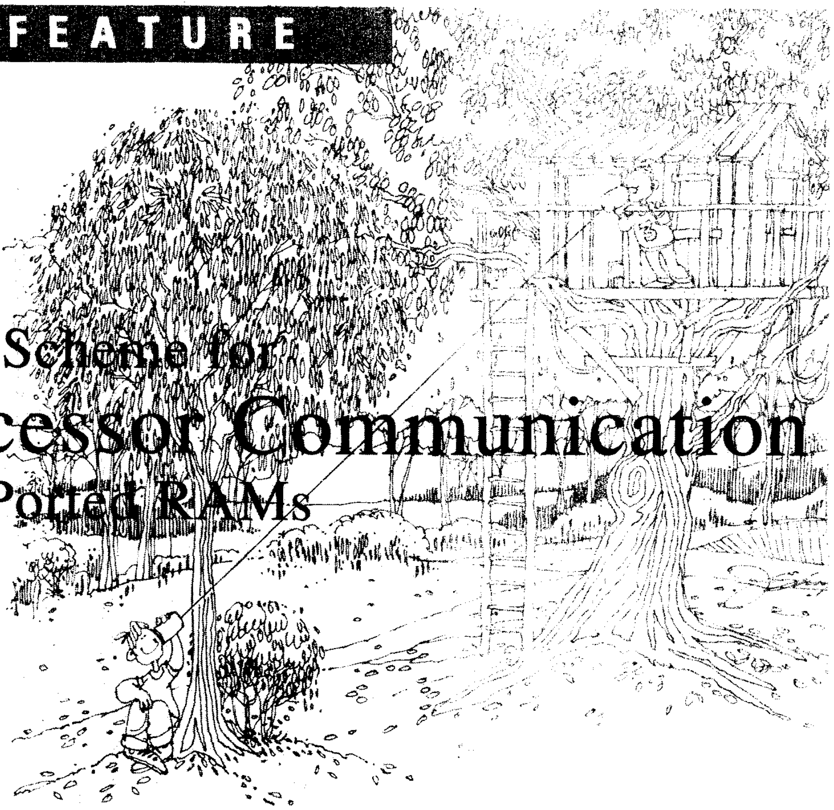
SPECIAL FEATURE



An Efficient Scheme for Interprocessor Communication Using Dual-Ported RAMs

Need a simple, fast, modular system to transfer messages in a multiprocessor? Try this low-cost duplex scheme.

N. Jagadish
J. Mohan Kumar
L.M. Patnaik

Indian Institute of Science

Successful interprocessor communication, a key factor in the design of any multiprocessing system, requires high bandwidth and reliability with minimal cost and software/hardware overheads. Here, we present such a communication scheme. It features simplicity, speed, modularity, and configurability to multiprocessing systems such as linear arrays, triangular arrays, meshes, systolic trees, and hypercubes.

Communication between any two processors in this scheme takes place through a common memory, independently accessible by both processors involved. The interprocessor interconnection scheme in a multiprocessor system directly affects system throughput and has a bearing on the modularity, reliability, and overall system performance. Yalamanchili and Aggarwal discussed the importance of the processor interconnection scheme when they characterized the capabilities of a multiprocessing system.[1]

Various interconnection schemes have been suggested for message passing between processor nodes.[2,3] Tuazon et al.[2] suggested a scheme that makes use of first-in, first-out, or FIFO, buffers and several communication channels. Their scheme involves data-shifting mechanisms and software for polling signals. In this scheme the transfer of a message between two processor nodes involves 1) transferring a message to the FIFO buffers in the source node, 2) converting a message from words to nibbles, 3) transmitting a message from source node to destination node, 4) reconverting the message from nibbles to words in the destination node, and 5) receiving message data from the FIFO buffer in the destination node after checking data-valid flags.

In the Hayes et al. scheme,[3] processor nodes communicate with one another by means of asynchronous direct memory access operations. The message moves through serial channels. Transmission involves 1) DMA transfer from the main memory to a buffer on the processor node, 2) conversion of the message into serial format, 3) transmission on a serial communication channel, 4) reconversion of the message into parallel format, and 5) another DMA transfer from a buffer to the main memory on the destination node.

0272-1732/89/1000-0010$01.00 © 1989 IEEE

The Intel iPSC,[4] a hypercube supercomputer, consists of eight communication channels per node. Intel built the Ethernet protocol-based iPSC using a special local communication coprocessor (82586). The iPSC scheme transfers data at the rate of 10Mbits per second. Carrier Sense Multiple Access with Collision Detection (CSMA/CD), a statistical medium access control system, implements the sharing of common channels.

