NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making
of photocopies or other reproductions of copyrighted material. Any copying of this
document without permission of its author may be prohibited by law.

Network-Based Multicomputers: Redefining High Performance Computing in the 1990s

H. T. Kung

January 20, 1989
CMU-CS-89-138

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Appeared in the *Proceedings of the Decennial Caltech Conference on VLSI*, edited by Charles L. Seitz, MIT Press, March 1989, pp. 49-66. (Invited Paper).

ABSTRACT

One of the most significant advances in computer systems over the past decade is parallel processing. This is a direct consequence of close interactions between system design and VLSI technology. This integrated approach has allowed insights in computations, systems and applications to have fundamental influences on the design of hardware structures, and vice versa. The cooperation will be even more crucial as we enter the next decade when parallel computers are expected to be not only powerful in performance but also easy to use. A key point here is to use the power of VLSI to implement *general* and very *high performance* networks for large-scale multicomputers. In this paper we briefly review current work at Carnegie Mellon in this area, and give a taxonomy to show the general architectural trends. It is concluded that multicomputers based on flexible, extensible and efficient network backplanes will be a major thrust in high performance computing in the 1990s.

The research was supported in part by Defense Advanced Research Projects Agency (DOD) monitored by the Space and Naval Warfare Systems Command under Contract N00039-87-C-0251, and in part by the Office of Naval Research under Contracts N00014-87-K-0385 and N00014-87-K-0533. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, the Office of Naval Research or the U.S. government.

Decennial Caltech Conference on VLSI
Pasadena, California, 20-22 March 1989

Network-Based Multicomputers: Redefining High Performance Computing in the 1990s

H. T. Kung

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Invited Presentation

One of the most significant advances in computer systems over the past decade is parallel processing. This is a direct consequence of close interactions between system design and VLSI technology. This integrated approach has allowed insights in computations, systems and applications to have fundamental influences on the design of hardware structures, and vice versa. The cooperation will be even more crucial as we enter the next decade when parallel computers are expected to be not only powerful in performance but also easy to use. A key point here is to use the power of VLSI to implement general and very high performance networks for large-scale multicomputers. In this paper we briefly review current work at Carnegie Mellon in this area, and give a taxonomy to show the general architectural trends. It is concluded that multicomputers based on flexible, extensible and efficient network backplanes will be a major thrust in high performance computing in the 1990s.

1 Introduction

As of 1989 there are about 350 supercomputers installed worldwide. However applications continue to demand more computing power. Computational problems in areas such as high-speed aircraft design and medical imaging, and research in advanced structural, electronic, and optical materials often require computers that are at least three orders of magnitude faster than the fastest computers presently available.

Although computing demands are increasing rapidly, the performance of conventional, sequential computers is approaching the point of diminishing return. High-performance sequential computers today are al-

University Libraries
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

ready bounded by, among other things, memory speed. Parallel computers, in which a number of processors can work in parallel on a single application, offer the only solution capable of providing orders of magnitude of improvement in computing performance without excessive costs.

There has been substantial investment in parallel processing over the past twenty years. In the past five years we have seen an accelerated effort in this area. During this period, at least thirty or forty companies started marketing parallel computer systems. Shared memory parallel computers include MIMD machines such as the Alliant, Encore, Sequent, and Cray Y-MP. Distributed memory computers include MIMD machines such as the Transputer [12], Warp [1], and hypercubes [3], as well as SIMD machines such as the Connection Machine [21] and DAP [11]. Many more parallel machines of enhanced capabilities are under development. Successful use of these parallel computers has been demonstrated in a number of application areas, including scientific computing, signal and image processing, and logic simulation. For some of these applications, the available parallelism increases as the problem size expands; therefore it is possible to achieve close to linear speedups on parallel machines.

The next generation of parallel computers will allow easier, more efficient and more flexible use of parallelism than the present generation. To be scalable to a large number of processing nodes, new parallel computers will be mainly distributed memory machines. However, for programmability these machines will support various shared memory programming models. To achieve the next level of performance, these machines will allow multiple levels and forms of parallelism, and include heterogeneous processing nodes whenever appropriate.

These requirements for next generation parallel computers call for *multicomputer* architectures that have general networking support and extremely low internode communication latencies. Existing message-passing machines [3, 19] already represent a strong starting point in this direction. To illustrate the capabilities of the new machines, we describe briefly in Sections 2 and 3 the iWarp [4] and Nectar [2] systems whose development Carnegie Mellon is currently involved in. iWarp represents a multicomputer built with an embedded network, whereas Nectar represents a multicomputer built around a switching network.

Every multicomputer supports a set of internode communication methods. It is basically the set of supported internode communication methods that makes one multicomputer differ from another and one machine generation differ from another. To aid in understanding the design space of multicomputers, we offer in Section 4 a taxonomy of internode communication methods. Using this taxonomy, the internode communication methods supported by various systems are classified in Section 5. Using this classification system, the capabilities of the new systems can be clearly contrasted with those of previous ones. Section 6 presents some concluding remarks.

