McGill University
School of Computer Science
ACAPS Laboratory

Advanced Compilers, Architectures and Parallel Systems

Performance Analysis of Multithreaded Architectures using an Integrated System Model

Shashank S. Nemawarkar†
Guang R. Gao

ACAPS Technical Memo 84-1 (Draft)
April 10, 1995

†Department of Electrical Engineering
email: {shashank,gao}@macs.ee.mcgill.ca

Submitted to the Parallel Computing journal published by North-Holland.

Abstract

In this paper we develop a performance model for multiprocessor systems with multithreaded processing nodes. Multithreading presents challenging issues to the performance tuning because multiple requests from a processor may be outstanding and being processed in the system, and the feedback effect of the load due to multiple threads at memory and network on the processor utilization must not be ignored. Our integrated system model, based on closed queuing networks, takes into account the behavior of processors, memories and interconnection network, and the interaction between them under various application program load. We present the formulation of the model, its solution techniques, its validation, and its application to the performance evaluation of multithreaded architectures.

Our integrated system model takes as input, workload (e.g. number of threads, thread runlength, remote access pattern, etc.) and architectural (memory access time, network switch delay etc.) parameters — which are directly provided by users (programmers or compiler writers) and architects. Given these parameters, we show how to derive key performance measures of interest– processor utilization, message rate to the network, and observed network latency. We apply our model to quantitatively characterize variations of these performance measures, understand system bottlenecks, and provide insight to the impact of performance related optimizations. Concurrently analyzing the network and processor performance, we show how significant is the *network capacity* to tune the workload characteristics to achieve a high processor utilization. Usefulness of our closed system model is two-fold– one, users can work directly with the program workload and architectural parameters which are familiar, and two, near the network saturation, unlike open system models, problems of dealing with high sensitivity of performance to input parameters, do not arise. Lastly, we compare the robustness of our model with open system models in prediction of the performance, and demonstrate their weaknesses and tradeoffs with respect to our model.

Key Words : Parallel and high performance computer architectures, multithreaded architectures, performance modeling and evaluation, interconnection networks, mean value analysis.

Contents

1 Introduction **1**

2 Multithreaded Execution Model **3**

3 Results **7**

4 Validation **9**

5 Processor Utilization **10**

5.1 Model Parameter Characterization 11

5.2 Analysis of Performance Transitions 13

5.3 Summary 14

6 Message Rate to the Network **14**

6.1 Capacity of the Network 14

6.2 Model Parameter Characterization 15

6.3 Summary 18

7 Network Latency **18**

7.1 Parameter Characterization 18

7.2 Summary 20

8 Usefulness and Robustness **21**

8.1 Usefulness of Closed System Model 22

8.2 Robustness of Processor Performance Prediction 23

8.3 Summary 28

9 An Example for Workload Optimization **29**

10 Related Work **29**

11 Conclusions **31**

12 Acknowledgment **32**

A Approximate Mean Value Analysis **35**

B Symbols and Their Meanings **36**

List of Figures

1 4×4 Multiprocessor with 2-dimensional mesh . . . 4

2 A Processing Element . . . 5

3 Queuing Network Model . . . 7

4 Model and Simulations . . . 10

5 Effect of n_t and p_{remote} on U_p . . . 11

6 Effect of S on U_p . . . 12

7 Effect of R and L on U_p . . . 13

8 Effect of n_t and p_{remote} on λ_{net} . . . 16

9 Effect of S on λ_{net} . . . 16

10 Effect of R and L on λ_{net} . . . 17

11 Effect of n_t and p_{remote} on S_{obs} . . . 19

12 Effect of S on S_{obs} . . . 21

13 Effect of R and L on S_{obs} . . . 21

14 Operating Point in a Multiprocessor Network . . . 24

15 Impact of Locality on Operating Point . . . 25

16 S_{obs} and λ_{net} variations at $p_{remote} = 0.5$ . . . 26

17 Feedback Algorithm . . . 27

List of Tables

1 Default Settings for Model Parameters . . . 9

2 Network Latency Tolerance, with $R = 10$ and $R = 20$ . . . 20

3 AMVA Algorithm . . . 35

4 Model Parameters . . . 36

1 Introduction

In this paper, we develop an analytical model for the performance analysis of multiprocessor systems with multithreaded processing nodes. A processing node consists of the processor and memory (a part of distributed shared memory) subsystems, and communicates with other nodes through the interconnection network (IN) subsystem. Latency for this communication significantly affects the performance of the multiprocessor system [8]. The latency of a subsystem is the response time of the subsystem to a request for an access. The longer the latency, the larger the waiting time for the computation requesting the access. This may decrease the processor utilization.

Multithreading technique has been proposed to improve the performance of a processor by overlapping the communication with the computation (an in-depth introduction to multithreaded architectures can be found in [14]). A processor in a multithreaded multiprocessor system (MMS) supports multiple outstanding requests for accesses to allow for this overlap. The higher the overlap, the better the performance. The workload for a multithreaded multiprocessor system is an application program in the form of partially ordered threads. Each thread is a set of instructions for computation followed by a long latency operation (like a memory access or a synchronization). A processor performs the computation, issues a request for a long latency operation, and then switches to another thread. By context switching to another thread (rather than idling), the processor utilization improves.[1] But a side-effect of multiple outstanding requests is to increase the contention at the memory and interconnection network, which may further increase the memory and network latencies.

The overlap of computation poses challenges to the performance modeling of a multithreaded multiprocessor system. First, unlike in systems with single-threaded processors, a processor can continue the execution (on another thread) after issuing a long-latency memory request (be its local or remote), and after the memory access request is serviced for a thread, the processor may not execute on the thread immediately. Second, accesses from multiple threads contend at subsystems, increasing their observed latencies (even at local memories). In turn, longer latencies for individual accesses delay the execution on the waiting threads. So a complete performance model should capture the feedback effect of the concurrent activities at various system resources on the access rate and latency of a subsystem. This is essential in our investigation of the overlapping of computation on multiple threads, and its impact on the performance of the system.

Our performance model is motivated by the needs of our anticipated users: architects and users (programmers and compiler writers) of multithreaded computers. Given a set of architectural parameters (e.g., number of processing nodes, memory access time, network switch delay, context switch time[2]), users would like to know:

- How to achieve high processor utilization on a target set of program workload?
- How does the processor utilization vary with the workload?
- How does the network performance vary with the program workload?

[1] Major differences between multithreading and multitasking are that multithreading uses fine grain threads with very small context switch overhead comparable to the latency for single access through interconnection network; and these threads interact towards completion of a single task.

[2] Context switch time comprises of a fixed cost for the thread management and a variable cost for the number of registers to be saved.

- How does the network performance affect the processor utilization?

For compiler writers, above information is useful to effectively partition a program workload. For system architects, the information is needed to tune the architectural parameters for a target set of workload. A concern for users is whether they can use the parameters they are familiar with, or do they need a precomputation of input parameters for use of a performance model. And finally, how robust is the prediction of the performance model?

We propose an integrated system model, based on closed queuing networks (CQN), which takes into account the behavior of processors, memories and interconnection network, and the interaction between them under various application program load. Input to our integrated system model are workload (e.g., number of threads, thread runlength, remote access pattern, etc.) and architectural (memory access time, network switch delay etc.) parameters — which are directly provided by users (programmers or compiler writers) and architects. Our integrated system model predicts the processor utilization, the network latency and the message rate to the network. We present the formulation of this model, its solution techniques, its validation, and its application to the performance evaluation of multithreaded architectures. One of the advantages of using queuing models is that they are structurally similar to the real system, and they can be analyzed in a reasonable time (e.g., our model takes $<$ 20 seconds for a 64-node multithreaded multiprocessor system, on SparcStation-20).

Previous analytical studies of multithreaded architectures have mostly focused on the effect of workload parameters on processor performance (see Saavedra-Barrera et al. [23], Alkalaj et al. [6], and Agarwal [4]), but their models do not incorporate the effect of contentions at memory and network. On the other hand, studies on interconnection networks model the effect of contentions, but assume that the effect of workload is reflected by a constant input message rate (see Abraham [1] and Dally [13]). We call these performance models as *open system models*.[3] Perhaps the work most related to this paper are by Johnson [16], and Adve and Vernon [2]. Both work model a multithreaded multiprocessor system as a closed system to capture the subsystem interactions. Adve and Vernon analyze the performance of interconnection networks (in a multithreaded multiprocessor system) with pipelining and virtual channels, and assume that the workload characteristics are embedded in the message rate sent to the network.

We construct our integrated system model based on a set of assumptions, which are reasonable for current multithreaded multiprocessor systems implementations (like TERA [7] and Alewife [5]). Our model is amenable to *product-form* solution, which makes it easy to analyze especially when the multithreaded multiprocessor system under study is large, yet the model captures the essential aspects for performance prediction. We solve for the performance measures of the closed queuing network using an Approximate Mean Value Analysis (AMVA) [18].[4] We verify the correctness of analysis using a Stochastic Timed Petri Net model of the multithreaded multiprocessor system.

Given architectural and program workload parameters, our model permits us to derive key performance measures— the processor utilization, message rate to the network, and observed network

[3] We use the term "open system" model and "open queuing network" (OQN) model interchangeably, in this paper. Both capture the same behavior of system and OQN is one type of the open system models.

