Performance Evaluation of a Parallel I/O Subsystem for Hypercube Multicomputers

Joydeep Ghosh, Kelvin D. Goveas and Jeffrey T. Draper

Department of Electrical and Computer Engineering,
University of Texas, Austin, TX 78712-1084.

Proposed Running Head:

Parallel I/O Subsystem for Hypercube Multicomputers

Contact Author:
Joydeep Ghosh,
Department of Electrical and Computer Engineering,
University of Texas, Austin, TX 78712-1084.
Tel: (512)471-8980; FAX: 471-5907
email: ghosh@pine.ece.utexas.edu

Abstract

Though input/output (I/O) from mass storage continues to be a bottleneck in current generation distributed-memory multicomputers, few studies have been conducted on the system-level impact of an *independent* I/O network and distributed file system for such machines. This paper quantifies the improvement in performance obtained in a hypercube multicomputer by providing a separate network that serves multiple I/O nodes operating under a distributed file system. First, the concept of balanced architectures is used to show that for a general purpose multicomputer with N processing elements, the I/O subsystem should scale as $\frac{N}{\log N}$. Analytical and simulation studies are performed on a hypercube architecture that is augmented by an independent I/O network that achieves this scale factor. For wormhole routing, a significant improvement in latency can be obtained for both small inter-processor messages and much larger I/O transactions, provided the bandwidth of a channel in the I/O network is slightly higher as compared with a PE-PE channel. Moreover, performance becomes relatively insensitive to data locality, and thus to the file allocation policy, when this extra bandwidth is available. Simulation results are presented for a 128-node hypercube, and are validated by an analytical model.

List of Symbols

1) Symbols in script:
$\mathcal{C}, \mathcal{M}, \mathcal{IO}$ stand for overall computational bandwidth, memory and I/O respectively, and are used in Sec.3 (introduction; Sec 3.1) only.

2) Greek Symbols:

Θ: Big Theta, used in Sec 3.1, 3.2.

ρ, λ: rho, lambda, used in sec 5.

μ: used as micro in μsecs.

All other symbols are in regular font.

1 Introduction

The throughput and capacity of I/O subsystems have not kept in step with the enormous increase in the speed and capacity of main memory and of processor power in the past two decades [26]. The resulting mismatch now limits system performance for many I/O intensive applications. In most first generation message-passing parallel computers, also known as *multicomputers* [3], the parallel computer is installed as a "back-end" to a host machine which serves all I/O requests. For example, the Intel iPSC/1 has a single ethernet connecting the host machine with all the processing nodes. The Symult 2010 is typically hosted from a Sun-3 workstation. While second generation multicomputers exhibit an order of magnitude improvement in I/O capability through the use of concurrent I/O, the I/O mismatch continues to be observed for a wide class of problems [3, 4].

The importance of balancing I/O bandwidth with computational power was highlighted in [22] where it is shown that for some applications, the I/O problem cannot be alleviated by simply adding more memory at each processor node. Current research on I/O problems is reflected in [4]. An extensive taxonomy and historical perspective is given in [35], while state-of-the-art I/O systems and future trends are highlighted in [15].

A pioneering study of disk I/O architectures for hypercube systems was done by Reddy and Banerjee [29, 30], who simulated the performance of several algorithms under varying degrees of disk synchronization and declustering. In this study, I/O nodes are embedded in selected nodes of a hypercube-based multicomputer, and the interprocessor communication links are used for routing both interprocessor and I/O traffic. Similarly, the Concurrent File System of the Intel iPSC/2 does not utilize a separate I/O network [27].

If there is little overlap between interprocessor communication and I/O traffic, then there is little degradation in using a common network for both purposes. It has been observed however, that there can be substantial overlap of these two distinct types of traffic in second generation multicomputer systems that have virtual memory and can run a number of processes in a time-shared manner [11]. Since the local memory at each processing node is limited, there is typically a considerable amount of paging between main memory and secondary storage (disks). Moreover, blocking circuit-switched routing mechanisms such as wormhole routing [7] are becoming the norm in current-generation multicomputers with direct-connect topologies because of the availability of fast routing automata [3, 9]. In such scenarios, if both I/O and interprocessor communication use the same set of links, the transfer of a large file can block communication links for extended periods and can cause high variability even in the latencies seen by shorter messages exchanged amongst other processing nodes.

The trends outlined above motivate the use of a separate network for interconnecting the I/O nodes that can cater to most of the I/O communication and minimize interference with the interprocessor communication network. Some other researchers have also argued the need for such a network for improved scalability, configurability, connectivity, and for separation of the application software from the software responsible for I/O operation which can be primarily located on the I/O nodes [13, 25, 37]. Among the commercially available systems, the NCUBE multicomputer can be purchased with attached I/O nodes that can be connected separately in a ring network. However, there has been no detailed analytical or simulation studies that quantify the advantages of, and the cost/performance tradeoffs involved in using an independent network.

Preliminary studies performed by us recently showed that a separate I/O network can not only decrease average network latency, but also reduce its sensitivity to locality of I/O traffic [11]. In that study, packet-switching was used in conjunction with a simple model for computation and communication. Since wormhole routing and its variants have quite different characteristics as compared to packet-switching, especially in the abruptness of the network saturation phenomenon, it is not immediate whether qualitatively similar results can be obtained for this routing technique of choice for current-generation multicomputers. In this paper, the usefulness of a separate and independent I/O network is studied in detail and quantified for various data access patterns in a multicomputer that uses wormhole routing. Section 2 provides further background by summarizing different ways in which computer architects have realized high bandwidth I/O systems in the past. In section 3, the concept of *balancing* is used to determine a good scaling factor for the I/O subsystem proposed in this paper. A scheme based on incomplete hypercubes is given that yields a desirable scaling factor for the growth of the I/O subsystem. In section 4, a concurrent I/O subsystem is described. The next two sections study the impact of this I/O subsystem for a hypercube-based multicomputer with 128 nodes, using both simulation and analytical studies. The results quantify the need for an independent I/O network, and demonstrate its superiority over a system without an I/O network under various scenarios.

The issues explored in this paper are largely orthogonal to approaches that use arrays of I/O disks for improved performance [26], or to methods such as disk striping [34] and disk synchronization [19]. These techniques can be used to increase bandwidth and decrease the observed latency at each I/O node, and are thus complementary to the approach taken by us. Also, in this paper the I/O subsystem augments the multicomputer, as opposed to architectures such as the hypernet where the I/O nodes are an integral part of system design [14].

2 The Case for Parallel I/O

The capacity of a single large magnetic I/O disk is limited by the number of bits that can be stored per square inch. Its speed is limited by the seek and rotational latency delays, which are mechanical and have been decreasing at a rate of only 7% per year [26]. Recently, optical disk technology for massive data storage and retrieval is becoming attractive [36]. Optical disks have greater data storage capabilities, but are slower, with seek times of 60-90 milliseconds (ms) as compared to 14 ms for the fastest Winchester drives. They also lack a direct overwrite capability at present.

At present, the most feasible solution for obtaining a high performance I/O system is to resort to parallelization at various levels. A number of projects such as RAID ([26], [34]) have proposed a *low level parallelization* of I/O by dividing data blocks into smaller sub-blocks and storing these sub-blocks in different disks which are organized in an array. This strategy speeds up the transfer rate of the system by an order of magnitude and improves reliability. Also, since the data is split between a number of disks, the individual capacity of each disk can be less. This helps in bringing down both disk costs and seek time. Studies have shown that this strategy speeds up performance by 10% to 70% [15]. Further improvement is limited largely because the dominating seek and rotational latency delays are not reduced substantially.

In a multiprocessor system with shared memory, a number of concurrently active I/O channels

can be used to *directly access* the main memory. Such *medium-level parallelization* is used extensively by mainframe computers and multiprocessor systems, including the Cray 2, ETA-10 and the Alliant FX/8.

For distributed-memory multicomputers, it becomes expensive to use I/O channels directly addressing every local memory module. However, *high level parallelization* can be exploited through the use of special I/O processors. These I/O processors are associated with selected multicomputer nodes, and are typically of the same complexity as the processing elements (PE) used in the computing nodes. These I/O nodes provide access to I/O devices such as RAID arrays through standard interfaces such as SCSI and HIPPI, or through a disk controller [28]. They can also function as device/file servers, and use part of their local memory as a disk cache. A good comparative description of the I/O nodes used in the Intel iPSC/2 and NCUBE/9 multicomputers can be found in [28].

The performance of an I/O subsystem using multiple I/O nodes is strongly influenced by the schemes used to store, maintain and access data. Using disk striping, one can interleave data across independent disks under the control of a single file system [34]. This technique is able to provide adequate bandwidth for most uniprocessor systems. For multiprocessors, a scheme for having parallel interleaved files has been proposed in [8] so that the file system is run on multiple processors. A simpler mechanism, namely the interleaving of data bits across multiple processors, can be used for fine-grain SIMD machines. This mechanism is implemented in the Data Vault of the Connection Machine 2 [24], where each word can be written or read in parallel.

For any of the schemes for data storage and access outlined above, an immediate question that arises is how many I/O nodes should be used, and where should they be located. How should these choices depend on the number of computing nodes used? Finally, if an independent I/O network is to be used, how should the I/O nodes be connected and addressed? In the next section, we start addressing these questions by examining the I/O requirements of computer systems for typical algorithms in order to determine by how much the effective bandwidth of an I/O subsystem should increase with the net computing power in order to obtain a scalable architecture.

3 Scalable Architectures

A *scalable architecture* for a given algorithm is one in which the number of processing units can be increased to any arbitrary size without decreasing the relative efficiency of the parallel system for that algorithm. This means that the system achieves linear speedup for that algorithm. The idea of balancing, proposed in [22], is used to determine the scaling desired for the I/O subsystem. A PE is said to be balanced with respect to an algorithm (or computation) if the total I/O time, t_{io} equals the computation time, t_{comp} [22]. The same definition also holds for a multicomputer system. It is shown in [1] that a balanced architecture implies a scalable one and vice versa.

Let C be the computational bandwidth, in MIPS, of a single PE, IO be its external I/O bandwidth in MBytes/second and M the size of its local memory in MBytes. For a multicomputer system with N identical PEs, the overall computational bandwidth and memory is given by $\mathcal{C} = N \times C$ and $\mathcal{M} = N \times M$ respectively. However, the net external I/O bandwidth, $\mathcal{IO}$, is typically significantly less than $N \times IO$, since a lot of the I/O bandwidth of each PE is used within the multicomputer system for inter-PE communication.

For computationally intensive algorithms such as matrix multiplication, where the number of processing steps required is a super-linear function of the problem size, we can re-balance an architecture in two ways. The first method is to *increase the size of the data set* of the problem. This will increase the number of computational steps by a larger factor than it will increase the number of I/O steps. This is often the preferred method in practice [10]. However, it may not be possible to increase the problem size indefinitely because of application characteristics or memory limitations. The second solution is to *increase the external I/O bandwidth* of the computer system. This approach does not require increasing the internal memory of each PE every time the computer system size is increased.

In I/O intensive algorithms, the number of processing steps required is a sub-linear function of the problem size. For example, it requires $\log N$ steps to search for a value in a sorted array of N numbers. Searching is therefore an I/O intensive algorithm. For these algorithms, there is no realistic way of rebalancing the architecture because as the problem size increases, t_{io} increases at a faster rate than t_{comp}. This means that the I/O bandwidth for a multicomputer must increase at a rate greater than the computational bandwidth.

