Proc. 4th Conf. Hypercubes, Concurrent Computers & Applications, vol. I, Mar. 1989, pp. 285-291

Short Latency Routing for Hypercube Multiprocessors

Gregory Buzzard[*] and Trevor Mudge[†]

Abstract

This paper presents a new strategy for routing messages between processor nodes in a hypercube multiprocessor. The strategy minimizes the inter-processor latency, that is, the transit time of the start of the message, and still keeps the overall transit time for the entire message low. This strategy depends upon support from the communication system to allow computations on *currently arriving* message data. As a result, message dependent computations in the majority of present-day applications need only delay until the arrival of the first few bytes of message data and the overall performance becomes more dependent on the message latency than on the entire message transit time.

1 Introduction

The performance of computer communication systems can generally be evaluated as a function of two parameters, message latency (t_l) and message bandwidth (BW). Message latency is the amount of time that elapses from the moment that the sending processor executes the message send operation until that message begins to arrive at its destination. Message bandwidth is the rate at which successive bytes of a message arrive at the destination once the first byte has arrived. The abundance of communication links in the hypercube interconnection structure provides a tremendous amount of potential message bandwidth. There are $n2^n$ distinct communication links in a hypercube of degree n. It is the job of the communications system to convert this raw link bandwidth into a usable and efficient message passing system, that is, one that provides low latency as well as ample message bandwidth. Additionally, the communications system should abstract the details of message routing from application programs.

The effective communication bandwidth that is observed by an executing process depends upon the speed and utilization of the communication links. The message latency is dependent upon the availability and utilization of communication links. Communication link speed is fixed by the hardware design. Availability and utilization, however, depend upon the choices of routing and transport strategies.

An obvious figure of merit to use in the evaluation of the performance of these strategies is the total message delivery time ($t_t = t_l + \frac{L}{BW}$, where L is the message length). However, most existing parallel algorithms process message data in a sequential order. Thus, if we allow the processing of message data to proceed as it is arriving, the primary figure of merit for communication performance becomes the latency, or time to begin the delivery of the message. This will be the case as long as the actual message delivery rate meets or exceeds the rate at which the processor consumes the message data. Our message routing proposal is based upon the fact that hypercube systems are capable of delivering messages more quickly than they can be consumed and that we can design a system in which some of this excess bandwidth can be effectively traded for decreases in message latency.

As message traffic increases, the contention for communication links becomes the chief performance bottleneck for efficient message delivery. The partially adaptive routing strategy that we propose reduces such bottlenecks by spreading a given communication load across a larger set of links. Our strategy is unique in that it avoids the deadlocks that can occur in more general adaptive routing schemes[CMP+88], while still selecting from multiple possible communication paths. Additionally, we fragment large message into smaller packets which can then be interleaved with packets from other messages onto a common communication link. Such interleaving, in conjunction with our adaptive routing strategy, leads to large increases in communication performance because the substantial reduction in message latency far outweighs the decrease in effective message bandwidth.

[*]Gregory Buzzard is with the Department of Computer Science, Naval Postgraduate School, Monterey, California 93943-5110

[†]Trevor Mudge is with the Advanced Computer Architecture Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109-2110

Preprint from
Fourth Conference on Hypercube Concurrent Computers and Applications

Transport	Routing	Message Delivery Time
Datagram	Fixed (E-cube)	$t = k(t_{l-dgm} + \frac{L+H}{BW})$
Circuit	Fixed (E-cube)	$t = kt_{l-cct} + \frac{L+H}{BW}$
Circuit	Adapt $K(K-1)$	$t = kt_{l-kk1} + \frac{L+H}{BW}$
Packet	Quasi-adaptive	$t = kt_{l-qap} + \frac{L+pH}{BW}$

Table 1: Approximate Message Delivery Times

The next section of this paper describes common message transport and routing schemes. In Section 3 we discuss our scheme, which we call quasi-adaptive packet routing (QAP). After describing how QAP works, we explain one of the chief principles behind its performance and then discuss its deadlock avoidance characteristic. Simulation studies that compare the performance of QAP with the best of the circuit based schemes are discussed in Section 4.

