffers and input controllers at each direction. The MRC handles 4 links to the neighbours and 1 link to the local node.

Communication is further supported in a node by two block transfer engines and by the Network Interface Controller (NIC), a custom VLSI chip. These three units together permit simultaneous inbound and outbound communication at 200 MB/sec. NIC also provides a pipelined interface between the node memory and the coupled MRC of the node.

17.5.2 Hybrid architectures

In hybrid architectures, high performance commodity microprocessors are employed as computation processors while for the purpose of communication processors less powerful microprocessors are built into the node. In several cases the communication processor is a Transputer the internal structure of which was designed to support communication intensive applications. There were two motivations leading to the design of hybrid architectures employing Transputers as communication processors:

1. It turned out that the T9000 Transputer can not compete with the floating point performance of advanced superscalar processors.

2. Whilst superscalar RISC processors generally have hardware support for shared memory parallel architectures, they have no provision for communication links necessary for a distributed memory system.

In order to combine the advantage of superscalar RISC processors and Transputers, hybrid architectures integrate high performance superscalar RISC processors with 1st generation Transputers in the processing nodes of the multicomputers. The powerful RISC processors play the role of computation processors whereas Transputers work as communication processors. Based on these principles several machines were built: Parsytec GC/PowerPlus, Transtech TTM200 and Paramid. The advantage of such hybrid architectures is that message routing operations are performed on the Transputer and do not detract from the processing power of the computation processor. There are several approaches to distribute the operating system among the processors of a hybrid architecture node:

1. The operating systems resides on the computation processor and uses the Transputer just as a communication device.

2. The operating system and the application code run mainly on the Transputer and the computation processor is used simply as a co-processor.

Parsytec GC/PowerPlus

In the GC/PowerPlus machine, a node is analogous to a T9000 Transputer as it is demonstrated by Figure 28. The two PowerPC 601 CPUs replace the CPU of the T9000 and they have much higher numeric performance than the T9000. Four T805 Transputers are employed to substitute the VCP (Virtual Channel Processor) and link interfaces of the T9000. Together they provide 16 bidirectional Transputer links instead of the four links of the T9000. They realise virtual channel processing by software over the 16 physical links and that is why they are also called Virtual Channel Processors (VCP).