3.1 Scaling factor for sorting

Let us determine the factor for sorting when the problem size is much larger than the memory of the multicomputer. Sorting of S values by comparison can be done in two phases. In the first phase, $\frac{S}{\mathcal{M}}$ subsets of $\mathcal{M}$ values each are sorted. For the first phase, $t_{comp} = \Theta(\frac{\mathcal{M}\log\mathcal{M}}{\mathcal{C}})$ and $t_{io} = \Theta(\frac{\mathcal{M}}{\mathcal{IO}})$ time units. Note that t_{comp} includes the time needed for interprocessor communication during sorting. In the second phase, the sorted subsets are merged together using an $\mathcal{M}$-way merge algorithm. It can be shown [22] that the time requirements for the second phase are the same as for the first phase. Thus,

$$\frac{t_{comp}}{t_{io}} = \Theta(\frac{\mathcal{IO} \times \log\mathcal{M}}{\mathcal{C}}). \tag{1}$$

If a single PE is balanced for sorting, then

$$\frac{IO \times \log M}{C} = \Theta(1). \tag{2}$$

From Eq. (1) and (2), we obtain for the system of N PEs with M MBytes of memory per PE,

$$\frac{t_{comp}}{t_{io}} = \Theta\left(\frac{\mathcal{IO} \times (\log N + \log M)}{IO \times N \times \log M}\right). \tag{3}$$

Thus, for the multicomputer to remain balanced, we need

$$\frac{\mathcal{IO}}{IO} = \Theta\left(\frac{N \times \log M}{\log N + \log M}\right).$$

In other words, the I/O capacity must be increased by about $\frac{N}{log N}$ to rebalance. A similar analysis is given in [1] to show that the scaling factor for matrix multiplication is $\sqrt{N}$.

For a general purpose computer system, it is not possible to predict the job mix that will be used. However, noting that typically a large percentage of work done on such systems is related to

high flux problems such as sorting, we consider that a suitable scaling factor for the I/O subsystem is $\frac{N}{logN}$. To achieve this scaling factor, the number of I/O nodes required is $\Theta(\frac{N}{\log N})$. Then, each I/O node will service an average of $\Theta(\log N)$ processing nodes.

3.2 Allocation of I/O nodes

Each I/O node can be assigned to a cluster of PEs such as a subcube of a hypercube network, or a group of consecutive PEs in a linear or ring topology. For graphics applications which demand that PEs write to a graphical device with equal ease, it is desirable to have the I/O node connected directly to all the PEs in its assigned cluster [37]. However, for most other applications this added complexity is not warranted, and it suffices for each I/O node to have a direct link to only one PE node. A PE node which has a direct link to the I/O node will see a network latency corresponding to 1 hop. The other PE nodes serviced by this I/O node will see a network latency corresponding to a small number of hops depending on the allocation algorithm.

An optimal allocation is one in which every PE is either 1 or 2 links away from exactly *one* I/O node. In [30], a scheme based on Hamming codes is used to show that an optimal allocation exists for a binary hypercube if and only if the number of dimensions in the cube, n, is given by $n = 2^m - 1$. In such cases, the node positions chosen for I/O augmentation correspond to code words, and all other nodes are at a distance of one (single error correction) from exactly one node with I/O (code word). For networks in which an optimal allocation is not possible, one can determine a minimum allocation, which is the smallest allocation such that if we deallocate any of the I/O nodes, at least one of the processing nodes will be more than 2 hops away from an I/O node. Finding a minimum allocation is analogous to the dominating set problem in graph theory, and is known to be NP-complete.

We propose to follow the Hamming code based embedding scheme of [30] that was outlined above, with the added observation that this scheme meets the desired scaling factor as determined in the previous section. The coupling between the PE and I/O subsystems, described in Section 4, however differs from that proposed in [30]. An n-dimensional hypercube is augmented with an I/O system by first extending it to the $(n+1)^{th}$ dimension. About $\frac{N}{\log N}$ of the 2^n added positions are used for placing the I/O nodes, in accordance with the scalability results of the previous section. In effect, an incomplete hypercube is obtained. If $n = 2^m - 1$, the positions for the I/O nodes are decided by the Hamming code based allocation scheme. Hamming codes use $2^m - 1 - m$ bits for code vectors and m bits for error correction. Thus the I/O nodes can be connected together as a smaller binary hypercube of dimension $\log \frac{N}{\log(N)+1}$. For other sizes, smaller cubes (which have a perfect mapping) are combined. For example, we can combine two 3 dimensional cubes which have a perfect mapping to get a 4 dimensional cube as shown in Figure 1. The I/O nodes are in the 5^{th} dimension at positions 10000, 10111, 11001 and 11110. The other positions in the 5^{th} dimension are empty. In such cases the I/O interconnection network is a binary hypercube with $\lceil \log \frac{N}{\log(N)+1} \rceil$ dimensions.

The proposed augmentation provides the I/O network with properties of scalability, fault tolerance, connectivity and broadcasting facility derived from the symmetric and rich interconnection properties of the hypercube. In case an I/O node fails, the I/O requests meant for that node can be routed to a nearby I/O node by simply changing the forwarding address at the PE node connected

to the failed I/O node. As will be seen later, the I/O network has direct access to the I/O device interface. Hence, the devices under the control of an I/O node are still accessible even if that node fails. Broadcast on this structure takes $\Theta(\log N)$ communication steps. Also, this network can more easily support a distributed file system because it has identical I/O nodes, a simple routing mechanism and a $\Theta(\log N)$ distance between any I/O node and PE pair.

4 System Modeling

4.1 PE and I/O node configuration

The incorporation of an I/O subsystem differentiates the PE nodes into two classes: those that are directly attached to an I/O node (class I), and those that are not (class II). The proposed interface for PE nodes of class I is shown in Fig. 2. Extra links are added to the routers of these PEs to provide two levels of communication with the I/O subsystem: a direct connection to the local I/O node, and an indirect connection to remote I/O nodes via the local I/O node's router. PEs in class II have ordinary routers that are linked only to other PE routers, and not to any I/O nodes or I/O routers. The scheme given in Fig. 2 has the merit that while a remote PE needs to access a local PE's router to reach an I/O node, there is no store-and-forwarding or copying involved at the local PE. Similarly, by symmetry, a remote I/O node can satisfy an I/O request via the local I/O-I/O router without bothering the local I/O node.

Letters associated with communication channels in the diagram represent the width of the corresponding links. Unlabelled links have only one channel, whose bandwidth is considered to be 20MB/sec for obtaining the results of Section 5. PEs can sink or source up to k messages simultaneously, while the r-channel links allow I/O nodes to sink or source r messages concurrently. A link with i channels does not exactly have i times the bandwidth of a link with unit weight. If a single message arrives at an i-channel link, it can only use one channel, not i. However, if multiple messages arrive for transmission, up to i messages can travel on the same link with each message using a separate channel. Physically, an i-channel can be realized as a channel with i times the bandwidth which is time-multiplexed to feed the i buffers associated with that channel in order to support the pipelined "wormhole" routing described in the following section.

For the experiments of Section 5, we primarily considered a 7-dimensional 128-node binary hypercube with 16 I/O nodes. This choice for the hypercube size was made because:
(i) It has a perfect Hamming embedding, and nearest other sizes with perfect embedding, namely $n = 3$ and $n = 15$ were either too small or too big.
(ii) Results for 3, 4, 5 and 6-dimensional hypercubes shown in Section 5.1.1 are qualitatively similar to those observed for the 7-d hypercube. Quantitatively, the smaller the hypercube, the higher is the traffic generation required per node in order to saturate some network link, which is as expected.

For the 7-d cube, the I/O nodes were connected to PE nodes 0, 7, 25, 30, 42, 45, 51, 52, 75, 76, 82, 85, 97, 102, 120, and 127. This node allocation corresponds to the Hamming-code embedding explained in the previous section. It is optimal in the sense that every PE in class II has exactly one neighboring PE that belongs to class I. In other words, PEs can be separated nodes into disjoint

groups of 8, with an I/O node directly connected to one member in each group. This I/O node is referred to as the local I/O node of the 8 PEs because it is their nearest I/O node. All I/O read and write requests are directed to a local I/O node, which then forwards them to other I/O nodes if necessary. Thus the file management system is symmetric and distributed.

For the "test" hypercubes, the 16 I/O nodes are interconnected to form a secondary 4-dimensional binary hypercube with varying channel bandwidths. The "control" hypercube is identical to the test cubes except that the I/O nodes are not connected via a secondary network. Thus, links labelled b, r, and n would be absent from the interface diagram shown in Fig. 2 for the control hypercube.

The internal architecture of an I/O node is similar to I/O nodes used in the Intel iPSC/2 [5]. The local memory is dual ported so that it can be accessed by both the internal bus and the I/O bus which interfaces with the disk controllers. This facilitates the use of this memory as a disk cache, besides its use in file management duties. System parameters that are used for simulation studies are given in Table 1, and they reflect typical values observed in current-generation multicomputers [3, 15]. Files larger than 32 blocks are broken up into smaller files for simulation purposes. While different disk cache hit rates were considered, we note that setting the hit rate to a 100% allowed us to detect changes in the I/O response time of the order of a few μs. A lower hit rate does not statistically affect the *network* latencies unless saturation occurs. This is because, in steady state, disk cache misses do not change the pattern or intensity of communication traffic, only responses to the corresponding requests are delayed. Thus they offset the average latency figures by (miss rate) $\times$ (av. disk access time).

4.2 Routing Mechanism

The flow control of PE-PE messages as well as I/O block transfers are governed by a pipelined circuit-switched routing technique called wormhole routing [7]. This technique can be considered as a blocking variant of virtual cut-through routing [17], and is employed in several multicomputers including the *i*WARP, Symult 2010 and Intel Paragon. This technique can be efficiently supported in hardware, and at low loads, is characterized by very low network latencies which makes the network transit time almost independent of the number of hops taken. However, the blocking nature causes an avalanche effect if network traffic is high, with an abrupt increase in latencies leading to network saturation.

In wormhole routing, each message is segmented into small units called "flits" (flow-control units) which are typically one or a few bytes long. A flit can be regarded as the unit of a message which may traverse a channel in one time step. The first flit of a message advances along a route with the remaining flits following in pipelined fashion. This first flit, called the "head", contains routing information for the message. If the head flit blocks due to the unavailability of a channel, the flow control within the network blocks the trailing flits. Therefore, only the head is processed at an intermediate node before the message is forwarded.

This scheme has two advantages over the store-and-forward method of first-generation multicomputers:

1. It avoids using storage space in the intermediate nodes through which a message is routed.

Parameter	Value
Message Startup Overhead	$200\mu s$
I/O Node Latency for block access from cache	$400\mu s$
Average Disk Access Time (Seek + Rotational Latency)	$20ms$
Average disk cache hit rate	70-100%
Single Channel Bandwidth	20MB/sec
Average PE-PE Message Length	32 bytes
Flit Length	1 byte
I/O Block Length	4096 bytes
Size of I/O File Files larger than 128K considered as several 128K files	1 - 32 blocks

Table 1: System parameters

2. It makes the message latency largely insensitive to the distance in the network under contention-free conditions.

The routes messages follow are specified by the e-cube or "left-right" (LR) routing algorithm, a deterministic algorithm which provides deadlock-free routing on binary hypercubes. In the e-cube algorithm, the n hypercube dimensions are strictly ordered as $d_{n-1}, d_{n-2}, \ldots, d_0$ using the relation $d_i > d_j$ if and only if $i > j$. At the source node and each intermediate node of a message traversal, the dimensions are inspected from highest to lowest in order. The first dimension in which the destination address differs from the current address is selected for traversal, and the message is forwarded along the corresponding channel.

4.3 I/O Traffic Patterns

I/O accesses can be explicit such as distribution of code and data to the private memories, and off-loading of results, or implicit due to activities such as paging and file migration. While there have been some studies on the nature and volume of I/O activity for multi-user environments on mainframes [12] and workstations [16], almost no I/O traces are publicly available at present for applications on distributed memory machines with concurrent I/O facilities.

