Comparative Performance Evaluation of Hot Spot Contention Between MIN-Based and Ring-Based Shared-Memory Architectures

Xiaodong Zhang, *Senior Member, IEEE*, Yong Yan, and Robert Castañeda

Abstract—**Hot spot contention on a network-based shared-memory architecture occurs when a large number of processors try to access a globally shared variable across the network. While Multistage Interconnection Network (MIN) and Hierarchical Ring (HR) structures are two important bases on which to build large scale shared-memory multiprocessors, the different interconnection networks and cache/memory systems of the two architectures respond very differently to network bottleneck situations. In this paper, we present a comparative performance evaluation of hot spot effects on the MIN-based and HR-based shared-memory architectures. Both nonblocking MIN-based and HR-based architectures are classified, and analytical models are described for understanding network differences and for evaluating hot spot performance on both architectures. The analytical comparisons indicate that HR-based architectures have the potential to handle various contentions caused by hot spots more efficiently than MIN-based architectures. Intensive performance measurements on hot spots have been conducted on the BBN TC2000 (MIN-based) and the KSR1 (HR-based) machines. Performance experiments were also conducted on the practical experience of hot spots with respect to synchronization lock algorithms. The experimental results are consistent with the analytical models, and present practical observations and an evaluation of hot spots on the two types of architectures.**

Index Terms—**Hierarchical Rings (HR), hot spot, Multistage Interconnection Network (MIN), performance modeling and measurements, slotted rings, shared-memory, the BBN TC2000, the KSR1.**

I. Introduction

Hot spot contention on a network-based shared-memory architecture occurs when a large number of processors try to access a globally shared variable across the network. This topic has been studied extensively on MIN-based architectures. Pfister and Norton [11] generalize the problem as a type of network traffic non-uniformity. Their analytical models and simulation results for blocking MIN-based architectures show that the hot spot effects may severely degrade *all* network traffic, not just the traffic to shared variables. The effect is defined as *tree saturation*, where traffic to the hot memories backs up at the switch and interferes with other traffic, including that to non-hot memories. In addition, they indicate that the combining of messages within a MIN-based architecture is an effective technique for dealing with the hot spot problem. Thomas [13] presents a set of experiments designed to measure the behavior of the Butterfly I system in the presence of memory hot spots. The experimental results reported in the paper show that the tree saturation effects do not generalize to the Butterfly I machine because nonblocking networks are used in the system.

The work we describe here differs from the studies cited above in several important respects. First, we describe analytical models for evaluating hot spot effects on nonblocking MIN-based architectures. We show that significant delay may also happen in the presence of memory hot spots, when network contention inherent in a MIN-based architecture is potentially high. The analytical results are confirmed by our experiments on the TC2000. We also present analytical models for evaluating hot spots on HR-based architectures. Comprehensive performance evaluation results provide deeper understanding and comparisons of hot spots on both architectures. Second, we conduct a series of experiments for evaluating hot spot effects on the BBN TC2000, a MIN-based architecture and the KSR1, an HR-based architecture. The experiments on the TC2000, which has heavier network contention than the Butterfly I which was used by Thomas [13], present much stronger hot spot effects. Finally, comparative and quantitative performance analysis and evaluation of hot spot effects on both MIN-based and HR-based architectures are presented based on modeling and experimental results.

The organization of this paper is as follows. Section II presents performance models for evaluating hot spot effects in terms of remote access delay and network contention on nonblocking MIN-based and HR-based architectures. Based on the analytical models, comparative hot spot performance evaluation of the two architectures is presented. In Section III, the network and system structures of the TC2000 and the KSR1 are briefly overviewed, experimental results are reported, and comparative behavior and analysis are addressed. Performance experiments were also taken on the practical experience of hot spots with respect to synchronization lock algorithms. Summaries and conclusions are given in Section IV.

Manuscript received July 9, 1993; revised Dec. 3, 1994.

Xiaodong Zhang, Yong Yan, and Robert Castañeda are with the High-Performance Computing and Software Laboratory at the University of Texas at San Antonio, San Antonio, TX 78249; e-mail: zhang@ringer.cs.utsa.edu, yyan@dragon.cs.utsa.edu, and rcastane@dragon.cs.utsa.edu, respectively.

IEEECS Log Number D95015.

II. Analytical Performance Evaluation

A. Analytical Models for Nonblocking MIN Architectures

In a MIN architecture, the network contention is defined as a conflict where two messages need to access the same portion of a path at the same time. The network can be designed either in blocking form or nonblocking form. In the blocking net-

work, a protocol organizes a message queue while all the conflicting traffic comes to a standstill (each of the other conflicting messages sits and holds its path). When the path is cleared, the next selected message proceeds. The major problem of the blocking network is the so called cascade effect—where each new message tends to run into other blocked messages, and gets blocked itself. This ties up more resources in the switch, and increases the chance that subsequent messages will also block. This is the main reason why the tree saturation described in [11] may easily appear in a blocking MIN architecture. The nonblocking network may reduce the traffic conflicts: when conflicts happen, the switch has all but the "first" message retreat back to its source and free up their path. It then selects an alternative route, and after a random delay, tries again. Zhang and Qin [16] present a remote access delay model for a nonblocking MIN architecture where the behavior of a remote memory access is described by a state transition diagram called the *drop approach* [6]. Here a processor makes a remote memory access by formulating requests for access to the set of switches along that path. If it cannot obtain a switch, it abandons its request at that point and will try again at some later time. In Fig. 1, state 0 represents some processor in quiescent state, while state $n + 1$ represents an ongoing successful access. State b represents the processor when it has dropped its request because of the switch contention.

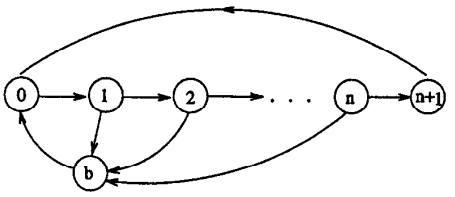


Fig. 1. State transition diagram for remote-memory access through an n-stage interconnection network.

Based on the general access delay model described in [16], we present a remote access delay model in the presence of memory hot spots. Here we briefly give some of the major results of the model. For detailed analyses and proofs of the model, the interested readers may refer to [14].

Assume that the size of each switch in the MIN network is $k \times k$. The probability for a hot spot access request to acquire the switch on state $i + 1$, denoted as q_i^{hot} is

$$q_i^{hot} = \left[1 - \frac{\pi_i\left(\lambda_h k^i + 1\right)}{k\lambda_h\left(k^{i-1} - 1\right) + k}\right]^{k-1}, \qquad (2.1)$$

where π_i, $(i = 0, ..., n + 1)$ is the steady-state probabilities for a request at state i, T is the switch delay, and λ_h is the average hot spot request rate.

Including a hot spot to the remote memory delay model described in [16], we present an average memory latency model in the presence of a hot spot on a nonblocking MIN-based network:

$$T_{hot} = nT + \frac{1}{\phi_s} + \frac{\left(1 - P_s^h\right)}{P_s^h}\left(T_f^h + \frac{1}{\phi_b}\right) \qquad (2.2)$$

where n is the number of MIN stages, T is the switch delay time, $\frac{1}{\phi_s}$ is an average memory access time, P_s^h is the probability of a successful memory access through all stages in the presence of a hot spot, T_f^h is an average delay of unsuccessful memory access in the presence of a hot spot, and $\frac{1}{\phi_b}$ is an average delay time spent in state b in Fig. 1. For detailed mathematical derivation of (2.2), the interested reader may refer to [14] and [16].

Fig. 2 presents a comparative view of a group of memory delay curves. The access delays with different hot spot fractions (vertical axis) grow as the access request rate λ grows (horizontal axis). The network has six stages connected by a group of 4×4 switches. The memory delay in Fig. 2 is represented by factors of a normal memory access (delay factor = 1). A normal access has the request rate of 0.175 without the presence of hot spots. Examining Fig. 2, one can notice that memory latency is increased significantly but no tree saturation is observed. For example, the memory latency would be eight times higher in the presence of 32% hot spot fraction. In contrast, with a hot spot in a blocking MIN-based network, the latency climbs to an asymptote at the point the traffic and hot spot percentage combine to saturate the entire network. The model and the figure on page 945 in [11] address the reasons why cool memory requests are dramatically delayed, and describe the fact of tree saturation.

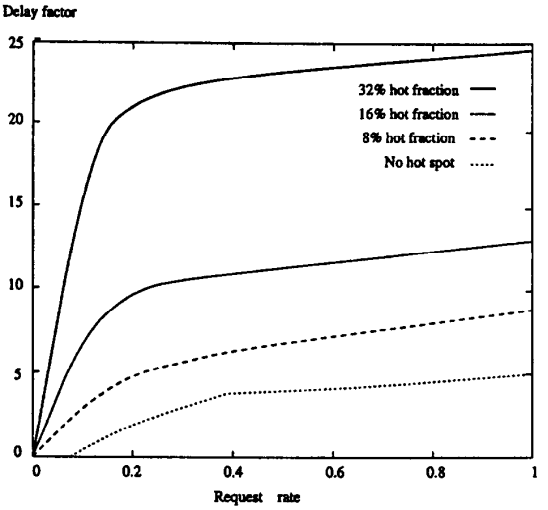


Fig. 2. Average memory latency in the presence of a hot spot on a 6 stage and $4 \times$ switch nonblocking interconnection network.

