High Performance Hypercube Communications

Gregory Buzzard and Trevor Mudge*

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109-2110

Abstract

In this paper we present a high performance message transport scheme that is based upon packet routing. The advantages of our scheme are derived from two observations. The first is that given sufficient support from communications hardware, decreasing the message latency time can be much more important to total program execution time than increasing the bandwidth. The second observation is that, from the standpoint of message latency, existing routing strategies tend not to make productive use of much of their available link capacity. The scheme that we propose addresses these issues and leads to substantial performance improvements across a broad range of message loads.

1 Introduction

Hypercube multiprocessors are scalable to very large numbers of processors, in part, because they avoid the memory contention problems that limit the size of shared memory machines. Instead of communicating through a shared memory, hypercubes pass messages between distributed memories over serial links. For algorithms in which there is a high degree of data sharing this mode of communication can also become a limitation. Techniques to achieve fast communications are essential in the application of hypercubes to problems with high degrees of data sharing.

This paper begins by describing the communication system requirements of parallel software. We then offer a set of communication instructions that both implement these requirements and that are suitable for efficient architectural implementations. Next, we summarize the results of an investigation that we have made into the usefulness of various general design alternatives for message transport mechanisms. Based upon this summary and the software requirements, two schemes are selected for further investigation; one is wormhole routing, the second is our packet-based scheme. The drawbacks of the wormhole scheme are then pointed out, and an initial justification for our scheme is given. We then develop our scheme further. Finally, simulation results for the two schemes are presented and discussed.

*This work was supported in part by Department of Defense grant number DOD-MDA904-87-C-4136

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specfic permission.

2 Software Issues

In this section we identify three requirements of parallel software that we would like to see met by the underlying communications architecture. The first of these is the support for arbitrary length messages. This is a basic requirement of most communication systems. At the level of the application program one program statement should suffice to send a message of any size to any destination node. If support for such messages is not provided by the architecture, system software must take the responsibility for providing this appearance to the application program. Several time consuming context switches would then be required to reassemble the message. Thus it is desirable that support for arbitrarily large messages be provided by the architecture.

The second requirement is that communication operations be handled with a minimal number of user executable machine instructions. Typically such operations are performed in an operating system call. The motivations for performing operations within the context of operating system calls are to provide security and to facilitate the sharing of resources. Neither security nor sharing are requirements of single user operating systems. If multitasking is demonstrated to be cost effective in a hypercube environment the communication instruction semantics that we offer below can still handle multiple simultaneous communications without direct operating system intervention. The architectural support for multitasking would consist of informing the operating system at the end of message send and receive operations so that task scheduling operations may be performed, if they are necessary. In essence, we would like to have communication operations treated like co-processor floating point instructions. This is motivated by a desire to eliminate the context switch and call/return overheads for communications that are incurred on existing systems.

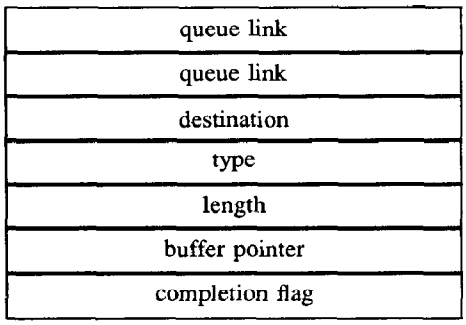

Figure 1: Send header block.

Finally, we would like to be able to use message data as it arrives, rather than having to wait until the entire message is received. Many parallel algorithms access message data in sequential order. For such algorithms, especially those that receive large messages, this allows us to further overlap communications with computations. The major design consequence of this scheme is that the communication system need only maintain a point-to-point bandwidth that is equal or greater to the consumption rate of data. Once this bandwidth requirement is met, the remaining design decisions can be made to minimize message latency. Clearly, though, for such a scheme to be useable we must have a mechanism to prevent the accessing of data before it has arrived.

2.1 Communication Instruction Semantics

The instruction semantics that we propose below assume a hypercube of 2^n nodes; at each node there is a node CPU, a node memory, and a communication co-processor. The instruction set architecture of the communication co-processor is intended to provide a basic efficient interface. This interface should be useable with a minimum of compiler and library support.

Send. The send instruction requires a pointer to a message header block in which the user specifies message destination and length values, a pointer to the message buffer, and an optional message type value. This block should also contain space for a completion flag and two queue links for use by the communication processor. If the communication processor cannot immediately handle the request, the message header block is enqueued in a wait list for later processing. The completion flag is set to 1 once the message send is in progress, and to 3 once the send is complete. A diagram of the message header block for the send instruction is shown in Fig. 1.

Broadcast. Broadcasts occur frequently in numeric codes. Several efficient algorithms are known [HJ86,SWar]. However, none have yet been incorporated into hypercube hardware designs. An integrated architectural solution would eliminate the need to involve node CPUs in the dissemination of the broadcast message. It would also lessen the bandwidth requirement between the communication network and the node memories because the need to forward the message from the node memory is eliminated. The fields within the message header block and the completion flag actions for the broadcast are similar to those for the send instruction, except that a destination field is no longer required. Message transmission will proceed in approximate synchrony as outbound ports become available. This synchrony arises from the fact that the transfer from node memory to communication chip occurs only once, regardless of the number of communication processor output ports that the message is sent through. Broadcast messages will have specially tagged headers so that later communication chips will be able to recognize the potential need to forward the packet through more than one output port. A diagram of the message header block for the broadcast instruction is shown in Fig. 2.