Our scheme for interprocessor interconnection using dual-ported RAMs and network controllers follows. In this scheme, communication between the processor nodes involves writing into and reading from a common memory area. The communicating processors do not have to contend for a common bus as in the case of shared-memory systems, since they have independent access to the common memory units shared between them. Only the memory access time of the processors limits the communication speed. Processor-to-processor communication does not use intermediate buffers, input/output ports, or DMAs. We consider the example of a three-dimensional cube to illustrate the advantages of this scheme. Further, we discuss the implementation of the interprocessor communication scheme on a 64-node cube configuration.

Processor-to-processor communication

Dual-ported RAMs now available in VLSI form operate at static RAM speeds (50 to 150 nanoseconds) and have two independent left and right ports. Figure 1 illustrates a message transfer between two neighboring processors using dual-ported RAMs. This DPR area is common to both processor elements (PEI and PE2). In other words a portion of the memory space of PE1 "overlaps" a portion of the memory space of PE2. We shall refer to this shared DPR area as the common memory.

Either processor can access the dual-ported RAM independently, since this memory area lies in the memory space of each processor. However, PE1 and PE2 access this area with different addresses. On-chip arbitration logic within the dual-ported RAM handles address contention to ensure maximum speed. In case of contention one of the ports must wait until the other port's access is complete; a BUSY signal on the dual-ported RAM indicates contention.

As shown in Figure 1, the common memory between PE1 and PE2 is logically divided into upper and lower halves. PE1 writes into the upper half and reads from the lower half. Similarly, PE2 writes into the lower half and reads from the upper half. In this way we minimize the probability of access contention. To transmit a message packet to PE2, PE1 writes the message packet in the common memory it shares with PE2. Communication between neighboring processors does not involve intermediate devices. A message written by the transmitting processor in its own memory is accessible to the receiving processor.

Communication between noncontiguous nodes (those not directly connected with each other) can be carried out with the help of an intermediate processor.' Figure *2* illustrates the methodology for communication between noncontiguous processors. A network controller (NC) becomes the intermediate link for message transfer between noncontiguous nodes. (We

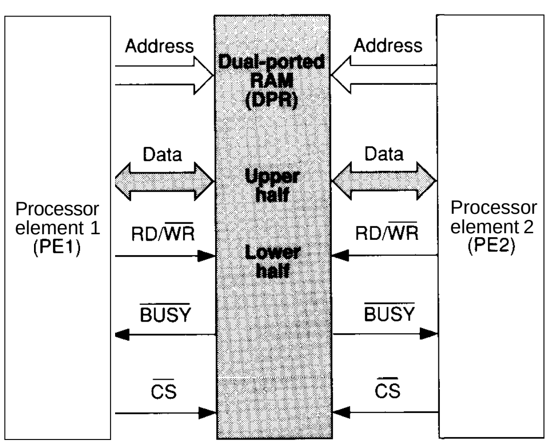


Figure 1. Two processors sharing a dual-ported RAM.

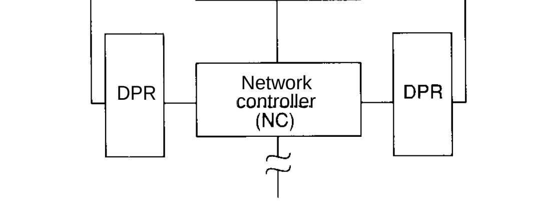


Figure 2. Communication pathways among noncontiguous processors.

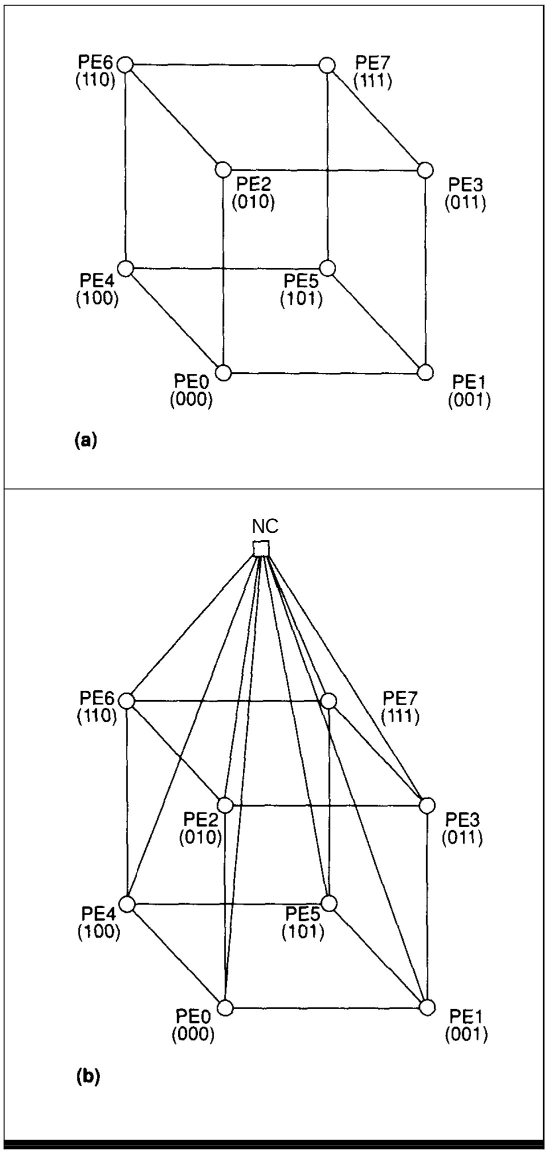


Figure 3. A 3D cube (a) and a 3D cube with a network controller (b).