Another reason why nonblocking MIN architecture may not suffer tree saturation is that a cool memory access request in the presence of the hot spot may eventually escape the contention switches at some stage. Let's define this as stage i. Then the request will reach the target cool memory through noncontention switches. The probability for a successful access to a cool memory module at state j is

$$q_j^{cool} = \left(1 - \frac{\pi_j}{k}\right)^{k-1}, \; i < j \le n. \qquad (2.3)$$

The probability model (2.3) indicates that

$$q_j^{cool} = q_j^{hot}, 1 \le j < i,$$

and

$$q_j^{cool} > q_j^{hot}, i \le j \le n.$$

Comparing (2.1) and (2.3), we have

$$q_i^{hot} < q_i^{cool} Q \tag{2.4}$$

where

$$Q = \left(1 - \pi_i \frac{\lambda_h k^{i-1}}{\lambda_h\left(k^{i-1}-1\right)+1}\right)^{k-1}.$$

This means that the successful access probability of a hot memory request on state i is Q times less than the probability of a cool memory access at the same state. Therefore, an access request to a cool memory in the presence of memory hot spots may be significantly reduced in a nonblocking MIN architecture. The experimental results on the Butterfly I reported in [13] confirms this. However, as network contention inherent in a MIN architecture is increased, an access request to a cool memory in the presence of memory hot spots may be significantly delayed. This is because the state number i where a cool memory request leaves the hot spot path may be increased when the network contention is increased. Zhang and Qin [16] have studied the potential network contention in computation predicted by the existing MIN speed (the network bandwidth) and the processor speed (CPU clock rate) in the system. According to their analytical model and experiments, if the MIN speed remains the same in two systems, and the speed of processors is doubled in one system, the potential network contention in a computation in that system is at least doubled. The major difference in architecture between the Butterfly I and the TC2000 we used is the type of processors. The Butterfly I is the first generation of BBN MIN-based multiprocessors. Each processor node contains an 8 MHz MC68000. The BBN GP1000 uses slightly faster Butterfly network but replaces the MC68000 processors by much faster 20-MHz 88100 processors. Based on the performance evaluation in [16], the relative amount of overhead caused by network contention for the same computation on the TC2000 is at least two times higher than the overhead on the Butterfly I. In Section III, we will report the experimental results of hot spots on the TC2000, which give different performance results than the ones presented by Thomas [13] on the Butterfly I.

B. Analytical Models for HRs

For generality and simplicity of modeling, we define the target HR-based architecture model as a two level hierarchical ring with the following structures, functions and parameters:

1) The basic structure of the HR-based architecture is described in Fig. 3 where each of the m local rings (LR) are connected to a global ring (GR) through a link with a pair of ports on its two ends. The global ring has m equally sized slots connecting to m local rings. Each local ring has n equally sized slots, each of which connects to a processor node. A local memory is associated with each processor node and can be accessed globally by other processor nodes through the ring network.

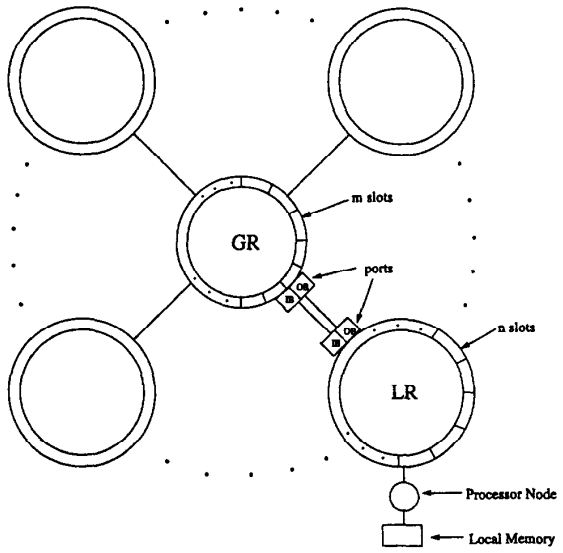


Fig. 3. The hierarchical ring architecture in the analytical model.

2) There are two buffers in each pair of ports of the link to connect the global ring and a local ring—an input buffer (IB), and an output buffer (OB). The IB is used to buffer coming messages, and the OB is used to buffer outgoing messages.
3) Both the global ring and local rings are rotated constantly. A processor node in a local ring ready to transmit a message waits until an empty slot is available. The rotation period is defined as t_r which also reflects the size of the slot.
4) A message will be unloaded to the destination processor slot and an acknowledging message will be loaded to the slot from the processor to be transmitted to the source processor. These two operations are conducted within the rotation period.
5) There are two different schemes for an access/storage of variables in an HR-based architecture. A *fixed type* variable defines a variable in a fixed physical memory location which has a permanent ownership during a program execution unless it is invalidated due to writing. The variable can be accessed through remote accesses by other processors. The fixed type operations are performed on a cache coherence nonuniform memory access architecture (CC-NUMA) [12]. Examples of CC-NUMA machines are Stanford DASH multiprocessor [9] and MIT Alewife machine [1]. In such a machine a home node for the corresponding physical address is required for operations of data access and cache miss. A *movable type* variable may be moved around among memory modules in the whole HR system, and its ownership is dynamically assigned to the processor node where it is

currently located. In addition, copies of the variables may be duplicated among the memory modules. The movable type operations are performed on a cache only memory architecture (COMA). Examples of COMA machines are the Swedish Institute of Computer Science's Data Diffusion Machine (DDM) [7] and the KSR1 machine. In this type of machine, the location of a cache block is totally decoupled from its physical address. Data are dynamically migrated and replicated in the entire cache system.

6) Cache coherence is maintained by a directory-based, write-invalidate cache coherence protocol. An analytical model and experimental case studies for data migration including cache coherence effects on CC-NUMA and CC-COMA ring architectures will be published in [17].
7) The frequency of memory accesses is defined as request rate λ, (the number of the requests per time unit). The request pattern sent from a processor is assumed in a Poisson function.

Furthermore, we assume:

1) The request miss rate of each local cache follows a Poisson process.
2) One message packet can be completely carried by one slot which only conveys this message packet, so the successive slots behave independently.
3) When a station receives a message packet from a slot, it will produce a reply into the same slot without any delay.

B.1. Remote Access Delay in the Presence of Fixed Hot Spots

Since the targeted HR system, such as the KSR-1, only allows one outstanding memory access request per processor, it is a closed system. Using a closed model by queueing theory to analyze the network contention would be a reasonable choice. This is because the close model is relatively precise. However, the closed model for a complex system needs to solve a large nonlinear system of equations. The solutions of the system may not be easy to get because the initial points of the system may not be close enough to the solutions. The results of the model solved by iterative methods are often approximate. In contrast, an open model is usually simple, which allows us to take more parameters and more performance issues into consideration. However, it may not be as precise as a closed model. In a closed system, if we can identify and distinguish the effects among multiple servers, it is still possible to use an open model to reasonably capture the performance characteristics of the system. Examples of using an open model to study a closed ring system are reported in [5] for network analysis and in [4] for cache coherence protocols. The modeling results in [4] are well supported by the simulation.

Besides its practical applications in performance analysis, there are three reasons why using an open model to study the ring hot spot effects is acceptable and sufficient. First, the open model could be reasonably precise for this study because we have special information for the architecture and the system, such as memory access distribution patterns. Second, the contention points in the ring system exist in the interface ports between the global ring and the local ring, and between a processor and the local ring. The communication distribution in the steady state can be completely determined by the given conditions. Thus, for each contention point, its contention degree can be reasonably modeled by the gated /M/G/1 queue of an open model. Finally, the analytical results of the models have been consistent with the experiment measurements on the KSR-1.

We use an open model to study network contention in the HR system with the following considerations of memory access distributions:

1) Only the performance in steady state is considered. In the steady state, assume the following two performance parameters are given.
   a) λ_h: a fraction of λ contributed to a hot-spot memory module.
   b) a fraction of $\lambda(1 - \lambda_h)$ directed to memory modules in its local ring. Therefore the nonlocal request rate is $(1 - \lambda_l)(1 - \lambda_h)\lambda$. In addition, the local request rate and nonlocal request rate is assumed to be uniformly distributed.
2) The numbers of request packets and acknowledgement packets traveling in the ring at any time are the same.

In modeling hot spot effects, the hierarchical rings are divided into three parts in terms of network activities: the local ring in the presence of the hot spot called the hot local ring, the global ring, and the rest of the local rings without the presence of hot spots, called cool local rings. A comprehensive access delay model for the entire HR-system in the presence of memory hot spots is presented based on modeling the non-hot spot local rings, the hot spot local ring and the global ring.

A. Modeling the Non-Hot Spot Local Rings

Due to the similarity among the access patterns of each non-hot local ring, the analysis on the non-hot local rings can be focused on a single non-hot local ring without loosing generality. Based on the ring architecture described in the previous section, accesses to the interface ports between the global ring and a local ring can be modeled by a gated M/Q/1 queue with a mean packet arrival rate λ_{np}, which may be formed by a steady state formula,

$$\lambda_{np} = n\lambda\lambda_h + 2n(1-\lambda_h)(1-\lambda_l)\lambda \tag{2.5}$$

This arrival rate is the sum of the probe packet arrival rate and the data packet arrival rate.

