Hot Spot Analysis in Large Scale Shared Memory Multiprocessors *

Karim Harzallah and Kenneth C. Sevcik

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

Technical Report CSRI-280

Currently under revision

January 1993

Computer Systems Research Institute
University of Toronto
Toronto, Canada
M5S 1A1

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

Abstract

Scalable multiprocessors that support a shared-memory image to application programmers are typically based on physical memory modules that are distributed. Consequently, the access times for a particular processor to various parts of physical memory differ. In this paper, we explore the implications of this non-uniformity in memory access times. In particular, we study the effect of hot-spots in hierarchical large scale NUMA multiprocessors. Hot-spot analysis is of interest because coordinated threads of parallel programs lead to hot spots whose impact on performance may be substantial or even dominant. We have developed an analytical model of access latencies and contention for shared resources in the interconnection network that links the processors and memory modules. Our objective is to provide a better understanding of non-uniform memory access times in scalable architectures. We show the extent to which a variable can be shared before it becomes a performance bottleneck, and assess the potential gain from replication of shared data items. We also demonstrate that the backoff value (after a memory request rejection) must be chosen carefully to balance memory access time and network utilization. Finally, we show that memory utilization is improved by allowing memory request buffering.

1 Introduction

Parallel applications can be most easily designed and expressed in terms of a (virtual) global shared memory. To support this image in a scalable system, however, physical memory modules are distributed and an interconnection network is used to carry read and write requests from processors to memory modules. In terms of access latency, each memory module can be considered "close" to some processors, but "distant" from others. Non-Uniform Memory Access (NUMA) architectures make it possible to build large-scale, parallel processing systems that provide a shared memory programming image. Existing commercial systems that can be classified as large scale NUMA include the BBN TC2000 [BBN90], the CM5 from Thinking Machines Inc, and the KSR1 [Dun92] from Kendall Square Research. Among the academic research prototypes, we find the MIT Alewife [ALKK90], the Stanford DASH [LLG+90], and the Hector multiprocessor at the University of Toronto [VSLW91].

Several studies have been conducted to understand and improve the performance of NUMA multiprocessors. Nanda et al. [NSTN91] compared the performance degradation resulting from contention for shared resources in the BBN GP 1000, the BBN TC2000, and the Sequent Balance 21000. They experimented with various workload models, and found that the interconnection latency is the largest source of delay. Their results also showed that, in the presence of widely shared locks, the queueing delay for locks is the most influential factor on performance.

Gupta et al. [GHG+91] compared four techniques for reducing memory latency using a common set of architectural assumptions and benchmarks. 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.

Other work by Zhang and Qin [ZQ91] examined several analytical models to predict and evaluate the overhead of interprocessor communication, processor scheduling, process synchronization, and remote memory access where network contention and memory contention are considered. To support their models, they presented measurements collected from experiments done on the BBN GP 1000. In their treatment of the remote memory access delay, the authors assumed that accesses are uniformly distributed over the shared memory space, thus limiting the scope of potential NUMA applications to evaluate.

Sevcik and Zhou [SZ91] have studied the performance benefits of large NUMA multiprocessors, and assessed their limitations. They introduced the notion of NUMA expansion ratio, which is a measure of how well an application is suited for the non-uniform characteristics of the architecture. Their approach involved both analysis and experimentation. They concluded that data reference locality is an important determinant of an application's performance in a NUMA system.

Another approach to evaluating the interaction between the hardware architecture and the policies managing it is to use off-line optimal analysis. 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.

Related work by LaRowe, Holliday, and Ellis [LHE92] analyzed the memory system performance of local/remote NUMA machines. They explored the effectiveness of dynamic page placement, and in particular, dynamic multiple-copy page placement. Their analysis was based on approximate mean-value analysis techniques. They validated their model against experimental data obtained while running a synthetic workload.

The non-uniformity of memory access effect grows more significant as improvements in CPU speed continue to outstrip improvements in memory performance, forcing future processors to spend a larger fraction of their time waiting for memory [NSTN91] [BSF+91]. Previous studies have shown that ignoring the location of data in NUMA systems leads to a significant performance penalty [SZ91] [AE91]. In this paper, we investigate the effect of a nonuniform traffic pattern consisting of a single hot spot. We analyze the performance impact of such contention, and consider, as one example architecture, the Hector system. Our analysis attempts to answer the following questions within the framework of our model:

- To what extent can a variable be shared before it becomes a bottleneck?
- What can be gained from replication?
- How much is performance improved by buffering memory requests at memory modules?
- What impact does the backoff value after a memory request rejection have on performance?

Our work is different from most previous work in that it replaces the assumption of uniformly distributed memory accesses with an assumption at the opposite extreme: all memory accesses in the system are directed to a single memory module (the "hot spot"). It captures the main features of a NUMA system, while still being simple to understand and to evaluate. Our hardware model involves a multiple level memory hierarchy reflecting the structure found in scalable architectures.

The rest of the paper is organized as follows. In Section 2, we describe the hardware model and the software model assumed for this study. Section 3 presents the analysis of the model. In Section 4, experimental results based on the model are discussed, and we summarize our findings in Section 5.

2 The System Model

We represent a multiprocessor architecture for supporting a shared-memory programming image as a number of processors interconnected in a hierarchical fashion: Each processor has a local memory module. The virtual global shared memory is supported by mapping portions of virtual memory into the physical pages of memory local to some processor. Under some conditions, a single virtual page may be replicated in several physical memory modules in order to localize read accesses for multiple processors. This architecture is generally applicable to several of the shared memory systems mentioned in Section 1. Our analytic approach is also sufficiently general to be relevant to several systems. For the sake of concrete experiments, we specialize the model and its analysis to reflect the Hector multiprocessor.

