Microprocessor-based Parallel Architectures Using Multiport-Memory Interconnection Networks

Paul J. Wilder, University of Southern Mississippi
Ronald F. DeMara, University of Central Florida

ABSTRACT

Parallel computer interconnections based on multiport memories offer attractive alternatives to link-oriented or bus-oriented interconnection networks (ICNs) for the rapid prototyping of microprocessor-based parallel machines. This paper presents an overview of multiport memory ICNs. It focuses on the *MemNet* hypercube interconnection network, which uses overlapping groups of four-port memories. The network provides each of the *N* processing elements (PEs) with Concurrent Read Exclusive Write (CREW) access to $\log_4 N$ multiport memory modules. Along each of the cube's *n* dimensions, memory is shared with three other PEs for a connectivity of 3^n, where $n = \lceil \log_4 N \rceil$. High connectivity is achieved while requiring on the order of $N \log N$ memories. Details of a one-dimensional four-processor system are described, including a basic multiprocessing laboratory outline.

INTRODUCTION

Multiprocessing systems are infiltrating the workplace, and are currently available from several workstation platform manufacturers. Additionally, many PC graphics accelerator cards use dual-port video RAM and multiprocessing to display real-time graphics and animation, allowing load sharing between a separate graphics processor and the system processor, ultimately speeding up graphics presentation. In an effort to keep pace with and promote current technology in computer and electronics engineering technology programs, the curricula should provide exposure to simple multiprocessing concepts and techniques.

The advent of low-cost multiprocessor system architectures based on microprocessors has enabled a large increase in processing potential over uniprocessor architectures. This has resulted in a shift of the processing and implementation limitations back to the peripheral devices, I/O interface, and the networking strategy that interconnects processors to create a coherent computing structure. These processor interconnection strategies, if not carefully designed, can become a limiting factor in the efficient solution of application problems. Data, control information, and messages must be routed through this network with maximum throughput and minimum processing overhead. Many new ICN schemes have been described for interprocessor communication. Requirements for these ICNs include: low cost, low overhead for both software and hardware, high bandwidth, redundancy, and modularity.

Low cost refers to the size and complexity of the hardware requirements for interconnecting *N* PEs together into a network structure. For example, $N = 8$ PEs can be interconnected such that a link exists between each possible combination of two PEs, as shown in figure 1. This is referred to as *completely connected*, and results in a total of $N \times (N - 1)/2 = 28$ bidirectional links. Costs under full interconnection will scale $O(N^2)$, where the *Order* notation $O(N^2)$ denotes the set of functions which grow proportionally to N^2 as *N* increases.

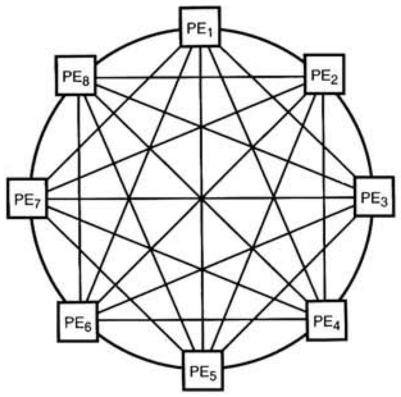


Figure 1. The completely connected case for eight PEs

The same eight PEs can be connected as a linear sequence requiring only seven links, as shown in figure 2. In the completely connected case, a message can be sent from one PE to any other PE in only one hop. In the linear array case, a message may have to traverse as many as seven hops through intermediate PEs. This demonstrates the tradeoffs between connectivity and complexity. A compromise must be sought to create efficient use of interconnection resources which achieve high connectivity at low expense.

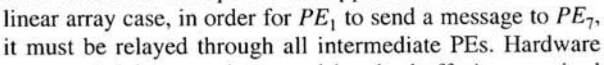


Figure 2. Linear array case for eight PEs

Low overhead refers to the amount of software processing resources dedicated to the transfer of messages. In the above example, the overhead is low for the fully connected case. If a message is sent by PE_1 and received directly by PE_7, very little overhead is required to support the transfer. In the linear array case, in order for PE_1 to send a message to PE_7, it must be relayed through all intermediate PEs. Hardware costs and delays are increased by the buffering required during message transfer.

Bandwidth refers to a link's data transfer capabilities. Larger bandwidths imply greater throughput. In the example above, assuming links with a constant bandwidth, higher throughput is achieved by the fully connected ICN. This is because the distance between all PEs remains constant at unity. Each link can operate at full bandwidth yielding maximum throughput. In the linear array case, the links are shared for all communication, direct as well as intermediate. The intermediate traffic loads each link, reducing net throughput.

Redundancy refers to the ability of the ICN to withstand the loss of communication links. If the ICN can gracefully degrade its operation as links are lost, the design is more robust. Consider again the above ICN examples. The completely connected case is mildly impacted if one or two links are lost, as many redundant communication paths exist. If a single link is lost in the linear array case, the ICN is divided into two parts and becomes unusable.