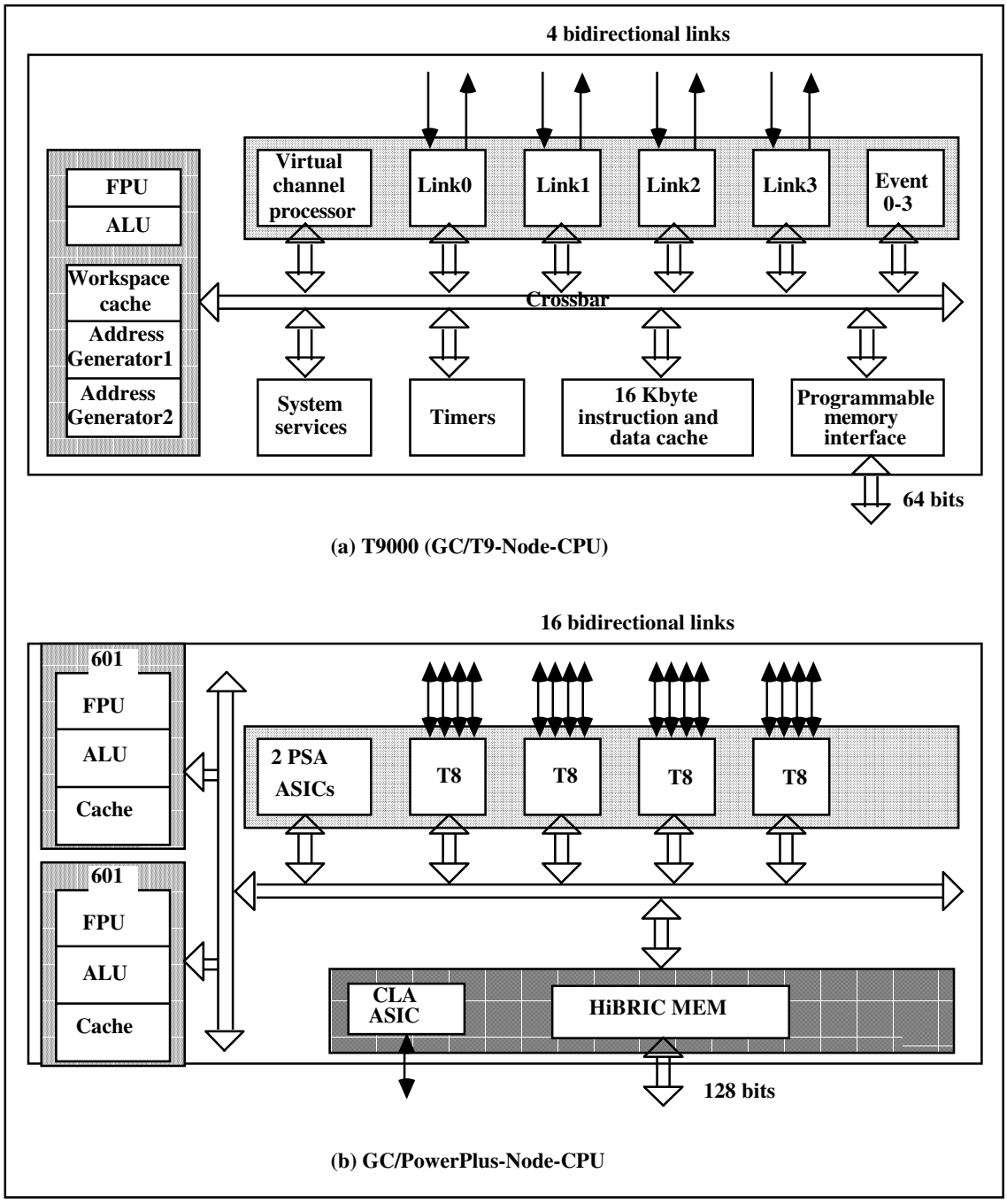

Figure 28. Comparison of the T9000 and the GC/PowerPlus-Node-CPU

The GC/PowerPlus machine implements a variable balance architecture concept whereby the number of CPUs, the number of communication processors, and the amount of memory per node can be configured to customers' needs. The machine node is a multiprocessor with physically shared memory among computation and communication processors. The CPUs and VCPs are able to signal each other with low overhead and their communication takes place in the user-address space. Notice that while the T9000 is a single chip implementation of this concept, the GC/PowerPlus node is a multi-chip implementation in which the balance between CPUs and VCPs can be customised. In the

GC/PowerPlus node, an ASIC interface has been designed to turn the four Transputers efficient bus-masters at the PowerPC bus.

The interconnection network has a 3-D mesh topology derived from the original GC-architecture. One GC/PowerPlus cube consists of 4 clusters of 4 + 1 nodes interconnected by 16 C004 switch units. This topology is logically equivalent to a 4 x 4 grid of nodes as illustrated in Figure 29. The application of a redundant node in each cube renders fault-tolerant the GC/PowerPlus machine. Like other third generation multicomputers, the GC/PowerPlus applies wormhole routing, too. The next version of the GC/PowerPlus will support the ATM (asynchronous transfer mode) protocol which is the forthcoming industry standard for communication.