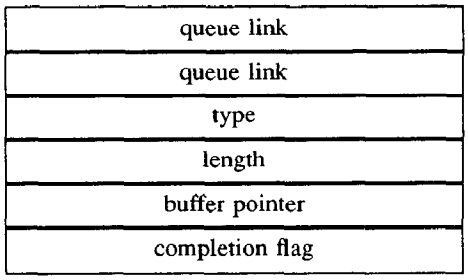

Figure 2: Broadcast header block.

The special case of broadcasts to near-neighbors will be handled with a separate instruction. Near-neighbors are nodes that lie one link away from each other. The number of near-neighbors of a node is equal to n, the degree of the cube. For this instruction we require a list of near-neighbor destinations and a value indicating the number of entries in the list, rather than the implicit destinations assumed for the cube-wide broadcast instruction described above. The remainder of the instruction arguments and the completion flag actions are similar to the send instruction. Message transmission will, again, proceed in approximate synchrony as described above for the general broadcast case. A diagram of the message header block for the near-neighbor broadcast instruction is shown in Fig. 3.

Receive. The receive instruction also requires a pointer to a message header block. In this block the user specifies message source and type values (either of which may assume the value *any*), the length of the message buffer, and a pointer to the message buffer. This block should also reserve space for the completion flag to be written by the communication processor. A null message pointer indicates that the user would like to use a system allocated message buffer. This is useful in a few cases: when the user suspects that the message may have already begun to arrive and does not want to incur a block move overhead; when the user does not want to be burdened with managing message buffers; and when the length of the incoming message is unknown. In such cases, the communication processor writes the address of the message buffer and

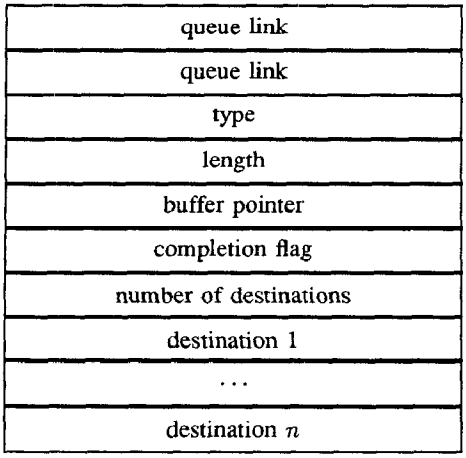

Figure 3: Near neighbor broadcast header block.

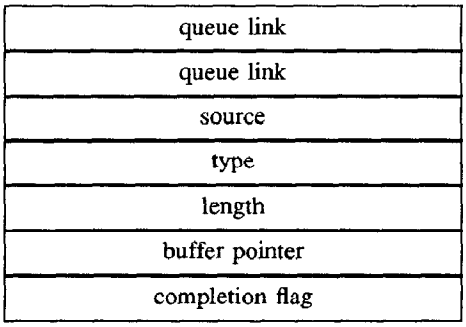

Figure 4: Receive header block.

its length into the message header block, and the user assumes responsibility for returning the buffer space, via the buffer release instruction, to the system. When a valid message buffer pointer is provided the specified message buffer will be used. If the message has already begun to arrive it will be copied from the system buffer to the user specified buffer. The completion flag is set to 1 once the message receive is in progress, and to 3 once the reception is complete.

The completion flag together with the message overrun exception mechanism allow the user to begin processing arriving data before the message reception is complete. This is accomplished by waiting for the completion flag to indicate receive-in-progress before starting to process the data. If a memory request is made before the data arrives an exception will occur. This request may be continually retried until the data arrives. The more conservative user may just wait on the message completion flag before processing the message. The flags are specified such that a process waiting for the receive-in-progress indication may also be released by the reception-completed flag. A diagram of the message header block for the receive instruction is shown in Fig. 4.

Buffer Release. System buffer space is returned via the buffer release instruction. The instruction requires only a single operand that points to the buffer.

3 General Message Transport Issues

In this section we present the results of a general investigation that we have made into design alternatives for message transport mechanisms. The base cases in our comparison are datagram[1], datagram with cut-through [KK79], and circuit switching. For a further discussion of message transport mechanisms see [RF87]. The specific protocol that we have simulated for the datagram with cut-through case, in which the forwarding of a message may begin before its storage at an intermediate node has completed, is analogous to the one described in [ABG85]. Our implementation of circuit switching is persistent. That is, blocked circuits do not surrender their resources and try again. Messages wait until the blocking condition disappears. In this sense it is similar to wormhole routing [Dal86]. With adaptive routing schemes such behavior can lead to deadlock. Several deadlock-free routing schemes exist for store-and-forward transport schemes [Gel81,Gun81,MS80,TU79,Tou80]. A deadlock-free routing scheme for k-ary n-cubes employing wormhole routing has also been developed [DS87]. For our simulation we break potential deadlocks by assigning priorities to messages in order of their creation and requiring that a message never be allowed to block another message of higher priority (earlier creation) unless it is in the process of transmitting. Though there are several methods for avoiding or breaking deadlocks, we feel that this scheme should yield simulation results that provide a fair basis for comparison. We have also considered the additional effects of adaptive routing and independent simultaneous bidirectional communications. Hypercube communication protocols typically employ some form of handshaking that requires acknowledgement information to be passed from a message destination back to its source. Since the original sending node may not continue to process messages on a given link until any expected acknowledgements are received, true bidirectional communications may be difficult to achieve if the acknowledgement information is blocked behind another message on the return link. For further discussion on this topic see [MBA87]. By allowing independent simultaneous bidirectional communications in our simulation model we can evaluate the benefit of removing acknowledgement or confirmation messages from the link that runs in the opposite direction of the the original message. Such a scheme may be implemented at the expense of additional circuitry at the I/O pads of a custom chip. Alternatively, Intel circumvents this problem by interspersing small amounts of control information, which may include acknowledgements, into the data stream of the link that runs in the opposite direction of the original message.