Modularity refers to the expandability of the ICN. Since computing problems come in all shapes and sizes, the ICN should be scaleable to meet the demands of a particular task. Many different topologies have been proposed to meet the above criteria.[1]

This paper describes *MemNet*, which is a memory-oriented ICN. MemNet employs multiport memories, rather than shared buses or dedicated communication links, to optimize performance in terms of the above parameters. In a uniprocessor system, CPU data and address buses are directly connected to a memory device's single port. Using a four-port memory, the MemNet ICN retains the simplicity of single processor operation respective to each PE in the system, yet provides the performance benefits of a fully connected ICN for a specific memory device. A distinct memory port is provided for each PE, supporting interprocessor connectivity through the multiport memory, which serves as the communication link. Essentially, the ICN is contained within the memory device itself.

The design of the MemNet architecture is shown in the following sections, along with a performance comparison to existing ICNs. Also included is a discussion on the memory hardware requirements for the ICN. A simple one-dimensional four-processor system example is described, which is suitable for intermediate-level microprocessor architecture laboratories.

PREVIOUS WORK

Buses are the simplest ICN, and are very easy to reconfigure. Advantages are low cost in hardware and software. Buses typically have adequate reliability. The average distance is 1, and the diameter is 1. Disadvantages are: low throughput due to time sharing of the communications resource; contention, a problem which may require arbitration hardware and/or software reducing the cost advantages; and fault tolerance, where loss of a bus can cause a complete loss of communication and catastrophic system failure. Expandability is not particularly difficult, but can increase contention, thus reducing throughput. Arbitration costs may increase also.

A simple approach to increase performance and fault tolerance when using buses is to increase the degree, or dimension, of the ICN from one to two. This change yields the two-dimensional spanning bus hypercube shown in figure 3. This increases the complexity of the ICN. The hardware cost increases dramatically, as well. The number of nodes is $N = W^D$, where N equals the number of PEs, W is the width, and D is the dimension. This requires $DW^{(D-1)}$ buses for complete communication.[2]

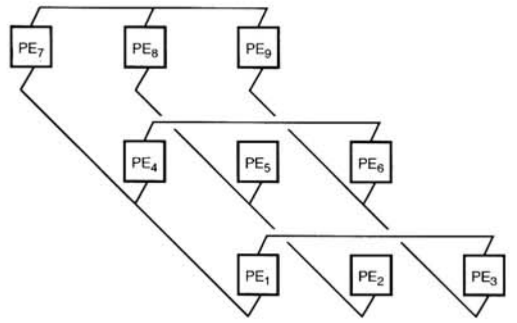


Figure 3. Two-dimensional spanning bus hypercube static network

In figure 3, $W = 3$ and $D = 2$, thus $N = 9$ resulting in six buses. Path redundancy is achieved, improving fault tolerance. The number of communication ports per PE, i.e., the degree of the node, increases by one, yielding two communication ports per processing element. The average distance is given by

$$\frac{D \times (W-1) \times W^{(D-1)}}{N-1}.$$

A generalized hypercube ICN strategy was proposed by Bhuyan and Agrawal for multiprocessing systems.[3] The topology allows a variety of hypercube structures for differing network sizes, dependent on network diameter. Their paper outlines both the link-oriented and the bus-oriented hypercube structures. The generalization removes a limiting factor in the binary hypercube structure by relaxing the constraint that $N = W^D$ must have an integer value for W or D.

The results showed that the generalized hypercube structures have lower costs than other hypercube structures, high connectivity, and good fault tolerance. The hypercube structures reduce average message traffic density and average message distance. As shown in the section *Performance Parameters*, these grow $O(\log N)$. This value is also the degree of a node in the network. The performance of these strategies lies somewhere between that of a ring (the lower limit of performance) and a completely connected ICN (the upper limit of cost).

DUAL-BUS HYPERCUBE

The dual-bus hypercube, shown in figure 4, was introduced by Wittie as an alternative to the spanning bus hypercube and the cube connected cycles network strategies.[1,2] The only major difference is that the dual-bus hypercube has a fixed number of connections per node as compared to the spanning bus hypercube's $\log_2 N$ connections per node. Every node in the base dimension connects to a spanning bus. The other dimensional planes are divided up such that each of the nodes in the $D - 1$ perpendicular planes have their second connection buses spanning the same dimension. This second bus orientation differs from plane to plane, and is repeated if $W > (D - 1)$.

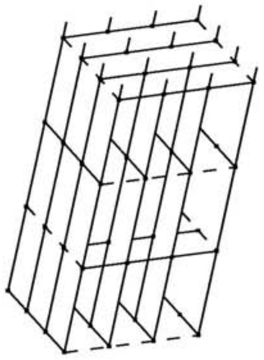


Figure 4. Dual-bus hypercube with vertical base direction

Although the reduction of node degree permits connection costs per node to remain constant, the higher integration available today easily allows for an increased number of ports per PE, mitigating this advantage. Since each message uses nearly twice the number of buses in transport and there are $D/2$ fewer buses, the average message traffic density is D times that of the spanning bus hypercube. Also, due to an asymmetrical structure, bus allocation in the $D - 1$ nonbase dimensions requires careful task mapping to assure equal traffic density.