A general network utilization is defined by

$$U = \lim_{t\to\infty} \frac{C}{t} \tag{2.6}$$

where C is the total number of bytes transmitted to the network from all processor nodes during the time period of t. Based on (2.5) and (2.6), a local ring utilization is formulated as

$$U_l = t_r(\lambda_{np} + n\lambda) = n\lambda t_r(1 + \lambda_h + 2(1 - \lambda_h)(1 - \lambda_l)). \tag{2.7}$$

Because the successive slots have been assumed to behave independently, the probability of a slot on a cool local ring to be full is U_l. The time needed for a packet in each station to find an empty slot can be approximated to have a geometric

distribution. The analyses in [5] show that the error caused by the independence assumption is trivial. The probability of this event is $U_l^i(1-U_l)$. Thus, the average time of finding an empty slot is

$$\begin{aligned}\bar{d}_n &= \sum_{i=0}^{\infty} i t_r (1-U_l) U_l^i \\ &= \frac{t_r U_l}{1-U_l} \\ &= \frac{n\lambda t_r^2\left(1+\lambda_h+2(1-\lambda_h)(1-\lambda_l)\right)}{1-n\lambda t_r\left(1+\lambda_h+2(1-\lambda_h)(1-\lambda_l)\right)}.\end{aligned} \tag{2.8}$$

The average delay for an access from an interface port to a processor in this ring, denoted as $\bar{d}_{pnl}$, consists of a delay $\bar{w}$ in the input buffer and the average destination searching time $nt_r/2$:

$$\bar{d}_{pnl} = \bar{w} + \frac{nt_r}{2}.$$

The average request queue length in the input buffer may be calculated by Little's law,

$$Q = \lambda_{np}\bar{w} \tag{2.9}$$

where $\bar{w}$, an average waiting time in the queue may be calculated by

$$\bar{w} = \left(\bar{d}_n + t_r\right) + Q\left(\bar{d}_n + t_r\right). \tag{2.10}$$

Combining (2.9) and (2.10), $\bar{w}$ becomes

$$\bar{w} = \frac{\bar{d}_n + t_r}{1-\lambda_{np}\left(\bar{d}_n + t_r\right)}.$$

Finally, the average delay for a nonlocal ring access gives

$$\bar{d}_{pnl} = \frac{\bar{d}_n + t_r}{1-\lambda_{np}\left(\bar{d}_n + t_r\right)} + \frac{nt_r}{2}. \tag{2.11}$$

B. Modeling the Hot Spot Local Ring

The hot spot local ring can be modeled similarly as a cool local ring in A. Let λ_{hp} be the packet arrival rate on the hot interface port from the global to the hot local ring. It has

$$\lambda_{hp} = (m-1)n\lambda\lambda_h + 2n(1-\lambda_h)(1-\lambda_l)\lambda. \tag{2.12}$$

Based on (2.12) and (2.6), the utilization of the hot local ring is

$$U_h = (\lambda_{hp} + n\lambda)t_r = n\lambda t_r(1 + (m-1)\lambda_h + 2(1-\lambda_h)(1-\lambda_l)).$$

So the mean time to find an empty slot on the hot ring is

$$\bar{d}_h = \frac{U_h t_r}{1-U_h} = \frac{t_r^2\left(\lambda_{hp}+n\lambda\right)}{1-t_r\left(\lambda_{hp}+n\lambda\right)}. \tag{2.13}$$

Let $\bar{d}_{phl}$ be average delay for an access from the hot port to a processor in the hot ring. Using (2.9), (2.10) and (2.11), $\bar{d}_{phl}$ can be expressed as

$$\bar{d}_{phl} = \frac{\bar{d}_h + t_r}{1-\lambda_{hp}\left(\bar{d}_h + t_r\right)} + \frac{nt_r}{2}. \tag{2.14}$$

Moreover the average length of input queue in the hot interface port is

$$\begin{aligned}\bar{L}_h &= \frac{\lambda_{hp}\left(t_r + \bar{d}_h\right)}{1-\lambda_{hp}\left(t_r + \bar{d}_h\right)} \\ &= \frac{\left((m-1)n\lambda\lambda_h + 2n(1-\lambda_h)(1-\lambda_l)\lambda\right)t_r}{1-n\lambda t_r\left(1+2(m-1)\lambda_h+4(1-\lambda_h)(1-\lambda_l)\right)}.\end{aligned} \tag{2.15}$$

C. Modeling the Global Ring

Access to a slot in the global ring from a local ring via the interface port can also be modeled as a gated M/Q/1 queue. Let λ_{gh} and λ_{gn} be the packet arrival rates of the hot port and a non-hot port, then we have

$$\lambda_{gh} = (m-1)n\lambda\lambda_h + 2n(1-\lambda_h)(1-\lambda_l)\lambda, \tag{2.16}$$

$$\lambda_{gn} = n\lambda\lambda_h + 2n(1-\lambda_h)(1-\lambda_l)\lambda. \tag{2.17}$$

Based on (2.16), (2.17), and (2.6), the utilization of the global ring can be expressed as

$$\begin{aligned}U_g &= (\lambda_{gh} + (m-1)\lambda_{gn})t_r \\ &= (2(m-1)n\lambda\lambda_h + 2mn(1-\lambda_h)(1-\lambda_l)\lambda)t_r.\end{aligned} \tag{2.18}$$

The average delay for a processor node to find an empty slot in the global ring, denoted as $\bar{d}_g$, can also be determined by (2.18) and (2.8),

$$\bar{d}_g = \frac{2n\lambda t_r^2\left((m-1)\lambda_h + m(1-\lambda_h)(1-\lambda_l)\right)}{1-2n\lambda t_r\left((m-1)\lambda_h + m(1-\lambda_h)(1-\lambda_l)\right)}. \tag{2.19}$$

Moreover, by (2.9), (2.10), and (2.17), the queue length, denoted as L_{gh}, on the hot port can be expressed as

$$L_{gh} = \frac{\lambda_{gh}\left(t_r + \bar{d}_g\right)}{1-\lambda_{gh}\left(t_r + \bar{d}_g\right)}. \tag{2.20}$$

The average queueing delay on the hot port, denoted as $\bar{w}_h$, is

$$\begin{aligned}\bar{w}_h &= \frac{\bar{d}_g + t_r}{1-\lambda_{gh}\left(\bar{d}_g + t_r\right)} \\ &= \frac{t_r}{1-n\lambda t_r\left(3(m-1)\lambda_h + 2(m+1)(1-\lambda_h)(1-\lambda_l)\right)}.\end{aligned} \tag{2.21}$$

The average queueing delay on the non-hot port, denoted as $\bar{w}_n$, is

$$\begin{aligned}\bar{w}_n &= \frac{\bar{d}_g + t_r}{1-\lambda_{gn}\left(\bar{d}_g + t_r\right)} \\ &= \frac{t_r}{1-n\lambda t_r\left((2m-1)\lambda_h + 2(m+1)(1-\lambda_h)(1-\lambda_l)\right)}.\end{aligned} \tag{2.22}$$

D. Access Delay Analysis in the Hierarchical Rings

Here we summerize various average access delays in the entire HR network, where each access takes a round trip from the source back to source again.

1) Average local request delay in a non-hot spot ring, denoted as d_{lcool}, consists of the time of finding an empty

slot on a cool local ring and the request travel time on the local ring. By (2.8) we have

$$d_{lcool} = \bar{d}_n + nt_r. \tag{2.23}$$

2) Average local request delay in the hot local ring, denoted as d_{lhot}, consists of the time of finding an empty slot on hot local ring and the request travel time. By (2.13) we have

$$d_{lhot} = \bar{d}_h + nt_r. \tag{2.24}$$

3) Average request delay from a cool ring to another cool ring, denoted as d_{cc}, consists of four travel timing parts: time from the source cool ring to the global ring, the time from the global ring to the destination local ring, the time for searching the destination node in the destination ring and the time for the data packet to come back to the source node. By (2.8), (2.11), and (2.22), we have

$$d_{cc} = \bar{d}_n + 2\bar{w}_n + 2\bar{d}_{pnl} + t_r(m+n). \tag{2.25}$$

4) Average request delay from a cool ring to the hot ring, denoted as d_{ch}, consists of four travel timing parts: the time from the source cool ring to the global ring, the time from the global ring to the hot ring, the time for searching the destination node in the hot ring and the time for the data packet to come back to the source node. By (2.8), (2.11), (2.14), (2.21), and (2.22), we have

$$d_{ch} = \bar{d}_n + \bar{w}_n + \bar{d}_{phl} + \bar{w}_h + \bar{d}_{pnl} + t_r(m+n). \tag{2.26}$$

5) Average request delay from the hot ring to a cool ring, denoted as d_{hc}, consists of four travel timing parts: the time from the hot ring to the global ring, the time from the global ring to the destination cool ring, the time for searching the destination node in the destination cool ring, and the time for the data packet to come back to the source node. By (2.13), (2.11), (2.14), (2.21), and (2.22), we have

$$d_{hc} = \bar{d}_h + \bar{w}_n + \bar{d}_{phl} + \bar{w}_h + \bar{d}_{pnl} + t_r(m+n). \tag{2.27}$$

6) Average delay to access a cool ring in the presence of hot spots, denoted as D_{cool}, comes from (2.23), (2.25), and (2.26):

$$D_{cool} = \lambda_h d_{ch} + (1-\lambda_h)(\lambda_l d_{lcool} + (1-\lambda_l)d_{cc}). \tag{2.28}$$

7) Average delay to access the hot ring, denoted as D_{hot}, comes from (2.24) and (2.27):

$$D_{hot} = (\lambda_h + (1-\lambda_h)\lambda_l)d_{lhot} + (1-\lambda_h)(1-\lambda_l)d_{hc}. \tag{2.29}$$

