Alleviation of Tree Saturation in Multistage Interconnection Networks

Matthew Farrens
Division of Computer Science
University of California
Davis, CA 95616
tel: (916) 752-9678
fax: (916) 752-4767
email: farrens@cs.ucdavis.edu

Brad Wetmore
Division of Computer Science
University of California
Davis, CA 95616
tel: (916) 752-2149
fax: (916) 752-4767
email: wetmore@cs.ucdavis.edu

Allison Woodruff
Division of Computer Science
University of California
Davis, CA 95616
tel: (916) 752-8878
fax: (916) 752-4767
email: woodruff@cs.ucdavis.edu

Abstract

This paper presents an examination of two distinct but complementary extensions of previous work on hot spot contention in multistage interconnection networks. The first extension focuses on the use of larger queues at the memory modules than are traditionally studied. The second extension explores a simple feedback damping scheme, which we refer to as *bleeding*, which allows selected processors to ignore feedback information.

The impact of memory queue size, feedback threshold value, and bleeding on system performance (specifically maximum bandwidth per processor) is evaluated by analyzing the results of extensive network simulations. These results indicate that combining these approaches can significantly improve the effective bandwidth of a multistage interconnection network.

1. Introduction

In order to achieve the goal of teraflop computing speeds by the end of the century, the number of processing nodes in a multiprocessor will have to increase substantially. However, as the number of processors grows, so does the impact of two problems inherent to multiprocessors: how can memory and processors be efficiently connected, and how can the side effects of memory contention be controlled?

There are several ways to interconnect a large number of processors. A shared bus memory architecture is the least expensive, but is inappropriate for systems with many processors and memory modules; the amount of traffic far exceeds the capacity of a single bus. A substantially more costly solution is a crossbar network in which every processor has a connection to every memory module via a set of intersecting wires. Such a network requires at least N^2 crosspoint switches (where N is the number of processors and memory modules), an impractical solution for large values of N. A popular compromise is to use multistage interconnection networks consisting of $\log_k N$ levels of $k \times k$ switches. Since these networks are inexpensive enough to be feasible, and provide sufficient bandwidth for many processors and memory modules to communicate simultaneously [Sieg85], they will be the focus of this paper.

Unfortunately, sharing memory leads to memory contention. Furthermore, the larger the number of processors, the greater the degree of contention [Lee89]. The result in multistage interconnection networks is an undesirable phenomenon known as *hot spots* [PfNo85]. Hot spots are locations (nodes within the network, memory modules, or specific addresses) which receive a nonuniform distribution of traffic. This concentration of traffic can be caused by events such as references to shared or global variables (e.g. for barrier synchronization or operations on semaphores [Lee85, TaYe90]); block transfers [Thom86]; a coincidental concentration of requests to a single memory module [Lee85, NoPf85]; or a coincidental concentration of traffic through an internal switch (nonuniform traffic spots, or NUTS [LaKu90]) which can occur even if the requests to memory are evenly distributed. The model used in this study assumes that queues exist between processors and switches, switches and switches, and switches and memory modules, in order to buffer requests (see Figure 1). When nonuniform traffic occurs, the queue(s) at the hot spot(s) become full and can not accept further requests; requests which are blocked will remain in the switches feeding the hot spot. These queues at these switches can also fill, etc., and the effects will propagate backward through the network, forming a tree of nodes whose queues are full.

Hot spots degrade performance of requests to both hot and cold modules. Since a hot module or queue has several pending requests but can only service one at a time, the mean service time for these requests is increased. Requests to cold memory modules ("cold" requests) can encounter full queues; therefore, the latency of the cold requests is increased, and the level of congestion in the network is heightened [PoHa89, ScSo90]. This degradation is called "tree saturation" and is compounded by the fact that its onset can be rapid [KuPf86] and may take a long time to alleviate. In addition, it takes very little nonuniform traffic to induce this condition [PfNo85, Ston87].

This work was supported by the National Science Foundation under Grant CCR-90-11535.

