IEEE Transactions on Computers 93

Performance Evaluation of Hierarchical Ring-Based Shared Memory Multiprocessors*

Mark Holliday
Dept. of Computer Science
Duke University, Durham, NC 27706

Michael Stumm
Dept. of Elec. and Comp. Eng.
Univ. of Toronto, Toronto, Canada M5S 1A4

November 1992 (revised April 1993)

Abstract

This paper investigates the performance of word-packet, slotted unidirectional ring-based hierarchical direct networks in the context of large-scale shared memory multiprocessors. Slotted unidirectional rings are attractive because their electrical characteristics and simple interfaces allow for fast cycle times and large bandwidths. For large-scale systems, it is necessary to use multiple rings for increased aggregate bandwidth. Hierarchies are attractive because the topology ensures unique paths between nodes, simple node interfaces and simple inter-ring connections.

To ensure that a realistic region of the design space is examined, the architecture of the network used in the Hector prototype is adopted as the initial design point. A simulator of that architecture has been developed and validated with measurements from the prototype. The system and workload parameterization reflects conditions expected in the near future.

The results of our study show the importance of system balance on performance. Large-scale systems inherently have large communication delays for distant accesses, so processor efficiency will be low, unless the processors can operate with multiple outstanding transactions using techniques such as prefetching, asynchronous writes and multiple hardware contexts. However with multiple outstanding transactions and only one memory bank per processing module, memory quickly saturates. Memory saturation can be alleviated by having multiple memory banks per processing module, but this shifts the bottleneck to the ring subsystem. While the topology of the ring hierarchy affects performance, the ring subsystem will inherently limit the throughput of the system. Hence increasing the number of outstanding transactions per processor beyond a certain point only has a limiting effect on performance, since it causes some of the rings to become congested. An adaptive maximum number of outstanding transactions appears necessary to adjust for the appropriate tradeoff between concurrency and contention as the communication locality changes. We show the relationships between processor, ring and memory speeds, and their effects on performance. Using block transfers for prefetching seems unlikely to be advantageous in that the improvement in the cache hit ratio needed to compensate for the increased network utilization is substantial.

Keywords: communication locality; hierarchical ring-based networks; hot spots; large scale parallel systems; memory banks; performance evaluation; prefetching; shared memory multiprocessors; simulation.

*To appear in IEEE Trans. on Computers; This research was supported in part by NSF grant CCR-9113170, by the Natural Sciences and Engineering Research Council of Canada, and by the Information Technology Research Center of Ontario.

1 Introduction

In this paper we study the performance of large-scale, shared-memory multiprocessors that use a word-packet, ring-based hierarchical network. This class of architectures is of interest for several reasons. First, by distributing the shared memory among the processor modules, associating caches with each processor module, and locating the processor modules at the nodes of the network (that is, using a *direct* network), communication locality is exploited to reduce network traffic and memory latency.

Second, bit-parallel, unidirectional, slotted rings have been found to be effective at maximizing link bandwidth in direct networks [26]. The advantages of unidirectional rings include: 1) with their point-to-point connections, they can run at high clock speeds, 2) it is easy to make full use of their bandwidth, 3) they provide a natural broadcast mechanism, and 4) they allow easy addition of extra nodes.

Third, a single slotted ring does not scale well, so multiple rings need to be interconnected. A hierarchical ring interconnection is attractive since it allows simple node interfaces and inter-ring connections. A node need only interface with its two ring neighbors. All inter-ring connections (regardless of system size) can be implemented using a two-by-two crossbar switch. Moreover, a hierarchy provides a unique path between any two nodes, which can be useful in the implementation of some cache consistency protocols [13]. A disadvantage of a hierarchy is the limited bandwidth near the root. However, this disadvantage is mitigated when there is sufficient communication locality.

Our interest lies in the performance of this type of network within the context of a shared memory multiprocessor, not just in isolation. Consequently, in evaluating overall system performance, the effects of memory cycle time, memory utilization, and aspects of the processor design that effect the request rate need to be considered. Specific issues of interest are the effectiveness of techniques to hide memory latency such as multiple outstanding transactions and non-blocking reads, the use of memory banks, ring topology, communication locality, hot spots, and the relative speeds of the processors, memories, and rings. The transactions we consider are memory reads and writes of single words and block transfers.

Our approach in evaluating these issues is to accurately simulate an existing system using a detailed, packet-level simulator that can be validated against the existing system. For this purpose, we use the Hector prototype which is a multiprocessor of this class of architecture. Since we are interested in the performance of systems with on the order of 1024 processors, and since it is not clear how to extrapolate results from small systems, we have found it necessary to use synthetic workloads. Simulating instruction execution (as with Tango [12]) for a large system would take prohibitively long, and using address traces from other systems is highly questionable.

The results of our study show the importance of system balance on performance. Large-scale systems inherently have large communication delays for distant accesses, so processor efficiency will be low, unless the processors can operate with multiple outstanding transactions using techniques such as prefetching, asynchronous writes and multiple hardware contexts. However with multiple outstanding transactions and only one memory bank per processing module, memory quickly saturates. Memory saturation can be alleviated by having multiple memory banks per processing module, but this shifts the bottleneck to the ring subsystem. While the topology of the ring hierarchy affects performance — we show that topologies with a similar branching factor at all levels, except, possibly, for slightly smaller rings at the root of the hierarchy tend to perform best — the ring subsystem will inherently limit the throughput of the system. Hence increasing the number of outstanding transactions per processor beyond a certain point only has a limiting effect on performance, since it causes some of the rings to become congested. An adaptive maximum number of outstanding transactions appears necessary to adjust for the appropriate tradeoff between concurrency and contention as the communication locality changes. We show the relationships between processor, ring and memory speeds, and their effects on performance.

In the next section we describe in more detail the systems we are examining, the simulation methodology, the system parameters, and the workload parameters. The experimental results are reported in Section three, and we conclude in Section four, together with a discussion of related work.

2 System and Workload Description

We have chosen the Hector architecture, developed at the University of Toronto, as the initial design point for our study. because it was designed specifically for ring-base hierarchies and was implemented successfully. We choose to start from a design that has actually been implemented for two reasons. First, basing the study on an implementation helps to ensure that the performance of all system modules and their interactions are correctly captured; a more abstract system model might miss some of these. Second, by restricting the study to designs related to a carefully thought out implementation, we are focusing our attention on a realistic section of the design space and one we believe to be relatively promising with respect to scalability. We briefly describe Hector below; a more detailed presentation is given in [29, 27]. We then comment on the simulator, the system parameters, and the workload parameters.

2.1 The Hector Architecture

Hector is a shared-memory multiprocessor consisting of a set of *stations* that are interconnected by a hierarchy of unidirectional rings. Each station contains a collection of processor modules containing a processor, a local cache, and part of the main memory. A station is connected to the lowest level (or *local*) ring. Hector provides a flat, global (physical) address space, and each station is assigned a unique contiguous portion of that address space, determined by its location. All processors can transparently access all memory locations in the system. Information transfer takes place by means of fixed-size packets transferred in a bit-parallel format along a unique path through the ring hierarchy.

Two types of ring interfaces control packet transfers, both of which are simple to realize. *Station controllers* control on-station traffic as well as local ring traffic at the station. They gate incoming packets from the ring onto the station, outgoing packets from the station onto the ring, and continuing packets from the previous ring interface on to the next ring interface. Packets on the ring have priority over packets from the station to minimize the time packets are buffered at the station controllers. *Inter-ring interfaces* control traffic between two rings. Logically, an inter-ring interface corresponds to a 2×2 crossbar switch with FIFO buffers. FIFO's are needed in order to be able to store packets if *collisions* occur, which can happen when, in a given cycle, input packets from both rings are to be routed to the same output. In order to minimize the remaining delay of packets that are descending the hierarchy, packets from the higher-level ring have priority over packets from the lower-level ring.

Figure 1 depicts a Hector configuration with two levels in the ring hierarchy, where a global ring connects several local rings that in turn connect multiple stations. In this example, which corresponds to a prototype, a bus connects several processing modules to the local ring interface.

All communication in Hector occurs synchronously. During a given clock cycle, a packet can be transferred between two adjacent ring interfaces, from a station controller onto the station, or from the station onto the ring. A request for access to a non-local memory location initiates a

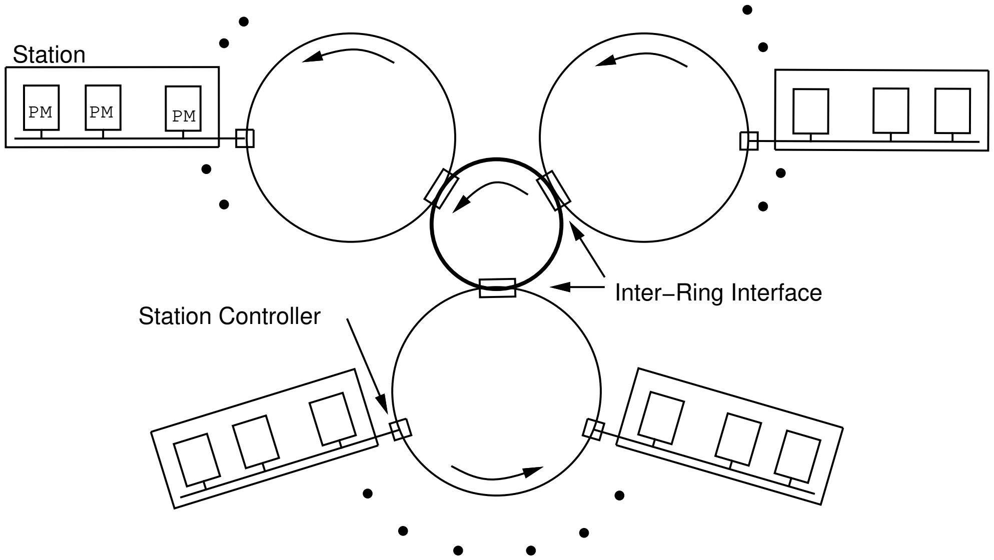


Figure 1: Structure of Hector with two levels of ring hierarchy.

packet transfer across the network. Following the terminology of the Scalable Coherent Interface protocol [22], each transaction involves a request and response subtransaction. A subtransaction typically entails the transmission of a single packet, but in the event of collisions and timeouts, several packets may be used.

For example, to access a remote memory location, a *request* packet containing the address of the target memory is formed in a processing module and transferred to the ring via the local station controller. The packet then travels around the ring visiting one segment in each ring cycle. When the packet reaches the first inter-ring interface on its path, it is either switched onto a higher level ring, or passed on to the next station controller on the same ring, depending on the destination address. The packet first travels up the hierarchy to the level needed to reach the target station, and then descends the hierarchy to the target station where it is removed from the local ring.

