MASSACHUSETTS INTITUTE OF TECHNOLOGY

VLSI PUBLICATIONS

VLSI Memo No. 89-567
October 1989

Modeling a Circuit Switched Multiprocessor Interconnect

Daniel Nussbaum, Ingmar Vuong-Adlerberg, and Anant Agarwal

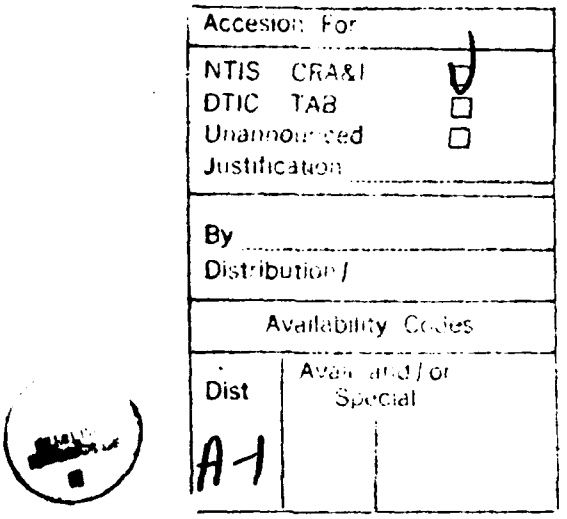
Abstract

This study began as an attempt to understand discrepancies between Patel's classic model of a circuit-switched interconnection network and simulations of the same, as part of the MIT ALEWIFE Multiprocessor project. After a careful analysis of Patel's model, we developed a model with fewer approximations that produced results generally closer to detailed simulation.

The major source of inaccuracy in Patel's model is the *unit-request* approximation, which treats a t-cycle request as t 1-cycle requests producing significant inaccuracies for networks with many switching levels and for small packet sizes. Our model followed the behavior of the network more closely, explicitly modeling the effects of switch size, network depth, packet size, and memory latency, thereby alleviating some of the inaccuracies in Patel's model. However, despite the slightly lower accuracy of Patel's model, we believe that its simplicity makes it the practical choice for most applications. Our main contribution, then, was to understand the causes of inaccuracies in both models, allowing us to predict the quality of the estimations they yielded.

Another result of this research is the validation of a complicated simulator using relatively simple models, not a common use for a model. We found that some of the original discrepancies between Patel's model and the simulator were due to hidden inconsistancies between parameters used by the model and those used by the simulator, and that others were due to bugs in the simulator code. Once these problems were corrected, both Patel's model and our model followed results from the simulator more closely.

Microsystems Technology Laboratories | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Room 39-321 Telephone (617) 253-0292



Acknowledgements

This research was supported by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825, and by grants from the Sloan Foundation and IBM.

Author Information

Agarwal: Laboratory for Computer Science, Room NE43-418, MIT, Cambridge, MA 02139. (617) 253-1448.

Nussbaum: Laboratory for Computer Science, Room NE43-624, MIT, Cambridge, MA 02139. (617) 253-6082.

Vuong-Adlerberg: Laboratory for Computer Science, Room NE43-219, MIT, Cambridge, MA 02139. (617) 253-1439.

Modeling a Circuit Switched Multiprocessor Interconnect

Daniel Nussbaum, Ingmar Vuong-Adlerberg*, and Anant Agarwal
Laboratory For Computer Science
Massachusetts Institute of Technology
Cambridge, Mass. 02139

September 22, 1989

Abstract

This study began as an attempt to understand discrepancies between Patel's classic model of a circuit-switched interconnection network and simulations of the same, as part of the MIT ALEWIFE Multiprocessor project. After a careful analysis of Patel's model, we developed a model with fewer approximations that produced results generally closer to detailed simulation.

The major source of inaccuracy in Patel's model is the *unit-request* approximation, which treats a t-cycle request as t 1-cycle requests producing significant inaccuracies for networks with many switching levels and for small packet sizes. Our model followed the behavior of the network more closely, explicitly modeling the effects of switch size, network depth, packet size, and memory latency, thereby alleviating some of the inaccuracies in Patel's model. However, despite the slightly lower accuracy of Patel's model, we believe that its simplicity makes it the practical choice for most applications. Our main contribution, then, was to understand the causes of inaccuracies in both models allowing us to predict the quality of the estimations they yielded.

Another result of this research is the validation of a complicated simulator using relatively simple models; not a common use for a model. We found that some of the original discrepancies between Patel's model and the simulator were due to hidden inconsistancies between parameters used by the model and those used by the simulator, and that others were due to bugs in the simulator code. Once these problems were corrected, both Patel's model and our model followed results from the simulator more closely.

*On leave from I.N.R.I.A. during 1989.

Contents

List of Figures

List of Tables

1 Introduction

The performance of the interconnection network is an important factor in determining the performance of medium or large-scale multiprocessors. Consequently, such a multiprocessor design must consider the effects of communication delays with respect to almost every major issue. In the ALEWIFE Multiprocessor project at MIT, we considered a low-latency circuit-switched network as one possible medium for the processor-memory interconnect.

The behavior of even relatively simple communication networks invariably becomes complicated and difficult to analyze. For this reason, simulation has often provided the answers to questions about network behavior. Unfortunately, simulations can become prohibitively expensive as the size of the network increases. Furthermore, simulation results provide scant intuition about the relative effects of parameters as systems grow, and the time required to do exhaustive simulations of all cases is prohibitively high. For these reasons, the development of models that approximate network behavior is useful. Perhaps even more importantly, such models can be used to verify simulations, as well as substituting for them when exact behaviors are not needed.