show the usefulness of the network controller in hypercube configurations in Figure 3 and discuss it later.)

In Figure 2, two noncontiguous processors PE1 and PE3 share separate dual-ported RAMs with the network controller. This controller provides an alternative path for the transfer of messages between two noncontiguous nodes. As indicated in Figure 2, PE1 can transfer messages to PE3 along two alternative paths: one path via PE2, which involves two memory transfers through PE2, and another path through the network controller. In the first method, PE2 participates in the message transfer between PE1 and PE3.

In a multiprocessing system dedicated PEs perform subtasks of a main task. The throughput of the multiprocessing system would be significantly reduced if the PEs were used for communication purposes. We include the network controller seen in Figure 2 to transfer messages between noncontiguous nodes. To transmit a message packet from PE1 to PE3, PE1 writes the message packet into the common memory shared with the network controller; this controller performs a block transfer to shift the message packet to the common memory shared with PE3.

With this kind of design, the PEs in a multiprocessing system need not participate in the communication between noncontiguous nodes as the network controller exclusively performs communication tasks. Obviously, processor-to-processor transfer is most effective in the case of contiguous nodes. Message transfers between noncontiguous nodes must use the path through the network controller. Thus the two paths for message transfers complement each other.

Implementation on a 3D cube

To understand the implementation aspects of our scheme, we suggest a three-dimensional cube, since such a topology has attracted wide interest among researchers in recent years. An n-dimensional hypercube[5] is a multiprocessor characterized by the presence of $N = 2^n$ processors interconnected as an n-dimensional binary cube. Each node of the cube consists of a central processing unit and local main memory. Each PE of the cube directly communicates to n other PEs of the cube; the communication paths correspond to the edges of the cube. The length of the path between any two nodes is simply the number of edges of the path. The minimum distance between any two nodes in an n-cube equals the Hamming distance between them.

Implementing the interconnection network on a 3D cube occurs as follows. The nodes of the cube are numbered as indicated in Figure 3a and b. Each node consists of a PE, which includes the numeric data coprocessor. Each PE shares common memory units with other PEs located at a Hamming distance of one. In addition, the PEs share common memory units with the network controller, as illustrated in Figure 4a. Figure 4b shows the memory map of a typical node processor.

As can be seen in Figure 4c, the processing node contains an 8088 processor with an 8087 numeric data coprocessor, address decoding logic, a wait-state generator, system ROM, local RAM, and dual-ported RAMs. The address decoding logic selects among the system ROM, local RAM, and the dual-ported RAMs