Dual-Port Memory Interconnection Network

A variation on the link-oriented interconnection strategy was previously proposed by Jagadish et al. as a preliminary memory-oriented ICN which offered advantages over link-oriented communication channels.[4] It offered built-in buffering capability and reduced contention, while still maintaining the scalability advantages of the hypercube structure. Dual-port memories are used in place of dedicated communication links.[5]

This topology relied on a *network controller*, an intermediate processor for global communication between nonadjacent nodes, to help reduce the routing overhead. For example, a three-dimensional cube of 8 PEs required 1 network controller and 20 dual-port memories. A 64-node extension uses 9 network controllers and 168 dual-port memories. If the network controllers are not used, component count could be reduced to 12 and 108 memories respectively, at the expense of decreased performance. If this scheme is used to build a complete 64-node hypercube without network controllers, the same number of dual-port memories as links, 192, are required to construct the ICN.

MEMNET TOPOLOGY

This paper proposes a parallel machine, *MemNet*, based on communication through multiport memories, which eliminates the network controllers. A performance comparison of existing ICNs with MemNet is shown in the section *Performance Parameters*. An efficient memory partitioning strategy is formalized. Utilization of a combination firmware/hardware approach virtually eliminates contention on the network.

Interconnection Strategy

The topology of the MemNet ICN is based on a 4-ary hypercube. It is similar to the binary and the spanning bus hypercubes, but it differs significantly because of its use of multiport memory modules in place of point-to-point communication links or buses. Each memory interface in the multiport network is reserved for a single processor-memory pair. PEs exchange messages by writing to a shared memory region in a multiport memory. The motivation for this approach includes decreased average distance, lower hardware cost, rapid implementation, improved scalability, and a simple routing algorithm with low software overhead.

Commercially available technology for multiport memories includes two- and four-port designs with eight-port modules under development.[6] Each port can address any location independently and simultaneously for read or write except for two constraints.[7,8,9] First, concurrent writes from two ports to the same location are indeterminate. Second, simultaneous reads and writes to the same location are prohibited. A memory partitioning strategy is developed to avoid contention problems.

An example of a 32-processor network composed of four-port memories is shown in figure 5. Note that the three dimensions of this network are depicted as horizontal (x-dimension, between the two sections, PEs 0–15 and PEs 16–31), vertical (y-dimension, between PEs within a unit cell as shown in figure 6), and depth (z-dimension, into the page) for each processing node.

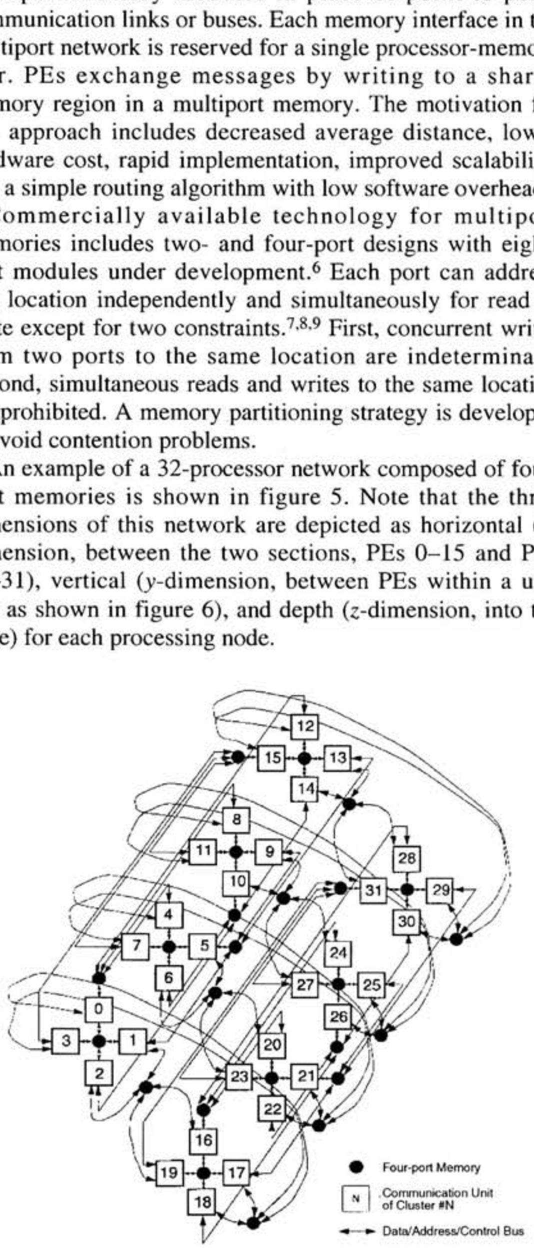


Figure 5. A 32-CPU 4-ary network composed of $p = 4$ port memories

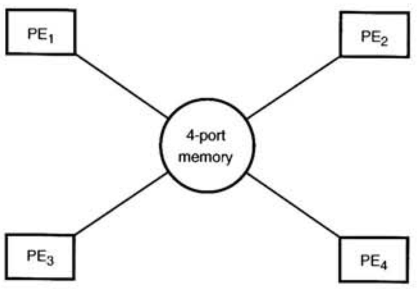


Figure 6. The unit cell using four processors with a single four-port memory

Routing Permutation

Routing is performed using the Hamming distance between corresponding fields of the source and destination PE addresses.[3] Consider an example of a message transfer in the 32 PE system of figure 5. Routing from PE_3, address 000011, to PE_{12}, address 001100, can be performed in two hops. The dimension of the network is $D = \lceil \log_4 32 \rceil = 3$. Each bit field contains $\log_2 W = 2$ bits, since $W = 4$ by virtue of the four concurrent ports available in each memory. Let the three bit fields be denoted as x, y, and z such that each address is represented as $xxyyzz$ along the dimensions defined above. In this example the y and z bit fields differ between source and destination, while the x bit fields are the same. The first routing step can occur along either the y- or z-dimensions based on a predetermined, random, or load-adaptive strategy. If the y-dimension is traversed first, then PE_3 will transfer the message to address 000000, which represents PE_0. When the message is received at PE_0, it only needs to be transferred along the z-dimension. The message is then sent from address 000000 to 001100, the destination node PE_{12}, thus completing the message transfer.