[4] Use of product-form network, and AMVA, allows us to apply heuristics to capture the implementation specific behaviors. For example, a local memory access in a processing node may keep the system bus busy during the access. This "simultaneous resource possession" property violates the application of a product form solution, but can be captured through heuristics. We do not report here further discussion on the implementation specific models.

latency. We quantitatively characterize variations in these performance measures, identify the system bottlenecks, provide an insight to the impact of performance related optimizations, and finally, reveal the operating ranges for workload characteristics which yield a high processor utilization.

Our closed queuing network based model enabled us to identify the *feedback* effect of subsystems on the processor performance, and the capacity of the network to deliver messages at a maximum rate (per processor) for a given remote access pattern. Our model helps a user to locate the operating point for a specific workload, and indicate whether the network capacity has been reached, thus provides a guidance on how to tune the workload parameters.

Usefulness of closed system model, like ours, is two-fold— *one*, users can work directly with the program workload and architectural parameters which are familiar, and *two*, near the network saturation, unlike open system models, problems of dealing with high sensitivity of performance to input parameters, do not arise. Contrary to the open system based network studies, our analysis indicates that it is possible that the network latency reduces with a simultaneous increase in the message rate to the network because of an increased locality in the remote access pattern or the use of a faster network! We, then, compare the robustness of our model with three open system models employing feedback to correct errors in their processor performance predictions, and demonstrate their weaknesses and tradeoffs with respect to our model.

Finally, we show through a simple example, how a compiler writer may progressively tune a program workload. We show that the processor utilization increases with an increase in the number of threads, as long as the message rate to the network is below the network capacity, even when the observed network latency for individual accesses experienced per thread is large. Once the capacity is reached, a higher locality in remote access pattern (through data set partitioning) can improve the capacity value, and thereby improving the processor utilization. If processor utilization is still low, then increasing the thread runlength improves the processor utilization.

Organization: The rest of the paper is organized as follows. In the next section, we outline our multithreaded multiprocessor system model, present the multithreaded execution model, and discuss the assumptions made to develop the analytical model of the multithreaded multiprocessor system based on closed queuing network. In Sections 3 to 9 we show how to evaluate the performance of our multithreaded multiprocessor system using the results derived from the closed queuing network model. In Section 4 we validate the analytical results using Stochastic Timed Petri Nets (STPN) simulations. In Section 10 we discuss the related work. Finally in Section 11, we present the concluding remarks.

2 Multithreaded Execution Model

Our MMS consists of processing elements (PE) connected through a 2-dimensional torus (as shown in Figure 1). Each PE has a multithreaded processor, and a part of distributed shared memory. A PE is interfaced to a switch on the IN. Each PE contains three components: a *processor*, a *memory module* and a *switch* on the IN (as shown in Figure 2). We model the MMS based on Closed Queuing Networks (CQN). Nodes in the CQN model represent the components of a PE and edges represent their interaction through messages (see Figure 3). We discuss the model in more detail, below.[5]

[5]While the analytical model is fairly simple, detailed descriptions accompanying the notations are for the sake of completeness.

We assume that the application program exhibits similar behavior at each PE, and that the load is evenly distributed (like a Single-Program-Multiple-Data, SPMD, model) [20]. The application program is a set of partially ordered threads. A thread is a sequence of instruction followed by a memory access. A thread repeatedly goes through the following sequence of states– *execution* at the processor, *suspension* after issuing a memory access and after arrival of response, *ready* for execution. Threads interact through memory locations.

We now describe the components of a PE as shown in Figure 2, and discuss the assumptions in their analytical modeling. We use simple models of the subsystems and show their impact on the system performance.[6] All symbols are summarized in Table 4 in the Appendix B.

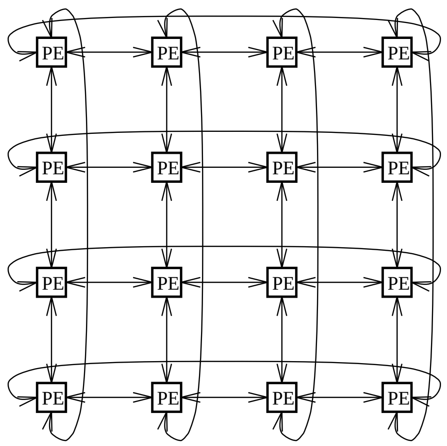


Figure 1: 4×4 Multiprocessor with 2-dimensional mesh

Processor Node:
Each processor executes a set of n_t threads (for example, these threads may be n_t iterations of a `doall` loop). The number of instructions within a thread (including the issue of a memory access) is the *runlength* of a thread, and its average is denoted by R. After issuing a memory access (whether local or remote), the processor context switches to another ready thread. C denotes the *context switch time*. λ_{net} denotes the rate of messages exchanged between the processor and the IN during the program execution.

In the CQN model, a processor is modeled as a *processor node*. Ready threads in a processor are serviced in First Come First Served (FCFS) discipline. Their service time is exponentially distributed with a mean value R. That is, a thread on average runs for R time units before it is suspended for a memory access. We assume that threads do not migrate, so threads at a processor i belong to a class i in the CQN model.

Processor utilization, is defined as the fraction of time a processor performs useful work, and is denoted by U_p. Idling for a memory response or context switching is not considered as useful work.

Memory Node:
When a processor issues a shared memory access, let p_{remote} be the probability of a remote access

[6]The model for each subsystem can be improved to suit the requirements of a performance evaluation study.

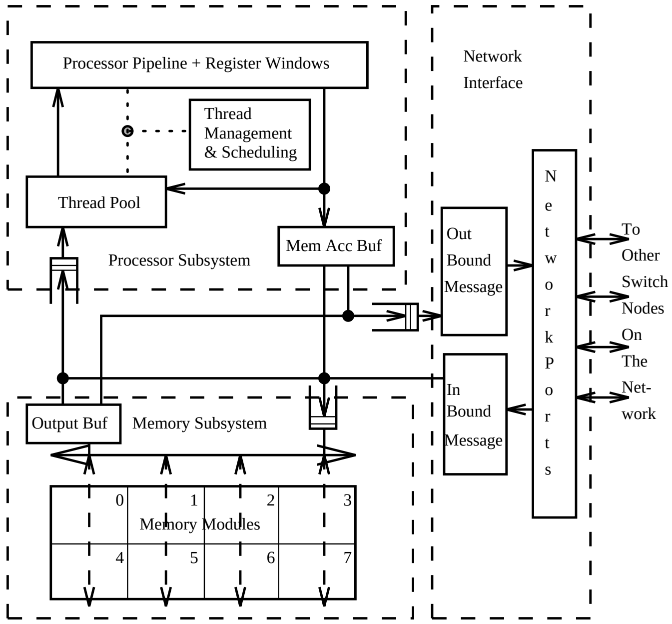


Figure 2: A Processing Element

(i.e. $1 - p_{remote}$ is the probability of a local access). Let the memory latency, L, be the time for accessing a local memory module (without queuing delay) and *observed memory latency*, L_{obs}, be the latency with the queuing delay at the memory.

A memory module is modeled as a *memory node* which services requests in FCFS discipline. The service time is exponentially distributed with L time units as the average value. For requests from a thread at processor i to memory at node j, $em_{i,j}$ denotes the *visit ratio*[7]. The value of $em_{i,j}$ depends on the distribution of remote memory accesses across the memory modules. We assume a geometric distribution for remote access pattern. With a *geometric distribution*, the probability of a memory access to a remote memory module at a distance of h hops from the local processor, is p_{sw}^h/a, where p_{sw} is the locality parameter and a is a normalizing constant. As the maximum distance in an MMS increases (with increase in network size), a approaches $\frac{1}{1-p_{sw}}$. A low value of p_{sw} embodies a higher locality in memory accesses. The average distance traveled by a remote access is:

$$d_{avg} \;=\; \sum_{h=1}^{d_{max}} \frac{p_{sw}^h}{a} \times h, \quad \text{where } d_{max} \text{ is the maximum distance.}$$

We focus on the geometric distribution because it captures locality in remote accesses. A similar access pattern has been studied by Agarwal [3]. Also a recent study of communication profile of workloads on a 32-node message-passing system suggests that a strong locality is indeed exploited in practice (see Figure 4 in [26]). We show that performance of a system with an enhanced locality

[7]The *visit ratio* for a subsystem at a node j for a class i of threads (i.e. executing on processor at node i) in a closed queuing network is the number of times a thread belonging to the class i requests an access to that subsystem at the node j, between two consecutive executions of that thread at the processor. Note that the *visit ratio* can be a fraction.

(like the geometric distribution) is better than with less locality. This also confirms with the impact of locality reported by Johnson [16].

IN Switch Node:
IN is a 2-dimensional torus with k PEs along each dimension. Figure 1 shows an example of an IN with 16 PEs. Figure 2 shows the organization of a PE. We assume that each processor is interfaced to the IN through an *inbound* switch and an *outbound* switch.

The *inbound* switch accepts messages from the IN and forwards them either to the local processor or to other switching nodes for their destination PE. The *outbound* switch sends messages from the host processor on the IN. A message from a PE enters the IN only through the *outbound* switch.

The time taken by a message from its entry on the IN till its exit to the destination PE is called the network latency. A message requires S time units for routing at each switch on the IN. S_{obs} is the one-way network latency with the queuing delays. In the CQN, we model the switch as two separate nodes, *inbound* and *outbound*. We assume that each node has a single server with an exponential service time distribution with mean value S.[8] The effect of message length on service time is included in S. A switch node interfaces its local PE with four[9] neighboring switch nodes. The visit ratios $ei_{i,j}$, from processor i to inbound switch j, are calculated as the sum of the remote memory accesses which pass through the switch j. The visit ratio $eo_{i,j}$ for the outbound switch is same as $em_{i,j}$.