Hector is a hierarchical NUMA multiprocessor being developed at the University of Toronto [VSLW91]. The lowest level of the hierarchy consists of processing modules (PMs), each comprising of a MC88100 processor running at 16MHz, two distinct 16KB instruction and data caches, 16MB of memory, and various I/O interfaces. Multiple PMs are connected by a bus, forming a station. In the current configuration, every station is composed of four PMs. The stations, in turn, are connected by a unidirectional bit-parallel ring, called the local ring. At the next higher level of the hierarchy, communication between local rings is done via another unidirectional ring called the global ring. Every processor has hardware access to the memories of all processors in the system, forming a large distributed shared memory. The high level architecture of Hector is presented in Figure 1.

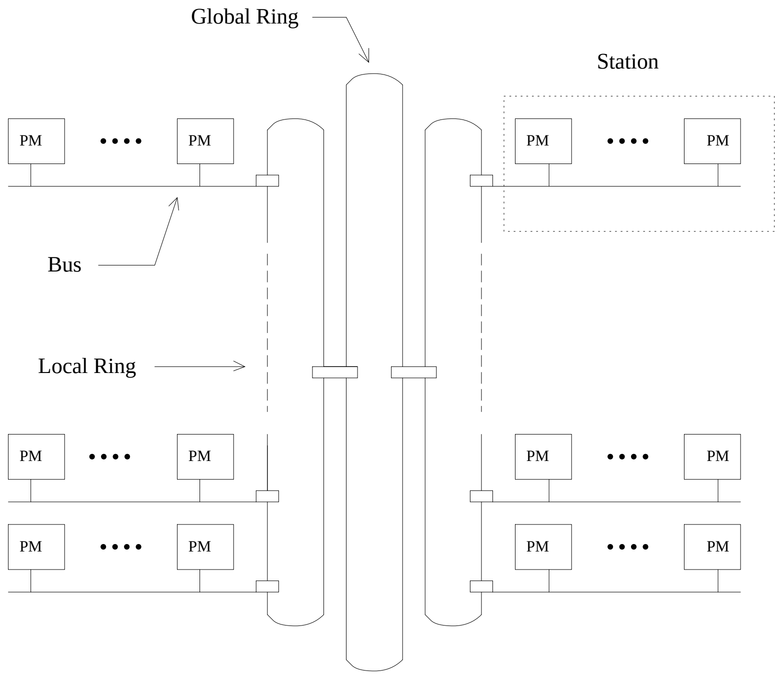


Figure 1: Hector: A hierchical shared memory multiprocessor

We have abstracted away many of the system specific details of Hector in order to keep the model sufficiently general to be of use in the evaluation of other systems. The dimensions of the system are defined by N and K, where N is the number of stations in the system, and K is the number of processors per station making the total number of processors NK. The degree of buffering at each memory module is determined by M, the maximum number of requests that can be queued at a memory module. S_m, S_b, and S_r are the mean service times of the memory, the bus, and the ring (one full cycle) respectively. In the absence of contention, the memory access time varies with respect to the level from which the request is made. We denote by C_i the memory access time from level i. (In our study, $i = 1, 2, 3$ correspond to local, on-station, and off-station access costs respectively.) The model hardware parameters are summarized in Table 1.

2.1 The Software Model

Previous work [SZ91] [AE91] has indicated that code and data placement is critical to good performance in NUMA systems. It is desirable that the data reside in the memory of the processor that is accessing it most frequently. This results in fewer remote accesses, and hence lower latency and less contention. Also, in proposed scalable systems, contention for memory access is a potential

N	the number of stations in the system
K	the number of processors in each station
M	the number of requests that can be queued at a memory module
S_m	the number of cycles that a memory is busy serving a single request
S_r	the ring service time
S_b	the bus service time
C_i	the cost of a memory access from level i (relative to the memory), ignoring contention

Table 1: Model hardware parameters

bottleneck. This is attributable to the fact that variables may be shared among most, if not all, the processors, or, in many cases, to false sharing, as has been observed by Bolosky, et al. [BSF+91]. In shared-memory multiprocessors, locks are commonly used to synchronize activities among processors. This use of synchronization variables often leads to widespread sharing. Unfortunately, this type of sharing can drastically impair performance in large-scale systems. This leads to the conclusion that if stringent measures are not taken to alleviate the contention, the overhead can sometimes offset any benefit from parallelism in programs. Motivated by the importance of the issue, we investigate the implications of traffic non-uniformity. Our workload model is thus a set of threads, each requiring access to the same memory module at the rate of one request every a clock cycles. A hot spot is therefore created simulating high demand for a lock, or a high degree of false sharing of a page, for example.

Although this extreme stress case workload is not a "real" parallel application, as it may not reflect the memory reference patterns of a real program, it is interesting in that it depicts the kernel of misbehaving applications that lead to poor performance. It captures some of the reference behavior of a certain class of programs and provides insight into the potential cost of memory latency in poorly designed applications.

The software model parameters are summarized in Table 2. Caches alleviate memory latency, but cannot eliminate first time references and reloads due to invalidations needed to maintain the consistency of the data. The parameter a is the average number of cycles between physical access attempts to the hot module from each processor. When a memory request is negatively acknowledged, it is retried after an interval of time, $C_{backoff}$ cycles. For mathematical tractability, we will assume a constant backoff interval.

a	the number of cycles between successive memory requests
$C_{backoff}$	the backoff time

Table 2: Model software parameters

3 Analysis of the Model

In this section, we develop an analytical model of a hierarchical NUMA shared memory multiprocessor. The model is based on simple equations describing the system interconnection network. A similar approach has been used elsewhere for treating the case where all memory modules are accessed symmetrically by the processors [SZ91]. The model assumes that read and write requests to memory are serviced in the same amount of time. For our analysis we consider a 24-processor Hector prototype, as illustrated in Figure 2. The main measures of performance addressed are the hot spot memory module utilization and the memory access latencies incurred by processors at various locations relative to the hot spot. Traffic to a single hot spot has a global effect, so we will also examine the implications of contention on the communication bandwidth. Hence, the network utilization will be considered as a measure of performance as well.