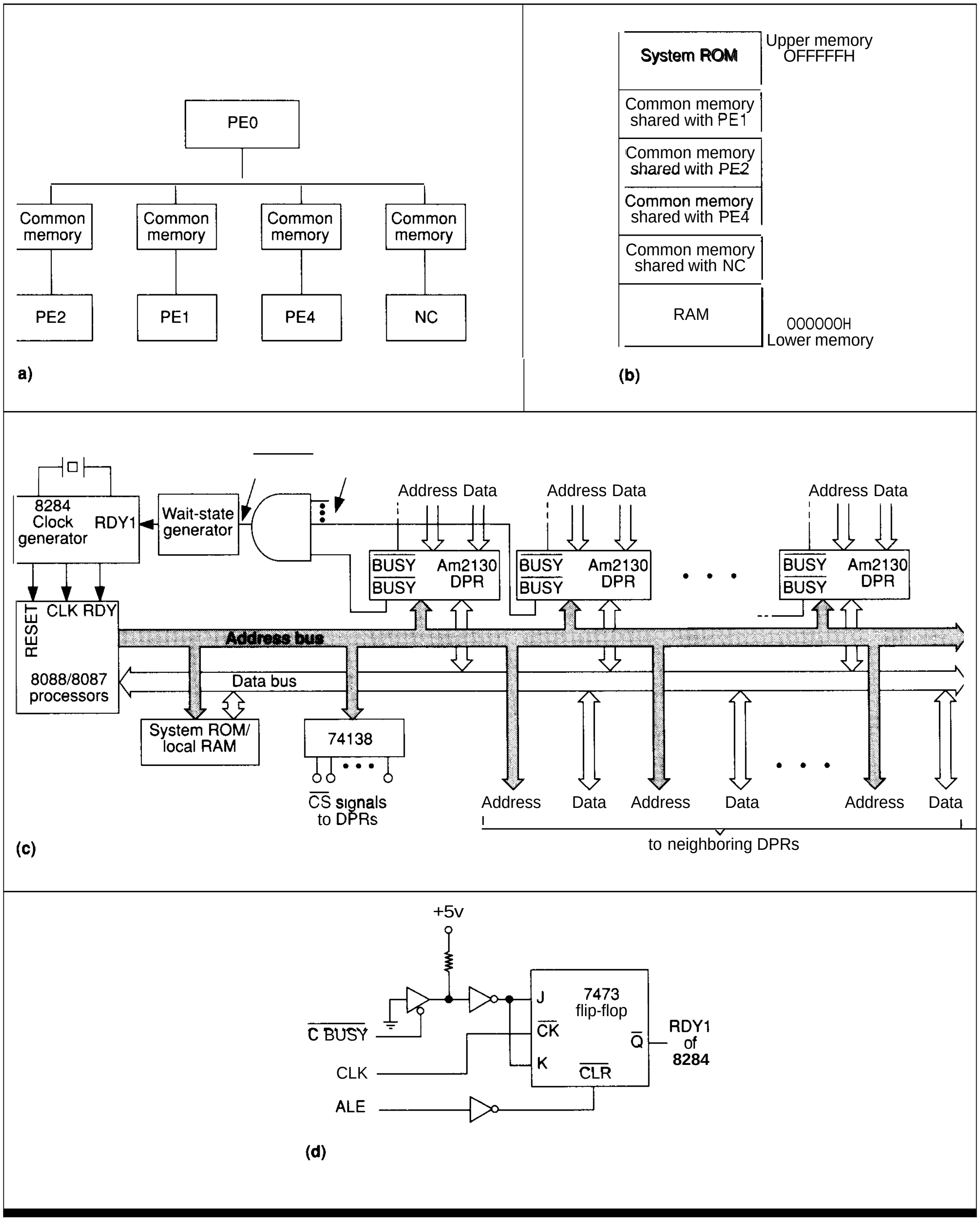


Figure 4. Common memory units at a typical node called PEO (a); a memory map of PEO (b); node processor hardware (c); and a single-wait-state generator (d).

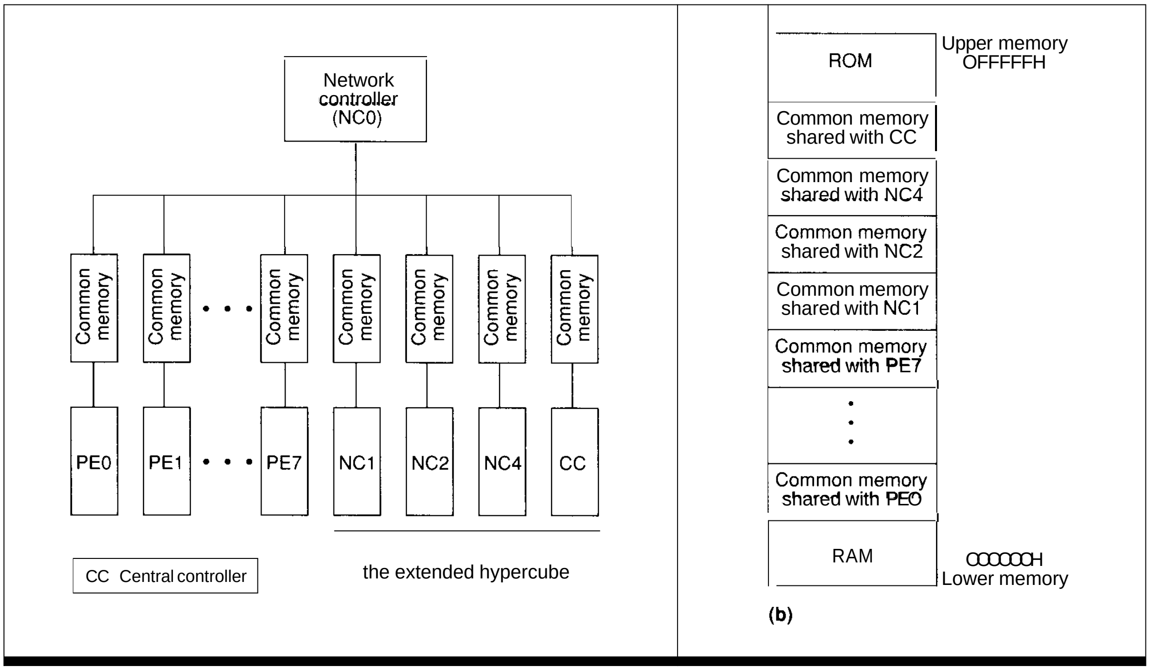


Figure 5. Common memory units at a network controller (a) and a memory map of the network controller (b).

shared between the neighboring processors and the network controller. The wait-state generator takes care of address contention as follows. On detection of a BUSY signal from the dual-ported **RAM,** the wait-state generator disables the ready-line input RDY 1 of the 8284 clock generator for one clock state. See Figure 4d.