We make the following assumptions in our CQN model to simplify its analysis: (1) Every thread in a processor that is waiting for a memory request will eventually get its request in finite time. So there is no deadlocking of threads waiting for memory requests due to finite resources. (2) There are only finite number of active threads in a processor at any instant. This helps to avoid deadlocking at network switches due to finite buffer capacities of real network switches. These two assumptions are reasonable for practical systems like TERA [7] and Alewife [5].

The CQN model shown in Figure 3, has a *product-form* solution (see Baskett *et al.* [9] and Reiser [21]). As we will see, this simple model serves our objective of providing a good insight into the performance of MMS.

Performance Measures :
Obtaining performance measures of our CQN model, accurately is computationally intensive. The state space to be considered for a *product form* solution [21] is enormous. So we prefer the use of *Approximate Mean Value Analysis (AMVA)* [18]. With n_t threads on each processor in the system, for each class i of threads AMVA evaluates :
(i) the arrival rate λ_i of threads to (and from) processor i; (ii) the waiting time $w^*_{i,m}$ at each node m; and (iii) the queue length $n^*_{i,m}$ at each node m. The AMVA algorithm is outlined in Appendix A. Based on λ_i, $w_{i,m}$, service times and visit ratios, we obtain the following performance measures.

1. *Observed Network Latency* : The average network latency S_{obs} includes the queuing delay and service time at each switch. So S_{obs} for a processor at node i, is the sum of waiting time

[8] Though this is a simple switch model, such switches are found in commercial systems, e.g. Intel's iPSC/2 and iPSC/860. In both systems, a switch operates in one direction at a time while other links are idle. Our switch model performs similar to other analytical models for IN [3], at moderate to high load, if the service rate is scaled according to the number of servers. This is because at high loads, the waiting time dominates the network performance more than the pipelining. The details are omitted in this paper.

[9] This number is dependent upon the topology of the interconnection network.

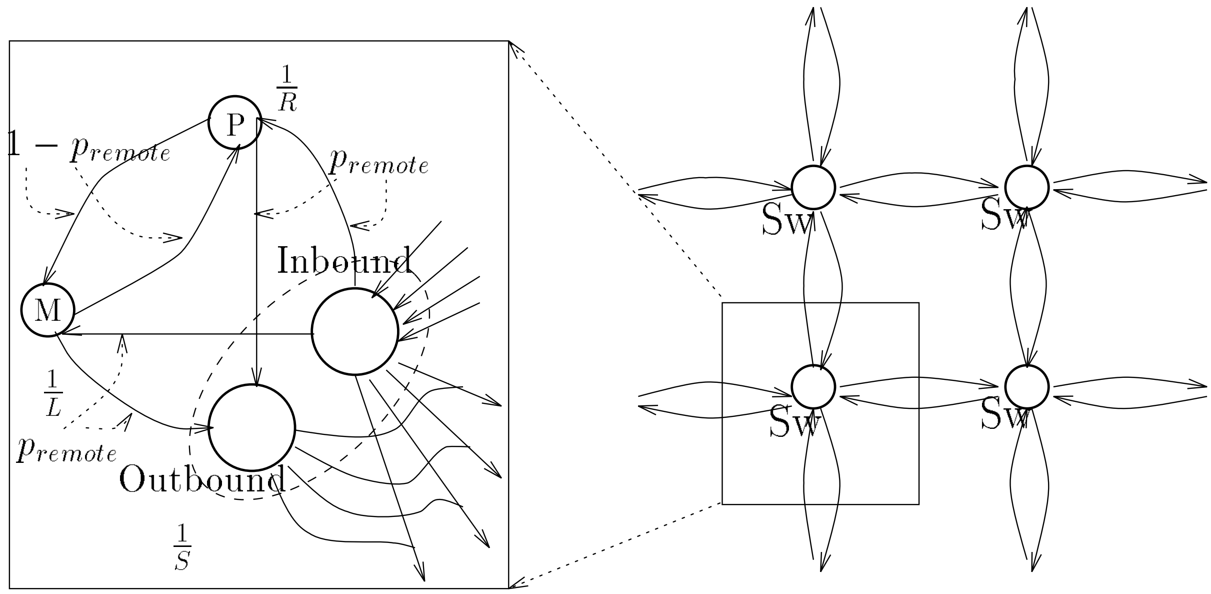


Figure 3: Queuing Network Model

at a switch node (weighted by the visit ratio of a class i thread to that switch node) over all the switches in the IN:

$$S_{obs} \quad = \quad \sum_{j=1}^{P}(w^{*}_{i,j,I} \times ei_{i,j} + w^{*}_{i,j,O} \times eo_{i,j}) \tag{1}$$

where $w^{*}_{i,j,I}$ and $w^{*}_{i,j,O}$ are waiting times at inbound and outbound switches.

2. *Message Rate to the Network*: λ_{net} is the average message arrival rate from a processor i to the IN. λ_i is the arrival rate of threads, and a fraction p_{remote} is sent to the IN. So,

$$\lambda_{net} \quad = \lambda_i \times p_{remote} \quad = \frac{n_t \times p_{remote}}{total\ wait\ time\ for\ a\ class} \tag{2}$$

3. *Processor Utilization*: U_p is obtained from the arrival rate of threads λ_i at processor i and their runlength R:

$$U_p \quad = \quad \lambda_i \times R \tag{3}$$

Using the above CQN model, we analyze the performance of processor and IN subsystems in Section 3. In Section 4, we present the details of a simulation model based on Stochastic Timed Petri Nets (STPN) used for the validation of some of the analytical results.

3 Results

In this section, we present the highlight of the results and observations (to be detailed in Section 5, 6, 7 and 8) when using our analytical model to address the performance issues of multithreaded architectures. To illustrate our results, without loss of generality, we analyze the MMS described in Section 2 as a case study with default values of some workload and architecture parameters as presented in Table 1.

The highlights of the results are as follows.

- We show that, for a given architecture and a given program workload, our analytical model permits a user (programmer or compiler writer) to derive a key performance parameter of interest— the processor utilization U_p. Furthermore, our model provides a quantitative characterization how the processor utilization varies with the program workload parameters (e.g. number of threads per processor, thread runlength, locality (p_{remote} and p_{sw})) as well as the architecture parameters (e.g. the switch delay and memory access time). In particular, our model explains the significant changes in U_p between operating regions, and backup our model with a back-of-the-envelope bottleneck analysis.

- We show that, for a given architecture and a given program workload, our analytical model allow us to study the network performance by deriving the network message rate as well as the network capacity — another key performance parameter of interest. We provide a quantitative characterization how network performance varies with workload and architecture parameters. We explore relation between network performance and processor utilization, and identify where the system bottleneck lies with a particular application program.

  For a user (compiler writer) our model helps to identify the operating region where the operating point of a specific program workload lies— e.g. whether the network capacity has saturated— thus providing a guidance on how should the workload parameters be tuned to improve the performance. If, after the workload tuning, the processor utilization is still low while the network is already saturated, then this suggests that network is a bottleneck for this application. For a system architect, this application provides a test case where the system design is not well balanced.

- We demonstrate how to apply our model to analyze the variation of observed network latency with respect to the changes in program workload and architecture parameters. We examine the latency tolerating aspect of a multithreaded system quantitatively based on our analytical model. Our results point out that higher utilization may sometime be achieved by adjusting the number of threads while the observed network latency has increased for each thread. This happens as long as the network has not reached its capacity limit.

- We demonstrate the usefulness of closed system performance model to users (compiler writers and system architects) — they can work directly with program workload parameters and architecture parameters which are familiar. In particular, when the system is operating in a region where the limit of network capacity is approached, working with an open system model of the network has the disadvantage of dealing directly with the high sensitivity of performance to input parameters such as λ_{net}. Such problems do not arise when a closed system model is employed. We also compare the robustness of our model with open system models. We present three scenarios where an open system model may be applied to estimate the overall performance (such as U_p), and demonstrate their weaknesses and tradeoffs with respect to closed system model.

We present the results as follows. In Section 4, we validate some of the analytical results from our model using Stochastic Timed Petri Net (STPN) simulations. In Section 5, we characterize the processor utilization with model parameters, and highlight operating range of workload parameters to achieve high U_p. In Section 6, we analyze the network performance (λ_{net}), show the feedback effect of network subsystem on the processor performance, and illustrate the importance of network capacity in tuning the model parameters for high processor performance. A large network latency S_{obs}, is considered as a fundamental cause of performance drop in single-threaded systems [8], so in Section 7, we investigate the performance behavior of S_{obs}, and its impact on U_p. In Section 8, we demonstrate the usefulness of using our integrated system model. We also demonstrate the robustness of estimating U_p by analyzing a closed queuing network model, and comparing with other approaches based on open system models. Finally in Section 9, through a simple example we show the usefulness of our results for optimization of workload parameters to achieve high performance.

While all the analytical results are derived from the queuing model discussed in Section 2, we explain significant transitions in performance behavior at various places using simple bottleneck analysis.