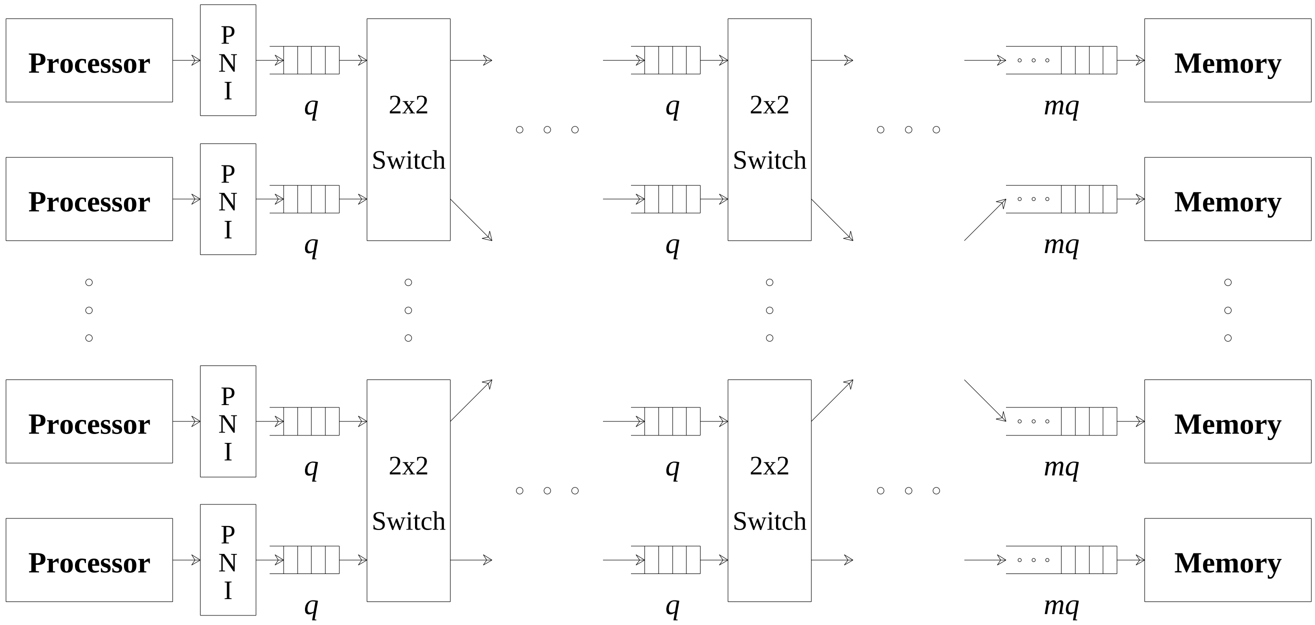

Figure 1.

Components of a Multistage Interconnection Network

2. Previous Approaches

Several methods of controlling tree saturation have been proposed. Most, however, have been impractical in terms of technology or expense. Three of these methods are described below.

2.1. Combining

One commonly discussed approach to reducing tree saturation is to *combine* requests to the same address. Both hardware [GGKM83, LeK86, PfNo85] and software [TaYe90, YeT87] solutions have been proposed. These approaches reduce the number of requests to a hot memory location, as well as minimize network traffic.

However, combining has several disadvantages: large hardware expense[1]; lack of scalability in practical applications [Lee89]; and failure to alleviate tree saturation arising from causes other than excessive competition for a single address.

2.2. Feedback

The technique of *feedback* can be used to control tree saturation as follows: a threshold value for the queues at all memory modules is chosen. When a queue's threshold is reached, all processors are notified to hold requests to the corresponding (hot) memory module. Once the queue length falls below the threshold, processors are informed that they may once again issue requests to the formerly hot module.

This can considerably decrease network congestion by limiting requests which would contribute to tree saturation, and therefore improve the performance of requests to cold modules. Unfortunately, because of the inherent latencies involved in the network, a new batch of requests may not arrive before the hot module services all of its queued requests (leaving the memory module idle). Scott and Sohi label this *undershoot*, an analogy from classic control systems theory [ScSo90].

Another problem with feedback occurs when several of the processors have saved a number of requests for a hot module. When the processors are notified that the module is again accepting requests, the stored requests will be simultaneously released by the processors, possibly flooding the network. This re-creation of tree saturation is termed *overshoot* [ScSo89, ScSo90].

2.3. Limiting

A *limiting* scheme assumes the presence of a global arbiter which controls the number of requests to each memory module that enter the network per cycle. Limiting can be used by itself as a scheme to control tree saturation, although it is difficult to predict how many requests to allow per cycle to maximize bandwidth and minimize the idle time of the memory modules.

[1] Lee *et al* found that the hardware cost is even higher than originally suspected, because suggested methods of combining are ineffective, necessitating even more sophisticated hardware to achieve favorable results [LeK86].

This scheme can be enhanced by using limiting in conjunction with feedback to temper both undershoot and overshoot. Undershoot will be avoided because there is a constant, albeit reduced, flow of requests to a hot module. Overshoot will be avoided because requests to the hot memory module will be released gradually. Unfortunately, the drawback to limiting is a direct result of its most prominent characteristic: it assumes global coordination, a complex and expensive solution.

3. Extensions to Previous Work

This paper examines two methods of controlling tree saturation: the use of feedback with larger queues at the memory modules, and the use of a damping mechanism which releases requests gradually into the network. These methods should reduce tree saturation occurring from nonuniform traffic to a given memory module, but should not affect tree saturation due to nonuniformity of requests to switches internal to the network.

3.1. Larger Queues at the Memory Modules

Several authors have suggested that larger queues at the memory modules could reduce the negative effects of nonuniform traffic. Stone has suggested using large queues at the memory modules in regular multistage interconnection networks (i.e. networks without feedback) [Ston87]. Although it has been shown that simply increasing memory queue sizes is not an effective method of controlling tree saturation [GGKM83, Lee85], this paper shows that this technique is significantly more useful when used in conjunction with feedback.

3.2. Damping

Because hardware costs and complexity of a global arbiter are prohibitive, some other form of controlling requests to a hot module is desirable. A better solution [ScSo90] is to allow each processor to independently limit its requests to hot and/or cold memory modules, using some method that (with a high degree of probability) will reduce tree saturation.

By using the queue status information, for example, it is possible to limit requests to the hot modules, rather than stopping them completely as would occur in a straight feedback scheme. This paper explores a simplification of the limiting scheme proposed by Scott and Sohi, who suggest that whenever a module is hot, at most 1 request to a hot module and 2 requests to a cold module be allowed to enter the network each cycle [ScSo89].

Requests are damped as follows: at each network cycle, a number of processors are selected and allowed to submit a request to the hot module, if they have one pending. Cold requests are not restricted from entering the network. We call this technique *bleeding*. The motivation for this approach stems from a desire to keep a small number of hot requests in the network at all times in order to reduce the possibility of undershoot. It also provides a more gradual release of blocked requests into the network, reducing overshoot.