In this paper, we consider the behavior of an unbuffered, circuit-switched network composed of $k \times k$ crossbar switches. The behavior of a circuit-switched network can be contrasted with that of a packet-switched network, in which the network nodes become available for use immediately behind a packet travelling through the network, instead of being held for the duration of the transaction. See [1] for an analysis of such networks. Circuit-switched networks, although less efficient in their use of network switches, are generally much simpler to construct, and attempt to make up the lost efficiency by running a faster clock. Furthermore, circuit-switched networks are generally less prone to deadlocking and flow-control problems.

A circuit-switched network was analysed by Patel [2]. Patel's model, although quite good over a large range of parameters, employs one principal simplification, the *unit-request* approximation, which hurts it accuracy. In this paper, we discuss Patel's analysis and develop our own model for comparison. Like Patel, we explicitly model network depth and switch size, but unlike Patel, we explicitly take into account the effects of packet size and memory latency.

The next section of this paper describes the network. Section 3 presents Patel's model and Section 4 presents our improved model. Section 5 describes the architecture of the simulator that was used in this research. Section 6 contains the results of simulations, a comparison of these results against both models, and a discussion of the comparisons. Finally, Section 7 summarizes this work.

n	Number of network stages
k	Switch size ($k \times k$)
s	Packet size
T_M	Memory latency (cycles)
m	Cache miss rate
U	Processor utilization

Table 1: *Parameter Terminology*

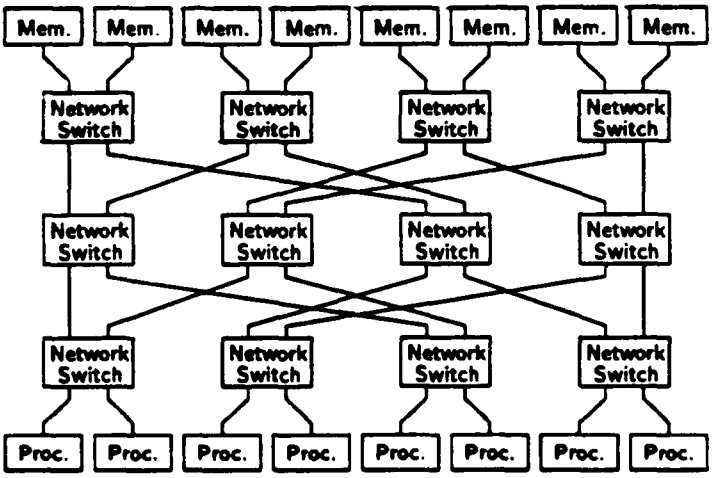


Figure 1: *Example Omega Network; 3 stages of 2 × 2 switches service 8 processors and 8 memories.*

2 Network Description

We consider the standard architecture of a multistage unbuffered circuit-switched network for connecting processors to memory modules. Table 1 contains a summary of the network parameters. Such an Omega[3] network is composed of n stages of $k \times k$ crossbar switches (see Figure 1. Other networks termed Banyan[4] or Delta[5] have a slightly different topology but the analyses are similar.) On any given cycle, a processor node will issue a network request with probability Um, where U is the processor utilization and m is the cache miss rate. The cache miss rate is the parameter that characterizes a cache; it is the ratio of the number of cache misses and the total number of processor non-idle cycles. Each request travels through the network stages toward the memory, establishing a path between the processor and the memory module to transfer the packet. As a packet makes its way from processor to memory, at each cycle, the head of the packet issues a request to the next network stage.

A collision occurs at network stage i either because two or more inputs to this stage need to use the same switch output or because the requested switch output is already part of a previously established path. When such a collision occurs, the node at which

the collision occurs first waits for the rest of the packet to stream in. The request path is then turned around and a one-word collision response packet is sent back to the point where the request originated. Therefore, a switch input is occupied for $s + 1$ cycles, where s is the packet size. This protocol is followed in the TRANSIT network [6]. Other protocols for collision resolution in circuit-switched networks, such as that in the Butterfly Network[7], can be modelled similarly.

The ultimate responsibility for ensuring completed transactions lies with the processing node, not the network. When a processor receives a collision response packet, it immediately reissues the request, repeating the process until the packet makes it to its intended destination.[1] Eventually the packet head successfully reaches the last stage of the network. At that point, the link to the requested memory module remains busy for $T_M + 2s$ cycles: T_M cycles for the memory operation itself and s cycles each for reading in of the request and sending out of the response. The total time between a (successful) memory request and positive acknowlegement is therefore $T_M + 2s + 2n$ cycles, the request and response packets' trips between processor and memory accounting for the additional $2n$ cycles.

Note that this network has no mechanism for backing up or queueing packets. The collision recovery process outlined above is carried out to its completion, even if the cause of the collision is removed on the very next cycle after it occurred. Therefore, a packet head moving from the input side to the output side of the network is known to take exactly n time units to do so, assuming no collisions, and a packet of size s takes exactly s cycles to travel past a given point in the network.

The analysis presented in this paper is based upon the following two assumptions:

(1) Cache requests to main memory are random, independent, and uniformly distributed over all memory modules. In addition, the structure of the net is such that requests at the inputs of any switch are uniformly distributed over all ouputs.

(2) The probability that a given packet head has a collision at a given switch is independent of the packet's history.

The legitimacy of the first of these assumptions is heavily dependent on the software that causes the requests to be generated. Programs for which the assumption is not true will not be modelled well by the analyses presented in this paper. In particular, programs that contain hotspots suffer contention-related performance losses that are not captured in any of the models we discuss. The second assumption will always be an approximation irrespective of program behavior. We will see below how this inaccuracy impacts the results of both models presented.

3 Patel's Model

Patel's model estimates the network delay using the *unit-request* approximation. Let the cache miss rate be m and the transaction time be t, where $t = T_M + 2s + 2n$ (as described

[1] Network congestion due to excessive retry traffic can be controlled by means of backoffs, as suggested in [8].

above). The unit-request approximation views this as equivalent to a situation in which the cache miss rate is mt and the transaction time is one. In other words, the processor splits up a memory request that would require t cycles to service into t independent and uniformly distributed unit-time sub-requests.

