A layered MIMD Communication Processor Design and Classification Model

Technical Report 1-68340-44(1993)06

Eddy Olk and Henk Corporaal

Section Computer Architecture
Department of Electrical Engineering
Delft University of Technology
P.O.Box 5031
2600 AG Delft
email: E.Olk@et.tudelft.nl H.Corporaal@et.tudelft.nl

Contents

Abstract

MIMD systems consist of many processing nodes which are connected by an interconnection network. The communication between nodes through this network is supported by special hardware; besides a data processor, each node includes a communication processor. The number of design decisions for these communication processors is rather large, which is illustrated by the many different communication processor designs made and proposed in the past.

Inspired by the OSI communication layer model, this paper systematically introduces the most important design parameters for communication processors in message passing distributed memory MIMD systems. The communication layer model distinguishes physical, data link, network, and transport layers. Each layer has its own functionality (realized by using lower layers) and implementation. For every layer a set of parameters can be discriminated which characterize the layer: virtual layer support, entity representation, buffering, flow control, routing, and error handling.

Together, all design parameters span a multidimensional design space. Points in this space are clarified through classification of a number of existing communication processors. The layered classification model facilitates comparison of various communication processor designs and allows simple recognition of similarities and differences.

Chapter 1

Introduction

Currently, development of parallel computers is driven by problems demanding far more computing power than can be offered by single processor systems. Examples of such problems are scientific computations which work on very large data spaces, and artificial intelligence applications exploring very large search spaces.

Parallel computers consist of many processor nodes which must communicate and synchronize in order to work on the single application. From the possible parallel computer architectures, distributed memory MIMD systems using message passing for exchanging data between nodes seem to have the best prospects for massive parallel computing because of easy architectural extensibility and controllable exploitation of computational and communication locality. Remember that massive parallelism results in small tasks having a relative high communication to processing bandwidth ratio.

Early message passing computers, like transputers, did not offer special support for routing of messages between physical non neighboring nodes. There was no distinction between logical and physical connection. In order to make transparent communication between all nodes possible, software routing solutions had to be used (see e.g. [1]). This resulted in a substantial performance loss (e.g. very long node to node message latency).

Flexible and fast communication requires hardware communication support, i.e. each node is provided with a special communication processor (CP). Many designs for CPs are proposed and made recently [2, 3, 4, 5, 6, 7, 8, 9, 10]. and the design tradeoffs made resulted in very different processors. E.g. [8] uses messages with unrestricted length and messages contain explicit routing information. However, [3] limits the size of its packets and routing of these packets is determined by the network.

For inter-process communication in Wide and Local Area Networks there exists the OSI reference model (see e.g. [11]), which structures the communication into 7 layers, ranging from the physical link layer to the application layer. Inspired by this model, this paper presents a design and classification model for inter-processor communication in distributed memory MIMD systems.

The communication design parameters are structured into four layers, which resemble the four lower layers from the OSI reference model. Chapter 2 introduces this classification model, and its corresponding design parameters. Chapter 3 describes in more detail the design parameters common to each layer. The interface between the communication layers and the outside world, is treated in Chapter 4. In Chapter 5, different existing designs are characterized according to our classification model. The final chapter summarizes this paper and states several conclusions.

Chapter 2

CP design space

When looking at communication between nodes in Multiple Instruction Multiple Data (MIMD) systems a layer structure can be defined, similar (but not identical) to the ISO OSI reference model [11] for computer networks. The most obvious difference with the OSI model is that we look at processor networks, not computer networks. Another difference is that we attempt to describe the functionality of each layer using the same parameters and terminology. The remainder of this chapter first introduces hardware communication and then describes the functionality and design parameters of the required layers for hardware communication support.

2.1 Hardware communication

For large processor networks, providing full connectivity, i.e. directly connecting every pair of nodes in the network, is not feasible. Communication, in the form of messages, between two nodes may thus have to take place via other intermediate nodes which consequently need to forward (route) the messages sent. Performing this routing task in software has proven to be very inefficient [1]. Adding communication hardware to each node offers a flexible and fast communication mechanism, transparent to the processors; besides the (data)processor (DP) and local memory (M), each node now also has a communication processor (CP) which routes the communication data and makes the DP unaware of passing messages.

Figure 2.1 pictures involved CPs and DPs for communication of a message from a node A to

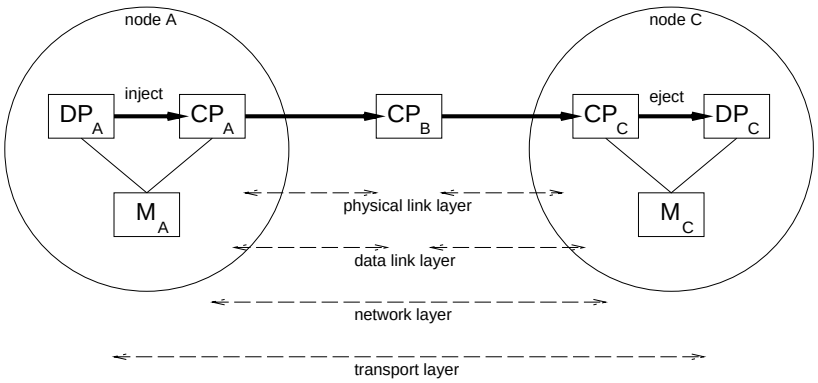

Figure 2.1: Involved CPs and DPs for message communication between node A to C via B.

a node C using an intermediate node B. The DP at node A injects the message in the network by giving it to its CP[1]. This CP then looks at the destination and decides it must be send to node B. The CP at node B receives the message, examines the destination and forwards it to node C. At node C, the CP concludes the message has arrived at its destination and delivers it to the DP, i.e. the message is ejected from the network. Clearly visible is that at node B the communication of the message takes place without DP intervention, only the CP is invoked.