4. The Simulator

In this study we used a slightly modified version of the model-driven simulator used by Scott and Sohi, thus allowing our results to be directly compared to their work [ScSo89, ScSo90].

4.1. The Simulation Model

The simulation model assumes an equal number of processors (P) and memory modules (M); in this study M and P are set to 256. The processors and memory modules are connected by a standard Omega network with 2×2 crossbar blocking switches. The number of levels in our network is $\log_2 256 = 8$, implying a total of $128 \times 8 = 1024$ switches (nodes).

Each node is capable of receiving one request from each node input per cycle, and will place that request in a FIFO queue in time for the next cycle; queues can accept up to two inputs per cycle, as in [LeK86, Lee89]. Each queue feeds the next stage of the network, and can hold up to four elements. The queue that lies between the final stage of the network and the memory modules is called the memory queue (*mq*) and has a variable length.

A queue length of four is the same length selected by several previous studies [KuPf86, PfNo85, ScSo90]. The appropriateness of a small queue size is shown by both Lee and Gottlieb, who suggest increasing the queue size beyond size five [Lee85] or size eight [GGKM83] does not help alleviate tree saturation. (However, neither study takes feedback into account.)

The model also assumes two complete interconnection networks, one to propagate requests from the processors to the memory modules, and one for the reverse trip. Each processor/memory module has a network interface which interacts with both the forward and the reverse networks and is responsible for requests being released or received. Each processor may make at most one request per network cycle, via its processor network interface (PNI), and there is no limit on the number of requests a processor may have outstanding. The memory modules are assumed to have a latency equal to one network cycle.

In order to force hot spots to exist within the network the simulator directs a user-selectable proportion of the requests to a designated memory module. This module is called the "hot" module, although it may be either hot or cold at any point during the simulation.

The model does not differentiate the percentage of time spent in hot spot versus uniform traffic conditions. It has been noted that hot spots are transient [LeK86], but the amount of time spent in different states has not been fully explored, and is not examined here. We assume a steady-state representation. Further, the model is somewhat optimistic in forcing only one hot spot at a time. It will allow more than one hot spot to arise, but this will only happen if the random distribution of uniform requests leads to an additional hot spot. In practice there may be nonuniform requests to several modules simultaneously, although Pfister and Norton suggest that typically there are only one or two memory modules hot simultaneously [PfNo85].

5. Simulation Results

The simulator was run under a variety of conditions, varying parameters such as memory queue size, feedback threshold, hot rates, degree of bleeding, etc. In discussing our results, we use the following variables, chosen to allow comparison with previous models [PfNo85, ScSo89, ScSo90]:

- N: The number of processors and memory modules.
- n: The number of levels in the network. ($n = \log_2 N$).
- P_h: The fraction of "hot" processors: those processors making nonuniform requests to a hot memory module.
- h: The percentage of nonuniformity of requests from the hot processors. h is also known as the hot rate. The uniform requests from all processors are distributed evenly among all memory modules, including the hot module.
- T_f: The threshold value for feedback. When the number of elements in a module's queue exceeds this threshold, processors are notified that that module has become hot.
- mq: The size of the queue at the last level of the network.

Each processor will attempt to offer a request during each network cycle. If the network cannot accept the request, either because the first-stage queue has no room, or because the request is to a forbidden hot spot, the processor must wait to resubmit. For each processor, effective maximum bandwidth is calculated by the following formula:

$$Maximum\ Bandwidth = \frac{Total\ Number\ of\ Requests}{TIME \times N}$$

where $TIME$ is the simulated number of network cycles elapsed, accumulated from the first cycle in which the request was generated. The maximum bandwidth of a network can only approach 1.0, even if there are no hot spots, because collisions in both the nodes and the memory modules occur even under uniform traffic [DiJu81].

In these simulations, the term "relative bandwidth" refers to the performance of the modified network under study compared to that of a standard Omega network with no modifications.

$$Relative\ Bandwidth = \frac{Bandwidth\ of\ Modified\ Network}{Bandwidth\ of\ Unmodified\ Network}$$

Therefore, if the maximum bandwidth of a modified network is .76, and that of an unmodified Omega network is .38, the relative bandwidth of the network under study is 2.0 (meaning the new method effectively doubled the bandwidth).

5.1. Effects of Increasing the Memory Module Queue Size

Figure 2 shows the results of varying the size of the queue at the memory module (the *mq*) with no feedback mechanism provided. The dashed line represents the theoretical degradation which would occur if tree saturation did not affect the bandwidth [ScSo90], i.e. if all degradation could be attributed to the fact that the hot memory module can process only one request per cycle.

As expected, increasing the size of the memory queue provides virtually no improvement in bandwidth. Without feedback, a large queue experiences the same problems that smaller queues do; the only difference is the amount of time before saturation occurs. As shown in [Lee85] and [GGKM83], employing larger memory queues without supplying some kind of feedback mechanism does not eliminate saturation conditions (although it might be useful for reducing the effects of transient hot spots).

In order to test how well feedback would work in conjunction with larger memory queues, another set of simulations was performed. Figure 3 shows the results of simulations with a hot rate of 2% and various memory queue sizes and threshold values. Data is presented for the most effective threshold values, T_f=1,2,3,4.

In Figure 3, the line of $mq = 4$ can be directly compared with Scott and Sohi's work, and is consistent with their results. From the figure we see that larger values of *mq* yielded relative bandwidths greater than those of Scott and Sohi.