This approximation greatly simplifies the analysis. Introducing the definition:

$$r_i \equiv \text{Probability that any link at level } i \text{ of the network has been requested,}$$

Patel develops the following set of equations, which can be solved numerically:

$$U = 1 - r_0 \tag{1}$$

$$r_{i+1} = 1 - \left(1 - \frac{r_i}{k}\right)^k \tag{2}$$

$$r_n = Umt \tag{3}$$

In this model, a processor is involved in a network access whenever it is not executing instructions, yielding equation (1). The model assumes that if a processor issues a network transaction with probability m on any given non-idle cycle, and if the request desires t cycles of network service time, then the effective network request rate is mt. Equation (3) equates the request service rate at the output of the network to the rate at which new requests are injected into the network in the steady state. Equation (2) states that the request rate r_{i+1} at an output of a network switch is the probability that at least one of the input requests is routed to that output port.

Simulations match the model reasonably well over some ranges of independently variable system parameters (k, n, s, T_M, m). However, Patel's approximations contain three major sources of error:

1. The actual network time needed by a request varies with the depth of the network being considered. For example, if a non-colliding packet holds a switch at level i of the network for t cycles, it will hold the switch at level $i+1$ of the network for $t-2$ cycles. By not taking this into account, the expected contention at stages of the network nearer the memories is artificially elevated. This would tend to make the predicted values for U more *pessimistic*.

2. In reality, after a collision occurs, the path established so far by the colliding packet takes some time to break down. This results in a time lag that is not modelled, and tends to make the predicted values for U more *optimistic*.

3. When a packet collides at a given switch, the chances that a corresponding retry will collide at the same switch can change significantly (usually going up), especially for large T_M, s, or n. This effect is not modeled, making the predicted values for U more *optimistic*.

In the next section, we introduce a model which answers the first two of these objections. Our model doesn't address objection (3), but we will analyze its effect in Section 6.

4 A More Accurate Model

In Patel's model, a network link is in one of two states: requested or idle. In our view, a link has three states: newly requested, held, or idle. At level i of the network, we introduce the following definitions:

$h_i \equiv$ Probability that a given link is newly requested.
$z_i \equiv$ Probability that a given link is being held.
$c_i \equiv$ Probability that a new request that made it to level i didn't make it to level $i+1$.

These definitions are used in the following equations:

$$h_n = Um \tag{4}$$

$$z_n = h_n\,(T_M + 2s - 1) \tag{5}$$

$$c_n = 0.0 \tag{6}$$

$$h_{i+1} = \sum_{d=0}^{d=k} \left[\binom{k}{d} (1-z_i)^{k-d}\, z_i^d\right] \left[\frac{k-d}{k} + \frac{d}{k}\frac{c_i}{z_i}(s-1)\right] \left[1 - \left(1 - \frac{h_i}{k(1-z_i)}\right)^{k-d}\right] \tag{7}$$

$$= [1 - h_i - h_{i+1} - z_{i+1}] \left[1 + \left(1 - \frac{h_i}{k}\right)^{k-1}\right] - \left[\frac{h_i}{k}\left(1 - \frac{h_i}{k}\right)^{k-1}\right] \tag{8}$$

$$c_i = h_i - h_{i+1} \tag{9}$$

$$z_i = z_{i+1} + 2h_{i+1} + sc_i \tag{10}$$

$$U = 1 - (h_0 + z_0) \tag{11}$$

These forumlas, although more complex, are similar to those from Patel's models. Equation (4) simply states that the completion rate equals the utilization times the offered load (since all requests that make it to stage n of the network are guaranteed to complete without collision). Equation (5) says that stage n of the network is held for a total of $T_M + 2s$ cycles (including the initial requesting cycle). Equation (6) says that there are no collisions once a packet makes it to stage n.

Equation (9) indicates that the number of collisions at level i equals the difference in the number of new requests between levels i and $i+1$. Equation (10) relates the hold time for a link at stage i to the hold time at stage $i+1$, taking into account the collisions that happen in the switch at stage i. Equation (11) states that a processor is involved in a network access whenever it is not executing instructions; network busy time at level 0 equals processor idle time.

Equation (7) is the core of our model, in that it exactly reflects the probabilistic behavior of a switch. It is similar to (2), except it includes contention between new

requ ing packets and held paths. It is expressed as a sum over the number of input links that are being held on any given cycle. The first term is the probability that d inputs are being held. The second term is the probability that an output is not being held, given that d inputs are being held. Finally, the third term is the probability that an output is newly requested, given that d inputs are being held. Equation (8) is a simplification (no approximations made) of Equation (7).

Note that the first two of the three objections presented in Section 3 do not apply to this model. The source of error in this model is that successive retries of the same packet are not independent. In fact, the actual probability of collision is usually higher for a retry packet than for the original request; therefore this model usually gives slightly optimistic results. We address this issue further in Section 6.

5 Simulation

Both models were compared with our simulator, which took a set of memory traces as input (one for each processor in the network) and produced performance statistics as output. Ideally one would like to use real address traces from a multiprocessor. Unfortunately, it is hard to compare results generated by real traces to those from models, due to statistical fluctuations in the behaviors of the traces and in the algorithms they came from. Initially, we did use real multiprocessor traces of parallel applications, and we thought that discrepancies between the models and simulations were due to trace peculiarities.

To eliminate trace peculiarities as a source of error, we generated parameterized synthetic traces. These synthetic memory traces were generated using independent identically distributed geometric random variables parameterized by m to give the memory access inter-arrival times. For each memory reference on each processor, the time until the next reference is predicted by the following probability distribution:

$$P(t = t_0) = (\frac{1-m}{m})m^{t_0} \quad t_0 = 1, 2, 3, ...$$

Memory access addresses were generated using independent identically distributed uniform random variables spread out over the entire address space:

$$P(a = a_0) = \frac{1}{M}; \quad a_0 = 0, 1, ..., M-1$$

A file containing p traces (one for each of p processors) reflecting these statistics was used as input to the simulator.