Combining the (2.28) and (2.29), the average delay D of a request in an HR in the presence of a hot spot is

$$D = \frac{(m-1)D_{cool}}{m} + \frac{D_{hot}}{m}. \tag{2.30}$$

E. Hot Spot Effects on the Hierarchical Ring Network

The effects of the hot spot in the hierarchical ring network may be determined by the average access request rate λ. It gives a quantitative view of network contention caused by the hot spot. We assume that each access request is done when the corresponding message from the target processor node comes back to the source node. The average access request rate λ is bounded by the average time of completing a request, T. In a real system, the speed of network transactions will be stable when the access time is stable. Let λ_s and T_s be a request rate and an average access request time respectively in a steady state, we have

$$0 \le \lambda_s \le \frac{1}{T_s} \le 1. \tag{2.31}$$

Network transactions can be determined by utilizations of the three classes of rings. The network untilization in a cool local ring is

$$U_{cool} = n\lambda t_r[(1 + \lambda_h + 2(1-\lambda_h)(1-\lambda_l)], \tag{2.32}$$

in the hot local ring is

$$U_{hot} = n\lambda t_r[(1 + (m-1)\lambda_h + 2(1-\lambda_h)(1-\lambda_l)], \tag{2.33}$$

and in the global ring is

$$U_g = 2n\lambda t_r[(m-1)\lambda_h + m(1-\lambda_h)(1-\lambda_l)], \tag{2.34}$$

An upper bound of the average access request rate, denoted as λ_u, in the presence of the hot spot can be calculated by substituting the maximum access rate $\lambda = 1$ to (2.33).

$$\lambda_u \le \frac{B}{t_r} \tag{2.35}$$

where

$$B = \frac{1}{n\left[(1+(m-1)\lambda_h) + 2(1-\lambda_h)(1-\lambda_l)\right]}.$$

Formula (2.35) indicates that network transactions will be slowed down (λ_u is decreased) when the numbers of slots of the rings (m and n) are increased, and when the frequency of access to the hot spot (λ_h) is increased.

Quantitative evaluation of effects of a given buffer length on an average access request rate in the presence of the hot spot is another important factor to be considered. Let L_{ob} be an average length of the access request queue buffered in the port entering the hot local ring. It then can be calculated by Little's law and the Untilization law based on the different types of request rates (λ, λ_h and λ_l), the numbers of slots in the global ring (m) and in a local ring (n), and the ring rotation period (t_r):

$$L_{ob} = \frac{\left[(m-1)n\lambda\lambda_h + 2n(1-\lambda_h)(1-\lambda_l)\lambda\right]t_r}{1-n\lambda t_r\left[3(m-1)\lambda_h + 2(m+1)(1-\lambda_h)(1-\lambda_l)\right]} \tag{2.36}$$

For a given buffer length in design, denoted as L_d, the buffer in the port of the hot ring is full or overflow when $L_d \le L_{ob}$. Another upper bound of access request rate based on the buffer length, denoted as λ_{buf} can be calculated based on (2.36),

$$\lambda_{buf} \le \frac{A_1 A_2}{t_r}, \tag{2.37}$$

where

$$A_1 = \frac{L_d}{2n(n+mL_d+L_d)(1-\lambda_l)},$$

and

$$A_2 = \frac{2(n+mL_d+L_d)(1-\lambda_l)}{(m-1)(3L_d+n)\lambda_h + 2(n+mL_d+L_d)(1-\lambda_h)(1-\lambda_l)}.$$

Formula (2.37) indicates that the upper bound access request rate of the system, λ_{buf} is increased when queue length L_d is increased, and it is decreased when the the ring rotation period t_r is increased. Furthermore, Formula (2.37) presents a clear view of hot spot effects on the average access request rate λ_{buf}. Substituting $\lambda_h = 0$ to (2.37), we have $\lambda_{buf} \le \frac{A_1}{t_r}$, ($A_2 = 1$), which gives the average access request rate without the presence of the hot spot. In other words, A_2 is the reduction factor for the access request rate in the presence of the hot spot. Assume that non-hot spot requests are evenly distributed in the global ring, namely $\lambda_l = \frac{1}{m}$. For $\lambda_h = 1$, and $\lambda_l = \frac{1}{m}$, we have

$$A_2 = \frac{2}{2 + \frac{mL_d + m - 2 - 2L_d}{mL_d + L_d + 1}} \tag{2.38}$$

where

$$\frac{mL_d + m - 2 - 2L_d}{mL_d + L_d + 1} < 2.$$

Then the lower bound of A_2 is

$$A_2 > \frac{1}{2} \tag{2.39}$$

This gives an important performance result of hot spot effects on the ring network. It indicates that network transactions will be reduced no more than 50% in the presence of the hot spot comparing with the network transactions without the presence of the hot spot in the hierarchical ring architecture. The rotating ring orders and delays remote data access requests, this structure may naturally reduce network contention for programs with hot spots. We will confirm our comparative analytical results by experiments on the TC2000 and the KSR1 in Section III.

As our analytical study indicates, the reason why an HR-based architecture can significantly reduce network contention for programs with hot spots is because rotating rings delay remote data access requests in order. On the other hand, the memory access request rate would be limited by the system in the presence of a hot spot. Based on the analytical models, we can observe various performance effects in the presence of a hot spot. The modeled HR-based architecture has 32 rings in two levels. Each ring has 32 processor nodes.

Since the request rate is controlled by the ring structure, request delays would not be significantly increased in the presence of hot spots. In Fig. 4, a group of memory delay curves with different hot spot fractions (vertical axis) grows as the access request rate λ grows (horizontal axis), where the ring network has 32 local rings, each of which has 32 processors, and the request rate is bounded to 0.5×10^{-3}. Examing Fig. 4, one can observe three features of memory latency changes in an HR-based network in the presence of hot spots. First, hot spots have no significant effects on request access delays. Memory latency is slightly increased as the hot spot fractions are increased. Second, the memory latency is mainly determined by the access request rate. Especially, when the request rate reaches the upper bound, the memory latency is sharply increased. Finally, comparing with a nonblocking MIN-based network (see Fig. 2), high access request rate is a major factor to memory latency on an HR-based network, while the presence of hot spots is a major factor on a MIN-based network.

The analytical models can also help us to examine traffic distributions among cool rings and the hot ring in the presense of a hot spot. Fig. 5 illustrates the average delay factors for the three types of accesses: from a cool ring to the hot ring, from a cool ring to another cool ring and from the hot ring a cool ring. The delay factors were calculated in the presense of 32% hot spot fraction. An access from the hot ring to a cool ring has the longest delay, while an access from a cool ring to another cool ring has the shortest delay. However, only when the request rate reaches to the upper bound, can the delay factor differences

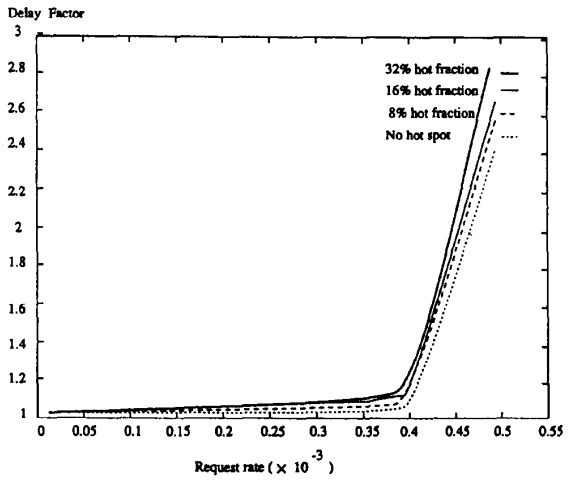


Fig. 4. Request delays in the presence of a hot spot on a two level 32 rings

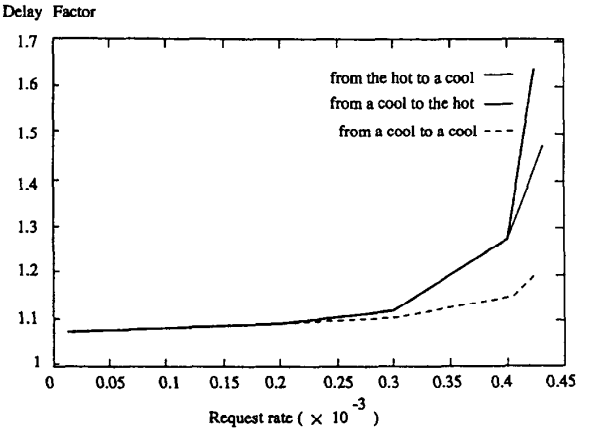


Fig. 5. Request delay distributions in the presence of a 32% hot spot fraction

among the three types of accesses be observed. For example, the difference could be at most 37% between a cool to cool access and a hot to cool access when the request rate is 0.43×10^{-3}. Fig. 5 indicates that the traffic in the ring network is reasonably balanced even when a hot spot appears in the system. This is because the rotating ring orders and delays remote access. The experimental measurements on the KSR1 presented in Section III confirm this analytical result.

B.2. Remote Access Delay in the Presence of Movable Hot Spots

Fixed hot spots described in the previous sections appear in Non-CC-NUMA and CC-NUMA architectures. A non-CC-NUMA architecture either supports no local caches (e.g., the BBN GP1000) or provides local cache but disallows caching of shared data to avoid the cache coherence problem (e.g., the BBN TC2000). In a CC-NUMA architecture, each processor node consists of a high-performance processor, the associated cache, and a portion of the global shared memory. Like CC-NUMA, each processor node has a high-performance processor, a cache, and a portion of the global shared memory. The difference, however, is that the memory associated with each node is augmented to act as a large cache. Consistency among cache blocks in the system is maintained using a write-invalidate protocol. A COMA system allows transparent migration and replication of data items to nodes where they are referenced. In this case, many processors access the same variable which is not fixed in one physical location. Therefore, the name of the variable is hot but not a single memory location.