The target station sends a *response* packet back to the requesting PM, along a similar path. In the case of a read transaction, the response packet contains the requested data. In the case of a write transaction, the request packet contains data in addition to the addressing information, and the response packet contains an acknowledgment. For writes, the response packet is sent back to the requesting station as soon as the write is queued at the target memory, so the latency of the actual memory operation is hidden.

It is possible that a request packet cannot be successfully delivered to the target memory. This can happen, for example, when there is congestion at the target memory and it cannot accept a further request when the packet arrives. In this case, the target station generates a negative acknowledgement packet, which is sent back to the requesting station so that it can retry the operation at a later time by retransmitting the request packet.

2.2 Simulator

We constructed a simulator that reflects the behavior of the packets on a cycle-by-cycle basis. The simulator was written using the *smpl* [19] simulation library. The batch means method of output analysis was used with the first batch discarded to account for initialization bias. In the batch means method a single long run is divided into subruns called *batches*. A separate sample mean is computed for each batch. These batch means are then used to compute the grand mean and confidence interval [19]. The batch termination criterion was that each processor had to complete at least some minimum number of requests. Early experiments showed that using a total number of requests completed over the entire system as the batch termination criterion can substantially underestimate mean response times since requests with long response times are underrepresented.

Using several workloads we validated the simulator from measurements collected on the Hector prototype. The base simulator was then extended to model features not present in the prototype, such as an arbitrary number of ring levels and ring cycle times different than the processor cycle time.

2.3 System Parameters

A hierarchical ring-based system can be characterized by the following parameters: system size (in processors), the relative processor, memory, and ring cycle times, the maximum number and type of transactions that a processor may have outstanding, whether a processors blocks until a read completes, ring topology, and the number of banks in each memory module.

Transaction latency refers to the entire time from when the request packet of a transaction is issued by a processor until the transaction *completes* (that is, until the response packet returns to the processor [1]). There is a base, or *contention-free*, fraction of the transaction latency which is the number of cycles required to traverse the network twice and, for reads, the time to actually execute the memory operation in the absence of contention. The remaining fraction of the transaction latency is the number of additional cycles due to contention.

Exposed transaction latency is the fraction of the transaction latency during which the processor is blocked waiting for the transaction to complete. Thus, a memory cycle time of 30 processor cycles may imply a transaction latency of 100 processor cycles in a large system when the target memory is far from the source processor. *Processor efficiency* is the fraction of time a processor spends doing useful work averaged over all processors. Useful work includes the delay for references that are cache hits (we assume each processor has a local cache), but does not include the additional delay for cache misses.

If the processor blocks until each read or write completes, then the large transaction latency relative to processor cycle time implies a low processor efficiency. A number of techniques have been proposed to increase processor utilization over this base case, including relaxed memory consistency models, prefetching, and multiple hardware contexts [16]. Instead of assuming one technique versus another, we characterize their effects by varying the maximum number of outstanding transactions a processor may have before blocking and by considering whether or not reads block. Thus, a processor does not block until either the number of its outstanding transactions has exceeded the maximum or, if reads block, a read cache miss occurs. The goal of both multiple outstanding transactions and non-blocking reads is to hide exposed transaction latency. One of the most effective methods of allowing non-blocking reads and multiple outstanding transactions is to use multiple hardware contexts. When we consider the case of multiple outstanding transactions, we intentionally do not take into account any cycles lost due to hardware context switches. This assumption is based on recent work that suggests such context switches can be scheduled so as to avoid any lost cycles [7].

To prevent network saturation we consider the implications of alternative ring topologies. The system topology can be specified by the branching factor at each level of the hierarchy starting at

[1] For writes the response packet returns upon queueing of the request at the target memory so it is possible for the target memory to be still processing a write request after the transaction completes in the above sense.

the number of stations on each local (or level 1) ring and with the last branching factor being the number of ring directly attached to the root ring. Thus, a topology $L = (2, 3, 4)$ refers to a topology with 2 stations per level 1 ring, 3 level 1 rings per level 2 ring, and 4 level 2 rings connected by the root ring. Throughout the paper we assume one processor per station.

To prevent memory saturation we consider the use of multiple memory banks per memory module. We assume that a transaction for a particular target memory is equally likely to access any of the banks at that memory.

2.4 Workload Parameters

A detailed system simulator is needed in order to ensure that the important features of the architecture being studied are captured. Simulating a large system is also important since extrapolating from the results of a small system is questionable. The key to satisfying both of these concerns is to use a synthetic workload model. Simulating instruction execution for a large system (as with Tango) would take prohibitively long, and using address traces from other, smaller systems is highly questionable. In contrast, with a synthetic workload model, the number of transactions that need to be issued by each processor in order to obtain system performance measures is dramatically smaller than the number generated when simulating the execution of actual application programs. Moreover, the use of a synthetic workload sometimes allows clearer understanding of the significance of different workload parameters. Of course, the concern that has to be addressed is the realism of the workload model.

Our approach is to characterize the workload by the mean time between cache misses given a non-blocked processor (or equivalently by the *request rate* which is the inverse of the mean time), the probability that the cache miss is a read, and the communication locality. For the read cache miss probability we assume 0.7 throughout the study which is consistent with empirical statistics [15].

For the request rate we consider a rate of 0.01 to 0.05 cache misses per processor cycle (equivalently, 20 to 100 cycles between cache misses). This range choice is supported by a recent study of a number of application programs that observed a mean number of processor cycles of between 6 and 137 for shared data reads [7]. We assume that code and private data references (such as to a stack) always hit in the local cache. Factoring in shared data writes and shared data cache hits, yields a more realistic mean number of processor cycles between cache misses of at least 20.

We chose for our workload characterization to avoid accounting (at least explicitly) for cache coherence traffic. Cache coherence traffic could be included within a low-level workload model such as ours by providing a translation from a high-level workload model to a low-level workload model. We did a preliminary study of the effect of such a translation for the case of software cache coherence using the approach developed by Adve, et.al. [1]. The resulting ranges for the low-level workload parameters were consistent with the ranges we consider [2].

Most of the results presented in this paper assume one word transfers. Transfers larger than a word can arise from several sources: page migrations and replications, cache line transfers, and prefetching. With regard to page migrations and replications, parameter values greatly effect the results. Since there are no clear value ranges to use, we chose to ignore them. For cache line transfers and prefetching of adjacent words on a cache miss we consider the use of page-mode DRAM access to transfer blocks of words. Thus, for example, upon a memory access, the first word is provided by the memory after, say, 30 processor cycles, and successive words might be provided at intervals of between 5 to 10 processor cycles.

[2]The low-level software cache coherence model includes traffic due to *posts* and *invalidates* of cache lines. Assuming single word cache lines, it is possible to include this traffic within the read and write parameters.

2.5 Communication Locality and Hot Spots

In a direct network, communication locality and hot spots can greatly affect performance. Our *clusters of locality* model attempts to model communication locality. Intuitively, this model, for each processor, logically organizes all processors into clusters around that processor independent of the network topology. The first cluster typically only contains the processor itself; the second cluster contains "near-by" processors; additional clusters contain processors further away. In our case, we view the processors as the leaves of the tree defined by the ring hierarchy and number them left to right. The clusters are then defined in terms of distance between two processors, which is the absolute difference (modulo the size of the system) between the two processor numbers. For example, in the two cluster case, cluster 0 may be the source processor module itself and cluster 1 all of the remaining processor modules. In the three cluster case, cluster 0 may be the source processor module itself, cluster 1 the source processor module's set of "closest neighbors", and cluster 2 the remaining processors modules. Defining clusters in this manner is reasonable since it is likely that applications will be programmed in a manner independent of the particular branching factors present in a certain ring topology.

The probability that the target memory of a transaction is in a processor module of a particular cluster depends on the cluster. Given that the target memory is in a particular cluster, the probability of a processor module within that cluster containing the target memory is uniformly distributed.

The communication locality model specifies the number of clusters, the number in each cluster, and the probability of a transaction's target being in each cluster. Thus, if there are 1024 processors in the system, then $S = (1, 4, 1019)$ and $P = (0.9, 0.8, 1.0)$ means that there are three clusters, cluster 1 has size 1 and probability 0.9 of being the target, cluster 2 has the 4 closest processing modules and has probability 0.8 of being the target given that the target is not in cluster 0, and cluster 3 has the remaining 1019 processing modules and has probability 1.0 of containing the target given that the target is not in cluster 1 or cluster 2.

The clusters of locality model has been adopted because similar but simpler models have been shown to be effective in studies of direct networks [25, 2, 3] and because the model exhibits memory access patterns similar to scientific applications that have been examined on Hector. This is especially true for applications that have been optimized for NUMA systems and migrate and replicate data objects to improve locality (including when the migration and replication is done by the operating system transparent to the application). Further study of the extent to which real shared memory programs conform to this model (allowing for application restructuring and hardware and software dynamic page (or cache block) placement) is of interest, but outside of the scope of this paper.

A major type of non-uniform traffic is a *hot spot*; that is, a single memory that has an unusually high probability of being accessed by all or many of the processors. Early papers [23, 30] identified hot spot memory modules as a major cause of performance degradation in shared-memory interconnection networks. The degradation is exacerbated by "tree saturation" [23] which even obstructs memory traffic to non-hot spots locations. Significant progress has been made in reducing hot spot traffic, especially hot spot traffic due to synchronization. Techniques include separate synchronization networks (possibly with combining) [18] and hot-spot-free software algorithms that use distributed data structures [20]. Furthermore, flow control mechanisms may be useful, especially when *hot senders* (processors with usually high request rates or a high favoritism to the hot spot) are a factor. Evaluating alternative techniques for reducing hot spot traffic is outside the scope of this paper. Instead, we investigate the significance of the effects due to hot spots on performance for this type of architecture.

Parameter	*Value*	*Description*
N	1024	number of processors (memories) in the system
B	1-8	number of memory banks
$L = (L_1, L_2, \cdots, L_n)$	(1024) to $\overbrace{(2, \ldots, 2)}^{10}$	topology; branching factor at each of the n level
$RXMY$	R1M10 to R4M60	ratio of ring and memory cycles to processor cycles
T	1-8	maximum number of outstanding transactions
λ	0.01-0.05	request rate
R	0.7	probability that a cache miss is a read
$P = (P_1, P_2, \cdots, P_m)$	$(0.95, 0.8, 1.0)$ to $(0.9, 1.0)$	cluster probability for each of the m clusters
$S = (S_1, S_2, \cdots, S_m)$	$(1, 4, 1019)$ to $(1, 1023)$	cluster size for each of the m clusters
F	0.0-0.025	favorite memory probability

Table 1: System and workload parameters used in the simulations and their value ranges.