2.2 Communication layers

To illuminate the communication process as introduced above and visualize the related topics, several communication layers can be defined. Relevant layers (whose scope is already loosely indicated in Figure 2.1) for communication hardware in MIMD systems are (listed bottom-up):

- **physical link layer**: provides physical communication between neighboring nodes using some transmission medium. It implements the CP-CP interface and defines the actual physical link implementation, e.g. timing discipline, and the data transfer between neighboring nodes.
- **data link layer**: provides virtual, optional error-free, links, or *communication channels*, between neighboring nodes in the communication network and regulates the flow control over these channels.
- **network layer**: responsible for routing and buffering of packets, containing message data, over the (virtual) communication network consisting of the set of all data links between neighboring nodes.
- **transport layer**: provides (virtual) point-to-point connections between non-neighboring data processors (DPs), or processes on those DPs. It takes care of ejection and injection of messages from and into the interconnection network. It also may need to divide messages in packets suitable for transportation by the network layer.

These layers are associated with the actual implementation of the communication hardware in the CP. Like the OSI reference model there can be other (higher) layers but these are (generally) realized in software and are only important from the application's point of view. They are not relevant when looking at the communication hardware.

How communication is handled by the communication layers, is pictured in Figure 2.2 which shows the layers passed when a sender (e.g. an application process) at node A communicates with a process at node C and communication takes place via an intermediate node B.

As shown in Figure 2.2 each communication layer has its own format for representing data, i.e. the *data entity*, visible to the layer above. The transport layer transports *messages* while the network layer transfers *packets* over the network. The data link layer sends *flits* [4] between neighbors using the physical layer which carries just *bits* and does the actual (physical) transportation of data between nodes. So besides the actual data of the message send by the sender, the physical layer will also carry control data associated with the transport, network and data link layers necessary for flow control of these specific layers. Note that flits are indivisible and the smallest data unit on which flow control is performed.

[1] Note that since the CP can have access to the node's memory M, the DP may just provide a pointer to the message's contents and let the CP transfer it from memory, freeing the DP for other tasks.

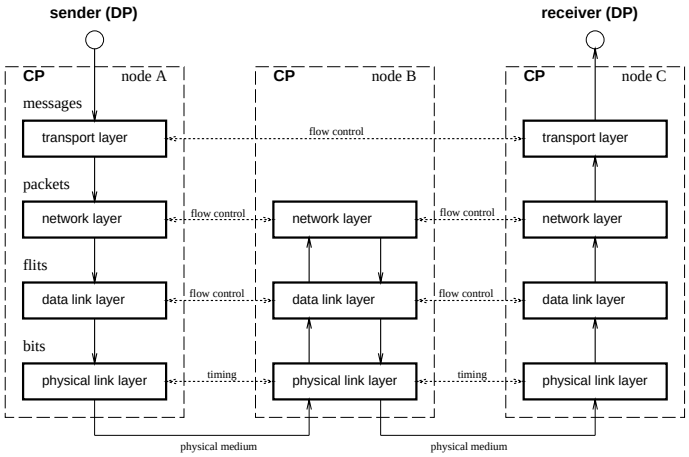

Figure 2.2: Message going from node A to node C, passing node B.

2.3 Design model parameters

Each communication layer performs a certain function and uses the lower layers, if present, to actually realize this function and offer it to the upper layer. So, for each layer we can distinguish:

- **Functionality**: the functional behavior seen by the layer directly above (superlayer interface) and the related layer's data entity representation. E.g. the network layer routes packets over the network using a certain routing strategy.
- **Implementation**: actual implementation of the layer itself. The implementation is based on the sublayer's interface and incorporates some internal representation of data. Usually, the internal representation of data consists of multiple data entities from the layer below.

Dividing the communication mechanism of CPs in various communication layers, helps the design process and also makes classification of CP designs reasonably easy. For the upper three layers, i.e. the transport, network, and data link layers, we can discriminate a set of parameters which characterize the layer, i.e. describe its functionality and implementation. We distinguish 6 parameters:

1. **Virtual layer support.** Offering of multiple, independent looking, layers to layer above.
2. **Entity representation.** Representation of the layer's data entity.
3. **Buffering.** Strategy for buffering communication data.
4. **Flow control.** Control of when data is transported within a layer.
5. **Routing.** Function which determines where data has to go to.
6. **Error handling.** Handling of errors that may occur.

These layer parameters are treated in the next chapter. The parameters associated with interfacing the communication layers of the CP to the outside world, i.e. the physical link layer (CP-CP interface) and the interface between the transport layer and the DP, (DP-CP interface) are discussed in Chapter 4.

Chapter 3

Layer parameters

As explained in the former chapter, the communication processor (CP) design model contains 4 layers, where each layer (except the physical link layer) is characterized by a set of 6 parameters. These parameters, (virtual layer support, entity representation, buffering, flow control, routing and error handling) together with their possible values, are listed in Table 3.1 and discussed in the remainder of this chapter.

3.1 Virtual layer support

To facilitate system management, improve bandwidth usage, and circumvent *deadlock* (see Section 3.5.5), a communication layer may offer virtual layer support by offering several virtual layers to the layer above. The virtual layers look like independent layers and may actually be realized using multiple (physically) independent layers or can just be time-multiplexed on the same single layer. So, the transport layer can offer *virtual connections* to the DP, the network layer can offer *virtual networks*, and the data link layer can offer *virtual data links*.

3.2 Entity representation

Figure 3.1 shows the data entity representations of the various layers. Several fields can be distinguished. Besides a field with the data to transport, a layer's data entity may contain an

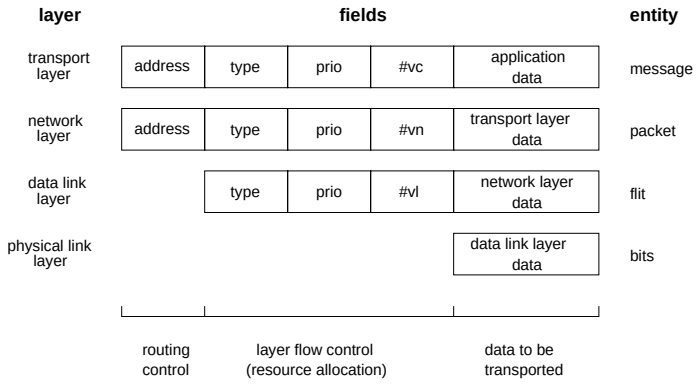

Figure 3.1: Data representation of the communication layers.