In general, the location of a movable hot spot is randomly distributed among the processors. Assume that each processor has the same rate of memory accesses, the distributions a hot spot over a period of time can be considered as an uniform distribution. Therefore, the modeling of the network contention in the presence of a movable hot spot is a special case in the models presented in the previous section. Considering the hot spot accesses as a portion of uniform memory accesses, and using formulas (2.5) to (2.25) with $\lambda_h = 0$, we can derive the average local request delay $d_{lmovable}$ and the average remote request delay $d_{rmovable}$ as follows:

$$d_{lmovable} = \bar{d}_{lfind} + nt_r , \tag{2.40}$$

$$d_{rmovable} = \bar{d}_{lfind} + 2\bar{d}_{gwait} + 2\bar{d}_{lwait} + (m+2n)t_r . \tag{2.41}$$

By (2.8), the average time of finding an empty slot in a local ring is

$$\bar{d}_{lfind} = \frac{n\lambda t_r^2(3-2\lambda_l)}{1-n\lambda t_r(3-2\lambda_l)} . \tag{2.42}$$

By (2.10) and (2.5), the average queueing delay in the interface port on a local ring is

$$\bar{d}_{lwait} = \frac{\bar{d}_{lfind} + t_r}{1-2n\lambda(1-\lambda_l)(\bar{d}_{lfind} + t_r)} . \tag{2.43}$$

By (2.22), the average queuing delay in an interface port on the global ring can be expressed as

$$\bar{d}_{gwait} = \frac{t_r}{1-2n\lambda t_r(m+1)(1-\lambda_l)} . \tag{2.44}$$

Our analysis shows that a movable hot spot reduces network contention in general. This analytical result has been verified by experimental results on the KSR-1 in Section III.

B.3. Some HR-Based Architecture Factors

Based on the analytical models of HRs described in the previous sections, we can also produce several important factors for HR-based architecture design. Fig. 6 illustrates the effects of the numbers of processors of a ring on the limit of memory access rate (upper bound of the request rate). It indicates that increasing the number of processors and slowing down the rotation speed (increasing the rotation period) of the ring will limit the request rate. Fig. 7 illustrates the effects of the queue length of a port buffer between a local ring and the global ring on the memory access request rate. It indicates that memory access request rate is almost independent of the queue length. This fact can be explained as follows. If the queue length in the buffer is increased, traffic contention may be reduced by increasing the waiting time for a request in the buffer. Therefore, the memory request rate limit will remain as a constant when the queue length is adjusted.

Fig. 8 shows the access delay is also dependent on the number of processors in each ring besides the request rate. For a given rotation rate and a given request rate without presence of hot spots, increasing the number of processors in a ring will increase the request delay. Another dependent factor of the access delay is the ring rotation period. Fig. 9 illustrates the request delay curves by changing the request rate for different given rotation periods. The faster the ring is rotated, the lower delay will be.

III. Experimental Performance Evaluation

A. The Experiment Testbeds

A.1. The BBN TC2000

The MIN-based execution testbed in our study is the BBN TC2000 [3], which is the latest and the most powerful member of the BBN family, supporting up to 512 Motorola 88100 processors nodes. A butterfly switch composed of 8×8 switches is used for the network. The processors are operated at a clock speed of 20 MHz. The bandwidth of each path of a switch is 38Mb per second in the TC2000. However, the network speed is still not fast enough to catch up to the fast processor speed, causing greater network contention. Each processor includes 16MBytes of memory that can be accessed from any processor in the system via the network. Each processor in the TC2000 has a Motorola 88200 paged memory management unit for virtual memory processing. Each processor provides a 32K byte code cache and a 16K byte data cache. The TC2000 avoids the cache coherence problem by disallowing caching of shared data. This scheme caches private data, shared data that is read-only, and instructions, while references to modifiable shared data bypass the cache. In addition, an interleaved shared-memory scheme is supported as an option in order to reduce memory contention in applications.

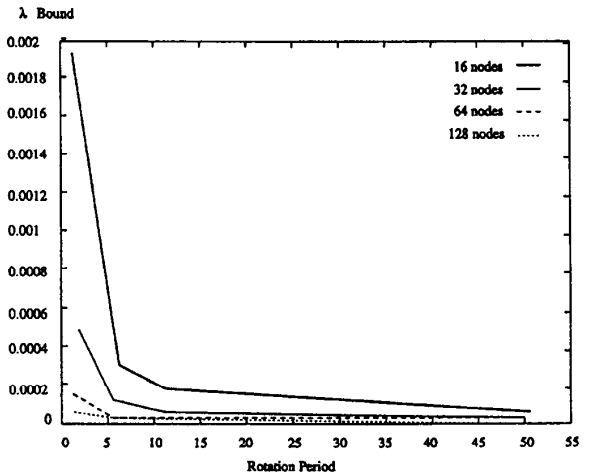


Fig. 6. Effects of the numbers of processors of a ring on the limit of memory access rate.

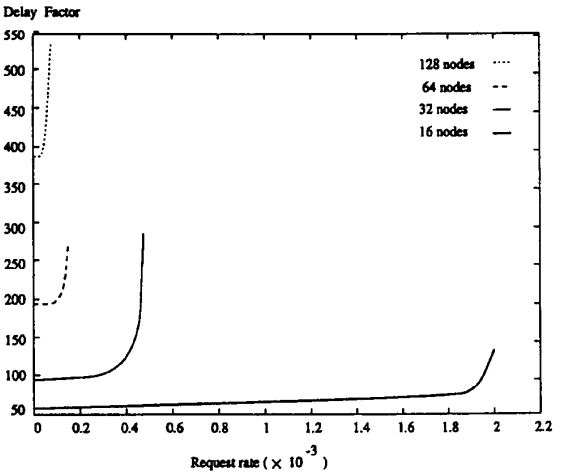


Fig. 8. Effects of the number of processors in each ring on average memory access request delay (1 delay time unit = 1 rotation period unit).

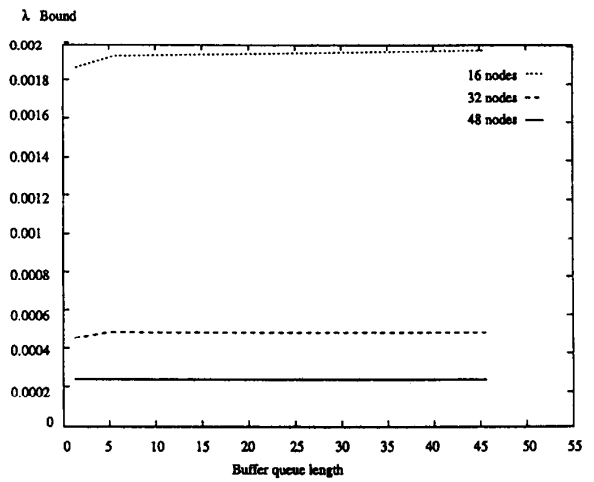


Fig. 7. Effects of the queue length of a port buffer between a local ring and the global ring on the limit of memory access rate.

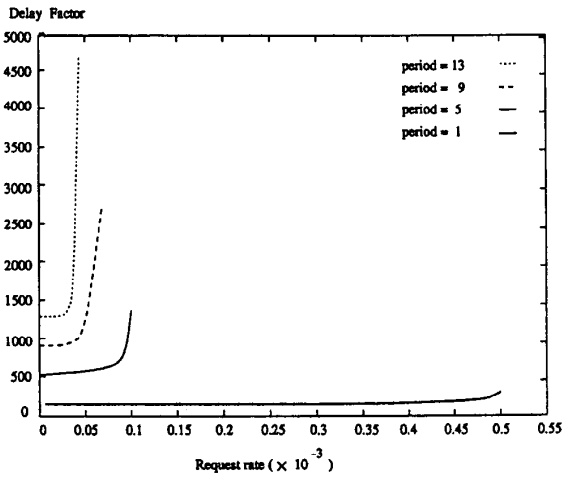


Fig. 9. Effects of the ring rotation period on average memory access request delay (1 delay time unit = 1 rotation period unit).

A.2. The KSR1 System

The KSR1 [8], introduced by Kendall Square Research, is an HR-based and cache coherent shared-memory multiprocessor system with up to 1,088 64-bit custom superscalar RISC processors (20 MHz). A basic ring unit in the KSR1 has 32 processors. The system uses a two-level hierarchy to interconnect 34 rings (1088 processors). Each processor has a 32MB cache.

The basic structure of the KSR1 is the slotted ring, where the ring bandwidth is divided into a number of slots circulating continuously through the ring. The number of slots in the ring is equal to the number of processors plus the number of routers connecting to the upper ring. A standard KSR1 ring has 34 message slots, where 32 are designed for the 32 processors and the remaining two slots are used by the directory cell connecting to level 1 ring. Each slot can be loaded with a packet, made up of a 16 byte header and 128 bytes of data which is the basic data unit in the KSR1, called a subpage. A processor in the ring ready to transmit a message waits until an empty slot is available. A single bit in the header of the slot identifies an empty slot. as the slot rotates through a ring interface of the processor.

B. The Structure of the Experiments

Two sets of experiments were conducted to measure and compare the performance of the KSR1 and the TC2000 in the presence of hot spots. The first set of the experiments measured the effects of hot spots on different remote cache/memory access and block transfer operations. This was done by comparing different access times in an environment without any hot spots and an environment where some processor nodes were used to generate a hot spot. The second set of experiments measured and compared simple spin-lock and distributed lock algorithms' performance.

In the first set of experiments, the hot spots were generated in two different ways:

1) *Via read and write references:* a set of processors were used to make a target cache/memory module hot by reading and writing the same location in that cache/memory module.
2) *Via block transfer:* a set of processors were used to make a target cache/memory hot by using the block transfer operation to copy data from that cache/memory module to their local cache/memory modules.