2 iWarp: Multicomputer with an Embedded Switching Network

iWarp is a multicomputer architecture being developed jointly by Carnegie Mellon and Intel Corporation. Evolved from its predecessor, Warp [1], iWarp is expected to support a wide range of applications including high speed signal, image and scientific computing.

An iWarp system is an array of identical processing nodes, called iWarp cells. Each iWarp cell is composed of the iWarp component and memory chips. As depicted in Figure 1, the iWarp component contains both a powerful computation agent (20 MFLOPS and 20 MIPS) and a high throughput (320 megabytes/sec), low latency (100-150 ns) communication agent for interfacing with other iWarp cells. Because of its strong computation and communication capabilities, the iWarp component is a versatile building block for various high performance parallel systems.

iWarp systems may range from special purpose systolic arrays to general-purpose distributed memory computers. They are able to support efficiently both fine-grain parallel and coarse-grain distributed computation models simultaneously in the same system. As in the hypercube and Transputer machines, a general communication network is embedded in the iWarp array to support a variety of communication methods (see Section 5).

In an iWarp cell the computation agent can carry out computations independently from the operations being performed by the communication agent. Therefore the cell may perform its computation while communication through the cell to and from other cells is taking place, without the cell program being involved with the communication. While separating the control of the two agents makes programming easy, having the two agents on the same chip allows them to cooperate in a tightly coupled manner. The tight coupling is required to implement architectural features such as streaming and spooling to be discussed below.

The communication agent has four input and four output physical

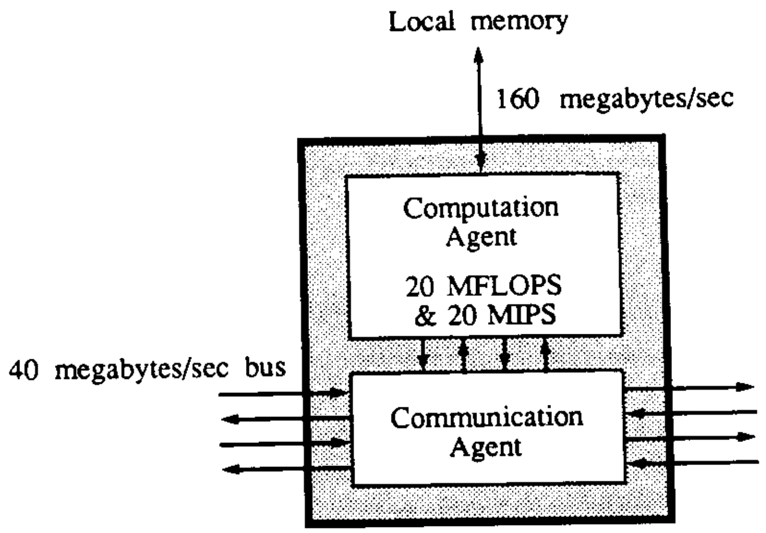

Figure 1. iWarp component overview

busses for connecting to other cells, as shown in Figure 1. In addition, it has two input and two output physical busses for connecting to the computation agent. Each bus has a data bandwidth of 40 megabytes/sec.

An important feature of these physical busses is that each can support a number of *logical* busses in the same direction. The logical busses share the physical bus in a time-multiplexing manner according to a round robin schedule on a word-level basis. The scheduler allocates cycles to *active* logical busses only; idle logical busses consume no physical bus bandwidth. Moreover, a flow control mechanism is implemented in hardware so that whenever a data word is transferred over a logical bus the receiver is guaranteed to have space to receive it. The logical bus architecture and its word-level flow control mechanism, made possible by the VLSI implementation, are essential for the efficient implementation of some sophisticated communication methods, which will be discussed in Section 5.

Logical busses are statically allocated to physical busses under software control. The hardware allows the total number of incoming logical busses in the communication agent of each cell to be as large as 20. For example, in a 2D array configuration, the logical busses can be evenly distributed among the four neighbors and the computation agent, as shown in Figure 2. In this case, the communication agent can be thought as a 20×20 crossbar that links incoming logical busses to outgoing logical busses.