2 Transport & Routing Schemes

The communication system design space comprises two axes, along the first axis lie the alternatives for message transport schemes. These schemes are classified into two categories store-and-forward and circuit switched[RF87]. Store-and-forward systems can be further divided into three groups: datagram, packet switched and virtual circuit. The alternatives for routing mechanisms lie along the other design axis. At one extreme, message routing can be fixed where the route is a function of only the source and destination nodes. At the other extreme, message routing can be fully adaptive where routing decisions are made at each step of the path from source to destination. Several of the schemes which have been used or proposed are described in[GR88], an approximation of the message delivery times based upon the choices of transport mechanism and routing strategy are given in Table 1. In the equations given in Table 1, k is the distance from message source to destination, t_l is the latency for acquiring a single communication link, L is the length of the message data, H is the length of the message header, BW is the bandwidth of the communication system and p is the number of packets that the message data is divided into.

Both of the schemes that employ adaptive routing techniques have been shown to provide superior performance over there non-adaptive counterparts[BM88, GR88]. These two adaptive schemes employ rather different approaches for both their routing decisions and their transport mechanisms. The $K(K-1)$ scheme is based on a circuit switched transport mechanism with a routing algorithm that will backtrack when necessary to choose among $K(K-1)$ of the $K!$ shortest paths between two nodes. The quasi-adaptive scheme is based on a packet switched mechanism without backtracking that will choose among k paths. The differences between the schemes are primarily reflected in the behavior of t_l terms as the system communication load increases.

In circuit switched systems, lengthy mesages can tie up a number of communication links for a significant amount of time. This effect, which is described in[BM88], can lead to substantial performance losses in systems with moderate to heavy communication loads. Packet switching avoids this problem by allowing packets from different messages to be interleaved on a common communication link. However, there is an additional cost associated with packet switching. Each packet is required to carry information that identifies the packet and its destination. This leads to an effective increase in the total amount of data that must be transmitted through the network to deliver message with L bytes of data from $L+H$ to $L+pH$. Even further costs would be incurred if the packet switching scheme were to allow different packets from the same message to travel via different paths. As a result, all packets from the same message are required to follow the same path in the quasi-adaptive scheme.

Another issue that arises for the adaptive routing schemes is deadlock. Deadlock is implicitly avoided by the *e-cube* routing algorithm[Lan82] that is commonly employed in the fixed routing schemes. However, the cost of avoiding deadlock in these schemes is rather high: only one predetermined path may be used to travel between two nodes. The $K(K-1)$ adaptive routing scheme must detect and resolve deadlocks in order to choose among the $K(K-1)$ paths. One of the key features of the quasi-adaptive packet routing scheme is that it allows an adaptive choice to be made between k different paths but still maintains the deadlock avoidance property of the fixed routing schemes.

3 QAP Routing

Quasi-adaptive packet routing employs a packet based transport mechanism 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 routing algorithm that depends only on the current and destination nodes. 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 communications architecture must now detect if the node CPU requests part of a message before it has arrived. If such an event occurs, the affected process must be suspended until the requested data has arrived, this is somewhat analogous to taking a page fault. An architecture that provides such a capability is described in[Buz88].

The performance of the quasi-adaptive packet routing scheme is, to a large extent, derived from its ability to substantially avoid message bottlenecks that occur frequently near busy source nodes. Such bottlenecks can significantly limit the performance of the communication system. The chief challenge in avoiding this problem is to spread the communications load among the many available links while still avoiding deadlock. While techniques for reducing local congestion away from source nodes have been discussed [Val82], our quasi-adaptive routing scheme provides a unique solution to reducing congestion near a busy source.

The primary factor that limits the speed with which messages can be moved off of their source node in fixed routing schemes is that they 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 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 that takes the message in a direction closer to its destination. All packets of the message still follow the same route; however, the direction of the first step in the route is no longer fixed. 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. We also investigated adaptively routing the message by choosing the first step at random. Both analytical and simulation analyses showed this scheme to be inferior to routing in the direction of the shortest queue.

Any routing scheme that makes adaptive routing decisions needs to address the issue of deadlock. Conversely, fixed routing schemes that preclude the formation of cyclical routing dependencies are inherently deadlock free. For example, circuit based schemes which use the (fixed) *e-cube* routing scheme, such as the Intel iPSC 2 [Nug88], are deadlock free. Store-and-forward schemes, which require buffers in addition to links to deliver messages, must additionally preclude the formation of *buffer-based* cyclical routing dependencies in order to avoid deadlock. The dependency on buffers can be merged into the link dependency by associating a specific buffer with each outbound link on every node. Thus, an idle link will always be able to provide buffer space to any packet that can reach its node.