Table 3.1: Layer design parameters with possible values.

parameter	**values**
virtual layer support	yes, no
entity representation	
1. addressing mode	absolute, relative
2. address field	fixed, varisized
3. data field	fixed, varisized
4. framing	yes, no
buffering	centralized, distributed
flow control:	
1. granularity	loose, tight
2. window	0, 1 or more
3. representation	shared, separate
routing	
1. routing path control	sender, network
2. routing path	(non)deterministic
3. path collision resolvement	buffering, blocking, dropping, derouting
4. network support	restricted, arbitrary
5. deadlock/livelock resolvement	none, detection, avoidance
error handling	none, detection, correction

address field to indicate the destination. This address field is needed by the transport and network layers to control the routing of the message or packet.

Furthermore, the layers data entity can have several other fields, i.e. *type*, indicating some sort of entity type, *priority*, representing the priority of the entity, and a virtual id: #vc, #vn, #vl for virtual connection, network and link respectively. These fields can be considered by the specific (intra)layer's flow control strategy when allocating the layer's resources (e.g. buffer space, link bandwidth). For example, packets with control information may be given priority over other packets containing normal data.

Note that not all fields are obligatory; actual hardware implementations need not, and almost certainly will not, use all fields specified in a specific data entity. Also contrary to the OSI model, hardware implementations will not perform explicit data entity conversions between layers; the data entity of a layer will consist of data entities of (one of) the lower layer(s).

Besides the actual presence of certain functional entity fields, other relevant entity design parameters can be defined:

- **Addressing mode:** Destination addressing can be absolute or relative. With absolute addressing the destination address stays the same during the whole path from source to destination. With relative addressing, however, the destination address is always relative to the current node and thus this address is constantly updated when traveling over the network.

- **Address field:** The address field may have a fixed length or may be varisized. The length of a varisized address field can be specified with a length field at the start of the address field, or by using a trailer, notifying the end of the address field.

- **Data field:** Similar to the address field, the size of the data field, i.e. the amount of data it can hold, can either be fixed or varisized.

- **Framing:** The division of the entity in multiple parts, suiting the sublayer's data entity size. This is necessary when the layer's entity size/format is not identical to that of the sublayer's entity.

3.3 Buffering

The point of buffering communication data is that communication resources, i.e. link bandwidth, are limited; sometimes data can not traverse a link immediately and needs to be buffered. Buffering can be considered at different layers, i.e. for the transport, the network, or the data link layer. However, for efficient hardware usage, data is usually only buffered at a specific layer and control over the buffer is transferred from one layer to another without actually moving the buffered data.

The design aspect of buffering is associated with the fact that the data to be buffered comes from various sources and also needs to be able to go to several directions. The buffering can be *centralized*, one shared buffer for all directions, or *distributed*, where a buffer is assigned to each incoming and/or outgoing direction. Centralized buffering may use the available buffer space more efficiently compared to distributed buffering but has the disadvantage of requiring a more complex control strategy.

3.4 Flow control

Since a communication network has always limited resources, e.g. buffer capacity and bandwidth, flow control is needed to manage allocation of the resources in order to forward data as efficient as possible. Flow control is defined to regulate the moving of the specific layer's data entities within the layer; to determine **when** the layer's resources (usually buffer space) are allocated to these data entities. In order to manage the resources, flow control information needs to be exchanged between (neighboring) nodes. For each of the layers, except for the physical link layer which does not implement flow control, the following flow control topics need to be considered:

- **granularity**: The number of entities the flow control is handling. A *tight* granularity means that every entity sent between two nodes is 'acknowledged'. On the other hand, with a *loose* granularity, no 'acknowledges' are used but the receiver signals when it can not handle the incoming data any more, i.e. it issues a negative acknowledge. After a negative acknowledge, the receiver has to signal again to resume the transfer of data.

- **flow window**: A flow control window indicates the number of acknowledges that may be pending before sending new data. It represents the available (i.e. guaranteed) free buffer space on the receiving site. So when there is still buffer space available, data can be send without waiting for acknowledgements of preceding data.

- **representation**: Flow control information can be represented *separate* from the communication data received from the above layer(s), i.e. the flow control information and communication data are offered in different data entities appropriate to the lower layer, or the information and the data can *share* the data entity, i.e. they are both present in the same data entity. E.g. the transport layer can add sequence numbers to each packet, thus mixing flow control information with message data.

These flow control aspects can be illuminated by looking at the connection management of the transport layer, i.e. the *communication protocol*. The communication protocol of the transport layer defines how communication of data between end-nodes takes place, i.e. the flow control of messages. Two protocols can be distinguished, differing by the size of the flow window.

When the flow window is 0, the protocol can be considered to be *connection-based*; the sender first sends a request and on acknowledgement of the receiver a connection is established. Consequently, the granularity is tight and the flow representation is separate; the connection setup is done before the actual message transfer.

When the flow window ≥ 1, the communication protocol of the transport layer is called *connection-less*. The flow granularity can be tight or loose while the representation may be separate, shared, or both. With connection-less communication a message with communication data is just send over the network without first setting up a communication connection.

3.5 Routing

Routing is the strategy deciding **where** data must be directed to. Thus where the flow control, as described in the previous section, determines **when** resources are to be used, routing determines **which** resources must be used. Routing can take place at three levels:

1. **DP:** The DP must decide which (virtual) connection to use for communicating with a certain node. So there is a routing relation $R_{dp} : dest \times process \rightarrow (virtual)\, connection$.

2. **Transport layer:** The transport layer needs to decide which (virtual) network has to be used for messages destined for a certain node and using some (virtual) connection. This is the routing relation $R_{tl} : dest \times connection \rightarrow (virtual)\, network$.

3. **Network layer:** The network layer has to direct data to (virtual) links given its destination and (a) the (virtual) network it is using, i.e. the packet is just injected in the network layer, or (b) the (virtual) link it is coming from, i.e. the packet was already in the network layer and needs to be forwarded to another node, or delivered at the current node. So here the routing relation consists of two parts:

$$R_{nl} : \begin{cases} a): & (virtual)\, network \times dest & \rightarrow (virtual)\, link \\ b): & (virtual)\, link \times dest & \rightarrow (virtual)\, link \end{cases}$$

From these three levels, routing of the network layer is the most interesting and the remainder of this section will further elaborate on the routing strategy aspects of the network layer.

