The OSI Model applied to MIMD Communication Processor Design

Eddy Olk and Henk Corporaal

Department of Electrical Engineering, Delft University of Technology,
P.O.Box 5031, 2600 GA Delft, The Netherlands

MIMD systems consist of many processing nodes that are connected by an interconnection network. The communication between nodes through this network is supported by special hardware; besides a data processor (DP) and some local memory, each node includes a communication processor (CP). 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. Together, these parameters span a multidimensional design space. Points in this space are clarified through classification of a number of existing communication processors.

1. CP design model

Figure 1 pictures involved communication processors (CPs) and data processors (DPs) for communication of a message from a node A to 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. This CP then looks at the destination address and decides it must be sent 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.

To illuminate the communication process as introduced above and visualise the related topics, several communication layers can be defined, as already shown in Figure 1. Relevant layers for communication hardware in MIMD systems are (listed bottom-up):

- **physical link layer**: provides physical communication between neighbouring nodes using some transmission medium. It defines the actual physical link implementation (e.g., timing discipline) and the transfer of data *bits* between neighbouring nodes.
- **data link layer**: provides, optional error-free, (virtual) data links, or *communication channels*, between neighbouring nodes in the communication network and regulates the flow control over these channels. The data unit transferred over a data link is called a *flit*.
- **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 neighbouring nodes.

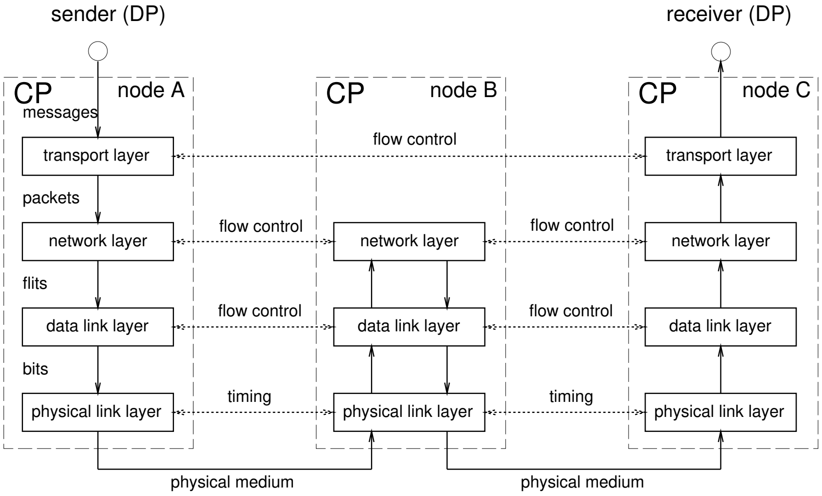

Figure 1. Message going from node A to node C, passing node B.

- **transport layer**: provides (virtual) point-to-point connections between non-neighbouring 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. Like the OSI reference model there can be other (higher) layers but these are realised in software and are only important from the application's point of view. They are not relevant when looking at the communication hardware.

Each communication layer performs a certain function and uses the lower layers, if present, to actually realise this function and offer it to the upper layer. So, each layer has its own functionality, and its corresponding data entity. For example, the network layer routes packets over the network using a certain routing strategy.

Dividing the communication mechanism of communication processors in various communication layers, helps the design process and also makes classification of communication processor designs reasonably easy. However, the layered CP design model is not meant to impose a strictly layered implementation as this would result in inefficient implementations, e.g. too much overhead would be involved in explicit conversion of data entities between layers. The purpose of the model is to present a clear view of different functional levels associated with communication.

2. Layer parameters

For each layer we can discriminate a set of parameters that characterise the layer, i.e. describe its functionality. For the upper three layers (transport, network, and data link layer) recognizable parameters (Table 1 lists these, together with some parameter values) are: virtual layer support, entity representation, buffering, flow control, routing[1], and error handling. The physical link layer is rather small and is mainly composed of a parameter describing the employed timing protocol. The remainder of this section briefly discusses the (common) layer parameters for the

[1]Not applicable for the data link layer.