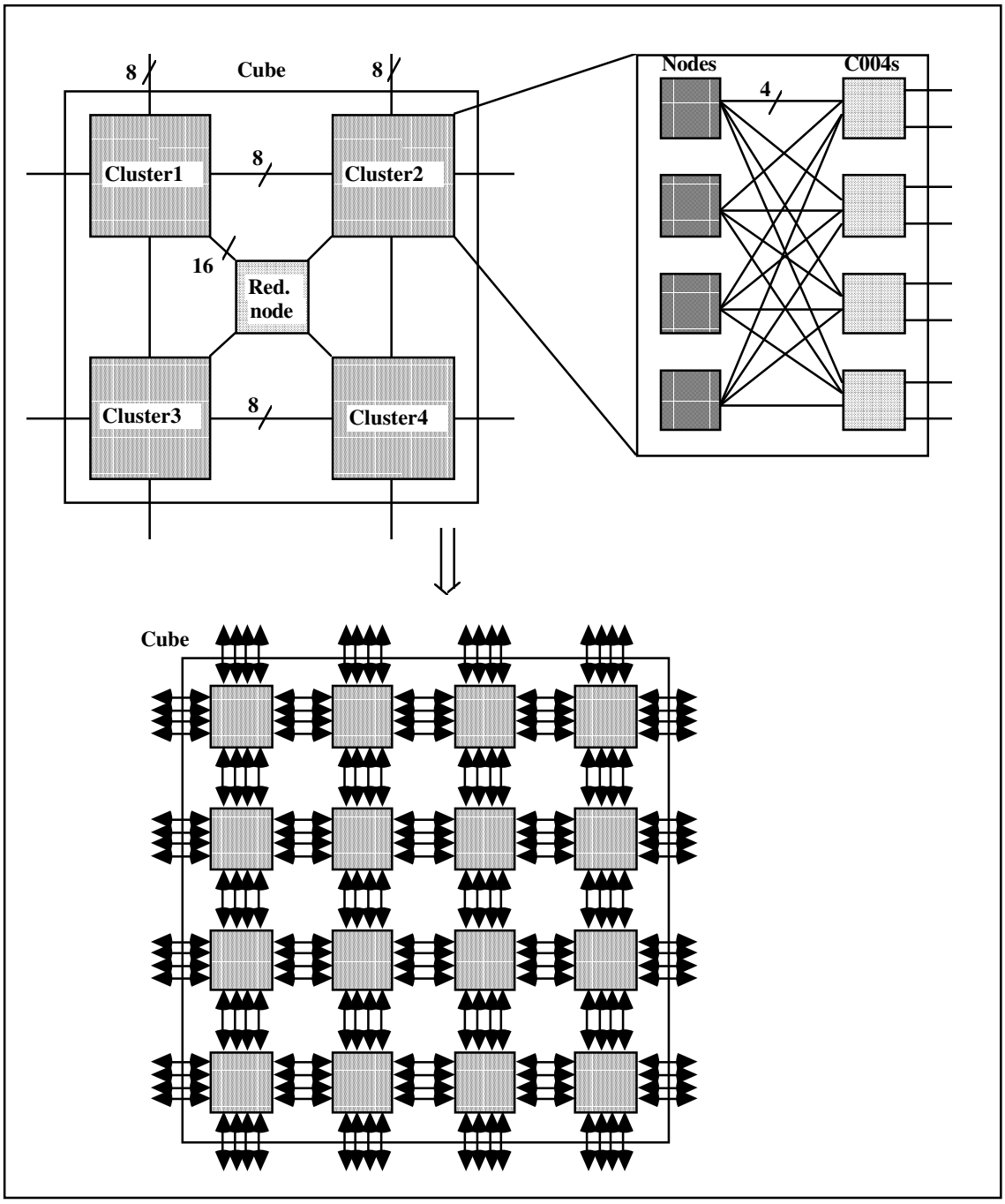

Figure 29. Interconnection network of a cube in the GC/PowerPlus

Though the communication model of the GC/PowerPlus is based on CSP, it involves a coarse-grain multithreaded concept in the sense that it emulates the virtual channel processor concept of the T9000. Application threads (processes) running on the CPU request communication service from the communication processors by placing the channel operation command and parameters in the shared memory and giving an interrupt to one of the VCPs. The application thread is then placed in a waiting queue just like it happens in the case of the T9000 (see Section 17.4.3). Meanwhile the VCP executes the requested remote communication, other threads can use the computation

processor. When the communication is finished the VCP notifies the CPU by an interrupt. The computation processor can access the message in the shared memory and can reschedule the corresponding waiting thread. The cost of a message then is the CPU-cycles lost due to scheduling overhead rather than the much longer time would be necessary to wait for a message transmission. As a result the average network round-trip latency which lays in the range of architectures with hardware-routers (about 100 us) is actually hidden except for the message setup-time (including thread scheduling) of only 4-5 us.