For the network layer a routing algorithm defines who establishes the routing path, i.e. *routing path control*, and what *routing path* packets can take, i.e. deterministic or nondeterministic. Table 3.2 shows some examples of routing algorithms for (non)deterministic path and sender/network path control combinations.

The routing strategy also defines how routing path collisions, i.e. packets competing for the same data link, are resolved, which networks are supported and how deadlock/livelock issues are handled. Hardware routing support is the most essential characteristic of second generation message passing MIMD systems.

Table 3.2: Examples of routing algorithms.

path	path control	
	sender	network
deterministic	street sign routing	E-cube
nondeterministic	random node routing	class climbing, route always, bounded box routing

3.5.1 Routing path control

The routing path can either be established by the **sender**, i.e. the transport layer or DP process, or the CP, i.e. the **network** layer. When the sender specifies the routing path, the path is embedded in the packet and the routing path is known when the packet is sent. The path can be defined at compile-time, i.e. the path is embedded in the source code, or at run-time when the path is established during the execution of the program. When the route of the path is settled in run-time, adaptive routing is possible. An example of a routing strategy with a sender controlled routing path is *streetsign* routing in the iWarp [8]. With streetsign routing the routing information of a message is composed of a sequence of streetsigns where a streetsign consists of two components: the *streetname* (e.g. Jones), identifying a specific node, and the associated *action* (e.g. turn left, or stop). So a routing path may be defined with: "go to Jones, turn left, go to Smith, and stop".

If the routing path is resolved by the network layer, the routing algorithm selects one or more nodes, which are valid as a next node in the path to the destination. The actual forwarding of a packet to such a node may depend on 'external' factors such as congestion or failing nodes.

An example of this strategy, called *class climbing* is implemented in DOOM [5]. With class climbing, a packet belongs to a specific prioritized class and may only use resources belonging to this, or a lower, class. As each packet is traveling towards its destination, it is tied to higher classes and consequently the amount of resources it can use increase, thus avoiding deadlock (see Section 3.5.5).

3.5.2 Routing path

The routing path a packet takes may either be *deterministic* or *nondeterministic*. Deterministic routing means that the path of a packet for a given source and destination node is strictly defined and will always be the same. At each intermediate node always a fixed output link will be selected for a given destination. A deterministic routing algorithm, is relatively simple but it can not react to congestion in the network or the (temporary) failure of a node.

An example of a deterministic routing algorithm is *E-cube* routing. With E-cube routing, data is always routed over the dimensions of the network in a fixed order, e.g. in a mesh messages are always routed north/south before they are routed in the east/west direction.

With congestion, performance of the network degrades because too many packets want to travel over one or more data links, i.e. the links are fully utilized. However, such hot spots can be circumvented by using (random) intermediate node routing (e.g. implemented in [12]) where a packet is first sent (using a fixed path) to a, software determined intermediate node, before it is routed to its real destination node. So intermediate node routing can be considered to be hybrid, since both deterministic and nondeterministic routing is possible.

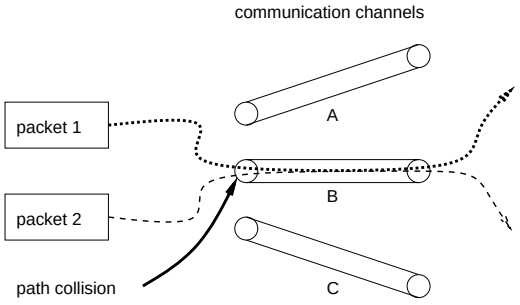

Figure 3.2: Routing path collision of two packets.

Nondeterministic, unlike deterministic, routing algorithms can react to certain (external) conditions, such as congestion, and thus the path of a traveling packet is not always the same for a given source and destination. Nondeterministic routing is, for example, possible using *bounded box* routing [9] where data can be routed freely in a certain bounded box defined by the combination of source and destination node. However, nondeterministic routing has several disadvantages:

1. Requires more complex hardware, for instance for table lookup of possible paths and assignment of paths to packets.

2. Packets arrive out of order. This is especially cumbersome in case of *store & forward* routing where packets are first fully stored at each node before they are forwarded to the next node. The transport layer has to put incoming packets in the correct order for message reassembly (introducing often intolerable overhead and memory usage).

3. Deadlock and livelock problems (see Section 3.5.5).

3.5.3 Path collision resolvement

When multiple packets are to be routed, according to the routing algorithm, over the same (virtual) data link, or communication channel, a conflict exists since only one packet can be send over the channel at the same time. Figure 3.2 pictures this and shows a situation where packets 1 and 2 both are to be routed over the single channel B. Dally [13] considers four possible ways of resolving such collisions between routing paths of two (or more) packets:

1. **Buffering**: one packet is granted the channel while the other is stored in a buffer and send as soon as the first packet releases the channel. This is also known as *virtual-cut-through* [14].

2. **Blocking**: one packet is send over the channel, the other simply blocks and is send when the channel becomes free.

3. **Dropping**: one packet gets the channel, the other packet is removed from the network. This solution is implemented in the BBN butterfly but is obviously not preferable since it wastes a large amount of resources, e.g. imagine the removal of a packet that has traveled a long way and is almost at its destination.

4. **Derouting**: one packet is routed over the channel, the other packet is routed in another direction. E.g. in Figure 3.2 the path collision may be resolved by derouting packet 2 over channel C. Two cases can be considered for derouting:

   (a) The derouted packet gets further away from its destination, i.e. it is *misrouted*. Misrouting, like dropping, wastes communication resources and also has problems to assure livelock freedom, i.e. the guarantee that a packet will arrive at its destination in finite time. Derouting allows *route always* where required buffer capacity is minimized by always keep packets 'moving' around, even if this takes them further from their destination.

   (b) Routing strategies are possible where a packet may have several optional routing paths, all destined to its destination. So then a collision is resolved (if possible) by simply routing a packet in one of its available (non-clashing) output directions.