The simulation itself used an event-based algorithm in which network nodes were only looked at when they had something to do. This simulation methodology was quite efficient, but for large networks and long runs, the computational load was heavy. This is the main motivation for modelling networks in the first place.

The network topology simulated consisted of an array of $k \times k$ crossbar switches connected in a Omega [3] configuration n levels deep to service k^n processors and k^n memories. All packets were the same size s and all memories had the same latency T_M.

Due to statistical unevenness in network delays experienced by different processors, a certain amount of skew was introduced into the processor clocks (a processor clock advances by one tick for each cycle that the processor is busy, that is, when the processor is not waiting for a network request to complete). Such skew would give reason to question results from traces generated by real programs (which might include synchronization delays and timing-dependent access patterns). However, since all simulations done here were based on random traces, the skew introduced should have no effect on the results.

Simulations measured the key statistic U, computed as:

$$U = \frac{\text{processor busy time}}{\text{total time}}$$

This statistic was taken as an average over all processors. Of course, higher values for U translate to more effective processor usage.

6 Results

The multiprocessor configuration we are most interested in includes caches using a directory-based coherence scheme [9, 10]. For this configuration, measurements from our parallel program traces yielded a typical point in parameter space: $k = 4$, $n = 3$, $s = 4$ (16-bit paths between switches), $T_M = 4$, $m = 0.1$. Data comparing the models and the simulation were taken by varying the relevant parameters around this point.

The network described by this point connects 64 processors to 64 memory modules and can be simulated in a reasonable amount of compute time. A 50000 processor-cycle simulation gives an estimate of the processor usage with a 0.1% error, and takes only 5 minutes on a Microvax III. Since the simulation is run for a constant number of non-idle processor cycles, the run time is inversely proportional to the simulated processor utilization(U). Packet size, memory latency and cache miss rate variations all have a near linear effect on U in the limit where U becomes small. Therefore, we were able to run experiments for a wide range of values for these parameters in a reasonable amount of time.

However, increasing the network size by varying the switch size or the network depth leads rapidly to very expensive computations. For example, simulating 50000 processor cycles on a network that interconnects 4096 processors and memory modules requires 50 megabytes of memory space and takes about 6 hours on a 10-Mips Convex computer containing 64 megabytes of main memory. The same computation on a Microvax III (3 Mips) containing 24 megabytes of main memory takes 3 days. 4096 processors was therefore our limit, allowing configurations ranging from $k = 2$ and $n = 12$ up to $k = 64$ and $n = 2$.

Obtaining iterative numerical solutions for our model was a delicate task, primarily due to the difficulty of inverting Equation 8. Iterative algorithms we used exhibited a trade-off between stablilty and convergence time, which were highly sensitive to the quality of the first guess. We found that second order Taylor approximations of the core equation for our model (Equation 8) yielded a new system of equations that was more

easily solved. We then used $U = [1 + m(T_M + 2s + 2n)]^{-1}$ as a first guess when solving the approximated sets of equations. The resulting solution was of sufficient quality as to guarantee convergence when used as the first guess in solving the original system of equations.

In fact, for the data points in this paper, the approximate result never differed from the exact result by more than 1%, because $\frac{h_i}{k}$ is always very small. We therefore found our approximation valuable not only as numerical support, but also as a simpler expression of our model. The following can be substituted to approximate Equation (8) in our model:

$$h_{i+1} = \left[\left(\frac{z_{i+1}}{2}\right)\left(3 + \frac{7}{k} - \frac{4}{k^2}\right) + \frac{5}{2k} - \frac{5}{2}\right] h_i^2 + \left[1 - z_{i+1}\left(1 - \frac{1}{k}\right)\right] h_i \quad (12)$$

It then can be used to invert equation (8) in order to express h_i as a function of h_{i+1} and z_{i+1} simply by solving a second degree polynomial.

Figures 2 to 12 represent processor utilization as a function of one of the five network parameters, estimated using both models, and obtained from simulation. In general, the estimates do not differ by more than 20% from the simulation. More specifically, except for one graph (variation in T_M, as T_M gets large), our model is consistently optimistic by up to 5%, while Patel's model is pessimistic for small values of parameters and becomes optimistic for large values, with an error that ranges from -20% to +15%, reflecting both pessimistic and optimistic approximations in Patel's model that often cancel each other.

At a finer level of detail, we can compare the ability to model the effect of each of the network parameters for both Patel's and our model. The following sections analyse the behavior of the models and simulation as the parameters are varied.

6.1 Switch Size

Figures 2 to 5 represent the variation of U with the size of the switch for $n = 1, 2, 3, 4$ respectivly. Both models are equally good, Patel's being pessimistic for small values of k by -5% while ours is optimistic by 5%. For larger values of k our model and the simulation tend to converge.

Interestigly, it appears that for very large k our model eventually becomes pessimistic as well (especially note figure 4). Careful thinking is necessary in order to understand this behavior. A given request is rejected for one of two reasons:

(a) Because it collides with other requests that were issued simultaneously by other processors.

(b) Because the desired switch is busy servicing a previous request.

