USING FEEDBACK TO CONTROL TREE SATURATION IN MULTISTAGE INTERCONNECTION NETWORKS

by

Steven Scott and Gurindar Sohi

Computer Sciences Technical Report #807

November 1988

Using Feedback to Control Tree Saturation In Multistage Interconnection Networks*

Steven Scott and Gurindar Sohi

Computer Sciences Department
University of Wisconsin-Madison
1210 W. Dayton Street
Madison, WI 53706

Abstract

In this paper, we propose the use of feedback schemes in multiprocessors that use an interconnection network with distributed routing control. We show that by altering system behavior so as to minimize the occurrence of a performance-degrading situation, the overall performance of the system can be improved.

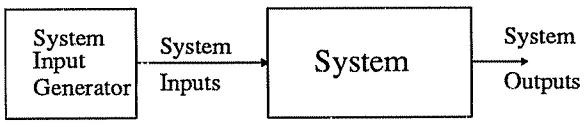
As an example, we have considered the problem of tree saturation caused by hot spots in multistage interconnection networks. Tree saturation causes degradation to all processors in a system, including those not participating in the hot spot activity. We see that feedback schemes can be used to control tree saturation, reducing degradation to other memory requests and thereby increasing total system bandwidth. As a companion to feedback schemes, damping schemes are also considered. Simulation studies presented in this paper show that feedback schemes can improve overall system performance significantly in many cases.

* This work was supported by National Science Foundation Grant CCR-8706722

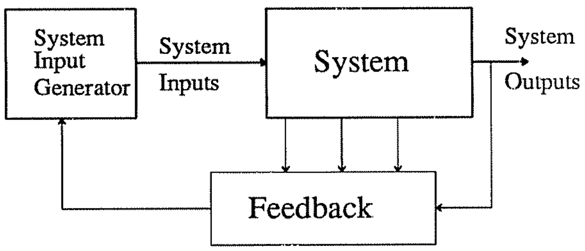
1. INTRODUCTION

One of the most important and widely used concepts in the design of engineering control systems is the concept of *feedback* [2]. Feedback is primarily used to: (i) prevent instability in a system and (ii) prevent the system from settling down into a stable but undesirable situation. Figure 1 illustrates how feedback works. Without feedback (Figure 1(a)), the inputs of the system are independent of events that might be occurring in the system. Consequently, an unstable or an undesirable situation could arise. With feedback (Figure 1(b)), the outputs of the system (and possibly other state values of the system) are fed back to the inputs. Based upon the feedback information, the system input generator attempts to modify the system inputs to prevent the occurrence of an unstable or an undesirable situation.

Modern computing systems have evolved into large-scale parallel processors that consist of possibly hundreds of processors and memory modules interconnected together in some fashion. Figure 2 illustrates a typical processing system based on a shared-memory programming paradigm [13]. The processing system consists of a set of processing elements, a set of memory modules and an interconnection network. The interconnection network is logically broken into a forward network and a reverse network though it is possible that the two networks could be the same physical network (for example a set of



(a) Open-Loop System



(b) Closed-Loop (Feedback) System

Figure 1: An Engineering Control System

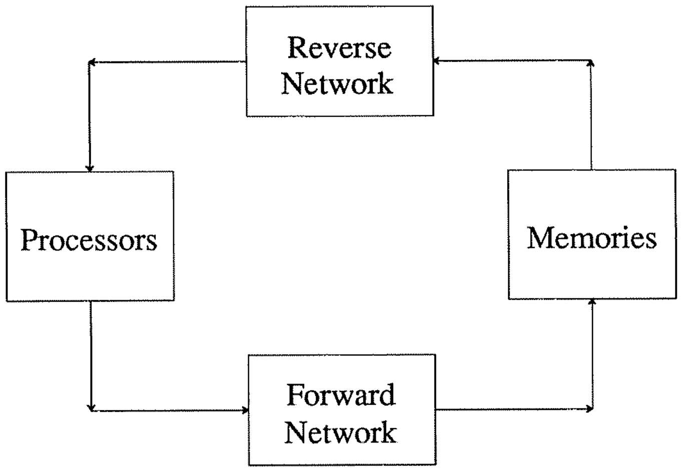


Figure 2: A Typical Shared-Memory Multiprocessing System

buses). It is important to realize that the overall performance of such a processing system is not determined solely by the performance of the individual components; it is affected by how the components interact when they are connected together.

Let us compare Figures 1 and 2. If we assume that the forward interconnection network is the system, then the inputs to the system are the requests generated by the processors and the outputs of the system are the inputs to the memory modules. An example of an undesirable situation in such a system is blockage or congestion which unnecessarily reduces the effective bandwidth of the interconnection network. Considering the resemblance between figures 1 and 2, we ask ourselves: (i) why has explicit feedback not been used thus far in the design of computing systems and (ii) why might it be useful now?

Traditionally, a computing system consisted of a single processor (system input generator). In a single processor system, the control mechanism of the processor has sufficient information about the state of the overall system (processor, network and memory) to prevent the occurrence of an undesirable situation. This is because the single processor is generally the only entity that is generating requests which can alter the state of the system. Moreover, there exists some implicit feedback in the responses to memory requests. The rate at which memory requests are entered into the network is directly influenced by the rate at which responses are received. In a multiprocessor system, however, many processors are generating requests without knowledge of the state of other components in the system. In such a processing system, it is possible that the collective input of the processors could interact in such a way as to cause undesirable degradation of the network. The implicit feedback (via the reverse network) to individual processors cannot generally convey enough information to correct the anomalous behavior. Thus, explicit feedback mechanisms may be warranted.

One could alter the processing system of Figure 2 to resemble the system of Figure 1(b) by providing an explicit feedback from points in the system to the system input generators (see Figure 3). This explicit feedback could then be used to detect potential undesirable situations and instruct the processors to modify their inputs to the network such that they do not contribute to the undesirable situation. If an undesirable situation is prevented, the overall performance of the system could be enhanced.

In this paper, we target one particular undesirable situation in a parallel computer system that uses multistage interconnection networks -- the problem of *tree saturation*. We demonstrate how feedback concepts can be used to instruct the processors to modify their requests to the interconnection network so

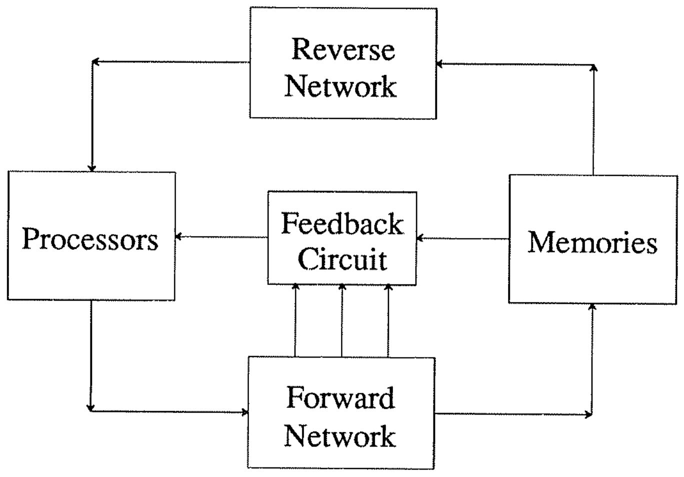


Figure 3: A Shared-Memory Multiprocessing System With Feedback

that the problem is alleviated.