Almost all recent CP designs implement a hybrid method of buffering and blocking.

3.5.4 Network support

Simplified implementation and performance optimalization of the network layer may place restrictions on the supported *degree* of network nodes and/or the network *topology*.

Degree The degree of a node is defined as the number of links the node has, or the number of neighboring nodes that are directly connected. Usually, CP designs only allow a fixed degree, but support for a variable degree may be present, for instance by the cascading of multiple CPs [4].

Although a CP may support only a fixed degree, this does not mean the possible topologies of the interconnection network are restricted to networks where all nodes have the same degree, like a ring. Topologies with nodes of different degree, e.g. a binary tree or a mesh where respectively the leave nodes and border nodes have a lower degree, can be realized by not connecting every link.

Topology Besides the supported degree of a CP design, feasible topologies are determined by the routing support supplied by the design. Figure 3.3 shows some possible network topologies. The routing scheme may allow arbitrary topologies, but can also be restricted to a certain class of topologies. So routing may be optimized towards a specific class of topologies, e.g. hypercubes, or flexible routing may be provided, for example using alterable routing tables.

3.5.5 Deadlock and Livelock

Deadlock is part of the livelock or starvation problem. A livelock situation exists when the time a process has to wait before gaining access to a resource is not bounded by a finite time. In a network a deadlock situation occurs when there is a cyclic dependency of blocked communication links, i.e. the resources. A packet is livelocked in a network when it can not be guaranteed to arrive at its destination in finite time, although it may move around over the network all the time, so it starves. Obviously, since livelock freedom is a stronger constraint than deadlock freedom, livelock avoidance always requires deadlock freedom.

In Figure 3.4 a deadlock situation of four nodes is shown, comparable to the 4-dining philosophers problem. Each node wants to forward a packet to the packet's destination node via

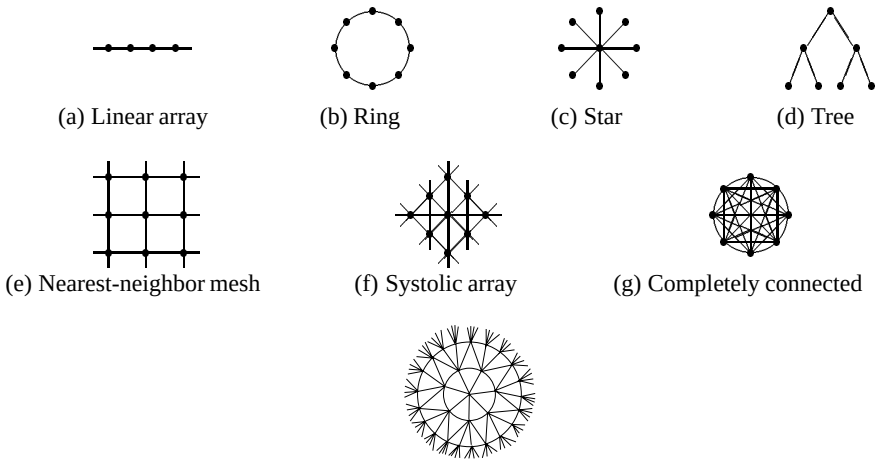

Figure 3.3: Possible interconnection network topologies.

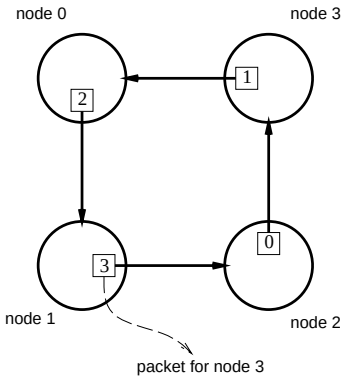

Figure 3.4: Cyclic dependency of communication links.

an intermediate node. E.g. node 1 wants to send a packet for node 3 via node 2. However, the related buffers of these intermediate nodes are occupied by the packets which must be forwarded and thus no (intermediate) node is able to accept the specific message. E.g. node 2 can not accept the message from node 1 (destined for node 3) since its buffer is occupied by a packet destined for node 0. This packet can not be forwarded since node 3 has no free buffer space due to a packet for node 1. So a deadlock situation exists which can only be broken by an external action.

CP designs can resolve deadlock either by simply avoiding it, e.g. using virtual links, or by detecting it and then take appropriate steps. Livelock freedom can, if deadlock is prevented, generally be established by *fair resource sharing*, e.g. packets never have to wait forever for the next buffer.

Deadlock resolvement It is clear that deadlock prevention is extremely important and deadlock occurrence can not be allowed in a network, unless the deadlock probability is so low that it is negligible compared to hardware faults. When deadlock is allowed however, a deadlock situation can be detected by examination of the process-resource interactions for the presence of cyclic wait [15]. So a CP design can allow deadlock, and possibly detect it, but generally it is better to avoid the occurrence of deadlock situations.

There are several ways to prevent deadlock which essentially all map acyclic graphs on the topology in such a way that any packet may reach its destination by traveling along one or more graphs. To each graph correspond resources (networks or buffers) which may only be consumed by a packet in a given order. Possible ways of preventing deadlock are:

- **restrictive routing:** choosing a deadlock free deterministic routing obviously avoids deadlock situations. Adaptive routing algorithm can also be used by analyzing directions packets can turn to and prohibiting just enough turns to break all possible cycles so packets never have to wait for each other in a cycle [16]. E.g. in Figure 3.4 one of the possible turns, like the right-to-up turn, could be disallowed and thus avoiding deadlock.

- **class climbing:** packets and buffers in the network are assigned a priority or class. A packet starts with a low class, but this class can grow in the path to the destination. A buffer only buffers packets with a class equal or higher than that of the buffer itself. So for a packet from a high class more buffer storage is available than for a packet from a low class. By increasing the class of a packet more buffer storage space becomes available and routing is easy. It is essential however to guarantee that in going from source to destination the class of a packet can not climb beyond the number of available classes. This prevents arbitrary paths containing cycles. In [17] and [18] it is proven that the construction of a loop-free directed 'buffer communication dependence graph' suffices to prevent deadlock. A comparable, but more complex, class climbing mechanism is used in the DOOM multi-processor [5].