When considered by themselves, all three of the transport schemes that we have evaluated have significant drawbacks.

[1] In this context datagram refers to a store-and-forward based transport scheme which stores and forwards messages in their entirety, i.e., the messages are not split into smaller packets

However, the primary purpose of this particular investigation was to determine the impact of various design decisions, not to advocate the adoption of any specific case. Nevertheless, the following general conclusions can be drawn. The inability of circuit switching to gracefully degrade in performance under heavy traffic loads renders it a poor choice for any environments where moderate to heavy bursts of traffic are likely to occur. Datagrams, both with and without cut-through, require buffers that are many times larger than the largest message to maintain reasonable performance. Maximum buffer requirements, without flow control, can quickly become excessive: ranging from about 100 K-bytes for light loads to in excess of 200 K-bytes for heavier traffic for messages averaging 8192 bytes in length. Clearly, for any store-and-forward based transport scheme to be viable it must employ flow control. Even with flow control, however, datagram based schemes require buffers at least as large as the largest allowable message size at each node. The advantages of cut-through are significant for all but the heaviest traffic conditions. Under moderate to heavy traffic conditions the most significant performance improvement, for both of the datagram cases, as well as for circuit switching, is gained by providing non-interfering bidirectional links. For datagrams, adaptive routing was also helpful, particularly in conjunction with non-interfering bidirectional links, for all but heavy traffic cases with small messages. In these cases, adaptive routing lead to performance decreases which were particularly significant for the uni-link case. The benefits of adaptive routing were typically much smaller than those of bidirectional links for all but the light traffic cases. Adaptive routing, in conjunction with circuit switching, showed similar results, except that performance decreases occurred for small messages at all traffic levels.

Message transport tradeoffs are also being studied by Reed and Grunwald [GR88]. Their work includes evaluating the different routing mechanisms that are possible with the JPL Hyperswitch in a JPL Mark III based computer. The general observations from their preliminary results appear to coincide with ours. In particular, for moderate to light traffic loads they show the best results for wormhole and an adaptive form of circuit routing known as K(K-1). In fact, they show very little performance difference between the two schemes.

4 Cases for Further Evaluation

In this section we develop a selection criteria and choose the message transport schemes that we will develop and evaluate further. Message transport performance for random communications is improved most by providing non-interfering bidirectional communications in moderate to heavy traffic and by avoiding the store-and-forward overhead in light traffic. The former can be provided for any of the transport schemes that we are considering by the underlying architecture. The latter improvement calls for considering a transport strategy based upon some variant of circuit switching, or datagram with cut-through. The buffer requirements of cut-through, however, conflict with the desire to support arbitrary length messages. Packet switching, in which individual messages are broken into fixed sized packets, provides a reasonable solution to this conflict by limiting the buffering requirements while maintaining the ability to handle arbitrarily large messages by using a similarly large number of packets. In order to preserve the ordering of packets within a message all packets must follow the same route from source to destination. Alternatively, one could attach sequence numbers to each packet and reassemble them in the correct order at the destination. Reassembly, however, is a very expensive operation whose cost cannot be justified in this context, particularly since we would like to allow calculations at the beginning of a long packet to overlap its arrival. This limits the application of adaptive routing for packet based transport mechanisms. Although, as we will show later, this limitation is not unduly restrictive. The overlapping of communications with computations on arriving messages is a task made easier when messages arrive in a contiguous non-interleaved fashion on a particular channel. However, with the communication semantics that we specified in Sec. 2, the interleaving of messages on a common channel may be accommodated at the expense of a slightly more complex architecture.

We will further investigate two schemes. The first is wormhole routing, a variant of fixed route circuit switching. This is one of the routing strategies available with the JPL Hyperswitch. Wormhole transports generally compare favorably with other proposed schemes. However, they have two drawbacks. One is their relatively poor performance for small messages in heavy traffic. We expect that for several algorithms a large proportion of messages under conditions of heavy traffic will be of relatively short length. Certainly, most request messages in algorithms that use a request/response communication paradigm will be short. Also, there are many amorphously structured algorithms, such as chess [FMO*87] or programs incorporating branch-and-bound techniques [AM88,AC87,Qui87,WLY85], that would like to make a random accesses to small amounts of global data. The second drawback is that the presence of a long message can effectively delay the delivery of other messages.

A simple example of the second drawback is illustrated in Fig. 5 where, for two messages starting at the same time, the circuit for the larger message gets established first, forcing the shorter message to wait until the larger message completes for its circuit to be established. At a high level, we can view the message transmission time of a circuit or cut-through based transport scheme as being proportional to $M + h\beta$, where M is the length of the message, h is the number of hops from source to destination, and β reflects the speed with which the communication links can be acquired. The magnitude of β is dependent upon the traffic load (i.e., number and size of active messages), and the availability of communication resources—in this case, links. Alternatively, if we could break messages into packets, then interleave the packets of the two messages, transmission time would be proportional to $\frac{M}{p}(p+h\alpha)$. Where α reflects the apparently reduced bandwidth of the link as viewed from the message level due to the multiplexing and additional overhead of packet headers, and p is the size of the packets. This, of course, can be rewritten as $M + \frac{M}{p}h\alpha$. When we take into account the fact that the message packets are pipelined through the network, the multiplicative effect of