As expected, the relative bandwidth is close to 1.0 for high values of P_h. In this case, a hot memory module's latency, and not tree saturation, is the bottleneck. The relative bandwidth for nearly uniform traffic (low values of P_h) never passes 1.0. This stems from the fact that if no processors are directing their requests to a particular module, the requests will be approximately uniform over the memory space. Temporary hot spots in random modules may occur, but a single module should not receive a disproportionate amount of traffic. The additional buffer space provided by a large memory queue will rarely be used, and larger queues will not substantially improve system bandwidth.

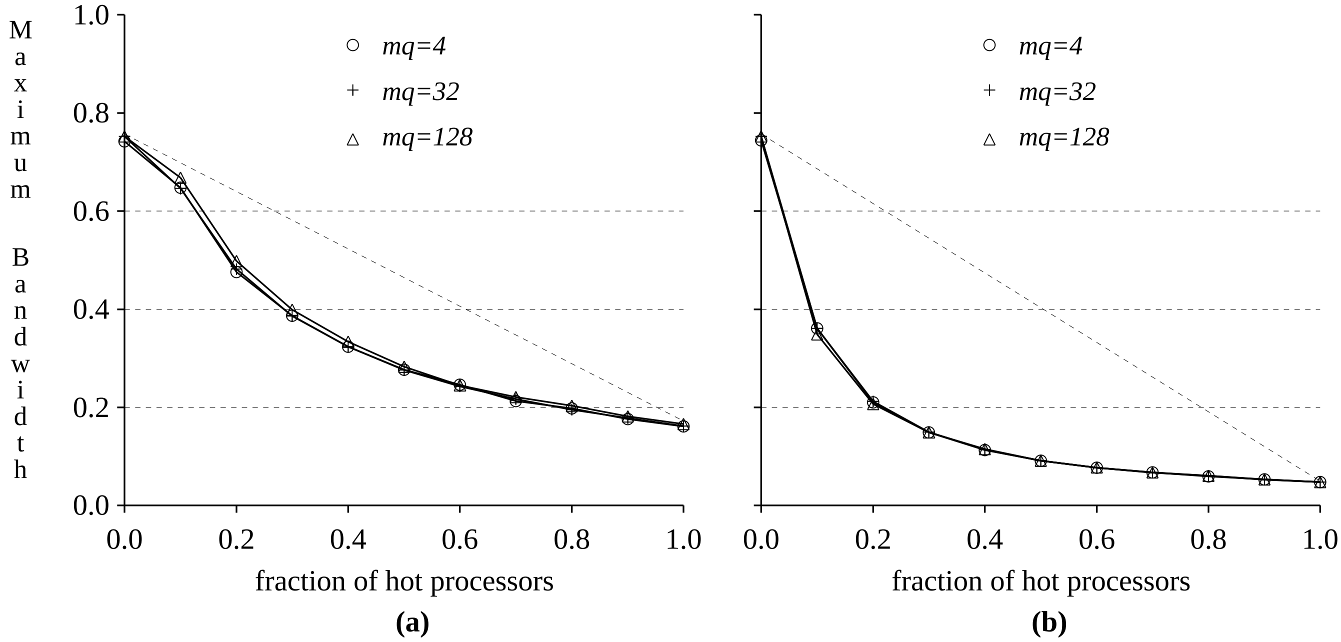

Figure 2.

Maximum Bandwidth per Processor in a Regular Omega Network
(a) h = 2%, (b) h = 8%

The lower values of T_f, 1 and 2, give relative bandwidths substantially below 1.0, evidencing undershoot; under uniform traffic conditions, lower threshold values tend to decrease network bandwidth since a random access pattern could declare a module hot when in fact it was experiencing a temporary locality of reference. In this case, requests that would not degrade the network are forbidden to enter, reducing relative bandwidth to values lower than 1.0.

Conversely, the higher the threshold, the more time will elapse before requests are forbidden to enter the network. Note that even under the most intensive nonuniform request conditions, the net gain in a module's *mq* length is only one request per cycle since the *mq* is fed by only two switches and the module processes one request per cycle. Since the onset of tree saturation is rapid, processors may be not be warned to stop issuing requests to the hot module until saturation is nearly complete. For example, if half of the 256 processors are making nonuniform requests with a hot rate of 8%, an average of ten requests are generated to the hot module every network cycle. By the time a threshold of 8 is reached, approximately 80 requests have been issued to the hot module, meaning the network has been clogged with requests to a single module. Having a large *mq* does not solve this problem entirely, since only two requests can enter an *mq* per network cycle.

Therefore, as we increase the threshold value, we also limit the bandwidth for the middle values of P_h; simulation shows higher thresholds yield lower throughput. Unfortunately, as previously described, lower thresholds give false "hot" readings which introduce more undershoot. These effects can be seen in Figure 2: T_f = 4 generates no undershoot, but has lower relative bandwidth improvements than the other threshold values. Therefore, T_f = 3 seems optimal, demonstrating little undershoot and yielding the greatest relative bandwidth improvements.

Figure 3 also shows that only small amounts of bandwidth improvement are achieved for middle values of P_h. This can be directly attributed to the low hot rate. With a hot rate of only 2%, tree saturation occurs less frequently and is less severe than with higher hot rates. In Figure 2, the distance between the actual curve and the theoretical curve (the dashed line) represents the room for improvement if all tree saturation is removed. This margin is much larger for h=8% than for h=2%.