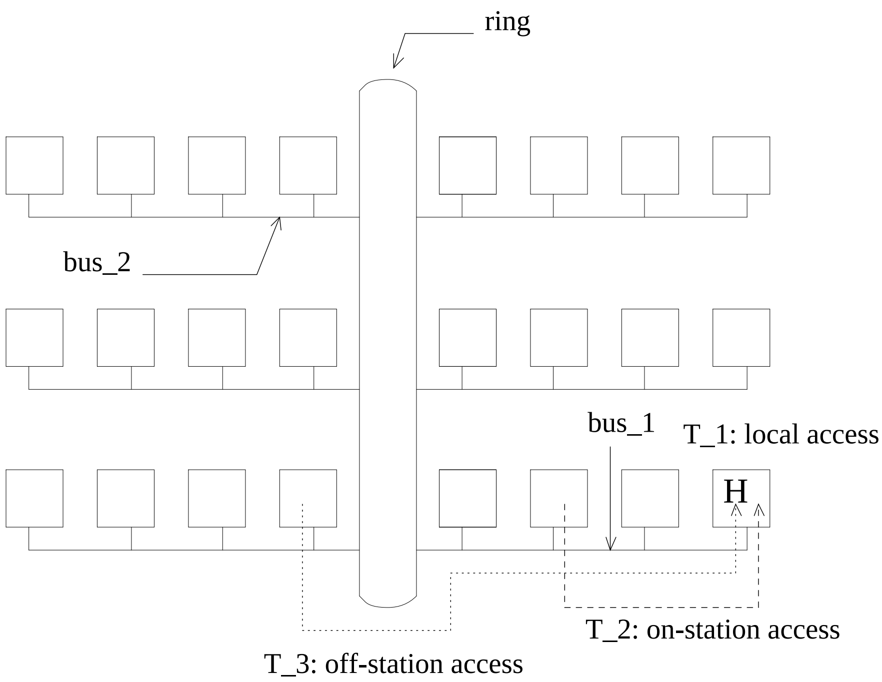


Figure 2: Hector Configuration ($N = 6, K = 4$)

3.1 Response Time Equations

The response time, T_i, of a memory request includes (1) the execution time between requests, a, (2) the time for a memory reference from the ith level of the memory hierarchy ignoring contention, and (3) the delays due to contention. The contention may occur along the communication path and at the memory module itself. The delays depend on the location of the issuing processor relative to the hot spot. Expressions for the various response times are given below:

$$\begin{aligned} T_1 &= a + C_1 + D_1 & (1) \\ T_2 &= a + C_2 + D_2 + 2D_{bus1} & (2) \\ T_3 &= a + C_3 + D_3 + 2D_{bus1} + 2D_{bus2} + 2D_{ring} & (3) \end{aligned}$$

where D_1, D_2, D_3 are the average memory contention delays incurred at the memory module by local, on-station, and remote requests respectively , and D_{bus1}, D_{bus2}, and D_{ring} are the latencies incurred by remote requests traveling across the network components. In figure 2, H denotes the hot spot location, bus_1 is the station bus where the hot spot is located, and bus_2 represents any other bus. Local accesses do not use a bus, on-station accesses use only bus_1, and off-station accesses use the ring and some bus_2 in addition to bus_1. The factors of two account for the fact that a request crosses each network component both on the way to and on the way back from the hot spot. The requests arrive from all processors at the hot spot memory module with total rate λ such that

$$\lambda = \frac{1}{T_1} + \frac{K-1}{T_2} + \frac{K(N-1)}{T_3} \qquad (4)$$

3.2 Delay Equations

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

$$\begin{aligned} U_{mem} &= S_m[\frac{1}{T_1} + \frac{K-1}{T_2} + \frac{K(N-1)}{T_3}] & (5) \\ U_{bus1} &= 2S_b[\frac{K-1}{T_2} + \frac{K(N-1)}{T_3}]V_{mem} & (6) \\ U_{ring} &= S_r\frac{K(N-1)}{T_3}V_{mem} & (7) \\ U_{bus2} &= 2S_b\frac{K}{T_3}V_{mem} & (8) \end{aligned}$$

where V_{mem} is the average visit count to the hot spot module per memory access. This permits us to estimate the delay in accessing memory as:

$$\begin{aligned} D_1 &= (C_{backoff} + C_1 - S_m)R_{mem} + D_{mem} & (9) \\ D_2 &= (C_{backoff} + C_2 - S_m + 2D_{bus_1})R_{mem} + D_{mem} & (10) \end{aligned}$$

$$D_3 = (C_{backoff} + C_3 - S_m + 2D_{bus_1} + 2D_{ring} + 2D_{bus_2})R_{mem} + D_{mem} \quad (11)$$

$$D_{mem} = S_m(1 - U_{mem}) \sum_{i=1}^{M-1} iU_{mem}^i \quad (12)$$

where R_{mem} is the average number of retries on the memory ($V_{mem} = 1 + R_{mem}$). The second term, D_{mem}, is just the memory service time times the expected number of requests in queue. The expected delay at the bus and at the ring can be approximated by the product of the estimated queue length and the service time per request.

$$D_{bus1} = S_b \frac{U_{bus_1}}{1 - U_{bus_1}} \quad (13)$$

$$D_{ring} = S_r \frac{U_{ring}}{1 - U_{ring}} \quad (14)$$

$$D_{bus2} = S_b \frac{U_{bus_2}}{1 - U_{bus_2}} \quad (15)$$

3.3 Retry Equations

When a memory module cannot enqueue all the requests made to it, the remaining requests must be rejected and later retried. We may estimate the probability of rejection as a function of the memory module utilization. For any particular processor k, the probability of rejection on each attempt to access a given memory module with a buffer of size M can be approximated by:

$$\begin{aligned} p_k &= \text{Prob[the } M^{th} \text{ slot of the buffer is occupied} \mid \text{processor } k \text{ is not using it]} \\ &= \frac{P_B - P_B(k)}{1 - P_B(k)} \end{aligned}$$

where P_B denotes the probability that the request queue at the memory module is full, and $P_B(k)$ is the blocking probability seen by processor k. The conditional probability is due to the fact that processor k does not see its own contribution to the memory module's utilization. We would like to express P_B as a function of U_{mem}, where U_{mem} is the utilization of the memory. If we assume a geometric queue length distribution, then

