Evaluating Memory System Performance of a Large Scale NUMA Multiprocessor *

Karim Harzallah and Kenneth C. Sevcik

e-mail: {karim,kcs}@csri.toronto.edu

Technical Report CSRI-283

Currently under revision

March 1993

Computer Systems Research Institute

University of Toronto

Toronto, Canada

M5S 1A1

The Computer Systems Research Institute (CSRI) is an interdisciplinary group formed to conduct research and development relevant to computer systems and their application. It is an Institute within the Faculty of Applied Science and Engineering, and the Faculty of Arts and Science, at the University of Toronto, and is supported in part by the Natural Sciences and Engineering Research Council of Canada.

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.

Abstract

The effectiveness of large scale computing depends to a great extent on the performance of the memory system. As shared memory multiprocessors grow in size, their memory hierarchy deepens, resulting in a design with non-uniform latencies. In this paper, we explore the implications of multi-valued memory latencies. In particular, we study the effect of a non-uniform traffic distribution on a hierarchical large scale NUMA multiprocessor named Hector. Memory analysis is of interest because memory is a frequent source of poor performance in large scale multiprocessors. We have developed an analytical model that includes the effects of increased contention for system resources, and the impact of the arbitration algorithm on the network traffic. Our analysis has been validated with a detailed simulator. Also, we have examined two techniques for reducing memory latency. We assess the potential performance gains from replication of data and investigate the improvement in memory utilization by allowing memory request buffering. Furthermore, we studied the sensitivity of the memory performance to changes in background traffic. We found that inter-station traffic has a significant performance effect.

1 Introduction

Reducing and hiding memory latency becomes increasingly important and challenging as more processors are added to a system. Numerous researchers have addressed this problem in the design of scalable shared-memory multiprocessors. Rothberg and Gupta [RG92] studied the parallel performance of the Incomplete Cholesky Conjugate Gradient on a four processor SGI 4D/340. They found that the level-scheduled and the self-scheduled methods for solving the ICCG problem yielded poor speedup and even slowdown in few cases. They conjecture that the problem is particular to multiprocessors with a deep memory hierarchy, and that the problem will get even worse as processors become faster. Their investigation revealed that the parallel version generates much more memory system traffic than a sequential approach resulting in high contention for shared resources and significant increases in waiting times for data. They described modifications to their initial approach that managed to reduce the unnecessary memory traffic. Their study suggests that it is more difficult to attain good performance from multiprocessors with a hierarchical memory system, since the program has to exploit the parallelism within the application without destroying the data access locality.

Torellas et al. [THW90] analyzed the traffic of a Hierarchical Shared-Memory Multiprocessor loosely based on the Stanford's DASH [LLG+90]. They studied the sensitivity of the latency of intercluster and intracluster transactions, the locality in shared data referencing, and the amount of data sharing. Their results confirmed the importance of data reference locality, and the amount of data sharing in the application. Their model also showed that the performance behavior of a hierarchical shared-memory multiprocessor is sensitive to the bandwidth within a cluster.

It is often difficult to understand the effectiveness of particular architecture choices in isolation from the software policies guiding the hardware. Bolosky and Scott [BS91] used a trace-driven dynamic programming algorithm, and computed the policy decisions that would maximize the system performance for a given program execution. The off-line optimal analysis is interesting in that it provides a baseline against which to compare real policy behavior. The authors applied this technique to investigate migration and replication policies. Although the off-line analysis allows the investigation of a relatively large range of shared memory architectures, their model assumed that contention is not a major factor in performance, which is not the case for many real parallel applications.

Recently several techniques have been developed to cope with the impact of latencies on performance and scalability. Gupta et. al. [GHG+91] compared four techniques for reducing memory latency on a uniform basis. They evaluated hardware coherent caches, relaxed memory consistency, software controlled prefetching, and multiple-context support, both individually and in combinations. Their simulation results showed that the major gains in performance are attributed to coherent caches, and that the improvements due to relaxed consistency were less pronounced. The effectiveness of prefetching varied with the type of instructions and the consistency model assumed. It was observed that the reads were more successful in reducing stalling than writes under the strict consistency model. Finally, multithreading provided mixed results depending on the cost of context switching. Later work by Gharachorloo et. al [GGH92], showed that dynamic scheduling in combination with relaxed consistency models is successful in hiding a substantial fraction of the read latency. Their results however assumed a high bandwidth memory system, and ignored the effect of contention in the network and at the memory system.

Wilson and LaRowe evaluated the memory performance of the Galactica Net, a distributed shared-memory multiprocessor under development at the Center for High Performance Computing [WL92]. They combined hardware and software techniques to ensure the coherency of shared data. They used a trace-driven simulation to compare the sequential, weak, and release consistency models within the context of the Galactica Net. Their preliminary results show that the weaker consistency models were successful in hiding the latency of memory update operations, thus increasing the scalability of the architecture.

In the compiler world, much effort has been devoted to better manage memory accesses by taking into account the hierarchical structure of the memory [WL91]. Methods that improve locality and adjust to the dynamic pattern of the memory references were found to be effective not only in shortening the latency, but also in reducing the bandwidth requirements of each processor, thereby reducing contention in the memory system. Hudak and Abraham [HA92] introduced compiler techniques that classify shared data according to how it is used. Each class is then associated with specific consistency maintenance requirements. They implemented ADAPT (Automatic Data Allocation and Partitioning Tool) to analyse parallel loops for the BBN TC200, a scalable shared-memory multiprocessor.

In related work Gamsa [Gam92] suggested involving the programmer in the process of coherency management, taking advantage of the availability of knowledge about the data structures and the sharing patterns among the threads. It is the user's task to define the domain of sharing and to call the proper synchronization routines to reserve access to the *shared-regions*. An implementation of this approach showed increased performance when the reuse rate is high enough to amortize the cost of coherence management overhead.

In this paper, we investigate the effect of non-uniform traffic patterns on memory performance. We analyze the performance impact of such contention, and assess potential solutions to the high latency problem. We examine these solutions in the context of one specific shared-memory multiprocessor architecture, Hector.

The rest of this paper is organized as follows. We first describe our approach. We then describe the software model used in this study. Section 3 presents the analysis of the model. In Section 4 we validate our analytic model against simulation. The benefits of replication are examined in section 5. Section 6 evaluates the impact of memory request queueing on system performance. In Section 7 we evaluate the sensitivity of the memory performance to changes in background traffic. Finally, we summarize our findings in section 8.

2 Methodology

In a system with non-uniform memory access times (NUMA), many factors interact in complex ways to produce the behavior of the memory system. It is difficult to isolate the performance impact of individual components as they contribute to the whole. This interaction may be evaluated in many ways: simulation, analytical modeling, and experimentation with real implementations. The last is the most convincing of all the techniques as it tends to make the fewest assumptions and abstractions. However, experimentation has its limitations. For instance, it is impractical to conduct a comparative study of various design alternatives within reasonable time and under a limited budget. In addition, it may be difficult to draw useful lessons and to interpret results, as they tend to be concealed by implementation quirks. The ability to predict a proposed system's performance without having to construct it is highly desirable. Our approach is therefore analytical in nature. This section describes the model: first, we present the hardware platform and its architectural parameters, and then we characterize the application workload.

2.1 Overview of The Hector Architecture

Hector is a scalable distributed shared memory multiprocessor being developed at the University of Toronto [VSLW91] (see Figure 1). It has a hierarchical interconnection structure in which the cost of a memory access grows incrementally with the distance between the processor and the memory location. The lowest level of the hierarchy consists of a set of processor-memory (PM) pairs linked together by a bus forming a station. Each PM includes a MC88100 processor running at 16MHz, two distinct 16KB instruction and data caches, 16MB of memory, and various I/O interfaces. Several stations are interconnected by a bit-parallel local ring. Local rings are in turn connected by a global ring. The rings are unidirectional bit-parallel slotted rings. The busses and rings can transfer data independently of each other so the aggregate bandwidth scales with size. Hector provides a single global address space, within which each PM is assigned a unique range of addresses. All processors can transparently access all memory locations. Information transfer takes place in a packetsized format using an asynchronous packet transfer scheme. Only one outstanding memory access request per processor is allowed. The NUMAness of Hector is considered mild (compared to other scalable systems [ALKK90, Dun92, LLG$^+$90]) as it takes 20 cycles to read a cache line from local memory, but only one more cycle per ring segment to read from a remote memory.