Regarding the *volume* of I/O activity, recent studies indicate that the long-honored Amdahl-Case rule of one bit of I/O required per instruction understates the I/O demands [2]. In fact, for distributed memory machines, I/O traffic is expected to be much higher because of the limited local memory on each processing node. A further increase in I/O traffic is observed in multicomputer systems with virtual memory, where several processes can be running in a time-shared manner. Also, typical frequency of I/O messages, as a fraction of the regular communication messages, is in the range of 3 to 20% [21, 29].

The *nature* of I/O activity varies widely with the application mix. For scientific workloads, I/O traffic is not uniform with time but is typically concentrated towards the beginning (loading of data, code) and the end of the computation stage, with occasional I/O in the middle stages for data access and synchronization. File/transaction system workloads, on the other hand are more uniformly distributed in time. For both types of applications, large files are broken into blocks which can be declustered, i.e., distributed on a number of disks such that different blocks of the same file can be accessed concurrently. The block access locality, which gives the probability of finding a requested block at a local I/O node, is thus affected by the extent of declustering [29]. Overall, I/O requests for file accesses can be categorized as broadcast, signifying that a block is required for all nodes; gather and scatter operations that involve a subset of the nodes, and independent file requests from single nodes [37].

For the analysis and simulation study reported in this paper, a spherical model of locality [14, 32] is used for I/O requests, which occurs concurrently with a globally uniform PE-PE traffic generated by short messages among the PEs. Various mixtures of broadcast, multicast and individual I/O requests can be well approximated with the spherical locality model by choosing an appropriate value for the data locality. The following approach is used for generating both I/O and PE-PE traffic for simulation purposes, in a manner consistent with the traffic model.

Each PE generates messages at intervals derived from an exponential distribution with mean

time between arrivals = $\overline{t_a}$. A message is designated as an I/O request (read/write) with probability P_{io}. Otherwise, it is counted as a "PE-PE" message of size 32 bytes, whose destination is selected from the other 127 PEs with uniform probability.

When an I/O message is generated, the probability that it is marked as a read request is P_r, the same as the read rate. A read request results in a 32 byte "PE-I/O" message that includes a description of the requested blocks, identity of the requesting node and process etc. This message is directed to the local I/O node where the actual location(s) of the block(s) requested is determined. If a remote block is to be read, then the local I/O node sends a 32 byte "I/O-I/O" request-forward message to the remote I/O node. Requests that are satisfied locally return via the m-channel link from the I/O node to the PE-PE router. Requests that are satisfied remotely return via the n-channel link of the I/O-I/O router at the local I/O node that forwarded the request. They thus bypass the local I/O node. Block transfers are referred to as "IO-PE" messages. For both local and remote I/O nodes, the startup and access latencies used are given in Table 1, and are representative of current technology [15].

Figure 3 shows the model used for I/O read locality. The probability that a requested block resides at the local node, either in the disk cache or in the disk, is $P_l = h$. Otherwise it will be found at any one of the other I/O nodes with equal probability, which is simply $(1 - P_l)/(N_{IO} - 1)$ for a system with N_{IO} I/O nodes. The same traffic model and read-write protocols are used for both the control cube and the test cube. In the control cube, all I/O-I/O and I/O-PE messages are routed via the PE interconnection network whereas in the test cubes these messages are routed via the independent I/O network.

All write requests are satisfied locally. The requesting PE sends both request and data in one large message to the local I/O node via the m-channel link where it is either cached or written to disk.

5 Simulation Results

The simulation studies attempt to quantify the effect of the independent I/O network on both PE-PE message latencies and the response time of I/O requests. Of particular interest is the identification of the traffic patterns and the intensities that cause saturation in the various network configurations. For blocking routing mechanisms, like wormhole routing, saturation occurs very abruptly, leading to a drastic increase in latencies for a small increment in traffic. In fact, in a saturated network, I/O response times are dominated by network latencies rather than by disk-cache miss rates. Documenting traffic intensity at saturation is therefore equivalent to determining maximum throughput sustainable by the network.

In the preliminary experiments, the sourcing/sinking capabilities of the PE and I/O nodes, denoted in Fig. 2 by k and r respectively, are made equal to the net bandwidth of the node. This is motivated by the observation that in recent machines such as the *i*WARP [23] and the Intel Paragon [6], the compute nodes provide matching bandwidths even if all channels are active concurrently. Similarly, the initial experiments ensure that the coupling between an I/O node and the local PE is not a bottleneck by choosing appropriate high values for n and m. In this way, the impact of overall traffic rate, I/O traffic rate and nature, and bandwidth of the I/O network, on overall system

performance can be isolated. Later on, it is shown that even moderate values of n and m are sufficient to attain similar performance.

Simulation was conducted at the flit level. In one time step (0.05 μs), each message advances by one flit, provided of course that there is no contention. We varied $\overline{t_a}$, m, n, b, P_{io} (I/O rate), P_r (read ratio), and P_l (locality) and monitored the average PE-PE, PE-I/O, I/O-I/O, and I/O-PE latencies, as well as the PE and I/O waiting times. The first four monitored averages represent header latencies only. They measure the delay encountered by the head flit as it travels from the router of the sending node to the interior of the receiving node. The waiting time indicates how long the head flit must wait at the source node before it can enter the network. Such a wait is necessitated when the requested outgoing channel is being used by some other message and is thus unavailable. Blocked messages are placed in a FIFO queue associated with the outgoing channel. Total message latencies for each of the four message types can be calculated from the header latencies by adding a constant c to the header latency, given by

$$c = \text{STARTUP TIME} + (\text{MESSAGE SIZE - 1}) * 0.05\mu s.$$

Each of the points in the graphs that follow represents an average of 10 runs. Within each run, statistics were collected after a total of 50,000 messages had been received to escape warmup effects. A simulation run was terminated when a total of 150,000 messages were received. Each figure shows a family of curves. If some point is not plotted on a curve even though a value for the same abscissa is given for some other curve, it indicates that saturation has occurred on the first curve.

5.1 Effect of I/O network bandwidth on I/O response

Figure 4 shows the effect of increased communication traffic on both the processor interconnection network and the I/O network. The I/O network bandwidth, b, was set at 0 (no I/O network), 1, 2, 3 and 4. For each value of b, message interarrival times ($\overline{t_a}$) of 1.25ms, 0.625ms, 0.313ms, 0.156ms, 0.078ms and 0.039ms were used. The following parameters were constant for each cube: locality = 0.2, I/O rate = 5%, read ratio = 1.0, hit rate (P_c) = 1.0, $k = 9$, $m = 8$, $r = 4 * b$, $n = 4 * b$. As mentioned earlier, the bandwidths represented by k, r, m and n were set to maximum values to factor out their effect on the response times.

Since the control cube does not have an I/O network, all inter-I/O read requests and I/O-PE data messages were routed via the PE network causing an increase in the PE-PE latency over and above normal traffic levels that occurred in the test cubes, as is verified by Fig. 4(a). The figure also shows that the average PE-PE latency in each of the test cubes at identical message rates is the same up to the knee, but after that point, the curve for $b = 1$ separates. In general, it will be true that the PE-PE latencies of all unsaturated test cubes will be identical. By definition, an unsaturated network has to supply the PE network with data at the same rate as it is requested. Either the rate of data blocks departing an I/O node is the same as the rate of read requests entering it, or it is lower, in which case the network will eventually saturate. For a fixed message rate, the only way I/O traffic can affect PE-PE latency is through the intensity of data traffic on the set of links labelled D in Fig. 2. In all the unsaturated networks, the rate of data blocks returning via D is statistically equivalent to the rate of read requests entering the local I/O node via A, and is independent of the value of b in the test cubes, as corroborated by the figure. Saturated networks, however, have lower

PE-PE latencies as the rate of data blocks departing the local I/O node via D is lower than in the unsaturated networks.

While the average PE-PE latency seen by the unit bandwidth test cube is lower than the control cube's, it produces higher I/O response times, saturating earlier than the control cube. This surprising result is shown in Fig. 4 (b), which gives the overall header latency for an I/O read request. The figure suggests that it is better to route read requests and data blocks via the 7-dimensional PE network instead of the smaller 4-dimensional unichannel I/O network. However, when b is increased to two or more a drastic decrease in the I/O latency occurs. The $b = 2$ I/O cube has twice as much bisectional bandwidth as the $b = 1$ cube. We conclude that an I/O network significantly improves performance, but only if its communication links are of higher bandwidth than the regular PE interconnection network.

The knee of a curve in Fig. 4(b) indicates the value of $\overline{t_a}$ at which the I/O network will saturate. The symptoms of saturation are higher I/O network latencies, and in particular, very high I/O waiting times. At the highest traffic rate simulated, network latencies account for 6% to less than 1% of the I/O-I/O latency, the rest being waiting time. A non-zero waiting time at an I/O node indicates that the head flit is in a queue waiting for an outgoing channel that is currently occupied by request-forward messages or data blocks exiting the node or request-forward messages/data blocks flowing through the router en route to other destinations. Figure 4(c) shows average latencies encountered by a 32-byte request-forward message sent from the local I/O node to a remote node, to highlight the fact that the poor performance of the cube augmented with a $b = 1$ I/O network is due to congestion in the I/O network and not in the main PE-PE network. Thus, even these small "request-forward" messages see high latencies.

5.1.1 Results for Hypercubes of other Sizes

To determine whether the plotted trends were characteristic of our I/O architecture in general or if they were peculiar to a 7-dimensional hypercube, we simulated a 3, 4, 5 and 6-dimensional hypercube. Fig. 5 shows the number and placement of I/O nodes in each hypercube. The 3-dimensional cube, like the 7-dimensional one, has a perfect embedding but the 4, 5 and 6-dimensional cubes do not. As a result, the number of PE's serviced by an I/O node remains fixed at 4 even though the dimension of the I/O hypercube scales with the PE hypercube.

Fig. 7(a) shows the effect of increased communication traffic on the average PE-PE network latency in the 3-cube and the 4-cube. Fig. 7(b) shows the results for the 5-cube and the 6-cube. The effect on overall network latency in shown in Fig. 7(a),(b), where log plots are used to show the separation among curves more clearly. In each cube, b was set to 0, 1, 2, 3 and 4. The $b = 3$ and $b = 4$ curves have not been plotted as they follow the $b = 2$ curve very closely. For all non-saturated data points, the average PE-PE latency increases slightly as the hypercube is scaled commensurate with the logarithmic growth in internode distance. For instance, at $\overline{t_a} = 0.156ms$ and $b = 1$, the PE-PE latency increases from 3.3 μs in the 3-cube, to 3.5 μs in the 4-cube, to 3.6 in the 5-cube, to 3.8 in the 6-cube. At higher message rates, the rate of increase is slightly higher as messages in a larger cube are likely to encounter more contention in their paths. The latency in the 3-cube's $b = 0$ network at $\overline{t_a} = 0.078ms$ is higher than in any of the other cubes, instead of lower as expected, because it is saturated. The $b = 1$ plots in all the cubes saturate at approximately the same message

rate as is evident from Figs. 6 and 7. But, once saturated, I/O response times in the bigger cubes deteriorate more rapidly. Even though the number of requests forwarded to each I/O node in all cubes is statistically equivalent, the I/O networks of the larger cubes behave as if they have been subjected to a heavier workload. Waiting times at the I/O nodes are very sensitive to any increases in network contention during saturation as the network delay is multiplied by the number of requests queued for entry at that node. As larger I/O networks have higher network latencies, their waiting times increase more rapidly.

Figs 6 and 7 also show that the $b = 0$ cube has a better I/O response time than the $b = 1$ cube for all cube sizes. We conclude that the results for the 3, 4, 5 and 6-cubes are qualitatively similar to those obtained with the 7-cube. Consequently, we focussed on the 7-cube for the rest of the experiments.