IBM SP2

The IBM SP2 is another hybrid architecture, however, instead of Transputers it employs Intel i860 XR processors for supporting communication on nodes. The computation processor is the new POWER2 processor of the RISC System/6000 workstations. The main objective of the SP2 design was to integrate RISC System/6000 workstations into a distributed system by adding a powerful communication subsystem. The innovative features of the SP2 communication subsystem enumerates as follows:

1. Application of a high speed communication adapter containing an Intel i860 XR communication processor.

2. Introducing the Vulcan switch chip for connecting SP2 nodes.

3. Application of a new protocol that maintains bandwidth over long links.

4. Introducing a unified data and service network.

The SP2 is built of 16-node bidirectional logical frames shown in Figure 30.a. A logical frame consists of a two-stage interconnection network where nodes are connected to the first stage and the second stage provides free links to connect the logical frame to other frames in order to construct large networks of computing nodes. Topology to create a 128-node systems is shown in Figure 30.b. The intermediate switch boards of Figure 30.b are the same as the logical frame shown in Figure 30.a but differently drawn to emphasise the similarity of the topology to the fat tree network.

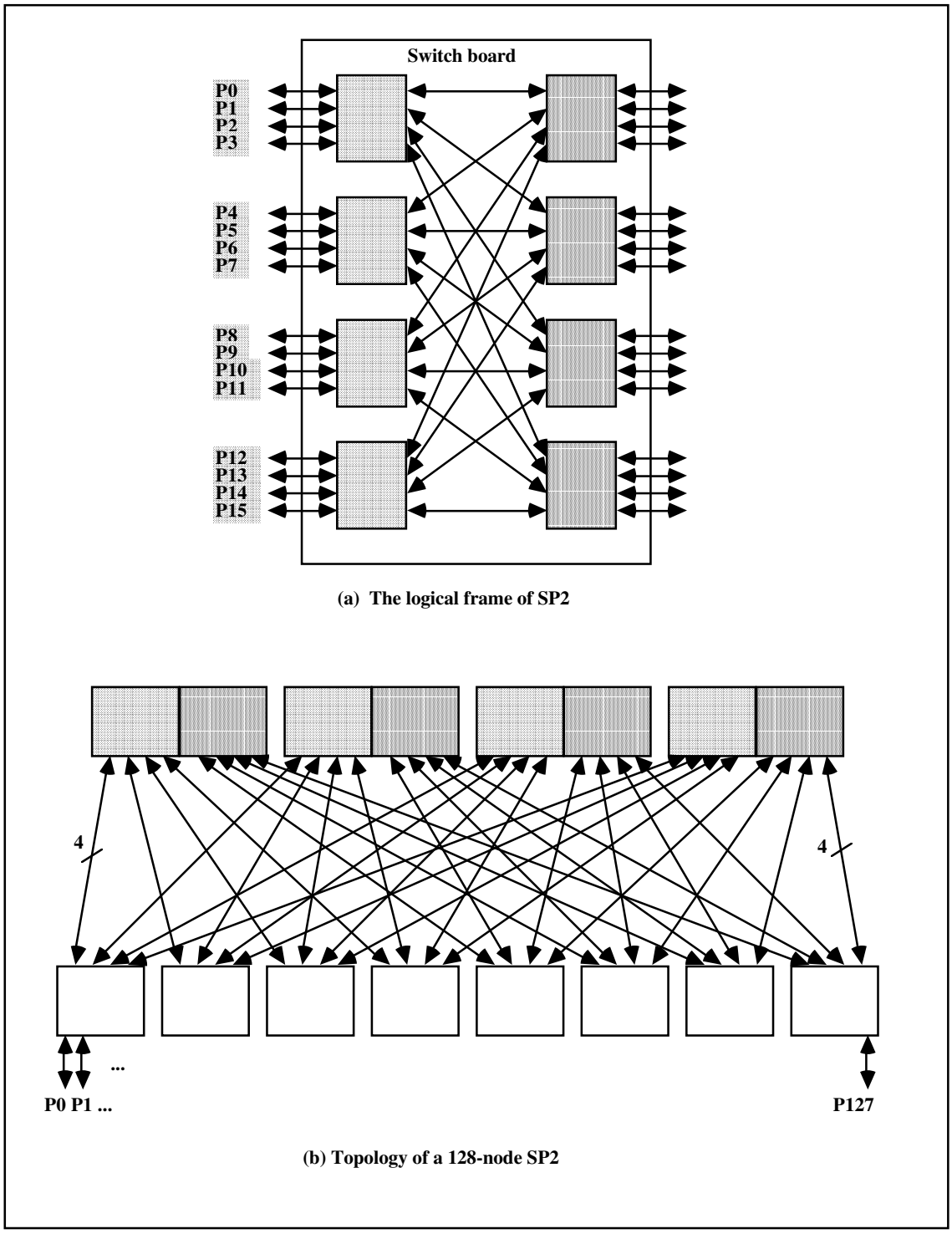

Figure 30. The interconnection scheme of SP2

The switch boards of logical frames contain 8 switch units which are wired as bidirectional 4-way to 4-way elements though, physically they are 8 input 8 output devices. The applied switching technique is a modified wormhole routing, called the **buffered wormhole routing**. The main difference appears in the application of relatively large shared buffer, called the **central queue**, in each switch unit. The central queue is dynamically allocated to blocked packet flits from any input channel of the switch unit.