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

If we assume that the probabilities of failure on various access attempts are independent (i.e., that the retry interval is sufficiently long), then the expected number of retries is

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

In the case of equal contribution among the m processors (towards the utilization of the memory module), the expected number of retries simplifies to

$$R = \frac{p_k}{1 - p_k} = \frac{m-1}{m} \frac{P_B}{1 - P_B}$$

(Note that the assumption of equal contribution is seldom correct in a NUMA architecture, as the contribution of a processor may depend on its distance from the memory.) 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. To account for this dependency, we assign higher than P_B probability of rejection for the retries generated after a small backoff time. Let $H(i)$ denote the probability of rejection associated with the ith try. Assuming that an immediate retry fails with certainty, that in equilibrium a request fails with probability P_B, and that the dependence decays exponentially, we let

$$H(i) = P_B + (1 - P_B)e^{-i\frac{C_{backoff}}{S_m}} \tag{16}$$

$$\text{then,} \quad R = \sum_{j=1}^{\infty} \prod_{i=1}^{j} H(i) \quad \text{and} \quad V = 1 + R \tag{17}$$

3.4 Special Cases

If the memory requests cannot queue (i.e., $M = 1$) then

$$U_{mem}^{M} = U_{mem} \quad \text{making} \quad R_{mem} = \frac{U_{mem} - U_{mem}(k)}{1 - U_{mem}}$$

$$\text{Since requests cannot queue,} \quad S_m(1 - U_{mem}) \sum_{i=1}^{M-1} iU_{mem}^{i} = 0$$

If all requests can queue (i.e., $M \geq NK$) then

$$U_{mem}^{M} = 0 \quad \text{making} \quad R_{mem} = 0$$

$$\text{thus} \quad D_1 = D_2 = D_3 = S_m \frac{U_{mem}}{1 - U_{mem}}$$

3.5 Evaluation of the model

Since the utilizations are known only once the model has been evaluated, we solve the model by the iterative scheme shown below.

1. Set all utilizations to zero.

2. Estimate the number of retries from eqs. (16) and (17).

3. Evaluate the various delays D_i, $i = 1, 2, 3$, D_{mem}, D_{bus1}, D_{ring}, and D_{bus2} using eqs. (9), (10), (11), (12), (13), (14), and (15) respectively.

4. Calculate the response times T_1, T_2, and T_3, based on the new values of the delays using eqs. (1), (2), and (3) respectively.

5. Evaluate the utilizations from eqs. (5)-(8).

We iterate steps 2 through 5 until successive estimates of memory utilization, U_{mem}, are sufficiently close, and then obtain performance measures from the final iteration.

$S_b = 1$	$S_r = 1$	$S_m = 8$
$C_1 = 10$	$C_2 = 12$	$C_3 = 19$
	$C_{backoff} = 12$	

Table 3: The default values for model parameters (in cycles)

To establish confidence in the robustness of our analysis, we modified our model to reflect our Hector prototype peculiarities such as limited buffering, priority of resource acquisition among the different classes of requests, and optimization of negatively acknowledged packets. We then validated our model with a detailed simulator proven to be quite accurate when compared against measurements obtained from experimentation on the prototype [HS92]. Our results [HS93] from simulation and analysis showed good agreement despite the assumptions made in deriving the model. We believe that the model captures the key features of the hierarchical structure, while making it possible to quickly study the effects of varying system parameters.

4 Results

Figure 3(a) shows how the system utilizations vary with the mean time between memory reference requests (a). The chosen range of a represents values likely to occur in practice. For moderate values of a (150-300 cycles between requests), the hot memory module is the system's bottleneck. As a gets smaller, the bus of the station accommodating the hot spot, bus_1, becomes the limiting factor of the system performance. As a decreases the probability for a request to be rejected becomes greater, causing the bus to be overloaded with retries. An important feature of the delay curves is that there is a range of values of a (< 150) for which a small change in the mean time between memory reference requests produces a disproportionately significant increase in delays. Consequently, bursty memory requests should be avoided as much as possible, as was previously observed by Anderson [And90].

The latency curves of Figure 3(b) show that at low values of a, off-station requests suffer the most from contention. (This may not be the case in systems such as Hector where remote requests are given a higher priority over local ones. This priority in service has not been incorporated into this model.) For large values of a, the difference in latencies is primarily due to the physical path a request has to travel. On the other hand, the contention factors dominate for small values of a.

It is important to fully understand the interactions among the various model parameters. For instance, one can notice that a 60 percent variation in the utilization ($30 < a < 300$) of bus_1