The probabilities P_a and P_b that a packet gets rejected because of the effect of (a) and (b) respectively, can be approximated by:

$$P_a = \frac{1 - P_b}{1 - U}\left[1 - \left(1 - \frac{1 - U}{k}\right)^k\right] \quad (13)$$

$$P_b = Um(T_M + 2s + 2n)$$

These simple equations actually explain why, under certain circumstances, our model eventually becomes pessimistic. If (b) is predominant, then successive retries are more likely to collide again, since when the packet is reissued, the switch may still be busy, particularly if the average time that a switch is busy is high compared to the lapse time between retries (which is the case for instance when T_M is high in the simple case $n = 1$). If (a) is predominant, then successive retries are less likely to collide again, since the reason for failure is competition between requests, and that at each new retry, one colliding request is serviced, thereby reducing the competition after the subsequent service time is exhausted. This means that when (a) is the main source of collisions, our model is pessimistic, and the network appears to be routing more traffic than one would expect. We will refer to this effect as *super-throughput* in the rest of this paper. When (b) is predominant, which is normally the case, our model is optimistic.

To confirm this hypothesis, we made a rough estimate of the relative importance of (a) and (b) which gave $P_a = 0.44$ and $P_b = 0.37$ for the case represented on Figure 4 where our model becomes pessimistic, and $P_a = 0.39$, $P_b = 0.47$ on Figure 2 where the model stays optimistic for large k. In all cases, our model tends to become less optimistic as k grows, but it is only for some particular sets of parameters that it actually becomes pessimistic. An average switch hold time that is small compared to the average time between retries will render the effect of (b) combined with the approximation (3) small. In particular, when the utilization happens to be low (UmT_M under 0.3), the effect of (a) is amplified. This usually corresponds to small values of T_M (a switch holding a path to the memory remains busy for shorter periods) or to large values of n (bottom switches are very likely to be holding paths to packets that will collide before getting to the memory, therefore becoming rapidly free in most cases).

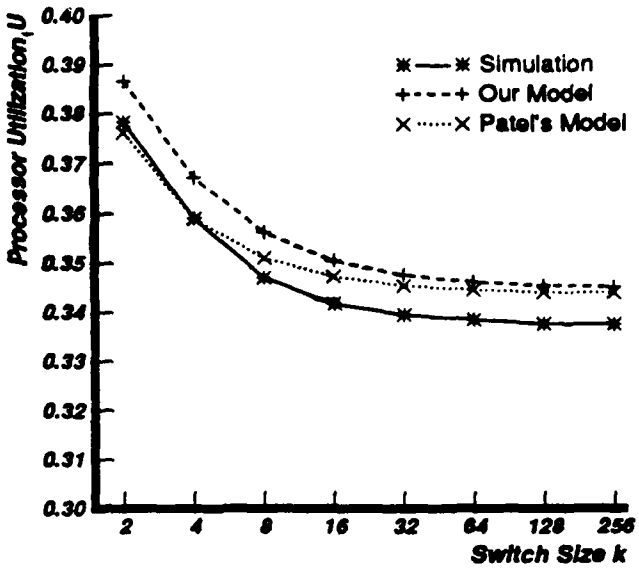


Figure 2: *Utilisation Vs. Switch Size k* ($m = 0.1$, $n = 1$, $s = 4$, $T_M = 4$).

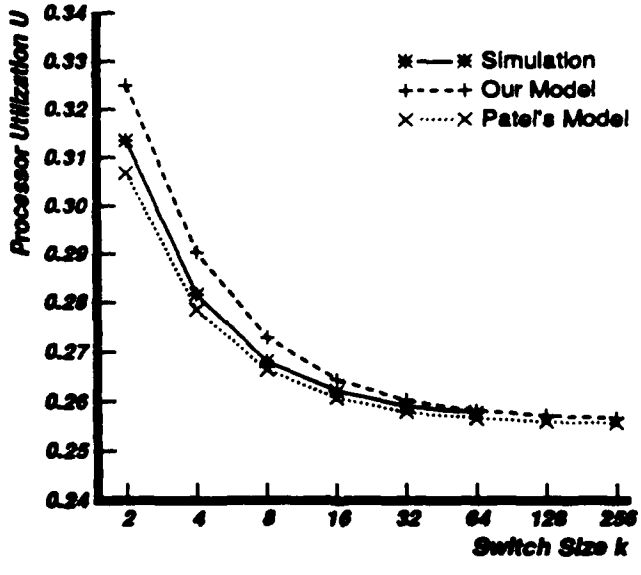


Figure 3: *Utilisation Vs. Switch Size k* ($m = 0.1$, $n = 2$, $s = 4$, $T_M = 4$).

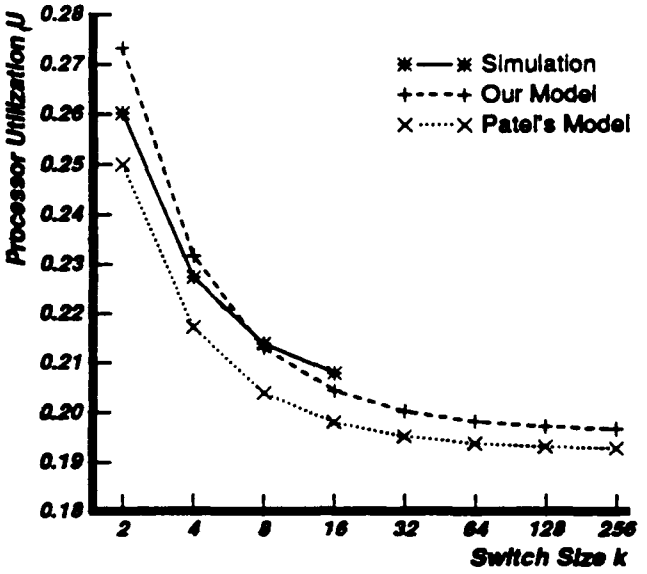


Figure 4: *Utilization Vs. Switch Size k* ($m = 0.1$, $n = 3$, $s = 4$, $T_M = 4$).