While the use of fixed routing schemes can preclude deadlock, they eliminate all choice in the formation of a path between source and destination nodes. Consequently, they are unable to adapt to changing system communication loads. Our quasi-adaptive routing scheme allows any of k (where k is the distance between the source and destination) paths to be chosen when the packet enters the communication system. One of the key features of QAP is that it allows adaptive routing decisions while avoiding deadlock. Cyclical buffer dependencies are avoided by associating a specific buffer with each outbound link on every node. The routing in QAP follows the deadlock avoiding *e-cube* scheme on all routing steps after the first. Thus, any routing cycles that may occur will be caused by a packet that is leaving its source node. The only resources held by a packet in QAP are the buffer that it occupies and, possibly, a link to the next node. QAP packets that are on their source node do not occupy buffer space in the communications switch. Thus, the only communication resources that they may posses are links to adjacent nodes. We can ensure that deadlock will never occur by allowing any packet that is buffered in a communication switch (i.e., one that is not on its first routing step) to reappropriate a required link from a packet that is on its first step. This reappropriation may occur whenever the link is not actively transmitting a packet, a link that is transmitting is guaranteed to finish in a short time.

4 Simulation Comparisons

Extensive simulations were run for the K, $K(K-1)$ and QAP routing schemes. The performance of $K(K-1)$ routing is superior to that of K routing, therefore, results are shown for only the $K(K-1)$ simulations. The timing parameters for the K-family simulations were taken from [CMP+88]. The time values recorded by the simulator include the time to determine the next routing step and acquire a communication link for each step along the path and the time for the transmission of data once the full circuit has been established. The overhead that is initially incurred by the transmitting processor is not counted in the simulation time. Similarly, the overhead incurred by the receiving processor is ignored. Since the K-family routing mechanisms are not persistent (that is, they do not block waiting for links to become available) they may acquire several links toward the destination before arriving at a node at which none of the necessary links are currently available. Some of the previously acquired links must then be dropped in order to backtrack and try another of the K or $K(K-1)$ routes. We assume that the K-family communication protocol allows these links to be dropped instantaneously upon the receipt of a *link drop* control signal, thus no time is added for backtracking. It may also be the case that it is not possible to acquire all of the links necessary to construct a full path from source to destination in a single search of the K or $K(K-1)$ possible routes. If this happens, all of the acquired links are released and the sending processor is interrupted. For the purposes of our simulation, the sending processor immediately attempts to resend the message. These complete retry actions do incur the cost of one reception and one transmission overhead.

The time values recorded by the simulator for the QAP routing scheme include the time to determine the next routing step and acquire a link and buffer at all intermediate nodes, the time to determine the routing step and acquire a link to the destination node (but no buffer), the time to transmit the data, and all of the time spent waiting to acquire the necessary links and buffers. As with K-family routing, the overheads incurred by the sending and receiving processors are ignored. The QAP routing scheme is persistent and deadlock free, thus all messages will eventually make it to their destinations without further intervention by the sending node. Similar link bandwidth and routing logic switching times are used in all simulations.

The chief external characteristic that distinguishes the K-family routing schemes from QAP is that the K-family protocols require bidirectional end-to-end communications in order to establish routes and acquire and release links (such communication requirements are typical of circuit based schemes). This end-to-end communication requires two lines. One line is required from source toward the destination for the acquisition of links during the route setup phase and later for the transmission of data. Another line is required from the destination (or *blocked* end point) back toward the source to control the acquisition and release of links during the route setup phase and, ultimately, to control the flow of data during the transmission phase. While it may be possible to construct a single line communication protocol for K-family routing, such a protocol will be complex and costly to implement. The QAP protocol requires only point-to-point flow control. This simple point-to-point flow control information may be passed back from an interim receiving node to its predecessor on the same line that is used for the transmission of data (see [Buz88]).

In an attempt to measure the impact of this design characteristic, we provide further simulation results that compare the performance of QAP and $K(K-1)$ with an equal number of interconnects between nodes. There are two choices for constructing systems like $K(K-1)$ that require two lines for each unidirectional data path. The system can be built with four lines (two in each direction) connecting adjacent nodes. This allows simultaneous message transmission in opposite directions between two adjacent nodes for the K-family routing schemes because the logical path in each direction has its own pair of data and control lines. In a four line system the QAP scheme is assumed to make use of the additional lines to double its link bandwidth. Alternatively, the system can be built with two lines (one in each direction) connecting adjacent nodes. In this configuration, the K-family must acquire both lines to support message transmission in either direction. Thus, there is only one logical path between adjacent nodes and it must be shared between message transmissions in both directions. In the comparisons illustrated in this paper we have used this latter configuration. We have also run comparisons that assume four lines between adjacent nodes. The relative performance characteristics are comparable to the two line comparison.