In the Hector prototype, remote memory access requests that reach a memory module that is already busy serving a request are rejected and must be retried. The retry scheme built into the Hector prototype involves requiring a processor to wait for a backoff interval

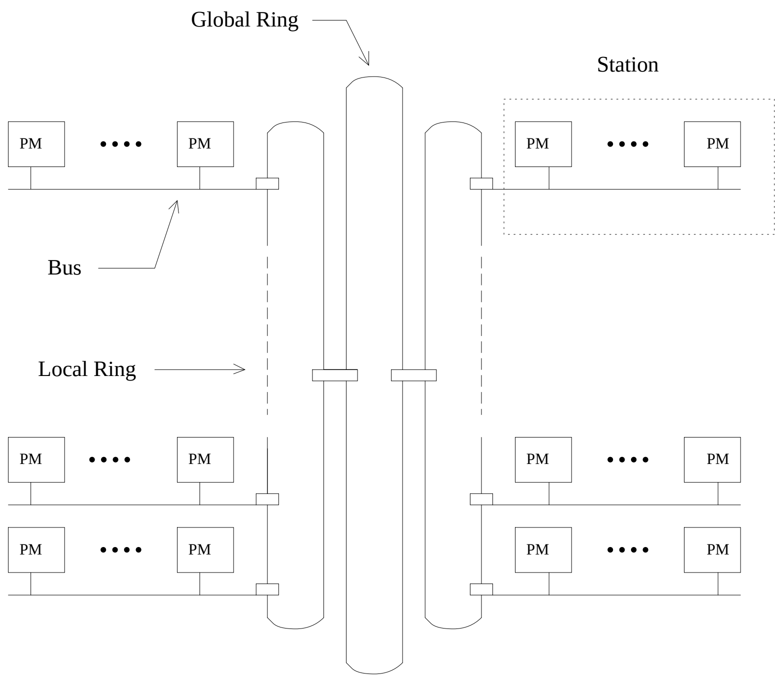


Figure 1: HECTOR: A hierchical shared memory multiprocessor

(of length t_b) before retrying a rejected request. If queues are added to each memory module, fewer requests need be rejected, and the load on interconnection network components due to retry traffic can be reduced. We will use M to denote the number of remote requests (including the one in service) that can be queued at a memory module. Our prototype presents a three-level (four including the cache) memory hierarchy with 4 processors per station, and 4 stations, for a total of 16 processors in the system (see Figure 2). We use the following symbols to denote the model hardware parameters.

- n is the number of stations in the system.
- k is the number of processors in each station.
- M is the number of requests that can be queued at a memory module.
- b_b is the bus service time per memory operation.
- b_r is the ring service time per memory operation.
- S_m is the number of cycles that a memory is busy serving a single request.
- C_l, C_o, C_f are latencies to service memory accesses at different levels of the hierarchy in the absence of contention. The indices l, o, and f denote local, on-station, and off-station respectively.
- t_b is the backoff time (a key parameter that controls the backoff mechanism protocol).

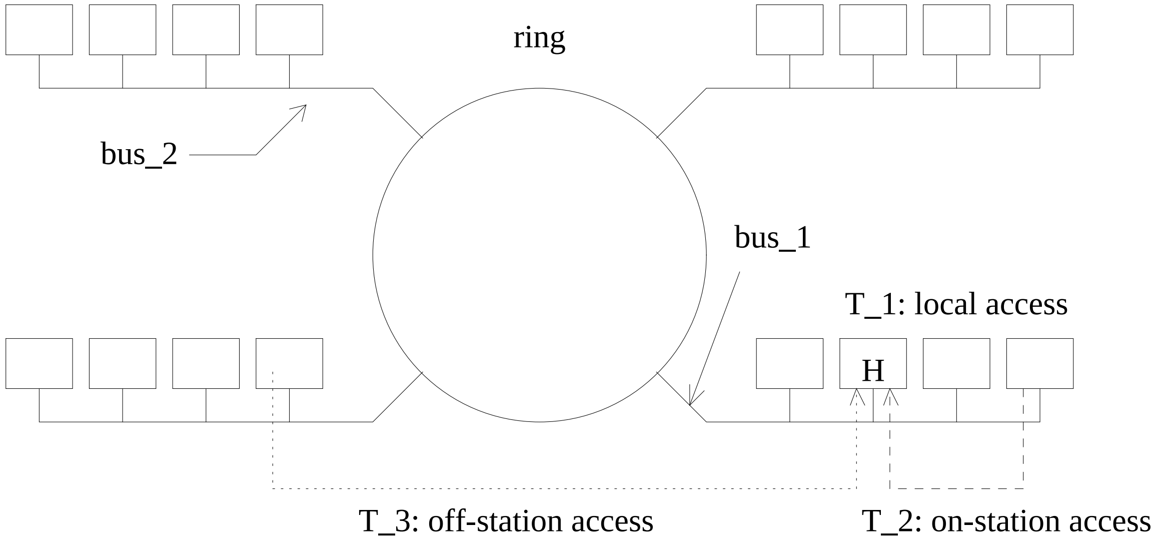


Figure 2: The Hector Prototype

2.2 The Workload Model

It is often possible to partition an application into roughly steady state phases [Gam92], where each phase involves a different memory access pattern. To estimate the total response time, we sum the response time of each phase. Such partitioning helps pinpoint memory system bottlenecks, since the performance attained by an application is largely defined by its memory reference behavior. A particular type of phase of interest is the hot spot access model, in which all (or most) processors are trying to access a single memory module. Hot spots are poorly understood and are often the cause of disappointing performance. In the face of contention, evaluating latencies is more challenging because of the effect of priorities and retries of unsuccessful requests. We anticipate that the contention problem will become even more important in the future as the difference between processor speed and main memory speed increases. Hot spot access is quite common in parallel applications. It may occur due to widespread sharing of synchronization variables such as locks [AC89], or the result of false sharing, as has been observed by Bolosky, et al. [BSF+91]. This type of sharing can drastically impair performance in large-scale systems. Our workload model is thus a non-uniform pattern consisting of a hot spot of higher access rate superimposed on a background of uniform memory request traffic. The background traffic mirrors the application's memory references to the other modules. It can also represent the behavior of competing applications in a multiprogramming environment, or it may reflect some traffic needed to maintain cache coherency. The primary model workload parameter is g, the average number of cycles between physical access attempts to the hot spot module from each other processor. The background traffic is specified by the parameters I_m, I_r, I_d, I_s that define the background traffic utilizations of the memory, the ring, the hot spot bus, and the other busses respectively.

3 Analytical Model

This section presents an analytical performance model of the memory system of the Hector prototype. It allows the quick exploration of a large variety of design variations. The model is derived from one defined by Sevcik and Zhou to evaluate the benefits and limitations of large NUMA multiprocessors [SZ91]. The queueing model represents each congestion point in the system as an M/M/1/N queue, augmented to include Hector's peculiarities such as the effect of priorities among the different classes of requests, the retry algorithm and the optimization of negatively acknowledged packets (where no explicit negative acknowledgement packet is transmitted over the destination station when a request cannot be received) . The general modeling approach, which can be adapted to other systems, is described in another paper [HS92].