3 Results

In this section we present the results of our experiments. The primary performance metric used is processor efficiency since it reflects overall system performance. To further understand the differences identified by processor efficiency, a number of secondary metrics are examined. These include mean transaction latency and mean remote transaction latency (for accesses directed to non-local memory). Mean local transaction latency is relatively constant, because network traversal and contention is avoided and because local transactions have priority over remote transactions. We also consider memory and ring utilizations. For ring utilization we report only the average utilization of the rings at the level with the largest average utilization since that ring level has the dominant performance effect.

The ranges of the input parameter values are shown in Table 1. For all systems under consideration, the system size is 1024 processors, the probability that a transaction is a read is 0.7, and the size of cluster 1 is 1 ($S_1 = 1$). The system parameters studied are the system topology, the maximum number of outstanding transactions per processor, the use of blocking versus non-blocking reads, the number of memory banks per memory module, and the relative speed of the processor, ring, and memory. We refer to the latter as the *cycle ratio* and specify it by $RXMY$ which means that each ring cycle is X times as slow as a processor cycle and the memory requires Y processor cycles to service one memory access. The workload parameters studied are the request rate, the communication locality, and the presence of hot spots. All simulation results reported in the section have confidence interval halfwidths of 1% or less at a 90% confidence level, except near saturation where the confidence interval halfwidths sometimes increase to a few percent.

Section 4.1 describes the base system and its performance under different request rates and degrees of communication locality. In the following sections we examine the issues mentioned in Section 1. In Section 4.2 we consider the effects of increasing the maximum number of outstanding transactions and whether reads block. Section 4.3 presents results on the use of multiple memory banks. The effect of variations in ring topology is considered in Section 4.4. Section 4.5 considers hot spot traffic. Finally, Section 4.6 considers changes in the relative speeds of the processors, memories, and rings.

3.1 Base System Performance

For the base system, some variable parameters are fixed as follows: a system topology of $L = (16, 4, 4, 2, 2)$, one memory bank per memory module ($B = 1$), at most one outstanding transaction per processor ($T = 1$) (so that all transactions, including reads, block), and a cycle ratio of $R2M30$ (that is, the ring has a cycle time twice as long as that of the processor and memory requires 30 processor cycles per access).

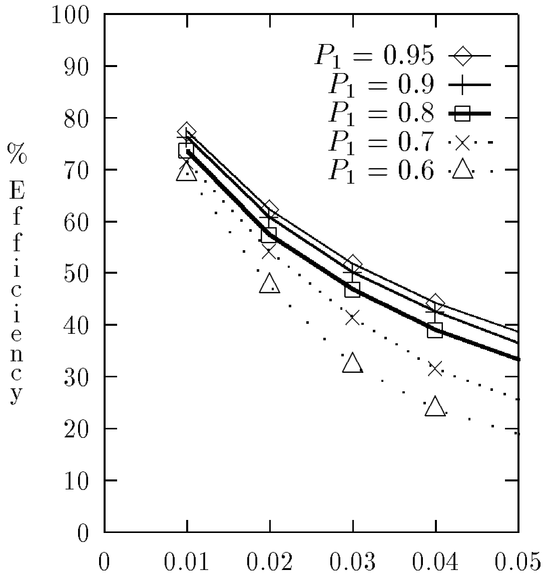
The cycle ratio chosen is based on near-term expected timings. In particular, high-performance processors are now obtaining cycles times on the order of 5ns (such as in the DEC Alpha [11]). Although the IEEE SCI standard specifies a ring cycle time of 2ns, we assume a ring cycle time of 10ns. We do so because we define ring cycle time as the time required for a packet to move from the input of one station to the input of the next station. Such a transfer need not occur in a single ring cycle. For example, a recent performance study of the SCI ring [26] assumes (with no contention) four ring cycles for a packet to traverse a station and the link to the next station. The assumption that a memory cycle takes 30 processor cycles follows values used in recent studies [10].

Figure 2(a) shows how efficiency varies with the request rate, λ, for the base system as the cluster probabilities are varied. As the request rate increases, efficiency drops sharply to less than 40% at a request rate of 0.05 even for $P_1 = 0.95$ (where 95% of all accesses go to local memory). When there is a high degree of locality (that is, a cluster 1 probability of 90% or higher), memory utilization and maximum ring utilization are far from saturation. Hence, the non-contention component of the transaction latency is the primary cause of the decline in efficiency. In other words, efficiency is low because all of the transaction latency is exposed. For cluster 1 probabilities less than 90%, ring contention (for the level 4 rings, one level below the root ring) becomes substantial enough to effect the latency and thus further decrease efficiency.

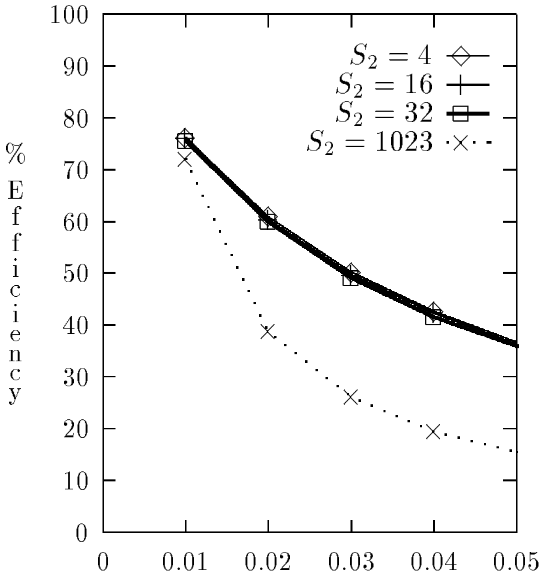
Next we considered the effect of cluster size for the base system for several different cluster 1 probabilities, $P_1 = 0.95, 0.9, 0.8$. Figure 2(b) plots efficiency as a function of the request rate for the case of $P_1 = 0.9$. Increasing the cluster 2 size from 4 to 16 or 32 (thus, spreading non-local accesses over a wider range) has minimal effect on efficiency for any of the cluster 1 probabilities considered. One reason for this invariance in the base case is that the level 1 ring contains 16 processors; a transaction to a target memory on the same level 1 ring as the source processor imposes the same load regardless of the logical distance between the target memory's processor and the source processor. Consequently, the primary cause of increased load by increasing the cluster 2 size from 4 to 16 is due to increased traffic to adjacent level 1 rings.

On the other hand, increasing the cluster 2 size to 1023 (that is, causing all non-local transactions to be uniformly distributed across the machine), has a major effect on efficiency due to ring saturation (primarily the level 3 rings). Even for high locality (that is, $P_1 = 0.95$), a request rate of 0.03 causes a maximum ring utilization of over 90%. For $P_1 = 0.80$, the maximum ring utilization is 100% even at a request rate of 0.01.

We conclude that for the base system, processor efficiency can be quite low at high request rates. At cluster 1 probabilities of 90% or above and cluster 2 sizes on the order of 4 to 32 processors, the low efficiency is due to the long contention-free latency being exposed, which stems from the limit of one outstanding transaction per processor. At cluster 1 probabilities below 90% or cluster 2 size on the order of the system size for any considered cluster 1 probability, ring contention becomes a factor in increasing transaction latency. We examine below techniques to address both causes of low efficiency.



(a) Vary Cluster Probability



(b) Vary Cluster-2 Size

Figure 2: Base system experiments with $T = 1, B = 1, F = 0.0, R2M30, L = (16, 4, 4, 2, 2)$. In Figure 2(a) $P = (x, 0.8, 1.0), S = (1, 4, 1019)$. In Figure 2(b) $P = (0.9, 0.8, 1.0), S = (1, x, 1023 - x)$.

3.2 Maximum Outstanding Transactions and Non-Blocking Reads

We next examine two techniques for increasing efficiency in the presence of long exposed latency: increasing the maximum number of outstanding transactions and allowing non-blocking reads. Our assumptions are the same as the base system with cluster sizes $S = (1, 4, 1019)$ and cluster probabilities $P = (0.95, 0.8, 1.0)$. The effects of allowing the maximum number of outstanding transactions, T, to be 1, 2, 4, 6, and 8 with and without reads blocking are described below.

The first experiment (not shown) varied the request rate for different T values assuming *blocking reads*. This might reflect, for example, a single context that is using a relaxed memory consistency model. Increasing T from 1 to 2 causes a small increase in efficiency (1% absolute change) and further increases in T have no significant effect. Given that 70% of all transactions are reads, it is not surprising that increasing T has limited effect. The remainder of this study, therefore, only considers non-blocking reads.

Figure 3(a) plots efficiency versus the request rate for different T values assuming *non-blocking reads*. Increasing T is effective at substantially improving efficiency (to approximately 75% for T=4 and a request rate of 0.05). However, this increases traffic and hence contention, which in turn, increases latency, thus reducing the effectiveness of increasing T to reduce exposed latency. For the cases of T=6 and T=8, the simulations did not complete under high request rates because of an excessive number of packets in the system arising from severe contention. Figure 3(b) shows how mean transaction latency (in processor cycles) increases for different T values as the request rate increases.

Figure 4 shows that excessive memory utilization is the primary cause of the contention. The increase in ring utilizations at higher request rates is an indirect result of memory contention. Increased packet traffic arises from an increase in packet retries, which in turn is due to requests

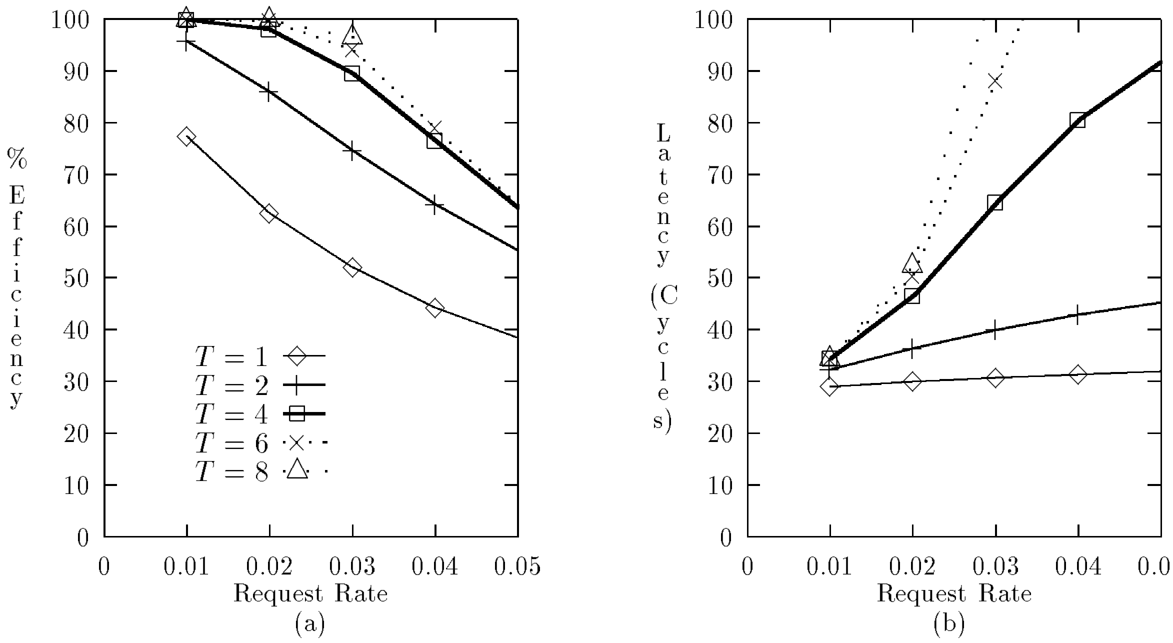


Figure 3: Varying T. Non-blocking reads. $P = (0.95, 0.8, 1.0), S = (1, 4, 1019), F = 0.0, B = 1, L = (16, 4, 4, 2, 2)$.

being turned back at saturated memories. Were it not for the retries, a doubling of the request rate from $\lambda = 0.02$ to $\lambda = 0.04$, should only increase ring traffic by 10%. We conclude that increasing the maximum number of outstanding transactions, given non-blocking reads, is effective at increasing efficiency for the communication locality considered, but that memory saturation limits further efficiency improvements.