Figure 4 shows the results of set of simulations with a higher hot rate (h = 8%). We chose to simulate this higher hot rate on the basis of current trends in multiprocessing systems. As the number of processors in a system becomes larger, it is anticipated that the severity of tree saturation caused by a given hot rate will increase, leading to decreased network performance [ScSo90]. By using the higher hot rates in smaller systems, we can estimate the performance of lower hot rates in larger systems.

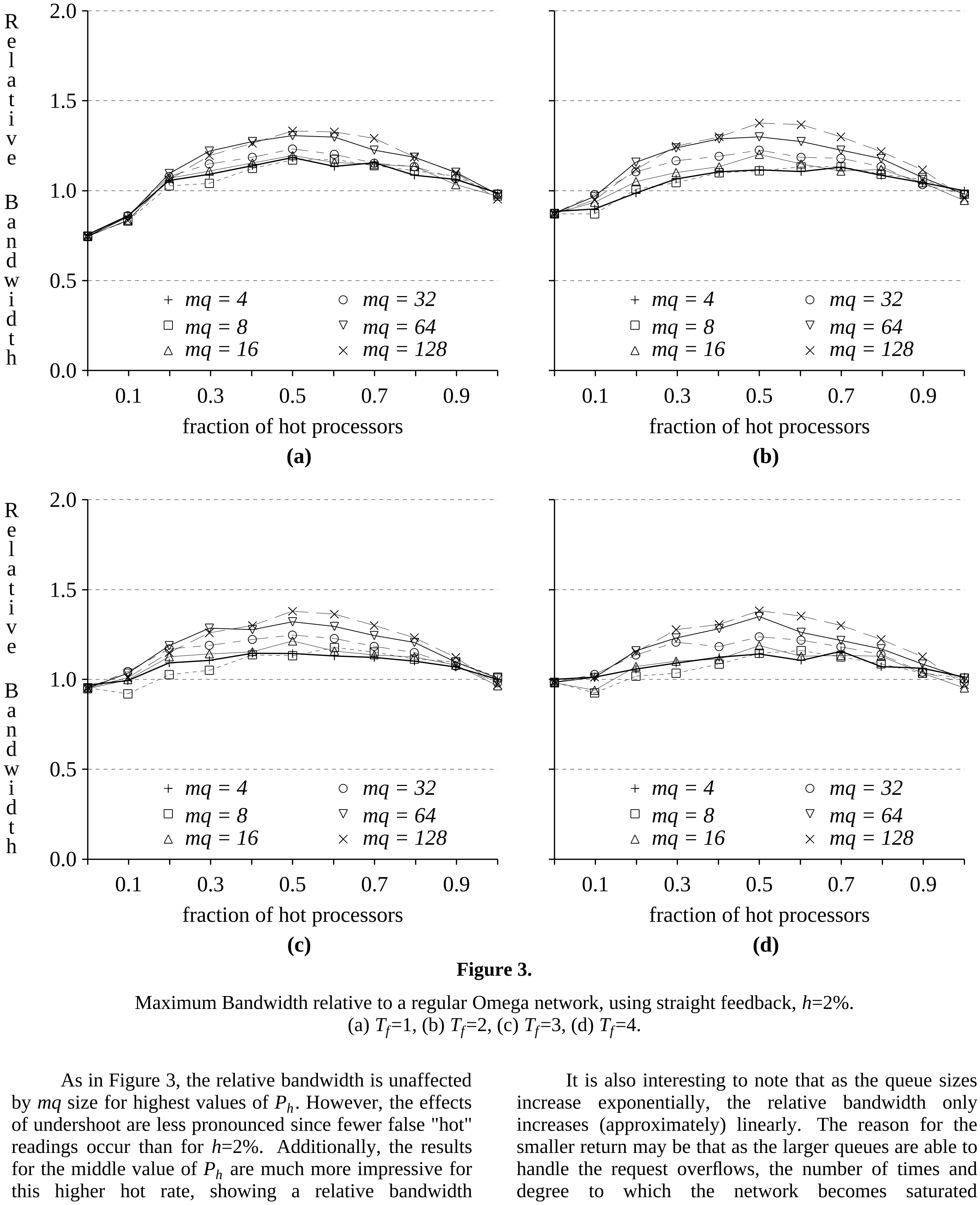

Figure 3.

Maximum Bandwidth relative to a regular Omega network, using straight feedback, h=2%.
(a) T_f=1, (b) T_f=2, (c) T_f=3, (d) T_f=4.

As in Figure 3, the relative bandwidth is unaffected by *mq* size for highest values of P_h. However, the effects of undershoot are less pronounced since fewer false "hot" readings occur than for h=2%. Additionally, the results for the middle value of P_h are much more impressive for this higher hot rate, showing a relative bandwidth increase of over 3.0.

It is also interesting to note that as the queue sizes increase exponentially, the relative bandwidth only increases (approximately) linearly. The reason for the smaller return may be that as the larger queues are able to handle the request overflows, the number of times and degree to which the network becomes saturated decreases. Since there is less saturation, the amount of improvement gained by adding additional queue elements drops. Therefore, at some point, it may become cost-ineffective to further increase the size of the *mq*. In fact, preliminary runs with larger queue sizes were performed, with no appreciable difference in results observed.