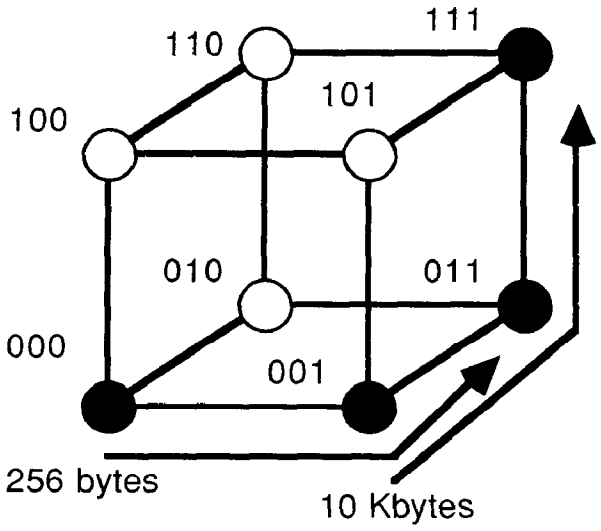

Figure 5: Wormhole Routing Blockage.

the number of packets ($\frac{M}{p}$) on $h\alpha$ is reduced to one. Thus, the better performing transport scheme will depend on which of α or β is lower.

Returning to our simple example, we see that with wormhole routing the transmission times for the long (t_l) and short (t_s) messages are:

$$t_l = M_l + h$$

$$t_s = M_s + ht_l$$

And with a packet based scheme:

$$t_l = M_l + h\alpha_l \qquad \text{where, } \alpha_l \approx 1$$

$$t_s = M_s + h\alpha_s \qquad \text{where, } \alpha_s \approx 2$$

The effect of α_l is negligible due to the size of M_l relative to M_s, and α_s is slightly larger than 2 to account for the additional overhead of packet headers. In general, the outlook is quite not this bleak for wormhole transports. However, it is important to note that while α tends to be, at worst, linearly proportional to the average number of packets vying for a link, β suffers additionally from the effect of link starvation. This occurs as a result of messages being blocked after having acquired, possibly several, links. Thus, in addition to waiting on a given link, the waiters also hold out of service all of the links that they have already acquired. Similar effects with packet buffers, in the case of packet based transports can be easily avoided with a modest number of buffers. These effects can be seen in the results reported in Sec. 6.

The second scheme that we will consider employs a packet based transport scheme that allows packets from different messages to be interleaved. It uses a combination of fixed and adaptive routing in which the first packet may adaptively select its first routing step in any direction that will take closer to its destination. All subsequent packets from the message are routed in the same initial direction. All routing decisions, other than the first, are determined by a fixed route. Thus, packets from the same message will always arrive in order at their destination. There are two primary costs associated with this scheme: 1) each packet must now carry its source and destination node numbers, thus lowering the effective bandwidth for message data as discussed above; and 2) the architectural scheme for detecting if the node CPU requests part of a message before it arrives is more complicated than the comparable implementation for wormhole routing. Implementation restrictions limit the number of concurrently arriving messages that a communication processor can check. These checks can be made by having multiple pairs of comparators monitor the address bus for addresses that fall between the address of the most recent byte received and address of the last byte expected for messages that are actively being received. Since wormhole messages do not interleave on individual channels, the maximum number of concurrently arriving wormhole messages is at most n, the degree of the cube. Thus, potential overruns can can be detected for all arriving wormhole messages with n pairs of comparators. For packet routing, which can concurrently receive a far greater number of packets, we would have to maintain a *cache* of addresses that track the progress of the first n messages that arrive concurrently. This cache then feeds pairs of comparators that check these addresses against the node CPU memory requests that appear on the node address bus. Messages that have their arrival tracked by entries in the *comparator cache* have their completion flags marked to indicate that message arrival has begun and that monitoring for overruns will be performed. Upon completion of message arrival their completion flags are marked to indicate reception-completed. Any message that begins to arrive while the comparator cache is full will have its completion flag marked only upon completion of message arrival, thus indicating that overrun checking is not being performed for this message. While not all arriving messages may be able to be processed concurrently with their reception, in practice this should not be a problem since the node CPU will still have several arriving messages with which to overlap computations. Further, the efficiency of the cache mechanism could be improved by adding a parameter to the message header block of the receive instruction to indicate whether or not overrun checking is desired for the awaited message.

5 Packet Transport Issues

In this section we develop further the packet transport scheme that we introduced in the preceding section.

5.1 Sequential Bottleneck at Source

Originally, we used entirely fixed routing for all packets. However, initial studies revealed that the performance advantage of the packet based transport for heavy message traffic was quickly lost as the average message size was increased. Closer inspection determined that messages were blocking in sequential order on their source nodes, just as they do for the wormhole transport. When average message times were computed from the time messages began to leave their source nodes the performance advantage returned. In fact, the rela-

tive advantage was even greater than we had previously noted because the same effect had been occurring with the shorter messages as well.

One approach to breaking this sequential ordering is to interleave the packets of messages that are waiting to be transmitted from their source nodes. However, this is impractical to implement in a fair and efficient manner, particularly when the implementation is constrained by the fixed resources available on a single communication chip. It is difficult to determine the number of distinct messages that are represented by packets presently in the queue and the location of their head packets. Another factor that limits the speed with which messages can be moved off of their source node is that fixed routing schemes direct messages to half of the nodes in the cube out of a single channel, messages to half of the remaining nodes go out the next channel, etc. Under conditions of constant uniform traffic, of course, this scheme makes optimal use of the links by evenly distributing the message load from all nodes across all links. However, in reality we expect message traffic to be bursty and chaotic. This can lead to excessive link contention for the most popular link by messages that are attempting to leave their source nodes. As a simple example, with the all other communications quiescent, a pair of simultaneously generated messages on any given node in a cube of dimension n will collide while attempting to acquire their respective initial links with a probability given by:

$$P_f = \left(\frac{2^n}{2^n - 1}\right)^2 \sum_{k=1}^{n} \frac{1}{2^{2k}} \qquad (1)$$

The $\left(\frac{2^n}{2^n-1}\right)^2$ term accounts for the fact that a node will not send a message to itself. The term in the summation is derived from the $\frac{1}{2^k}$ chance, for each message, that the kth link is chosen.