3.1 Response Time Equations

The response time of a memory request includes the execution time between requests to main memory g, the cost of the memory reference to some level of the memory hierarchy, and the delays due to contention. The contention may occur both along the communication path and at the memory module itself. The delays depend on the relative location of the issuing processor with respect to the hot spot. The processors in Hector can only support one outstanding request, meaning that each processor has to wait for the outcome of one memory reference before it can issue another request. Consequently, the issuing rate from a processor is inherently limited. If we denote by T_l, T_o, and T_f the per class response time associated with local, on-station and off-station memory requests respectively, then the various response times are given by:

$$T_l = g + C_l + d_l \quad (1)$$

$$T_o = g + C_o + d_o + 2d_d \quad (2)$$

$$T_f = g + C_f + d_f + d_s + 2d_r + d_d \quad (3)$$

where d_l, d_o, d_f are the average memory contention delays incurred at the memory module by requests that originated locally, on-station and off-station respectively. The terms d_d, d_s, and d_r represent the delays incurred by remote requests traveling across the network components. d_d is the delay experienced at the destination bus, bus_1, d_s represent the latency at the source bus, bus_2, and d_r refers to the congestion at the ring. The factor two (in eqs. (2) and (3)) accounts for the fact that a read request crosses certain components both on the way to and on the way back from the hot spot. (Because requests entering a station bus from the ring have preemptive priority, off-station requests encounter delays at the source and destination busses only once each.)

3.2 Delay Equations

In order to determine the delays, we must first derive the utilizations of the various components. Estimates of the utilizations are given by:

$$U_m = I_m + S_m[\frac{k-1}{T_o} + \frac{k(n-1)}{T_f} + \frac{1}{T_l}] \quad (4)$$

$$U_d = I_d + 0.5b_b[\frac{k-1}{T_o}R_o + \frac{k(n-1)}{T_f}R_f] + b_b[\frac{k-1}{T_o} + \frac{k(n-1)}{T_f}] \quad (5)$$

$$U_d^* = 0.5b_b\frac{k(n-1)}{T_f}R_f + b_b\frac{k(n-1)}{T_f} \quad (6)$$

$$U_r = I_r + b_r\frac{k(n-1)}{T_f}(R_f+1) \quad (7)$$

$$U_s = I_s + b_b\frac{k}{T_f}(R_f+1) \quad (8)$$

where R_o, and R_f are the average number of retries on the hot spot module required per memory access for on-station and off station requests respectively. I_m, I_d, I_r, and I_s are the components of the utilizations due to background traffic. U_d^* is the contribution of off-station requests to the utilization of bus_1. The 0.5 factor reflects the optimized implementation of the ring, where a negative acknowledgement can be delivered without sending a separate packet over the bus from the destination station when a request cannot be received.

3.2.1 Delay Equations at the network level

The above utilizations permit us to estimate the delay in accessing the bus as:

$$d_d = \frac{b_b}{1-U_d^*}\frac{U_d[1-kU_d^{k-1}+(k-1)U_d^k]}{(1-U_d)(1-U_d^k)} \quad (9)$$

Bus_1 is modeled as an M/M/1/N queueing system. The second factor of the product is the expected number in a system that has a limit, $k-1$, on the number of customers that can be in the system (since one of the k processors of the station has the hot spot in local memory, hence, bypassing bus_1). The first term of the product is an expansion factor that accounts for the higher priority given to the off-station requests over the on-station ones. On-station requests are granted whenever (1) there is no incoming packet on the ring destined for this station, and (2) none of the station PMs has requested an off-station transfer. The expected delay incurred at the source bus, bus_2, for off-station reads is given by:

$$d_s = \frac{b_b}{1-\frac{n-2}{n-1}U_r}\frac{U_s[1-(k+1)U_s^k+kU_s^{k+1}]}{(1-U_s)(1-U_s^{k+1})} \quad (10)$$

An M/M/1/N queueing system has also been used to model bus_2, with a limit, k, on the number of customers that can be in the system. Because the off-station requests are granted only when there is a free slot on the local ring, the expected number in the system is inflated by $\frac{1}{1-\frac{n-2}{n-1}U_r}$ to reflect the above condition. The $\frac{n-2}{n-1}$ factor adjusts for the fact that a request will never have to queue behind another request from the same processor. Finally a simple M/M/1 model is used to describe the contention observed at the ring. d_r is thus given by

$$d_r = b_r\frac{U_r}{1-U_r} \quad (11)$$

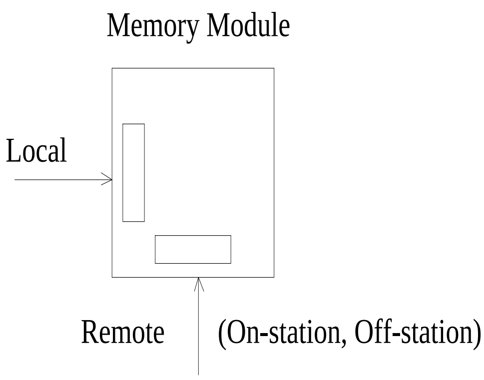


Figure 3: Layout of the buffers at the memory module

3.2.2 Delay Equations at the memory level

$$d_l = 0.5S_m(U_m - \frac{S_m}{T_l}) \tag{12}$$

In the actual memory implementation of Hector, there is an alternation between serving local and remote requests. This results from the fact that local and non-local requests have separate single buffers, as shown in Figure 3. This results in delays for memory service to local requests due to remote requests. $\frac{S_m}{T_l}$ is the contribution of the local processor to the utilization of the hot memory module. When a request packet is negatively acknowledged, it is retransmitted after an interval of time, t_b, the backoff time. For each retry, the total of wasted cycles is the sum of the backoff interval, the memory access latency, and the network contention.

$$d_o = (t_b + d_d + C_o - S_m)R_o + \beta \tag{13}$$
$$d_f = (t_b + 2d_r + d_s + C_f - S_m)R_f + \beta \tag{14}$$

$$\beta = S_m(1 - U_m)\sum_{i=1}^{M-1} iU_m^i \tag{15}$$

The second term, β, of d_o, and d_f is the memory service time times the expected number of requests in a buffer of size M. When $M = 1$, this quantity is zero because requests cannot queue.

3.3 Estimates for number of retries

The average number of retries is difficult to estimate because of the complex relationship between the rejection probability and the frequency of polling the memory cell. When a memory module cannot enqueue all the requests made to it, the requests that find the queue full must be rejected and later retried. We estimate the probability of rejection as a function of the memory module utilization. For processor k, the probability of failure on attempts to access a given memory (memory plus buffers) is approximately

$$\begin{aligned} p_k &= \text{Prob[memory is busy} \mid \text{processor } k \text{ is not using it]} \\ &= \frac{U_m - U_m(k)}{1 - U_m(k)} \end{aligned}$$

where $U_m(k)$ is the utilization of the memory system due to the requests associated with processor k. The conditional probability is due to the fact that processor k does not see its own contribution to the memory system utilization. If we assume that the probabilities of failure on various access attempts are independent, then the expected number of retries is

$$R = \sum_{i=1}^{\infty} p_k^i = \frac{p_k}{1 - p_k} = \frac{U_m - U_m(k)}{1 - U_m}$$

Let P_b denote the probability that the request queue at a memory module (with a buffer of size M) is full. If request arrivals are distributed uniformly in time, P_b is the same as the blocking probability (the probability that memory requests are turned away). If we assume a geometric queue length distribution truncated at M then

$$\begin{aligned} P_b &= 1 - [(1 - U_m) + U_m(1 - U_m) + U_m^2(1 - U_m) + ... + U_m^{(M-1)}(1 - U_m)] \\ P_b &= 1 - (1 - U_m^M) = U_m^M \end{aligned}$$

Based on assuming independent trials (i.e., that the retry interval is sufficiently long), the expected number of retries for on-station and the off-station retry numbers can be approximated for small values of M by:

$$R_o = \frac{\alpha(U_m, t_b)}{1 - U_m^M}(U_m^M - \frac{S_m}{T_o}) \tag{16}$$

$$R_f = \frac{\alpha(U_m, t_b)}{1 - U_m^M}(U_m^M - \frac{S_m}{T_f}) \tag{17}$$

The probability of finding the buffer full on the second and successive tries is in fact dependent on previous events, and on the backoff value. Several tests showed that the number of retries is thus intimately related to the utilization of the memory, and the frequency of polling. This dependence is captured by $\alpha(U_m, t_b)$. During the evaluation of the model, we approximated $\alpha(U_m, t_b)$ with a constant, $\alpha = 2$ which was found to lead to good validation relative to simulation. Extending the model to account for a more accurate function α introduces additional complexity to the solution without providing further insight into the issues at hand.