Hardware Requirements

Lower average distance can be achieved with reduced hardware as compared to link-oriented ICNs. Figure 6 shows that only a single four-port memory is needed to connect $N = 4$ PEs. If all memory hardware is to be fully utilized, then the ICN must grow with the number of PEs $N = O(p^i)$, otherwise $i = \log_4 N$ assumes a nonintegral value. This implies that all available memory ports are not connected to distinct PEs, as in the $N = 32$ PE example shown in figure 5. As a result, the smallest 4-ary MemNet ICN, having four PEs and one four-port memory, has a network dimension $D = 1$.

The growth of the number of ports per PE in a binary hypercube is $\log_2 N = O(\log N)$. This is directly related to the number of links required in the ICN for maintaining the hypercube structure. A similar result applies for the 4-ary case, but with a base 4 logarithm $\log_4 N = O(\log N)$. The number of four-port memories required is $O(N \log N)$. Since each multiport memory can connect to four PEs, the total number of PEs is divided by 4, $N/4$. Each time the ICN's dimension grows, the number of communication ports per processor grows $O(\log N)$. Multiplying these terms results in the number of multiport memories required:

$$\frac{N}{4} \times (\log_4 N). \qquad (1)$$

Further substantiation of the above can be seen by examination of dimensional analysis:

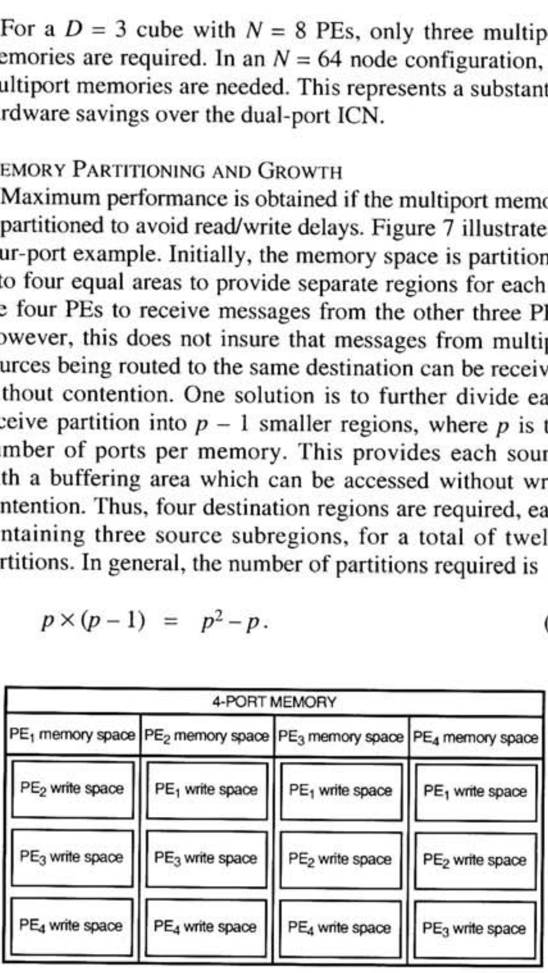
$$\text{Memories} = \frac{\#\text{ of PEs}}{\#\text{ of Ports/Memory}} \times \frac{\#\text{ of Ports}}{\text{PE}}. \qquad (2)$$

For a $D = 3$ cube with $N = 8$ PEs, only three multiport memories are required. In an $N = 64$ node configuration, 48 multiport memories are needed. This represents a substantial hardware savings over the dual-port ICN.

Memory Partitioning and Growth

Maximum performance is obtained if the multiport memory is partitioned to avoid read/write delays. Figure 7 illustrates a four-port example. Initially, the memory space is partitioned into four equal areas to provide separate regions for each of the four PEs to receive messages from the other three PEs. However, this does not insure that messages from multiple sources being routed to the same destination can be received without contention. One solution is to further divide each receive partition into $p - 1$ smaller regions, where p is the number of ports per memory. This provides each source with a buffering area which can be accessed without write contention. Thus, four destination regions are required, each containing three source subregions, for a total of twelve partitions. In general, the number of partitions required is

$$p \times (p-1) = p^2 - p. \qquad (3)$$



4-PORT MEMORY			
PE_1 memory space	PE_2 memory space	PE_3 memory space	PE_4 memory space
PE_2 write space	PE_1 write space	PE_1 write space	PE_1 write space
PE_3 write space	PE_3 write space	PE_2 write space	PE_2 write space
PE_4 write space	PE_4 write space	PE_4 write space	PE_3 write space

Figure 7. Memory partitioning scheme ensures contention-free operation

Moreover, this has significant implications on the *useful capacity* of multiport memories in a MemNet ICN. If C_1 is the capacity of the currently available p_1-port memory, then to maintain equivalently sized buffers for a p_2-port memory will require a capacity not less than

$$\frac{(p_2)^2 - p_2}{(p_1)^2 - p_1} \times C_1. \qquad (4)$$

Thus, as the number of ports increases by a factor of p_2/p_1, the memory capacity must increase $O[(p_2/p_1)^2]$. Current multiport memory technology typically opts for reduced density.