The results of all of the comparisons are shown in Figures 1 through 4. The simulations in all cases assume a hypercube of degree six. We ran simulations on hypercubes of up to degree 8, in all cases the relative results are similar. The first two figures show the results of comparisons in which the $K(K-1)$ scheme was run on a four line system while the QAP scheme was run on a two line system. The latter two figures show the results with the $K(K-1)$ scheme constrained to use the same number of communication lines as QAP. Each of the graphs show the performance of both $K(K-1)$ and QAP routing across a range of synthetically generated message traffic loads. The probability distribution functions that are used to generate these traffic loads are given in Table 2. These distribution functions determine message destinations, message lengths and the delay time between the generation of successive messages. They are generated independently for each node. In Table 2 the argument for the exponential distribution function is the mean, the arguments for the normal distribution are the mean and variance, respectively, and the arguments for the uniform distribution specify the range of values. The specific message destination is chosen uniformly from among all of the nodes that are a given distance away. Two curves are shown for each of the routing schemes, the lower curve indicates the time for the beginning of the data to arrive at the destination, the upper curve indicates the time for all of the data in the message to arrive. The graphs in Figures 1 and 3 show simulation results for an average message length of 512 bytes, Figures 2 and 4 show results for an average message length of 2048 bytes.

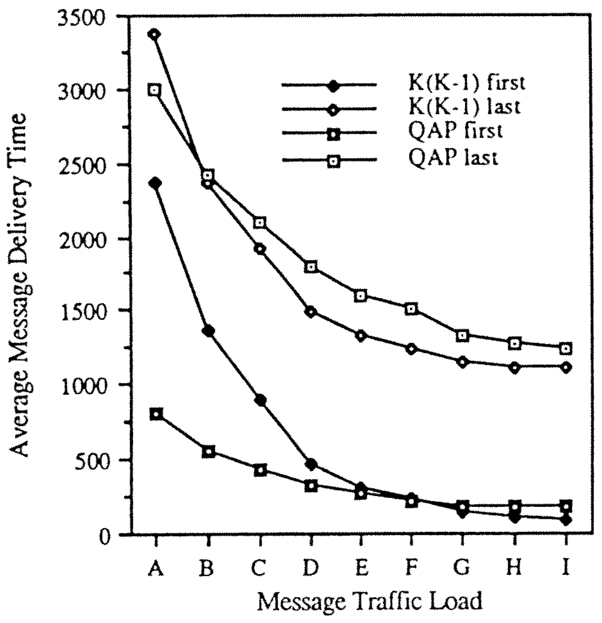

Figure 1: Small Messages, Non-equal Lines

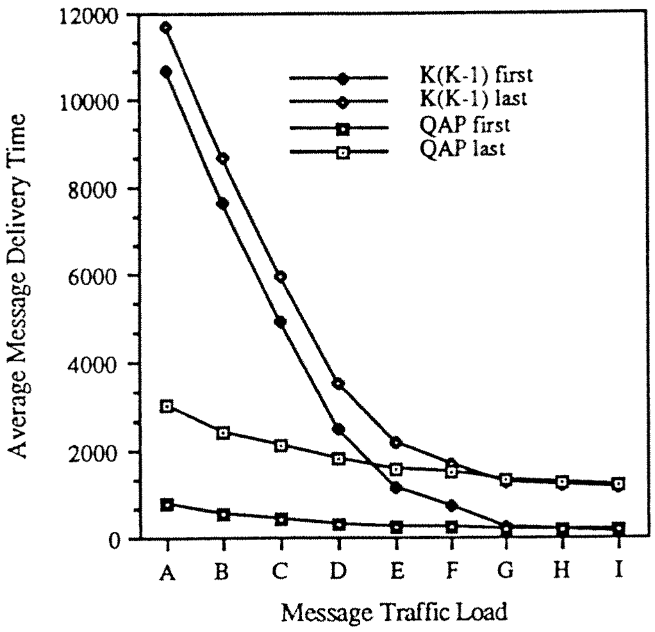

Figure 3: Small Messages, Equal Lines

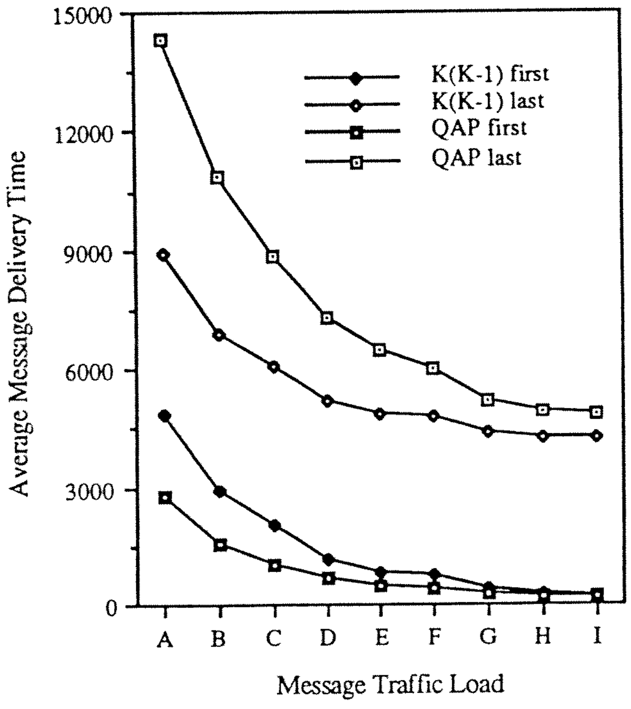

Figure 2: Large Messages, Non-equal Lines

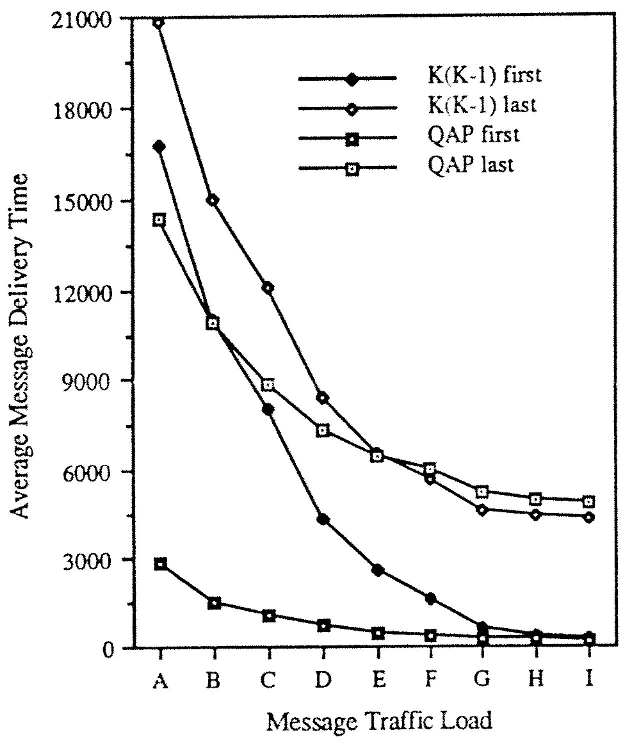

Figure 4: Large Messages, Equal Lines

case	length	intergeneration time	distance
A	exp(512)	normal(1024, 512)	uniform(1, 6)
	exp(2048)	normal(4096, 2048)	uniform(1, 6)
B	exp(512)	normal(1280, 640)	uniform(1, 6)
	exp(2048)	normal(5120, 2560)	uniform(1, 6)
C	exp(512)	normal(1536, 768)	uniform(1, 6)
	exp(2048)	normal(6144, 3072)	uniform(1, 6)
D	exp(512)	normal(2048, 1024)	uniform(1, 6)
	exp(2048)	normal(8192, 4096)	uniform(1, 6)
E	exp(512)	normal(2560, 1280)	uniform(1, 6)
	exp(2048)	normal(10240, 5120)	uniform(1, 6)
F	exp(512)	normal(3072, 1536)	uniform(1, 6)
	exp(2048)	normal(12288, 6144)	uniform(1, 6)
G	exp(512)	normal(5120, 2560)	uniform(1, 6)
	exp(2048)	normal(20480, 10240)	uniform(1, 6)
H	exp(512)	normal(7168, 3584)	uniform(1, 6)
	exp(2048)	normal(28672, 14336)	uniform(1, 6)
I	exp(512)	normal(9216, 4608)	uniform(1, 6)
	exp(2048)	normal(36864, 18432)	uniform(1, 6)