The outline of this paper is as follows. In section 2, we consider the undesirable situation of tree saturation in multistage interconnection networks that use a distributed routing control. We see that, if tree saturation could be controlled, the overall bandwidth of the network (and consequently the throughput of the multiprocessor) could be improved in many cases. In section 3, we propose schemes for controlling tree saturation. In section 4, we present the results of a simulation analysis carried out to test the effectiveness of the tree-saturation-controlling mechanisms. In section 5, we present a discussion of the feedback concept in light of the results of section 4, and in section 6 we present concluding remarks.

2. TREE SATURATION IN MULTISTAGE INTERCONNECTION NETWORKS

A popular interconnection network for medium to large scale multiprocessors is a blocking, $O(NlogN)$ multistage interconnection network (ICN) with distributed routing control. An example of such a ICN is the Omega network [8]. An Omega network consists of $logN$ levels of switching elements (or switches). Messages enter the network at the inputs to the first stage and proceed to the outputs of the last stage, one level at a time. Routing decisions are made local to each switch. Since there is no global control mechanism, the state of any particular switch is unknown to other entities (processors, memories, other switches) in the multiprocessor and a particular input request pattern to the network might cause an undesirable situation.

The problem of *tree saturation* is a classic example of what we consider to be an undesirable situation in a multiprocessor system. This problem was first observed by Pfister and Norton in conjunction with requests to a *hot spot* [11]. In their analysis, the hot spot was caused by accesses to a shared lock variable. When the average request rate to a *hot memory module* exceeds the rate at which the module services the requests, the requests will back up in the switch which connects to the hot memory module. When the output queue in this switch is full, it will back up the queues in the switches that feed it. In turn, those switches will back up the switches feeding them. Eventually, a tree of saturated switches results. Depending upon the number of outstanding requests and the reference patterns of the various processors, this tree of saturated queues may extend all the way back to every processor. Any request which must pass through any of the switches in the saturated tree, whether to a hot module or not, must wait until the saturated queues are drained. Thus, even requests whose destinations are idle will be blocked for potentially long periods of times, thereby leading to unnecessary degradation of the network bandwidth.

Since the problem of tree saturation (caused by hot spot activity) can be catastrophic to the performance of systems such as the NYU Ultracomputer [3], Cedar[5] and the IBM RP3 [12], considerable effort has been devoted to studying the problem and suggesting solutions to it [3, 9, 11, 15].

When the problem is caused by accesses to synchronization variables (or more generally, by accesses to the same memory location), *combining* can be used. *Hardware combining* uses special hardware in the network switches to combine requests destined to the same memory location. On the return trip, the response for the combined request is broken up into responses for the individual requests. It is estimated in[11] that using combining hardware would increase the cost of a multistage interconnection network by a factor of 6 to 32. *Software combining*[15] uses a tree of variables to effectively spread out access to a single, heavily-accessed variable. It is only applicable to known hot spot locations such as variables used for locking or barrier synchronization.

Since the overall bandwidth of the network is determined by the number (or equivalently the rate) of requests that have to be serviced by the hot module, combining can improve overall network bandwidth by reducing the number of requests that have to be serviced by the memory module. Combining also improves the latency of memory requests that do not access the hot memory module since it alleviates tree saturation. Unfortunately, combining cannot alleviate the bandwidth degradation or the tree saturation if the hot requests are to different memory locations in the same memory module, that is, the entire memory module is hot. Such a situation could arise from a larger percentage of shared variables residing in a particular module, stride accesses that result in the non-uniform access of the memory modules or temporal swings where variables stored in a particular module are accessed more heavily. In these cases, one module will receive more requests than its uniform share, just as if it contained a single hot variable. Recognizing this problem, the RP3 researchers have suggested scrambling the memory to distribute memory locations randomly across the memory modules [1, 10]. With a scrambled distribution, it is hoped that non-uniformities will occur less often, though we are unaware of any hard data to support this fact.

Even though processor requests may be distributed uniformly amongst the memory modules, tree saturation can still occur if any of the switches in the network has a higher load (in the short term) than other switches at the same level[7]. Alternate queue designs may improve the latency of memory requests that do not access the hot module [14], but eventually tree saturation will occur even with alternate queue designs [14].

To alleviate the problem of tree saturation in general, we need a mechanism that detects the possibility of tree saturation and instructs the processors to hold requests that might contribute to the tree saturation. If many of the problem-causing requests can be held outside the network (for example, in the processor queues), the severity of the problem can be reduced.

Before proceeding further, let us convince the reader that alleviating the undesirable situation caused by tree saturation can indeed result in an increase in the overall performance of the system. We shall only consider hot requests that cannot be combined in this paper since no solution to the problem of tree saturation is known in this case. We shall also restrict ourselves to $N \times N$ Omega networks.

2.1. Bandwidth Degradation Due to Tree Saturation

Consider a processing situation in which a fraction f of the processors are making requests to a hot module with a hot probability of h on top of a background of uniform requests to all memory modules, and the remaining processors are making only uniform requests. Processors may have multiple outstanding requests. This is a likely scenario if more than one job is run on the multiprocessor. Let r_1 be the rate at which the processors with hot requests can generate requests and let r_2 be the rate at which processors generating uniform requests can generate requests. The number of requests per cycle that appear at the hot module is therefore:

$$R_{hot} = f r_1 (hN + (1-h)) + (1-f) r_2 \tag{1}$$

Since the hot module can only service one request in each memory cycle, the maximum value of R_{hot} is one. Equating the right hand side of equation (1) to 1 and rearranging terms we get:

$$r_1 = \frac{1-(1-f)r_2}{f(1+h(N-1))} \tag{2}$$

To calculate the overall bandwidth of the network, we observe that fN processors have a throughput of r_1 and $(1-f)N$ processors have a throughput of r_2. Therefore, the maximum bandwidth per processor is:

$$BW = \frac{r_1 fN + r_2(1-f)N}{N} = \frac{1+(1-f)h(N-1)r_2}{1+h(N-1)} \tag{3}$$