5.2. Using Bleeding in Conjunction with Feedback to Control Undershoot and Overshoot

Overshoot and undershoot can be seen as results of the bursty nature of the model when feedback is used to abruptly stop and start requests from being released into the network. A better idea is to promote a smoother flow of requests by allowing a small number of hot requests to

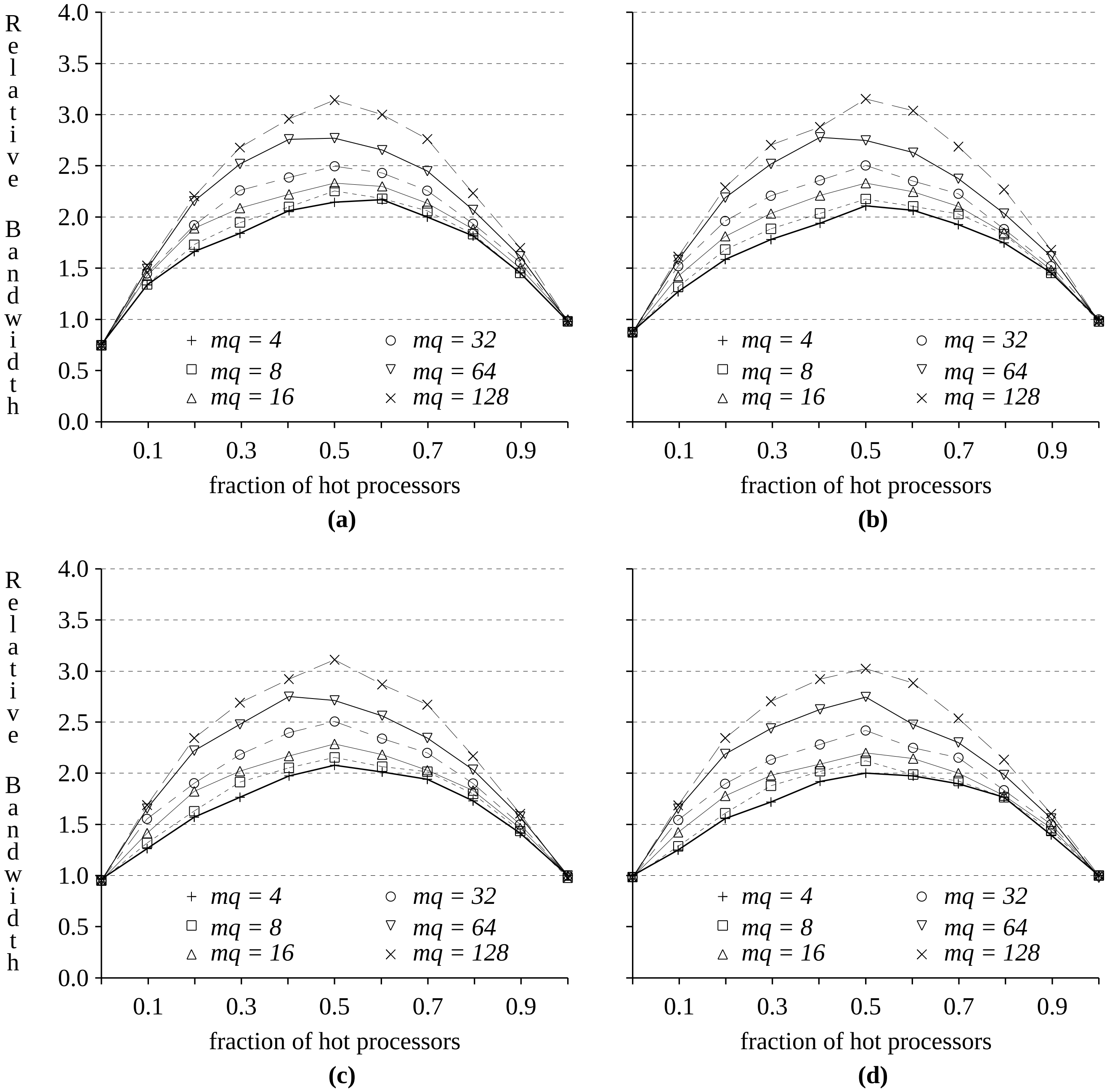

Figure 4.

Maximum Bandwidth relative to a regular Omega network, using straight feedback, h=8%.
(a) T_f=1, (b) T_f=2, (c) T_f=3, (d) T_f=4.

ignore the thresholding scheme and enter the network every cycle. This *bleeding* of requests into the network has the additional advantage that it decreases the latency of these hot requests which are permitted to enter the network during hot spot conditions.

The most promising such scheme is a round robin one in which, during each cycle, a specific number of hot processors may issue a blocked hot request. Cold requests are not limited, and may be released by any processor.

Figure 5 shows the results obtained by a bleeding scheme in which one processor per cycle is permitted to make a request to the hot module, if it has one pending. The results show a marked improvement over the straight feedback scheme. This is partially because overshoot is decreased. Although the degree of undershoot seems to be unaffected by the addition of bleeding when traffic is uniform, undershoot is lessened for the middle and higher values of P_h, contributing to the increased relative bandwidth.

Threshold size does not have the impact on performance at the middle values of P_h that it did in the straight feedback scheme. Because additional requests are in transit, the queues rarely empty to leave the memory module idle.

There is obviously an optimal number of requests to bleed to the network per network cycle. Performance plummets when the simulation allows two hot requests to enter the network per cycle, eliminating most of the beneficial effects of feedback because the network can not clear and tree saturation is enhanced. The optimal number appears to be one request per cycle, since that is the latency of the memory module in servicing requests.