Table 2: Message Distribution Functions

As traffic loads increase, both schemes encounter a knee in their performance curves which begins to significantly limit performance. This effect is most apparent for $K(K-1)$ routing in the equal line comparisons. The performance of QAP in these cases ranges from about equal to that of $K(K-1)$ under light message traffic loads to significantly better than $K(K-1)$ under moderate to heavy loads. Based upon the results illustrated in Figures 3 and 4, it is easy to argue that QAP is the preferable choice for systems in which the number of communication lines is equal.

For the comparisons in which $K(K-1)$ is allowed twice as many communication lines as QAP, the performance of QAP is still much better for moderate to heavy message traffic loads. In systems that allow the processing of message data as it arrives, it is the lower (*first*) set of curves that are most important. They represent the time delay between the sending of a message and the processing of the initial message data. The consumption of message data frequently occurs at a fraction of the peak message delivery rate. For example, we have shown that the message delivery bandwidth of the NCUBE is more than four times greater than the consumption rate of message data when performing the double precision vector operation: $\vec{X} = a\vec{X} + \vec{Y}$ (see [BM88]). The gap between the *first* and *last* curves provides a measure of message delivery rate. Circuit based schemes like $K(K-1)$ deliver messages at peak rates. Given the example above, the gap between the *first* and *last* curves for QAP would have to be more than four times greater than the gap for $K(K-1)$, before any performance degradation would occur. This never occurs across the range of simulated messages loads and even if it did, it would be offset by the ability of QAP to begin processing message data much sooner. In the cases with the lighter message traffic loads and shorter messages $K(K-1)$, routing shows a performance advantage. However, QAP can recover much of this difference by using smaller packets, this can be seen by looking at the simulations with 32 byte packets which are shown in Figure 6.

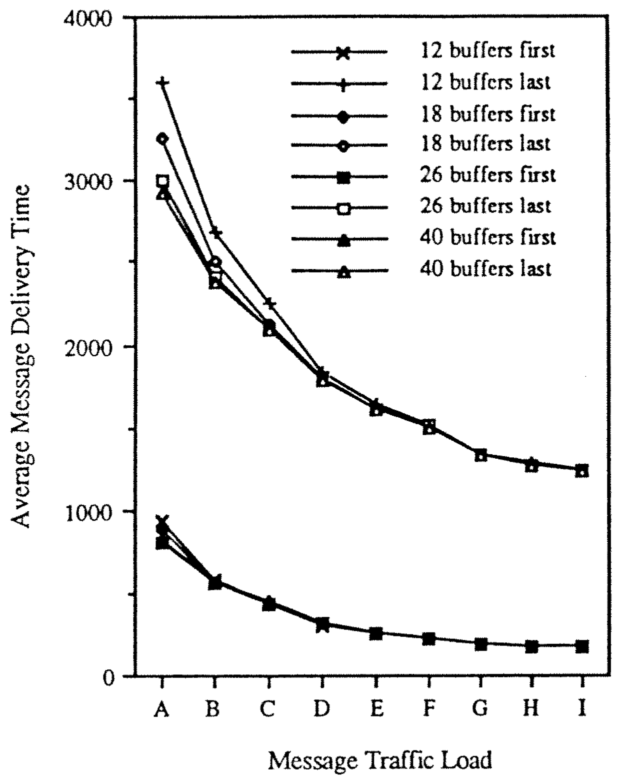

Figure 5: QAP -- Buffer Number Comparison (small messages)

4.1 Selecting Buffer and Packet Sizes for QAP

There are two architectural design parameters that must be selected for the QAP routing scheme, the size of the message packets and the number of packets that may be buffered at each node. The number of buffers that are provided at each node is a very stable parameter. It may be varied over a large range with negligible performance impact. Our simulation results, shown in Figure 5, indicate that performance effects are negligible across all message traffic loads for all cases where the number of buffers provided at each node is greater than about three time the number of neighboring nodes. A small performance impact occurs under heavy message traffic loads when the number of buffers is small. The incremental improvement that is gained by adding additional buffers also becomes minimal once the number of buffers reaches about four times the number of neighboring nodes.