- **multiple acyclic networks:** the network is split in a set of independent acyclic subnetworks. Each subnetwork can transport packets, has its own queues and can be proven to be deadlock free. Because each subnetwork is deadlock free the whole network must also be deadlock free. An example of how a mesh can be split in four acyclic networks is shown in Figure 3.5. This is an expensive solution when realized physically, especially with high dimensional networks, i.e. where nodes have a high degree, but using virtual data links of the data link layer makes this approach attractive.

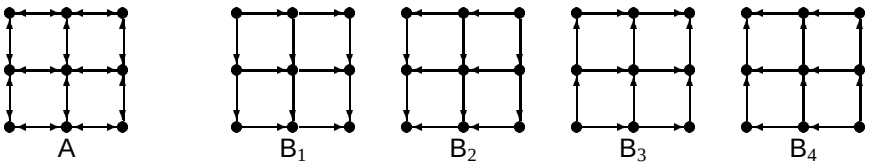

Figure 3.5: Division of cyclic mesh network A in four acyclic subnetworks B_1 to B_4.

- **route always:** a rather different approach compared to the ways of deadlock prevention listed above which does not uses acyclic graph mapping. The routing algorithm routes the packets always, i.e. packets are routed always in some direction also when the packet moves further from its destination. This routing strategy is know as the 'hot potato' principle [19]. This algorithm assumes that it is better to move packets around in the network rather than to keep them stored in a buffer. Deadlock is prevented because packets are always 'moving', i.e. only one resource is claimed at a time, and thus buffer resources are allocated and freed frequently. However, special care must be taken to assure that packets do not stay in the network forever and a livelock situation arises. [10] shows the implementation of a CP with the route always scheme.

Livelock resolvement Since livelock freedom is a stronger constraint than deadlock freedom, livelock avoidance always requires deadlock freedom. Livelock freedom in a network can, if deadlock is prevented, be established by avoiding packets to wait forever for the next buffer. This can be done by limiting the maximum time a packet may occupy a buffer. This assures frequently freeing of buffer space. Fair arbitration between multiple packets for the same buffer space can be realized by using a FIFO buffer structure.

This is sufficient for routing algorithms where a packet always moves towards its destination. However, when packets may be routed further away from their destination, e.g. when route always is used, additional requirements are needed. The network must then guarantee packet delivery and packet injection [20]. It turns out that a minimal amount of buffer space (measured in number of packets) is necessary, to assure that at least one packet is routed in the right direction.

3.6 Error handling

Although probability of hardware faults is rather low, occasional errors must be expected in systems using large numbers (> 1000) of computing and communicating nodes. Therefore, error handling must be considered when designing communication hardware. For each of the layers three cases can be distinguished for the layer's error handling support:

1. **None.** No error handling has the advantage of not requiring additional hardware and may be a good solution when the higher (software) communication layers or the application itself can provide sufficient support for recovering from (occasional) communication errors. However, although the specific layer does not handle errors, it may still require some means of letting higher layers reset its state in order to recover from the errors.

2. **Detection.** A layer can detect errors by (a) receiving an error signal from the layer below, or (b) by adding additional error detecting bits, e.g. for a Cyclic Redundancy Check (CRC),

to the data it sends via the lower layer. When occurring errors are detected the layer can recover from them by issuing a request for a retransmission of the specific data or simply report the error to the communication layer above.

3. **Correction.** Instead of requesting retransmission of faulty data, the layer may add error correcting information, e.g. using Hamming codes, to data send via the lower layer(s) and simply correct possible errors at the receiving side. Of course this does not recover from all errors, e.g. packets that got somehow lost completely.

Chapter 4

CP interface

This chapter discusses the interface of the CP to the outside world:

1. The interface between the DP and the transport layer. Data transfer between the transport layer and the layer above, e.g. the application layer. Since this is actually the transfer of data between the DP and the CP, this is identified with *DP-CP interface*.

2. The physical link layer which determines data transfer between neighboring CPs using some sort of transmission medium. We name this *CP-CP interface*.

Table 4.1 shows the CP interface parameters with possible values which are discussed below.

Table 4.1: CP interface parameters and possible values.

parameter	**values**
DP-CP interface	
message initialization	DP, CP
message transfer	DP, CP
CP-CP interface	
pin type	uni/bi-directional
timing	ALT, SGT, SLT
data word size	n

4.1 DP-CP interface

The DP-CP data transfer determines how messages are exchanged between the DP (e.g. a process) and the CP (transport layer) and consists of the initialization of a message transfer (either message *injection* or *delivery*), and the actual data transfer.

Message initialization Initialization sets up the sending or receiving of a message, i.e. the *injection* in the network or the *delivery* to the node, and can be done by the DP or the CP. The initialization for sending a message is usually done by the DP, but a smart CP may (to avoid additional DP overhead) initiate sending of messages on its own without consulting the DP, e.g. to send a certain memory page on request of another node. The initialization for the receipt of

a message is usually done by the CP, e.g. by interrupting the DP, since the DP can not know precisely when a new message is going to arrive.

Message transfer Like the message initialization, the message transfer can also be performed by either the DP or the CP. When the CP is smart, it can for example use DMA to transfer the data to/from the memory of the DP. This leaves the DP free to execute the application's computational code which is useful when communication and calculations are loose, i.e. not closely coupled.

4.2 CP-CP interface

The CP-CP interface defines the link implementation between two neighboring CPs. The most relevant parameters here are the pin implementation of the link, the timing discipline and the data word size, the number of data bits that can be transported at once (this last parameter is not discussed below).

Pin implementation Usually physical links are implemented using unidirectional pins, a pin is capable of handling either incoming or outgoing data. It is however possible to use bidirectional pins where the data transfer direction of the pins can be reversed (half duplex), or even data transferring can take place in both directions at the same time (full duplex) [21].

Timing Signals between nodes must be properly interfaced in order to guarantee reliable transmission of information. The signal interfacing can be classified in three categories[1] (see [22]):

1. **SGT**: Synchronous Globally Timed. With SGT communication, valid data is always indicated with reference to a global clock signal, identical to all communication nodes of the MIMD system.