Rather than interleaving the message with packets that have been queued for transmission, we have chosen to adaptively select the first routing step for the message based upon the shortest queue in a direction that takes the message closer to its destination. All packets of the message still follow the same route, however, the first step in the route is no longer fixed for the first packet. In this case, with all the other communications quiescent, a pair of simultaneously generated messages on any given node in a cube of dimension n will collide while attempting to acquire their respective initial links with a probability given by:

$$P_a = \left(\frac{2^n}{2^n - 1}\right)^2 \sum_{k=1}^{n} \left(\frac{1}{2^k}\right)\left(\frac{1}{2^n - 1}\right) \qquad (2)$$

In this case, there is a $\frac{1}{2^k}$ chance that the first message chooses the kth link, and a $\frac{1}{2^n-1}$ chance that the second message cannot use any of the remaining links. Comparing (1) and (2), we can show that $\lim_{n\to\infty} P_f = \frac{1}{3}$ and $\lim_{n\to\infty} P_a = 0$, also $P_f \gg P_a$ for all $n \geq 2$, therefore we conclude that our adaptive packet transport scheme significantly reduces the problem of messages colliding prior to entering the communication network.

5.2 Deadlock Avoidance

By relaxing the fixed routing requirement we have introduced the possibility of deadlock. Let us refer to packets that are taking their first routing step in a direction that differs from the direction that they would take in the fixed routing scheme as *contrary* packets. Once a contrary packet has left its source node it will be routed along a fixed path, thus we will no longer consider it to be contrary. To avoid deadlock, we require that at least one buffer slot for each channel in every node be either free, or occupied with a *non-contrary* packet at any given time. This guarantees that all non-contrary packets will eventually progress to their destinations since fixed routing assures that there are no cyclical resource dependencies. Contrary packets exist for at most one routing step, and for any cycle through a cube at least two nodes will not issue contrary packets. This is assured because at least two messages (from nodes opposite each other in the cycle) will take their first routing step in each of the dimensions (or routing *directions*) represented in the cycle. This being the case, at least two messages will have picked the same direction that they would have under fixed routing. Thus, contrary packets cannot form cyclical resource dependencies and deadlock cannot occur.

5.3 Buffer and Packet Sizes

Our packet based scheme dedicates a fixed number of packet buffers to each output port on the communications chip. Since the total amount of packet storage that can be implemented on a communications chip will be limitied, attempts at tuning performance can be made by varying the packet size and the number of buffers while holding the product of the two constant. We have not yet investigated this tradeoff. The simulation results reported below assume packet lengths of 20 bytes with 16 packet buffers per port. The packets contain 4 header bytes (two bytes each for the destination and source) and 16 data bytes. The message length (4 bytes) and type (2 bytes) values must appear in the first packet, thus limiting the effective data length in the first packet to 10 bytes. The required source and destination fields comprise all of the information that is needed to route packets through the network and to the correct location on the destination node. The source field is required by the final node since packets from different sources may be interleaved on a common port into the destination communications chip.

6 Simulation Results

In this section we explain the simulation metrics and parameters, and present and discuss the results for wormhole routing as well as our original fixed packet routing and our quasi-adaptive version of packet routing. The primary performance metric presented is the average elapsed time from the moment a message is queued for sending on its source node until it reaches its destination. We have also individually compared the times for messages of specific lengths and distances. These results correlate well with the average values that are presented. All results are given for two different average message

lengths across a variety of traffic loads. As a further check, we have also compared both the maximum times taken by any message and the total simulation times. Neither of these checks indicate anomalies in any of the simulation cases.

For all cases, message destinations are chosen uniformly. The message lengths are given by an exponential distribution with a mean of 512 bytes for the first set of results and with a mean of 2048 bytes for the second set. The intergeneration time for messages at each node is given by a normal distribution. For messages of each length, data was gathered at nine different rates of message generation. For the shorter messages the mean of the intergeneration time ranges from 1024 ticks to 9216 ticks, with two increases of 256 ticks, followed by three increases of 512 ticks and, finally, three increases of 2048 ticks. The variance is always equal to half of the mean. For the longer messages the mean of the intergeneration time ranges from 4096 ticks to 36846 ticks, with increases of 1024 ticks, 2048 ticks, and 8192 ticks. The link transfer rate is 2 ticks per byte and arbitration for shared resources is assumed to take 4 ticks. These values are derived from our initial thoughts on implementation. The same pseudo-random distributions are generated for the message destination, length, and, intergeneration time for all routing schemes. This ensures that the kth message will be of length l and will travel from source s to destination d starting at time t for all of the routing schemes evaluated. In all cases, simulations are performed for hypercubes of degree 6.

The intergeneration time for messages is expressed as an ideal link utilization value. This value is derived by calculating the total amount of link time required to handle the transfer of all of the messages generated during the simulation. For example, a message of length M that travels L hops will contribute $2ML$ link ticks to the link time total. This time is then divided by the product of the total number of links in the hypercube and the time from the start of the simulation until the last message is generated. This is not a "true" utilization value in the sense that we only intergrate the total available link capacity up until the time of the last message generation. However, this does yield an ideal utilization value that is the same for all transport schemes with the same simulation parameters, which is our primary objective. This is guaranteed because the time of the generation of last message is always the same; whereas, the total time varies somewhat for each different routing mechanism.