The network controller is dedicated to the task of overall interprocessor communication management. The controller shares common memory units with each of the eight nodes in the cube, as illustrated in Figure 5a and b, and its hardware configuration is similar to that of the nodes. **As** indicated in Figure ***6*,** the network controller contains parallel and serial ports for communication with the host system and other input and output devices. In addition, the network controller initializes the cube and distributes tasks.

Common memory units exist between all pairs of neighboring nodes and between the network controller and each node. **As** mentioned earlier, neighboring nodes communicate by directly writing into the common memory located between the two nodes. For communication between nodes located at a Hamming distance greater than one, the network controller performs a memory block transfer from the common memory shared with the transmitting node to the common memory shared with the receiving node. **A** message packet between two noncontiguous nodes can also be routed through one of the parallel paths between the two nodes depending on the availability of the processor~The parallel paths between two noncontiguous nodes may consist of one or more nodes that contribute to the message transfer by block transferring the message packet from the memory space of the transmitting node to the memory space of the receiving node.

Message transfer protocol

The message packet shown in Figure 7a consists of the semaphore/address byte, packet-size byte, and the actual message. The semaphore/address byte (Figure 7b) has three subdivisions. The most significant bit indicates valid data, the next bit indicates processor-busy status, accompanied by three bits for addressing processors in an extended hypercube, and the last three bits indicate the address of the node. The next byte gives the total length of the message in bytes, followed by the message itself. The source node checks the semaphore bits for data validity and writes the message packet either in its common memory shared with the destination processor (if the Hamming distance between them is one), or in its common memory shared with the network controller (if the Hamming distance between the source and destination nodes is greater than one).

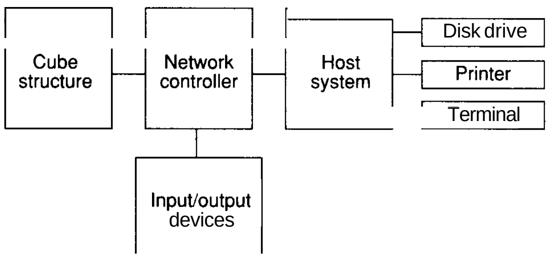


Figure 6. Block diagram of the multiprocessor system with peripherals.

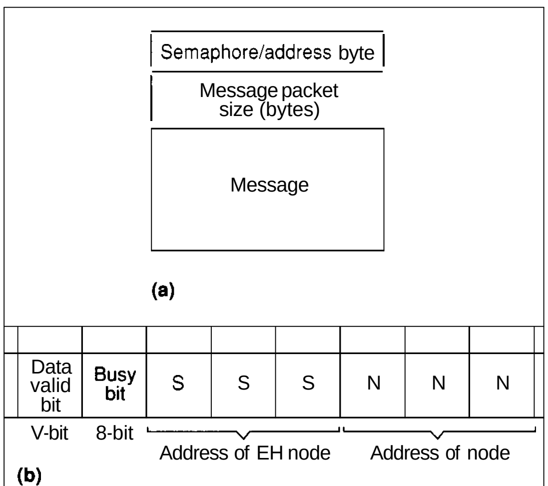


Figure 7. A message packet (a) and a semaphore/address byte (b).

```
Procedure Initialization for network controller
    for PE0 to PE7
      begin
        V-bit := 0; B-bit := 0;
      end;

Procedure Network Controller Block Transfer
  repeat
    for PE0 to PE7
      begin
        if ((S-V-bit) = 1) AND ((D-V-bit) = 0) then
          block transfer data;
          S-V-bit := 0; D-V-bit := 1;
      end;
  forever.

Procedure Send /* for PE */
      begin
        if hamming distance >1 then
          block transfer data to network controller;
          network controller-V-bit := 1;
        else
          block transfer data to destination;
          D-V-bit := 1;
      end;

Procedure Receive /* for PE */
      begin
        if S-V-bit = 1
        then
          block transfer data;
          S-V-bit := 0;
      end;

/* B-bit is busy bit
   S-V-bit is source data validity bit
   D-V-bit is destination data validity bit
   Network controller V-bit is network controller data
   validity bit */
```

Figure 8. The procedure for a message transfer.