In Table 1, the value of S indicates that a message length is 10 times the width of interconnection link, which is commonly observed on real machines. The base value of S_{obs} is 27.33 time units.[10] U_p value indicates the processor busy time when n_t is 1 and thread is executed locally. We do not report the effect of context switching on utilization, which limits the maximum U_p to $\frac{R}{R+C}$ as shown in [23, 4]. In our results, R also embodies the effect of context switch value. Finally, the trends we have reported for an MMS with 4×4 mesh, are also observed for larger size systems (we have tested up to an MMS with 10×10 mesh).

Workload Parameters				Architecture Parameters			Base Output Parameters	
n_t	p_{remote}	R	$p_{sw}(\Rightarrow d_{avg})$	L	S	k	$U_p(\%)$	S_{obs}
8	0.5, 0.8	10	0.5($\Rightarrow$ 1.733)	10	10	4	50	27.33

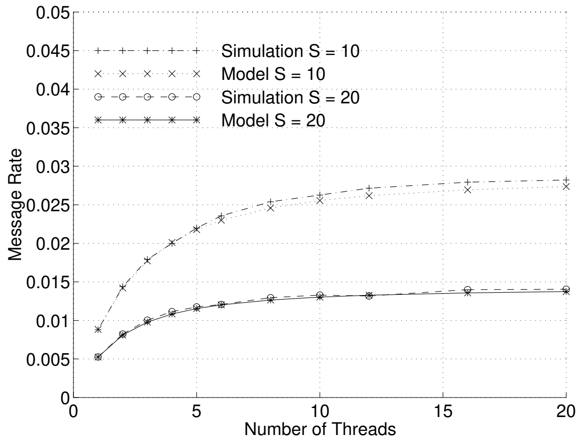
Table 1: Default Settings for Model Parameters

4 Validation

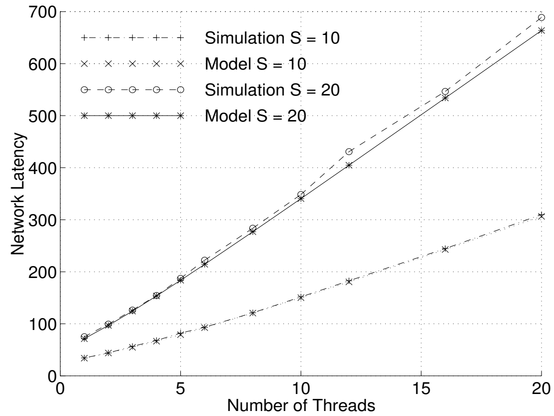
We validate some of the results obtained from the analytical model, using the simulations of Stochastic Timed Petri Net (STPN) model for the MMS. The assumptions made for the STPN model are same as those for the analytical model. While such a validation by no means is complete, it tests the accuracy of the analysis. Since it is difficult to obtain performance measures at a steady state of program execution, and without operating system interventions, for network and processor especially in an MMS, most network performance models report these validations [1, 2, 3, 13].

[10] For comparison, a pipelined network will have a base value of 11.733 time units. However, with contentions, the advantage of pipelining on S_{obs} is quickly lost.

We simulated the STPN model, and also solved the analytical model for the following values of parameters: $p_{remote} = 0.5$, and $S = 10, 20$. Each simulation was carried out for 100,000 time units, thus each transition fires from 50 to 5000 times, depending upon the parameters. For variations with respect to n_t, Figure 4(a) shows the values of λ_{net}. Initially λ_{net} increases with n_t and reaches close to saturation by $n_t = 6$. For $S = 10$, the model predictions are almost identical to the simulation results. For $S = 20$, the model predictions are within 2% of the simulations. Further, model predictions in both cases are slightly lower than the simulations.



(a) Message Arrival Rate



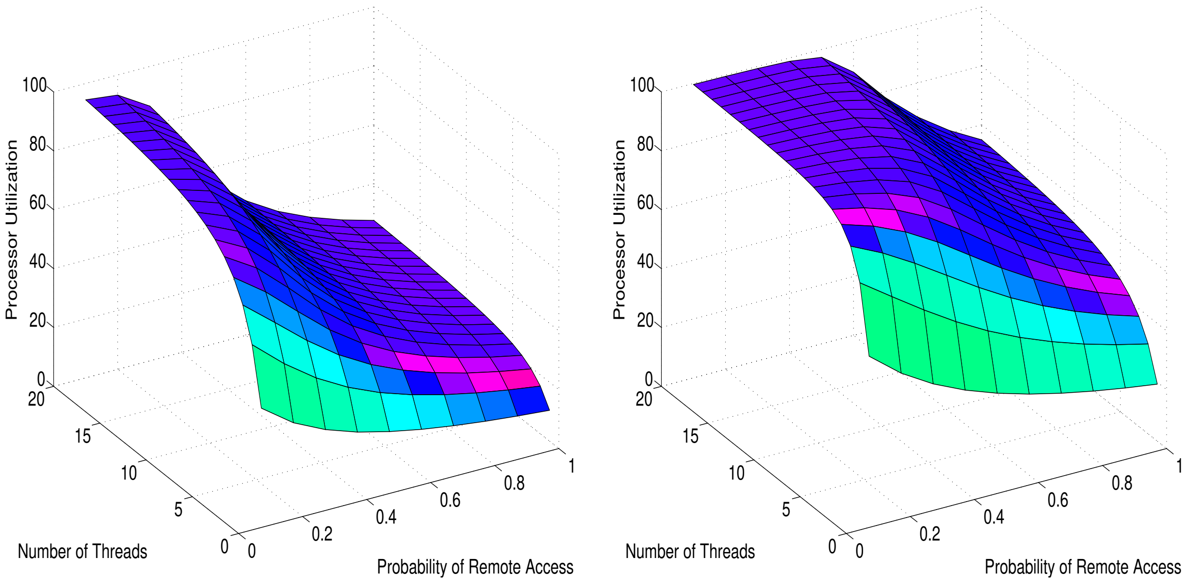
(b) Network Latency

Figure 4: Model and Simulations

Figure 4(b) shows the variations in S_{obs} for the same set of parameters. The effect of n_t is a linear increase in S_{obs}. While S_{obs} obtained from simulations match closely with the analytical results for $S = 10$, the values are close within 5% for $S = 20$. We also studied a change in the service time distribution for memory access time (L), from exponential to deterministic, only for the STPN simulations, and found that S_{obs} values were still within 10% of original model prediction. This indicates that the small error in prediction is because of the variance in service time distributions, and demonstrates the robustness of the model.

5 Processor Utilization

In this section, we show how to apply our integrated system model to study the processor utilization U_p — a key performance parameter of interest. Our underlying objective is to explore the effectiveness of multithreading, i.e. *for what values of workload parameters is U_p high?* Section 5.1 illustrates how to compute U_p for a given program workload. Then we present a quantitative analysis how U_p varies with the program workload parameters (e.g. number of threads per processor, thread runlength, locality (p_{remote} and p_{sw})) as well as the architecture parameters (e.g. the switch delay and memory access time). Section 5.2 explores the significant changes in U_p during transitions between operating regions. Using our model, we identify critical values of certain program workload parameters at which such changes occur, and verify them via an intuitive back-of-the-envelope calculation based on bottleneck analysis.



(a) $R = 10$

(b) $R = 20$

Figure 5: Effect of n_t and p_{remote} on U_p

5.1 Model Parameter Characterization

Given the input workload parameters such as the runlength of a thread, number of threads at a processor, fraction of accesses serviced remotely, and remote access distribution, we solve the analytical model using AMVA (shown in Appendix A) for architectural parameters in Table 1. For a thread executing on a processor i, AMVA yields the waiting time at each service node (say, a processor), and the arrival rate (λ_i) at the processor i. Since the processor is kept busy for duration R by a thread, the processor utilization is $\lambda_i \times R$ (see Equation 3). Variations in input model parameters may affect the waiting time at each service node, thereby changing the values of λ_i and U_p.

Let us consider the effect of changes in workload parameters n_t and p_{remote} on U_p (Figure 5). We observe that a decrease in p_{remote} (i.e. more accesses are serviced locally), yields a higher U_p. Intuitively, a smaller fraction of messages are sent to the network at low p_{remote} values, so few threads suffer additional wait time due to network latency S_{obs} experienced for their responses. Thus processor receives the responses faster, and is less likely to idle. We also observe from Figure 5 that U_p increases with an increase in n_t. For a large n_t (say ≥ 5), responses to memory accesses are received before the processor runs out of threads which are ready for execution. Thus the two effects, together, yield a high U_p.

Low value of U_p occurs when either n_t value is small or p_{remote} value is large. In the first case (low n_t), the processor has less number of threads to execute on when the remote requests are serviced. In the second case (high p_{remote}), we expect the contention on the network to be high. Note that for the parameter values in Table 1, U_p values decrease when p_{remote} is increased beyond 0.2 for $R = 10$ (and 0.6 for $R = 20$), however large n_t may be. In Section 5.2, we analyze this transition in performance behavior. Later in Section 6 and Section 7, we will investigate the following questions:

- How does the network performance affect the processor performance?
- What is the performance behavior of network latency when processor utilization is high?