Sentinel Element

One problem with using multiport memories is that a protocol is needed for accessing the shared regions between processing elements. One technique employs a *sentinel element*. This approach operates independent of hardware or operating system support since it does not require semaphores, monitors, nor shared counter variables. It is applicable for broadcast modes between a single producer and multiple consumers where each consumer has its own input buffer.

Classical solutions to the *bounded buffer* problem employ synchronization mechanisms such as semaphores and monitors. For systems with a single-producer and a single-consumer, neither mutual exclusion nor semaphores are strictly required. A simpler, more parallel implementation can be achieved using the sentinel element approach. The technique for a single-producer/single-consumer system is presented, followed by a description of a constant time complexity extension to $p - 1$ consumers. In particular, due to the CREW capability of the multiport memory, there is no increase in protocol execution time regardless of the number of consumers added, up to and including the number of ports available on each memory.

The pseudocode for the sentinel algorithm is shown in tables 1 through 3. Initially, the entire buffer pool is filled with the sentinel element as shown in table 1. During operation of the algorithm, the producer and consumer execute asynchronously. Whenever the producer generates data or the consumer requires data, they execute the code in tables 2 and 3, respectively. In the producer code, a quick test for the sentinel element is performed to guarantee that a write will occur only if the location is empty. Likewise, a test for the sentinel by the consumer ensures that the consumer will only read from a location if it has been filled. Thus, statement *L1* prevents overflow of the buffer, while statement *L2* prevents underflow. Since no counter variables are shared between the producer and consumer, each may access the ring buffer in an asynchronous manner without the need for semaphores. Busy/waiting time may also be utilized effectively in this algorithm by performing housekeeping or other operations between retry iterations of the *L1*/*L2* loops.

Table 1. Sentinel protocol initialization procedure

```
p_ptr : = 0                   /* producer pointer */
c_ptr : = 0;                  /* consumer pointer */
S : = sentinel element;       /* unique nondata element
                                 e.g., 0 or other End-of-File indicator) */
buf[0 . . bufsize-1] : = S;   /* initially clear the buffer */
```

Table 2. Producer process

```
L1: if buf[p_ptr] ≠ S then goto L1    /* wait for free location */
buf[p_ptr] : = item;                  /* place item in buffer */
p_ptr : = p_ptr + 1 mod bufsize;      /* increment pointer */
return
```

Table 3. Consumer process

```
L2: if buf[c_ptr] = S then goto L2    /* wait for item */
item : = buf[c_ptr] ;                 /* retrieve item from buffer */
buf[c_ptr]: = S;                      /* indicate the cell is cleared */
c_ptr : = c_ptr + 1 mod bufsize;      /* increment pointer */
return
```

Extension to $p - 1$ consumers is straightforward if each consumer has its own buffer and if the properties of the application preclude overflow. In particular, if the producer and consumers periodically synchronize before the buffer fills, the test for overflow may be omitted. The code for consumer processes then only needs to be modified to provide a distinct consumer pointer for each consuming process (i.e., replace the simple variable *c_ptr* by the arrayed variable *c_ptr*[1 . . $p - 1$] where *c_ptr*[$p - 1$] is the pointer for consumer process number $p - 1$).

The sentinel approach provides a simpler solution to the bounded buffer problem than semaphores/monitors, while allowing maximum speed of operation. To see the potential speed increase, consider that classical solutions give exclusive access rights for the entire buffer to a single user. Under the conditions stated above, this is stricter than necessary. Rather than reserving the entire buffer, it is possible instead to allow the producer and consumer to utilize the simultaneous access capability of the multiport memory. Thus, the producer can add new items to the buffer at the same time that the consumers are removing old ones.

Example Architecture

The simplest example of the MemNet topology is the unit cell shown in figure 6. This one-dimensional system is constructed of four PEs and a single four-port memory. Implementation can be achieved using any off-the-shelf microprocessors/microcontrollers and a four-port memory. Processors can even be mixed and matched, using different clock speeds. The data bus from each PE is designated as the communication port; therefore each of the four PEs in the system is connected to one of the ports on the four-port memory.

Communication in multiprocessing systems becomes readily demonstrable in an inexpensive hands-on technology laboratory. Memory interface design is typically covered in an introductory microprocessor architecture course in most computer engineering and technology programs. This presents an opportunity to introduce concepts in inter-processor communication techniques and protocols as advanced topics. This is also appropriate in a second semester computer architecture course which addresses contemporary uses of distributed processing. Once the system is operational, experiments can be designed to implement message passing, i.e., data and control information, to perform multiprocessing tasks. These can range from multiuser games and elementary multiprocess control problems, to high-performance real-time applications.