In the second set of experiments, two types of synchronization locks are used: simple spin-lock and distributed lock. The simple spin-lock consist of a globally shared lock allocated on a single processor to which all access to it is serialized. When a processor gets the lock, it sets the lock busy immediately and starts the atomic operations. An unlock is set by the processor immediately after the critical section work is finished. The rest of the processors have to busy-wait for the lock using remote memory accesses through an interconnection network. We construct the simple lock using the atomic *test_and_set* operation.

In order to reduce contention in the network and in the memory module holding the lock, it would be more efficient to have each processor busy-wait only on a locally-accessible variable for the lock. A distributed algorithm decentralizes locks throughout the memory modules. The concept of the distributed algorithm has been implemented and tested on various shared-memory systems [2], [10], and [15]. In fact, the execution behavior of spin-locks is significantly different between MIN-based and HR-based architectures. For example, while best algorithms for the KSR1 are simple locks with delay options, best algorithms for the TC2000 are distributed locks.

C. Comparative Hot Spot Effects on Cache/Memory Access

Four types of cache/memory accesses were measured:

1) Single word read (4 bytes) accesses;
2) Single word write (4 bytes) accesses;
3) Block transfer of data from the remote cache/memory;
4) Block transfer of data to the remote cache/memory;

where the block length is defined as 128 bytes. Different number of blocks are applied for block transfer accesses.

To clarify the terminologies that will be used in this section pertaining to hot spots, here we describe the important definitions in the experiments:

- **Remote access** refers to when a processor makes a memory/cache reference to another processor. In the KSR1, this access could be a remote cache access in the local ring or a remote access in a remote ring. In the TC2000, all remote memory accesses are of the same distance.
- **Hot spot** is a shared variable that is accessed simultaneously by a large number of processors which are called **hot processors.**
- **Hot memory/cache** is the memory/cache module where the hot spot variable resides.
- **Cool processors** are the processors in the system which do not access the hot spot.
- **Cool variables in hot memory/cache** are non-hot spots in the hot memory/cache, and will be accessed by the cool processors remotely.
- **Cool variables in cool memory/cache** are non-hot spots in non-hot memory/cache, and will be accessed by cool processors remotely.

Fig. 10 illustrates the above definitions for the hot spot experiments, where variable X is the hot spot, variables a and b are cool variables in the hot memory/cache and variables c and d are cool variables in a cool memory/cache. The system in Fig. 10 assumes p processors are used in the system where h nodes are used for hot processors and $p - h$ nodes are used for cool processors, and $h >> p - h$.

C.1. Performance on the KSR1

The hot spot on the KSR1 is allocated either in a fixed location, called the *fixed hot spot* or in movable locations, called *movable hot spot.* The fixed hot spot remains physically on one processor as other processors try to read it with a single variable or a block of data. The movable hot spot will be migrated around the ring on demand of any processor who does a read with a single variable or a block of data. This data migration is a feature of the KSR1 intended to enhance the data locality.

In the fixed hot spot experiment on the KSR1, the hot spot must be directly and intentionally placed in a fixed memory location. This action is necessary because when a memory reference is normally referenced, the system migrates that data to the cache of the requesting processor. Therefore, a scheme is required to access a variable or a block of data and have it remain there during the reference. It is accomplished in the following way. The hot variable was represented as a long vector. Each hot processor read a different element of the hot variable vector. This ensures that only that element will be read and that the entire hot variable vector will not be migrated over to that hot processor. The number of times that the hot variable was read by each hot processor was 1000 times (1000 distinct elements per hot processor). The type of the hot variable is also important because types differ according to size and the unit of transfer in the KSR1 is 128 bytes. If the hot variable vector is of type "integer" then each element is 4 Bytes long. Therefore when a hot processor reads the next element it must be increased by 32. Since when one element is read, an entire subpage (128 Bytes) will be actually read from the hot variable. If the next element to be read is not increased by 32, then the next element to be read will already reside in the hot processor's cache, thus making this reading a local read within its own cache. If the hot variable vector is of type "block" then each element is 128 bytes long. Therefore when a hot processor reads the next element it must be increased by only one (because each element in the hot variable vector is equal to a subpage). The reading of the cool variables is handled in the same manner. Since the cool variables can be either a single variable or a "block" the increase to access the next element will be 32 and 1, 2, or 3, respectively. The access of the cool variables was also done 1,000 times per cool processor. The average

timing results are used as the final results. Since a remote-write which updates a variable in a remote cache module is not supported on the KSR1, we only used read operations for the hot spot experiments.

The movable hot spot experiments were implemented using normal cache access on the KSR1, where the hot spot migrates across the ring from processor to processor on reference demand.

In this implementation, any cache among the hot processors would be "hot" at any given time. Since we cannot possibly know which cache is hot, the measurements of accessing cool variables at the hot cache were not conducted.

For both fixed and movable hot spot experiments on the KSR1 system, 57 processors were used to generate the hot spot. Since only 64 processors are available for computing in the two ring system, there was one remote hot cache module, 6 remote cool cache modules. Before presenting our experimental results of hot spot effects, we list the standard access times in Table I published by the KSR [8], for a comparison and a reference.

TABLE I
STANDARD CACHE ACCESS TIMES ON THE KSR1

	Access time (μs)
Subcache	0.1
Local cache	1.0
Remote cache in the local ring	7.5
Remote cache in a remote ring	28.5

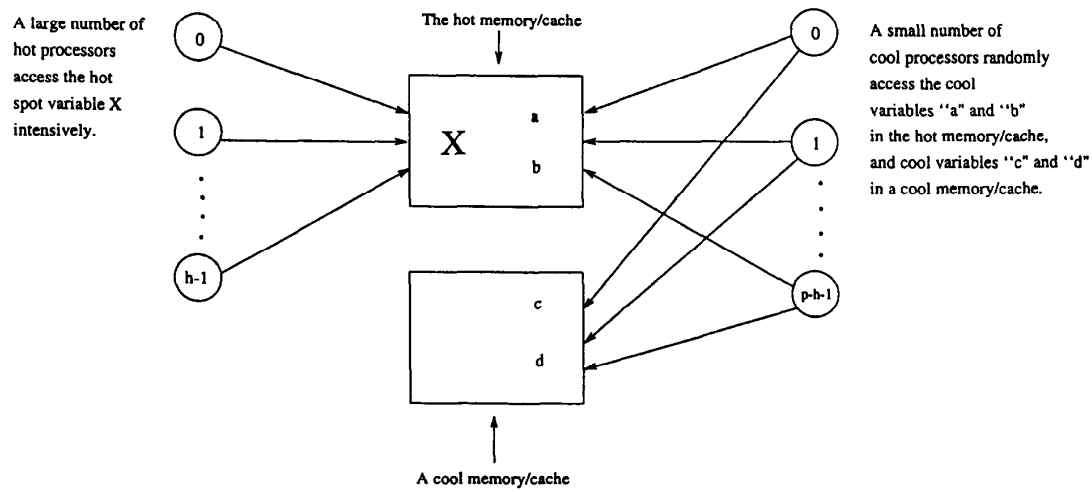
Table II reports the average timing results of the fixed hot spot experiments. The first data column lists the results of remote-read of one word; the second, third and fourth data columns list the results of remote-read of one block, two blocks and three blocks respectively. These data transactions are under the environment without any hot spots (the 1st data row), the environment with the hot spot generated by cache references in a word unit (the 2nd data block row), and the environment with the hot spot generated by cache references in a block unit (the 3rd data block row).

TABLE II
VARIOUS REMOTE CACHE ACCESS TIME MEASUREMENTS (μs) IN THE PRESENCE OF A FIXED HOT SPOT ON THE KSR1

	read (1 int)	read (1 blk)	read (2 blks)	read (3 blks)
No Hot Spot	9.57	14.64	28.04	42.14
Hot Spot (Var)				
From Cool Cache	9.70	14.28	28.07	42.23
From Hot Cache	83.08	81.74	103.85	117.29
Hot Spot (Block)				
From Cool Cache	9.79	14.26	27.89	42.06
From Hot Cache	82.60	84.22	105.52	118.67

Considering the different level accesses and block transfers in the experiments, the remote cache access measurements (first data row in Table II) in a non-hot-spot environment are quite consistent with the standard results in Table I. When there was a hot spot present, remote access to cool variables in the cool cache modules were not affected. However, there is a degradation in performance when referencing the cool variables from the hot cache module. According to present information available on the KSR1 pertaining to the network structure (HR-based), a claim is made that the rotating ring(s) that make up the network of the KSR1 rotate in a balance efficient manner. This means that a hot spot should have little effect on other processors who need referencing through the ring.

In practice, the claim of the balanced network ring rotation is indeed true. The rotating ring orders and delays remote access and makes access path to each processor of the system almost equally busy in the presence of hot spots. The cause of the degradation is actually at the processor location itself. When a processor is receiving many requests for it to process, a *queue* will made of these requests so that it will be able to eventually process them. Since the job of the CIUs (Cell Interconnecting Units)



of a processor is to extract the incoming requests and pass them to the CCUs (Cache Control Units) to act upon them, the queue will be located at the CCU to be able to eventually process them. This will obviously slow down the time for the cool processors that are trying to reference the cool variables that are located on the hot cache. Further, if a processor is receiving an overwhelmingly number of requests, then that processor will eventually begin to ignore other additional requests that may also be coming to it (because the queue has become full since the processor speed is not fast enough to satisfy all the requests in time). This will then force the requesting processor(s) to reattempt to try again to succeed in either retrieving the data item it needs or be inserted into the hot processor's queue. Either way will cause a considerable delay in access time.