3.4 Evaluation of the model

The general approach to evaluating the model is outlined as follows:

1. Set all utilizations and delays to zero, and initialize the background traffic by setting I_m, I_d, I_r and I_s.

2. Iterate as follows
   2.1 Use equations (16) and (17) to compute the expected numbers of retries.
   2.2 Estimate the delays d_d, d_s, d_r, d_o, d_l, and d_f using the equations (9), (10), (11), (12), (13), and (14).
   2.3 Estimate the utilizations by applying equations (4), (5), (6), (7), and (8).
   2.4 Evaluate the response times T_l, T_o, and T_f using equations (1), (2), and (3).

   Repeat step 2 until successive estimates of memory utilization are sufficiently close.

3. Obtain performance measures from the final iteration.

4 Model Validation

In order to investigate the accuracy of the analytical model, we compared its predictions to those of an event-driven simulator of Hector called NetSim. NetSim multiplexes the requests of processors among the various activities in the simulated hierarchical memory system to provide accurate information about timing delays and behavior of the memory and the network under a given workload. It is easily configured to simulate a wide range of variants of Hector, hence, giving insight into issues like design scalability and memory network interactions. NetSim was written using the *smpl* [Mac87] simulation library, and was designed with a modular structure to simplify changes to specific parts of the system. It includes modules for the inter-ring interfaces, the station controllers, and the station-bus-interfaces. In addition, it implements several workload models and backoff protocols, and allows the user to vary parameters such as number of processors per station, number of stations within the ring, network channel widths and buffer sizes, relative speed of memory and ring, and memory and cache sizes.

The simulator itself was validated with measurements obtained from experiments on the Hector prototype [HKS91]. Experimentation showed it was necessary to account for the effects of memory refresh, for accurate simulation. NetSim gathers a large set of execution-level statistics, such as average response times, various hierarchical delays, various average number of retries, and utilizations of different components of the system. The simulation results of mean values reported have confidence intervals of 10% or less at a 95% confidence level. In our simulations, we used the actual parameters from the Hector prototype wherever possible. Table 1 shows the parameters used for our validation experiment, in which no background traffic was assumed ($I_m = I_d = I_r = I_s = 0$). Initial iterations of each simulation run were discarded to avoid transient start up effects.

Figure 4 shows the component utilizations under various workload intensities, comparing analytic and simulation results. As g gets smaller, the bus accommodating the hot spot, bus_1, becomes overloaded due to an increasing number of packets rejected while attempting to access the same memory module. The utilization curves illustrate that congestion at one memory module propagates to the rest of the system (not just the memory module in question), degrading the entire system performance. Figure 5 reveals that the memory

$b_b = 2$	$b_r = 1$	$S_m = 6$
$C_l = 10$	$C_o = 18$	$C_f = 22$
$t_b = 11$	$C_r = 1$	$C_b = 1$
$n = 4$	$k = 4$	$M = 1$

Table 1: Parameters used for validation experiments.

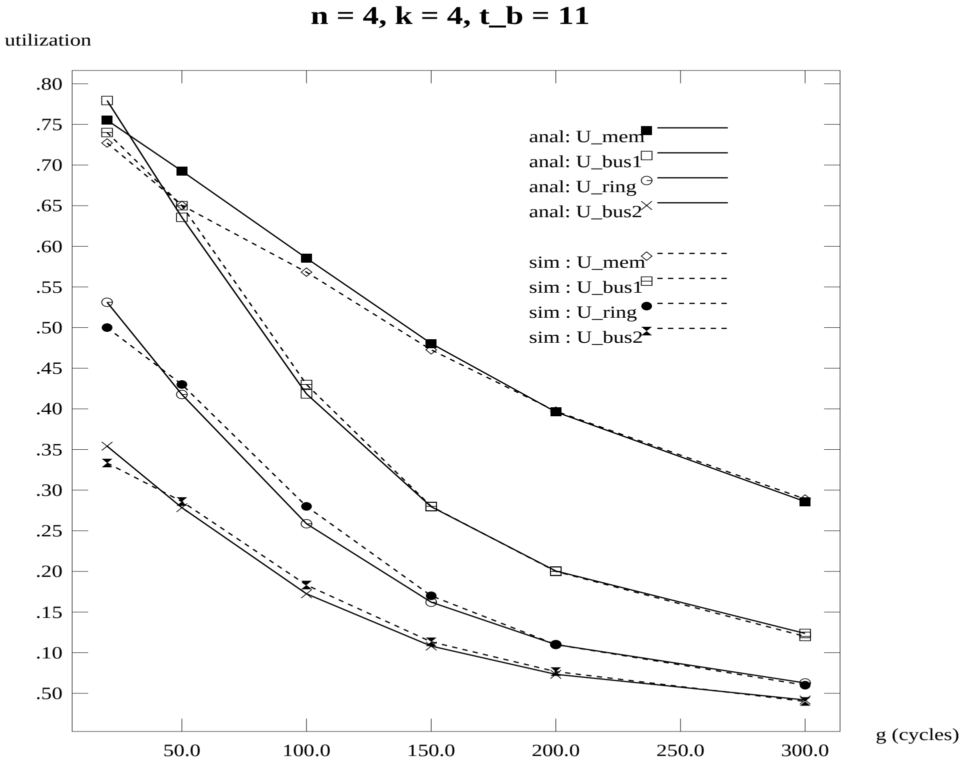


Figure 4: Component utilizations versus g

access times are highly sensitive to the nonuniformity of the workload traffic distribution. It is important to realize that for values of g less than 150, a small change in the mean time between memory references produces a disproportionately significant increase in delays. Local accesses are immune to the nature of the workload, hence rewarding good exploitation of locality. The discrepancy between local and remote latency is high enough that it cannot be neglected when developing applications for Hector. Finally, we notice that the difference between on-station and off-station access times remains acceptable due in part to the priority in bus access given to off-station requests. Within the framework of our model, we found that memory requests can be classified with respect to the level in the hierarchy from which they originate. This categorization suggests that ignoring the relative cost of the different requests imposes a significant penalty. Figure 4 and Figure 5 show good agreement between simulation and analysis. Our model is consequently accurate enough to capture the essential performance characteristics of the memory system under non-uniform traffic in which all

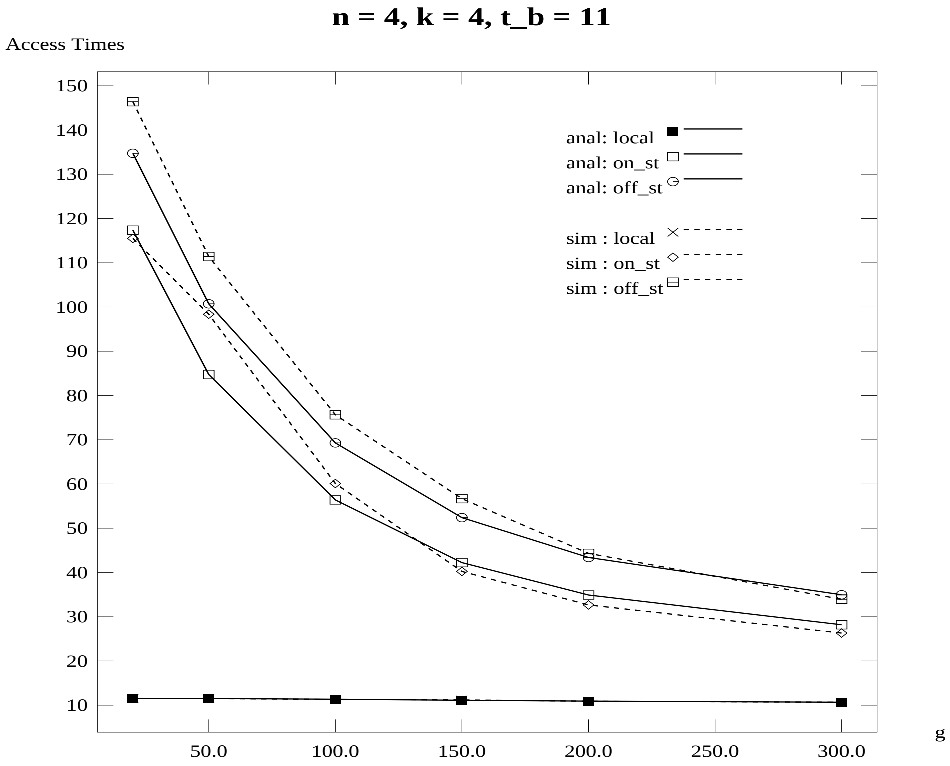


Figure 5: $T_l - g$, $T_o - g$, and $T_f - g$ versus g

processors attempt to access a single hot memory module.