Using the logical busses, a cell can maintain many connections simul-

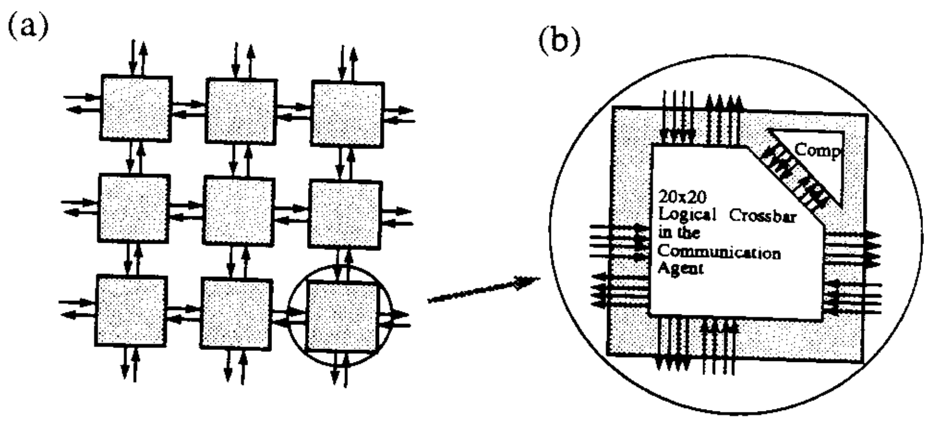

Figure 2. (a) Physical 2D network (b) logical busses of a cell

taneously, including some statically allocated connections called "system pathways"devoted to system uses only. Figure 3 shows an example of three connections through cells in a 2D array. Connection 1 is from the computation agent of cell B to that of cell A. Connection 2 passes through cell A, turns a corner at cell C and then reaches the destination D. Finally, connection 3 passes through both cells C and D. Note that on the same physical bus from cell C to cell D, two connections are maintained at the same time using two logical busses.

Programs can read or write data from or to a message buffer via the side effects of special register references. These special registers are called *streaming gates*, because they provide a "gating" or "windowing" function allowing a stream of data to pass, a word at a time, between the communication agent and the computation agent. There are two input gates and two output gates. These gates can be bound to different logical busses dynamically. A read from the gate will consume the next word of the associated input message; correspondingly, a write to an output gate will generate the next word of the associated output message. The instruction spins when reading from an empty gate or writing to a full gate.

iWarp also provides a transparent, low-overhead mechanism for transferring data between the communication agent and the local memory. The transfer is done via *spooling gates*. Spooling has low overhead to avoid significant reduction of the efficiency of any ongoing computation. Spooling is transparent to software except for delays incurred due to either cycle stealing or local memory access interference from other memory references.

As of the end of 1988, the architecture and logic designs for iWarp were completed. In the software area, the optimizing compiler developed for Warp [10, 16] has been retargeted to generate code for iWarp. Using

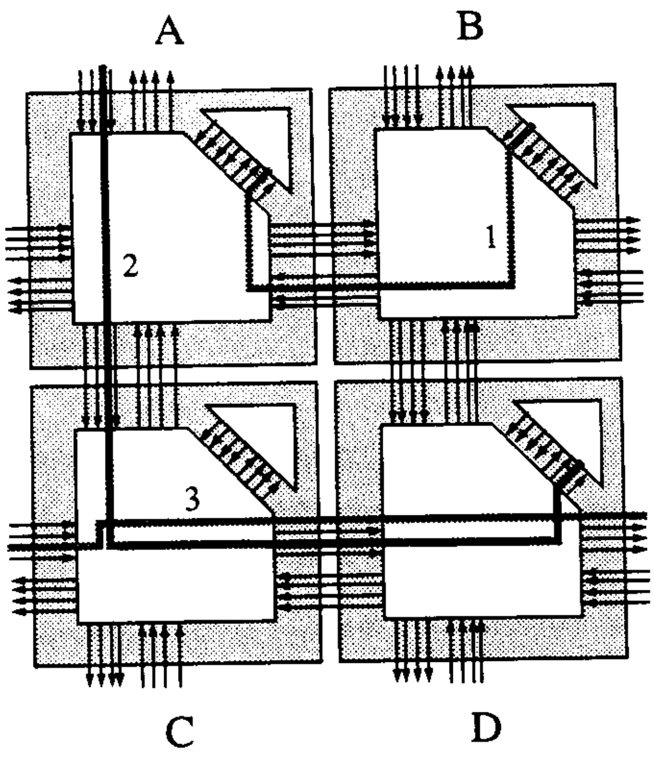

Figure 3. Multiple connections in a 2D iWarp array

this compiler together with an architecture simulator, the iWarp architecture and performance on realistic programs have been evaluated [7]. A prototype iWarp system is expected to be operational by the end of 1989. Three 1.28 GFLOPS demonstration systems, each consisting of an 8×8 torus of iWarp cells, are scheduled to be operational in the middle of 1990. The same system design is extendible to a 20.48 GFLOPS, 32×32 torus.

3 Nectar: Multicomputer around a Switching Network

The Nectar (NEtwork CompuTer ARchitecture) system developed by Carnegie Mellon consists of a *Nectar-net* and a set of identical CABs (Communication Accelerator Boards), as illustrated in Figure 4. The system connects a number of existing (and possibly heterogeneous) systems, called *nodes*, to the Nectar-net. Therefore, unlike an iWarp system, Nectar is a multicomputer built around a switching network rather than a multicomputer with an embedded network. By being able to include commercially available computers and machines optimized for special applications as nodes, Nectar can take advantage of performance improvements