If the $(1-f)$ processors that are generating uniform requests can do so without any interference from the fN processors that are generating hot requests, then we can calculate an upper bound for the system bandwidth by setting r_2 to 1. This yields an average cutoff bandwidth per processor of:

$$BW_{cut} = \frac{1 + (1-f)h(N-1)}{1 + h(N-1)} \tag{4}$$

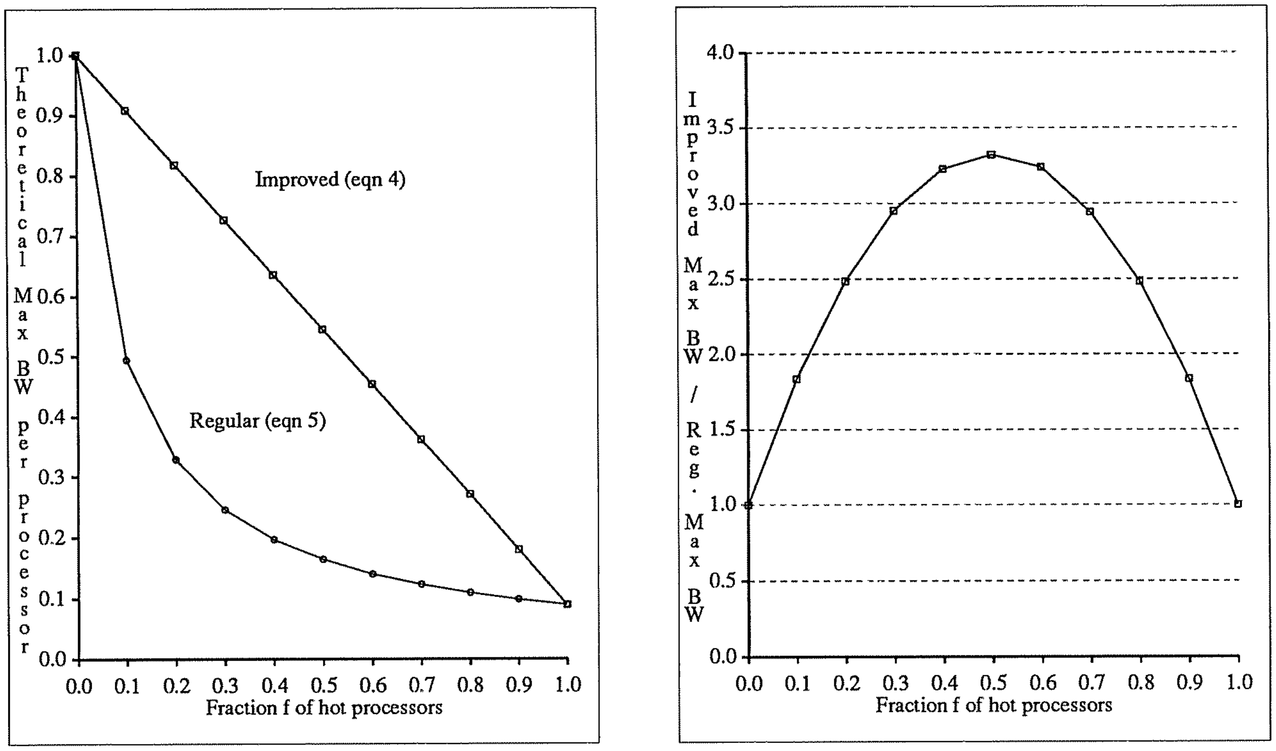
with r_1 limited to

$$r_1 = \frac{1-(1-f)r_2}{f(1+h(N-1))} = \frac{1}{1+h(N-1)}$$

However, tree saturation normally prevents the uniform requests from proceeding without interference. When tree saturation is present, all requests in the system can become blocked, and the rate at which the $(1-f)N$ processors generating uniform requests is limited by the rate at which the fN processors are generating hot requests. Based upon experiments that we have carried out, it is observed that when this occurs, the throughput of the system appears as if all N processors had a smaller hot spot of fh rather than fN processors having a hot spot of h and $(1-f)N$ processors having no hot spot. In this case, the average cutoff bandwidth per processor is:

$$BW_{cut} = \frac{1}{1+hf(N-1)} \tag{5}$$

Figure 4 plots the bandwidths suggested by equations (4) and (5) as a function of f, for $h = 4\%$, and $N = 256$. As can be seen from the figure, it appears that the overall bandwidth of the network can be improved significantly if the problem of tree saturation is alleviated, allowing the processors generating uniform requests to access the network without interference. The bandwidth improvement is zero at the



(a) Absolute

(b) Relative (Improved / Regular)

Figure 4: Theoretical Limiting Maximum Bandwidth vs f With h=4%, N=256.

endpoints, and largest for values of f near 1/2. Our experimental results in section 4 will confirm this.

3. CONTROLLING TREE SATURATION

As mentioned earlier, tree saturation occurs if the rate at which the processors are generating requests to a hot module is greater than the rate at which the memory module can service them. Once requests to a hot module enter the network, they block within the network and eventually lead to tree saturation. When tree saturation is present, even processors that do not participate in the hot spot activity are penalized.

To alleviate the problem of tree saturation, requests that would compound the problem must be prevented from entering the network until the problem has subsided. Ideally, requests to the hot module must be made to wait outside the network (at the processor interface) until the hot module is ready (or slightly before it is ready) to service them, and then proceed at a rate at which they can be serviced by the hot module. Now, we present two schemes that try to achieve this goal.

The two schemes are *limiting* and *feedback*. Let us discuss each of these schemes and their implementation in some more detail.

3.1. Limiting the Number of Requests

One way of preventing the problem of tree saturation is to limit the number of requests to each memory module that enter the network each cycle to the number of requests that the memory module can service in one cycle. Requests that cannot enter the network in a particular cycle must be blocked until a later cycle. Unfortunately, in its complete generality, limiting suffers from two problems.

The first problem is that limiting may unnecessarily constrain the available bandwidth of the system when no hot spot exists (as we shall see in section 4). With limiting, multiple requests to the same memory module will not be allowed to proceed in the same cycle, thereby delaying each processor whose request was held back. When no hot spots exist, this delay is unnecessary, as multiple requests could enter the network in the same cycle (thereby allowing their processors to continue) and proceed to the final stage of the network without hindering requests to other memory modules. In the final stage, they can queue up and be serviced one at a time.

The second problem is the cost of implementing a full-blown limiting scheme. To implement limiting from all N processors to all N memory modules requires a global arbiter that is capable of performing N arbitrations every cycle (one for each memory module) where each arbitration has an input from each of the N processors. The hardware costs of doing so can be prohibitive. Furthermore, limiting is not scalable. As an alternative to full-blown limiting, limiting could be restricted to a single hot spot. This would require a single arbiter to control access to the module currently identified as the hot module. While still expensive, this scheme is much more practical than full limiting. We will discuss this more in section 5.

3.2. Use of Feedback

Figure 3 presents a multiprocessing system with feedback. Select state information is tapped from the ICN and the memory modules and fed back to the processors. The processors respond by holding back problem-aggravating requests.

The feedback scheme that we use in this paper is very simple. The only state information that we monitor is the size of a queue at the input to each memory module (or output of the network). If the size of the queue exceeds a certain threshold T, we assume that the module is hot and notify the processors. The processors respond by holding back requests to the hot modules. When the size of the queue falls below the threshold T, the module is considered to be cold and the processors can again submit requests to it.