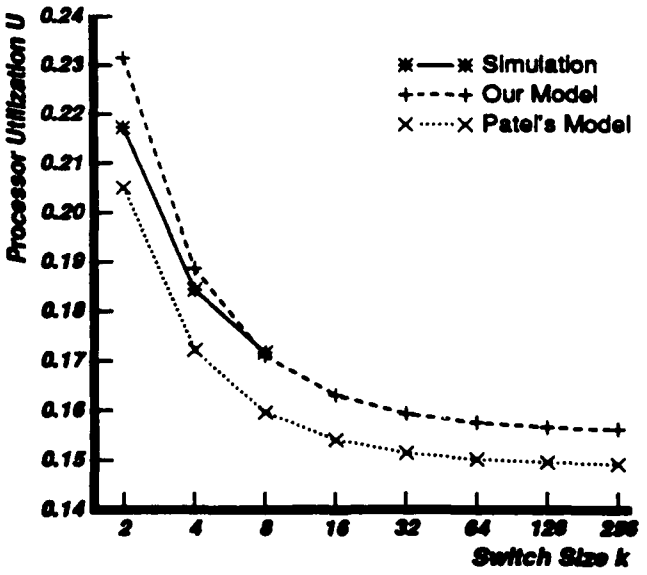


Figure 5: *Utilization Vs. Switch Size k* ($m = 0.1$, $n = 4$, $s = 4$, $T_M = 4$).

6.2 Network Depth

Figures 6 to 9 represent the variation of U with the network depth for $k = 2, 4, 8, 16$ respectivly. In this case, our model follows the simulation curve quite closely. Patel's model works quite well for small values of n but diverges to pessimistic values rapidly as n increases. For instance for $k = 4$ and $n = 6$, Patel's model is pessimistic by 20% while ours is still accurate within 1%. Nevertheless, it seems, looking at the shape of the curves, that for much larger values of n our model eventually becomes pessimistic, particularly for large values of k, but even then remains better than Patel's model. The nature of this phenomenon was described in the previous paragraph as super-throughput, and actually can be generalized not only for large n or large k but for any large number of processors: the importance of super-throughput increases with both k and n. When calculating the number of processors, k is the base and n is the exponent, and this effect is therefore more sensitive to changes in k. Our approximation is therefore better with a large n configuration than with a large k configuration.

Finally, these graphs also show that Patel's model of the network's depth is inaccurate, and eventually diverges from the simulation by pessimistic values. This phenomenon can be easily explained: Patel describes the successive attemps of a given packet to make it through the levels of the net as if all of them had to cross the entire net. This is inaccurate because as a packet gets closer to the memory it needs less service time since it has less levels to cross. In particular, if $2n \simeq T_M + 2s$ then Patel's model performs quite badly and as a consequence, as n grows, Patel's estimate is more and more pessimistic.

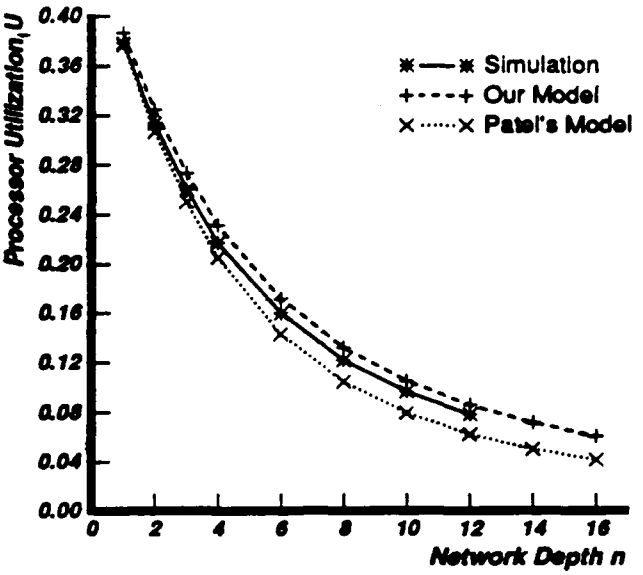


Figure 6: *Utilization Vs. Network Depth n* ($k = 2$, $m = 0.1$, $s = 4$, $T_M = 4$).

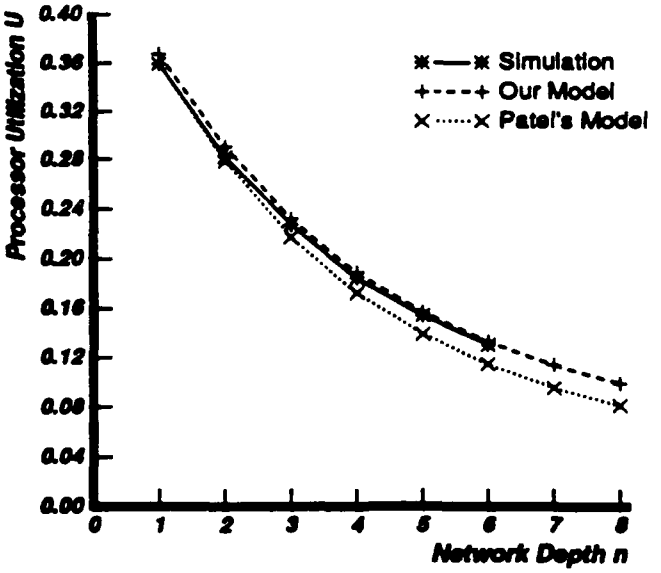


Figure 7: *Utilisation Vs. Network Depth n* ($k = 4$, $m = 0.1$, $s = 4$, $T_M = 4$).