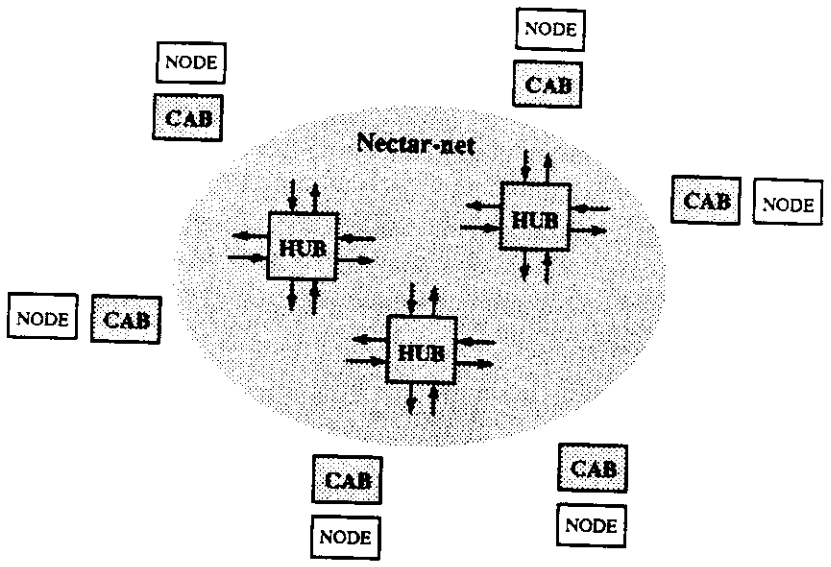

Figure 4. Nectar system overview

in these systems.

The Nectar-net is built from fiber-optic lines and one or more HUBs. As depicted in Figure 5, a HUB is a crossbar switch with a number of fiber I/O ports. Each I/O port contains circuitry for optical to electrical and electrical to optical conversion. From the functional viewpoint, a port consists of an input queue and an output register.

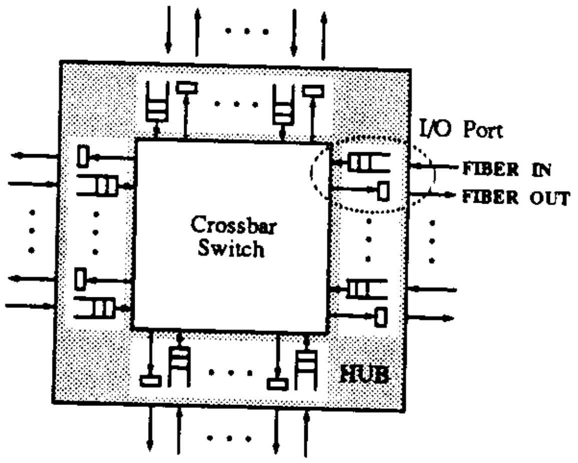

Figure 5. HUB overview

The CAB is a RISC-based processor board serving three functions: it implements high-level network protocols; it provides the interface between the Nectar-net and the nodes; and it off-loads application tasks from nodes when appropriate. Every CAB is connected to a HUB via two fiber lines carrying signals in opposite directions.

To build large systems, multiple HUBs can be connected together. In such systems, some of the I/O ports on each HUB are used for inter-HUB fiber connections, as shown in Figure 6. The HUBs may be connected in any topology appropriate to the application environment. Since the I/O ports used for HUB-HUB and for CAB-HUB connections are identical, there is no *a priori* restriction on the number of links that can be used for inter-HUB connections.

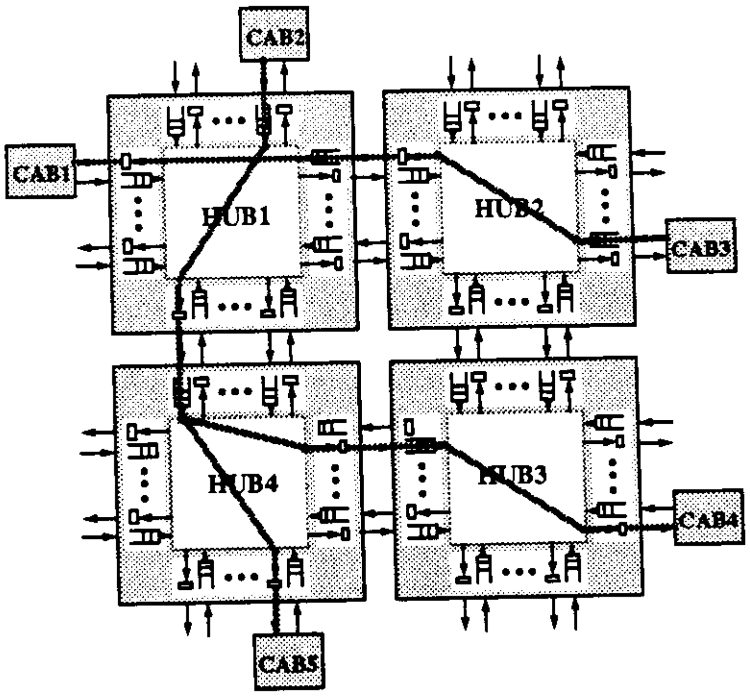

Figure 6. A 4-HUB system example, with some point-to-point and multicast connections shown

A packet-level flow control mechanism for inter-HUB communication is implemented in hardware. When the input queue of a CAB or a HUB I/O port becomes available to store a new packet, the CAB or HUB will automatically send out an acknowledgment signal to the connecting HUB.