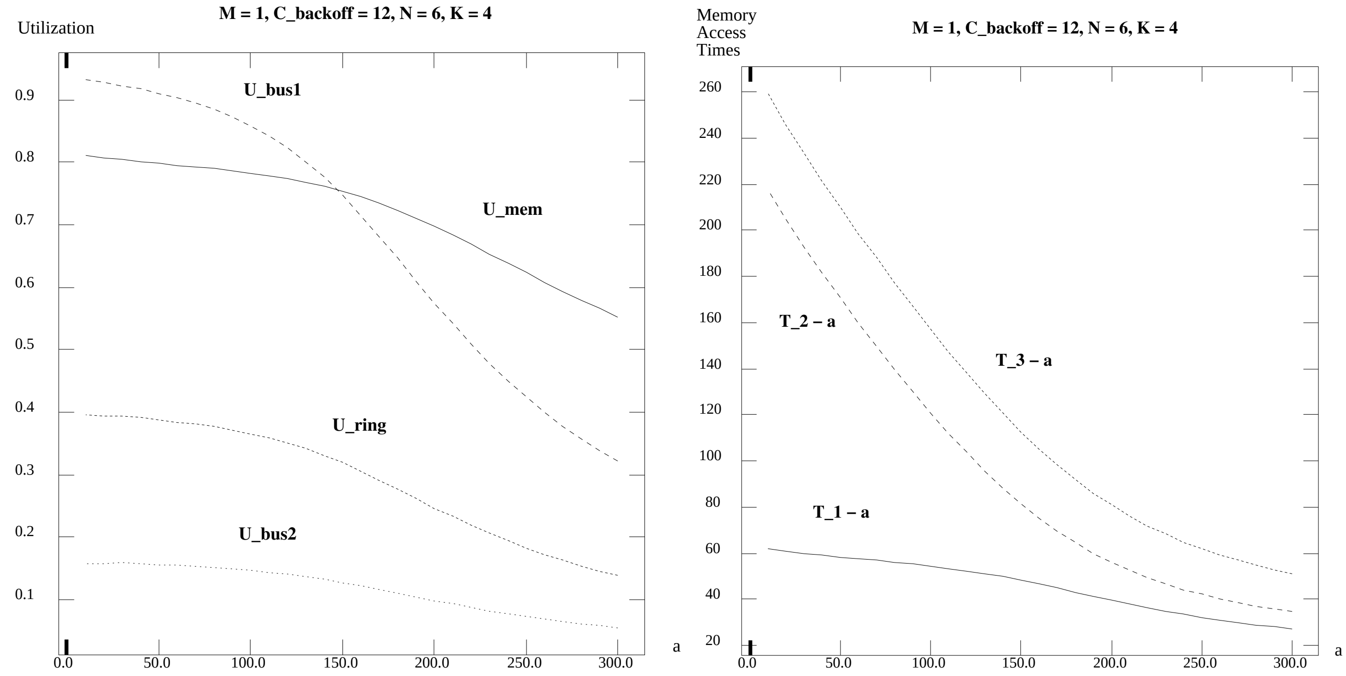


Figure 3: (a) Utilizations versus a. (b) Memory access times versus a.

translates to a 500 percent change in memory access time for remote processors.

4.1 Eliminating Hot Spots Through Replication

When the hot spot memory module is frequently accessed, the local bus becomes the performance bottleneck. To avoid this problem, we consider replicating the data structure on two stations (two way replication). Such replication has an initial cost for creating the replica and ongoing costs for maintaining the consistency between the two copies. Here, we assess only the potential gain in order to determine whether further exploration of the possibility is justified in a particular situation or not. 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 a k-way replication scheme (where $k \leq N$), decrees new rates of access that affect equations (5),(6), and (7). The new equations describing the model with a k-way replication (established by arguments analogous to those in Section 3) are as follows:

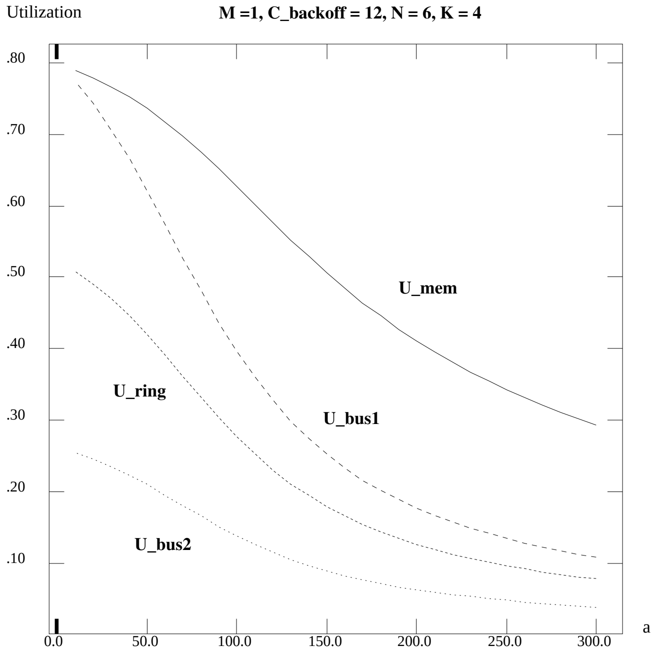
$$\begin{aligned} U_{mem} &= S_m[\frac{1}{T_1} + \frac{K-1}{T_2} + \frac{K(N-k)}{kT_3}] \\ U_{bus1} &= 2S_b[\frac{K-1}{T_2} + \frac{K(N-k)}{kT_3}]V_{mem} \\ U_{ring} &= S_r\frac{K(N-k)}{T_3}V_{mem} \end{aligned}$$

The results of this section were obtained using the equations given in the appendix which apply for two way replication. The curves in Figure 4(a) show the change in utilizations as the reference frequency increases for a replicated hot spot. Replicating the hot spot reduces the contention at the memory level and along the communication network. In contrast to the single copy case in Figure 3, bus_1 utilization remains below the memory utilization for the whole range of a values, suggesting that the number of retries indeed drops with replication. For small values of a, the utilizations of bus_2 and of the $ring$ are greater than those incurred with a single hot spot. This is due to an increase in the throughput of the requests as a result of replication. The benefits of reduced contention and latency gained from replication must be balanced against the cost of maintaining consistency of the replicas.