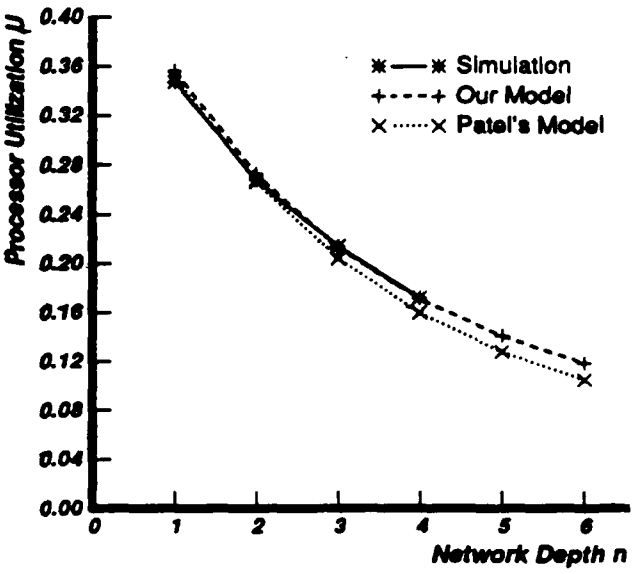


Figure 8: *Utilization Vs. Network Depth n* ($k = 8$, $m = 0.1$, $s = 4$, $T_M = 8$).

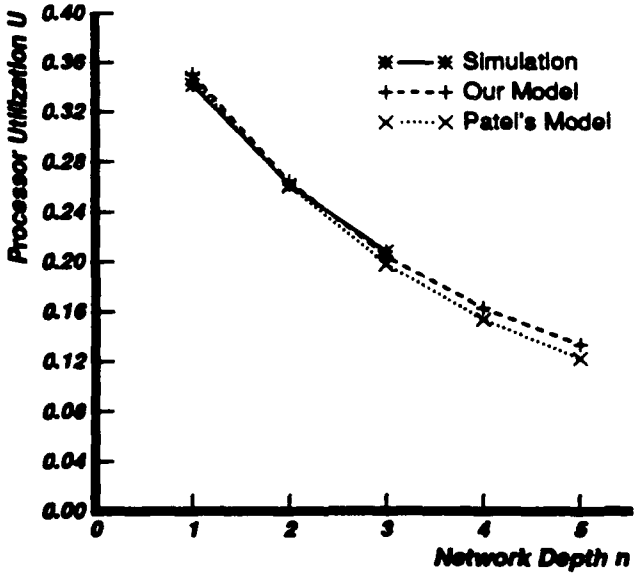


Figure 9: *Utilization Vs. Network Depth n* ($k = 16$, $m = 0.1$, $s = 4$, $T_M = 8$).

6.3 Miss Rate and Packet Size

The miss rate and packet size are functions of the cache size and the applications software.

Figure 10 shows that both models describe the effect of varying the miss rate very well.

Figure 11 shows that our model handles the effect of packet size better than Patel's. Our model follows the simulation curve very accurately, becoming slightly optimistic for large values of s. This optimism increases slowly because the non-independance of successive retries is important if the time between two retries is small compared to the average time that a switch remains busy. As the packet size becomes large, the average time that a switch stays busy increases (since a larger packet has to travel toward the memory), but the average time between retries also increases since a larger packet has to bounce back.

6.4 Memory Latency

Figure 12 shows the effect of memory latency T_M. It appears that large memory latency has a strong effect on the non-independence of successive retries. The reason is simply that, unlike in the previous case (packet size increasing), an increase in memory latency increases the average time that a switch remains busy, without increasing the average time between succesives retries. As a consequence, our model works well for small values of T_M but rapidly diverges. Patel's model diverges initially, but for larger values of T_M, the other approximations of his model compensate for this effect. It appears that this is the only case where our model performs relatively badly, and that Patel's model gives better results. Nevertheless, our model always works for small values of T_M, (T_M smaller than $n + s$) while it is unclear how accurate is Patel's model for small values of T_M for different sets of parameters.

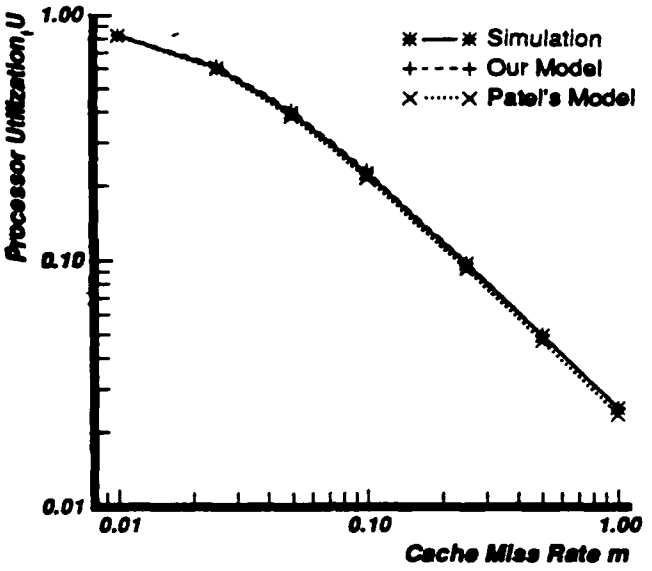


Figure 10: *Utilization Vs. Request Rate m* ($k = 4$, $n = 3$, $s = 4$, $T_M = 8$).

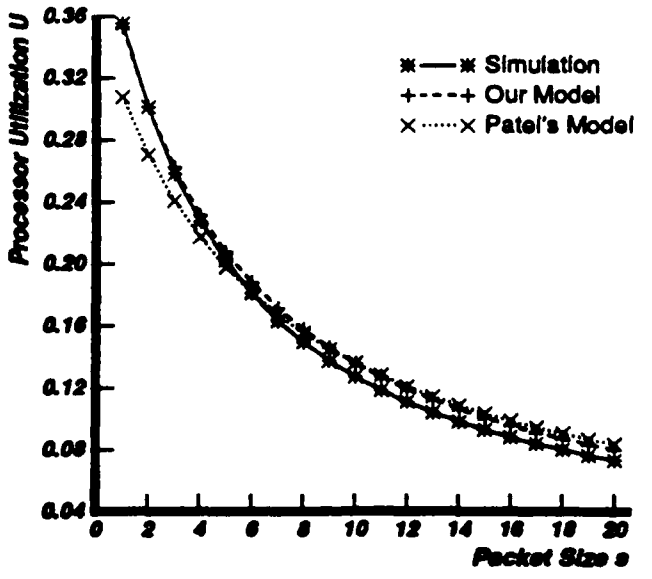


Figure 11: *Utilisation Vs. Packet Size s* ($k = 4$, $m = 0.1$, $n = 3$, $T_M = 8$).