The width of the channels is identical to the length of the flit (one byte) and thus, one cycle is enough to read one flit on any channel.

The routing protocol is deterministic, minimal and source-based (see Figure 10. classification of routing protocols). Messages are packetised and each packet contains route information in route flits as shown below:

length, route 1, ... ,route r, data 1, ... ,data n

Each route flit contains two route fields and a flag bit indicating which the valid one is. When the first route field is read, the flag bit is changed and the flit is forwarded to the next switch unit. When the second route field is read by the next switch unit, the flit is discarded from the message and the length field of the packet is decremented.

The two main components of the SP2 communication subsystem are the communication adapter and the Vulcan switch unit.

Communication adapter connects the POWER2 processor through the Micro Channel interface to the switch units. It turned out in the SP1 system (an early version of SP2) that SP1 could not use all the link bandwidth available due to lack of local intelligence in the communication adapter. It was the computation processor that had to transfer data to and from memory and perform all communication processing. The SP2 communication adapter, shown in Figure 31, introduces a variety of techniques to offload communication tasks from the computation processor:

(a) Incorporates an Intel i860 XR microprocessor with 8 MByte of four-way interleaved DRAM for communication coprocessing.

(b) Uses the Micro Channel's bus master and streaming capabilities.

(c) Moves CRC generation into hardware.