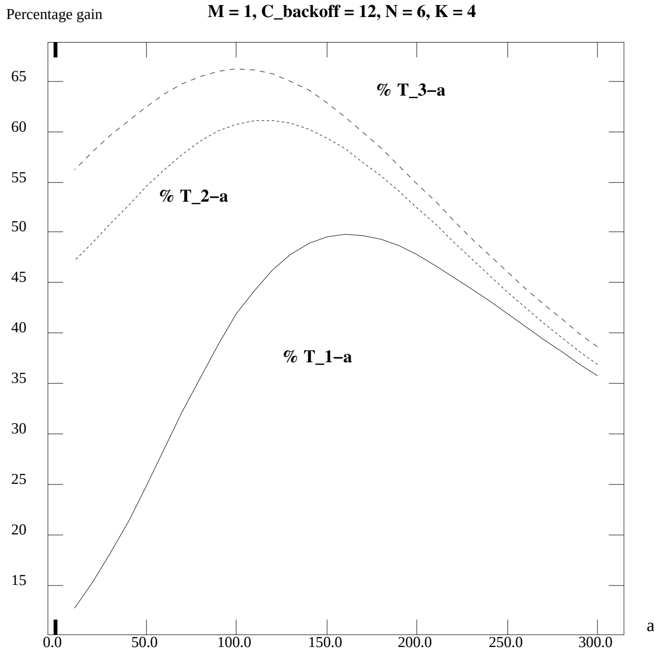
Replicating data allows a greater number of requests to be satisfied locally, thus removing some load from the ring and bus_1. It also indirectly decreases the number of retries, which has the effect of reducing the load on the network. However, this means that the requests can progress at a faster rate (because of the closed feedback loop in the model), resulting in a more highly utilized network, and subsequently leading to higher delays. In Figure 4(a) U_{ring} and U_{bus2} are higher (for $a \leq 100$) than in the non-replicated case. The shape of the curves in Figure 4(b) illustrates how the memory access times involve delays that depend on the dynamic state of the network, which in turn depends on the rate of requests. For low values of a, the progress of the requests is limited by the observed delays rather than by the number of cycles between memory requests. Hence, the contention becomes more pronounced, translating into a declining relative gain (with respect to the non replicated case). Figure 4(b) also indicates that off-station accesses benefit the most from replication, because this type of access is the most sensitive to contention and thus gains the most from a strategy that alleviates memory congestion. As the access frequency increases, the marginal gain improves to a maximum. It is interesting to note that the marginal gain for local, on-station, and off-station access manifest maxima for different values of a. This is due to the fact that memory accesses from different levels exhibit different sensitivity to contention.

4.2 Impact of Backoff Values

In this section, we evaluate the effect of backoff values on system performance using our analytical model, and discuss the trade-offs between reduced traffic bandwidth and wasted cpu cycles. Figure 3(a) suggests that the overall performance is impacted by the network traffic that includes regular and hot-spot traffic contention. Figure 5(a) compares the system performance for backoff values of 2, 4, 8, 12, and 30 cycles. The curves illustrating the utilization of bus_1 for different $C_{backoff}$ values show that large $C_{backoff}$ results in more efficient use of bus_1, since a larger backoff value results in spreading out arrivals, thus reducing the contention. The memory utilization curves associated with $C_{backoff} = 2$, and $C_{backoff} = 30$ suggest that a large $C_{backoff}$ gives better performance for low values of a, and a small backoff value works better for situations with lower contention. Therefore, by using a large backoff value, we reduce network traffic, but this saving comes at the expense of

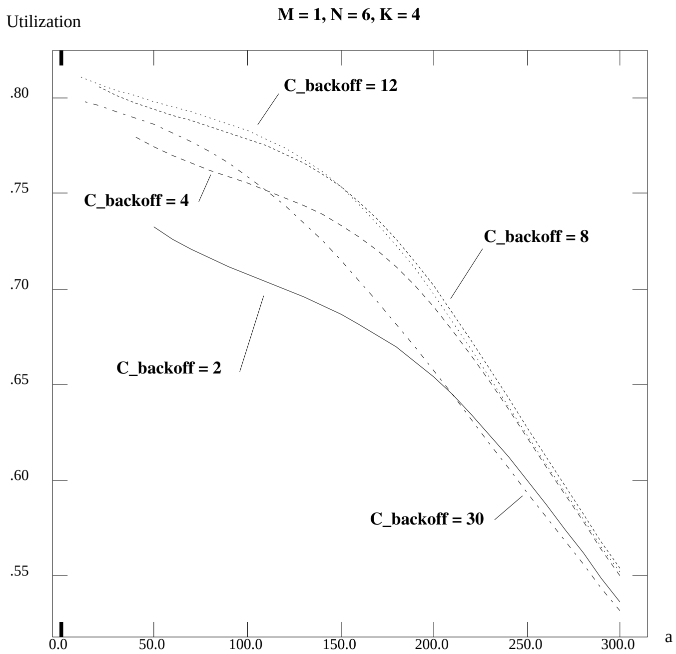


(a) Effect on utilizations.

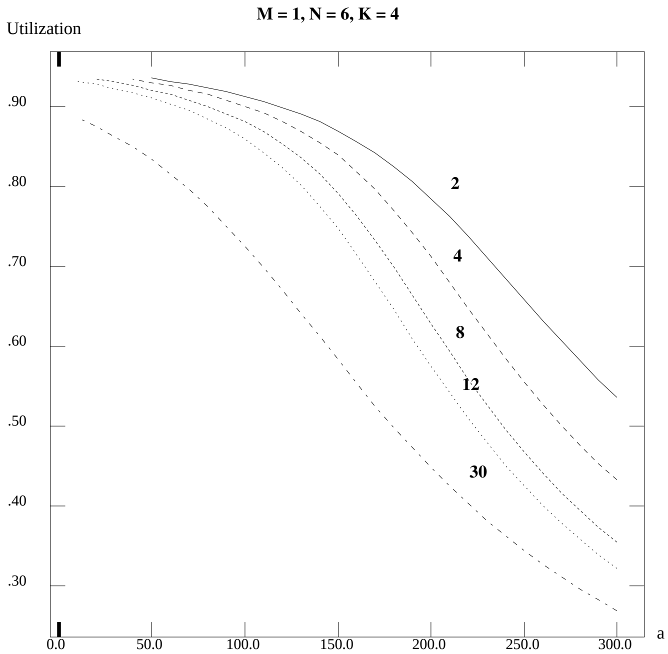


(b) Effect on latencies.

Figure 4: Effects of Replication.