6. Hardware Requirements

One of the goals of this study was to examine tree saturation limiting schemes which could be implemented easily and inexpensively (unlike combining, which has been estimated to increase hardware cost by a factor of between 6 and 32 [PfNo85], or global limiting, which not only increases hardware cost and design difficulty, but also potentially lengthens the critical path for submitting requests to the network). A brief estimate of the hardware complexity of the approaches examined in this paper are presented below.

6.1. Feedback

Measuring the size of queues to see if they have exceeded their threshold value can be implemented trivially. A bus to allow the memory/network interface to communicate with the processor/network interface (PNI) each time a module changes temperature is slightly more complicated; since multiple transitions can occur per cycle [ScSo90], the memory/network interface may have to wait for the bus. However, the bus will only require *O(logN)* wires to uniquely identify the memory module whose temperature has changed.

Finally, a buffer with a list of hot modules can be implemented at the PNIs (if the scheme allows more than one module to be designated "hot" at any given time). The size of the buffer is dependent on the number of hot spots expected to occur simultaneously.

6.2. Large Queues at Memory Modules

Increasing the size of the queues at the level of switching nodes closest to the memory modules entails minimal cost in actual components. While the memory queues will be slightly different than the other queues, the design and fabrication of this additional component should be trivial.

6.3. Damping

The bleeding scheme presented in this paper can be implemented in a straightforward manner. Each PNI keeps an internal counter which increments on each synchronous network clock pulse and runs from 0 to N-1. On each network cycle, the PNIs compare the processor id and internal counter to either permit or forbid a request from entering the network.

7. Future Work

There are a number of topics yet to be investigated. Hot spots on the return network should be examined in non-combining networks; in [LeK86], it was discovered that with combining, return networks needed larger queues because requests came out in bursts. Further, the effects of caching have yet to be examined, although Pfister and Norton suggest that caching does not help to alleviate tree saturation related to global variables (since they can not be cached) [PfNo85].

In addition, an examination of the literature reveals that the frequency of occurrence of hot spots and degree of hot rate have not been explored in detail. A study of the percentage of time spent in various hot spot conditions, for different values of P_h and h, would facilitate performance analysis of practical applications of any tree saturation mechanism. Additional research is also needed regarding the behavior of tree saturation (e.g. onset, frequency, effect on bandwidth) in larger networks.

We are currently investigating a variation of bleeding in which each processor is allowed to issue a request to a hot module with a given probability, for example $1/N$. We are also trying to ascertain the optimal *mq* size. Additionally, we are looking at an extension of feedback designed to eliminate undershoot and overshoot, which is to use different threshold values for temperature transitions.