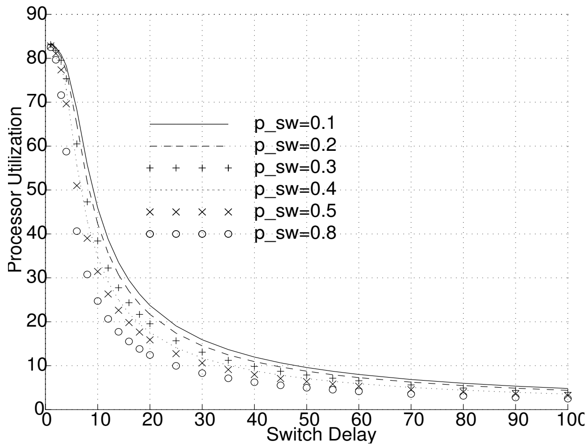


Figure 6: Effect of S on U_p

Now we consider the effect of network related parameters (S and p_{sw}) on U_p as shown in Figure 6 for p_{remote}=0.5 and $n_t = 8$. U_p value decreases when either S or p_{sw} increases. An increase of S means that a message suffers larger delay on each network switch, so a *suspended* thread expecting this response, has to wait for a longer duration. The result is a lower U_p value. Similarly, when p_{sw} is increased (i.e. the locality is decreased) a message travels longer distance on the IN. This *feedback* due to delay for responses from the IN shows that rather than a large n_t (in this case 8), a high U_p depends on the rate at which the messages are returned by the system resources. We also note two points from Figure 6: *First*, for a very fast IN (i.e. low S), the locality has no impact on U_p, because the delay for a message on the IN is negligible compared to that on other system resources (say, memory). *Second*, effect of the locality on U_p is at most the order of the ratio of d_{avg} for the remote access patterns i.e. $\frac{U_p(at\ p_{sw}=0.1)}{U_p(at\ p_{sw}=0.8)} < 2$ at $S = 10$ (we will see in detail with respect to λ_{net}, in Section 6). Even at very high S values, we observe that this condition holds true. The second observation confirms the impact of locality shown by Johnson [16].

Figure 7 shows the effect of R and L on U_p, when p_{remote} is 0.8 (we choose a high p_{remote} to accentuate the effect on U_p). Observe two halves of U_p surface divided by a hypothetical $R = L$ plane. When $R \geq L$, U_p is high, otherwise U_p is low. In general, a high R/S yields high U_p even when L is high. An increase in R has two effects:

1. Since for each execution of thread, the processor is kept busy for longer duration at higher R, U_p values are higher, as observed from a linear increase with R in Figure 7. Observations from Figure 5 also support this claim, where for higher p_{remote}, U_p values for $R = 20$ are twice of those for $R = 10$. Reasoning for the claim is based on Equation 3, where we note that U_p linearly increases with R till λ_i is significantly affected by R.

2. The processor issues memory requests at a reduced rate (for same values of n_t and p_{remote}), so the critical value of p_{remote} up to which U_p remains high, increases. For example, critical p_{remote} increases from 0.2 at $R = 10$ to above 0.6 for $R = 20$ in Figure 5.

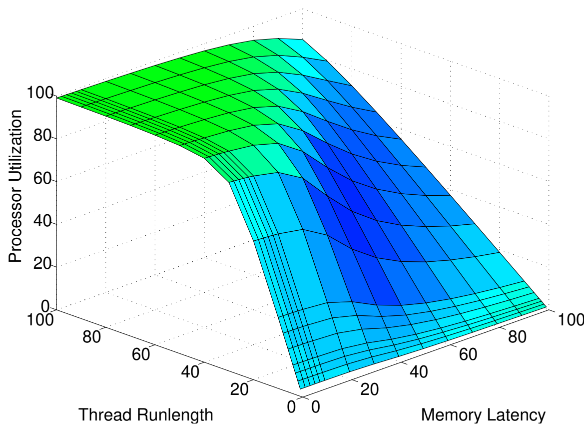


Figure 7: Effect of R and L on U_p

5.2 Analysis of Performance Transitions

Preceding section showed that processor performance varies widely with various parameters. For example, U_p values are high till p_{remote} is lower than 0.2, for $R = 10$ (Figure 5), and U_p values drop there after. To compute this critical p_{remote} up to which U_p remains high, let us apply a back-of-the-envelope bottleneck analysis at the processor node.[11] From a processor node (see Figure 3), a remote access travels $2d_{avg}$ hops for a round trip on the IN, and spends $2S$ time units to get on/off the IN. The remaining fraction of accesses, i.e. $1 - p_{remote}$, is serviced locally. Thus the message rates at the processor node are:

$$\begin{array}{rll} \textit{message rate from processor} & \leq \textit{local memory service rate} & + \textit{message rate from network} \\ \dfrac{1}{R} & \leq \frac{1-p_{remote}}{L} & + \dfrac{1}{2(d_{avg}+1)S} \\ p_{remote} & \leq 1 + \left(\frac{L}{2(d_{avg}+1)S} - \frac{L}{R}\right) & \end{array} \tag{4}$$

Till the critical value of p_{remote} is reached in Equation 4, the processor node remains a bottleneck, i.e. it never runs out of threads for execution, so U_p remains high. Critical value of p_{remote} is 0.18, when $p_{sw} = 0.5$ and $R = 10$ (as confirmed by Figure 5(a)). Note that the critical p_{remote} value depends on various parameters, including R, but does not depend on n_t. For example, increasing the thread runlength to 20 reduces the rate at which the processor issues memory requests, so the critical p_{remote} rises to 0.7 (also confirmed by Figure 5(b)).

When p_{remote} is increased beyond the critical value, U_p value diminishes irrespective of n_t. Such a condition arises because of a high S (or L) value which makes the network (or the memory) the bottleneck.

At critical value of p_{remote}, the service times are balanced (i.e. $p_{remote} = 0.18$ when R, L and S are 10), such that on average three threads (or accesses) on three units–processor, memory, and network switch– are serviced. Figure 5 also shows that 5–8 threads per processor are sufficient to achieve most of the gain in U_p. So remaining 2–5 threads (or accesses) are essentially required in the

[11]Note that the bottleneck analysis confirms only the critical input parameter values at which transitions occur, but for accurate computation of performance measures, we need to solve the integrated system model.

waiting queues to tolerate the differences in service times and their distributions. An imbalance in service times (R, L and S) leads to a distribution of threads (or accesses) based on the service times, so the performance gains are realized up to a smaller value of n_t. These n_t values are independent of other parameters and system size, as long as the MMS uses only one network switch per processor.

5.3 Summary

In this section, we showed how to compute U_p given workload and architectural parameters, characterized the processor performance with model parameters, and analyzed the significant performance transitions. We summarize the results on U_p below:

1. A high U_p can be achieved by increasing n_t as long as the fraction of remote accesses (p_{remote}) is below a critical value, determined by thread runlength, memory latency, switch delay and remote access pattern.
2. With 5 to 8 threads per processor, most of the performance gain due to a high n_t is achieved.
3. A high R increases the critical value of p_{remote} up to which U_p remains high. A high R also keeps processor busy for longer duration during each execution of the thread, thus increasing the U_p.

To optimize workload parameters (for compiler writers) and architectural parameters (for system architects), it is important to understand the performance of other subsystems as well, especially in the operating region of MMS for which U_p values are low.

6 Message Rate to the Network

In this section, we apply our model to study the network performance parameter of interest—λ_{net}, the message rate to the network. In Section 6.1, we show how to compute the message rate to the network. We also compute the *network capacity*, the value at which λ_{net} saturates. Since λ_{net} can saturate even at a low value of n_t, the performance behavior of an MMS near the network saturation is essential for performance tuning (unlike in single-threaded system [3]). In Section 6.2, we quantitatively characterize the network performance behavior with workload and architecture parameters. We also explore the question raised in Section 5.1: *How does the network performance affect the processor utilization?* This information helps us identify whether the network is a bottleneck on an application.

6.1 Capacity of the Network

Given the input workload parameters and the architectural parameters (as mentioned in Table 1), we solve the analytical model using AMVA. For a thread executing on a processor i, AMVA yields the waiting time at each service node, and the arrival rate of threads λ_i at the processor i. Since each thread issues one memory access, and a fraction p_{remote} are sent to remote memory modules, the message rate to the network is $p_{remote} \times \lambda_i$ (see Equation 2).

A processor allows multiple outstanding memory accesses, so λ_{net} rises close to saturation even with n_t as low as 6 (see flat, dark surface in Figure 8 shown for $R = 10$). We call the saturation value of λ_{net}, i.e. the maximum number of messages delivered per processor per unit time, as the *capacity* of the network for a given access pattern, and denote it by $\lambda_{net,saturation}$.

Let us compute the capacity as follows. A message travels d_{avg} hops on the IN, and so does its response. With a delay of S time units on each switch, overall duration for a message on the IN is 2 d_{avg} S time units. So the maximum rate at which a message is received by a processor (=capacity of the IN) is:

$$\lambda_{net,saturation} = \frac{n_t \times p_{remote}}{total\ wait\ time\ for\ a\ class} = \frac{1}{2\ d_{avg}\ S} \quad (= 0.029,)\ \text{at}\ p_{sw} = 0.5 \tag{5}$$

Value of $\lambda_{net,saturation}$ obtained from integrated system model (Equation 2) is shown as the dark, flat surface at $\lambda_{net} \approx 0.029$ in Figure 8, which confirms with the intuitive computation of Equation 5.[12] Note that if there were only one switch (instead of an *outbound* and an *inbound* switch), the contention for that switch also would have affected the network capacity, restricting it to $\frac{1}{2\ (d_{avg}+1)\ S}$. Because each message may travel more than one hop on the IN, the capacity of the IN for a given access pattern is typically lower than the bandwidth of the IN (per processor). However the maximum value of the capacity is the bandwidth.