3.3 Multiple Memory Banks

We next consider dividing each memory module into multiple memory banks as a means of decreasing memory utilization. The system considered initially is the same as that considered in the previous subsection, $P = (0.95, 0.8, 1.0), S = (1, 4, 1019), L = (16, 4, 4, 2, 2)$, but varying the number of memory banks from $B = 1$ to $B = 8$. The first series of experiments varies the number of memory banks for the case of $T = 2, T = 4$, and $T = 6$. We then fixed the number of outstanding transactions, $T = 4$, and varied the communication locality by considering $P_1 = 0.9$ and $P_1 = 0.8$ for the $S = (1, 4, 1019)$ case, and by considering $P_1 = 0.95$ and $P_1 = 0.9$ for the $S = (1, 1023)$ case.

For the case of $T = 2$ (not shown), going from 1 to 2 memory banks has some effect on processor efficiency (the efficiency at request rate 0.05 increases from 56% to 64%). Adding more memory banks has little effect on efficiency, since the exposed part of the contention-free transaction latency is the limiting factor on efficiency, not memory contention.

With $T = 4$, on the other hand, there is enough concurrency so that the contention-free transaction latency is hidden and memory contention is the limiting factor for the ranges considered. This is shown in Figure 5(a), which displays efficiency versus request rate for different numbers of memory banks. Efficiency is now increased, at a request rate of 0.05 to over 90% for 4 memory banks, ($B = 4$). The cause for this improvement is the reduction in memory utilization as shown

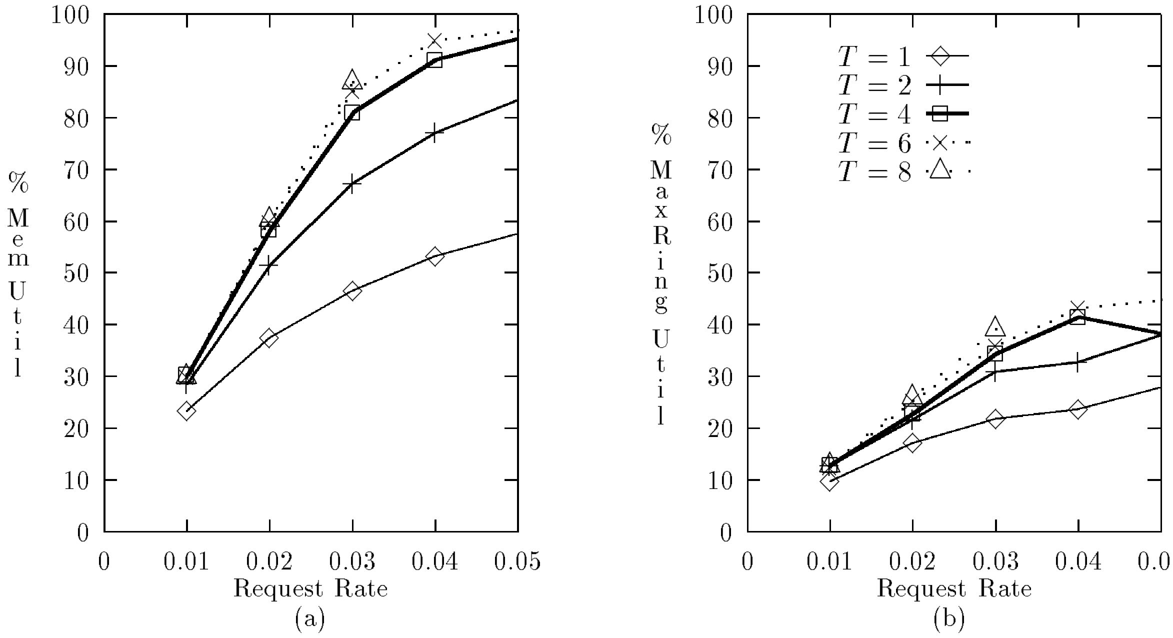


Figure 4: Varying T. Non-blocking reads. $P = (0.95, 0.8, 1.0), S = (1, 4, 1019), F = 0.0, B = 1, L = (16, 4, 4, 2, 2)$.

in Figure 5(b).

The results for $T = 6$ are similar to those for $T = 4$ except that the efficiency is somewhat higher (over 95% at request rate 0.05 for $B = 4$) and memory and maximum ring utilizations change by a few percent.

Figure 5(c) plots the maximum ring utilization as a function of the request rate. The ring utilization is still low enough so that it has little effect on transaction latency, but at the higher request rates it is approaching levels where it will have effects. With only 1 or 2 memory banks, ring utilization is constrained at higher request rates by memory contention. A higher number of memory banks increases the processor efficiency by removing the memory bottleneck, which, in turn, increases the offered load to the network and thus the ring utilization. However, the increase in ring utilization due to packet retries as seen in Figure 4(b) is no longer present.

The increase in ring utilization shown in Figure 5(c) identifies a fundamental tradeoff between maintaining a small value for T and improving processor efficiency. Increasing T has the potential for increasing contention by increasing memory and ring utilization. Consequently, increasing T in order to reduce the exposed latency can, after a point, provide minimal improvement. Thus, the tradeoff is to choose a T value that is as small as possible while still achieving almost all of the possible improvement in processor efficiency. The above experiments indicate that for $P = (0.95, 0.8, 1.0)$ and $S = (1, 4, 1019)$, a value of $T = 4$ would be a good balance in this tradeoff.

To further investigate this tradeoff point, its sensitivity to the degree of communication locality was examined. The results (not shown) indicate that the tradeoff point is highly sensitive. For $P_1 = 0.9$ ($P_1 = 0.8$) with $S = (1, 4, 1019)$, request rates of 0.04 (0.02) or higher cause almost 100% maximum ring utilization. For $P_1 = 0.95$ ($P_1 = 0.9$) in the $S = (1, 1023)$ case, request rates of 0.02 (0.01) or higher cause 100% maximum ring utilization. In all cases, the level 4 rings always have

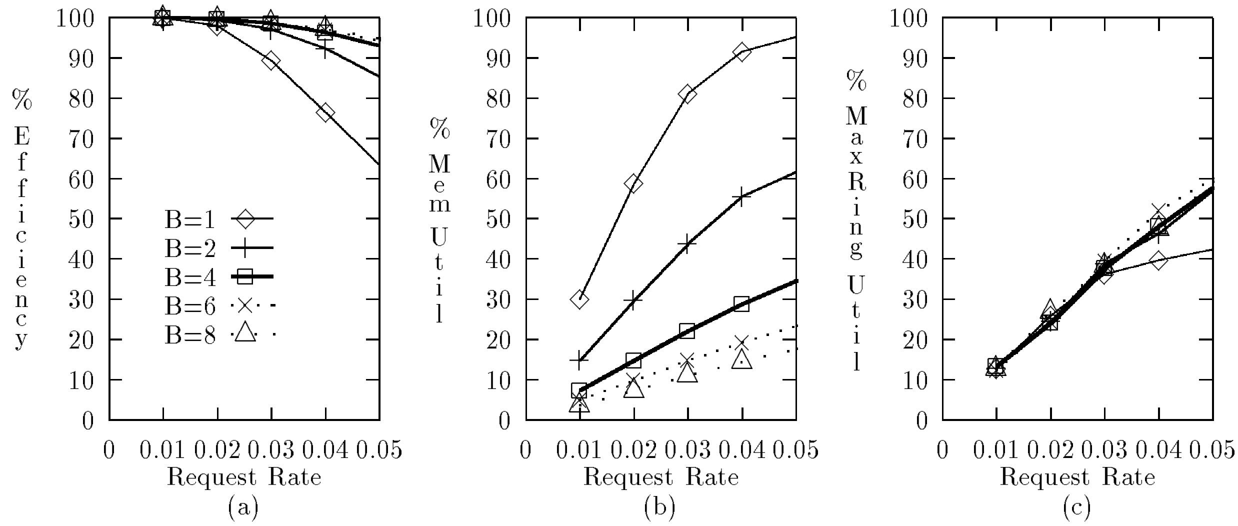


Figure 5: Varying B. $P = (0.95, 0.8, 1.0), S = (1, 4, 1019), F = 0.0, T = 4, L = (16, 4, 4, 2, 2)$.

the highest utilization. Clearly, for these degrees of communication locality, a lower T is needed.

Given the sensitivity of T relative to communication locality, an adaptive T level algorithm is clearly desirable. One decentralized approach is for each processor to decrease T when it observes latency of its transactions higher than expected, an indication of high network or memory contention. Likewise, each processor can increase its T level when the actual latency of its transactions is as low as the contention-free latency would be. An alternative approach, using a centralized scheme that allows the T level to be adjusted system-wide, avoids potential fairness problems with the decentralized approach while introducing coordination overhead.

The above discussion shows that memory saturation can be a major limiting factor to increased processor efficiency and that the use of multiple memory banks is an effective technique to remove this bottleneck. The increase in the offered load due to multiple outstanding transactions and the removal of the memory saturation causes ring utilization to increase and to saturate for some reasonable traffic patterns. The appropriate number of outstanding transactions is a tradeoff between concurrency and contention, which is highly sensitive to the degree of communication locality. An adaptive maximum number of outstanding transactions may be useful for adjusting to this tradeoff as the communication locality changes.