This feedback scheme prevents a module from causing full tree saturation, because as soon as the module becomes hot, requests to that module are stopped from entering the network. However, there are some problems. First, there is a finite delay between the time when requests enter the network and when they reach their destination memory module and trigger the feedback to the processors (if need be). At the instant that a module becomes hot, there may be many requests for that module already in transit. These requests may temporarily cause some tree saturation and congest the network. However, the resulting tree saturation will not be as severe as the tree saturation caused when requests can enter the network arbitrarily. It has been estimated in [6] that the onset of full tree saturation occurs as quickly as several network traversal times (the time for a packet to traverse the network in one direction, equivalent to the depth of the network). The feedback scheme outlined above allows only a single network traversal time before stopping requests to a hot module.

A second problem is that if the threshold value is less than the number of levels in the network, a hot module may become become cold and service all its queued requests before any newly released requests arrive, thus laying idle for some number of cycles. A final problem is that when a hot module becomes cold, many hot requests blocked in the processors may be released simultaneously, leading to overflow at the memory module queue when the requests arrive. These problems are very similar to overshoot, undershoot and oscillation in engineering control systems with feedback [2].

To reduce overshoot, undershoot and oscillation, some form of *damping* may be introduced [2]. The damping must allow a systematic release of requests to the hot module into the network. This is precisely what a limiting scheme accomplishes. Limiting could be used to dampen a feedback scheme as follows. When a module is hot, only one request to that module is allowed to enter per cycle. Up to two requests for every cold module are allowed to enter the network each cycle. Allowing one request per cycle to a hot module prevents the module's queue from becoming empty. Allowing only two requests per cycle for each cold module prevents queues from overflowing quickly, keeping any temporary tree saturation to a minimum.

In this paper, we have limited our simulations of feedback to straight feedback and feedback with the limiting-damping discussed above. As will be seen, this strong form of damping is highly effective. In section 5, we discuss several other aspects of feedback system design which may be used to improve upon simple feedback at a more reasonable cost.

The hardware complexity of implementing a feedback scheme is minimal. To implement the scheme that we have described (without damping), all that we need to do is to monitor the size of the queue at each memory module and notify the processors if it exceeds a threshold. Doing so requires only a single wire (signal) per memory module, that is, a total of N signals that the processors must monitor. Processors decode the destinations of their requests, and only issue a request when its destination is cold. If we further assume that only a single module is hot at any given time instant, we can convey the same information (hot module number) with only $logN$ wires. Clearly, this is a small overhead compared to the complexity of the ICN.

4. SIMULATION MODEL AND RESULTS

4.1. Network Model

For all our experiments we considered an N×N Omega network connecting N processors to N memory modules. A forward network carries requests from the processors to the memories and a reverse network is used for responses from the memory modules to the processors.

A 2×2 crossbar switching element (shown in Figure 5) is used as the basic building block. The size of the queue at each output is Q requests and each queue can accept a request from both inputs simultaneously if it has room for the requests. The order that multiple inputs are gated to the same output is chosen randomly.

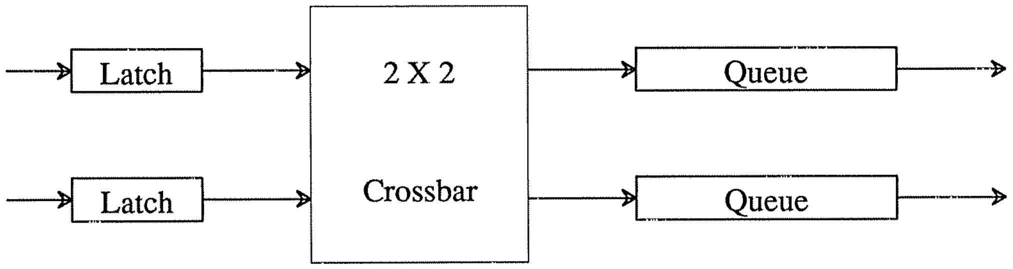


Figure 5: Switch Used in Simulation and Model

Requests move from one stage of the network to the next in a single network cycle. Each memory module can accept a single request every network cycle and the latency of each memory module is one network cycle. Therefore, the best-case round trip time for a processor request is $2\log_2 N + 2$ network cycles [issue (1) + forward network hops ($\log_2 N$) + memory module service (1) + reverse network hops ($\log_2 N$)].

In each network cycle, a processor makes a request with a probability of r. A fraction f of the processors make a fraction h of their requests to a hot memory module and the remaining $(1-h)$ of their requests are distributed uniformly over all memory modules. The remaining fraction $(1-f)$ of the processors make uniform requests over all memory modules.

4.2. Simulation Results

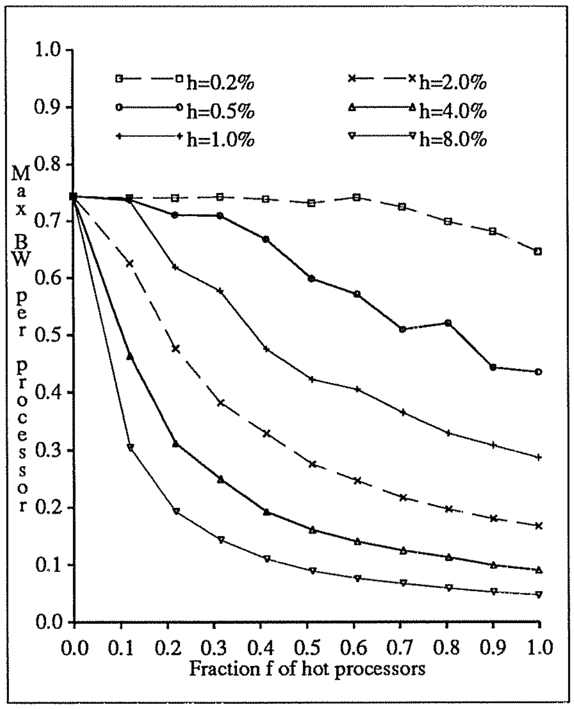
The results presented in this section are for 256×256 (N=256) Omega networks with queue sizes of 4 elements (Q=4) at each switching element output. We also simulated 64×64, 128×128 and 512×512 Omega networks, each with varying queue sizes and memory latencies. The results follow a similar trend to the results we report for 256×256 networks with queue sizes of 4 and memory latencies of 1, though the magnitude of the results are different. For reasons of brevity, we shall not present those results in this paper.