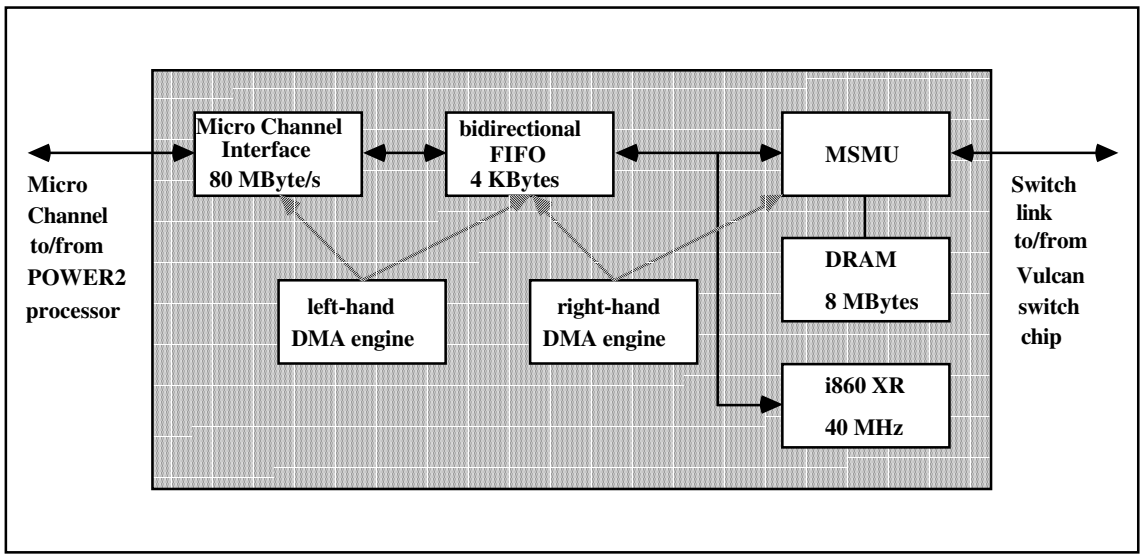

Figure 31. Structure of the SP2 communication adapter

Notice on Figure 31. that data transfer through the communication adapter involves two buses: the Micro Channel and the i860 bus. The two buses are connected via a 64-bit-wide, 4 KByte bidirectional FIFO based on two DMA transfer mechanisms: the "left DMA engine" transfers data for the Micro Channel and FIFO; the "right DMA engine" for the i860 bus.

The i860 initiates both receive and send DMA transfers. In case of a receive operation the i860 writes the header into the FIFO. Afterwards the right-hand DMA engine transfers data received from the network into the FIFO. When the header of the message reaches the head of the FIFO, the left-hand DMA engine decodes it and transfers the message data from the FIFO onto the Micro Channel. In case of a send transfer the i860 writes a header into the FIFO for requesting data from the Micro Channel. When the header reaches the head of the FIFO, the left-hand DMA engine decodes it and transfers the message data from the Micro Channel into the FIFO. It also indicates the completion of the transfer by incrementing a hardware counter. In case of a nonzero count the i860 writes a header which initiates the work of the right-hand DMA engine to transfer data from the FIFO to the MSMU (Memory and Switch Management Unit).

The left-hand DMA engine supports all Micro Channel transfer rates up to 80 MB/s which is roughly equal to the bandwidth of the switch channel in the Vulcan chip. The i860 bus bandwidth is 160 MB/s and thus, half of it can be used by the right-hand DMA engine and half remains for executing the i860 protocol activities like to initiate DMA, query MSMU and DMA status, load and store packet headers in the MSMU, and so on. Accordingly, the communication adapter can perform four operations simultaneously resulting in high throughput:

- left-hand DMA operation
- right-hand DMA operation
- i860 protocol execution
- MSMU switch access

The switch unit is realised by the Vulcan switch chip (Stunkel & al. 1994) illustrated in Figure 32. It provides full crossbar connection among 8 input and 8 output channels by 8 receiver and 8 transmitter modules, and an unbuffered 8x8 crossbar. Each channel is one flit (one byte) wide. Between any two input and output channels the chip provides two data paths. An unbuffered data path through the crossbar unit requires only 5 cycles of latency. This data path can be used when the incoming packet encounters no contention for the selected output channel. Otherwise, the receiver module deserialises the incoming packet into 8-flit chunks and sends them to the central queue for temporary buffering.