3.4 Ring Topology

Ring utilization is also affected by the topology of the ring hierarchy. To evaluate the significance of the ring topology, we first analyzed the maximum transaction latency in a ring hierarchy when each access goes to the most distant memory; that is, when the request and response packets must traverse all of the levels in going from the source to the target memory and back. If both request and response subtransactions are considered together, then each traversed ring is completely traversed exactly once. A complete traversal of a ring with a branching factor L_i takes $L_i + 1$ ring cycles if the ring is not the root ring (one cycle for each L_i child inter-ring interface or station plus one for the parent inter-ring interface). A complete traversal of the root ring with a branching factor L_i takes L_i ring cycles. Thus, the contention-free maximum transaction latency, ML, for a configuration

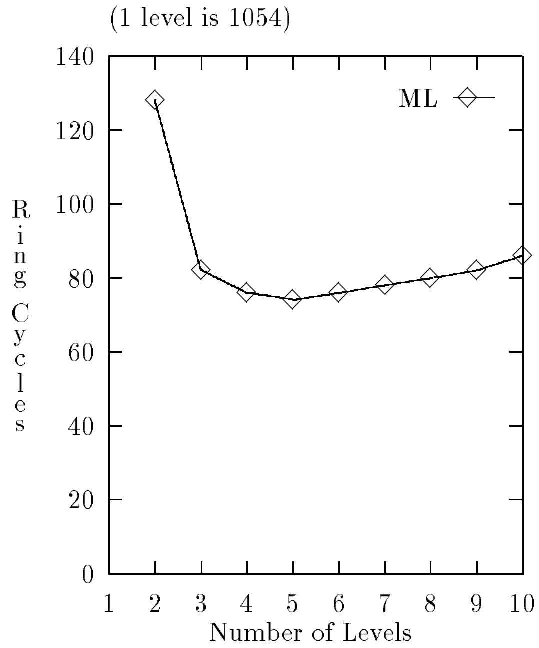


Figure 6: Contention-free maximum transaction latency in ring cycles for the optimal topology at each number of ring levels. $mem = 30$.

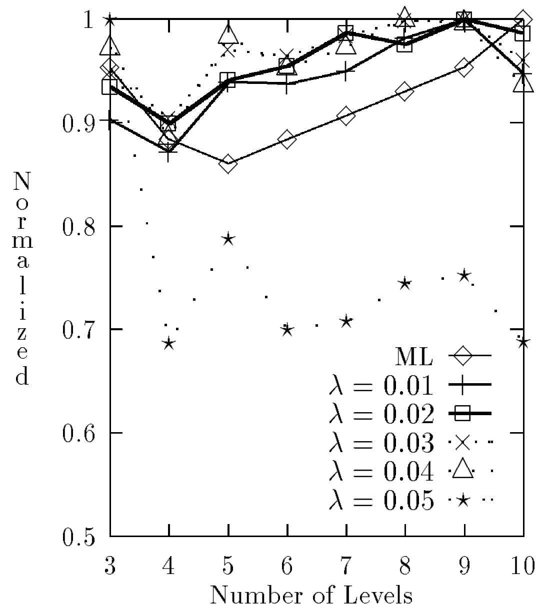


Figure 7: Comparison of normalized contention-free maximum transaction latency with normalized mean remote transaction latency for traffic pattern $P = (0.95, 0.8, 1.0), S = (1, 4, 1019), T = 4, B = 4$.

$L = (L_1, L_2, \ldots, L_m)$ is $ML(L) = \sum_{i=1}^{m-1} 2(L_i + 1) + L_m + mem$, where mem is the memory cycle time in ring cycles.

We computed ML for a number of different topologies. For all the topologies that have the same number of levels we chose the one with the lowest ML value and plotted them in Figure 6. Although a memory latency of 30 is assumed in order to be consistent with the $R2M30$ cycle ratios used in the other simulations, mem simply causes a constant shift. The optimum topology has 5 levels and is $L = (4, 4, 4, 4, 4)$. For a given number of levels, topologies with the best ML values minimize large branching factors except possibly for a larger branching factor at the root since the root ring is only traversed once. The relative performance of topologies with different numbers of levels is determined by the balancing of two factors: 1) larger branching factors have the disadvantage of requiring the packet to traverse extra links to reach the next level, and 2) larger number of levels has the disadvantage of increasing the total number of inter-ring interfaces in the system.

Although the above analysis shows that balanced topologies are best, the analysis ignores more realistic traffic patterns and contention. The above analysis represents the worst case, in which all accesses are to memories for which the contention-free transaction latency is maximum. This traffic pattern will cause ring saturation unless the request rate is extremely low and is unrealistic. For the next set of experiments we consider traffic patterns that are more realistic and that our previous experiments indicate will probably cause the network to be heavily utilized, but not saturated. The preliminary comparison considered the topologies with the best ML value for each level and used

Index	*Topology*
1	(4,4,8,8)
2	(4,4,4,4,4)
3	(2,2,4,4,4,4)
4	(16,8,4,2)
5	(8,4,4,4,2)
6	(8,8,4,2,2)
7	(16,4,4,2,2)
8	(4,4,4,4,2,2)
9	(8,8,2,2,2,2)
10	(16,4,2,2,2,2)

Table 2: Ring topologies considered for different communication localities. The index denotes the topology in Figure 8.

the traffic pattern $P = (0.95, 0.8, 1.0), S = (1, 4, 1019)$ with $T = 4, B = 4$. For each request rate, we normalized the results against the worst mean remote transaction latency for that topology and plotted them in Figure 7.

The qualitative behavior of the alternative topologies is similar to that of the normalized ML values in that topologies with 4 to 6 levels tend to be superior. As Figure 7 shows, under high traffic loads, having too few levels can hurt performance substantially and, under low traffic loads, having too many levels also hurts performance. For request rate 0.05, a large number of levels does not degrade performance. This invariance is due to root ring contention. Some of the best topologies by ML value have a large branching factor at the root. For these topologies the root ring contention masks any other performance changes due to a large number of levels.

Because topologies with 4 to 6 levels appear to perform better, we then examined in more detail the performance of alternative topologies with that many levels under three traffic patterns. The three traffic patterns we considered are:

1. $P = (0.95, 0.8, 1.0), S = (1, 4, 1019)$, with $T = 4, \lambda = 0.05$,
2. $P = (0.90, 0.8, 1.0), S = (1, 4, 1019)$, with $T = 2, \lambda = 0.05$, and
3. $P = (0.95, 1.0), S = (1, 1023)$, with $T = 1, \lambda = 0.02$.

Besides the topologies that the ML analysis indicated were optimal we considered several others that had somewhat larger branching factors at the lower ring levels. The rationale is that the previous experiments indicated that the higher ring levels had the highest utilization. Consequently, reducing the branching factor at the higher ring levels might be advantageous.

Table 2 lists the topologies considered. The index associated with each topology in Table 2 denotes that topology in Figure 8. The topologies with indices 1-3 are the best for their number of levels by ML values. Figure 8 plots the maximum ring utilization for the investigated topologies for the three traffic patterns. The best topologies, as measured by ML, do not perform especially well. The baseline topology $(16, 4, 4, 2, 2)$ that we have been using in earlier experiments performs well, but the relative performance depends on the traffic pattern. Any of the topologies numbered 4 through 10 seem to be an acceptable choice with none of them clearly superior.

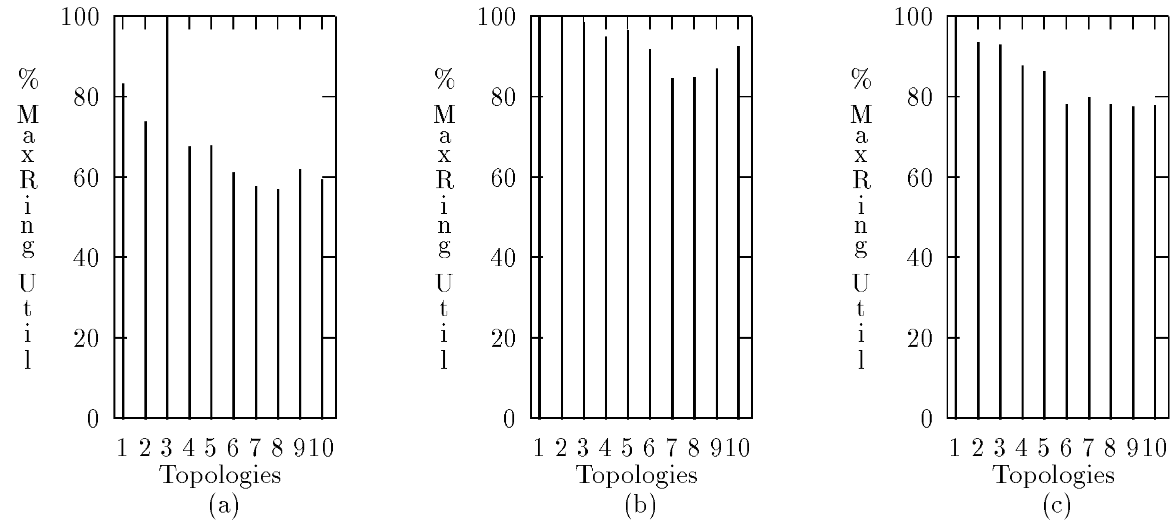


Figure 8: Comparison of the maximum ring utilizations of alternative topologies for three traffic patterns. The plots are for the topologies patterns listed in Table 2.

We conclude that the best contention-free topology, as measured by maximum transaction latency, and the one least sensitive to contention has between 4 and 6 levels. The best of these topologies tend to have uniform branching factors at the different levels except that somewhat larger branching factors at the lower levels also do well. The relative performance of the best topologies depends on the traffic patterns and also on the performance measure (mean remote transaction latency or maximum ring utilization).

3.5 Hot Spot Traffic

The results presented so far are based on a communication locality model where the traffic within a cluster is uniformly distributed. In this subsection we examine the behavior of the system in the presence of a hot spot. To perform the analysis, we assume that a transaction from any processor has probability F of addressing the hot spot memory. The remaining transactions are distributed according to the standard communication locality model. Moreover, we assume $P = (0.95, 0.8, 1.0), S = (1, 4, 1019), B = 4, T = 1, L = (16, 4, 4, 2, 2)$. In all of the experiments reported so far, the simulated system has had a memory queue length of 9. When there is no hot spot traffic and four memory banks, this queue length is sufficient so that queue overflow is extremely unlikely at traffic loads which do not saturate the memory itself. It is possible that with with a hot spot a longer memory queue is needed. Consequently, in this section we consider queue lengths of 9 and 36.