To compute p_{remote} at which the IN saturates, let us consider how a PE communicates with the rest of MMS, and apply a bottleneck analysis at the *inbound* switch. The processor and memory at a PE send the outgoing messages through the *outbound* switch and receive the incoming messages through the *inbound* switch. At steady state, the rate of outgoing messages from a PE equals the rate of incoming messages. Additionally, an *inbound* switch also forwards messages from neighboring switches to other neighboring *inbound* switches. This additional traffic (reflecting the contention) changes according to the locality in remote memory access pattern. Capacity of the IN is reached when the throughput at an *inbound* switch reaches $\frac{1}{S}$. Thus balance of message rates, at an inbound switch, yields the critical p_{remote} value for which the capacity is reached:

$$\begin{aligned} &(maximum\ throughput\ out\ of\ a\ switch) \qquad -(forwarded\ messages\text{: contention})\\ &\qquad = (rate\ of\ messages\ from\ processor\ and\ memory\ to\ local\ switch)\\ &\frac{1}{S} - \frac{\sum_{\forall r \& r \neq i} ei_{r,i}}{S} = \frac{p_{remote}}{R} + \frac{p_{remote}}{L} \end{aligned} \tag{6}$$

At $p_{sw} = 0.5$, Equation 6 yields $p_{remote} = 0.3$ for λ_{net} to saturate (as confirmed by results of the queuing model in Figure 8). Right hand side of Equation 6 assumes that before the network switch becomes a bottleneck, n_t is large enough (say, ≥ 6) to keep the processor and memory busy.

6.2 Model Parameter Characterization

Figure 8 shows the effect of changes in workload parameters n_t and p_{remote} on λ_{net}, when $R = 10$. An increase in either n_t or p_{remote}, increases λ_{net}. When p_{remote} is held constant, λ_{net} increases with n_t, because more memory accesses (local and remote) are requested by the processor. Saturation at high value of n_t indicates that either memory (at low p_{remote}) or network switches (at high p_{remote}) become a bottleneck. When n_t is a constant, λ_{net} increases with p_{remote}, because a larger fraction

[12] This expression in Equation 5 can also be derived from OQN models for the IN, e.g., Agarwal [3]. Note that the pipelining of network links has no effect on value of $\lambda_{net,saturation}$. Also, $\lambda_{net,saturation}$ is the value at which the actual traffic in the "actual traffic vs. attempted traffic" saturates [13].

of accesses is diverted to remote memory. λ_{net} saturates at $\lambda_{net,saturation}$ when $p_{remote} \geq 0.3$ and n_t is high (say, ≥ 6). $\lambda_{net,saturation}$ is 0.029 (flat, dark surface in Figure 8). Analyzing U_p values from Figure 5(a) with λ_{net} behavior from Figure 8, we note that once the capacity is reached, U_p values are low:

- No increase in n_t helps to increase U_p. So *what happens to additional messages at high n_t?* These messages are queued at the switches (we will elaborate this point in Section 7).

- At a fixed value of n_t, λ_{net} remains at its saturated value even with increase in p_{remote}. But U_p decreases, because a larger fraction (p_{remote}) of accesses experiences a higher latency (S_{obs}) in addition to the memory latency L_{obs} (at local or remote node).

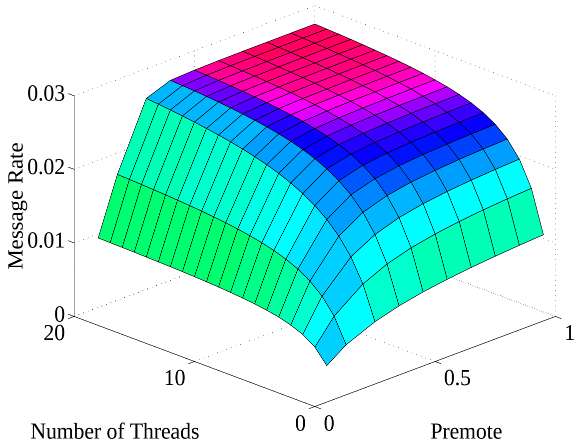


Figure 8: Effect of n_t and p_{remote} on λ_{net}

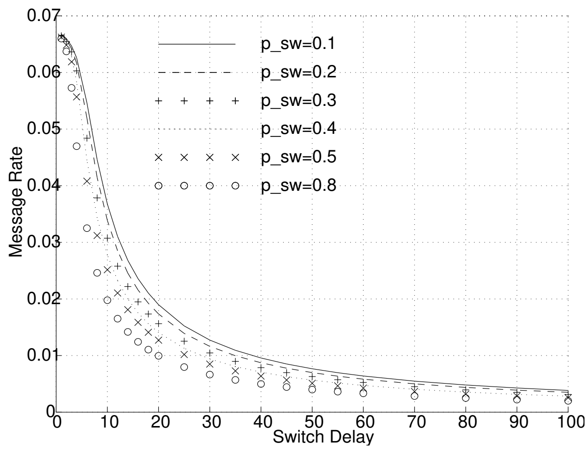


Figure 9: Effect of S on λ_{net}

Effect of network related parameters (S and p_{sw}) on λ_{net} is shown in Figure 9 for $p_{remote} = 0.5$ and $n_t = 8$. An increase in S increases the waiting and service time for a message at each network switch. The corresponding thread (at a processor) is suspended for a longer duration. This *feedback* through the duration of suspension of a thread, reduces λ_{net} at a higher S. Thus an S value changes the capacity of the IN. Equation 5 also demonstrates how λ_{net} depends on the waiting time at all nodes. For an OQN model, the message rate is an input parameter, and there does not exist any feedback mechanism to correct changes in λ_{net} because of subsystem interactions.

Note from the processor performance in Figure 6 that since the network is close to its capacity (at $n_t = 8$), U_p values track λ_{net} values, with respect to S and the locality. We expect this behavior, because in Equation 2 and Equation 3, all parameters except λ_i are constant.

The *feedback* is also prominent in the effect of locality on λ_{net} (see Figure 9). A decrease in p_{sw} increases the locality, because a message travels a shorter distance (d_{avg}) on the IN. Faster traversal of a message through the IN increases λ_{net}, and hence the capacity. Note two observations from Figure 9:

1. A fast network (i.e. low S) has a higher capacity, and delivers messages too quickly, so the gain due to locality is negligible.

2. Since $\lambda_{net,saturation}$ depends on d_{avg} (from Equation 5), the improvement in λ_{net} due to an increased locality is at most of the order of the reduction in d_{avg}. Figure 9 shows that for $S > 6$, $\frac{maximum\ \lambda_{net}}{minimum\ \lambda_{net}} < \frac{d_{avg}\ at\ p_{sw}=0.8\ (\Rightarrow 3.17)}{d_{avg}\ at\ p_{sw}=0.1\ (\Rightarrow 1.11)}$. Implications on U_p values are similar, based on Equation 2 and Equation 3, as confirmed by our observations, earlier in Section 5.1.

Figure 10 shows the effect of thread runlength R and memory latency L on λ_{net}, when p_{remote} is 0.8. Note the following from λ_{net} values (and corresponding U_p values from Figure 7):

- A decrease in R reduces the time spent by a thread at the processor between successive memory accesses so U_p decreases, but λ_{net} increases. When values of R and L are low (for example, from $4 \times S$ down to $2 \times S$), λ_{net} rises rapidly to 93% of saturation value.

- The λ_{net} surface is symmetric with respect to R and L. For remote accesses, a thread alternates between execution at the processor, and a service at the remote memory. So the lower of R and L values dominates the rate at which the accesses are sent on the network. However, for U_p values, apart from this rate, R/L ratio is also important.

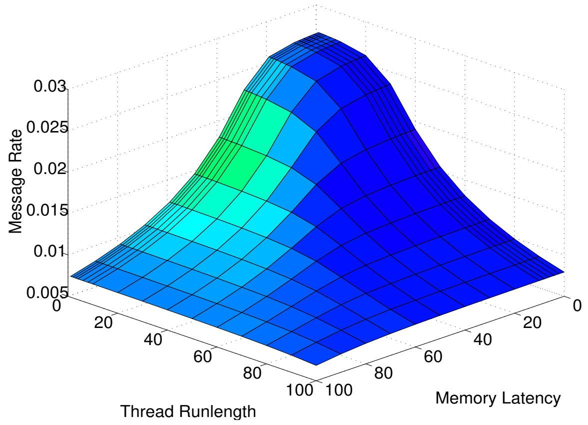


Figure 10: Effect of R and L on λ_{net}

6.3 Summary

We summarize the results on λ_{net}, capacity of an IN, and their impact on U_p, below:

1. An increase in p_{remote} as well as n_t (for their low values) increases λ_{net} because the number of messages on the IN increases. At low p_{remote}, U_p is mainly governed by n_t. Once the capacity of the IN is reached, λ_{net} saturates with n_t as well as p_{remote}. However U_p does not improve improve with n_t, but decreases sharply with p_{remote}.

2. Capacity of the IN is determined by S and d_{avg}. A faster network, i.e. low S, has a higher capacity. Also, a higher locality in accesses increases the capacity. The higher the capacity, the higher can be the achieved U_p.

3. R and L have an identical effect on λ_{net}. An increase in λ_{net} occurs when both R and L are low. But U_p value depends on $\frac{R}{L}$ and $\frac{R}{S}$ ratios.