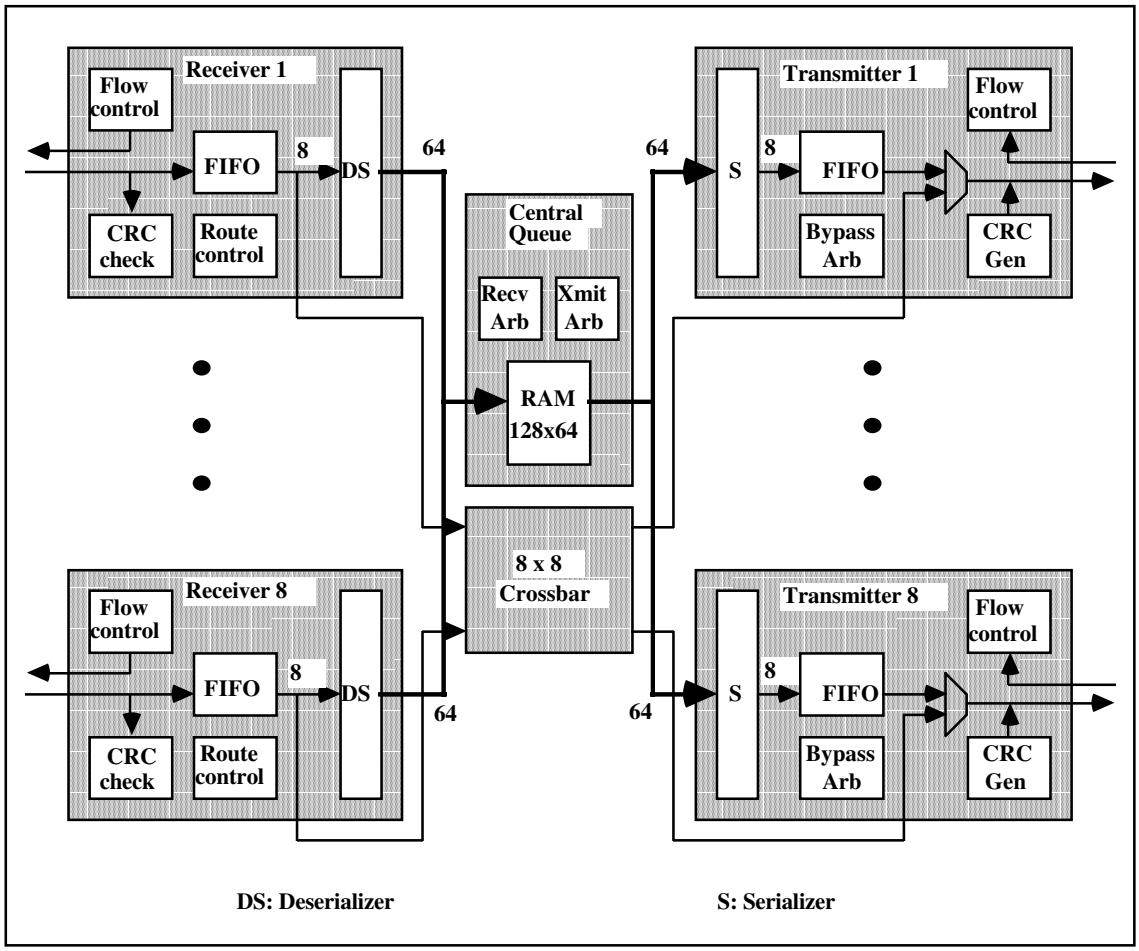

Figure 32. Structure of the Vulcan switch unit (Stunkel & al. 1994)

The central queue can accommodate up to 128 eight-flit chunks in a 128x64 RAM memory which is allocated dynamically for the output channels according to demand. For each output channel a FIFO list is maintained for queuing message chunks waiting for the availability of that channel. The bandwidth into and out of the switch chip is eight flits per cycle that matches the bandwidth of the central queue which can serve one receiver and one transmitter module in each cycle as long as free queue space and message data are available in the central queue memory.

Packet chunks are divided into critical and non-critical chunks in order to prevent starvation. Chunks that are ready for immediate transfer to their destination transmitter are called critical chunks. These have higher priority than non-critical chunks. The transmitter module accepts packet chunks from the central queue, serialises them, supplies them with CRC and transmits them on the corresponding output channel.

The unified data and service network is one of the main innovative features of SP2. The same network is used both for normal data transmission and for network service. Accordingly, the network operates in one of the two modes:

- run mode

- service mode

The two modes apply different communication protocols. In run mode the link protocol is *token based* to tolerate link propagation delays of several cycles. The input port sends token signals to its partner output port. Each token represents the readiness of the input port to accept another flit. The service mode applies a *circuit switching protocol* for reliable communication. The separation of run mode and service mode makes it possible to collect monitoring information without perturbing run mode operations.