Figure 9(a) evaluates the effect of the hot spot on overall system performance by plotting mean remote transaction latency (for all remote transactions in the system) versus request rate for different favorite memory probabilities for a memory queue of length 9. The request rate needed to cause latency to significantly increase depends greatly on the favorite memory probability. It is clear that favority memory probabilities of 1% or more are not supportable at reasonable request rates with respect to overall system performance. Figure 9(b) uses a queue length of 36. The change in queue length has some effect, but the qualitative behavior is the same.

The degradation in performance in Figure 9 is large once a critical request rate is reached, because of the negative feedback effect of packet retries. The favoritism to the hot spot memory imposes additional load on three resources: the hot spot memory, the queue for the hot spot memory, and the network near the hot spot memory. As contention for the hot-spot memory increases, some of the access request packets destined for the hot spot memory will be negatively acknowledged and then resent. The above experiments indicate that all three resources contribute to performance degradation. The hot spot memory itself is clearly a factor due to its high utilization, but we can not actually saturate it. For both queue lengths of 9 and 36, workloads that cause mean remote latencies to be in the hundreds, have hot memory utilizations between 85% and 98%. Memory queue overflow is a factor since a memory queue of 36 has lower mean remote latencies and higher hot memory utilizations than a memory queue of 36. Network saturation of rings near the hot memory is a factor; measurements of the queues at the IRIs near the hot memory show long queues.

As mentioned in Section 2, our goal, with respect to hot spots, is to understand their significance within this class of systems, instead of evaluating alternative solutions. Without techniques to alleviate hot spots, our experiments indicate that such systems can become unstable at favorite memory probabilities on the order of 1% to 2% under reasonable request rates. If the memory queues are of inadequate length, significantly lower favorite memory probabilities can cause instability. The techniques proposed in the synchronization literature (such as separate synchronization networks with combining or software algorithms using distributed data structures [20, 18]) may well reduce the likelihood of hot spots. The simulated system does provide flow control in that the number of cycles before a source processing module submits a retry is a function of the number of retries previously sent [27]. More sophisticated flow control mechanisms could be considered. One possibility is that the destination module could return with the negative acknowledgement an indication of how congested it is. The source module could use this information to choose a wait period.

3.6 Relative Speed

So far we have assumed a fixed ratio of the processor, memory, and ring speeds, namely $R2M30$. In this section, we examine the effect of varying this ratio on the conclusions drawn in the previous sections. Since processor speeds seem to be increasing faster than memory speeds, three cases are considered for which the processor speed is increased by 100% and the memory cycle remains unchanged (and thus is now 60 processor cycles). The first case assumes that the ring cycle time remains unchanged at 4 processor cycles, the second case assumes a 50% reduction and the third case, assumes a 75% reduction in ring cycle time. The cases are denoted $R4M60$, $R2M60$, and $R1M60$, respectively. We assume $T = 4, B = 4, L = (16, 4, 4, 2, 2), P = (0.95, 0.8, 1.0), S = (1, 4, 1019)$.

Figure 10(a) plots efficiency versus request rate for the three cases as well as for $R2M30$, our base case. The faster processor causes efficiency to drop somewhat in comparison to $R2M30$, but, surprisingly, the drop is essentially identical for all three cases. (In noting the high efficiency of $R2M30$ it is important to remember that, relatively speaking, the processor in this case is only executing instructions at half the speed of the other cases.) Figure 10(b) plots maximum ring utilization versus request rate for all of the cases and shows that the similar behavior with respect to efficiency masks major differences with respect to ring utilization. (There is little difference among the cases with respect to memory utilization.) In fact the ring saturation of $R4M60$ suggests that processor efficiency for that case should start dropping if the offered load is increased further. To test this hypothesis we redid the experiment with $P = (0.9, 0.8, 1.0)$. Figure 11(a) and Figure 11(b) plots efficiency and maximum ring utilization, respectively, versus request rate for this communication

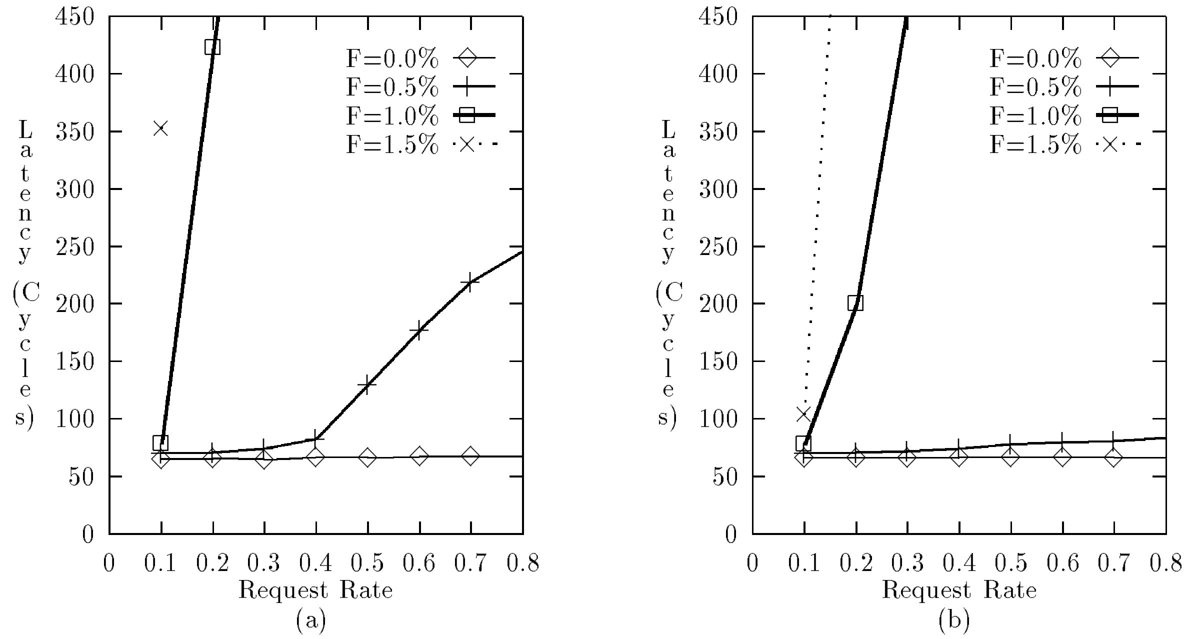


Figure 9: Hot spot. How large of a request rate can be supported for different favorite memory probabilities before overall system performance degrades as measured by mean remote transaction latency. Figure 9(a) has a memory queue 9 deep. Figure 9(b) has a memory queue 36 deep.

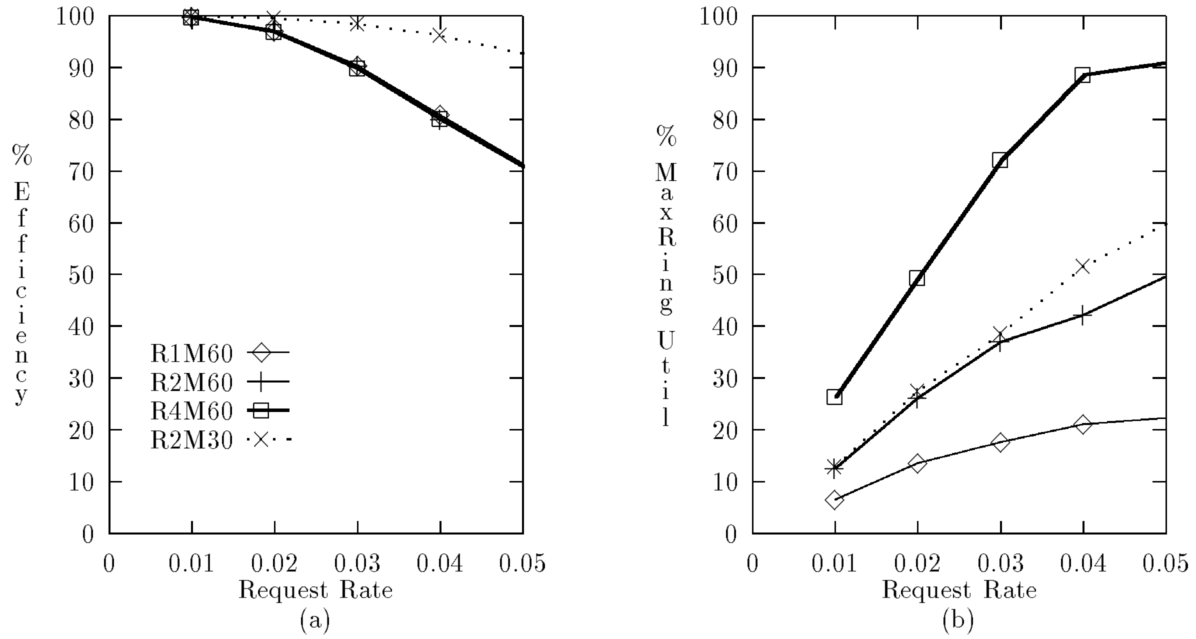


Figure 10: Effect of different processor and ring speeds. $P = (0.95, 0.8, 1.0)$.

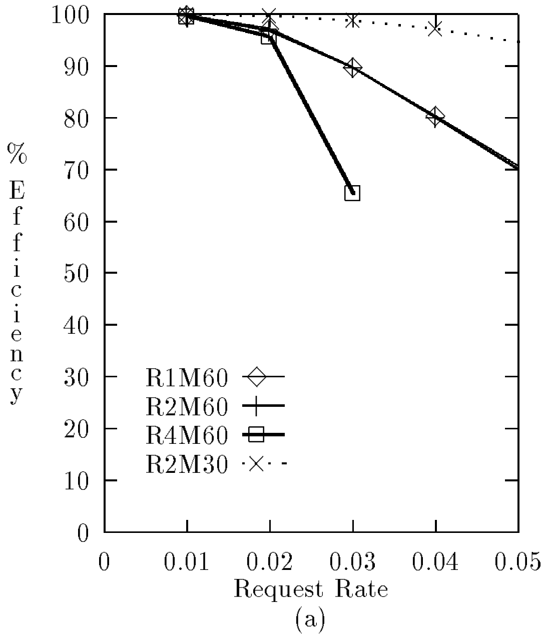


Figure 11: Effect of different processor and ring speeds. $P = (0.9, 0.8, 1.0)$. The simulation for $R4M60$ at $\lambda = 0.03$ aborts due to saturation. The efficiency and maximum ring utilization points reported for that case is the average of the batches that completed.

locality. The results confirm our hypothesis by the sharp drop in processor efficiency for $R4M60$ when at request rate 0.03. The collapse in performance when increasing request rate beyond 0.02 is so sharp, that the request rate 0.03 simulation does not complete. The point reported is the mean of the batches before the simulation aborts.