A prototype Nectar system is operational at Carnegie Mellon. The prototype uses 16×16 HUBs, SPARC processors as RISC CPUs in CABs, and 100 megabits/sec fiber-optic lines. Currently a node interfaces to the CAB via a VME interface. Initially, the prototype Nectar system at Carnegie Mellon has Sun-3s, Sun-4s and Warp systems as nodes.

A major goal of the Nectar architecture is to have small communication latencies. Excluding the transmission delays of the optical fibers, the latency for a message sent between processes on two CABs is under 30 microseconds; the corresponding latency for processes residing in nodes is under 100 microseconds; and the latency to establish a connection through a single HUB is under 1 microsecond.

Since the latency through the HUB is relatively small, similar process-to-process latencies can be observed on a large Nectar system with, for example, 1,024 hypercube-connected HUBs. Such a system will be able to accommodate 6,144 nodes using the current 16×16 HUBs. The same configuration can have more nodes if HUBs use larger crossbars. For example, if 32×32 HUBs are used, then the system can have up to 22,528 nodes!

The efficiency of Nectar is achieved with a combination of hardware support and carefully tailored network software. In particular, the most frequently used datalink protocols are implemented directly in hardware, while high-level protocols are handled by CABs using streamlined interrupt mechanisms and light-weight processes.

When the prototype has demonstrated that the Nectar architecture and software works well for applications, we plan to re-implement the system using custom or semi-custom VLSI. This will lead to larger systems with higher performance and lower cost. In particular, it will be possible to scale up the current Nectar fiber links from 100 megabits/sec to about 1 gigabit/sec, by taking the following steps:

- The CAB's current bandwidth of 133 megabits/sec can be increased by a factor of eight to 1.062 gigabit/sec using the following method. (1) A factor of two can be obtained by using two DMA devices, one working on one memory bank for packet transmitting and one working on another memory bank for packet receiving, (2) a factor of two by doubling the datapath width from 32-bit to 64-bit, and (3) another factor of two by reducing the cycle time from 60 nanoseconds to 30 nanoseconds. Items (1) and (2) will increase the CAB gate count by roughly a factor of 2, while item (3) can be accomplished with a 30 MHz RISC processor.

- Suppose that the datapath of the current HUB is widened from 8-bit to 64-bit. Then with the current cycle time of 60 nanoseconds, this will yield a bandwidth of 1.062 gigabit/sec. This method requires about eight times more gates for the

HUB crossbar than the current implementation.

Therefore by implementing the CAB and HUB in VLSI, the Nectar architectures can be scaled to accommodate 1 gigabit/sec fiber-optic links.

4 Taxonomy of Communication Methods

One of the most essential properties of a multicomputer is the kind of internode communication methods the machine supports. Many communication methods have been supported by multicomputers and proposed in the literature. This section gives a taxonomy of these methods so that the relations and differences between them can be precisely evaluated.

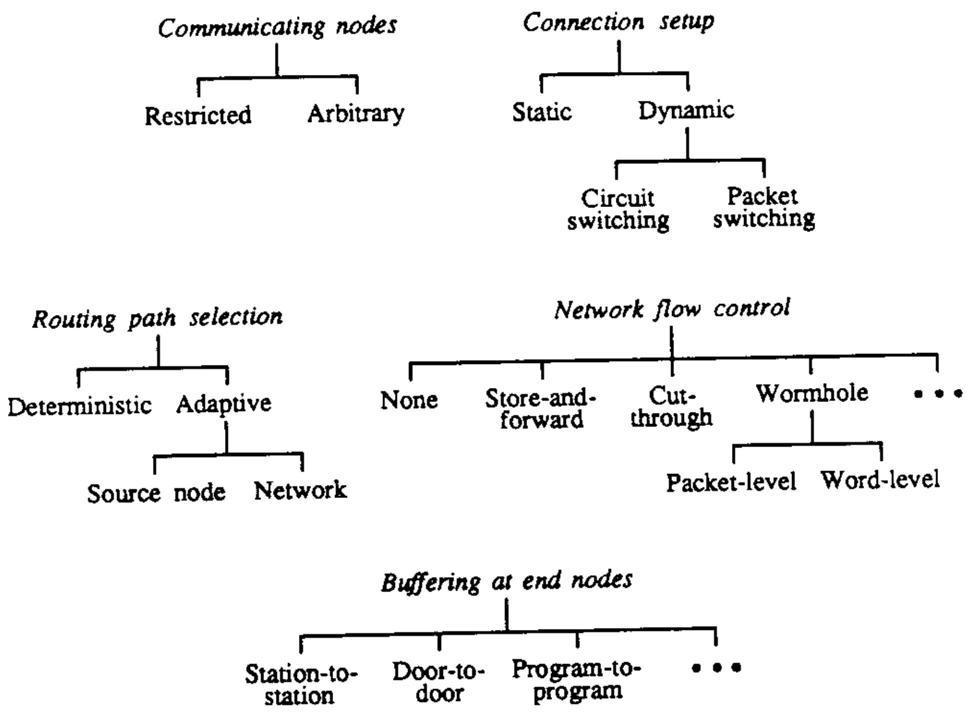

Figure 7. Taxonomy of internode communication methods

As shown in Figure 7, our taxonomy is based on five dimensions. These dimensions are orthogonal to each other in the sense that selection of an alternative in one dimension does not restrict options in other dimensions. In the following we define these dimensions and describe major alternatives with respect to each dimension.