Table 1
Layer design parameters with possible values.

parameter	values
virtual layer support	yes, no
entity representation	address, type, prio, virtual id, data
buffering	centralised, distributed
flow control:	
1. granularity	loose, tight
2. window	0, 1 or more
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

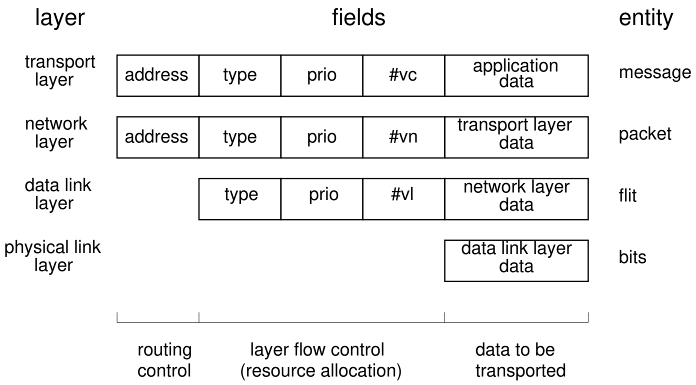

Figure 2. Data representation of the communication layers.

transport, network, and data link layer. For a more extensive discussion of the communication layer parameters we refer to [1].

2.1. Virtual layer support

To facilitate system management, improve bandwidth usage, circumvent *deadlock*, or routing flexibility, 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 realised 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*.

2.2. Entity representation

Figure 2 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 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. Addressing can be *relative* or *absolute*. The sizes of the data and address fields can either be *fixed* or *varisized*.

Furthermore, the layers data entity may have several other control fields, such as type, priority, virtual connection (#vc) / network (#vn) / layer (#vl), which can be considered by the specific layer's flow control strategy when allocating the layer's resources (e.g., buffer space, link bandwidth). 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.

2.3. 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 the transport, the network, and 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 *centralised*, one shared buffer for all directions, or *distributed*, where a buffer is assigned to each incoming and/or outgoing direction. Centralised buffering may use the available buffer space more efficiently compared to distributed buffering but has the disadvantage of requiring multiple access ports and a more complex control strategy.

2.4. 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 movement 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 (neighbouring) 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; 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 sent without waiting for acknowledgements of preceding data.

2.5. 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, the packet is just injected in the network layer, or (b) the (virtual) link it is coming from, 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:

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

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

For the network layer a routing algorithm defines the following routing parameters:

1. **Routing path**: what path do packets take; a *deterministic* or *nondeterministic* path.
2. **Routing path control**: who establishes the routing path. 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*.
3. **Path collision resolvement**: resolvement of packets competing for the same data link. Dally [2] considers four possible ways of resolving such collisions between routing paths of two (or more) packets: (1) *buffering* where 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, (2) *blocking* where one packet is sent over the channel, the other simply blocks and is sent when the channel becomes free, (3) *dropping* where one packet gets the channel and the other packet is removed from the network, (4) *derouting* where one packet is routed over the channel, the other packet is routed in another direction. Almost all recent CP designs implement a hybrid method of buffering and blocking.
4. **Network support**: Simplified implementation and performance optimalisation of the network layer may place restrictions on the supported *degree* of network nodes and/or the network *topology*.
5. **Deadlock/livelock resolvement**: Deadlock is part of the livelock or starvation problem. 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. In a network a deadlock situation occurs when there is a cyclic dependency of blocked communication links (i.e., the resources). Obviously, since livelock freedom is a stronger constraint than deadlock freedom, livelock avoidance always requires deadlock freedom.

   Possible ways of preventing deadlock are (see [1]): restrictive routing, class climbing, acyclic networks, or route always. Given deadlock freedom, livelock is easily avoided by fair sharing of the resources.

2.6. 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) 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 to data sent 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.

3. Classification

The previous sections 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 characterise communication processor designs. Such a classification of CP designs makes comparison easier and highlights differences in design parameters.

Table 2 shows the classification of several CP designs (empty fields in the table indicate parameters not present or not applicable for the design): the Connection Machine [3], the Torus Routing Chip (TRC) [4], the communication processor for the Decentralised Object-Oriented Machine [5], the iWarp [6], the Inmos T9000/C104 [7], and the scalable VLSI MIMD routing cell designed for MOVE [8].