We have not explicitly considered a faster ring cycle time in our experiments. The $R1M60$ results in the Figure 10 and Figure 11 plots make clear that a faster ring cycle time would have little effect since at $R1$, the memory is the limiting factor, not the ring utilization [3].

Returning to our base case, we then considered the effect of our assumption that a memory cycle equals 30 processor cycles. One of our conclusions had been that as we increased T to hide the exposed transaction latency, memory saturation became a limiting factor. We then used multiple memory banks to allow higher processor efficiency. Now we return to one memory bank to see how sensitive our results are to memory cycle time. We assume $T = 4, B = 1, L = (16, 4, 4, 2, 2), P = (0.95, 0.8, 1.0), S = (1, 4, 1019)$ and that a ring cycle equals 2 processor cycles and vary memory cycle time from 10 to 40 processor cycles.

Figure 12(a) plots efficiency versus request rate as memory cycle time, M, is varied. As M increases, efficiency drops which is partially due to an increase in the base transaction latency and partially due to increased contention. To understand the degree to which contention contributes to the drop in efficiency, we plot memory and maximum ring utilization versus request rate in Figure 12(b) and Figure 12(c), respectively. As memory cycle time increases, memory utilization increases and maximum ring utilization decreases. For the longer memory cycle times and high

[3]Some ring-based systems (such as the KSR1 [8, 6]) have a ring cycle time faster than the processor cycle time. Most often these systems (as does the KSR1) require several ring cycles for a transaction to pass through a ring node.

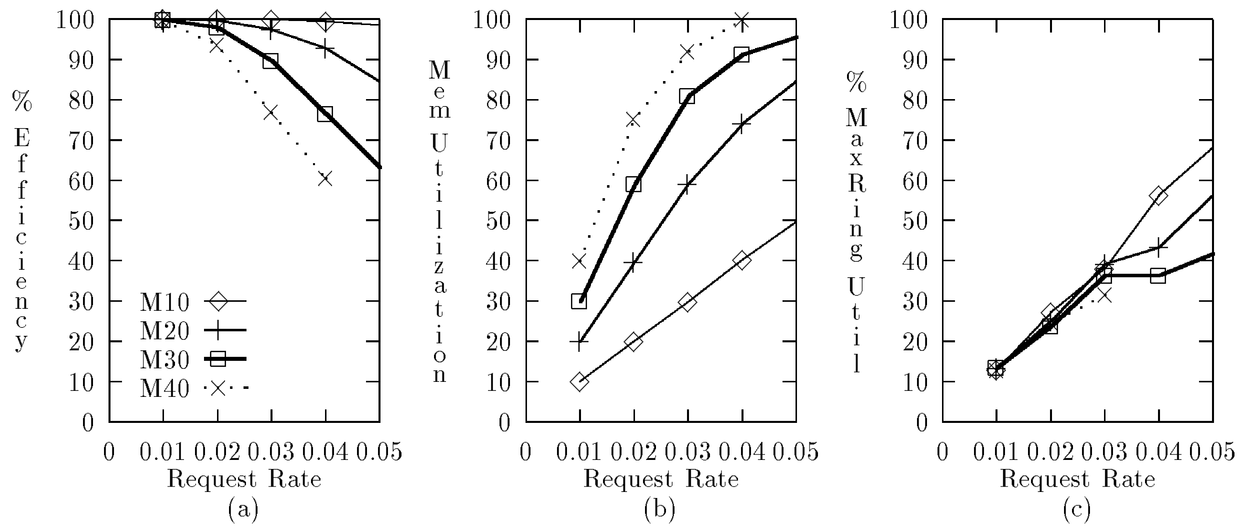


Figure 12: Effect of different memory cycle times.

request rates, memory saturates and the maximum ring utilization is constrained by the memory saturation. The case of $M = 10$ significantly differs from larger M values in that for $M = 10$ memory utilization is less than maximum ring utilization. It appears that there is a significant transition as M increases above 10 that involves whether memory utilization becomes a bottleneck before the rings become a bottleneck. For higher M values, multiple memory banks are useful for increasing efficiency. For lower M values, multiple memory banks will not be useful for increasing efficiency, since ring saturation dominates and since there is little room for improvement in efficiency given that T is raised adequately (efficiency is close to 100% with $T = 4$ in Figure 12).

3.7 Block Transfers

The increase in the ratio of memory cycle time to processor cycle time has motivated the development of innovative memory technologies [24, 21]. The one approach we examine here is page-mode DRAM access. With page-mode access, fetching the first word from memory requires raising both the row-access strobe line and the column-access strobe line. However, subsequent words on the same row in memory can be retrieved by just changing the column address and reraising the column-access strobe line. Thus, the access time for the first word is unchanged, but the additional access time for subsequent words on the same row is sharply reduced. Page-mode access is a natural mechanism to support block transfer.

The result of page-mode access is to increase the memory bandwidth and to reduce the average latency (as measured over all words fetched from memory). If we assume that the first word fetched is a cache miss and the subsequent words are prefetches, then the use of page-mode access is only advantageous to the extent that the cache hit ratio is increased by spatial locality through the prefetching. Page-mode access had disadvantages in that memory and network utilization increase and in that there is increased potential for cache pollution (to the extent that the prefetched words are not referenced and cause the replacement of words that will be referenced). In addition, if the processor stalls until the last word fetched reaches the processor (instead of just until the first word is fetched), the stall time on cache misses further increases.

The extent to which the use of page-mode access increases the cache hit ratio is highly program-dependent and out of the scope of this study. However, we can determine the extent to which the cache hit ratio needs to increase in order to compensate for the disadvantages of page-mode access. We conducted several experiments to determine the needed hit ratio increase at request rates 0.01% through 0.05%.

In all the experiments we fixed the ring topology and the ratio of processor, ring, and memory speeds to the base case: $L = (16, 4, 4, 2, 2)$ and $R2M30$, respectively. We assumed four memory banks, no hot spot traffic, and that each word of a block after the first word takes 5 processor cycles. We considered four cases of communication locality: $P = (0.95, 1.0), S = (1, 1023)$ and $P = (P_1, 0.8, 1.0), S = (1, 4, 1019)$ with $P_1 = 0.95, 0.9, 0.8$. For all four communication localities we considered having at most four outstanding transactions. For $P = (0.95, 0.8, 1.0), S = (1, 4, 1019)$ we also considered having at most one outstanding transaction. We considered four block sizes: 1, 2, 4, and 8 words. For all block sizes we assumed that the processor stall ends when the first word reaches the processor.

For all communication localities and maximum numbers of outstanding transactions considered, increasing the block size degrades performance at all request rates by all of the measures of processor efficiency, mean request latency (measured by when the processor stall ends), memory utilization, and maximum ring utilization. The percentage degradation depends on the measure. The memory and maximum ring utilizations are more fundamental in the sense that their proximity to saturation determine the contention components of processor efficiency and request latency. Since the maximum ring utilizations are substantially higher than the memory utilizations in all the experiments, we chose to report the change to the maximum ring utilization. We characterize this change by the percentage the mean number of processor cycles between cache misses must increase in order for the maximum ring utilization for a block size of b to equal the maximum ring utilization for a block size of 1. Thus, implicitly we are identifying the necessary change in the cache hit ratio.

The results for a communication locality of $P = (0.95, 0.8, 1.0), S = (1, 4, 1019)$ and $T = 1, 4$ are plotted in Figure 13. The behavior is surprising. The needed increase in the number of processor cycle between cache misses is increasingly close to linearly independent of the request rate and the maximum number of outstanding transactions. The behavior (not shown) for the other three communication localities is similar. The percentage increases are also quite large. For example, over all the experiments there is one case where a block size of two words requires an increase of 13%. For all other cases a block size of two words requires an increase of between 33% and 90%. Whether an increase in the block size to two words would cause such an improvement depends on the program and cache characteristics, but seems doubtful. Increasing the block size to more than two words would require much additional improvement, which seems even more doubtful.

The above experiments assume that the processor stall ends after the first word of the block reaches the processor. For the 8 word block size we repeated the above experiments considering the case in which the processor stall ends after the last word of the block reaches the processor. In comparison with the 8 word block size with the processor stall ending after the first word returns, the change is minor. The memory and maximum ring utilizations decrease by a few percentage points. As expected, the mean remote request latency increases by the extra length of the remaining words of the block plus a contention factor, and the processor efficiency drops be a few percent.

We conclude that using block transfers through page-mode DRAM access does not appear promising. Across many communication localities, request rates, maximum numbers of outstanding transactions, and block sizes, the extra traffic resulting from fetching the extra words significantly raises the network utilization as measured by the maximum ring utilization. The improvement in cache hit ratio due to a larger block size is program and cache dependent. However, the improvement

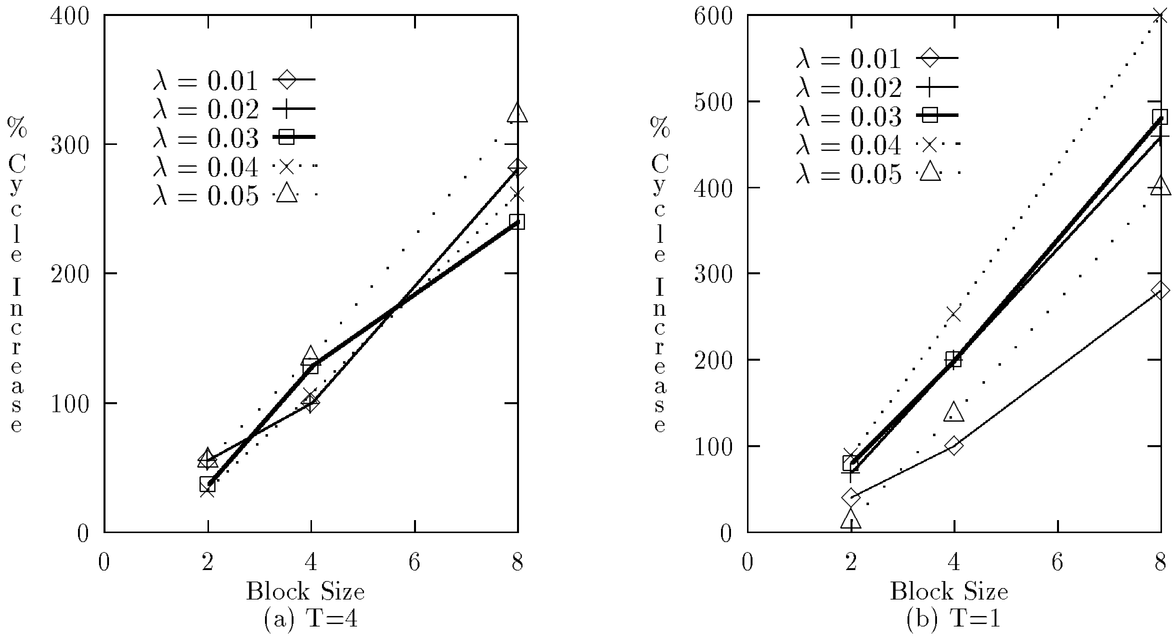


Figure 13: Effect of increasing the block size through page-mode DRAM access. The effect is indicated by the percentage that the mean number of processor cycles between cache misses must increase in order for the maximum ring utilization with a block size of b to equal the maximum ring utilization with a block size of 1. Communication locality is $P = (0.95, 0.8, 1.0), S = (1, 4, 1019)$.

in cache hit ratio needed to compensate for the increase in maximum ring utilization is large enough to be doubtful that the improvement can be achieved. This result is specific to very large systems with this type of network and might not apply to smaller systems or systems with other types of networks.