A stand-alone board can be fabricated to hold the required memories, decode logic, and header strips for ribbon cable connectors. Memory partitioning can be done through

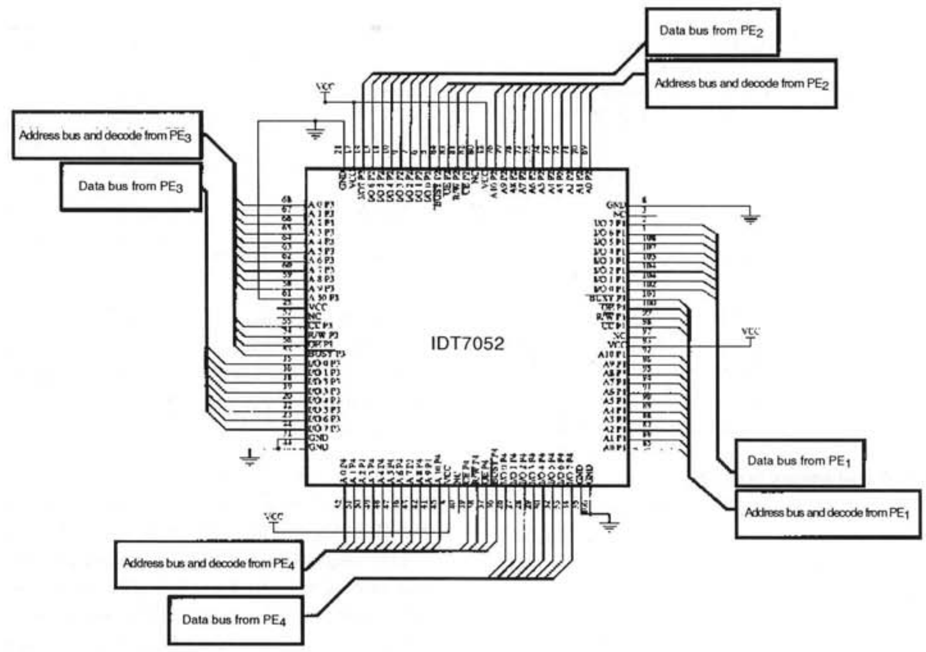

Data bus from PE2
Address bus and decode from PE2
Address bus and decode from PE3
Data bus from PE3
IDT7052
Data bus from PE1
Address bus and decode from PE1
Address bus and decode from PE4
Data bus from PE4
VCC

Figure 8. Schematic of stand-alone board

decode logic or in software. A generalized schematic is shown in figure 8. This board can be interfaced with a microprocessor training platform to illustrate interprocessor communication. Power can be provided from a separate supply, an on-board regulator, or through a ribbon cable.

The IDT7052 device shown in figure 8 is available in a variety of packages. It is TTL-compatible, and supports several different access times. Construction of the multiprocessing system in the laboratory follows similar procedures to that of single processor system interfacing. The data pins of each microprocessor are connected to the corresponding data pins of the multiport memory. Similarly, address and control lines are directly connected between processor and memory. After supplying power and ground connections, the rudimentary multiprocessing system is ready for laboratory use.

PERFORMANCE PARAMETERS

Under each of the following headings, a description of a performance criterion is given to assess MemNet performance relative to that for other ICNs. It compares several key factors by looking at their rates of increase as N, the total number of PEs in the network, grows large. Results are listed in tables 4 and 5. Each entry in a category is assessed a numeric point value listed in the lower half of table 4. These points are added to create a composite score for each network presented.

It is assumed that message delay and message traffic density have a uniform distribution of messages across the networks.[2] Thus, the average rate at which PE_i sends messages to PE_j is the same for all PEs in the network. PEs can send one message per time unit. The rate for relayed messages is not limited.

AMDPUT

The *average message delay in path use times* (AMDPUT) is derived from the average distance for message transfer. In some ICNs messages are routed along sequential links or buses by relay through intermediate PEs. The single-bus network requires arbitration schemes to alleviate contention, which are implemented as time slotting. While the average distance in a single-bus network is one, allowances must be made for the delay associated with the arbitration algorithm. This increases the effective average distance by adding latency, expressed as extra hops, to the total time for message transfer, thus increasing the AMDPUT. For the case of the single-bus with N time slots connecting N PEs, the AMDPUT is $O(N/2)$.

The near-neighbor mesh (two-dimensional hypercube), spanning bus hypercube, binary tree, cube connected cycles, dual-bus hypercube, and MemNet all have the same AMDPUT.[1] As N increases, the AMDPUT grows $O(\log N)$.

MLTU

The *message density in messages per link per time unit* (MLTU) is a measure of the communication load on a given channel. The single-bus network with N time slots connecting N PEs is loaded with N messages per time unit. As N gets large, this results in a value of $O(N)$ for the growth of the MLTU.

The binary tree network's communication load is nonuniform across the structure. As the root PE is approached from the leaf PEs, the communication load per link increases dramatically. The two links connecting the root PE to its two main branch PEs carry the greatest MLTU in the network. As N is increased, the average MLTU increases $O(2N)$.