Table III lists the average timing results of the movable hot spot experiments. In general, movable hot spots reduce the access delay time on different types of remote access in the presence of the hot spot. However, the overhead of moving data and corresponding data item invalidations in the ring network is also significant. In comparison with the fixed hot spot experiments, Table III indicates that a movable hot spot slightly increases the access delay to cool variables in the cool cache modules due to heavier traffic caused by more data movement. Although we were not able to measure the access delay to cool variables in the hot cache module for reasons stated above, we believe the delay would be about the same as the access delay to cool variables in the cool cache modules based on our analytical and fixed hot spot experimental results.

TABLE III
VARIOUS REMOTE CACHE ACCESS TIME MEASUREMENTS (μs) IN THE PRESENCE OF A MOVABLE HOT SPOT ON THE KSR1

	read (1 int)	read (1 blk)	read (2 blks)	read (3 blks)
No Hot Spot	9.57	14.64	28.04	42.14
Hot Spot (Var) From Cool Cache	9.52	15.27	30.35	44.42
Hot Spot (Block) From Cool Cache	10.07	15.59	30.75	45.87

Another experiment to verify our modeling work is to see if a hot spot can effect remote readings among processors that are not involved in the process of generating the hot spot. Again, two rings were used for the experiment. The difference between this experiment and the previous one is that all of the processors contributing towards generating the hot spot are on the same ring (the *hot ring*). While the other processors on the other ring (the *cool ring*) are not involved in generating the hot spot. The hot spot is fixed to one processor within the hot ring. We varied the experiment by increasing the number of processors to be involved in the hot ring to generate the hot spot. We chose to use 50% and 97% of the available processors on the hot ring to generate the hot spot for these variations. Thus, for 50% usage of the processors on the hot ring there were also 16 processors that were not involved in generating the hot spot. At the same time, there were also 16 processors to be used on the cool ring. These two sets of 16 processors were used to do remote readings of their counter processors, respectively. These remote readings were also unidirectional among the two rings during any run of the experiment. Thus, the 16 processors on the cool ring read the 16 processors on the hot ring during the hot spot activity during one run of the experiment. Then we reversed the remote reading and had the 16 processors on the hot ring read remotely the 16 processors on the cool ring during the hot spot activity during another run. The hot spot was generated by either reading a single variable or by reading a block of data. Thus, there were four different timings from the hot spot activity that was compared to the remote readings when there was no hot spot. For 97% of processors on the hot ring in usage there was one available processor on the hot ring and one available processor on the cool ring. As shown in Tables IV and V for the different variations, the hot spot has very little effect for all the remote reads on this experiment. This additional experiment on the KSR1 further strengthens our analysis that a HR-based architecture, such as the KSR machine handles the activity of the hot spot efficiently.

TABLE IV
READING MEASUREMENTS (μs) WHEN HOT SPOT IS GENERATED BY 50% OF PROCESSORS ON THE HOT RING ON THE KSR1

	read (1 int)	read (1 blk)	read (2 blks)	read (3 blks)
From $Cool_R$ To $Cool_R$	28.66	32.79	66.48	98.14
Hot Spot (Var) From Hot_R To $Cool_R$	27.97	32.92	66.55	98.03
From Coo_R To Hot_R	28.74	33.40	66.58	98.07
Hot Spot (Block) From Hot_R To $Cool_R$	28.80	33.13	66.56	98.64
From $Cool_R$ To Hot_R	28.70	33.01	66.61	98.96

TABLE V
READING MEASUREMENTS (μs) WHEN HOT SPOT IS GENERATED BY 97% OF PROCESSORS ON THE HOT RING ON THE KSR1.

	read (1 int)	read (1 blk)	read (2 blks)	read (3 blks)
From $Cool_R$ To $Cool_R$	27.97	32.29	63.40	94.80
Hot Spot (Var) From Hot_R To $Cool_R$	28.07	32.32	63.35	94.69
From Coo_R To Hot_R	28.27	32.95	63.85	94.95
Hot Spot (Block) From Hot_R To $Cool_R$	27.69	32.42	64.03	95.07
From $Cool_R$ To Hot_R	27.47	32.74	65.46	96.23

C.2. Performance on the TC2000

We conducted similar experiments described in the previous section on the TC2000, where remote write operations are included. The hot spot experiments on the TC2000 have the following three different features from the ones running on the KSR1 in terms of architecture support and memory systems. First, the hot spot can only be allocated in a fixed physical location. All memory accesses to the hot spot and cool memory modules are conducted through the Butterfly interconnection switching network. Hot spot experiments are performed by using remote read and remote write. Second, no global cache coherence protocols are supported in the system. Finally, the network contention happens when two messages need to access the same portion of a path at the same time. The TC2000 supports a nonblocking switching network. When

TABLE VI
VARIOUS REMOTE MEMORY ACCESS TIME MEASUREMENTS (μs) IN THE PRESENCE OF THE HOT SPOT MEMORY ON THE TC2000

	read (int)	write (int)	read (1 blk)	write (1 blk)	read (3 blks)	write (3 blks)
No Hot Spot	2.97	3.36	87.15	97.90	262.40	293.25
Hot Spot (Var) From Cool Memory	9.59	10.38	327.31	385.67	982.82	1155.47
From Hot Memory	106.87	111.79	510.96	568.72	1159.79	1331.49
Hot Spot (Block) From Cool Memory	9.64	10.75	326.38	384.22	980.18	1155.67
From Hot Memory	115.28	118.49	3429.60	3615.32	6972.84	7132.77

conflicts happen, the switch has all but the "first" message retreat back to its source and free up their path. It then selects an alternative route, and after a random delay, tries again.

In our experiments on the TC2000, 100 processors were used to generate the hot spot. There was one remote hot memory module, 11 remote cool memory modules. Table VI gives the measured access times on the TC2000. When there is a hot spot memory, remote accesses to cool memory were slowed down more than three to four times for all types of hot spot experiments. These results are considerably different from the results of similar experiments on the Butterfly I conducted by Thomas [13], where accesses to cool memory were slowed down slightly. In addition, accesses to cool memory are slowed down about the same amount as block transfer accesses to cool memory. As we briefly discussed in the previous section, the TC2000 has heavier inherent network contention than the Butterfly I because the faster processors generate more network traffic. Therefore, memory accesses are significantly delayed in the presence of the hot spot. Accesses to the hot spot memory are substantially slower on the TC2000. These results are consistent with the results on the Butterfly I reported in [13]. This is because an access to the hot memory must compete with all the other accesses to the same memory in all the switch states from its source processor to the target processor. Our experiments show that a nonblocking network suffers from large access time delay in the presence of the hot spot memory on the TC2000. The experiments also show that block transfer operations are substantially delayed on the TC2000 in the presence of memory hot spots. This is because a block transfer operation on the TC2000 will not release a sequence of switches along the path until the data transfer is done.

D. Comparative Hot Spot Effects on Synchronization Algorithms Between MIN and HR

D.1. Synchronization Algorithms

As we analyzed in the Section II, an HR-based architecture takes advantage of higher bandwidth links, efficient broadcast, and ordering properties of ring networks. Unlike a MIN-based architecture, each processor in the KSR1 does not have direct communication links to other processors except one neighbor. All the messages sent by a processor are loaded to the slotted ring to be delivered to a target processor. Since rotating rings delay remote data access requests in order, this structure can significantly reduce network contention for programs with hot spots. In order to further confirm this, we compare the hot spot effects of architectures between the TC2000 and the KSR1 by measuring, comparing and evaluating synchronization algorithms on the TC2000 and the KSR1.

- **The Simple Lock and Its Variations.** The simple lock consists of a globally shared lock allocated on a single processor with all access to it serialized. When a processor gets the lock, it sets the lock busy immediately, and starts the atomic operations. An unlock is set by the processor immediately after the critical section work is finished. The rest of the processors have to busy-wait for the lock using remote memory accesses through an interconnection network. Using the atomic *test_and_set* operation, the simple lock and its variations can be constructed. Variations of the simple lock may be implemented by applying time delays between attempts to access the lock variable for each processor.
- **Distributed Locks.** In order to reduce contention in the network and in the memory module holding the lock, it would be more efficient to have each processor busy-wait only on a locally-accessible variable for the lock. A distributed algorithm decentralizes locks throughout the memory modules.

D.2. Performance on the TC2000 and the KSR1

Recently, new experimental results of spin-locks on both the TC2000 and the KSR1 were reported in [15]. While the algorithm of the simple spin lock with no delay functions maximized the number of remote accesses to a single data location (lock) through the MIN network, the distributed lock algorithm minimized the number of remote accesses to the single location through the MIN network. However, on the KSR1 we have chosen the simple spin lock algorithms (using the *gspnwt* instruction) for a comparison instead of using the best overall compared to the worst overall in timings. The reason for this choice is because of how the KSR1 handled the *ticket* lock algorithm (the worst overall timing) in its implementation. As is described in [15], the "ping pong" effect that leads to the cache coherence problem coupled with the "skipping over" phenomenon causes its poor performance. More important, the architecture of the KSR1 did not truly exploit the algorithm as it should have. We feel that use of the timing of the *ticket* lock algorithm could be misleading in our comparison. In the algorithms using the simple lock with delay options, the KSR1 truly exploits the algorithms in its architecture. Therefore, for a fair comparison we have chosen to use the worst and the best timings in these simple lock algorithms, namely *s_lock* (no delay) and *s_lock* (exponential delay). Using a linear least square approximation, the slope of the timing curves from the experiments presented in [15], can be calculated. Each slope gives an average time increment in μs or *ms* for the addition of each processor for each synchronization algorithm. Table VII

hows the slopes for the simple lock (with no delay and exponential delay) and the distributed lock for the TC2000 (where applicable) and the KSR1 (where applicable). The ratios are calculated for the hot spot effect comparison between the TC2000 and the KSR1. The ratio is between the slope of worst algorithm timing and the best algorithm timing for each machine. The ratios also quantitatively represent the increase of network contention generated by the hot spot of the simple spin lock. The higher the ratio, the higher the network contention will be. Comparing the ratios for the two architectures clearly indicates that the KSR1 architecture is significantly less sensitive to the hot spot in synchronization program than the TC2000. Comparing the ratios of the two architectures the KSR1 would have about 14 times (*ratio*(TC2000) / *ratio*1(KSR1)) less network contention generated in its architecture for spin-lock synchronization hot spot.