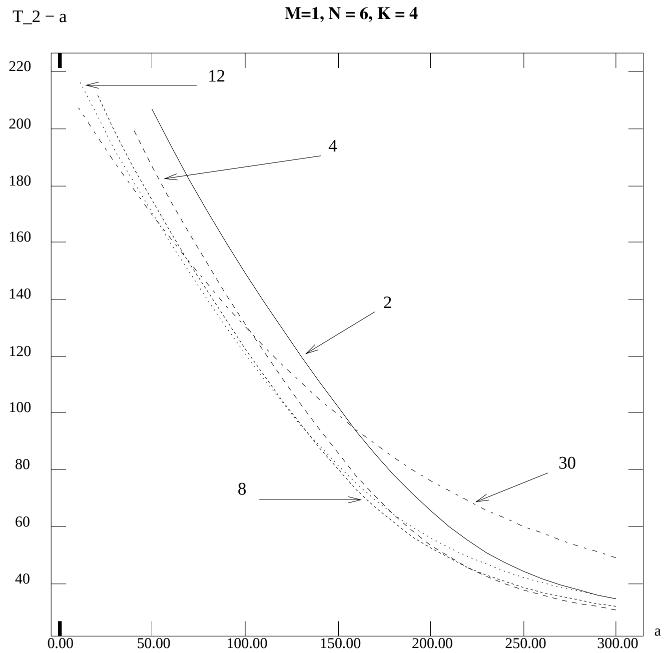


(a) Effect on memory utilization versus a.

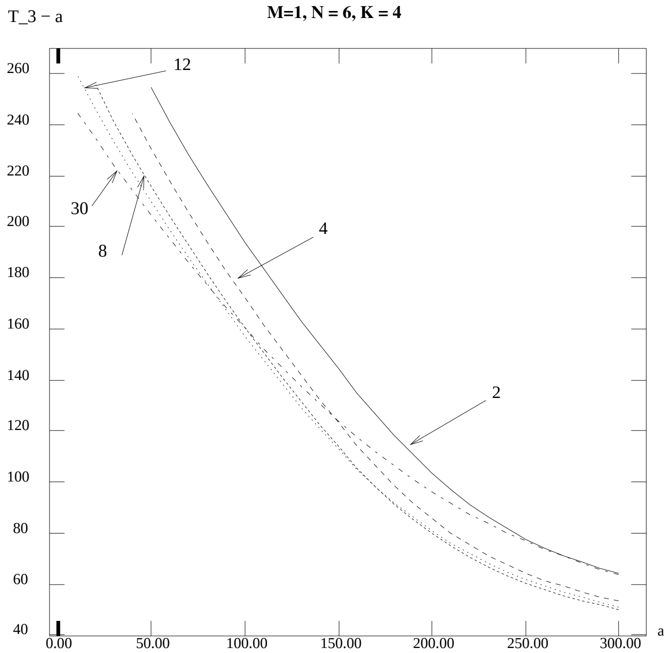


(b) Effect on bus_1 utilization versus a.

Figure 5: Effects of various backoff values, $C_{backoff}$.



(a) On-station access time versus a.



(b) Off-station access time versus a.

Figure 6: Effects of various backoff values, $C_{backoff}$.

increased memory access latency times. (Similar results were obtained by Agarwal and Cherian [AC89].) For low values of a, the curves converge, which is expected since most of the requests are satisfied on their first try. Note that the memory utilization curves associated with $C_{backoff} = 30$ and $C_{backoff} = 4$ intersect when $a = 100$ (i.e., resulting in the same system throughput), yet one backoff value ($C_{backoff} = 30$) achieves smarter use of the bandwidth which translates into a 25 percent of saving in bus utilization. The saving would be more significant with larger systems.

Figures 6(a), and 6(b) depict the effects of various backoff values on the on-station and off-station access times. Note that under heavy load the impact is qualitatively the same for both classes of requests. For light load, the access times for on-station and off-station accesses behave differently for corresponding backoff values.

4.3 Buffer Size Implications on Performance

Figure 7 shows the benefit of increasing the memory buffer size for various request frequencies. Extra buffers added to memory modules benefit the system the most during the critical phases of the lifetime of the applications (i.e., when the degree of contention is high). For instance, a buffer of size two improves the memory utilization (relative to utilization of a memory with a single buffer) only by 2 percent for $a = 300$, yet, the improvement is 8 percent when $a = 50$ cycles. Like other aspects of system design, memory buffer design is subject to the law of diminishing returns