5.2 Effect of I/O Locality

The most interesting observation from Fig. 8 is that while the I/O read locality significantly affects the performance of a system without an I/O network, it makes little difference to system performance if $b = 2$ or higher. As shown in Fig. 8(a), with increasing locality, PE-PE latency in the control cube decreases steadily as fewer data blocks from remote I/O nodes are released into the "interior" of the cube in response to requests filed at some other I/O node. PE-PE latency then decreases because fewer channels are tied up for long periods of time. Alternately, PE-PE latencies in the test cubes are unaffected by locality as long as they do not saturate. Also, as expected, all systems converge to the same response time when the locality approaches unity. At a locality of 1, no I/O-I/O messages are generated and the I/O networks of the test cubes will not be used. Figure 9 shows the results of the locality study performed at a higher message rate. With this rate, the decrease of inter-I/O traffic with increasing locality brings the unit channel I/O network out of saturation. For both message rates, the $b = 1$ cube still has a higher response time than the $b = 0$ cube.

5.3 Effect of I/O rate

On increasing the I/O request rate, the I/O-I/O latency increases due to higher I/O network utilization (see Fig. 10). Again, a higher bandwidth network will saturate at a higher request rate. The $b = 2$ network only starts to saturate at the highest I/O rate simulated in the study, while the $b = 3$ and $b = 4$ networks are able to supply data to the PE network even at this rate. Also, as in Fig. 4(a) the PE-PE latency for $b = 1$ levels off as the I/O rate increases, because of saturation in the I/O network.

5.4 Decreasing the read ratio

The previous experiments restricted all I/O requests to block reads. For a fixed b and locality, decreasing the read ratio has a qualitatively similar effect on I/O-I/O latency as increasing the locality - the I/O-I/O latency decreases. Since all writes are performed at the local I/O node, a

lower read ratio reduces inter-I/O traffic, in essence causing the I/O system to behave as if its locality had been raised (see Fig. 11). However, the effects on PE-PE latency are different. In this case, lowering the read ratio is accompanied by a slight decrease in PE-PE latency. Though the aggregate amount of I/O-related traffic on the two sets of links arriving at (A) and departing from (D) of the PE-PE router in Fig. 2 remains constant, the ratio of I/O traffic moving through D to the I/O traffic moving through A changes. When the read ratio is 1, every I/O request causes a 32-byte message to be transmitted on one of the A-links on its journey to the local I/O node. In response, a 4096 byte data block is transmitted via one of the D-links. When the read ratio is 0, every I/O request generates a 4128-byte message on one of the A-links and no I/O traffic on the D-links. Thus, as the read ratio is decreased from 1 to 0, I/O-related traffic gradually spreads from the D-links to the A-links. As the ratio approaches 0, all I/O-related traffic will be confined to the A-links. Fig. 11(a) shows that the PE-PE latency is lowest when the data traffic is spread evenly between the A-links and the D-links, and slightly higher when one set of links is more heavily loaded than the other. Fig. 12 shows the effect of locality on I/O response time at various read ratios.

5.5 Coupling of I/O Subsystem with the Compute Nodes

The coupling bandwidth between the two subsystems is determined by n, the channel width of the link departing the I/O router for the PE router, and m, the width of the link departing the local I/O node for the PE router (see Fig. 2). As the locality of requests decreases, more I/O traffic returns to the PE network via the n-channel links and less via the m-channel links. In the previous simulations, the values of n and m were chosen so that these links would not be bottlenecks. Fig. 13(a) and (b) show that a moderate value of n is sufficient to achieve this purpose. There is little difference in performance between $n = 3$ and n arbitrarily large which means that the assumption of $n = 4 * b$ is justified because similar performance can be obtained with a smaller value of n. However, $n = 1$ causes all networks to saturate when the mean inter-arrival time is 78μsecs or less.

For a given b and $\overline{t_a}$ the value of m needed to prevent saturation is also a function of locality. Simulations show that for $\overline{t_a} = 156\mu s$, $m = 1$ is sufficient to prevent saturation at a locality of 1.0. Doubling m to 2 provides a 40% improvement in response time, but further increases provide only marginal improvements. At higher message rates, the value of m needed to prevent saturation is higher, e.g, at $\overline{t_a} = 78\mu s$, m needs to be at least 2 to avoid saturation. Once again, doubling m provides a substantial improvement in response time, particularly when there is high locality.

6 Analytical Studies

The simulation studies yielded some non-intuitive results such as an actual degradation for some cases when the basic ($b = 1$) I/O network was added, and a dramatic improvement for $b = 2$. This motivates the development of an analytical framework to validate the simulation results as well as to obtain a better understanding of the network behavior and tradeoffs involved. Initially, we developed an analysis based on network bisection bandwidths. Note that the ratio of the bisection bandwidths of the PE and IO networks is $\frac{N_{PE}}{N_{IO} \cdot b}$. This indicates that with $b = 1$, it is possible to saturate the

I/O subsystem with a $\frac{N_{IO}}{N_{PE}}$ fraction of the traffic needed to saturate the PE subsystem, so diverting I/O messages from the PE network to the I/O network can actually degrade performance. However, results of detailed analysis using bisection bandwidths did not satisfactorily agree with the simulation. This is not surprising since bisection bandwidth assumes a high-flux scenario with uniform utilization of the bisection channels. However, in our case channels in the first dimension that exit from Class I PEs are utilized more often than other channels. More importantly, this analysis does not account for the blocking nature of wormhole routing. Therefore, a more accurate model was developed, as described below.

An exact analysis of wormhole routing is very involved due to its blocking nature which forces one to use a recursive formulation wherein one calculates the blocking probability for messages only a hop away from their destination, and works backwards [18]. We use a more tractable queuing model by observing that a blocked message can be considered as being queued up at the blocking channel (and associated flit buffer) for analysis purposes, even though different flits are at physically different locations along the path of that message [**?**]. This analogy is exact in the sense that there is no loss of accuracy so long as the waiting times of channels take into consideration blocking delays further down the route of the *blocking* message.

Each single-bandwidth channel in the network is modelled as an M/G/1 queue, while multiple-bandwidth channels are represented by M/M/m queues where m is the bandwidth. The M/M/m queue is used to model multiple-bandwidth channels because there are no closed-form solutions for M/G/m queues. For high k, m, n and r, there is little contention on channels at the node-router interfaces and the interfaces between PE routers and I/O routers. Thus, waiting times for these channels are therefore approximated as zero. The latency penalty due to contention on other channels is found by using the previously mentioned queuing models. The average waiting time in queue for an M/G/1 queue is given by [20]

$$\begin{aligned} W_{M/G/1} &= \frac{\rho\overline{x}(1+C_b^2)}{2(1-\rho)} \\ \rho &= \lambda\overline{x} \\ C_b^2 &= \frac{\sigma_b^2}{\overline{x}^2}, \end{aligned}$$

where λ is the mean arrival rate, $\overline{x}$ is the mean service time, and σ_b^2 is the variance of the service time distribution. The average waiting time for an M/M/m queue is [33]

$$\begin{aligned} W_{M/M/m} &= \frac{\rho(m\rho)^m p_0}{m!(1-\rho)^2\lambda} \\ p_0 &= [\sum_{k=0}^{m-1}\frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!(1-\rho)}]^{-1} \\ \rho &= \frac{\lambda\overline{x}}{m}. \end{aligned}$$

The average latency is given by:

$$\overline{latency} = \sum_{i\in C} P_i(W_i+1)$$

where P_i is the probability that channel i is traversed, W_i is the average waiting time for channel i, and C is the set of all channels in the network.

To calculate waiting time for a channel, the mean arrival rate of messages (λ), and mean service time $\overline{x}$ are required. Additionally, the variance in service times is needed for channels modelled by an M/G/1 queue. The arrival rate of messages on any network channel in an n-dimensional binary hypercube with globally uniform traffic is approximately $\lambda_g/2$, where λ_g is the mean injection rate of messages into the network from a node. This injection rate is equivalent to the generation rate unless the network is saturated. For traffic distributions in which the probability that a message traverses zero hops is zero, the actual message traffic rate on network channels is given by $\lambda_g \frac{2^{n-1}}{2^n-1}$. The service time distribution for a channel can be approximated by the distribution of lengths of messages which traverse the channel, if one neglects the effect on service time of blocking delays on subsequent channels. Thus, this approximation provides an upper bound for λ_{sat}, a value such that the network is guaranteed to be saturated $\forall \lambda \geq \lambda_{sat}$. This saturation point occurs when the utilization factor ρ of any channel exceeds 1. To predict this point more accurately, we formulate an adjusted service time ($\overline{x}_{adj,i}$) for channel i:

$$\overline{x}_{adj,i} = \overline{x}_i + \sum_{j \in S_i} P_j W_j,$$

where S_i is the set of channels which may be traversed after channel i. Observe that this calculation holds only when the length of messages (in flits) is longer than the diameter of the network.

To demonstrate the application of this analytical model, we first consider a 7-dimensional test hypercube coupled with a 4-dimensional I/O network with $b = 1$, $P_l = 0.2$, $P_r = 1.0$, and $P_{io} = 0.05$. Our goal is to approximate I/O-PE latency and find λ_{sat} for the I/O network. I/O-PE latency is considered since it responds to congestion in either of the two subnetworks. There are two types of network channels to be analyzed in this calculation: channels in the I/O network and channels exiting PE routers of Class I PEs, i.e. set D in Figure 2. The arrival rate of messages to channels in the I/O network is given by $\lambda_{IOch} = \lambda_{IO}/2$ where λ_{IO} represents the rate at which traffic is generated at an I/O node and is given by

$$\lambda_{IO} = \lambda_{PE} P_{io} P_r (1 - P_l) \frac{N_{PE}}{N_{IO}} \times 2,$$

where λ_{PE} is the rate at which traffic is generated at a PE node. This expression for λ_{IO} arises because traffic on the I/O network is due only to nonlocal I/O read transactions. The $\frac{N_{PE}}{N_{IO}}$ factor is present because each I/O node services the requests of $\frac{N_{PE}}{N_{IO}}$ PE nodes, while the factor of 2 is needed because each read transaction consists of two messages: a read request and a reply block of data. For the test case, $\lambda_{IOch} = 0.32 \times \lambda_{PE}$. If blocking on subsequent channels is neglected, the mean base service time is 2064 time cycles, where a time cycle is the channel cycle time of 0.05μs, because half of the messages are requests with lengths of 32 and half are block replies of lengths of 4096; $C_b^2 \approx 1$. Adjusted service times for each channel are then found by adding waiting times for channels subsequently traversed to this base service time. For globally uniform message traffic on a binary hypercube, all channels corresponding to the same dimension are statistically equivalent. Therefore, only n_{IO} service time calculations are needed to fully characterize the I/O network, where n_{IO} is the number of dimensions in the I/O network.

The arrival rate of messages to channels exiting Class I PE routers depends on two components: the PE-PE traffic and the I/O-PE traffic. This rate is given by

$$\lambda_{PEch} = \lambda_{PE}(\frac{1 - P_{io}}{2} + P_{io} P_r).$$

The first part of the expression above accounts for the uniform PE-PE traffic, while the last part gives the amount of traffic which results from I/O block replies to read requests. The mean service time is found by multiplying each possible service time by its probability of occurring and is given by

$$\overline{x}_{PEch} = \frac{\lambda_{PE}}{\lambda_{PEch}}(32\frac{1-P_{io}}{2} + 4096P_{io}P_r).$$

Note that the service time for PE-PE communications (32) is merely the message length and neglects the effect of blocking on subsequent channels. This approximation is made because the blocking effect is negligible for other channels when compare to those in set D of Figure 2. However, the probability that these channels are traversed during an average PE-PE communication is small. For this test case, $\overline{x}_{PEch} \approx 419$ time cycles with $C_b^2 \approx 8.1$. Given these values and a specified λ_{PE}, we may now calculate the waiting times for all channels relevant to I/O-PE latency measures and therefore approximate average latency.

The I/O-PE latency is characterized by four types of I/O message traversals, all of which are block replies to I/O read requests:

1. requests satisfied locally and destined for Class I PEs,
2. requests satisfied locally and destined for Class II PEs,
3. requests satisfied remotely and destined for Class I PEs, and
4. requests satisfied remotely and destined for Class II PEs.