TABLE VII
COMPARISONS OF THE LOCK EXECUTION TIME SLOPES ON THE TC2000 AND THE KSR1

	s_lock (no delay)	s_lock (exponential delay)	dis_lock	ratio
TC2000	0.68 *(ms)*	N/A	0.011 *(ms)*	61.82
KSR1	5.32 *(μs)*	1.20 *(μs)*	N/A	4.43

IV. SUMMARY

We have conducted analytical and experimental studies on the remote memory accesses and network contention in the presence of memory hot spots. Experiments were performed on the TC2000, a MIN-based system, and on the KSR1, an HR-based system. In both MIN-based and HR-based systems, a processor can access shared cache/memory space at different distances with different timing costs. For example, the TC2000 provides a local/remote memory access model, while the KSR1 provides a local/level0-ring/level1-ring cache access model. This type of shared-memory architecture is defined as a Non-Uniform Memory Access (NUMA) multiprocessor system. A potential problem of parallel processing in non-uniform memory access, defined as the NUMA problem, is that a user has to explicitly manage processor locality, the best use of the processor resources, and the reduction of different contentions in the execution of a program. "Hot spot" in a parallel program is an important source of degraded execution performance on NUMA architectures. In this study we investigated the following hot spot performance problems and their evaluation results based on analytical models and experiments:

- Analytical models of remote access delay in the presence of memory hot spots on a nonblocking MIN-based architecture are presented. Our analytical study indicates that an access delay to cool memory modules may be considerably smaller than an access delay to the hot memory. However, access delay to cool memory modules can be significantly delayed on a system with fast processors connected by relatively slow MIN where the chances that messages used to access cool memory modules collide with the messages used to access the hot spot, are potentially high. The tree saturation described in [11] showed that very slight non-uniformities in memory access patterns can lead to severely degraded performance for the entire system, including processor nodes that avoid accessing the hot spot. This may not apply to a nonblocking MIN-based architecture.
- In an HR-based architecture, a slotted ring orders and delays remote data access requests. This structure may naturally reduce network contention for programs with hot spots. Our analytical models indicate that ring network transactions will be reduced no more than 50% in the presence of memory hot spots. This analytical performance result has not been previously reported. Analysis also indicates that in the presense of hot spots overall ring traffic will be heavier but it will be distributed evenly in the ring network. Both analytical results have been verified by the experiments on the KSR1.
- Although there is no evidence that the tree saturation phenomenon occurs in the experiments on the TC2000, remote accesses to both hot and cool memory modules will be considerably slowed down, and performance may be significantly degraded. Our experimental results on the TC2000 show that the experimental results on the Butterfly I in [13] do not generalize to all nonblocking MIN-based architectures.
- In an HR-based architecture, a hot spot variable can be either fixed in a physical memory module or movable among the processor nodes. A movable hot spot may reduce network contention in general because accesses to the shared variable are distributed among the memory modules in the whole system. However, the overhead of moving data and cache coherence operations, such as data item invalidations, may bring extra costs. Our experiments on the KSR1 show that a movable hot spot provides higher memory latency than that of a fixed hot spot. We also show that block transfer operations on the KSR1 are much more efficient than that on the TC2000 in the presence of memory hot spots. This is because a block transfer operation on the TC2000 would not release a sequence of switches along the path until the data transfer is done, while a block transfer operation on the KSR1 breaks the block into a set of units and sends them one by one whenever a slot is available.
- Two major comparisons between the two network architectures are as follows. (1) High access request rate is a major factor to memory latency on an HR-based network, while the presence of hot spots is a major factor on a MIN-based network. (2) In the presence of hot spots, network traffic distributions could be much more balanced in an HR-based architecture than that of a MIN-based architecture.

Our current work includes proposing and examing efficient hardware modifications and system software on both nonblocking MIN-based and HR-based architectures to reduce effects of hot spot, cache coherence and processor locality management.

ACKNOWLEDGMENTS

We appreciate G. Butchee's careful reading of the manuscript and constructive comments. We wish to thank the anonymous referees for their helpful comments and suggestions.

This work is supported in part by the National Science Foundation under grants CCR-9102854 and CCR-9400719, by the U.S. Air Force under research agreement FD-204092-64157, by the Air Force Office of Scientific Research under grant AFOSR-95-01-0215, and by a fellowship from the Southwestern Bell Foundation. Part of the experiments were conducted on the BBN TC2000 in Lawrence Livermore National Laboratory, and on the KSR1 machines at Cornell University, and at the University of Washington.

REFERENCES

[1] A. Agarwal et. al., "APRIL: A processor architecture for multiprocessing," *Proc. of the 17th Int'l Symp. on Computer Architecture*, 1990, pp. 104-114.

[2] T.E. Anderson, "The performance of spin-lock alternatives for shared-memory multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 1, 1990, pp. 6-16.

[3] BBN Advanced Computer Inc., *Inside the TC2000*.

[4] L.A. Barroso and M. Dubois, "The performance of cache-coherent ring-based multiprocessors," *Proc. 20th Int'l Conf. of Computer Architectures*, IEEE Computer Society Press, Apr. 1993, pp. 268-277.

[5] L.N. Bhuyan, D. Ghosal, and Q. Yang, "Approximate analysis of single and multiple ring networks," *IEEE Trans. on Computers*, vol. 38, no. 7, July 1989, pp. 1,027-1,040.

[6] E. Gelenbe, *Multiprocessor Performance*, John Wiley and Sons, 1989.

[7] E. Hagersten, A. Landin, and S. Haridi, "DDM—a cache-only memory architecture," *IEEE Computer*, Sept. 1992, pp. 44-54.

[8] Kendall Square Research, *KSR1 Technology Background*.

[9] D. Lenoski et al., "The DASH prototype: Logic overhead and performance," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 1, 1993, pp. 41-61.

[10] J.M. Mellor-Crummey and M.L. Scott, "Algorithms for scalable synchronization on shared-memory multiprocessors," *ACM Trans. on Computer Systems*, vol. 9, no. 1, 1991, pp. 21-65.

[11] G.F. Pfister and V.A. Norton, "'Hot spot' contention and combining in multistage interconnection networks," *IEEE Trans. on Computers*, vol. 34, no. 10, pp. 943-948, 1985.

[12] P. Stenström, T. Joe, and A. Gupta, "Comparative performance of cache-coherent NUMA and COMA architectures," *Proc. 19th Int'l Symp. of Computer Architectures*, 1992, pp. 80-91.

[13] R.H. Thomas, "Behavior of the Butterfly parallel processor in the presence of memory hot spots," *Proc. 1986 Int'l Conf. on Parallel Processing*, pp. 51-58, 1986.

[14] Y. Yan and X. Zhang, *Performance modeling and analysis of MIN-based and HR-based networks*, Technical Report, High Performance Computing and Software Laboratory, University of Texas at San Antonio, Dec. 1993.

[15] X. Zhang, R. Castañeda, and W.E. Chan, "Spin-lock synchronization on the Butterfly and KSR-1," *IEEE Parallel and Distributed Technology*, vol. 2, spring issue, 1994, pp. 51-63.

[16] X. Zhang and X. Qin, "Performance prediction and evaluation of parallel processing on a NUMA multiprocessor," *IEEE Trans. on Software Engineering*, vol. 17, no. 10, 1991, pp. 1,059-1,068.

[17] X. Zhang and Y. Yan, "Comparative modeling and evaluation of CC-NUMA and COMA systems in hierarchical rings," to appear in *IEEE Trans. on Parallel and Distributed Systems*.

Xiaodong Zhang received the BS degree in electrical engineering from Beijing Polytechnic University, China, in 1982 and the MS and PhD degrees in computer science from the University of Colorado at Boulder in 1985 and 1989, respectively.

He is an asscoiate professor of computer science and director of the High Performance and Computing and Software Laboratory at the University of Texas at San Antonio. He has held research and visiting faculty positions at Rice University and Texas A&M University. His research interests are parallel and distributed computation, parallel architecture and system performance evaluation, and scientific computing.

Zhang has served on the program committees of several conferences, and is the program chair of the Fourth International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'96). He also currently serves on the editorial board of *Parallel Computing*, and is an ACM National Lecturer.

Yong Yan is a PhD student of computer science at the University of Texas at San Antonio. He received the BS and MS degrees in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1984 and 1987, respectively. He has been a faculty member there since 1987. He was a visiting scholar in the High Performance Computing and Software Laboratory at UTSA, 1993-1995. Since 1987, he has published extensively in the areas of parallel and distributed computing, performance evaluation, operating systems and algorithm analysis.

Robert Castañeda is a PhD student of computer science at the University of Texas at San Antonio. He received the BS and MS degrees in computer science from the same university in 1990 and 1994, respectively. He was a recipient of Southwestern Bell Foundation Graduate Fellowships, 1993-1995 and won the 1994 University Life Award for academic performance in his graduate study. He was a research associate at the High Performance Computing and Software Laboratory at UTSA, 1994-1995. His research interest is in the areas of performance evaluation of parallel and distributed architectures and systems.