Four varieties of networks were simulated:

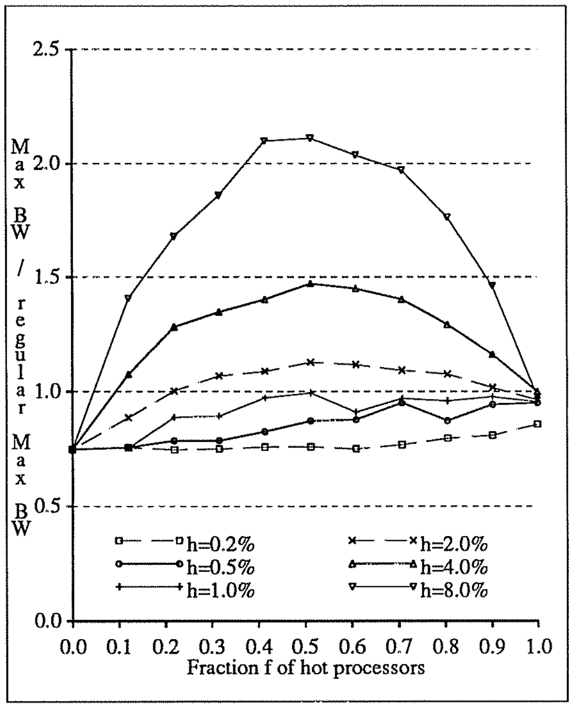
- Regular omega networks.
- Networks with feedback (T =1,2,3, and 4 queue elements).
- Networks with straight limiting (one request per module per cycle).
- Networks with feedback (T=1), plus limiting-damping.

In Figure 6, we consider the saturation bandwidth of the network in different situations. Figure 6(a) plots the saturation bandwidth per processor for a regular network (without feedback) as f varies from 0 to 1. Various hot rates h are considered. From Figure 6(a) we see that as the fraction of processors making hot requests increases, the overall system bandwidth decreases. The higher the hot rate, h, the faster the bandwidth drops off. When all processors are making hot requests, the bandwidth is severely affected by the hot rate.

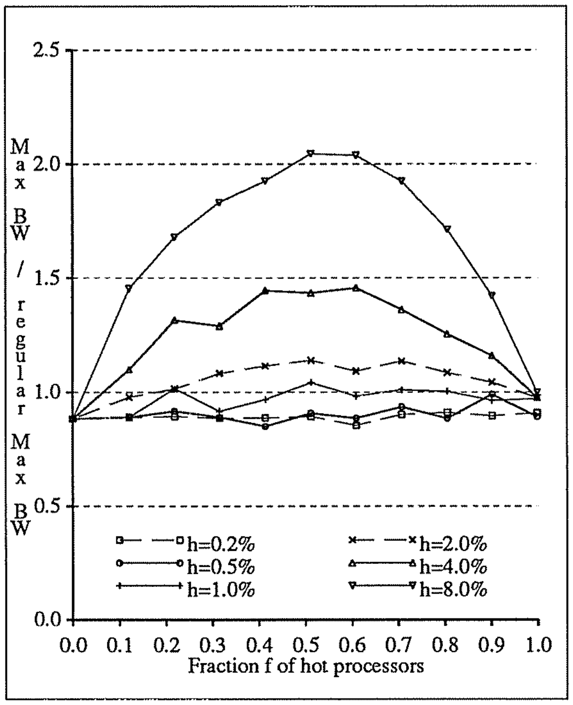
The purpose of feedback schemes and limiting schemes is to control tree saturation and consequently improve overall network bandwidth. At the end points of each curve in Figure 6(a), i.e., f =0 and f=1, feedback is of little use in improving the bandwidth (but as we shall see, it can still improve memory latency). This is because when all processors are making uniform requests (f =0), little tree saturation occurs, and when all processors are making hot requests (f =1), the bandwidth is limited by the rate at which the hot module can service requests and not by the tree saturation that is present. Overall bandwidth of the network can be improved by controlling tree saturation only when the tree saturation is



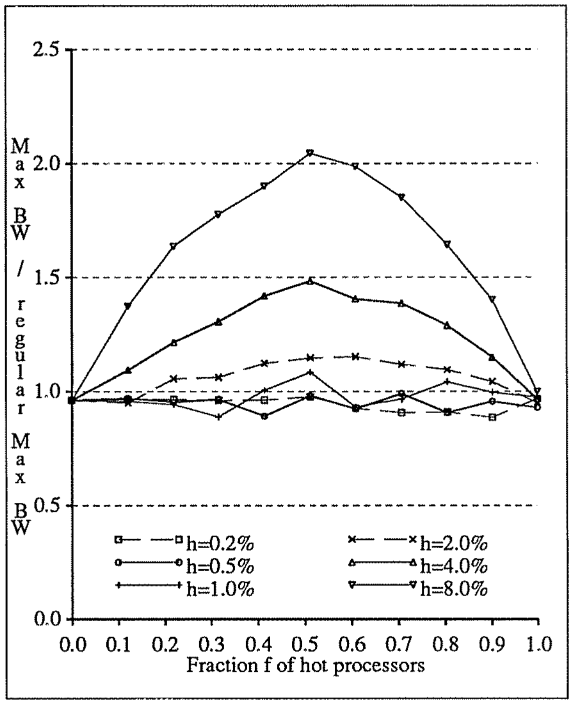
(a) Regular Network



(b) Feedback - Threshold 1 (Relative to regular network)

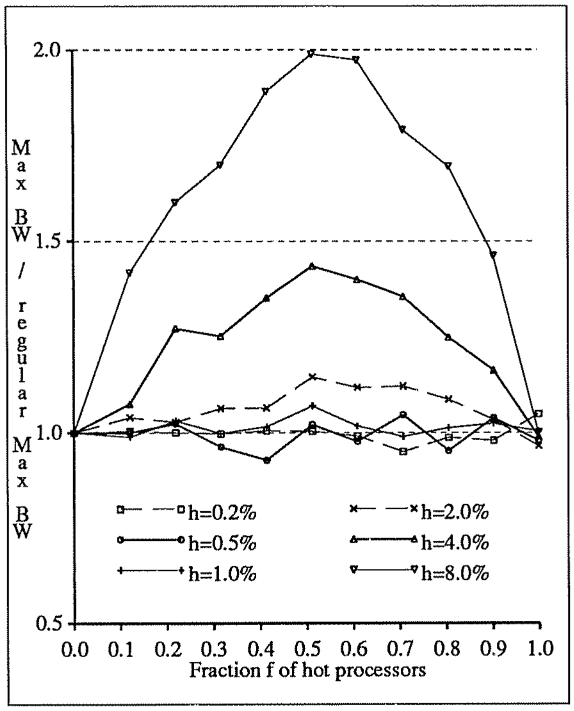


(c) Feedback - Threshold 2 (Relative)

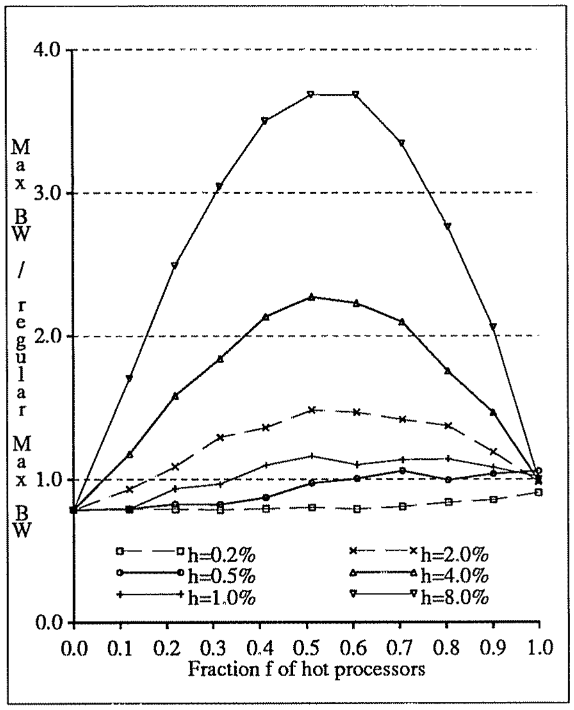


(d) Feedback - Threshold 3 (Relative)