Messages in category 1 have a latency of 2 (2 hops, each with negligible contention) and a probability of occurring given by $\frac{N_{IO}}{N_{PE}}P_l$. Messages in category 2 have a latency of $3 + W_{PEch}$ and $\frac{N_{PE}-N_{IO}}{N_{PE}}P_l$ probability of occurring. For category 3 messages, the latency is $3 + \frac{1}{2}\sum_{i=0}^{n_{IO}-1}(W_{IO,i}+1)$ where $W_{IO,i}$ is the waiting time for a channel in dimension i of the I/O network. The factor $\frac{1}{2}$ is present because there is one half probability that a dimension is traversed by a message under uniform traffic conditions. The probability of this type of message is $\frac{N_{IO}}{N_{PE}}(1-P_l)$. Lastly, messages in category 4 are likely to occur $\frac{N_{PE}-N_{IO}}{N_{PE}}(1-P_l)$ of the time and have latency of $3+\frac{1}{2}\sum_{i=0}^{n_{IO}-1}(W_{IO,i}+1)+(W_{PEch}+1)$. Latencies determined from this model were calculated for the test case and agree closely with simulation results.

A similar analysis was performed for $b = 2$. The only difference is that the M/M/2 queue model is used for I/O network channels rather than the M/G/1. Again, the analytical results closely match the simulation results. Plots of values obtained from the analytical model versus simulation for both $b = 1$ and $b = 2$ cases are given in Figure 2. A comparison of the saturation point computed from the model with the range in which the simulation indicates saturation occurs is given in Table 2.

The analysis of the "control" cube without the I/O network is also very involved, since the superimposed I/O traffic leads to hot-spots and traffic that is non-uniform in both destination and size. However, we may still obtain an upper bound for λ_{sat} by observing the traffic patterns on the most utilized channels. In general, the channels exiting a Class I PE router, i.e. channels in the set D of Figure 2, carry the most traffic. Specifically, an outgoing channel of such a router, that also corresponds to the first dimension in the e-cube dimension order will encounter more traffic than

I/O Network Bandwidth	Network Saturation Traffic Rates (message generations/ms per node)	
	Analysis	Simulation Range
$b = 1$	11.3	10 – 12.8
$b = 2$	23.7	22.2 – 25

Table 2: Saturation Point Predictions from Analysis vs. Simulation

other types of channels. By examining the utilization factor (ρ) of such a channel, we can determine for what value of λ_{PE} the utilization factor exceeds 1 and therefore when the saturation point is reached.

The arrival rate of messages on this channel is given by

$$\lambda_{max} = \lambda_{PE}(\frac{1 - P_{io}}{2} + 2\frac{N_{PE}}{N_{IO}}\frac{P_{io}P_r(1 - P_l)}{2} + P_{io}P_rP_l).$$

The first term in the parentheses is due to the PE-PE communication traffic. The second term accounts for remote I/O requests and remote I/O replies. The destination distribution for this portion of the traffic is approximately uniform; thus, half of these requests are injected into the network along the highest dimension. The last term in the parentheses accounts for local I/O replies. The mean service time for this channel is

$$\overline{x}_{max} = \frac{\lambda_{PE}}{\lambda_{max}}(\frac{1}{2}(1 - P_{io} + \frac{N_{PE}}{N_{IO}}P_{io}P_r(1 - P_l))(32) + (\frac{1}{2}\frac{N_{PE}}{N_{IO}}P_{io}P_r(1 - P_l) + P_{io}P_rP_l)(4096)).$$

Observe that blocking delays on subsequent channels are neglected in this service time approximation. For the test case considered, these calculations yield an upper bound for λ_{sat} of 26.3 generations/ms. The simulation indicates the network saturates around 22.2 generations/ms.

By applying the queuing model described above, one can obtain an estimate of the waiting times, latencies and saturation levels. More importantly, this model avoids the high complexity of more detailed analysis of wormhole-type routing [18] and provides a quick overview of the behavior of the coupled networks that have quite different characteristics. For the test case, the model indicates that the $b = 1$ case saturates at a lower traffic rate than the $b = 0$ case, which agrees with simulation results. The model also indicates that the saturation point of the I/O network is greatly affected by the wormhole routing property in which the service time of a channel depends on the waiting times for channels subsequently traversed. In addition, the analysis points out the disparity between traffic rates on channels of Class I and Class II nodes, which indicates that bisection bandwidth analyses are insufficient for accurately comparing the $b = 0$ case with other cases.

7 Concluding Remarks

While off-loading I/O traffic to a low bandwidth I/O network can actually degrade system performance, there is a dramatic improvement if we use an I/O network with links of slightly higher bandwidth than the interprocessor communication links. More significantly, system performance becomes relatively independent of I/O data locality. This key result is obtained from simulation studies and from the queueing theory based analysis which gives accurate results even in a situation when non-uniform traffic and the blocking nature of wormhole routing invalidates the use of simpler, bisection-bandwidth based arguments. The results indicate that provided files are more or less evenly distributed among the I/O nodes, and disk cache hit ratios are maintained, file allocation among I/O nodes is no longer critical. The lessened importance of file allocation facilitates the implementation of a distributed file system on the multicomputer without loss of performance, and simplifies the operating system.

With an independent I/O network, there is no need to use the multicomputer only as a back-end system to a host machine. Since the I/O network can maintain a distributed file system, it is possible to attach user interfaces directly to the I/O nodes, and use the multicomputer as a front-end processor. In the near future, designers will be able to build multicomputer systems with supercomputer performance by using off-the-shelf microprocessors and support hardware [3]. Simultaneous development of the I/O system will be essential to extract optimal performance from such systems.

Further extensions of this work include an efficient implementation of a distributed file system that uses the independent I/O network. In particular, the issues of data allocation, data replication and memory coherency need to be studied. The study of the tradeoffs between fault tolerance, reliability and the size of the I/O network with respect to disk, link and node failures in different multicomputers, needs to be investigated in more detail. Finally, due to the big disparity between access times for primary and secondary memory, disk cache hit rate is still the primary determinant of performance, and efficient cache management [31] and file prefetching techniques [21] are of utmost importance. The independent I/O network provides another way of achieving higher disk cache hit rates through file migration. This avenue can also be studied further.

Acknowledgements: This research was supported in part by an NSF Initiation Grant MIP 9011-787, Texas Advanced Technology Project grant No. 14-9712, and by a Faculty Development Award from TRW Foundation. K. Goveas was supported by an MCD Fellowship. We thank A. Reddy and P. Banerjee for helpful comments. Please contact ghosh@pine.ece.utexas.edu for comments or reprint requests.

References

[1] B. Agarwal. Parallel I/O subsystems for multicomputers. Master's thesis, Department of ECE, University of Texas at Austin, May 1990.

[2] J. Akella and D. P. Siewiorek. Modelling and measurement of the impact of input/output on system performance. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 390–399, 1991.

[3] W. C. Athas and C. Seitz. Multicomputers: message-passing concurrent computers. *IEEE Computer*, pages 9–25, August 1988.

[4] *Computer Architecture News*, September 1989. Special issue on I/O Architecture.

[5] Intel Corporation. iPSC/2 I/O facilities. Order number 280120-001, 1988.

[6] Intel Corporation. Paragon XP/S Product Overview. Portland, OR, 1991.

[7] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[8] P.C. Dibble and M.L. Scott. Beyond striping: the Bridge multiprocessor file system. *Computer Architecture News*, 17(5):32–39, September 1989.

[9] J. Draper, J. Ghosh, and W.C. Athas. The M-Cache: a message-retrieving mechanism for multicomputer systems. In *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, pages 258–265, December 1991.

[10] G. Fox et al. *Solving Problems on Concurrent Processors (I): General Techniques and Regular Problems*. Prentice-Hall, 1988.

[11] J. Ghosh and B. Agarwal. Parallel I/O subsystems for hypercube multicomputers. In *Proceedings of the Fifth International Parallel Processing Symposium*, pages 381–384, May 1991.

[12] S. Goldstein. Storage performance – an eight year outlook. Technical Report TR 03.308, IBM Corp., San Jose, 1987.

[13] H. Hadimioglu and R. J. Flynn. The architectural design of a tightly-coupled hypercube file system. In *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, pages 147–150, 1989.

[14] K. Hwang and J. Ghosh. Hypernet: A communication-efficient architecture for constructing massively parallel computers. *IEEE Trans. Computers*, C-36:1450–1466, Dec. 1987.

[15] R. Katz, G. Gibson, and D. Patterson. Disk system architectures for high performance computing. In *Proceedings IEEE*, pages 1842–1858, December 1989.

[16] R. H. Katz, J. K. Ousterhout, D. A. Patterson, and M. R. Stonebraker. A project on high performance I/O subsystems. *IEEE Database Engineering Bulletin*, 11(1):40–47, March 1988.

[17] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[18] J. Kim and C.R. Das. Modelling wormhole routing in a hypercube. In *International Conference on Distributed Computing Systems*, pages 386–393, May 1991.

[19] M. Y. Kim. Synchronized disk interleaving. *IEEE Transactions on Computers*, C-35(11):978–988, November 1986.

[20] L. Kleinrock. *Queueing Systems*. John Wiley & Sons, 1975.

[21] D.F. Kotz and C.S. Ellis. Prefetching in file systems for MIMD multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, pages 218–230, 1990.

[22] H.T. Kung. Memory requirements for balanced computer architecture. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pages 49–54, 1986.

[23] H.T. Kung. Network-based multicomputers: redefining high performance computing in the 1990s. In *Decennial Caltech Conference on VLSI*, 1989.

[24] Thinking Machines. Connection machine model CM-2 technical summary. Technical Report HA87-4, Thinking Machines, Inc., April 1987.

[25] U. Nagaraj, U.S. Shukla, and A. Paulraj. Design and evaluation of a high performance file system for message-passing parallel computers. In *Fifth International Parallel Processing Symposium*, pages 549–554, May 1991.

[26] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *ACM SIGMOD Conference '88*, pages 109–116, June 1988.

[27] P. Pierce. A concurrent file system for a highly parallel mass storage subsystem. In *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, pages 155–160, 1989.

[28] T.W. Pratt, J.C. French, P.M. Dickens, and S.A. Janet, Jr. A comparison of the architecture and performance of two parallel file systems. In *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, pages 161–166, March 1989.

[29] A. Reddy and P. Banerjee. Design, analysis and simulation of I/O architectures for hypercube multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):140–151, April 1990.

[30] A. Reddy, P. Banerjee, and S. Abraham. I/O embedding in hypercubes. In *Proceedings of the 1988 International Conference on Parallel Processing*, volume 1, pages 331–338, 1988.

[31] A.L.N. Reddy. A study of I/O system organizations. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 308–317, 1992.

[32] D. A. Reed and R. M. Fujimoto. *Multicomputer Networks: Message-Based Parallel Processing*. The MIT Press, 1987.

[33] T. Saaty. *Elements of Queueing Theory with Applications*. McGraw-Hill, 1961.

[34] K. Salem and H. Garcia-Molina. Disk striping. In *IEEE 1986 Conference on Data Engineering*, pages 336–342, 1986.

[35] M. Smotherman. A sequencing-based taxonomy of I/O systems and review of historical machines. *Computer Architecture News*, 17(5):5–15, September 1989.

[36] J. Voelcker. Peripherals: Higher capacity in smaller packages. *IEEE Spectrum*, pages 28–30, February 1990.

[37] A. Witkowski, K. Chandrakumar, and G. Macchio. Concurrent I/O system for the hypercube multiprocessor. In *Proceedings of the Third Conference on Hypercubes, Concurrent Computers, and Applications*, pages 1398–407, 1988.

List of Figure Titles

Figure 1: A 4-dimensional hypercube augmented with an I/O network.