7 Network Latency

We showed in Section 6 that λ_{net} has a significant impact on the processor performance. Also, the latency for an access affects the performance of a single-threaded system, and has been considered as one of the fundamental causes for decrease in the performance of multiprocessor systems [8], so we now investigate the following question: *How does the network latency affect high processor utilization?* First, we show how to compute the observed network latency, S_{obs}. Second, we analyze the variations of S_{obs} with workload and architectural parameters. We also discuss the latency tolerating aspect of an MMS.

7.1 Parameter Characterization

Given the input workload parameters and the architectural parameters, we solve the analytical model using AMVA to obtain the waiting time at each service node, for each class of threads. The *observed network latency*, S_{obs}, for an access by a thread executing on the processor i, is the waiting time for class i threads (accesses) at each switch node. Since one access does not visit all the nodes in the network, the waiting time of each switch node is weighted by the visit ratio with which the switch node is visited by a remote access for class i thread (see Equation 1). For every change in model parameters, above computation is performed to reflect the change in S_{obs} value.

Figure 11 shows how S_{obs} varies with changes in workload parameters (n_t and p_{remote}). Notice that S_{obs} increases with n_t. The rate of increase changes significantly with the value of p_{remote}. Using Equation 4 and Equation 5, we consider three distinct parts of the S_{obs} surface— `i`- $p_{remote} \leq 0.18$, `ii`- $0.18 \leq p_{remote} \leq 0.3$, and `iii`- $0.3 \leq p_{remote}$. For these three parts, we also consider the impact on the processor performance (from Figure 5(a)).

$p_{remote} \leq 0.18$: Since the processor is a bottleneck, the messages are sent to the IN at a lower rate than the IN can deliver ($\lambda_{net,saturation}$). So S_{obs} is close to its unloaded value, irrespective of an increase in n_t. Note that remaining fraction of accesses is serviced locally (and the observed memory latency, L_{obs}, is high). U_p values are high when n_t is large (say ≥ 5).

$0.18 \leq p_{remote} \leq 0.3$: Beyond $p_{remote} = 0.18$, number of messages on the IN (and λ_{net}) increases, hence the contention increases. So S_{obs} increases with n_t. Close to $p_{remote} = 0.3$, S_{obs} is very high, but so is U_p. The drop in U_p from its maximum value is not significant at high n_t.

$0.3 \leq p_{remote}$: Recall from Section 6 (Equation 6) that the capacity of IN is reached for $p_{remote} \geq 0.3$, thus the *inbound* switch becomes a bottleneck. When n_t is a constant, S_{obs} remains constant at a high value with respect to p_{remote} because the number of messages on the IN (either waiting or being routed) becomes a constant. However, since at higher p_{remote} a larger fraction of messages suffer additional delay of S_{obs} on the IN, U_p values decrease.

When n_t is increased, the number of threads (accesses) waiting at the network switches increase, so S_{obs} increases. But the network switches are the bottleneck, so λ_{net} does not change, and hence no increase is observed in U_p values.

Let us consider sample operating points of the MMS excerpted from Figure 5, Figure 8, and Figure 11, shown in Table 2. Refer to lines which are marked 'a' and are in bold-face. For $n_t = 2$ at $p_{remote} = 0.5$, S_{obs} is 43.8. For $n_t = 4$ at $p_{remote} = 0.5$, S_{obs} is 54.8. And for $n_t = 8$ at $p_{remote} = 0.3$, S_{obs} is 82.7. Notice that S_{obs} values for these cases are in increasing order, but so are the respective U_p values— 28.4%, 40.2% and 72.7%. However when λ_{net} is close to saturation value (≈ 0.029), even a large n_t does not improve U_p. This can be noted from U_p values for $n_t = 4$ and 8, at $p_{remote} = 0.8$.

Thus in summary, (1) a high U_p can be achieved on an MMS by increasing n_t, even though the resulting S_{obs} may be large, *as long as* the network capacity is not reached. (2) Otherwise, even for a small value of S_{obs}, U_p remains low. While item (1) is an intended objective of an MMS, the mentioned condition is not. Item (2) suggests to explore other mechanisms, if the increase of n_t does not improve the performance.

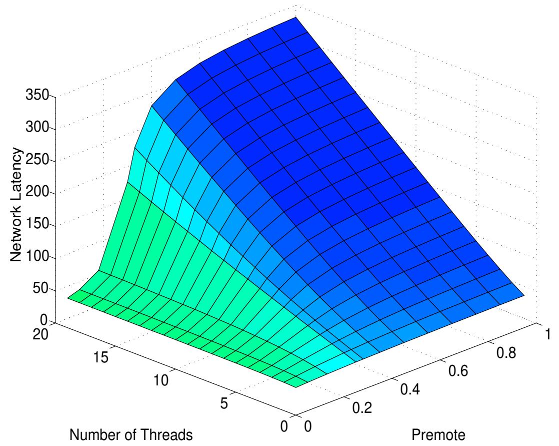


Figure 11: Effect of n_t and p_{remote} on S_{obs}

Effect of network related parameters (S and p_{sw}) on S_{obs} is shown in Figure 12, when $p_{remote} = 0.5$. An increase in S increases the service time at each switch, so a linear increase occurs in S_{obs} (also through feedback effect, U_p and λ_{net} are proportional to $\frac{1}{S}$, as shown in Figure 6 and Figure 9). A similar effect is observed when locality is changed. An increase in the locality (i.e. a low p_{sw}) decreases d_{avg} for each message, hence S_{obs} decreases. So, the messages return faster to the processor, improving the value of U_p.

	p_{remote}	R	n_t	L_{obs}	S_{obs}	λ_{net}	U_p	R	n_t	L_{obs}	S_{obs}	λ_{net}	U_p
	0.0	10	1	10.0	0.00	0.0000	50.00	10	2	15.0	0.00	0.0000	66.67
	0.1			10.9	29.6	0.0037	37.20			15.1	31.8	0.0056	56.30
b	*0.2*			*11.2*	*31.2*	*0.0059*	*29.36*			14.8	36.0	0.0094	46.96
	0.3			11.5	32.4	0.0072	24.13			14.2	39.4	0.0117	39.21
a	**0.5**			11.5	33.9	0.0088	17.67			**13.3**	**43.8**	**0.0142**	**28.44**
	0.8			11.3	35.1	0.0100	12.54			12.3	47.2	0.0157	19.64
	0.0	20	2	10.0	0.00	0.0000	84.53	10	4	25.0	0.00	0.0000	80.00
	0.1			13.0	30.2	0.0039	77.31			24.6	34.2	0.0074	74.07
b	*0.2*			13.1	33.2	0.0070	69.74			*22.7*	*43.5*	*0.0133*	*66.29*
	0.3			13.1	35.9	0.0093	62.30			20.0	53.4	0.0170	56.80
a	**0.5**			12.7	40.4	0.0124	49.44			**16.0**	**54.8**	**0.0177**	**40.21**
	0.8			12.1	44.5	0.0145	36.30			13.5	74.3	0.0213	26.68
	0.0	20	4	15.1	0.00	0.0000	94.68	10	8	45.0	0.00	0.0000	88.89
	0.1			15.8	31.2	0.0046	92.12			44.6	36.1	0.0086	86.26
b	*0.2*			15.9	36.1	0.0089	88.63			*40.7*	*52.7*	*0.0164*	*81.94*
a	**0.3**			15.8	42.1	0.0126	83.80			**31.5**	**82.0**	**0.0218**	**72.74**
	0.5			14.9	55.1	0.0175	70.18			19.0	120.6	0.0246	49.18
	0.8			13.3	67.5	0.0203	50.70			14.5	134.8	0.0251	31.45

Table 2: Network Latency Tolerance, with $R = 10$ and $R = 20$

Effect of thread runlength R and memory access time L on network latency (see Figure 13 for p_{remote} =0.8) shows that both parameters have similar effect on S_{obs}. Note that R includes the context switch time, C, as well. S_{obs} is significantly affected when both R and L are low. For example, S_{obs} rises sharply (even compared to λ_{net}, from Figure 10) from 50 to 140 time units, when both R and L decrease from 40 to 2 time units. When the thread runlength is high, or the memory access time is large, λ_{net} has a small value, and S_{obs} is close to its unloaded value due to less contention. Comparing U_p values from Figure 7, we note that to achieve high U_p, both $\frac{R}{S}$ and $\frac{R}{L}$ ratios should be high.

7.2 Summary

From the effect of various parameters on S_{obs} we show that:

1. An increase in n_t and p_{remote} increases S_{obs} to a large value, and U_p increases simultaneously. However, once the capacity of IN is reached, S_{obs} increases linearly with n_t, while U_p remains constant. In the network saturation, for an n_t, with an increase in p_{remote}, S_{obs} saturates at a high value, but U_p decreases.

2. S_{obs} increases linearly with S, and with a decrease in locality. On the other hand, due to the feedback effect of S_{obs} on the triggering of *suspended* threads, U_p (and λ_{net}) decrease.

3. Only when both R and L are low, S_{obs} is high (similar to λ_{net} behavior), otherwise S_{obs} is near its unloaded value. U_p on the other hand, depends on $\frac{R}{L}$ and $\frac{R}{S}$ ratios.