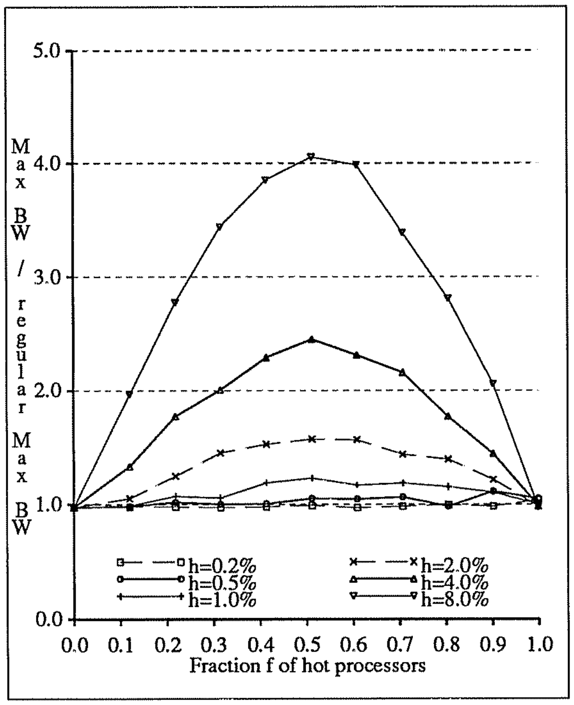
(All networks are 256x256 Omega networks, using 2x2 crossbar switches with 4-element queues on the switch outputs)



(e) Feedback - Threshold 4 (Relative)



(f) Limiting (Relative)



(g) Feedback + Limiting-damping

(All networks are 256x256 Omega networks, using 2x2 crossbar switches with 4-element queues on the switch outputs)

Figure 6: Maximum Bandwidth vs f (fraction of processors making hot requests)

actually limiting the bandwidth, i.e., f is between 0 and 1.

Now we consider the use of our feedback schemes of Section 3.2. Figures 6(b), 6(c), 6(d) and 6(e) plot the relative bandwidths for networks with feedback thresholds of 1, 2, 3, and 4, respectively. The relative bandwidth is the bandwidth of the modified network divided by the bandwidth of a regular network with no feedback or limiting. Note that these figures qualitatively confirm the results that were predicted in section 2. When f lies between 0 and 1, the use of feedback alleviates the tree saturation caused by the hot requests, allowing the processors making uniform requests to proceed with less interference, and increasing overall system bandwidth. The actual magnitude of the improvement is less than possible, due to the fact that tree saturation has not been eliminated, but rather just alleviated.

We can also make two additional observations concerning threshold values for feedback. The first observation is that under high hot rates, lower thresholds give more improvement than higher thresholds. This is due to the fact that lower thresholds prevent hot traffic from entering the network sooner, and thus have less temporary hot module queue overflow. With a threshold of 1, a hot module's queue can accept 3 more requests at the time it becomes hot, without overflowing and causing tree saturation. With a threshold of 4, the queue is already full by the time it becomes hot and further requests to the module that are already in the network will cause partial tree saturation.

The second observation is that using thresholds that are too small can limit bandwidth to less than the bandwidth of a regular network. The smaller the threshold, the more likely a request is to be blocked at the entrance to the network even though the destination memory module of the request is receiving an average of less than one request per cycle. Networks with larger thresholds are less likely to unnecessarily restrict bandwidth due to temporal fluctuations in the traffic pattern. Another reason that smaller thresholds restrict bandwidth is that they allow the hot module's queue to become empty for longer periods of time (as discussed in section 3.2). Under high hot rates, these problems are offset by the smaller threshold's ability to better control tree saturation.

On closer look at Figure 6(e), we see that even with a high feedback threshold T of 4, the bandwidth is sometimes slightly less than the bandwidth of a regular network. This can be attributed to the problem of the hot module's queue occasionally becoming idle for a few cycles. When $f = 0$ (no hot spots) the relative bandwidth is unity. This indicates that normal traffic is not being restricted. If the queue sizes permitted a threshold equal to the number of levels in the network, then the problem of hot modules' queues becoming idle could be eliminated. We have simulated larger queue sizes and thresholds and found this to be the case.

The higher the hot rate, the more the overall network bandwidth is improved by using feedback. With a hot rate of 4 or 8%, significant increases in system bandwidth occur even with a small percentage of processors making hot requests. As systems become larger, the tree saturation caused by a given hot rate will become more severe, and the hot rate needed to cause a given level of tree saturation will decrease. In such cases, the need for feedback is even more compelling.

Now let us examine the results of using limiting (Figure 6(f)) and feedback with limiting-damping (Figure 6(g)). From the figures, we see that both techniques are quite effective when the hot rate is high. Tree saturation is not allowed to develop and processors making only uniform requests see much less interference from processors making hot requests. For example, with 50% of the processors making requests with a hot rate of 8%, system bandwidth is improved by a factor of 4. It is worth noting here, that since the feedback and limiting are not improving the bandwidth of the processors making hot requests (they cannot, since the bandwidth of the processors making hot requests is being limited by the number of requests to the hot module), and since the *average* bandwidth is increased by a factor of 4, then the bandwidth of the processors making uniform requests is actually being increased by a factor of 8.

For low hot rates, straight limiting is overly conservative and unnecessarily restricts bandwidth, as pointed out in Section 3.1. For example, when $f = 0$ in Figure 6(f), the relative bandwidth of the network is somewhat less than 1. Feedback with limiting-damping (Figure 6(g)) does not unnecessarily restrict bandwidth since the limiting mechanism is triggered only when a hot module is actually encountered.

The relative bandwidth with this scheme never drops below 1. Moreover, it has the highest relative bandwidths of all the networks.

So far, we have seen how bandwidth can be improved when a fraction of processors are making hot requests and the rest are making uniform requests. The improvement stems from reducing the tree saturation that blocks the processors which are not participating in the hot spot activity. However, in all cases, no bandwidth improvement is obtained in the cases where all processors are making hot requests ($f=1$). How can we be sure that tree saturation is being controlled even in this case (and in cases where little bandwidth improvement is obtained)? As we have noted before, the maximum bandwidth is inherently limited by the number of requests that all must go to the same module and the only thing that can be done to improve the bandwidth is to cut down on the number of requests. However, the round trip latency experienced by the cold requests give us a good handle on the degree of tree saturation in the network.