2. **SLT**: Synchronous Locally Timed. With SLT interfacing, all systems are synchronous and synchronize to each other when data is exchanged. Valid data is indicated by control signals which are sampled on clock edges.

3. **ALT**: Asynchronous Locally Timed. ALT communication only requires control signals to indicate valid data. The difference with SLT is that these control signals are also used to actually transfer the data, e.g. the latching.

[1] The logically fourth category, i.e. Asynchronous Globally Timed (AGT) is non-existent.

Chapter 5

Classification

The previous chapters defined a communication layer model which spans the design space of CPs. Besides structuring the CP design process, the layer model can also be used to describe and characterize communication processor designs. Such a classification of CP designs makes comparison easier and highlights differences in design parameters.

Table 5.1 shows the classification of several CP designs (empty fields in the table indicate parameters not present or not applicable for the design):

- **Connection Machine** The CM-1 Connection Machine from Hillis [7] was targeted as a "symbol cruncher" for artificial intelligence problems. It consists of 65,536 1-bit processors which are connected in a 256x256 grid. Additionally, groups, or *clumps*, of 16 processors are also interconnected by a packet-switched 12-dimensional hypercube network for routing messages using a route always routing strategy. Within a clump only one processor at a time can communicate via the hypercube and therefore the processors are linked in a daisy chain fashion. For the classification we only consider the hypercube network because the grid has no routing capabilities.

- **Torus Routing Chip** The torus routing chip [4] developed by Dally provides, using two virtual channels per link and E-cube routing, deadlock-free packet communications in a k-ary n-cube (torus) networks with up to k = 256 nodes in each dimension. The ≈10,000-transistor TRC chip is implemented in 3μ CMOS technology and packaged in an 84-lead pin-grid array. The chip is intended for n = 2-dimensional networks but the chips can be cascaded, to handle arbitrary n-dimensional networks. The TRC is entirely self-timed to avoid the difficult distribution of a global clock over a large array of processors. The latency of messages traversing more than one link is reduced by using *wormhole switching* rather than packet switching. Messages are varisized and consist of several *flits*.

- **Bounding Box Router (BBR)**: A 2-dimensional bounding box router is designed and implemented by [9]. This packet switching router is capable of routing 48-bits messages over a 64x64 grid network. A prototype custom VLSI communication processor was implemented using a 2.5μm CMOS process technology on a silicon area of 7x7 mm. The chip uses 128 pins for data, control, power, clock and test signals.

- **DOOM**: The communication architecture described here is part of the architecture of DOOM, a Decentralized Object-Oriented Machine, which is designed in the Computer Science department of Philips Research Laboratories. The Communication Processor (CP)

Table 5.1: Layer design parameters of several CP designs.

	communication processor design						
parameter	CM	TRC	BBR	DOOM	iWarp	T9000/C104	MOVECP
DP-CP interface							
message initialization							
injection	DP	DP	DP	DP	DP	DP	DP
delivery	DP	DP	DP	DP	DP/CP	DP/CP	DP/CP
message transfer	DP	DP	DP	DP	DP/CP	DP/CP	DP
transport layer							
virtual layer support						256	
entity representation							
addressing mode	absolute	relative	relative	absolute	absolute	absolute	relative
address field	16 bits	varisized	12 bits	10 bits	varisized	varisized	varisized
data field	32 bits	varisized	36 bits	242 bits	varisized	varisized	varisized
framing						yes	
buffering					both		
flow control							
granularity					loose	tight	
window					0	1	
representation							
error handling							
network layer							
virtual layer support		2					yes
entity representation							
addressing mode	relative	relative	relative	absolute	absolute	absolute	relative
address field	12 bits	varisized	12 bits	10 bits	varisized	varisized	varisized
data field	52 bits	varisized	36 bits	242 bits	varisized	0-32 bytes	varisized
framing		yes			yes		yes
buffering	centralized			centralized			
flow control							
granularity			tight	tight			
window			2	0			
representation			separate	separate			
routing							
path control	netw.	netw.	netw.	netw.	sender	netw.	sender/netw.
path	nondet.	det.	nondet.	nondet.	det.	nondet.	det.
collision resolvement	deroute	buf-block	buf-block deroute	deroute	buf/block buf/block	buf/block deroute	buf/block
network support							
degree	12	4	4	4	4	arbitrary	arbitrary
topology	hypercube	k-ary n-cube	64×64 grid	unrestr.	unrestr.	unrestr.	mesh
deadlock resolvement	route always	acyclic		class climb.		restr.	acyclic
livelock resolvement		fair sharing		fair sharing		fair sharing	fair sharing
error handling							
data link layer							
virtual layer support					4		
entity representation							
data field	64 bits	8 bits	48 bits	256 bits	32 bits	8 bits	32 bits[a]
buffering		distr.			distr.	distr.	distr.
flow control							
granularity		tight			tight	tight	loose
window		1			2	≥ 8	1
representation		separate	separate	shared	shared	shared	separate
error handling					detection	detection	
CP-CP interface							
pin type	unidir.	unidir.	unidir.	unidir.	unidir.	unidir.	unidir.
timing	SGT	ALT	SGT	ALT	ALT	ALT	SGT
data word size	1	8	12	1	8	1	variable

[a] Due to a compression scheme, the actual size can be less.

[5] developed for DOOM uses a store & forward (packet switched) flow control with packets of 256 bits. A variant of class climbing is used to prevent deadlock and starvation in the network. A packet consists of 10 address bits, 4 class bits (used for class climbing) and 242 data bits. The network may contain up to $2^{10} = 1024$ nodes and is not constrained to certain topologies since the routing algorithm is suitable for all network topologies.

- **iWarp** The iWarp [8], product of a joint effort between Carnegie Mellon University and Intel Corporation, is a single chip processor for high speed signal, image and scientific computing, consisting of approximately 600,000 transistors. The iWarp contains a computation agent and a communication agent. Only memory must be added to form a complete system building block. The communication agent implements the streetsign routing scheme.