The simulation results are given in Figs. 6a through 7b. The figures for each of the two different message lengths are given in two parts with differing vertical scales. By examining the link utilization axis it can be seen that Figs. 6a and 6b overlap by two data points. Similarly, with Figs. 7a and 7b. The mean elapsed times for both the arrival for the first packet (or 16 data bytes) and the last byte are shown. This allows us to see the effect that the packet interleaving has on both latency and bandwidth. The *first* times (e.g., *the line labelled first wormhole*) indicate message latency. The difference between corresponding *first* and *last* times (e.g., *last wormhole* less *first wormhole*) provide an indication of the message bandwidth.

For the smaller messages our initial fixed routing packet scheme leads to message latency times that are about 70% of the wormhole latency times for the 6 highest link utilization values, and 85% of the wormhole latency time for the lightest value. On the other hand, bandwidth for the fixed packet scheme ranges from about 30% of that of the wormhole scheme for the heavier loads to about 70% for the lighter loads. For the larger messages, blockage at the source nodes preclude improvements in latency times over those for wormhole routing. Also, the bandwidth remains consistently poor as compared to the wormhole transport scheme—clearly this is not a desirable situation.

The change to the quasi-adaptive packet scheme leads to significant improvements. Message latency times become less than 20% of the times for wormhole routing for the 2 heaviest loads, and range to about 70% of the wormhole latency time for the lightest load considered. Bandwidth for this scheme ranges from about 20% of that for wormhole at the heaviest traffic load to about 70% at the lightest load. For the four heaviest traffic loads the average time for the messages to completely arrive is approximately equal to, or less than, the average time for them to begin arriving with wormhole routing.

The results are even better for the large messages. Message latency stays between 18% and 25% of that for wormhole routing across the entire range of loads. Bandwidth for the adaptive packet scheme ranges from about 25% of that of wormhole at the heaviest traffic loads to about 65% at the lightest.

The decreases in bandwidth will likely have an insignificant effect on most programs. Consider, that the NCUBE hypercube with on-chip floating point hardware consumes message data at a rate of 0.188 Mbytes per second when performing the double precision vector operation: $\vec{X} = a\vec{X} + \vec{Y}$. Peak message bandwidth on the same system is 0.77 Mbytes per second. Bandwidth would have to decrease by more than a factor of four in any system with a similar calculation to bandwidth ratio before performance in the above operation would begin to be affected. Even then, we still have the advantage of being able to start processing message data much sooner. It is also the case that the difference in bandwidth between wormhole routing and our packet scheme is the least significant in those cases where our improvement in latency times are also least significant. Considering all of the above, it seems reasonable to expect that most programs would be able to reap the benefits of reduced message latency times without incurring any costs from the effectively reduced message bandwidth by using our quasi-adaptive packet based routing scheme.

7 Conclusion

We have developed a new, quasi-adaptive, variation of packet routing and shown that it performs well. Message latency times are consistently improved by factors of up to 5 over the message latency times for wormhole routing. In all cases, the tradeoff in message bandwidth appears negligible. We have also introduced a set of communication instruction semantics that should work well with our "new" routing scheme. In future work we will examine buffer and packet size tradeoffs, and address architectural implementation details.

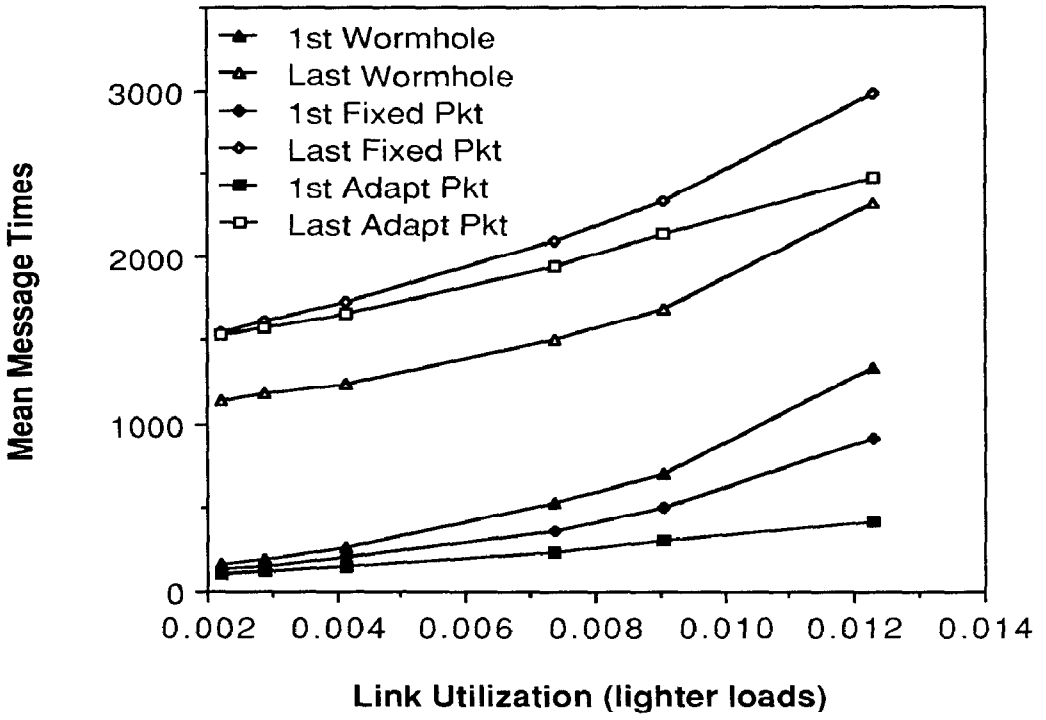

Figure 6a: Mean Message Times, Length = exp(512).