5 Benefits of Replication

The performance evaluation of parallel programs indicates that good locality of reference is crucial for scaling up the performance of parallel code [LHE92, WL91]. Replicating selected data structures in various parts of memory is one way of increasing locality by reducing the number of remote references. In this section we assess the potential advantage of replication. We consider here two way replication. Multiple way replication is analogous, but with potentially more expensive coherence cost. If we assume that each processor accesses the replica that is closest to it, and that for equidistant cases there is an equal probability of accessing either one, then the equations defining the model (established by arguments analogous to those in Section 3) are given in the appendix. Replication can help the performance for three reasons (1) it increases the throughput by reducing the contention, (2) it uses the memory hierarchy more efficiently due to better spatial locality, and (3) it gives an increased cache hit rate (due to a better processor locality).

Figure 6 shows that both on-station and off-station accesses benefit from a less focussed memory traffic. The gain tends to be more significant in regions (i.e. g less than 100) when memory request intensity is high. This is a good indication that replication does indeed alleviate the congestion and hence the delay by spreading the requests among the replicas. Local accesses seem to be indifferent to replication, since their high priority keeps them

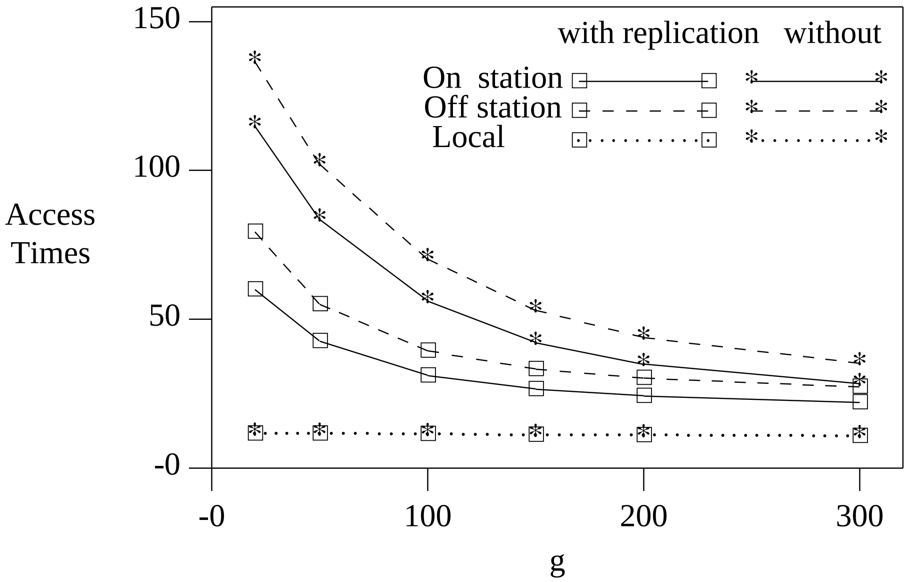


Figure 6: Impact of replication on memory access times

immune to system contention. Replication has the added benefit of reducing the contention along the interconnection network as is illustrated in Figure 7.

The improvement obtained from replication can be far greater than as shown because of an increased cache hit rate, which the model does not take into account. If the data is read only, then the only cost associated with replication is the cost of copying and the extra memory space consumed. In most applications, shared data is updated occasionally, so to insure the correctness of the program, a coherence scheme is required. One has to factor in the extra cost of coherency overhead in order to determine whether the benefits from replication offset the associated cost.

6 Buffering

Providing buffers at the memory level has the effect of reducing the number of rejected requests, allowing the waiting request to grab the memory as soon as it becomes available, thus improving the memory utilization, and consequently the throughput. Figure 8 shows the expected memory access times for both on-station and off-station requests with different buffer sizes. We note that most of the benefit occurs when we move from a single buffer to a buffer of size two. Increasing the buffer size further results in a diminishing return. The impact of buffering on memory utilization is illustrated in Figure 9. As expected, the gain is more significant when contention is high (g less than 150) where request rejection is a problem. Figure 10 demonstrates the extent of reduction in terms of numbers of retries.

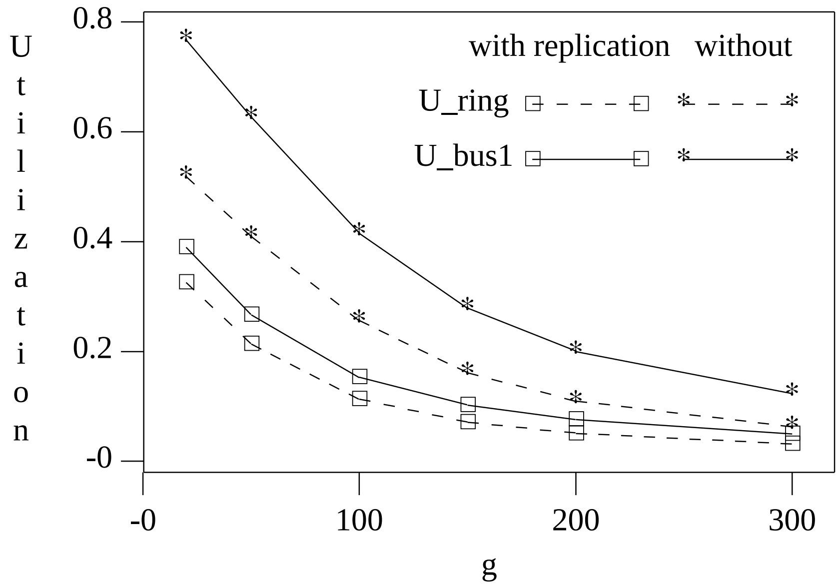


Figure 7: Component utilizations with and without replication.

7 Impact of Background Traffic

In this section, we study the sensitivity of memory performance to changes in the background traffic at the ring level. As I_r increases, the overall utilization of the ring increases, and the average latency of the off-station requests becomes larger. This leads to a lower injection rate and therefore a lower utilization of the memory as is shown in Figure 11.

It was expected that the latency would become higher, but what is surprising is the degree of degradation. Figure 12 shows that an off-station request can take 3.5 times longer due to a background ring utilization of $I_r = .18$. Figure 12 also shows that on-station requests experience lower access times. Because the off-station requests make it to the memory less frequently, the on-station requests are subject to lighter competition, and have better chance to access the memory.

A closer look at this sensitivity to background traffic reveals that presence of the background traffic can cause the ring to saturate (see Figure 13). In the face of background traffic, the ring becomes the bottleneck. Replication is a technique that reduces the number of off-station requests, therefore removing some load from the ring. Figure 14 evaluates the potential benefits of replication for a system subject to background traffic $I_r = .15$. Replication however does not come for free. The act of replicating data and insuring coherency consumes resources such as memory and network traffic. Since memory is cheap, we will consider the extra traffic for maintaining the coherency to be the key cost of replication. This overhead can be modeled as a background traffic at the ring level. (Analysis shows that memory performance is not sensitive to changes of background traffic at the bus level.) Therefore it is safe to consider the extra traffic at the ring to be the key cost of replication. Figure 15 displays the on-station and off-station memory access times due to replication subject to different costs (expressed in terms of I_r). The solid line is the cost of the access time without replication. Whenever the dotted curves lie below the solid one, performance is better with replication than without.