- **T9000/C104** The IMS C104 packet routing switch is part of the INMOS IMS T9000 transputer family product range [3] and shows that INMOS for these new transputers has acknowledged the need for hardware support for communication between non-neighbor transputer nodes. Interesting is that the router is implemented as a separate chip which communicates with the T9000 using "normal" links. This allowed INMOS to optimize both C104 and T9000 for their specific purpose. E.g. as will be clear from Figure 2.2, the C104 does not need support for the transport layer. Since the T9000 only implements the transport and physical layers and the C104 has the network, data link and physical link layers, they are considered as a single design in the presented classification.

- **MOVE CP** This CP [23] was designed at our laboratory as a parametrizable VLSI standard cell for our SCARCE architecture framework and is currently further developed within the MOVE architecture framework, the successor to SCARCE. The MOVE CP VLSI cell uses E-cube routing and has parameters for (among others) the node degree, number of virtual networks, link implementation, and internal buffer sizes. The ASA silicon compiler of Sagantec is used for actual chip layout generation.

Without getting too deep into the specific CP design implementations, some remarks can be made when looking at the shown classification (and considering other CP designs):

- Apparently error handling is not (yet) considered of significant importance since no error correction is implemented and just a few designs can detect errors. This most likely has to do with the fact that errors (normally) do not occur very often and can better be handled by (higher level) software than implemented in (expensive) hardware.

- Recent designs all implement the virtual layer concept for one or more layers in order to obtain more flexible routing capabilities, avoid deadlock, and yield a better resource usage, especially of link bandwidth.

- Designs implementing the virtual layer concept look similar on the surface view but the layered CP classification model reveals that they in fact implement the time-multiplexing of data transports each at different layers. For instance, iWarp implements virtual links at the data link layer, the T9000/C104 offers virtual channels at the transport layer, and the MOVE CP has both virtual.data links and virtual networks.

- Despite the advantages of bidirectional physical data transfer, no design implemented it yet. This will most likely change in the future when chips hold more and more transistors while physically the I/O bandwidth stays the same due to package pin limitations.

- Most designs allow connection-based communication which is advantageous for applications with systolic-like communication.

Chapter 6

Conclusions

The most important design parameters for communication processors (CPs) in MIMD systems were explored using a layered model inspired by the OSI reference model. Although not perfect, the model makes it possible to clearly distinguish various functional levels in a CP design; attaching a specific named data entity to each layer clarifies this even further.

The layered communication model greatly covers the CP design space, as was illustrated by classification of a number of CPs. Furthermore, the model can be used to compare existing designs and pin-point significant differences.

This layered design model for MIMD CPs is found to be useful in our current research which tries to obtain a thorough knowledge of the communication architecture versus application performance relation. The presented model facilitates relating new CP designs to existing designs and gives a wider insight in the architecture design process of CPs.

Bibliography

[1] Herman Roebbers and Marnix Vlot. A communication processor on the transputer. In *Proceedings 10th occam User Group Technical Meeting*, pages 143–151, April 1989.

[2] C.L. Seitz. The Cosmic Cube. *Communications of the ACM*, 28/1(22), 1985.

[3] INMOS Limited. The T9000 transputer products overview manual. INMOS Databook series, April 1991.

[4] William J. Dally. *A VLSI Architecture for concurrent Data Structures*. Kluwer Academic Publishers, 1987.

[5] J. K. Annot and R. A. H. van Twist. A Novel Deadlock free and Starvation Free packet switching communication processor. In *Parallel Architectures and Languages Europe*, page 68, 1987.

[6] Daniel A. Reed and Dirk C Gronwald. The performance of Multicomputer Interconnection Networks. *IEEE computer*, June 1987.

[7] Daniel Hillis. *The Connection Machine*. MIT Press, 1985.

[8] e.a. Borkar S. iWarp: An Integrated Solution to High-Speed Parallel Computing. In *Proceedings of Supercomputing '88*, 1988.

[9] W.G.P. Mooij. *Packet Switched Communication Networks for Multi-Processor Systems*. PhD thesis, University of Amsterdam, 1989.

[10] C. Germain et al. A new communication design for massively parallel message passing computers. In *IFIP Working Congerence on Decentralized Systems*, Lyon, December 1989.

[11] Andrew S. Tanenbaum. *Computer networks*. Prentice-Hall, 1989.

[12] INMOS Limited. The T9000 Transputer Products Overview Manual. INMOS Databook series, April 1991.

[13] William Dally. *Network and Processor Architecture for Message-Driven Computers*, chapter 3, pages 140–222. Morgan Kaufmann, 1990. winter school Banff Alberta.

[14] Parviz Kermani and Leonard Kleinrock. Virtual Cut-Through: A New Computer Communication Switching Technique. *Computer Networks*, 3:267–286, 1979.

[15] Mukesh Singhal. Deadlock Detection in Distributed Systems. *IEEE computer*, 22(11):37–48, November 1989.

[16] Christopher J. Glass and Lionel M. Ni. The Turn Model for Adaptive Routing. In *ISCA-92*, pages 278–287, Australia, May 1992.

[17] I.S. Gopal. Prevention of Store-and-Forward Deadlock in Computer Networks. *IEEE transactions on Communications*, COM-33(12), December 1985.

[18] Klaus D. Gunther. Prevention of Deadlocks in packet-switched data transport systems. *IEEE transactions on Communications*, 29(4):512–524, April 1981.

[19] G.P. McKeown and V.J. Rayward-Smith. Experience with CST: Programming and Implementation. Technical report, School of Computing Studies & Acountacy, University of East Anglia, UK, Mathematical Algorithms Group, October 1983.

[20] John Y. Ngai. *A Framework for Adaptive Routing in Multicomputer Networks*. PhD thesis, California Institute of Technology, 1989. CS-TR-89-09.

[21] K. Lam, L.R. Dennison, and W.J. Dally. Simultaneous Bidirectional Signalling for IC Systems. In *ICCD 90*, pages 430–433, 1990.

[22] Stephen A. Ward and Robert H. Halstead Jr. *Computation Structures*. The MIT Press, Cambrdige, Massachusetts, 1990.

[23] H. Corporaal and J.G.E. Olk. A Scalable VLSI MIMD Routing Cell. In *DMCC-6 conference proceedings*, Portland, April 1991.