Figure 7 shows the round trip latency of cold requests as a function of bandwidth for various values of h and the various networks. The latency is measured as the time taken by a request since its generation by the processor until the time the processor receives a response from the memory module (waiting times in all queues are included). All cases (Figures 7(a)-(d)) have the same saturation bandwidth (except 7(c), which is slightly lower as explained above) since $f=1$. However, the round trip latency of cold requests in each case is significantly different because of the different degrees of tree saturation present in the network in each case (note the difference in scales).

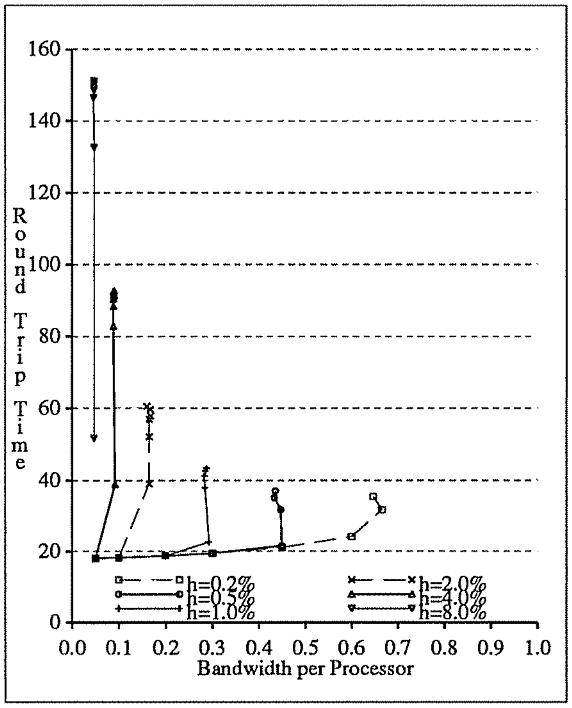
In a regular network (Figure 7(a)), the cold requests experience a long latency. This is consistent with the results reported by Pfister and Norton [11]. When simple feedback (with T=4) is used (Figure 7(b)), tree saturation is controlled somewhat and the latency is reduced, especially if the hot spot is more severe. Limiting (Figure 7(c)) is very effective in preventing tree saturation as is feedback with limiting-damping (Figure 7(d)). In the latter two cases, the hot rates restrict the bandwidth, but have very little effect on the latency of the cold requests. At the saturation bandwidth for a given hot rate, the cold requests encounter only slightly more contention than they would in a network with no hot spots carrying the same bandwidth. This is true because the bandwidth has been reduced by keeping the hot requests out of the network, rather than blocking them within the network.

5. DISCUSSION

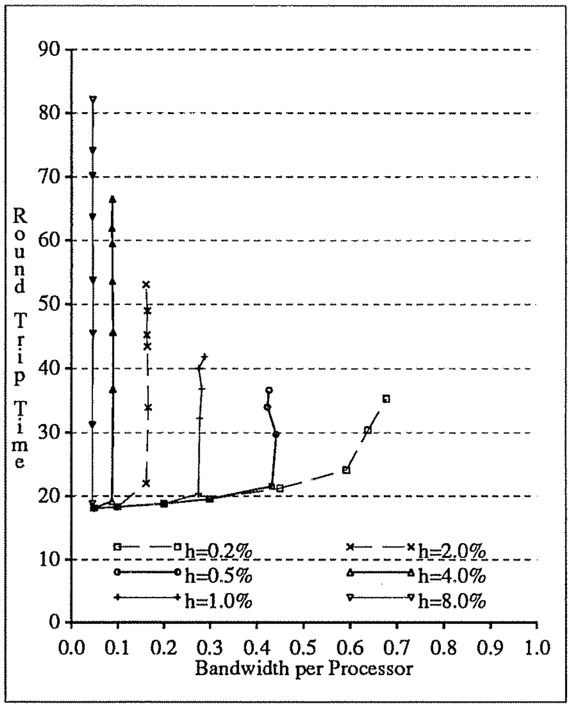
It is clear that using simple feedback can help alleviate the degradation caused by tree saturation in the network. This allows processors making uniform requests to proceed with less interference, thus increasing system throughput. It is also clear that this simple feedback method suffers from some problems analogous to overshoot and oscillation in classical control theory. If steps can be taken to reduce these problems, the effectiveness of feedback can be significantly enhanced. In this paper, we have discussed a strong form of damping which limited requests to cold modules to 2 per cycle and requests to hot modules to 1 per cycle. This proved to be very effective, but had a high hardware cost associated with it. We would like to explore other techniques to improve upon the basic limiting scheme.

One obvious improvement is to use large queues at the memory modules to increase the buffering of temporary tree saturation. Using larger queues toward the memory side of the switch has already been proposed in [13] for general networks. This technique is particularly appropriate for networks with feedback. First, it allows larger thresholds. Recall from the simulation results that the larger the threshold, the less bandwidth was unnecessarily restricted. With thresholds of 1 and 2, bandwidth was reduced to below that of a regular network for low hot rates. With a threshold of 4, bandwidth was not degraded at all when no hot spots were present. However, it was occasionally reduced slightly when hot spots were present, due to the hot module's queue becoming temporarily drained. If the threshold can be set to the number of levels in the network, then this degradation can be essentially eliminated.

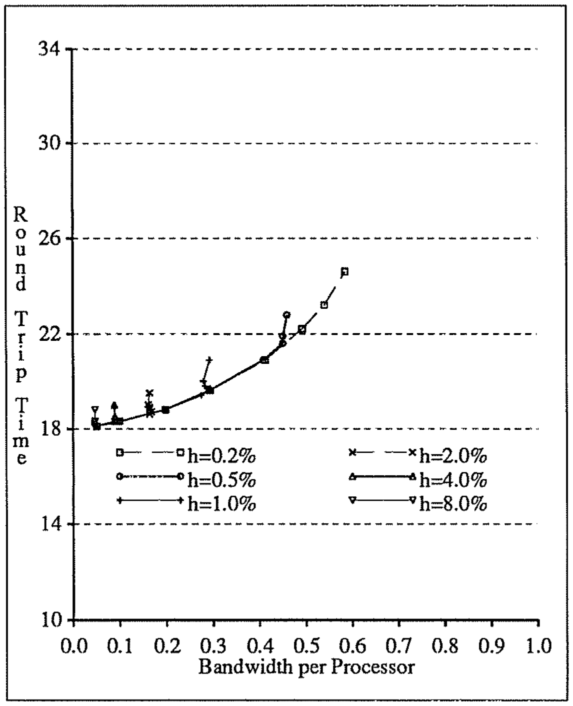
Larger queues at the modules will also buffer more of the overshoot tree saturation that occurs with feedback. Since the queue overflow in a network using feedback is temporary, and will be stopped by the feedback mechanism, larger queues can potentially absorb all or much of the partial tree saturation, even