Table 4. Performance parameters

	AMDPUT	MLTU	CPE	TCCN	Buffering	CNR	NC	TO	WLSCF
MemNet	log$N$	Fixed	log$N$	$N$log$N$	Integral	None	log$N$	Local	None
CCC	log$N$	log$N$	Fixed	N	N	log$N$	Fixed	None	None
Dual-bus	log$N$	log$N$	Fixed	N	N	log$N$	log$N$ - W	None	None
Span. bus	log$N$	Fixed	log$N$	$N$log$N$	$N$log$N$	log$N$	log$N$	None	None
Mesh	log$N$	Fixed	log$N$	$N$log$N$	$N$log$N$	log$N$	log$N$	None	None
Bin. tree	log$N$	$2N$	Fixed	N	N	log$N$	Fixed	None	Fail
Sing. bus	$N/2$	N	Fixed	N	N	N	N	None	Fail
Best	Fixed:6	Fixed:6	Fixed:6	N:6	Integral:6	None:6	N:6	Local:6	None:6
Good	logN/N:4	log$N$:4	Nlog/N:4	$N$log$N$:4	log$N$:4	log$N$:4	log$N$:4	Cluster:4	Local:4
Fair	log$N$:2	$N^{1/x}$:2	log$N$:2	N^2:2	N:2	log$N$ - W:2	Nlog/N:2	Global:2	Extra:2
Poor	N:0	N:0	N:0	N^3:0	$N$log$N$:0	N:0	Fixed:0	None:0	Fail:0

Table 5. Results of composite scoring

Network	Score
MemNet	42
Dual-bus	32
Spanning bus	32
Mesh	32
CCC	30
Binary tree	20
Single-bus	20

The dual-bus hypercube and the cube connected cycles both exhibit the same dependency on N for MLTU of $O(\log N)$.[2]

The MLTU is the same for the near-neighbor mesh, spanning bus hypercube, and MemNet. As N increases, $D = \log_W N$ increases to maintain the hypercube structure. This results in the MLTU not being dependent on the growth of the network, and therefore its values remain constant.

CPE

Connections per processing element (CPE) reflects the number of communication ports per PE or the degree of the node. It is also the network dimension W in hypercube topologies. The near-neighbor mesh, spanning bus hypercube, and MemNet are all hypercube structures. As a result, as N increases, the degree of the node for each structure grows at similar rates. This is given by $D = \log_W N$. CPE growth is therefore $O(\log N)$.

The single-bus, binary tree, dual-bus hypercube, and cube connected cycles all exhibit the same performance in this category. Since no dependence on N exists in these network structures, the degree of the node remains fixed.

TCCN

The *total connection costs per network* (TCCN) are based on the requirements for interconnection hardware. The number of links or buses is shown for each of the network strategies. The near-neighbor mesh, spanning bus hypercube, and MemNet structures exhibit the same TCCN. As N grows large, D and W must assume specific values to maintain the hypercube structure. Since $D = \log_W N$ is the number of communication ports per PE, and each port typically connects two PEs, the connection costs grow $O(N\log N)$.

The single-bus network, binary tree, dual-bus hypercube, and cube connected cycles all exhibit the same growth in this category. The connection costs increase linearly with the size of the structure. This is due to the costs associated with adding each PE to the network. For each PE added, a single link must be added to the network structure. This results in connection costs growing $O(N)$ as N grows large.

Buffering Capabilities

Input and output buffering is needed at each network interface. Since the number of interfaces per PE in the near-neighbor mesh and spanning bus hypercube grows $O(N\log N)$, and two buffers are needed for storage at each interface, the hardware requirements for these network strategies grows $O(N\log N)$.

Assuming bidirectional links, the binary tree and cube connected cycles require input and output buffering for message transfer at each port of a PE. The single-bus and dual-bus hypercube fall into this category since the buses act as time-shared links. This requires that both input and output buffering be present for each port of the PEs in these networks. The buffering hardware growth in these networks is $O(N)$.

The MemNet ICN does not require any external buffering devices. Buffering is integrated into the network structure. Since the information exchange between PEs occurs through multiport memories, automatic buffering is provided without increased cost.

CNR

Contention for network resources (CNR) occurs as a PE is accessing the network physical link. It directly impacts network throughput. Arbitration schemes are necessary for contention reduction. Costs increase as a result. Overall network performance can suffer if CNR remains high. With the exception of the MemNet ICN, all of the above interconnection strategies suffer from contention. The MemNet ICN is a contention-free ICN, thus resulting in no dependence on the growth of N. This is due to the memory partitioning strategy discussed in the section *Memory Partitioning and Growth.*

Node Connectedness

Node connectedness (NC) is a measure of how well connected a PE is to the network. It is the number of PEs at a given distance from the source PE. For example, a linear array of linked PEs has $NC = 2$ for all PEs which reside at a distance of one. In this example, as N grows large, the connectedness remains constant.

The binary tree and cube connected cycles exhibit similar results for connectedness. Since the number of connections per node is independent of the growth of N, the connectedness remains constant with $NC = 3$.

The dual-bus hypercube has a slightly higher connectedness. For the dual-bus hypercube, the connectedness is based on both CPE and W. Since the i^{th} bus connects W PEs, the connectedness is $2W$. This implies that connectedness grows $O(N^{1/D})$ for the dual-bus hypercube.