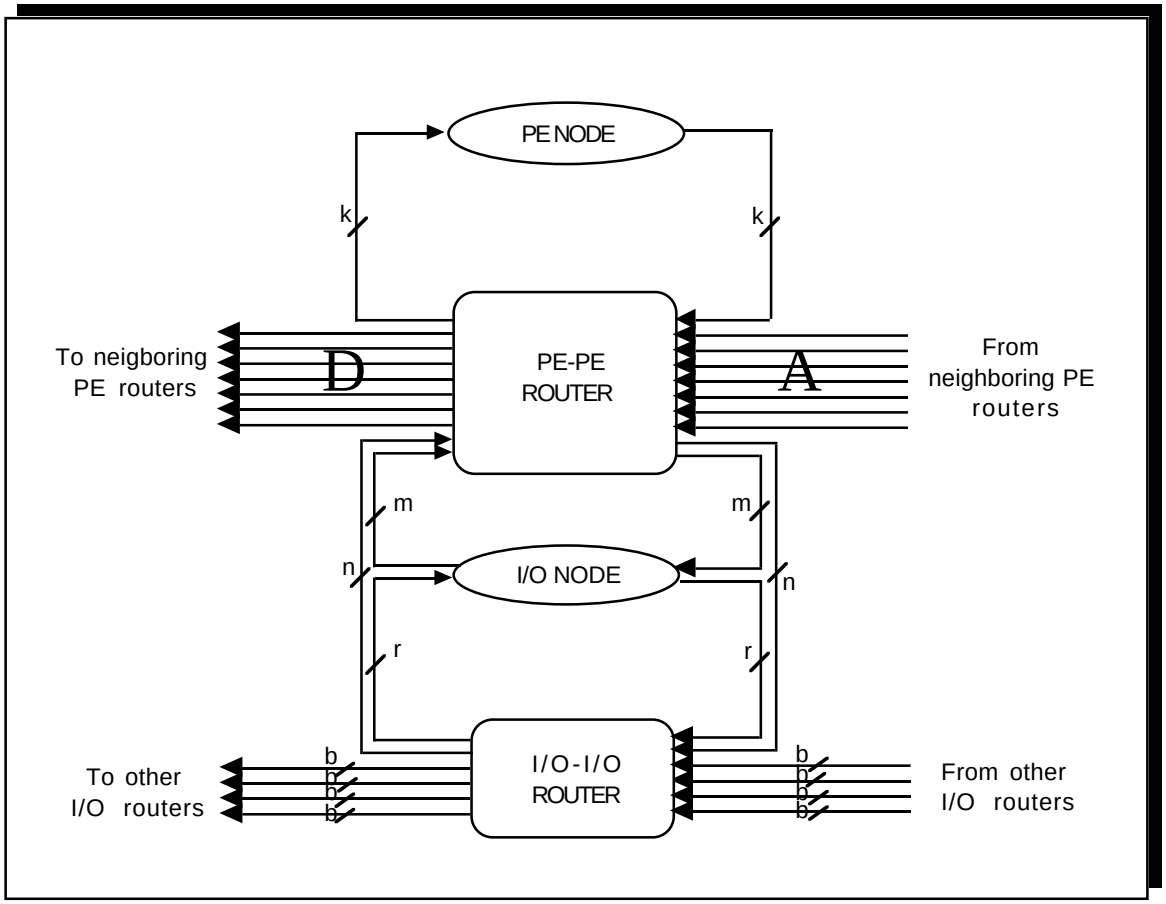


Figure 2: Network interface of an I/O node attached to a PE node.

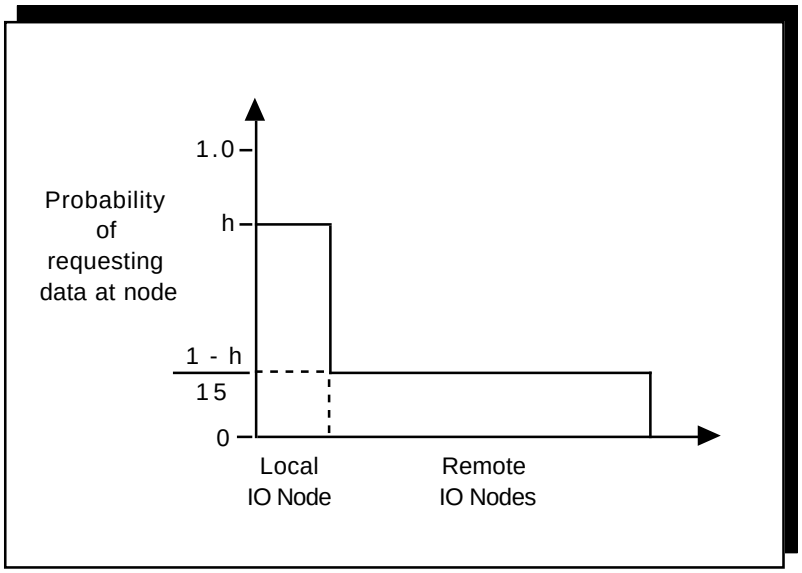
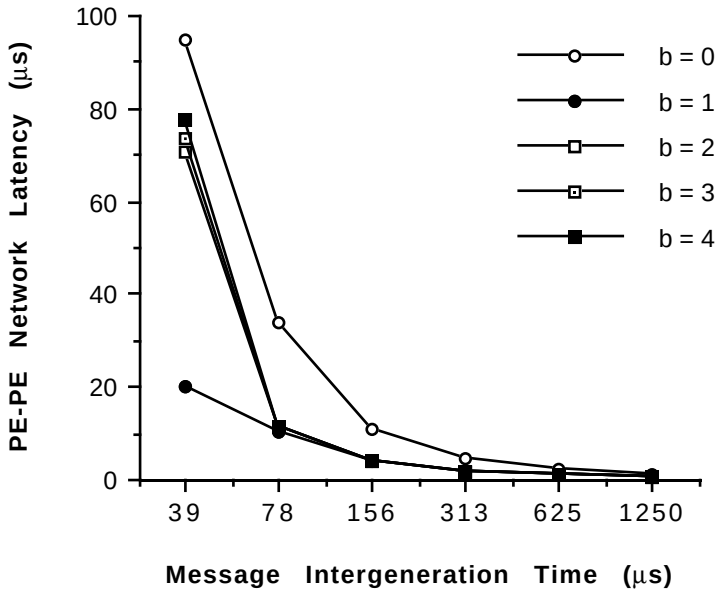
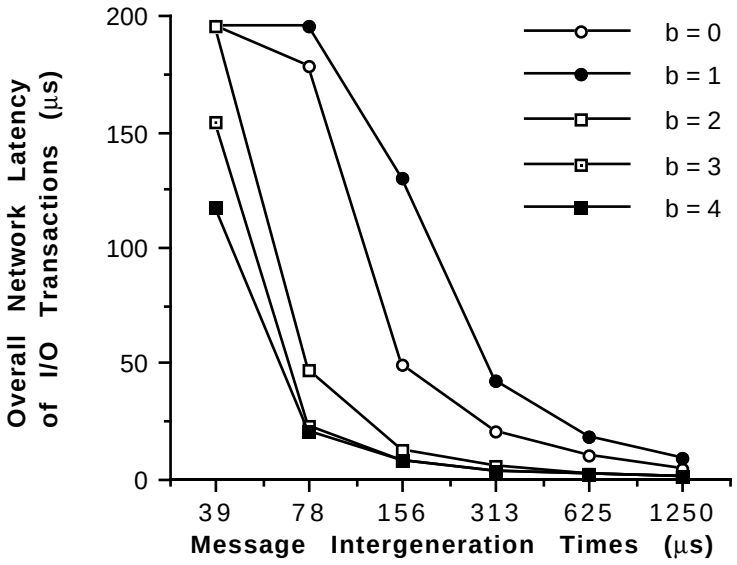


Figure 3: Model of locality for I/O reads.

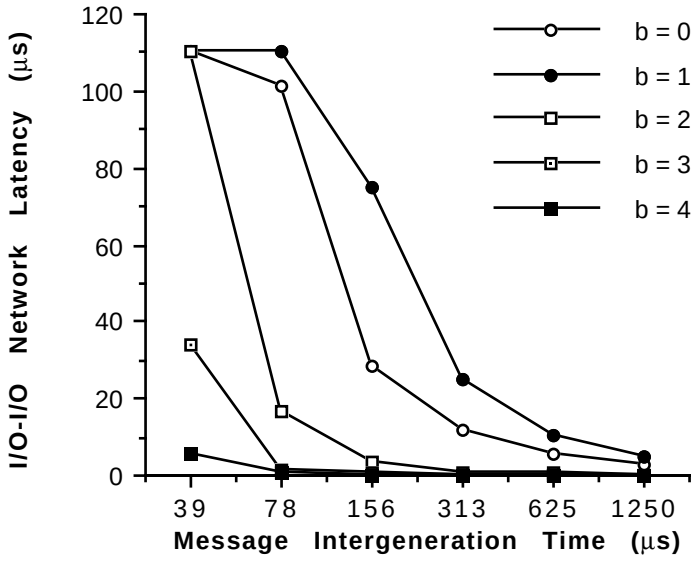


(a)

Figure 4: Effect of network traffic with 5% I/O requests on: (a) PE-PE network latency.



(b)



(c)

Figure 4: Effect of network traffic with 5% I/O requests on: (b) Overall network latency for I/O transactions and (c) I/O-I/O network latency.

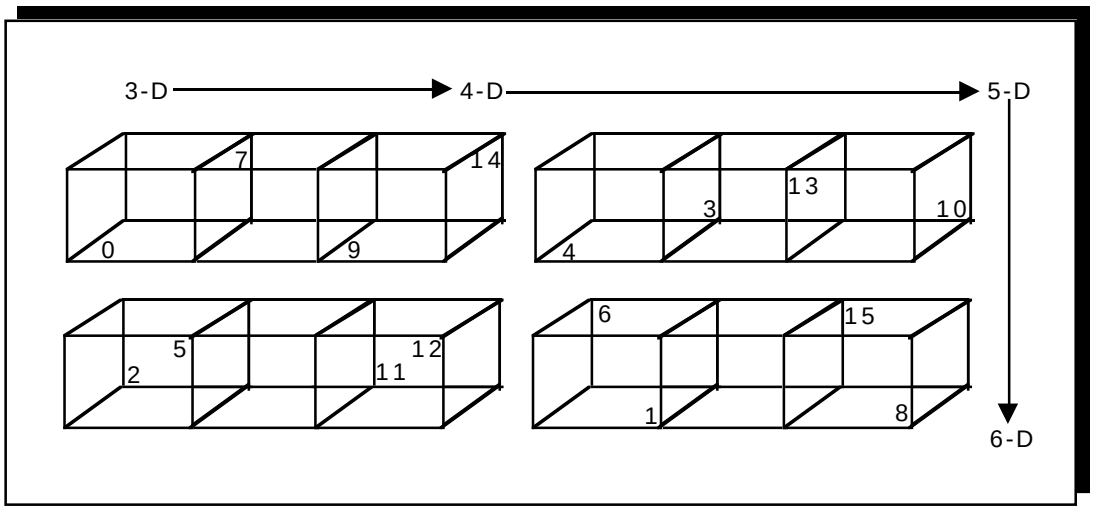
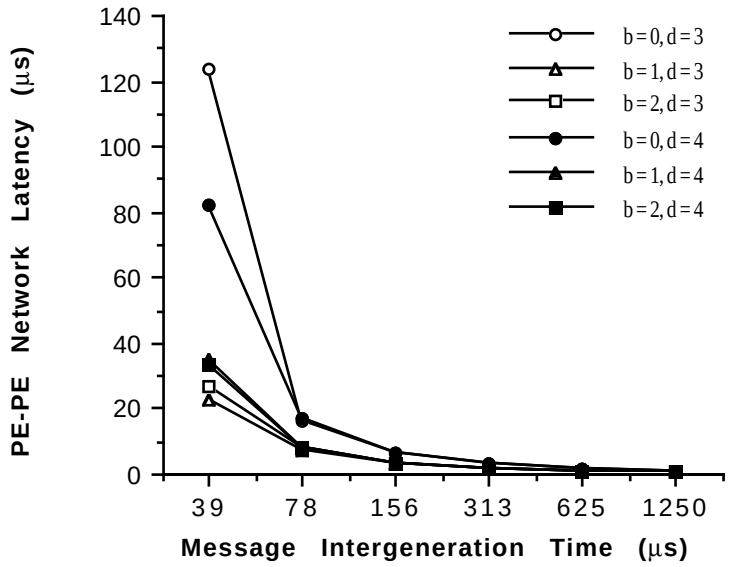
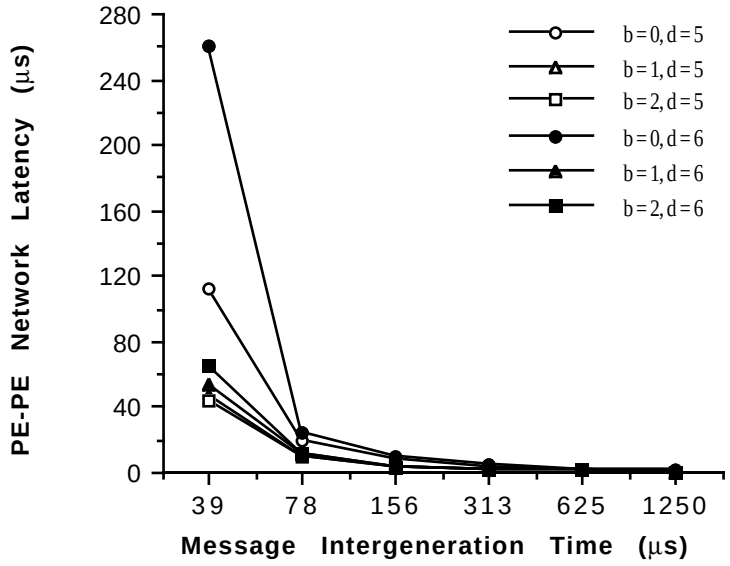