1. *Communicating nodes.* In an application program, a given

node may communicate directly with a number of other nodes. The node may communicate with either a *restricted* set of nodes (such as its neighboring nodes) or any *arbitrary* node in the system.

2. *Connection setup.* A connection is a physical routing path on the network allocated to carry out one or more communications required by the application during program execution. A connection may be set up either *statically* before program execution or *dynamically* during program execution. In the dynamic case, a connection can be set up using *circuit switching*, in which transmission starts only after the entire connection has been set up, or using *packet switching*, in which transmission can start as soon as the connection to the next node has been set up.

3. *Routing path selection.* To build a connection for a given communication between two nodes, there could be multiple choices for the physical routing path. The routing path selection can be either *deterministic* or *adaptive*. In the deterministic case, the route is totally determined by the source and destination addresses of the connection. In the adaptive case, the route selection can attempt to adapt to the network status. When setting up a connection dynamically during program execution, the route can be selected by the *source node*, which originates the communication, or by the *network* [5] to avoid congested nodes on-the-fly.

4. *Network flow control.* During transmission a message packet may not be able to proceed to the next node while the next node is busy. Network flow control is concerned with methods to avoid network queue overflows and underflows in the presence of the possibility that messages may be blocked inside the network.

Before describing some network flow control methods, we first point out that message blocking can be totally avoided by starting transmission only after the entire connection has been set up and the destination node has acknowledged its readiness to accept the package. This method does not require any network flow control support, but has a relatively long communication latency (to set up the connection and to wait for acknowledgement from destination).

A common way to implement network flow control is based on the *store-and-forward* method. When arriving at each intermediate node, the complete packet is first stored in the system memory of the node, and then sent forward to a selected neighboring node when the neighbor is not busy. Note that this method involves at most two nodes at any given time in transmitting a packet; therefore the implementation is relatively easy. However, the buffering of the packet consumes memory space and bandwidth of intermediate nodes and introduces communication delays.

A more sophisticated method, called *cut-through* [13], can avoid unnecessary buffering. When the header of a message packet arrives at an intermediate node whose selected output channel is free, the packet will be directed to the next node immediately (hence the name cut-through). Therefore a message packet is buffered at an intermediate node *only* when its selected output channel is busy.

Alternatively, message packets can stay inside the network while waiting for the selected output channel to become available. Methods of this kind, called the *wormhole* methods [9, 8] by researchers at Caltech, totally avoid the overhead of buffering packets in system memories of intermediate nodes. For the efficient support of these methods, the network must have hardware to implement low-level flow control. The flow control granularity can be at the *word-level* or *packet-level.*

5. *Buffering at end nodes*. For some communication methods, message packets need to be buffered in the system memory at the sending or receiving node, or at both ends. Here we describe some of these buffering schemes.

Consider first the case of buffering message packets at both end nodes. That is, packets can be sent and received only from system buffers. At the sending end, packets built in the user space need to be copied to the system buffer before they can be sent. Correspondingly, at the receiving end, packets received into the system buffer need to be copied to the user space before they can be read. Therefore the network is responsible for transmitting messages only between system buffers. We call this type of communication *station-to-station* communication.

A more efficient method, which we call *door-to-door* communication, allows packets to be sent and received from user spaces directly. This method has a smaller communication latency than station-to-station communication, because there is no need to copy data to and from system buffers.

A step further in this direction is to allow the application program at the sending or receiving end to send or consume individual data items as they are computed or received, respectively. The sending program does not need to build a complete packet before sending out a single data item, nor does the receiving program need to receive the complete packet before reading a single item in the packet. This method, which we call *program-to-program* communication, is the essence of systolic communication [4, 14]. Program-to-program communication can have the minimum-possible latency, since individual data items can be sent out as soon as they become available at the sending end, and can be used as soon as they are received at the receiving end. However, careful coordination between sending and receiving programs is needed to avoid deadlocks [15].

The buffering schemes at the sending and receiving ends do not have to be the same. For example, door-to-door communication may be used at the sending node, when station-to-station communication is used at the receiving node.

5 Communication Method Examples

The space of communication methods is the cross product of the sets of leaves of the trees in Figure 7. The communication methods shown in the table below are some interesting points in this space. These methods are supported by various multicomputers, which are now in existence or soon will be, as indicated in the table.