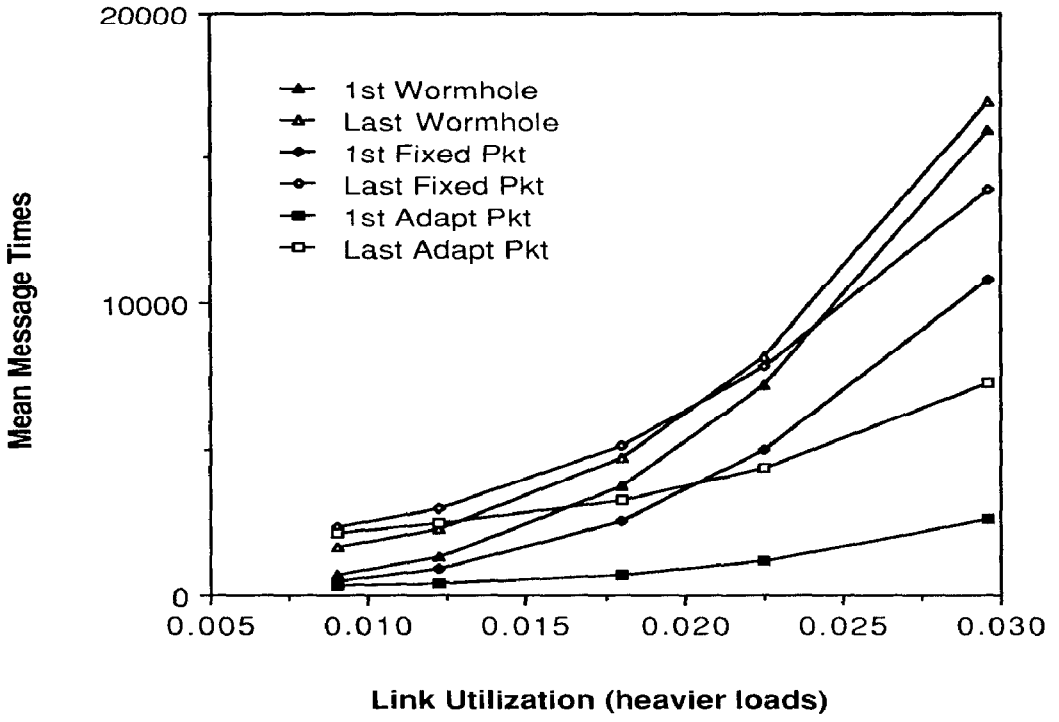

Figure 6b: Mean Message Times, Length = exp(512).

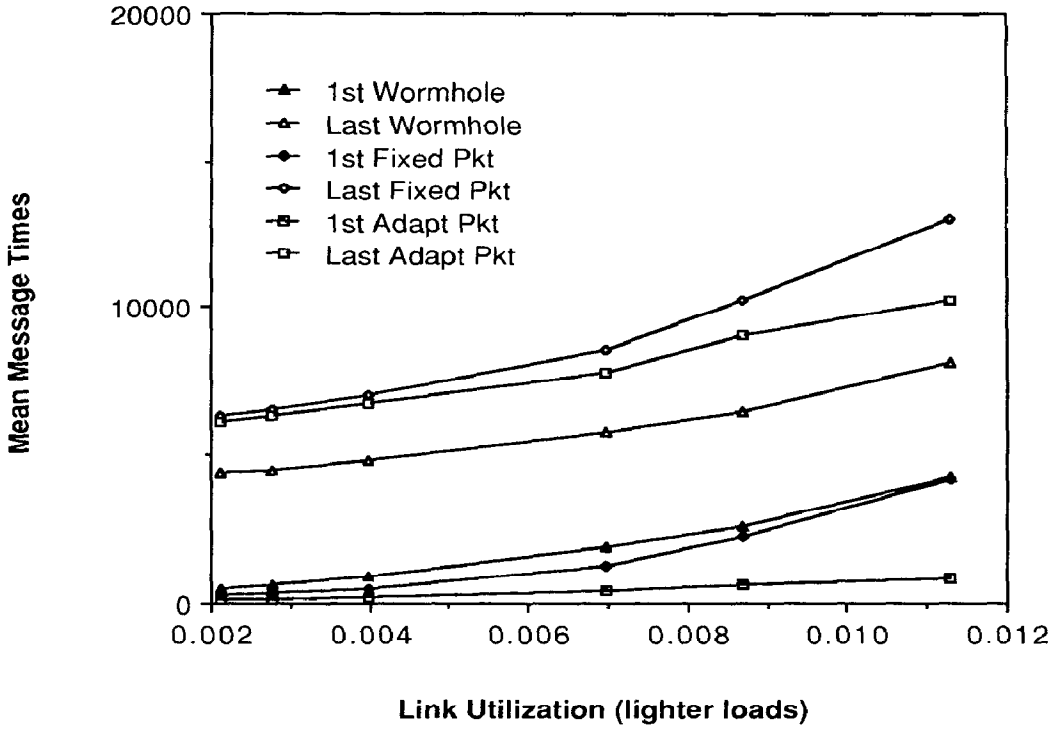

Figure 7a: Mean Message Times, Length = exp(2048).