Without getting too deep into the specific CP design implementations, some remarks can be made when looking at the various layer parameter values of the shown classification:

- 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 TRC and MOVE CP have virtual networks, and T9000/C104 offers virtual channels at the transport layer.
- All recent designs support varisized messages and packets, only CM-1/2 and DOOM use fixed sized packets. The CM-1/2 is the only design that uses a different addressing mode at the transport layer than at the network layer.
- Buffering is done either using distributed buffers at the data link layer or using a centralised buffer at the network layer. In addition, at the transport layer iWarp can also buffer data

Table 2
Layer design parameters of several CP designs.

	communication processor design					
parameter	CM-1/2	TRC	DOOM	iWarp	T9000/C104	MOVE
transport layer						
virtual connection support					256	
entity representation						
addressing mode	absolute	relative	absolute	absolute	absolute	relative
address field	16 bits	varisized	10 bits	varisized	varisized	varisized
data field	32 bits	varisized	242 bits	varisized	varisized	varisized
buffering				centralised	centralised	
flow control						
granularity				loose	tight	
window				0	1	
network layer						
virtual network support		yes				yes
entity representation						
addressing mode	relative	relative	absolute	absolute	absolute	relative
address field	12 bits	varisized	10 bits	varisized	varisized	varisized
data field	52 bits	varisized	242 bits	varisized	0-32 bytes	varisized
buffering	centralised		centralised			
flow control						
granularity			tight			
window			0			
routing						
path	nondet.	det.	nondet.	det.	nondet.	det.
path control	netw.	netw.	netw.	sender	netw.	sender/netw.
collision resolvement	deroute	buf/block	buf/block	buf/block	buf/block	buf/block
network support						
degree	12	4	4	4	4/32[a]	fixed[b]
topology	hypercube	k-ary n-cube	unrestr.	unrestr.	unrestr.	mesh
deadlock resolvement	route always	acyclic	class climb.	software	restr.	acyclic
livelock resolvement		fair sharing	fair sharing	fair sharing	fair sharing	fair sharing
data link layer						
virtual link support				yes		
entity representation						
data field	64 bits	8 bits	256 bits	32 bits	8 bits	32 bits[c]
buffering		distributed		distributed	distributed	distributed
flow control						
granularity		tight		tight	tight	loose
window		1		2	≥ 8	1
error handling				detection	detection	

[a]The T9000 has a degree of 4 and the C104 has a degree of 32.
[b]An arbitrary degree can be specified at design-time and the corresponding design implementation will be generated by the VLSI framework.
[c]Due to a (simple) compression scheme, the actual size can be less.

in the node's local memory using a (transparent) technique called *spooling*. The T9000 uses its local memory to store incoming or outgoing messages.

- The CM-1/2 has the most simple flow control strategy which just assumes that the receiver can always receive a packet. Also no acknowledges are required. With DOOM the receiver must explicitly inform a possible sender that it may send a packet. The other designs perform flow control at the data link layer. The T9000/C104 also has flow control at the transport layer.
- A packet's routing path is generally deterministic and controlled by the network. With MOVE the sender can have some influence on the routing path by specifying *intermediate nodes* which must be visited by the message. For iWarp the routing path is only determined by the sender which may lead to deadlock situations. Livelock is generally avoided by fair sharing of the resources but the CM-1/2 has problems here due to its route always scheme where packets may be routed further away from their destination. DOOM, iWarp, and T9000/C104 seem to have the most universal routing capabilities which can easily adapt to different network configurations.
- Error handling is not considered important enough to offer extensive error handling support in hardware. Only iWarp and T9000/C104 have error detection at the data link layer. So with all designs errors will have to be handled by software in one of the higher layers.

4. Conclusions

The most important design parameters for CPs in message passing distributed memory MIMD systems were discussed using a communication layer model similar to the OSI model. The layer model was illuminated by classification of several CP designs.

To our knowledge this is the first time CP designs for MIMD systems are classified in such a uniform manner. This made it quite easy to compare the various designs and recognise similarities and significant differences. Furthermore, the presented model facilitates relating new CP designs to existing designs and gives a wider insight in the architecture design process of CPs.

REFERENCES

1. Eddy Olk and Henk Corporaal. A Layered MIMD Communication Processor Design and Classification Model. Technical Report 1-68340-44(1993)06, Delft University of Technology, 1993.
2. William Dally. *Network and Processor Architecture for Message-Driven Computers*, chapter 3, pages 140–222. Morgan Kaufmann, 1990. winter school Banff Alberta.
3. Daniel Hillis. *The Connection Machine*. MIT Press, 1985.
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*, 1987.
6. e.a. Borkar S. iWarp: An Integrated Solution to High-Speed Parallel Computing. In *Proceedings of Supercomputing '88*, 1988.
7. INMOS Limited. The T9000 Transputer Products Overview Manual. INMOS Databook series, April 1991.
8. H. Corporaal and J.G.E. Olk. A Scalable VLSI MIMD Routing Cell. In *DMCC-6 Conference Proceedings*, Portland, April 1991.