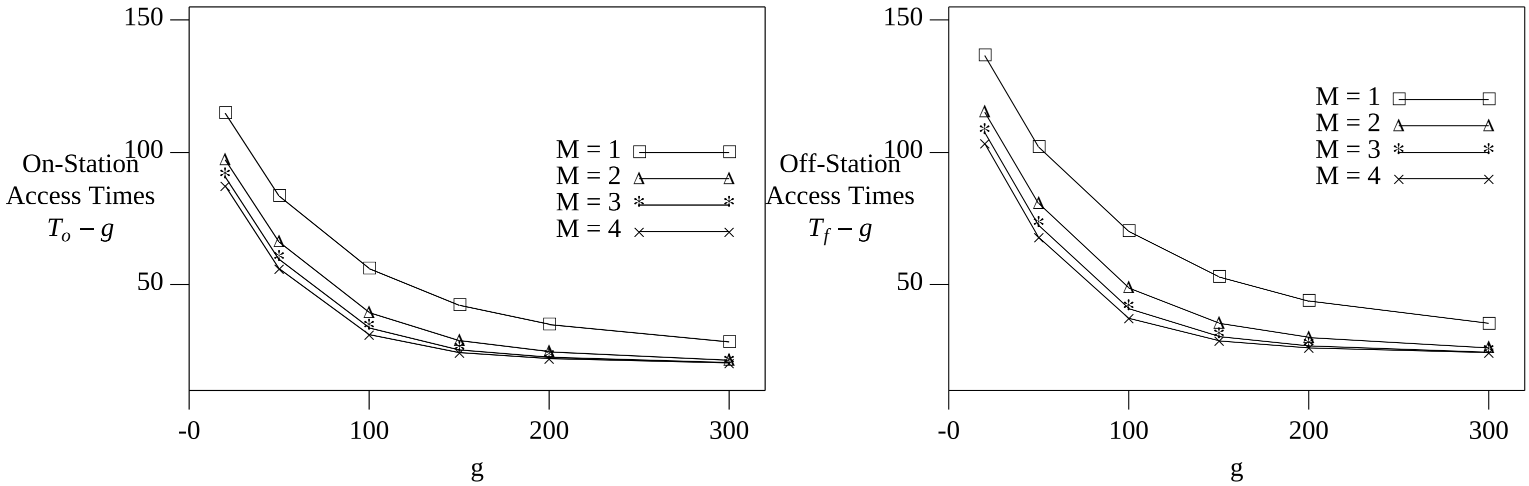


Figure 8: Memory access times for different values of M.

8 Conclusion

Because of their scalability, the class of NUMA shared memory architectures is becoming increasingly important. In this paper we have presented an analytical study of the performance of a large scale NUMA multiprocessor in the presence of non-uniform traffic. We developed an analytic model that captures the main features of this type of architecture, namely the non-uniformity of memory access times. The model includes the effect of increased contention for system resources, and the impact of the bus arbitration algorithm on the network traffic.

To establish confidence in the robustness of our analysis, we validated our model with a detailed simulator that was previously validated with experimentations on the Hector prototype. Our results demonstrate that memory latency is very sensitive to the non-uniformity of access over the memory. If stringent measures are not taken to alleviate the contention, parallel applications can suffer long memory access times. Our results also illustrate that a hot-spot produces effects that severely degrade all network traffic, not just the traffic to shared locks. This effect should be avoided especially in multiprogramming environments where it is possible for a single user to degrade the entire system's performance via hot spot traffic.

Our analysis also assessed the potential performance gain from replication, and quantified the expected improvement in memory utilization from allowing memory request buffering. Our results demonstrate the sensitivity of memory performance to various workloads, and hence the importance of having analytic tools to investigate design alternatives.

Acknowledgements: Our thanks to Darrell Kindred for use of the NetSim simulator and to Michael Stumm for answering innumerable questions about the implementation of the Hector prototype.

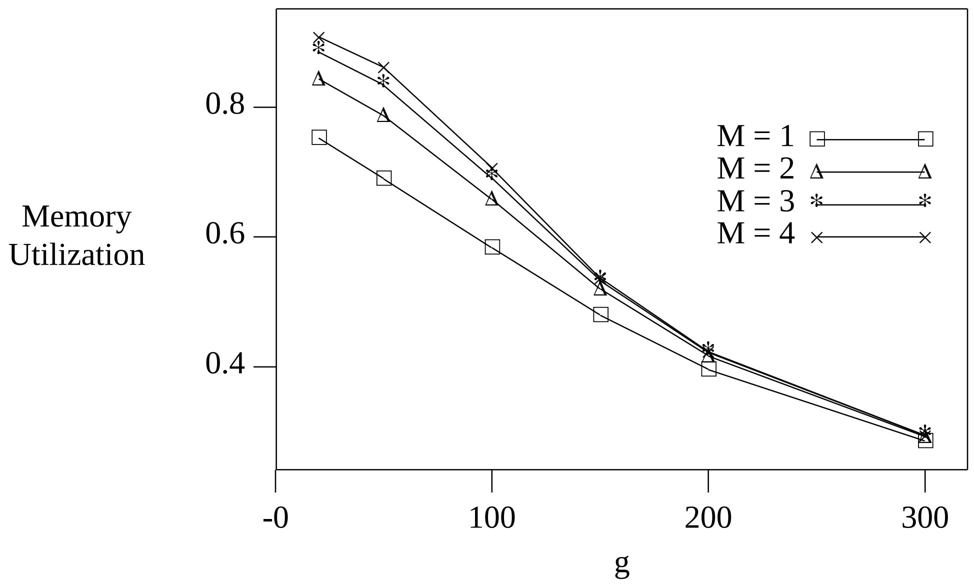


Figure 9: Memory utilization for different values of M.

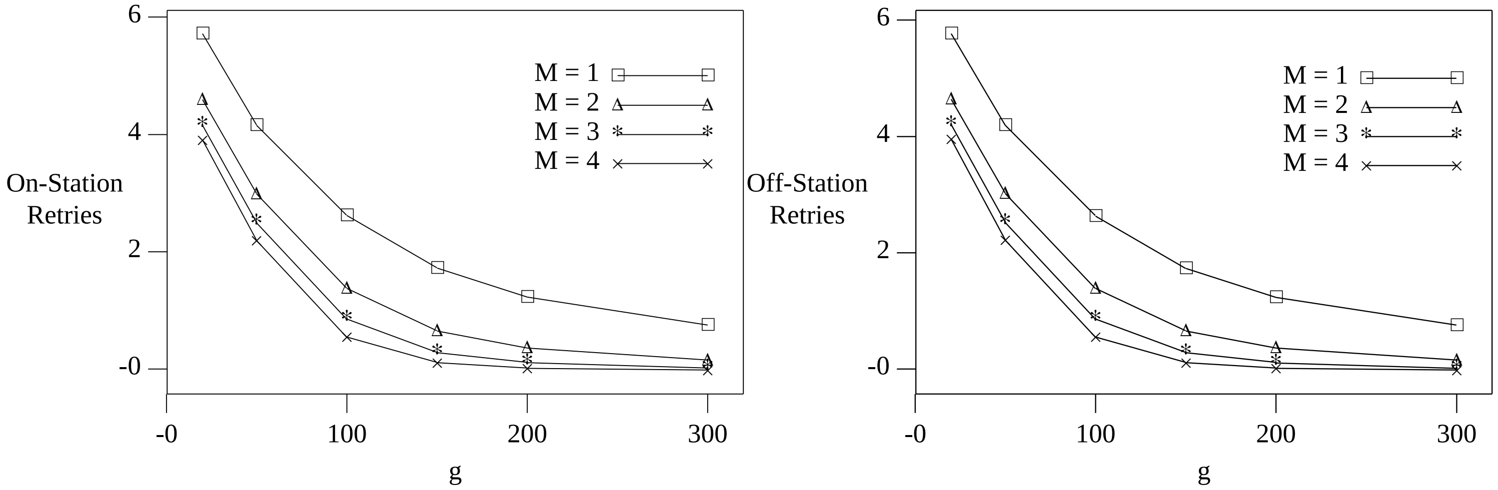


Figure 10: Numbers of retries for different values of M.