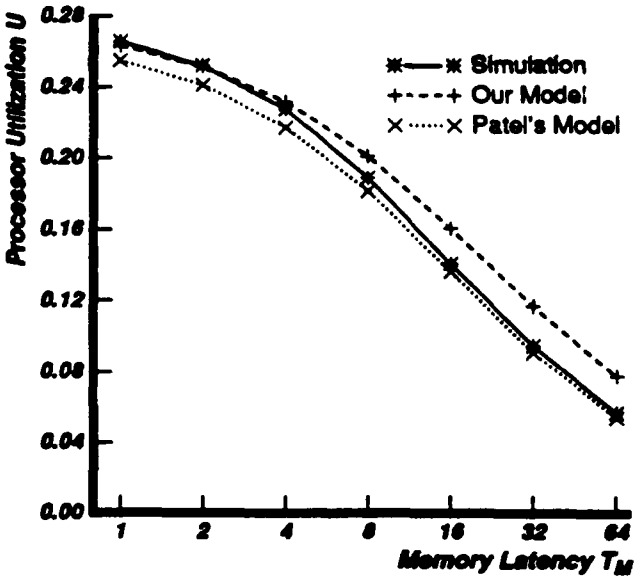


Figure 12: *Utilisation Vs. Memory Latency* T_M ($k = 4$, $m = 0.1$, $n = 3$, $s = 4$).

7 Conclusions

The original need for this research arose when the simulator built to evaluate a circuit switching network as a candidate network for the ALEWIFE machine[2] gave results that differed from those predicted by Patel's classical model for the processor utilization U. It turned out that this phenomenon was due to hidden inconsistancies between parameters used by the model and those used by the simulator and actual bugs in the simulator code, as well as approximations in Patel's model.

One of the major achievments of our work was to validate our simulator by comparing it to a model that we understood well in the case of a uniform distribution of memory adresses. We worked on both looking for bugs in the simulator and understanding the model until we got numbers that were consistent. We then felt confident enough to use the simulator on real program memory traces.

On the other hand, a careful study of the modeling process showed that Patel's classic model worked fairly well in our region of interest (a 3 level network built out of 4×4 switches, transmitting 4-word packets and having a memory latency of 4 cycles). Patel's estimates were all off by less than 10%, using a very simple model. Hence, for most practical purposes, Patel's model is probably the right one to use.

By developing our own model, we obtained better estimates in our region of interest (less than 3% error), and better accuracy around this region of interest. In addition to yielding more precise results, our model has given us a better understanding of the quality of the model as parameters vary. Unlike Patel's, our model is almost always optimistic. Except in the case of very large T_M ($> n + s$), our model gives a better functional profile in modeling the effect of each individual parameter on the processor utilization U. Finally, despite a more complicated set of equations, we were able to provide an approximation that made the numerical solutions converge as easily as the simpler classical set of equations.

8 Acknowledgments

The research reported here has benefited greatly from discussions with Gino Maa, David Chaiken, Greg Bromley and Tom Knight. This research was funded by DARPA contract # N00014-87-K-0825, and by grants from the Sloan foundation and IBM.

[2]The design currently uses a packet-switched direct network with wormhole routing [11] although evaluations comparing it with alternative interconnect configurations are still in progress.

References

[1] Clyde P. Kruskal and Marc Snir. The Performance of Multistage Interconnection Networks for Multiprocessors. *IEEE Transactions on Computers*, C-32(12):1091–1098, December 1983.

[2] Janak H. Patel. Analysis of Multiprocessors with Private Cache Memories. *IEEE Transactions on Computers*, C-31(4):296–304, April 1982.

[3] D. H. Lawrie. Access and Alignment of Data in an Array Processor. *IEEE Transactions on Computers*, C-24(12):1145–1155, December 1975.

[4] G. R. Goke and G. J. Lipovski. Banyan Networks for Partitioning Multiprocessor Systems. In *Proceedings of the 1st Annual Symposium on Computer Architecture*, pages 21–28, IEEE, New York, 1973.

[5] Janak H. Patel. Performance of Processor-Memory Interconnections for Multiprocessors. *IEEE Transactions on Computers*, C-30(10):771–780, October 1981.

[6] Thomas F. Knight Jr. Technologies for Low Latency Interconnection Switches. In *Proceedings of ACM Symopsium on Parallel Algorithms and Architectures*, June 1989.

[7] *Butterfly Products Overview.* Technical Report, BBN Advanced Computers, Inc., October 1987.

[8] Anant Agarwal and Mathews Cherian. Adaptive Backoff Synchronization Techniques. In *Proceedings 16th Annual International Symposium on Computer Architecture*, IEEE, New York, June 1989.

[9] Anant Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz. An Evaluation of Directory Schemes for Cache Coherence. In *Proceedings of the 15th International Symposium on Computer Architecture*, IEEE, New York, June 1988.

[10] Anant Agarwal, David Chaiken, Craig Fields, and Kiyoshi Kurihara. Scalability of Directory-Based Cache-Coherent Multiprocessors. September 1989. Laboratory for Computer Science, Massachusetts Institute of Technology. In preparation.

[11] William J. Dally. *A VLSI Architecture for Concurrent Data Structures.* Kluwer Academic Publishers, 1987.