	Communicating nodes	*Connection setup*	*Routing path selection*	*Network flow control*	*Buffering at end nodes*
Classical message-passing (e.g., first-generation hypercubes [3])	Arbitrary	Dynamic	Deterministic	Store-and-forward	Station-to-station
Classical systolic communication (e.g.,Warp [1])	Restricted	Static	Deterministic	Wormhole (Word -level)	Program-to-program
Efficient message-passing I (e.g., Ametek [20], iWarp [4])	Arbitrary	Dynamic	Deterministic	Wormhole (Word-level)	Station-to-station
Efficient message-passing II (e.g., Hyperswitch [18])	Arbitrary	Dynamic (Circuit switching)	Adaptive (Network)	None	Door-to-door
Efficient message-passing III (e.g., Nectar [2])	Arbitrary	Dynamic	Adaptive (Source node)	Wormhole (Packet-level)	Door-to-door
Efficient message-passing IV (e.g., bi-directional Transputer 1D array [17])	Arbitrary	Static	Deterministic	Store-and-forward	Door-to door
Efficient message-passing V (e.g., iWarp system pathways [4])	Arbitrary	Static	Adaptive (Network)	Wormhole (Word-level)	Station-to-station
Flexible systolic communication I (e.g., iWarp [4])	Arbitrary	Dynamic	Adaptive (Source node)	Wormhole (Word-level)	Program-to-program
Flexible systolic communication II (e.g., reconfigurable systolic arrays [6])	Arbitrary	Static	N/A	Wormhole (Word-level)	Program-to-program

From the table we can observe two trends in the multicomputer evolution. First, message-passing systems are becoming more efficient by supporting techniques such as wormhole, adaptive routing path selection by the network, and door-to-door communication methods. Second, systolic systems are becoming more flexible by allowing communication between arbitrary nodes, dynamic connection setup, and adaptive routing path selection by the source node.

It is interesting and useful to note that both making message-passing systems more efficient and making systolic systems more flexible have the same hardware requirements [4] such as those provided in iWarp (see Section 2). This is summarized in Figure 8. More precisely, hardware supported, low-level flow control prevents network queue overflows and underflows when wormhole, door-to-door and program-to-program techniques are used. Through the use of logical busses, a blocking message inside the network does not have to block physical busses from other uses. To reduce message congestion, the spooling mechanism makes it possible to remove messages from the network easily and quickly when they reach destinations. Using streaming, programs can read and write messages directly to implement program-to-program communication. To implement door-to-door communication, the same streaming mechanism allows the programs to read quickly the header of an arriving message to determine its destination memory location.

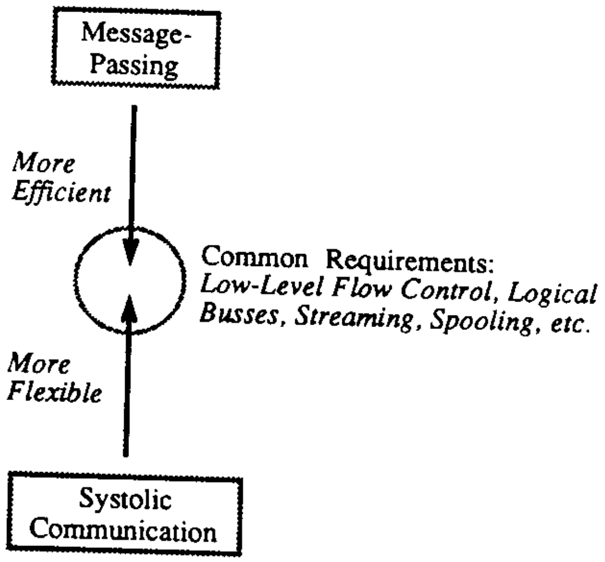

Figure 8. Common requirements for efficient message-passing and flexible systolic communication

6 Concluding Remarks

We have seen in this paper the feasibility of using VLSI to implement networking functions very efficiently. In the case of iWarp, network delays are comparable to access latencies to the local memory of a cell. In the case of Nectar, whose network operates at the packet-level, network delays are typically much smaller than file access latencies on a sequential machine. Based on these high-speed networks multicomputers can enjoy the communication flexibility of general networks without sacrificing performance.

As VLSI technology advances, sophisticated and well-understood communication methods will be supported directly by hardware. This can be clearly seen from the classification of communication methods given in the paper. The iWarp effort can be viewed as a part of this overall trend.

Nectar represents a more general approach than iWarp. Being able to use existing processors as nodes, Nectar can take advantage of rapid performance advances in commercial processors. Being able to include processors of different types, Nectar can support applications that require heterogeneous nodes. Moreover, the system is based on a general network architecture that is highly scalable. As pointed out in the paper, the system can be scaled up to have a large number of nodes by using multiple HUBs and fiber-optic links, and to support high bandwidth links by implementing the CAB and HUB in VLSI. Because of these advantages, we expect that network-based multicomputers such as Nectar will play an influential role in parallel processing in the 1990s.