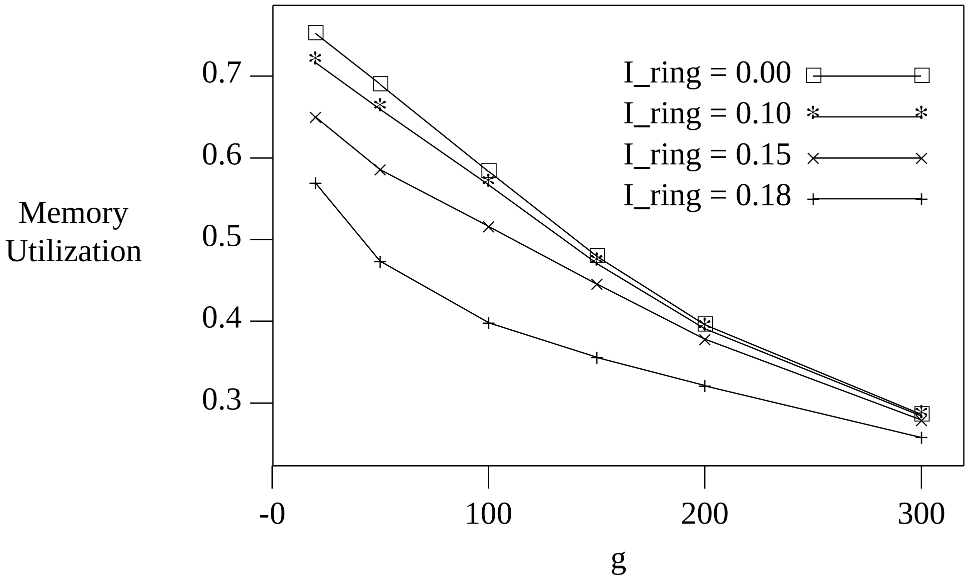


Figure 11: Memory utilization for different values of I_r.

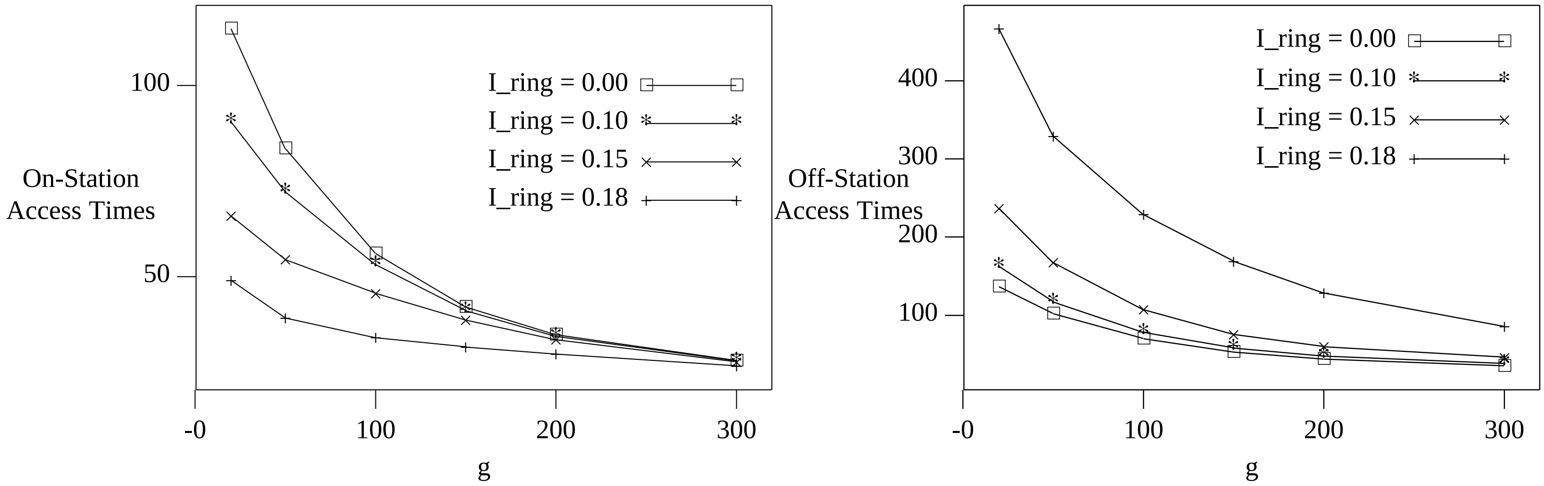


Figure 12: Memory access times for different values of I_r.

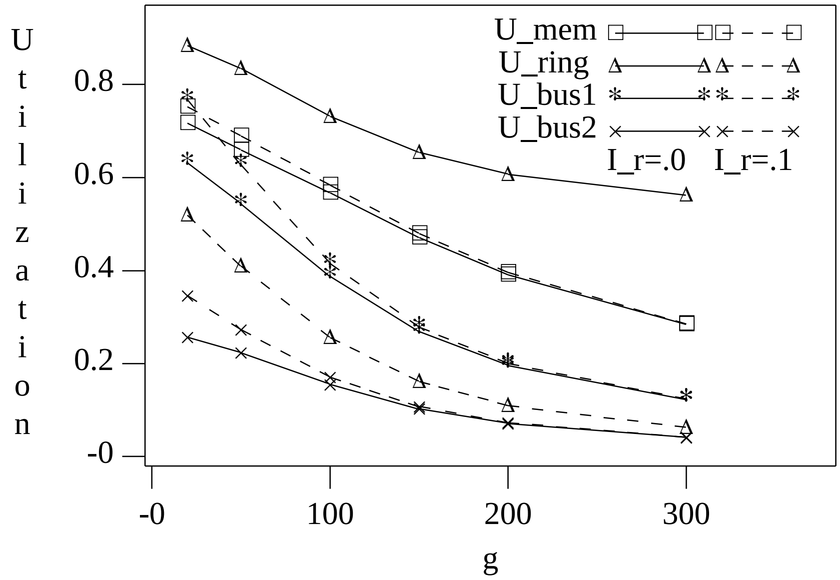


Figure 13: Component utilizations for $I_r = .0$ and $I_r = .1$.

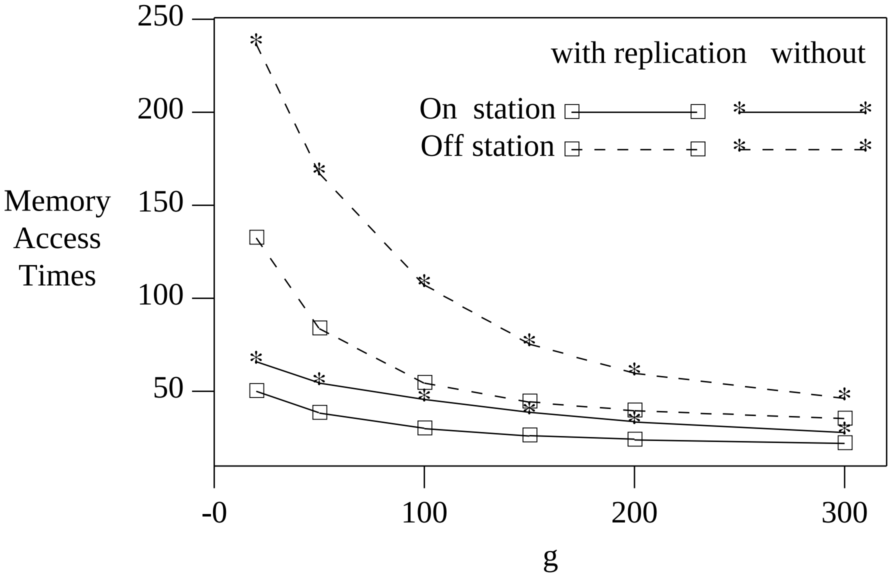
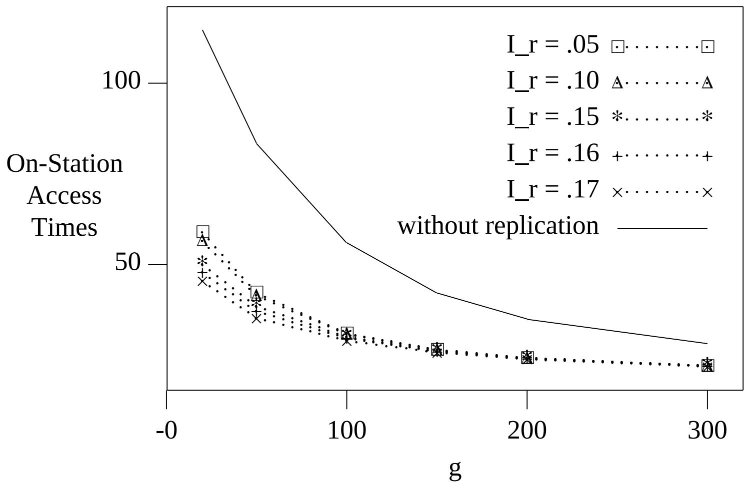


Figure 14: Benefit of replication when $I_r = 0.15$.