As indicated earlier in Figure 3b, PE3 and PE7 (with a Hamming distance of one between them) share a common memory through which they can communicate. The source processor checks the data valid V-bit. If the V-bit is 0, the source processor writes the message in the common memory space and sets the V-bit to 1 to indicate the presence of fresh data. The destination processor checks the V-bits in the common memory units shared with its neighbors. If any of the V-bits are valid, the processor copies into the local memory the message from the common memory following a valid V-bit, and resets the V-bit to 0.

The message transfer between two noncontiguous nodes involves a path through the memory space of the network controller. For example, consider the communication procedure between PE3 and PE4, which are at a Hamming distance of three from each other. The source processor PE4 determines the Hamming distance between itself and the destination processor PE3. Since the Hamming distance is greater than one, PE4 writes the message into the common memory space shared with the network controller if the corresponding V-bit is reset to 0. The network controller checks the V-bits of the control bytes in the common memory units shared between PE4 and PE3. Say the V-bit of PE4 is 1 (indicating fresh data) and the V-bit of PE3 is 0 (indicating previous data accepted). In this case, the network controller transfers a memory block to shift the data stored in the common memory space shared with PE4 to the common memory space shared with PE3. The procedure for message transfer appears in Figure 8.

Scheme extended to 64 nodes

The 64-node extended hypercube, or EH, consists of eight 3D cubes[5] and a central controller node, as illustrated in Figure 9. (We introduce the EH term to reflect that each node of the hypercube is a cube by itself.) Each 3D cube consists of eight individual nodes and the network controller, and we refer to this group as the EH-node (node of the EH). EH-nodes appear at the vertices of the EH. Each of the eight EH-nodes has topological and architectural features similar to that of the 3D cube discussed earlier.

As indicated by the dotted lines in Figure 9, the eight network controllers at the eight EH-nodes form a 3D cube, the EH. The network controller of each EH-node shares common memory units with its neighboring network controllers in the EH. In addition, a central network controller shares common memory units with all the eight network controllers at the vertices of the EH. There is no interconnection network between the individual nodes of different EH-nodes.

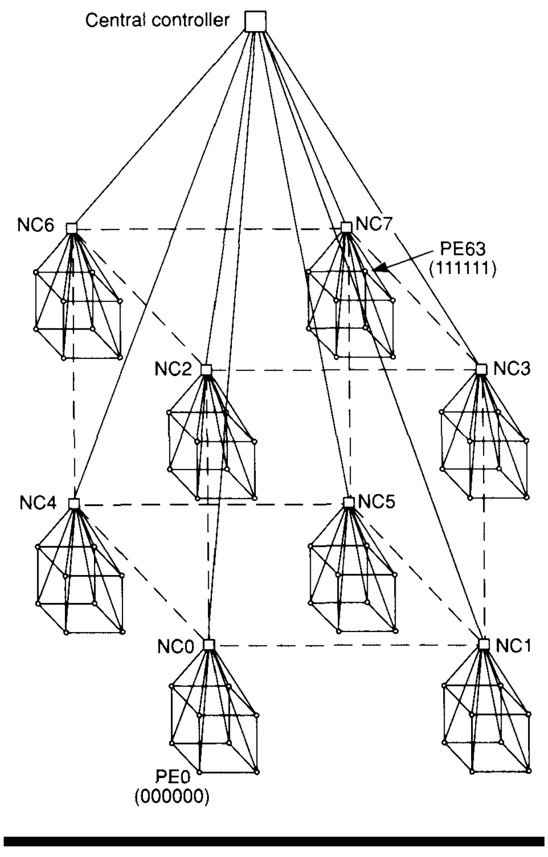


Figure 9. An extended hypercube.

The interprocessor communication scheme between individual nodes of the EH-node is similar to that explained earlier, as is the communication scheme between any two network controllers in the EH consisting of the network controllers and central controller.

A message between two individual nodes in different EH-nodes transmits via the memory space of the network controllers residing in the EH-nodes of the source and destination nodes. A message transfer between two individual nodes having a Hamming distance of six between them (and residing in two different EH-nodes that are themselves at a Hamming distance of three from each other) completes with just three memory transfer operations. No individual node processor (other than the source and destination node processors) participates in the memory transfer operation. Memory transfers can best be explained by considering two such nodes shown in Figure 9, PE0 and PE63, as source and destination nodes respectively. Messages transfer as follows:

1) PE0 writes the message packet in its common memory shared with NC0,

2) NC0 transfers the message packet to the common memory shared with the central controller,

3) the central controller performs another memory block transfer to shift the message packet to the common memory shared with NC7, and

4) NC7 performs a final memory transfer to place the message packet in the memory space of the destination node.

The **SSS** bits in the semaphore/address byte as shown in Figure 7b indicate the address of the EH-node. The network controllers in each EH-node keep track of the busy status of the individual PEs, and the central controller keeps track of the busy status of the network controllers.