Acknowledgments

The research was supported in part by Defense Advanced Research Projects Agency (DOD) monitored by the Space and Naval Warfare Systems Command under Contract N00039-87-C-0251, and in part by the Office of Naval Research under Contracts N00014-87-K-0385 and N00014-87-K-0533. Portions of Sections 2 and 3 are excerpted from existing articles on iWarp and Nectar [2, 4].

References

[1] Annaratone, M., Arnould, E., Gross, T., Kung, H. T., Lam, M., Menzilcioglu, O. and Webb, J. A., The Warp Computer: Architecture, Implementation and Performance, *IEEE Transactions on Computers C-36*, 12 (December 1987), 1523-1538.

[2] Arnould, E. A., Bitz, F. J., Cooper, E. C., Kung, H. T., Sansom, R. and Steenkiste, P. A., *The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers*, Proceedings of Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III), ACM, (April

1989).

[3] Athas, W. C. and Seitz, C. L., Multicomputers: Message-Passing Concurrent Computers, *Computer 21*, 8 (August 1988), 9-24.

[4] Borkar, S., Cohn, R., Cox, G., Gleason, S., Gross, T., Kung, H. T., Lam, M., Moore, B., Peterson, C., Pieper, J., Rankin, L., Tseng, P. S., Sutton, J., Urbanski, J. and Webb, J., *iWarp: An Integrated Solution to High-Speed Parallel Computing*, pp. 330-339, Proceedings of Supercomputing '88, IEEE Computer Society and ACM SIGARCH, (Orlando, Florida, November 1988).

[5] Chow, E., Madan, H., Peterson, J., Grunwald, D. and Reed, D., *Hyperswitch Network for the Hypercube Computer*, pp. 90-99, Conference Proceedings of the 15th Annual International Symposium on Computer Architecture, (June 1988).

[6] Cohn, R., Kung, H. T., Menzilcioglu, O. and Song, S. W., *A Highly Reconfigurable Array of Powerful Processors*, pp. 336-343, Proceedings of SPIE Symposium, Vol. 975, Advanced Algorithms and Architectures for Signal Processing III, Society of Photo-Optical Instrumentation Engineers, (August 1988).

[7] Cohn, R., Gross, T., Lam, M. and Tseng, P. S., *Architecture and Compiler Tradeoffs for a Long Instruction Word Microprocessor*, Proceedings of Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III), ACM, (April 1989).

[8] Dally, William J., *A VLSI Architecture for Concurrent Data Structures,* (Kluwer Academic Publishers, 1987).

[9] Dally, W. J., and Seitz, C. L., The Torus Routing Chip, *Distributed Computing 1*, 4 (1986), 187-196.

[10] Gross, T. and Lam, M., *Compilation for a High-performance Systolic Array*, pp. 27-38, Proceedings of the SIGPLAN 86 Symposium on Compiler Construction, ACM SIGPLAN, (June 1986).

[11] Hockney, R.W. and Jesshope C.R., *Parallel Computers,* (Adam Hilger Ltd., Bristol, U.K., 1981).

[12] Homewood, M., et al., The IMS T800 Transputer, *IEEE Micro 7*, 5 (October 1987), 10-26.

[13] Kermani, P., and Kleinrock, L., Virtual Cut-Through: A New Computer Communication Switching Technique, *Computer Net-*

works 3, 4 (1979), 267-286.

[14] Kung, H. T., *Systolic Communication*, pp. 695-703, Proceedings of the International Conference on Systolic Arrays, (San Diego, California, May 1988).

[15] Kung, H. T., Deadlock Avoidance for Systolic Communication, *Journal of Complexity 4*, 2 (June 1988), 87-105. (A revised version also appears in Conference Proceedings of the 15th Annual International Symposium on Computer Architecture, June 1988, pp. 252-260)..

[16] Lam, M., *A Systolic Array Optimizing Compiler*, Ph.D. Thesis, Carnegie Mellon University (May 1987).

[17] Manning, L. J., Dew, P. M., and Wang, H., Design and Analysis of Image Processing Algorithms for Programmable VLSI Array Processors, in: Page, I., Ed., *Parallel Architectures and Computer Vision*, (Oxford Science Publications, 1988), pp. .

[18] Peterson, J., Chow, E., Madan, H., *A High-Speed Message-Driven Communication Architecture*, pp. 355-366, Conference Proceedings of 1988 International Conference on Supercomputing, (St. Malo, France, July 1988).

[19] Seitz, C. L., The Cosmic Cube, *Comm. ACM 28*, 1 (January 1985), 22-33.

[20] Seitz, C. L., Athas, W. C., Flaig, C. M., Martin, A. J., Seizovic, J., Steele, C. S. and Su, W-K., *The Architecture and Programming of the Ametek Series 2010 Multicomputer*, pp. 33-36, The Third Confererence on Hypercube Concurrent Computers and Applications., (Pasadena, California, January 1988).

[21] Tucker, L. W., and Robertson, G. G., Architecture and Applications of the Connection Machine, *Computer Magazine 21*, 8 (August 1988), 26-38.