Figure 5: Allocation of I/O nodes in a 3, 4, 5 and 6-d hypercube.

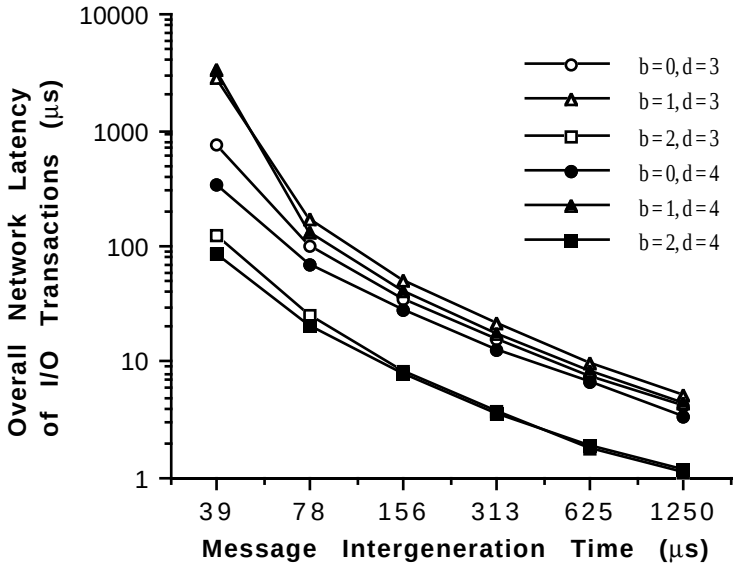


(a)

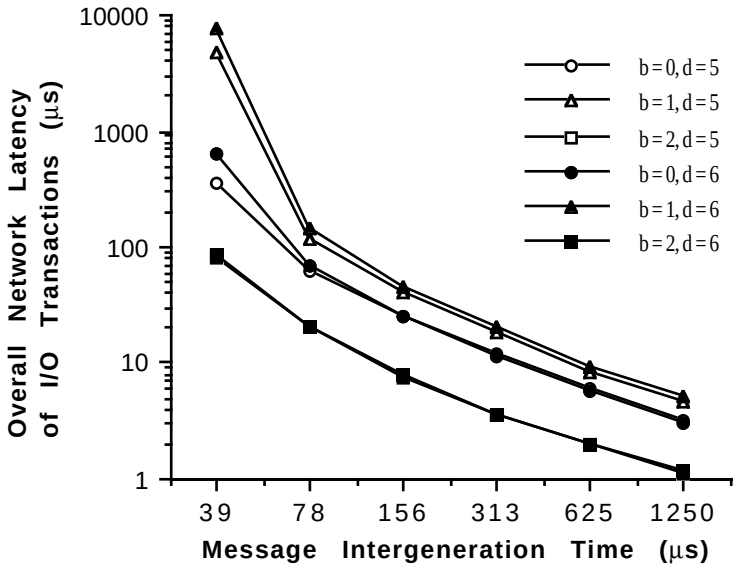


(b)

Figure 6: Effect of network traffic with 5% I/O requests on PE-PE network latency in: (a) the 3 and 4-d cubes, and in (b) the 5 and 6-d cubes.

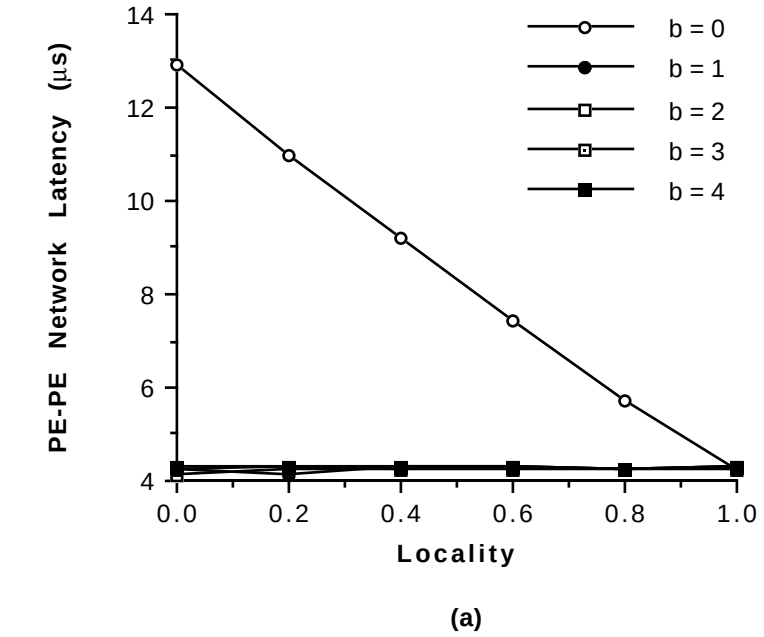


(a)

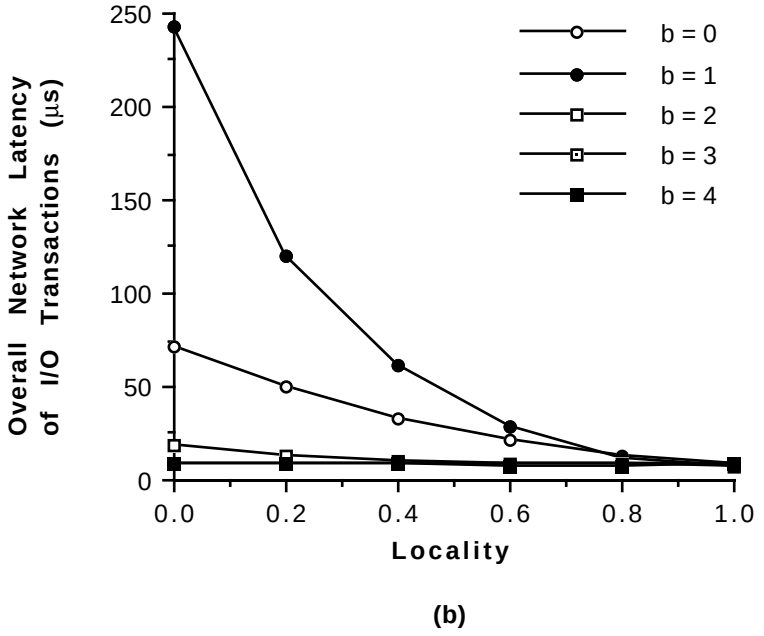


(b)

Figure 7: Effect of network traffic with 5% I/O requests on overall network latency for I/O transactions in: (a) the 3 and 4-d cubes, and in (b) the 5 and 6-d cubes.



(a)



(b)

Figure 8: Effect of I/O read locality on: (a) PE-PE network latency and (b) Overall network latency for I/O transactions, when $\overline{t_a} = 156\mu s$.

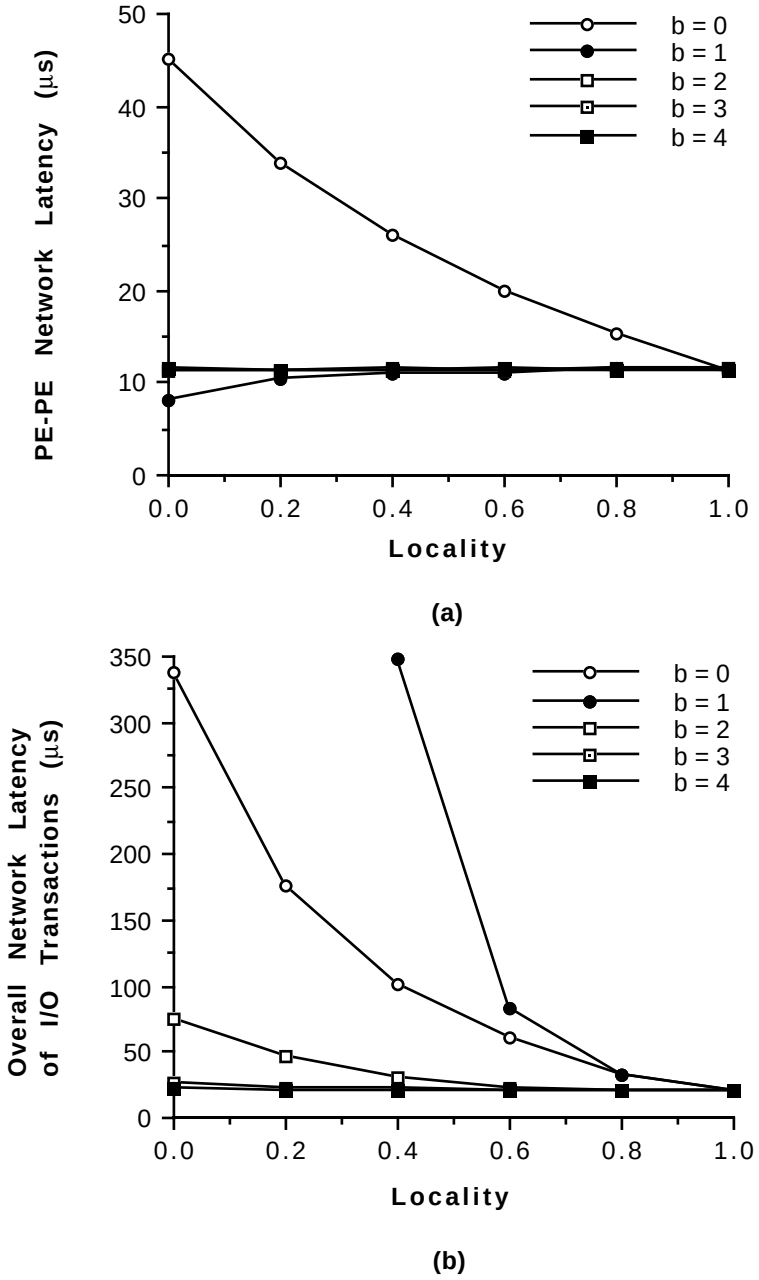


Figure 9: Effect of I/O read locality on: (a) PE-PE network latency and (b) Overall network latency for I/O transactions, when $\overline{t_a} = 78\mu s$.