Advantages of the scheme

The dual-ported RAM scheme presents a cost-effective method for data transfer between processor nodes in a multiprocessing system. Tuazon et al. discussed a scheme that yields a data transfer rate of 1.5 Mbits/second. Hayes et al. discussed another scheme using DMAs and serial channels with a transfer rate of 1 Mbyte/s. During DMA transfers, though, the processor must remain idle until the DMA transfers complete. Software overhead may further reduce the effective data transfer rate. The CSMA/CD scheme employed by the iPSC cube offers a data transfer rate of 10 Mbits/s, but has overhead related to the special communication coprocessor and its related initialization and control software.

In our scheme, any pair of processors of a hypercube can establish two-way communication. A processor can receive messages from three of its neighbors and

Table 1.
Data transfer rates for different processors.

Processor	Clock (MHz)	Key instruction	Overheads (incl. states)	Typical transfer rates (Mbytes/s)
8088	10	REP MOVS	33	0.58
8086	10	REP MOVS	33	1.17
80286*	10	REP MOVS	19	5.00
80386*	16	REP MOVS	21	6.00
68000	12.5	MOVE.l (a1 +,(a2)+	50	1.78

*In **real-address mode**

Table 2.
Instructions for data transfer.

Instruction	Clock states required
MOV AX, DATA–SEG	4
MOV DS, **AX**	2
MOV AX, EXTRA–SEG	4
MOV **ES, AX**	2
MOV CX, LENGTH–OF–PACKET	4
MOV SI, SOURCE–POINTER	4
MOV DI, DESTN_POINTER	**4**
REP MOVS	9 + 17(*n*)
Total no. of clock states	33 + 17(*n*)

n **is** ***the number of byte transfers.***

the network controller and send a message to one of its three neighbors or the network controller simultaneously. In other words, at a given time four communication paths of a PE can be active. One of these four can be a two-way communication path. In an 8-node cube with **20** memory units, nine paths can be active at any given time. For example, in Figure 3b PE5 can receive messages from three of its neighbors (PE1, PE4, PE7) and the network controller. PE5 can send a message to one of its neighbors, while other PEs (PE0, PE2, PE3, PE6) can have four active communication paths among them.

We implemented the dual-ported RAM scheme with Intel 8088s as node processors because of the availability of hardware/software development tools and the hardware's low cost. The high-speed communication technique has advantages in a multiprocessing system. The technique can be adopted for communication in multiprocessing systems based on advanced microprocessors like Intel's iAPX 80286 and 80386 and Motorola's 68000,68020, and 68030. Table 1 lists expected zero-wait-state data transfer rates when using typical instructions. This transfer rate is dependent on the bus bandwidth and the type of instructions available. With an 8088 processor operating at 10 MHz, we obtained a zero-wait-state transfer rate of 0.588 Mbytes/s (4.7 Mbits/s) in both directions (duplex). This transfer rate from one PE to another is computed as shown in Table 2. The REP MOVS string operation essentially achieves a block move of data from one part of memory to another.

The software overhead for initializing the various registers involves 33 clock states. The transfer rate actually depends on the REP MOVS instruction, which takes 17 clock states per transfer of a byte in the case of the 8088. This, when computed for a processor operating at 10 MHz, yields 0.588 Mbytes/s or 4.7 Mbits/s. The speed improves significantly if processors with wider data bus widths and higher clock frequencies are used.

Our fully duplexed, asynchronous, and zero-buffered communication scheme handles messages that are less than the maximum allowable packet size. Processor nodes operating at different speeds and different word lengths could be combined in the same multiprocessor system. The highly optimized dual-port technique allows the same memory to be used as working storage and for communication between nodes, avoiding the need for any special data communication controller. Message transfer is transparent to the user programs running on the nodes because no special communication channel must be set up and no need exists to keep track of packet sequence. We further reduce software overhead in that we do not need acknowledgment packets for memory-to-memory transfer. Advanced processors with higher addressing capability can support communication channels with larger sizes of dual-ported common memory and hence improve the throughput.

In an n-cube configuration each individual node connects to n neighboring nodes.[5] **A** message transfer operation between any two nodes with a Hamming distance of n involves $(n - 1)$ processor elements and transmission on n links. We discussed the implementation aspects of our scheme on an eight-node (2^3) cube and the extension of the scheme to a 64-node cube (2^6 cube) configuration. With the help of the dual-port technique and the use of the network controllers, a message can transfer between any two nodes in a 2^3 cube with a maximum of two memory transfers, even if the Hamming distance between the nodes is three. In the 64-node architecture, we achieve a message transfer between any two communicating nodes with **a** maximum of three memory transfer operations. **A** message can also be transmitted from a source node to a destination node through one of the several parallel paths consisting of the PEs, network controllers, and the central controller, depending on their availability.