The effects of packet size on performance are more noticeable. Above a certain level, which for our simulation cases has empirically been determined to be about 64 bytes, increases in packet size lead to simultaneous increases in first packet delivery time and decreases in

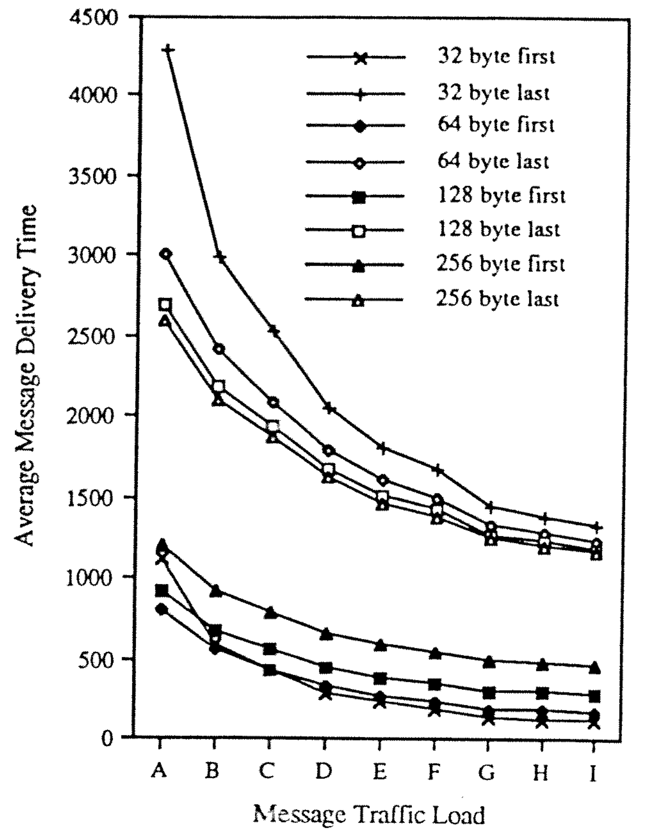

Figure 6: QAP -- Packet Size Comparison (small messages)

overall (last packet) delivery time. Thus, by increasing packet size we can make the effective bandwidth of QAP tend toward that of $K(K-1)$. Below about 64 bytes, further decreases in packet size lead to significant increases in the last packet delivery times and only minor decreases in the delivery times for the initial packet. This effect is a result of the overhead of packet handling beginning to dominate over the benefits of small packet interleaving. The effects of these tradeoffs can be seen in Figure 6.

5 Conclusions

Quasi-adaptive packet routing provides a simple and effective solution to the problem of reducing message delivery latency. It accomplishes this objective for a wide range of message traffic loads and message sizes. QAP performs particularly well when it is allowed to use the same number of communication lines as other schemes or when the communication loads are moderate to heavy. The key features of QAP are that it is an adaptive scheme, it avoids deadlock and it requires at most neighbor-to-neighbor communications to perform routing or message delivery actions. In future work we will investigate the use of packet based routing techniques to support shared virtual memory paging on hypercubes. This work will coincide with explorations into the feasibility of running the Mach [RTY+88] operating system on a hypercube.

References

[BM88] Gregory Buzzard and Trevor Mudge. High performance hypercube communications. In Geoffrey Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications*, pages 600–609. ACM, 1988.

[Buz88] Gregory D. Buzzard. *High Performance Communications for Hypercube Multiprocessors.* PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, August 1988.

[CMP+88] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed. Hyperswitch network for the hypercube computer. In *15th Annual International Symposium on Computer Architecture*, pages 90–99, May 1988.

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

[Lan82] C. R. Lang. The extension of object-oriented languages to a homogeneous concurrent architecture. Department of computer science, technical report, 5014:tr:82, California Institute of Technology, 1982.

[Nug88] Steve Nugent. The iPSC/2 direct-connect communications technology. In Geoffrey C. Fox, editor, *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 51–60. ACM, 1988.

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

[RTY+88] R. Rashid, A. Tevanian Jr., M. Young, D. Golub, R. Baron, D. Black, W. Bolosky, and J. Chew. Machine-independent virtual memory management for paged uniprocessor and multiprocessor architectures. *IEEE Transactions on Computers*, pages 896–908, August 1988.

[Val82] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal of Computing*, pages 350–361, May 1982.