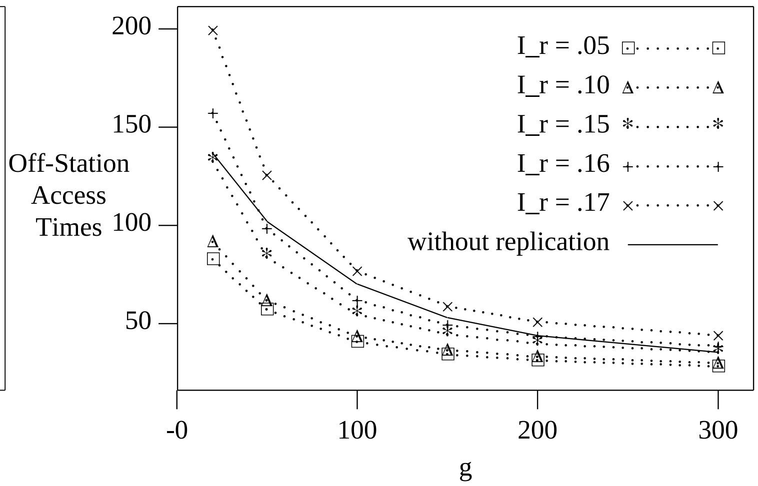


Figure 15: Memory access times with replication for different costs of I_r.

APPENDIX

The equations describing the system model under a replicated workload are as follows:

$$
\begin{aligned}
T_l &= g + C_l + d_l && (1') \\
T_o &= g + C_o + d_o + 2d_d && (2') \\
T_f &= g + C_f + d_f + 2d_r + d_s + d_d && (3') \\
U_m &= I_m + S_m[\tfrac{k-1}{T_o} + \tfrac{k(n-2)}{2T_f} + \tfrac{1}{T_l}] && (4') \\
U_d &= I_d + 0.5b_b[\tfrac{k-1}{T_o}R_o + \tfrac{k(n-2)}{2T_f}R_f] + b_b[\tfrac{k-1}{T_o} + \tfrac{k(n-1)}{T_f}] && (5') \\
U_d^* &= 0.5b_b\tfrac{k(n-2)}{2T_f}R_f + b_b\tfrac{k(n-2)}{2T_f} && (6') \\
U_r &= I_r + b_r\tfrac{k(n-2)}{T_f}(R_f + 1) && (7') \\
U_s &= I_s + b_b\tfrac{k}{T_f}(R_f + 1) && (8') \\
d_d &= \tfrac{C_b}{1-U_d^*}\tfrac{U_d[1-kU_d^{k-1}+(k-1)U_d^k]}{(1-U_d)(1-U_d^k)} && (9') \\
d_s &= \tfrac{C_b}{1-\frac{n-2}{n-1}U_r}\tfrac{U_s[1-(k+1)U_s^k+kU_s^{k+1}]}{(1-U_s)(1-U_s^{k+1})} && (10') \\
d_r &= C_b\tfrac{U_r}{1-U_r} && (11') \\
d_l &= 0.5S_l(U_m - \tfrac{S_l}{T_l}) && (12') \\
d_o &= (t_b + d_d + C_o - S_m)R_o + \beta && (13') \\
d_f &= (t_b + 2d_r + d_s + C_f - S_m)R_f + \beta && (14') \\
\beta &= S_m(1 - U_m)\textstyle\sum_{i=1}^{M-1} iU_m^i && (15') \\
R_o &= \tfrac{\alpha}{1-U_m^M}(U_m^M - \tfrac{S_m}{T_o}) && (16') \\
R_f &= \tfrac{\alpha}{1-U_m^M}(U_m^M - \tfrac{S_m}{T_f}) && (17')
\end{aligned}
$$

References

[AC89] Anant Agarwal and Mathews Cherian. Adaptive backoff synchronization techniques. In *16th International Symposium on Computer Architecture*, pages 396–406. ACM, May 1989.

[ALKK90] A. Agarwal, B. H. Lim, D. Kranz, and J. Kubiatowicz. April: A processor architecture for multiprocessing. In *17th International Symposium on Computer*

Architecture. ACM, May 1990.

[BS91] William J. Bolosky and Michael L. Scott. University of rochester, computer science department. Technical report, Evaluation of Multiprocessor Memory Systems Using Off-Line Optimal Behavior, September 1991.

[BSF+91] W. J. Bolosky, M. L. Scott, R. P. Fitzgerald, R. J. Fowler, and A. L. Cox. NUMA policies and their relation to memory architectu re. In *4th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 212–221. ACM, April 1991.

[Dun92] T. Dunigan. Kendall square multiprocessor: Early experiences and performance. Technical Report ORNL/TM-12065, Oak Ridge National Laboratory, March 1992.

[Gam92] B. Gamsa. Region-oriented main memory management in shared-memory numa multiprocessors. Master's thesis, University of Toronto, 1992.

[GGH92] Kourosh Gharachorloo, Annop Gupta, and John Hennesy. Hiding memory latency using dynamic scheduling in shared-memory multiprocessors. In *19th International Symposium on Computer Architecture*, pages 22–33. ACM, May 1992.

[GHG+91] A. Gupta, J. Hennesy, K. Gharachorloo, T. Mowry, and W.-D. Weber. Comparative evaluation of latency reducing and tolerating techniques. In *18th International Symposium on Computer Architecture*, pages 254–263. ACM, May 1991.

[HA92] David E. Hudak and Santosh G. Abraham. Compile-time optimization of near-neighbor communication for scalable shared-memory multiprocessors. *Journal of Parallel and Distributed Computing*, 15:368–381, August 1992.

[HKS91] Mark A. Holliday, Darrell S. Kindred, and Michael Stumm. On the scalability of ring-based, hierarchical shared memory multiprocessors. Submitted for publication, September 1991.

[HS92] Karim Harzallah and Kenneth Sevcik. Hot spot analysis in large scale shared memory multiprocessors. Submitted for publication, September 1992.

[LHE92] Richard P. LaRowe, Mark A. Holliday, and Carla Schatter Ellis. An analysis of dynamic page placement on a numa multiprocessor. In *Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 23–34. ACM, June 1992.

[LLG+90] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessey. The directory-based cache coherence protocol for the dash multiprocessor. In *17th International Symposium on Computer Architecture*, pages 148–159. ACM, May 1990.

[Mac87] M. H. MacDougall. Simulating computer systems: Techniques and tools. *MIT Press*, 1987.

[RG92] Edward Rothberg and Anoop Gupta. Parallel iccg on a hierarchical memory multiprocessor-addressing the trinagular solve bottleneck. *Parallel Computing*, 18:719–741, July 1992.

[SZ91] Kenneth C. Sevcik and Songnian Zhou. Performance benefits and limitations of large numa multiprocessors. Technical Report 256, University of Toronto, November 1991.

[THW90] Josep Torellas, John Hennessy, and Thierry Weil. Analysis of critical architectural and program parameters in a hierarchical shared-memory multiprocessor. In *Sigmetric 90*, pages 163–172. ACM, May 1990.

[VSLW91] Z.G. Vranesic, M. Stumm, D.M. Lewis, and R. White. Hector - a hierarchically structured shared-memory multiprocessor. *IEEE Computer*, January 1991.

[WL91] M. Wolf and M. Lam. A data locality optimization algorithm. In *4th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 34–44. ACM, October 1991.

[WL92] A. Wilson and R. LaRowe. Hiding shared memory reference latency on the galactica net distributed shared memory architecture. *Journal of Parallel and Distributed Computing*, 15:351–367, August 1992.