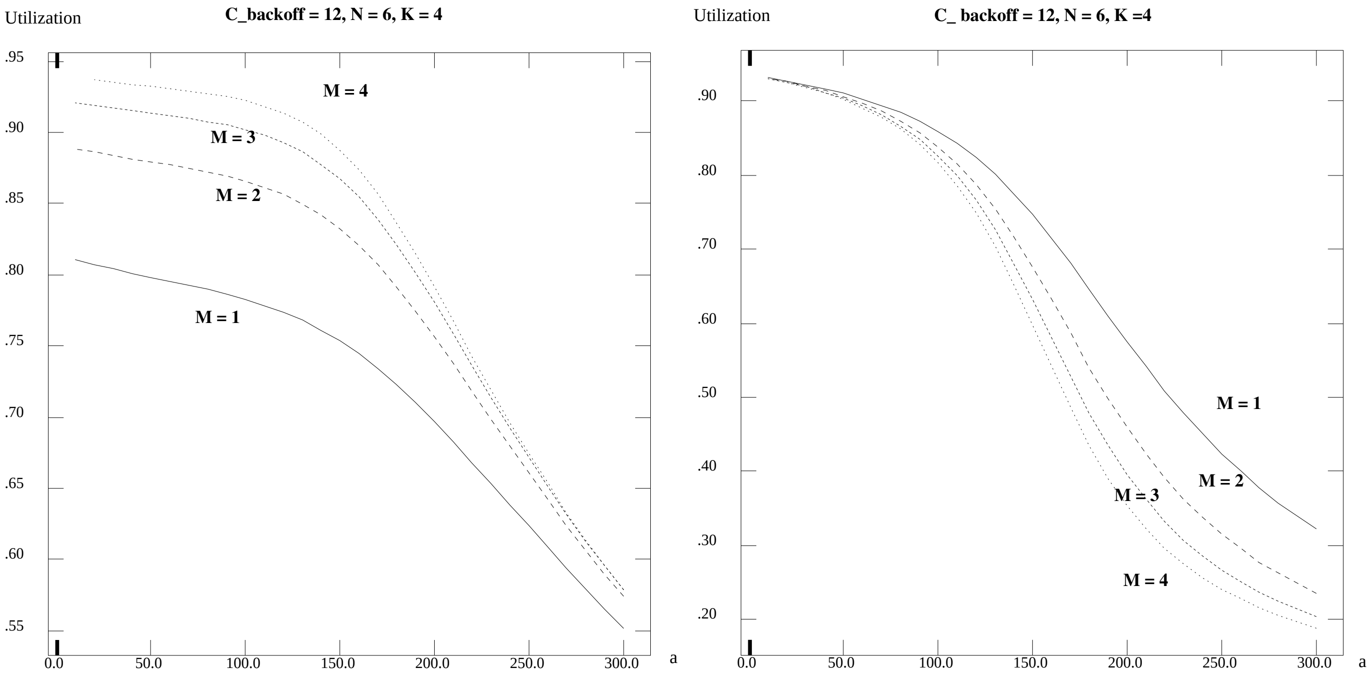


Figure 7: (a) Memory utilization versus a (b) Bus_1 utilizatio n versus a

as Figure 7 illustrates.

5 Conclusions

Memory allocation decisions are crucial to good performance in NUMA systems, and a model that provides insights into the design tradeoffs is valuable. In this paper, we have presented an analytical study of the performance of large scale NUMA multiprocessors in the presence of a single memory hot spot. We developed a simple 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 backoff algorithm on the network traffic. Our results reveal that memory latency is very sensitive to the non-uniformity of memory requests. We show that, if measures are not taken to alleviate the contention, parallel applications can suffer long memory access delays.

Our analysis also assessed the potential gain from replication, and quantified the expected improvement in memory utilization and network bandwidth by allowing memory request buffering. Finally, we investigated the effect of the backoff value (after a memory request rejection) on the network bandwidth, which is a valuable resource in large-scale machines. The results indicate that varying the backoff value can result in a tradeoff between throughput and network utilization. We are currently investigating the benefit of adaptive backoff values on the overall system performance.

Our experiments confirm the need for developing new techniques to manage the distributed physical memory found in NUMA multiprocessors. Traditional methods that ignore non-uniform memory latency characteristics are bound to impose significant communication overhead on scalable parallel systems. The model developed in this paper provides some insight into the NUMA problem, and can help in the development of better approaches to NUMA memory management.

Acknowledgements: Our thanks to Songnian Zhou for feedback on an earlier draft of the paper.

References

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

[AE91] R. L. Adema and C. S. Ellis. Duke university, computer science department. Technical report, Memory Allocation Constructs to Complement NUMA Memory Management, 1991.

[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.

[And90] T. E. Anderson. The performance of spin lock alternatives for shared-memory multiprocessors. *Transactions on Parallel and Distributed Systems*, 1(1), 1990.

[BBN90] BBN. Inside the butterfly tc2000. Technical report, BBN Advanced Computers, Cambridge, MA, February 1990.

[BS91] William J. Bolosky and Michael L. Scott. Evaluation of multiprocessor memory systems using off-line optimal behavior. Technical report, University of Rochester, Computer Science Department, 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 architecture. 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.

[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.

[HS92] M. Holliday and M. Stumm. Performance evaluation of hierarchical ring-based shared memory multiprocessors. Technical Report 1992–18, Duke University,Computer Science Department, 1992.

[HS93] K. Harzallah and K. C. Sevcik. Evaluating memory system performance of a large scale NUMA multiprocessor. Technical Report 283, University of Toronto, CSRI, available for anonymous ftp from ftp.csri.toronto.edu, March 1993.

[LHE92] R. P. LaRowe, M. A. Holliday, and C. S. 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.

[NSTN91] A. K. Nanda, H. Shing, T. Tzen, and L. M. Ni. Resource contention in shared-memory multiprocessors: A parameterized performance degradation model. *Journal of Parallel and Distributed Computing*, 12:313–328, 1991.

[SZ91] K. C. Sevcik and S. Zhou. Performance benefits and limitations of large numa multiprocessors. Technical Report 256, University of Toronto, November 1991. also to appear in the proceedings of Performance '93, Rome, September 1993.

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

[ZQ91] X. Zhang and X. Qin. Performance prediction and evaluation of parallel processing on a numa multiprocessor. *Transactions on Software Engineering*, 17(10):1059–1068, October 1991.

Appendix

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

$$
\begin{aligned}
U_{mem} &= S_m[\frac{1}{T_1} + \frac{K-1}{T_2} + \frac{K(N-2)}{2T_3}] \\
U_{bus1} &= 2S_b[\frac{K-1}{T_2} + \frac{K(N-2)}{2T_3}]V_{mem} \\
U_{ring} &= S_r\frac{K(N-2)}{T_3}V_{mem} \\
U_{bus2} &= 2S_b\frac{K}{T_3}V_{mem} \\
D_1 &= (C_{backoff} + C_1 - S_m)R_{mem} + D_{mem} \\
D_2 &= (C_{backoff} + C_2 - S_m + 2D_{bus_1})R_{mem} + D_{mem} \\
D_3 &= (C_{backoff} + C_3 - S_m + 2D_{bus_1} + 2D_{ring} + 2D_{bus_2})R_{mem} + D_{mem} \\
D_{mem} &= S_m(1 - U_{mem})\sum_{i=1}^{M-1} iU_{mem}^i \\
D_{bus1} &= S_b\frac{U_{bus_1}}{1 - U_{bus_1}}
\end{aligned}
$$

$$
\begin{aligned}
D_{ring} &= S_r \frac{U_{ring}}{1 - U_{ring}} \\
D_{bus2} &= S_b \frac{U_{bus_2}}{1 - U_{bus_2}} \\
R_{mem} &= \sum_{j=1}^{\infty} \prod_{i=1}^{j} [P_B + (1 - P_B) e^{-i \frac{C_{backoff}}{S_m}}] \quad \text{where} \quad P_B = \frac{(KN - 2)U_m^M}{KN - 2U_m^M} \\
V_{mem} &= 1 + R_{mem}
\end{aligned}
$$