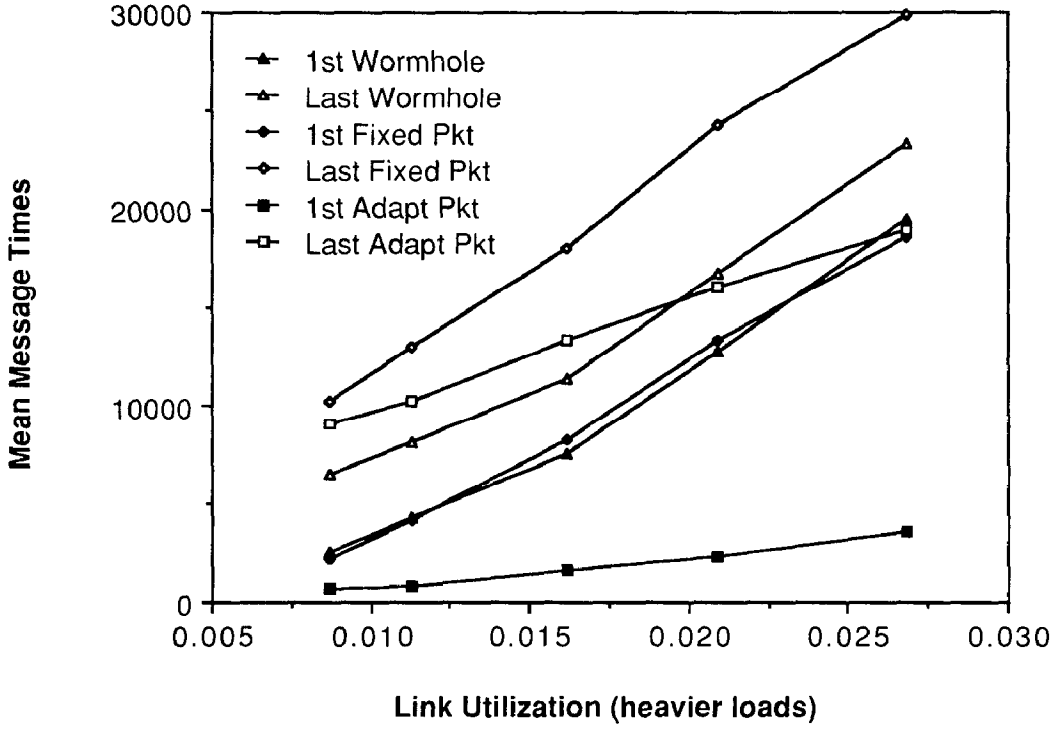

Figure 7b: Mean Message Times, Length = exp(2048).

References

[ABG85] Mauricio Arango, Hussein Badr, and David Gelernter. Staged circuit switching. *IEEE Transactions on Computers*, C-34:174–180, February 1985.

[AC87] Steven Anderson and Marina C. Chen. Parallel branch-and-bound algorithms on the hypercube. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987*, pages 309–317, SIAM, Philadelphia, 1987.

[AM88] Tarek S. Abdel-Rahman and Trevor N. Mudge. Parallel best first branch and bound algorithms on hypercube multiprocessors. In Geoffrey C. Fox, editor, *Proceedings Third Conference on Hypercube Concurrent Computers and Applications*, ACM, 1988.

[Dal86] William J. Dally. *On the Performance of k-ary n-cube Interconnection Networks.* Department of Computer Science Technical Report 5228:TR:86, California Institute of Technology, 1986.

[DS87] William J. Dally and Charles L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36:547–553, May 1987.

[FMO*87] E. Felton, R. Morison, S. Otto, K. Barish, R. Fatland, and F. Ho. Chess on a hypercube. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987*, pages 327–332, SIAM, Philadelphia, 1987.

[Gel81] David Gelernter. A dag-based algorithm for prevention of store-and-forward deadlock in packet networks. *IEEE Transactions on Computers*, C-30:709–715, October 1981.

[GR88] Dirk Grunwald and Daniel Reed. Multiprocessor computer networks: measurements and prognostications. In Geoffrey C. Fox, editor, *Proceedings Third Conference on Hypercube Concurrent Computers and Applications*, ACM, 1988.

[Gun81] K. D. Gunther. Prevention of deadlocks in packet-switched data transport systems. *IEEE Transactions on Communications*, COM-29:512–524, April 1981.

[HJ86] Ching-Tien Ho and Lennart Johnsson. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 640–648, IEEE, 1986.

[KK79] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communication switching technique. In *Computer Networks, Volume 3*, pages 267–286, North-Holland, Amsterdam, 1979.

[MBA87] Trevor N. Mudge, Gregory D. Buzzard, and Tarek S. Abdel-Rahman. A high performance operating system for the ncube. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987*, pages 90–99, SIAM, Philadelphia, 1987.

[MS80] P. M. Merlin and P. J. Schweitzer. Deadlock avoidance in store-and-forward networks–I: store-and-forward deadlock. *IEEE Transactions on Communications*, COM-28:345–354, March 1980.

[Qui87] Michael J. Quinn. Implementing best-first branch-and-bound algorithms on hypercube multicomputers. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987*, pages 318–326, SIAM, Philadelphia, 1987.

[RF87] Daniel A. Reed and Richard M. Fujimoto. *Multicomputer Networks: Message-Based Parallel Processing*, pages 138–144. The MIT Press, 1987.

[SWar] Quentin F. Stout and Bruce A. Wagar. Intensive hypercube communication I: prearranged communication in link-bound machines. *Journal of Parallel and Distributed Computing*, to appear.

[Tou80] S. Toueg. Deadlock-and livelock-free packet switching networks. In *Proceedings of the 12th ACM Symposium on the Theory of Computing*, pages 94–99, 1980.

[TU79] S. Toueg and J. D. Ullman. Deadlock-free packet switching networks. In *Proceedings of the 11th ACM Symposium on the Theory of Computing*, pages 89–98, 1979.

[WLY85] Benjamin W. Wah, Guo-jie Li, and Chee Fen Yu. Multiprocessing of combinatorial search problems. *IEEE Computer*, 93–108, June 1985.