The dual-ported RAM approach for message transfer between nodes. in a multiprocessor system offers cost and speed advantages. The extended hypercube is an example of a low-cost, compact multiprocessor system with minimal software and hardware overheads. With an 8088 processor operating at 10 MHz, we have achieved a data transfer rate of 0.588 Mbytes per second (4.7 Mbits per second).

Acknowledgments

We wish to thank R. Govindarajan for valuable discussions concerning our work. The suggestions provided by the reviewers and the editor have helped us improve the article's organization.

References

1. S. Yalamanchili and J.K. Aggarwal, "A Characterization and Analysis of Parallel Processor Interconnection Networks," *IEEE Trans. Computers,* Vol. C-36, No. 6, June 1987, pp. 680-691.
2. J. Tuazon, J. Peterson, M. Prifet, and D. Liberman, "Caltech/JPL Mark II Hypercube Concurrent Processor," *Proc. Int'l Conf. Parallel Processing,* IEEE *CS* Press (microfiche), Los Alamitos, Calif., Aug. 1985, pp. 666-678.
3. J.P. Hayes et al. "A Microprocessor-based Hypercube Supercomputer," *IEEE Micro,* Vol. 6, No. 5, Oct. 1986, pp. 6-17.
4. E.I. Organick, "Algorithms, Concurrent Processor, and Computer Science Education," *ACM SIGse Bulletin,* ACM, New York, Vol. 17, No. 1, Mar. 1985.
5. Y. Saad and M.H. Schultz, "Topological Properties of Hypercubes," Research Report, Dept. of Computer Science, Yale University, New Haven, Conn., June 1985.

Additional reading

Hayes, J.P., et al., "Hypercube Computer Research at the University of Michigan," *Proc. Second Conf. Hypercube Multiprocessors,* Univ. of Michigan, Ann Arbor, Sept.-Oct. 1986.

iAPX 286 User's Manual, Intel Corp., Santa Clara, Calif., 1984.

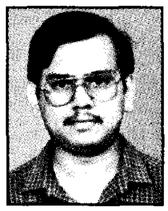
iAPX 80386 *Hardware Reference Manual,* Intel Corp., 1987.

iAPX 86/88 User's Manual, Intel Corp., 1983.

Osborne, A., and G. Kane, *Osborne 16-Bit Microprocessor Handbook,* McGraw-Hill, Berkeley, Calif., 1981.

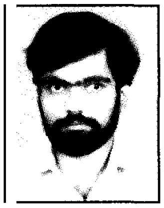
Peterson, J.C., et al., "Mark III Hypercube-Ensemble Concurrent Computer," *Proc. Int'l Conf. Parallel Processing,* IEEE *CS* Press (microfiche), 1985, pp. 71-73.

Seitz, C.L., "The Cosmic Cube," *Comm. ACM,* Vol. 28, No. 1, Jan. 1985, pp. 22-33.



N. Jagadish is a scientific assistant in the Department of Computer Science and Automation at the Indian Institute of Science in Bangalore. Previously, he was involved in building a part of the processor-based controls for motion of a 90-inch optical telescope at the Indian Institute of Astrophysics, also in Bangalore. His areas of interest include local area networks, performance evaluation, and multiprocessor systems.

Jagadish received the BE degree in electronics and communication from the Mysore University in Mysore, India. Currently, he is working toward the MSc (engineering) degree in computer science at the Indian Institute of Science.



J. Mohan Kumar is a scientific officer in the Microprocessor Applications Laboratory at the Indian Institute of Science and has been a lecturer at Bangalore University. His research interests include multiprocessor systems, neural networks, and parallel computing.

Mohan received the BE degree in electrical engineering from Bangalore University and the MTech degree from Indian Institute of Science, where he is working toward his PhD degree in computer science.

L.M. Patnaik, a professor of the Department of Computer Science and Automation, also chairs the Microprocessor Applications Laboratory at the Indian Institute of Science. His teaching, research. and development interests have been in the areas of parallel and distributed computing, computer architecture, computer graphics, computer-aided design of VLSI systems, and expert systems.

Patnaik obtained his PhD in the area of real-time software for industrial automation. He also holds the DSc degree for his research work in computer systems and architectures. He has published over 150 publications in refereed international journals and conference proceedings and coauthored a book on functional programming. He is a senior member of the IEEE and the Computer Society of India, a founder member of the executive committee of the Association for Advancement of Fault-Tolerant and Autonomous Systems, and a fellow of the Indian Academy of Sciences, National Academy of Sciences, and the Institution of Electronics and Telecommunications Engineers in India.

Questions concerning this article can be directed to L.M. Patnaik, Indian Institute of Science, Department of Computer Science and Automation, Bangalore 560 012, India.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 150 **Medium** 151 **High** 152