17.6 Summary

The three generations of multicomputers clearly show the trends of progress in this class of parallel computer architectures. Regarding the applied interconnection networks the turning point was the discovery of wormhole routing which led to the reduction of communication latency with three orders of magnitude. Another consequence was that communication latency became nearly the same for neighbour nodes and remote nodes. As a result, the network diameter that was the most important network parameter in case of the first generation multicomputers has lost its priority. In the third generation multicomputers scalability became the most crucial issue and hence those topologies became popular where the node degree is constant. Thus, gradually the fat tree and the 2-D and 3-D mesh topologies took over the originally favoured hypercube networks. This tendency of changes in communication networks is shown by the sixth column of Table 4 and by the first four rows of Table 3.

	1st generation	2nd generation	3rd generation
Topology	hypercube	hypercube/mesh	2-D and 3-D mesh
Switching technique	packet (store and forward)	circuit or wormhole	wormhole
Comm. latency of 100 byte message Neighbour (usec) Remote (usec)	 2000 6000	 5 5	 0.5 0.5
Node organisation	single computation processor	computation processor + router	computation proc + communication proc. + router
Typical node number	64	256	1024
Node memory (Mbytes)	0.5	4	32
MIPS	1	10	100
Mflops scalar	0.1	2	40
Mflops vector	10	40	200
System memory (Mbytes)	32	1K	32K
System MIPS	64	2560	100K
System Mflops scalar	6.4	512	40K
System Mflops vector	640	10K	200K

Table 3. Comparison of the three generations of multicomputers (extended from Athas and Seitz 1988)

The node organisation, too went through significant changes during the three generations. In the first generation systems the computation processor, communication processors and routers were not separated, all of these functions were realised by the computation processor via software techniques. In the second generation systems the router became separate hardware unit and finally, in the third generation the three units are clearly implemented by hardware either within the same chip (for example computation processor and communication processor in the T9000) or distributed among several chips (for example, Meiko CS-2). In some third generation multicomputers the node architecture became a small shared memory system where one or more computation processors share the local memory with the communication processor (for example, Intel Paragon). In the next chapter the organisational concepts of such small size multiprocessors will be described in detail. As a summary, Table 4. gives detailed comparison of the node and network organisations of the most important distributed memory machines.

Machine	Comput. Proc.	Comm. Proc.	Vector Proc.	Comm. Switch Unit	Network Topology
nCUBE 2	custom designed	-	-	custom designed	hypercube
iPSC-2	i386	-	yes	yes	hypercube
Intel PARAGON	i860 (1 or 4)	i860	-	custom designed	2D mesh
GENESIS	i870	i870	custom designed	-	2-level crossbar
MANNA	i860	i860	-	16x16 crossbar	hierarchical crossbar
Parsytec GC/PowerPlus	PowerPC601 (2)	T805 (4)	-	C004	3D mesh
Transtech Paramid	i860	T805	-	C004	variable
IBM SP-2	POWER2	i860	-	custom designed	fat tree
Meiko CS-2	SPARC	custom designed	Fujitsu uVP (2)	custom designed	fat tree
Parsys SN9800	T9000	T9000	-	C104	hierarchical switch
J-Machine	custom designed (MDP)	MDP	-	MDP	3D mesh

Table 4. Summary of multicomputer organisations

Obviously, the technological improvements of the last 15 years are also reflected in the size parameters of multicomputers. The typical node number in the first generation systems was 64 which increased over 1000 after two generations. Similarly, the memory size of nodes has been significantly expanded from 0.5 to 32 Mbytes. The total system memory size developed from 32 Mbytes to 32 Gbytes. The lines 6-8 of Table 3. summarise these improvements of multicomputers. Organisational and technological improvements led to enormous progress in the performance of multicomputers. For example, typical system performance in scalar floating point operations has reached the 40 Gflops. Performance parameters of the three generations are summarised by the lines 9-14 of Table 3.