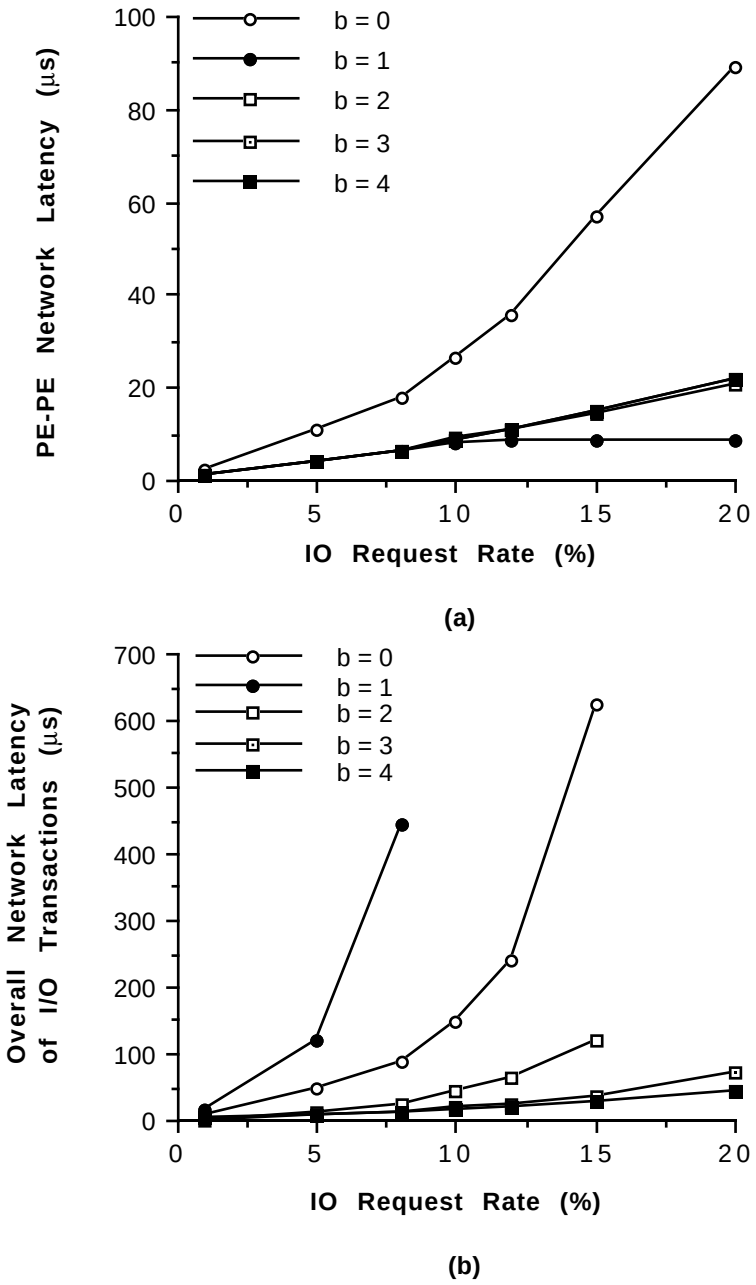


Figure 10: Effect of changing the proportion of I/O traffic on: (a) PE-PE network latency and (b) Overall network latency for I/O transactions, when $\overline{t_a} = 156\mu s$, locality = 0.2.

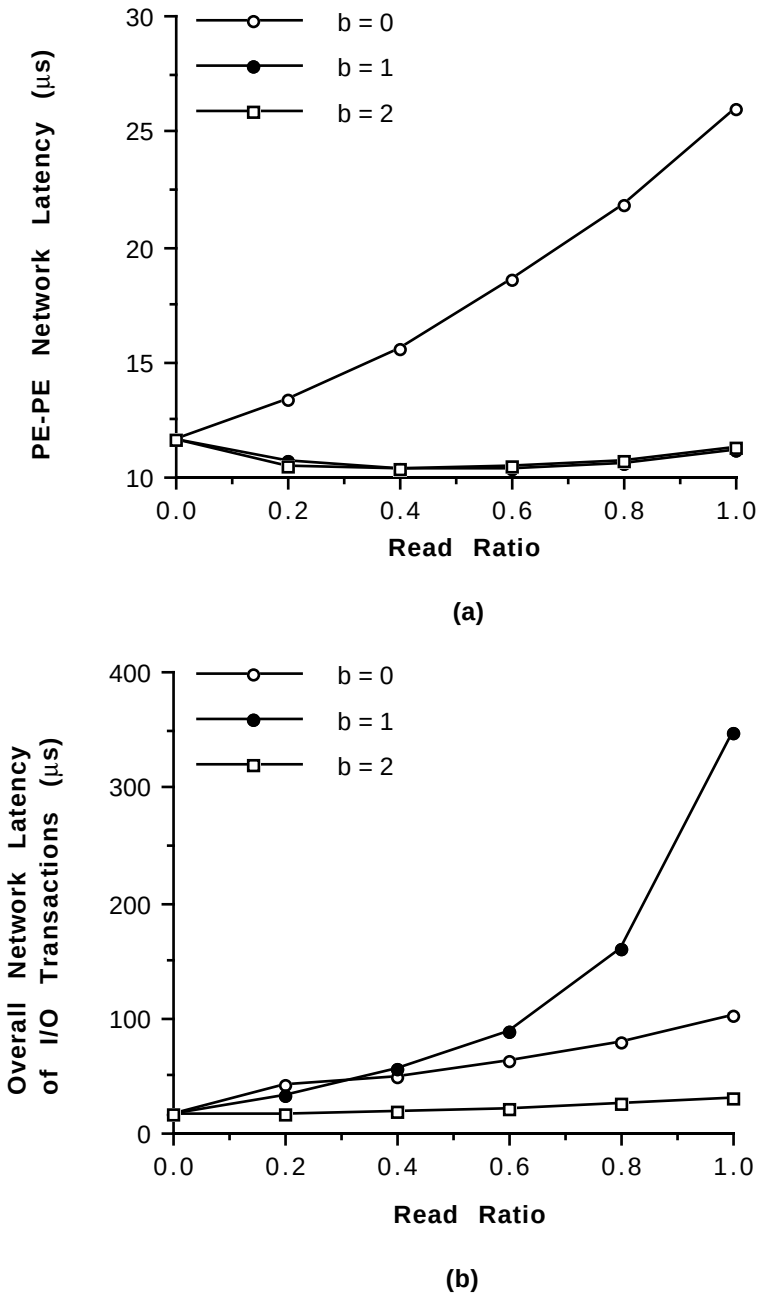


Figure 11: Effect of changing the proportion of I/O reads to writes on: (a) PE-PE network latency and (b) Overall network latency for I/O transactions, when $\overline{t_a} = 78\mu s$, locality = 0.4.

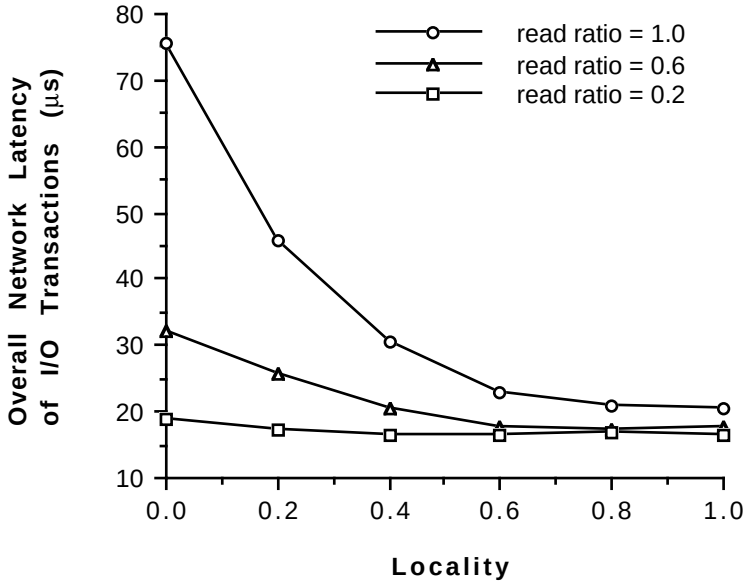


Figure 12: Impact of data locality on overall network latency for I/O transactions at various read ratios; ($b = 2, \overline{t_a} = 78\mu s$).

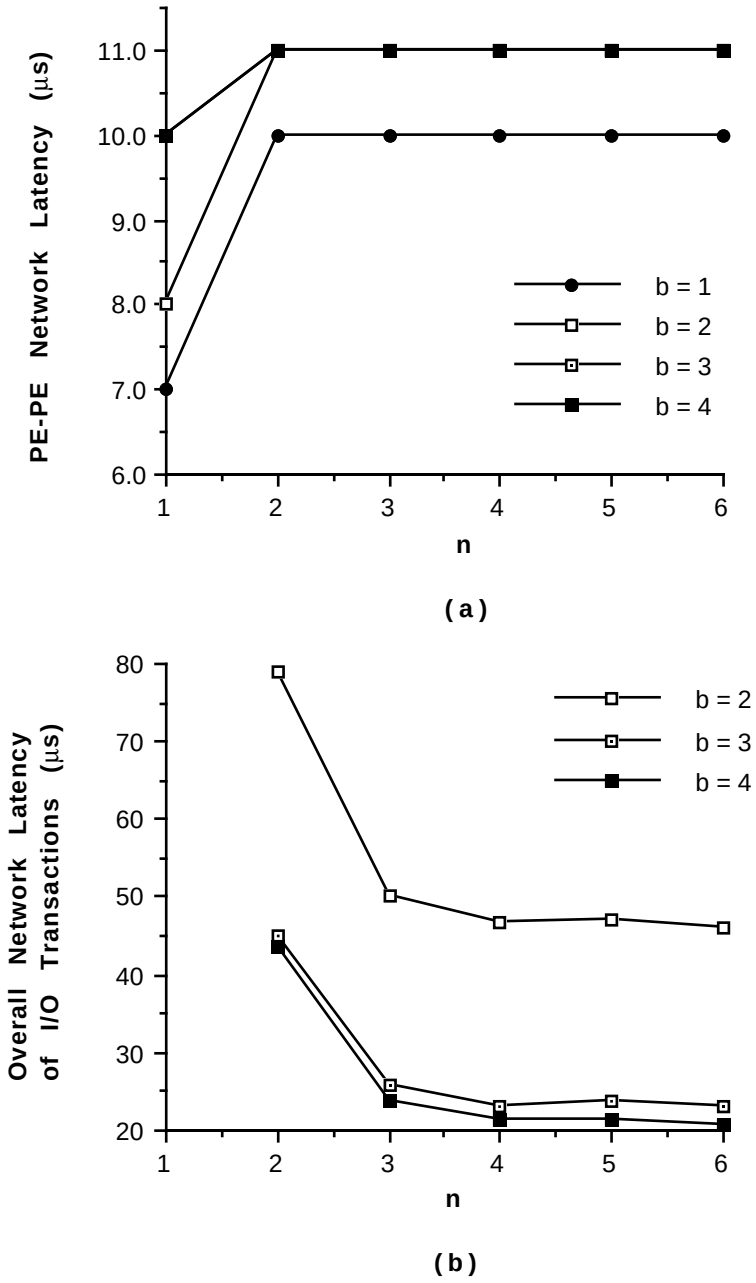


Figure 13: Effect of increasing n on: (a) PE-PE network latency and (b) Overall network latency for I/O transactions when $\overline{t_a} = 78\mu s$, locality = 0.2 and read ratio = 1.

Figure 14: Latency Predictions from Analysis vs. Simulation