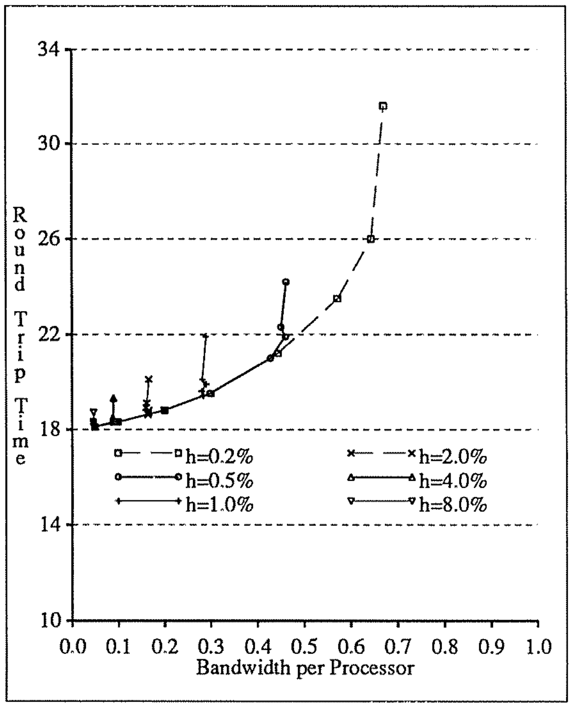
(a) Regular Network



(b) Feedback - Threshold 4



(c) Limiting



(d) Feedback + Limiting-damping

(All networks are 256x256 Omega networks, using 2x2 crossbar switches with 4-element queues on the switch outputs)

Figure 7: Delay of Cold Requests vs Bandwidth (f = 1)

in the presence of a steady state hot spot. In a regular network, there is nothing to prevent tree saturation from overflowing larger sized queues in the steady state.

Another simple way to improve upon our basic feedback scheme would be to shorten the delay between inputs and feedback. It is this delay which is primarily responsible for the overshoot in the system. Such schemes would involve feedback from points internal to the network. Performance would be enhanced by detecting congestion at earlier stages in the network and restricting requests that would aggravate this congestion. Alternately, mechanisms that fed information back *into* switches within the network could be constructed. The design of such mechanisms is beyond the scope of this paper.

Some form of damping would still be beneficial. Full-blown limiting as a damping mechanism may be impractical to build, but it may be reasonable to build a system which performs limiting on a single hot spot, as suggested in section 3.1. Memory queue threshold detectors and a single arbiter could be used to identify the hot module. Another arbiter would be used by processors attempting to make a request to the hot module. A global arbiter will not scale easily, but the cost for a single arbiter need not be prohibitive. An N-way arbiter can be built from a small tree of arbiters with lesser fan-ins. The arbiters need be only one bit wide, and can be constructed from readily available parts. As such their complexity should be small as compared to the interconnection network. It is not clear how effective a system would be that only dealt with one hot module, but preliminary experience shows [4, 11] that large scale parallel programs typically have only one or two hot spots at a time.

Other weaker forms of damping could be used as well. If limiting could be done separately in k slices of the processors, then the maximum number of requests to a particular module could be limited to k per cycle. This could significantly reduce the overshoot caused by many requests entering the network at once when a hot module becomes cold. Another possibility would be some sort of variable waiting time after a module becomes cold before different requests destined for that module enter the network (similar to the exponential back off scheme used with ethernets).

6. CONCLUSIONS

In this paper, we have proposed the use of feedback in multistage interconnection networks as an aid in the distributed routing process and evaluated the effectiveness of feedback mechanisms in controlling the tree saturation problem in such networks. We saw that, with feedback mechanisms, tree saturation can be controlled. That is, processors can avoid sending hot requests into the network where they will consume buffer space and block requests that could otherwise proceed. A network with feedback could be used in conjunction with software combining to provide protection against hot spots which were not caused by synchronization variables. Alternately, in systems with a general purpose and a combining network, feedback could be profitably applied to the general purpose network.

While we have only considered the example of tree saturation in multistage interconnections, feedback techniques are general enough to be used in any parallel or distibuted system where a resource can be accessed without the use of a global control mechanism and when contention for access to this resource can degrade the overall system. A network with feedback presents an alternative to a network with global control (that is expensive to implement) or a network with only a distributed routing control (that is prone to degradation because of non-uniform access of its resources). For example, feedback concepts could easily be applied to prevent undesirable situations in hypercube computers or in distributed computers.

The hardware requirements of feedback are modest. In a multistage interconnection network, feedback from the destinations to the sources requires no alteration of the interconnection network itself, and could thus be added to existing network designs with minimal upheaval. We believe that feedback could be used easily in many systems and specifically recommend its use for preventing undesirable situations in large-scale multiprocessors that use distributed routing controlled interconnection networks.

REFERENCES

[1] W. C. Brantley, K. P. McAuliffe, and J. Weiss, ''RP3 Processor-Memory Element,'' *Proceedings 1985 International Conference on Parallel Processing*, pp. 782-789, August 1985.

[2] W. L. Brogan, *Modern Control Theory*. New York, NY: Quantum Publishers, Inc., 1974.

[3] A. Gottlieb, et al, ''The NYU Ultracomputer -- Designing a MIMD, Shared Memory Parallel Machine,'' *IEEE Transactions on Computers*, vol. C-32, pp. 175-189, April 1987.

[4] M. Kalos, et al, ''Scientific computations on the Ultracomputer,'' Ultracomputer Note 27, Courant Institute, New York University, New York, NY.

[5] D. J. Kuck, et al, ''Parallel Supercomputing Today and the Cedar Approach,'' *Science*, vol. 21, pp. 967-974, Feb. 1986.

[6] M. Kumar and G. F. Pfister, ''The Onset of Hot Spot Contention,'' *Proceedings 1986 International Conference on Parallel Processing*, August 1986.

[7] T. Lang and L. Kurisaki, ''Nonuniform Traffic Spots (NUTS) in Multistage Interconnection Networks,'' *Proceedings 1988 International Conference on Parallel Processing*, August 1988.

[8] D. H. Lawrie, ''Access and Alignment of Data in an Array Processor,'' *IEEE Transactions on Computers*, vol. C-24, pp. 1145-1155, December 1975.

[9] G. Lee, C. P. Kruskal, and D. J. Kuck, ''The Effectiveness of Combining in Shared Memory Parallel Computers in the Presence of 'Hot Spots','' *Proceedings 1986 International Conference on Parallel Processing*, August 1986.

[10] A. Norton and E. Melton, ''A Class of Boolean Linear Transformations for Conflict-Free Power-Of-Two Access,'' *Proceedings 1987 International Conference on Parallel Processing*, pp. 247-254, August 1987.

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

[12] G. F. Pfister, et al, ''The IBM Research Parallel Processor Prototype (RP3): introduction and architecture,'' *Proceedings 1985 International Conference on Parallel Processing*, pp. 764-771, August 1985.

[13] H. S. Stone, *High-Performance Computer Architecture*. Reading, MA: Addison-Wesley, 1987.

[14] Y. Tamir and G. L. Frazier, ''High-Performance Multi-Queue Buffers for VLSI Communication Switches,'' in *Proc. 15th Annual Symposium on Computer Architecture*, Honolulu, HI, pp. 343-354, June 1988.

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