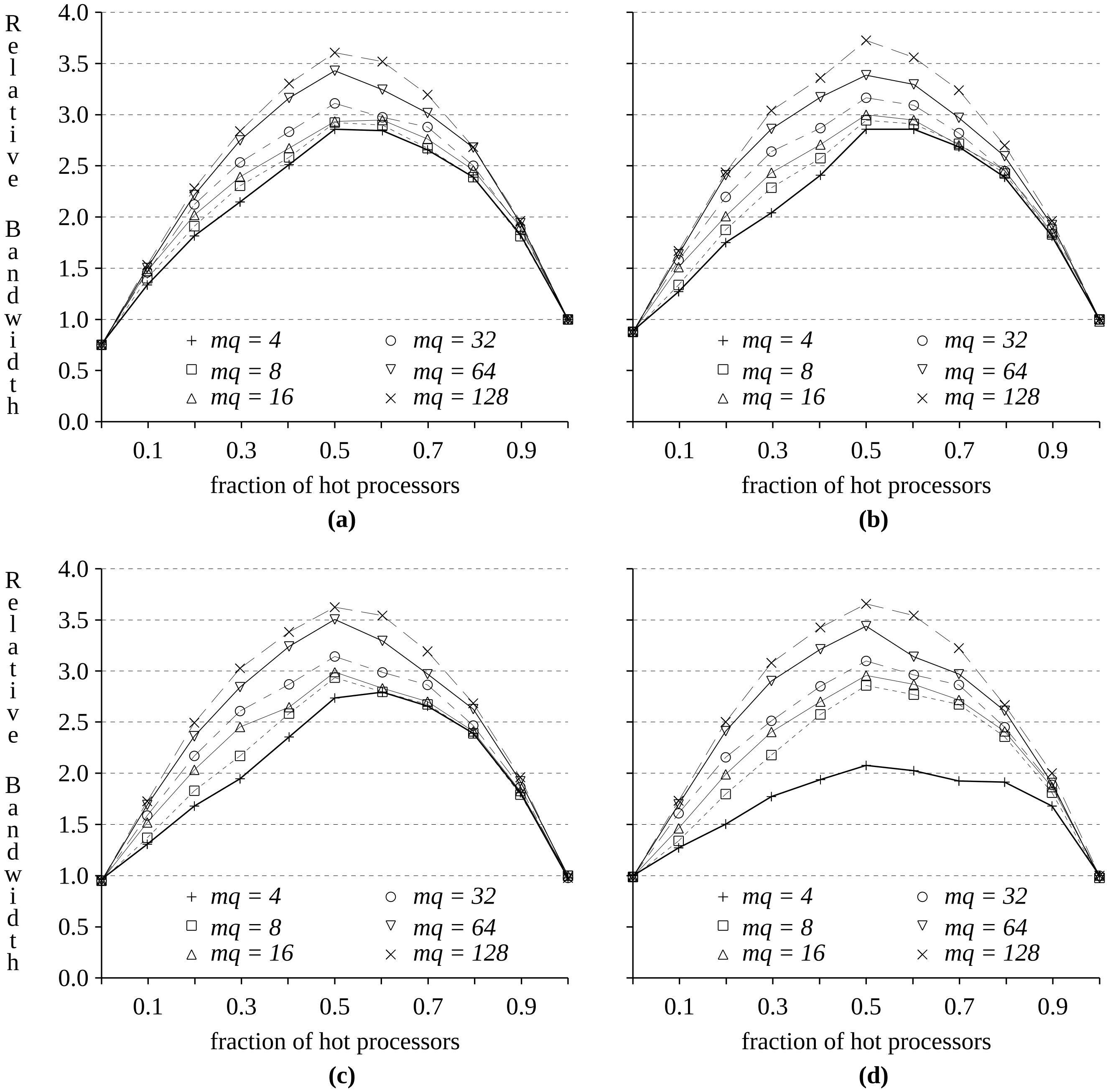

Figure 5.

Maximum Bandwidth relative to a regular Omega network, straight feedback + bleeding, h=8%.
(a) T_f=1, (b) T_f=2, (c) T_f=3, (d) T_f=4.

8. Conclusions

The techniques examined in this paper have shown that dramatic improvements in the overall bandwidth of multistage interconnection networks can be made with relatively little expense. Simulations have shown that by using a larger memory queue size in conjunction with feedback, relative bandwidth can improve by up to a factor of three. The addition of a simple bleeding scheme boosts performance even further, yielding relative bandwidth of over 3.7. This is especially significant given the minimal hardware complexity of the described methods.

9. Acknowledgements

We would like to express our profound gratitude to Steven Scott and Gurindar Sohi, for loaning us a copy of their network simulator and providing guidance while we were designing our simulations. Their help was invaluable. We would also like to thank Ron Maeder for his time and efforts.

References

[DiJu81] D. M. Dias and J. R. Jump, ''Analysis and Simulation of Buffered Delta Networks'', *IEEE Transactions on Computers,* vol. C-30:4 (April 1981), pp. 273-282.

[GGKM83] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Randolph and M. Snir, ''The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer'', *IEEE Transactions on Computers,* vol. C-34:10 (February 1983), pp. 175-189.

[KuPf86] M. Kumar and G. F. Pfister, ''The Onset of Hot Spot Contention'', *International Conference on Parallel Processing*(1986), pp. 28-34.

[LaKu90] T. Lang and L. Kurisaki, ''Nonuniform Traffic Spots (NUTS) in Multistage Interconnection Networks'', *Journal of Parallel and Distributed Computing*(1990), pp. 55-67.

[Lee85] R. Lee, ''On 'Hot Spot' Contention'', *ACM Computer Architecture News,* vol. 13:5 (1985), pp. 15-20.

[LeK86] G. Lee, C. P. Kruskal and D. J. Kuck, ''The Effectiveness of Combining in Shared Memory Parallel Computers in the Presence of 'Hot Spots''', *International Conference on Parallel Processing*(1986), pp. 35-41.

[Lee89] G. Lee, ''A Performance Bound of Multistage Combining Networks'', *IEEE Transactions on Computers,* vol. C-38:10 (October 1989), pp. 1387-1395.

[NoPf85] A. Norton and G. F. Pfister, ''A Methodology for Predicting Multiprocessor Performance'', *International Conference on Parallel Processing*(1985), pp. 772-881.

[PfNo85] G. F. Pfister and V. A. Norton, '''Hot Spot' Contention and Combining in Multistage Interconnection Networks'', *IEEE Transactions on Computers,* vol. C-34:10 (October 1985), pp. 943-948.

[PoHa89] A. Pombortis and C. Halatsis, ''Behaviour of Circuit-Switched Multistage Networks in Presence of Memory Hot Spot'', *Electronic Letters,* vol. 25:13 (June, 1989), pp. 833-834.

[ScSo89] S. L. Scott and G. S. Sohi, ''Using Feedback to Control Tree Saturation in Multistage Interconnection Networks'', *Proceedings of the 15th Annual Symposium on Computer Architecture*(1989), pp. 167-176.

[ScSo90] S. L. Scott and G. S. Sohi, ''The Use of Feedback in Multiprocessors and Its Application to Tree Saturation Control'', *IEEE Transactions on Parallel and Distributed Systems,* vol. 1:4 (October 1990), pp. 385-399.

[Sieg85] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, Lexington Books, Lexington, MA, (1985).

[Ston87] H. S. Stone, *High-Performance Computer Architecture*, Addison-Wesley Publishing Company, Reading, MA, (1987).

[TaYe90] P. Tang and P. Yew, ''Software Combining Algorithms for Distributing Hot-Spot Addressing'', *Journal of Parallel and Distributed Computing,* vol. 10 (1990), pp. 130-139.

[Thom86] R. H. Thomas, ''Behavior of the Butterfly Parallel Processor in the Presence of Memory Hot Spots'', *International Conference on Parallel Processing*(1986), pp. 46-50.

[YeT87] P. Yew, N. Tzeng and D. H. Lawrie, ''Distributing Hot-Spot Addressing in Large-Scale Multiprocessors'', *IEEE Transactions on Computers,* vol. C-36:4 (April 1987), pp. 388-395.