The spanning bus hypercube and MemNet share a similar growth of connectedness as network size increases. Both of these network strategies connect the W PEs, in the i^{th} position, along a given dimension. The spanning bus hypercube connectedness is given by the network width times the CPE, $W\log N$. For the MemNet, the width is determined by the number of ports in the memory. The growth for these structures is $O(N\log N)$.

The single-bus network exhibits the best result in this category. Since the single-bus connects all PEs in the network, its connectedness grows $O(N)$. The contention in the single-bus ICN significantly reduces the bandwidth available, mitigating this advantage.

Traffic Observability

The *traffic observability* (TO) of the network is a measure of the ease in which the messages can be observed as transfer takes place. It can aid in the setup, programming, application mapping, performance analysis, and debugging of the ICN. The growth of the interconnection network has no direct impact on its observability. Therefore, TO is not assessed as N increases. A network structure simply has or lacks this attribute.

The single-bus, binary tree, near-neighbor mesh, spanning bus hypercube, dual-bus hypercube, and cube connected cycles do not have built-in provisions for TO. The MemNet structure uses multiport memories for the network physical link. This allows messages to reside in the network for future access by diagnostic programs or monitoring PEs. This flexibility can facilitate network setup and diagnostics.

WLSCF

The *worst loss on single-component failure* (WLSCF) indicates the robustness of the network to single component failures. This can range from virtually no effect, to complete degeneration of the network. There are varying degrees of performance loss associated with a single point failure. If a network can still operate under the loss of a single PE, then the network is said to *gracefully degrade*.

The single-bus and binary tree exhibit the same problems with the loss of a single component. If a single-bus fails, the network will cease to function. The binary tree can suffer similar degradation if the root PE fails, or if one of the root PE links fails. The near-neighbor mesh, spanning bus hypercube, dual-bus hypercube, cube connected cycles, and MemNet all exhibit redundancy. If either a PE or link should fail, the impact to network performance is noncritical since multiple redundant communication paths exist.

CONCLUSION

A new memory-oriented ICN, *MemNet*, is shown for rapid prototyping of microprocessor-based parallel machine architectures. Performance is improved relative to link-oriented and bus-oriented structures. Table 5 shows the composite scores for the identified ICNs. The MemNet architecture has the highest composite score for the chosen attributes. It also has reduced hardware costs and provides a platform for exposure to multiprocessing and message-passing systems in the technology lab. Architectures can be implemented using off-the-shelf components with varying clock speeds, providing maximum flexibility in experimental platform operation.

REFERENCES

1. Agrawal, D. P., and V. K. Janakiram. "Evaluating the Performance of Multicomputer Configurations." *Computer* (May 1986): 23-37.
2. Wittie, L. D. "Communication Structures for Large Networks of Microcomputers." *IEEE Transactions on Computers*, Vol. C-30, no. 4 (April 1981): 264-73.
3. Bhuyan, Laxmi N., and Dharma P. Agrawal. "Generalized Hypercube and Hyperbus Structures for a Computer Network." *IEEE Transactions on Computers*, Vol. C-33, no. 4 (April 1984): 323-33.
4. Jagadish, N., J. M. Kumar, and L. M. Patnaik. "An Efficient Scheme for Interprocessor Communication Using Dual-Port RAMs." *IEEE Micro* 10, no. 10 (October 1989): n.p.
5. Cypress Semiconductor Corporation. *Understanding Dual-Port RAMS*. Available: http://www.cypress.com.
6. Forsell, M. J. "Are Multiport Memories Physically Feasible." *Computer Architecture News* 22, no. 5 (December 1994): 3-10.
7. Stodleck, Robert. *The IDT FourPort™ RAM Facilitates Multiprocessor Designs*. IDT Application Note AN-43. Santa Clara, Calif.: IDT, 1995.
8. Milas, N. "New Memory: The Shared Port RAM." *Electronic Engineering* 6, no. 8 (June 1995): 83-90.
9. Wilson, K. M., K. Olukotun, and M. Rosenblum. "Increasing Cache Port Efficiency for Dynamic Superscalar Microprocessors." *Proceedings of the 1996 IEEE International Symposium on Computer Architecture*, May 22-24, 1996, Philadelphia: 147-57.

Paul J. Wilder received the Bachelor of Science degree in Engineering from the University of Vermont in 1986, and the Master of Science degree in Computer Engineering from the University of Central Florida in 1996. He is currently an assistant professor of Engineering Technology, and the Computer, Electronics, and Software Engineering Technology program coordinator at the Univeristy of Southern Mississippi's Gulf Coast campus in Gautier, MS. From 1989-1991 he held a faculty position at Vermont Technical College in Randolph Center, VT. His research interests include multiprocessor design, memory design, and embedded microcontroller systems.

Ronald F. DeMara is a full-time faculty member in the Electrical and Computer Engineering Department at the University of Central Florida in Orlando, Florida. Dr. DeMara received the B.S. in Electrical Engineering degree from Lehigh University in 1987, the M.S. in Electrical Engineering degree from the University of Maryland, College Park in 1989, and the Ph.D. in Computer Engineering degree from the University of Southern California, Los Angeles in 1992. His research interests include parallel processing, computer architecture, and memory system design.