4 Conclusions

4.1 Related Work

Beside Hector, several architectures based on slotted rings have been proposed including the CDC CYBERPLUS [14], the Express Ring [5], the IEEE SCI (Scalable Coherent Interface) standard [17, 22], and the KSR-1 from Kendall Square Research [8, 6]. There have been performance studies of single slotted rings, but not of ring-based hierarchies. Previous studies of other shared-memory architectures with system models at the level of detail of our simulator have tended to only examine small systems (100 processors or less) [31, 9].

The performance of branching factor topologies has been considered for bus hierarchies. The experiments by Vernon, Jōg, and Sohi [28] indicate that the best topologies had large branching factors at the low levels and small branching factors near the root. In contrast, our contention-based experiments indicate that for ring-based systems, topologies with close to balanced branching factors are best. Somewhat larger branching factors at the lower ring levels sometimes do well, but very large branching factors at low ring levels perform poorly.

A recent analytical study by Agarwal examines the effectiveness of multithreading on increasing processor efficiency [4]. This is relevant since multithreading is one of the main approaches for allowing multiple outstanding transactions. However, the system and workload characteristics Agarwal considers are significantly different than ours. The network is a k-ary, n-cube, memory access time is 10 processor cycles, traffic is uniform (any memory is equally likely to be the target memory of a cache miss), the cache miss ratio is a function of the number of threads, and the context switch overhead is nonzero. The largest difference is in the role of memory. The Agarwal model does not take memory contention into account in determining transaction latency. Even if the model did, the significance of memory saturation occurring before network saturation (and thus, the importance of techniques to reduce memory utilization) would not be observed due to memory access time being 10 cycles (see our Figure 12).

4.2 Summary

In this paper, we presented the results of a simulation study to assess the performance of a shared-memory multiprocessor, based on a hierarchical network of rings. Ring based systems are of interest because they can be run at very fast rates due to their use of short, unidirectional and point-to-point connections, and because of the use of simple node interfaces and inter-ring connections. Hierarchical systems are of interest because they are scalable. To ensure that the systems we simulated were realistic and realizable, we based them on a specific system, Hector. To ensure that the simulator is correct and captures subtle system interactions, the simulator was validated using measurements of this prototype system.

Trace-driven simulation would produce questionable results and execution-driven simulation using complete applications would take prohibitively long when simulating a system this large at the level of detail desired for realism. Consequently, we introduced a synthetic workload model using parameter value ranges based on experience with applications. With this workload model we have shown how systems on the order of 1024 processors can be evaluated even with network and memory simulation at a detailed level.

The main results of this study are:

- Without a high degree of locality in the data accesses of the applications running on this type of system, ring contention will cause the memory access latency to increase significantly. Locality can be achieved by proper data placement and by migrating and replicating data objects.
- If processors do not have support multiple outstanding transactions (such as, prefetching, multiple hardware contexts, release consistency), then processor efficiency will be low due to the long memory access transaction latency even if there is no contention.
- With multiple outstanding transactions, memory will quickly saturate if there is only one memory bank per processing module. Using multiple memory banks per processing module is an effective way to reduce memory contention.
- It is necessary to limit the number of multiple outstanding transactions per processor in order to limit network contention. The appropriate number is a tradeoff between concurrency and contention, and we have found it to be sensitive to the degree of communication locality. We have proposed an adaptive approach for adjust for this tradeoff as the communication locality changes.
- With respect to topology, we have found that 1024 processor systems with between 4 and 6 levels in the hierarchy tend to perform better, although no one topology is best. We have also found that well-balanced systems (with a similar branching factor at all levels) tend to perform best. Slightly smaller rings at the root of the hierarchy reduce the potential for congestion at that point.
- Saturation of hot spot memories causes substantial degradation of overall system performance if 1% or more of memory traffic is targeted to a single memory.
- Doubling the processor speed and keeping the memory cycle time constant causes a drop in processor efficiency due to the increased relative transaction latency. At the traffic patterns considered, changes in the ring speed have a major effect on maximum ring utilization. The effect on processor efficiency is again sensitive to communication locality.
- Varying memory cycle time significantly effects processor efficiency, memory utilization, and maximum ring utilization. Most notably, with one memory bank, the presence of memory saturation at offered loads below those that cause ring saturation only occurs when the memory cycle time is 20 processor cycles or longer.
- Prefetching through the use of block transfers imposes an additional load on the network. The improvement in cache hit ratio needed to compensate for the increase in maximum ring utilization is large enough to make it doubtful such prefetching is advantageous.

Other issues that warrant further investigation include the effect of synchronization and flow control mechanisms on hot spot traffic, the effectiveness of the proposed adaptive maximum number of outstanding transactions, the use of other DRAM techniques to compensate for long memory access times, and including in the traffic pattern, page migration and replication transactions.

Acknowledgements

We thank Darrell Kindred for his work on an early version of the simulator and Keith Farkas for his thorough comments on a draft of the paper.

References

[1] S.V. Adve, V.S. Adve, M.D. Hill, and M.K. Vernon. Comparison of hardware and software cache coherence schemes. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 298–308, Toronto, ON, Canada, May 1991.

[2] V.S. Adve and M.K. Vernon. Performance analysis of multiprocessor mesh interconnection networks with wormhole routing. Technical Report CS-TR-1001, Computer Science Department, Univ. of Wisconsin—Madison, Madison, WI, February 1991.

[3] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[4] A. Agarwal. Performance tradeoffs in multithreaded processors. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):525–539, September 1992.

[5] L. Barroso and M. Dubois. Cache coherence on a slotted ring. In *Proceedings of the 1991 International Conference on Parallel Processing*, St. Charles, IL, August 1991. The Pennsylvania State University Press.

[6] G. Bell. Ultracomputers: a teraflop before its time. *Communications of the ACM*, 35(8):27–47, August 1992.

[7] B. Boothe and A. Ranade. Improved multithreaded techniques for hiding communication latency in multiprocessors. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 214–223, Gold Coast, Australia, May 1992.

[8] H. Burkhardt, S. Frank, B. Knobe, and J. Rothnie. Overview of the KSR 1 computer system. Technical Report KSR-TR-9202001, Kendall Square Research, Boston, MA, February 1992.

[9] D.-K. Chen, H.-M. Su, and P.-C. Yew. The impact of synchronization and granularity on parallel systems. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pages 239–248, Seattle, WA, May 1990.

[10] T.-F. Chen and J.-L. Baer. Reducing memory latency via non-blocking and prefetching caches. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 51–61, Boston, MA, October 1992.

[11] R. Comerford. How DEC developed Alpha. *IEEE Spectrum*, 29(7):26–31, July 1992.

[12] H. Davis, S.R. Goldschmidt, and J. Hennessy. Multiprocessor simulation and tracing using Tango. In *Proceedings of the 1991 International Conference on Parallel Processing*, pages (II)99–107, St. Charles, IL, August 1991.

[13] K. Farkas, Z. Vranesic, and M. Stumm. Cache consistency in hierarchical-ring-based multiprocessors. In *Proc. Supercomputing 92*, November 1992.

[14] M. Ferrante. CYBERPLUS and MAP V interprocessor communications for parallel and array processor systems. In W.J. Karplus, editor, *Multiprocessors and Array Processors*, pages 45–54. The Society for Computer Simulations, 1987.

[15] K. Gharachorloo, A. Gupta, and J. Hennessy. Hiding memory latency using dynamic scheduling in shared-memory multiprocessors. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 22–35, Gold Coast, Australia, May 1992.

[16] A. Gupta, J. Hennessy, K. Gharachorloo, T. Mowry, and W.-D. Weber. Comparative evaluation of latency reducing and tolerating techniques. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 254–265, Toronto, ON, Canada, May 1991.

[17] D.B. Gustavson. The scalable coherent interface and related standards projects. *IEEE Micro*, 12(1):10–22, February 1992.

[18] W. T.-Y. Hsu and P.-C. Yew. An effective synchronization network for hot spot accesses. *ACM Transactions on Computer Systems*, 10(3):167–189, August 1992.

[19] M.H. MacDougall. *Simulating Computer Systems: Techniques and Tools.* MIT Press, 1987.

[20] J.M. Mellor-Crummey and M.L. Scott. Synchronization without contention. In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 269–278, Santa Clara, CA, April 1991.

[21] R. Ng. Fast computer memories. *IEEE Spectrum*, 29(10):36–39, October 1992.

[22] P1596 Ballot Review Committee of the IEEE Microprocessor Standards Committee. Sci-scalable coherent interface, p1596/d2.00. Technical report, IEEE, November 1991.

[23] G.F. Pfister and V.A. Norton. "Hot spot" contention and combining in multistage interconnection networks. *IEEE Transactions on Computers*, 34(10):–395, October 1985.

[24] B. Prince. *Semiconductor Memories, Second Edition.* John Wiley, 1991.

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

[26] S.L. Scott, J.R. Goodman, and M.K. Vernon. Performance of the SCI ring. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 403–414, Gold Coast, Australia, May 1992.

[27] M. Stumm, Z. Vranesic, R. White, R. Unram, and K. Farkas. Experiences with the Hector multiprocessor. Technical Report CSRI Technical Report 276, Univ. of Toronto, Department of Computer Science, Toronto, Canada, 1992.

[28] M.K. Vernon, R. Jog, and G.S. Sohi. Performance analysis of hierarchical cache-consistent multiprocessors. *Performance Evaluation*, 9:287–302, 1989.

[29] Z.G. Vranesic, M. Stumm, D.M. Lewis, and R. White. Hector: A hierarchically structured shared-memory multiprocessor. *IEEE Computer*, pages 72–78, January 1991.

[30] P. Yew, N. Tzeng, and D.H. Lawrie. Distributing hot-spot addressing in large-scale multiprocessors. *IEEE Transactions on Computers*, 36(4):388–395, April 1987.

[31] R.N. Zucker and J.-L. Baer. A performance study of memory consistency models. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 2–12, Gold Coast, Australia, May 1992.