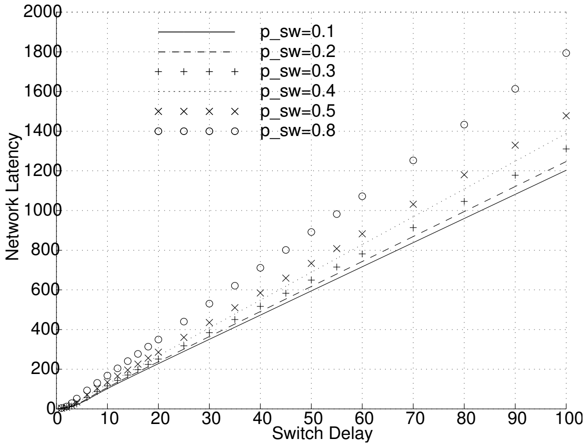


Figure 12: Effect of S on S_{obs}

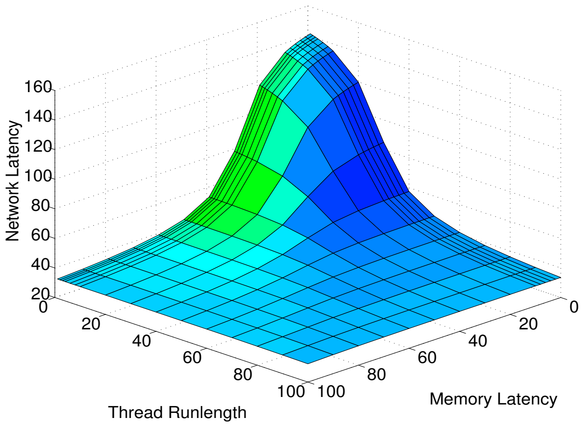


Figure 13: Effect of R and L on S_{obs}

Above results, in particular (1), suggest that unlike single-threaded systems, the processor performance is not significantly affected by S_{obs}, but instead is affected by λ_{net}.

8 Usefulness and Robustness

In Section 8.1, we demonstrate the usefulness of closed system performance model to users (compiler writers and system architects)— they can work directly with program workload parameters and architectural parameters which are familiar. In particular, when the system is operating in a region where the limit of network capacity is approached, working with an open system model of the network has the disadvantage of dealing directly with the high sensitivity of performance to input parameters such as λ_{net}. Such problems do not arise when a closed system model is employed. In Section 8.2, we also compare the robustness of our model with open system models. We present three scenarios where an open system model may be applied to estimate the overall performance (such as U_p), and demonstrate their weaknesses and tradeoffs with respect to closed system model.

8.1 Usefulness of Closed System Model

In this section, we compare the prediction of network latency using closed system model to that using open system model. One advantage of using closed system model is that users (compiler writers and system architects) can work directly with the familiar program workload parameters and architectural parameters. We show the second advantage that near the network saturation, the network latency is highly sensitive to λ_{net} (an input parameter of the OQN model) compared to n_t and p_{remote} (input parameters of our closed system model). Lastly, we show that through the use of feedback effect how the closed system model provides a broader perspective of the system performance than an OQN model.

Open system characteristics for IN are reported in literature by plotting observed network latency S_{obs} against message rate λ_{net}, e.g. Abraham [1], Agarwal [3] and Dally [13]. To draw a fair comparison of robustness in prediction, we plot such characteristics. During a program execution on a multiprocessor system, λ_{net} cannot be directly controlled, since it results from the interaction of various subsystems. So we consider two ways to vary λ_{net}: First uses workload parameters– n_t and p_{remote}– (Figure 14), and second uses network related parameters– S and p_{sw}– (Figure 15).

IN performance characteristics (in an OQN model) use λ_{net} as the input parameter (see [3, 13, 1]), so we obtain similar characteristics in Figure 14(b) to show the variation of S_{obs} versus λ_{net}. Following procedure shows how the IN performance characteristics of OQN model can be obtained using our integrated system model.

Figure 14(a) and Figure 14(c) show the effect of workload parameters on S_{obs} and λ_{net}, respectively. Using these two figures, we show a plot of S_{obs} versus λ_{net} in Figure 14(b), similar to the open system characteristics. To obtain each point in Figure 14(b), we consider each pair of n_t and p_{remote} values, read λ_{net} value from Figure 14(c), and read S_{obs} value from Figure 14(a). For example, at $n_t = 10$ and $p_{remote} = 0.4$, λ_{net} and S_{obs} values are 0.0265 and 125, respectively. Notice that each curve from Figure 14(a) and Figure 14(c) represents a fixed value of p_{remote}, and that it gets mapped on to a small part of the curve in Figure 14(b)– e.g. at $p_{remote} = 0.1$, λ_{net} changes from 0.0035 to 0.0095 only. We obtain a complete plot of S_{obs} against λ_{net} by overlapping projections for various values of p_{remote} and n_t.

Now let us study the sensitivity of the prediction of S_{obs} using OQN model (Figure 14(b)), and using integrated system model (Figure 14(a)). Recall from Section 6 that to use the MMS efficiently, the IN performance may be pushed to the network *capacity*. Consider that our assumption of workload requires a modification, and we need to know the network performance (S_{obs}) at the new workload. Let λ_{net}, the input parameter of the OQN model, be changed by 15% (say from 0.025 to 0.02875). We note that the prediction of S_{obs} changes from 100 to 290– a change of 190%! This shows that S_{obs} is highly sensitive to λ_{net}, near the network saturation. In reality, this change in λ_{net} can be brought only if we change n_t from 7 to 20, by 186% (at $p_{remote} = 0.5$). The reason for high sensitivity of S_{obs} to λ_{net} in the prediction using an open system model is the contention factor. The contention is reflected by a term $\frac{\rho}{1-\rho}$ (refer to [17] and [3]), where ρ is the utilization of a link (in our case, the switch). Switch utilization, ρ, is the sum of all service demands at a switch as given below:

$$\rho \quad = \textstyle\sum_{\forall r} S \times \text{visit ratio for class } r \times \text{throughput of class } r \quad = S \sum_{\forall r} (ei_{r,i}\ \lambda_{net,r}) \qquad (7)$$

$$\frac{\rho}{1-\rho} \quad = \frac{\lambda_{net}(S \sum_{\forall r} ei_{r,i})}{(\lambda_{net,saturation}-\lambda_{net})(S \sum_{\forall r} ei_{r,i})} = \frac{\lambda_{net}}{\lambda_{net,saturation}-\lambda_{net}} \quad \text{Contention factor} \qquad (8)$$

Near network saturation, λ_{net} approaches $\lambda_{net,saturation}$, so S_{obs} is highly sensitive to λ_{net}.

Let us consider the same change (15%) in the input parameter n_t, of the CQN model. n_t changes from 7 to 8. Resulting values of S_{obs} and λ_{net} change from 100 to 114–by 14%, and from 0.0250 to 0.0257– by 3%, respectively. We note that S_{obs} varies at most linearly with n_t (Figure 14(a)).

Above discussion brings out following advantages of using a CQN model for the network performance prediction, for users (compiler writer and system architects):

1. User is more familiar with an input parameter like n_t or p_{remote} than λ_{net}. For example, *how many iterations of a* `doall` *loop are to be exposed at a time? a la* loop-bounding[11].
2. The network performance (S_{obs}) is highly sensitive to λ_{net}, an input parameter of open system model, than to n_t or p_{remote}, input parameters of closed system model. This robustness to n_t and p_{remote} is also helpful to ascertain the buffer requirements at the network switches. For example, a small error in λ_{net} near the network saturation can lead to significant error in buffer requirements at the switches. Further in Section 8.2, we also show how this robustness is helpful for processor performance prediction.

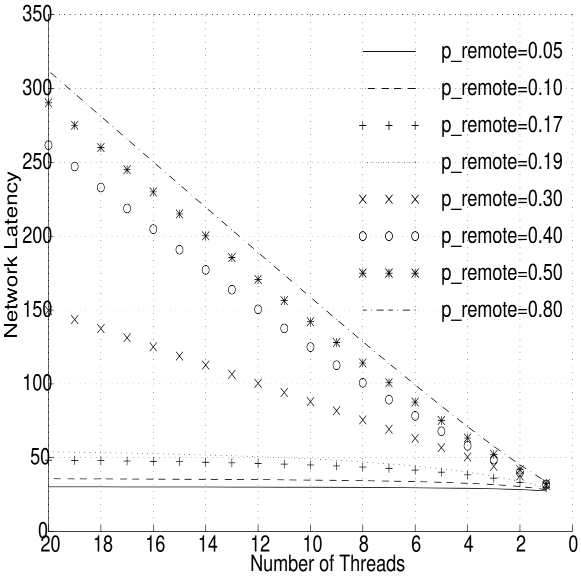
Now we discuss the variations in S_{obs} and λ_{net} using network related parameters, S and p_{sw}. Figure 15 shows these variations, and has been obtained similar to the way we obtained Figure 14. Value of p_{remote} is 0.5, and that of n_t is 8. Consider S_{obs} and λ_{net} values when S is 10. When we increase the locality (i.e. decrease the value of p_{sw}), d_{avg} decreases, so S_{obs} for a message decreases (Figure 15(a)). Since the response to a remote request is received faster, the corresponding thread is ready for execution earlier. Due to this *feedback* effect (also discussed in Section 6), λ_{net} increases (Figure 15(c)). Plotting S_{obs} and λ_{net} together (Figure 15(b)), notice the contrast to Figure 14(b), λ_{net} increases while S_{obs} decreases. Such characteristics are not reported in OQN model based studies [1, 3, 13]. But this broader perspective of the performance behavior due to locality variations is essential to the performance tuning techniques used by compiler writers.

Thus capturing the subsystem interaction helps our integrated system model to predict the effect of program workload on the IN performance (i.e. λ_{net} and S_{obs}). On the other hand, OQN studies focus on IN performance, independent of the system and program execution model.

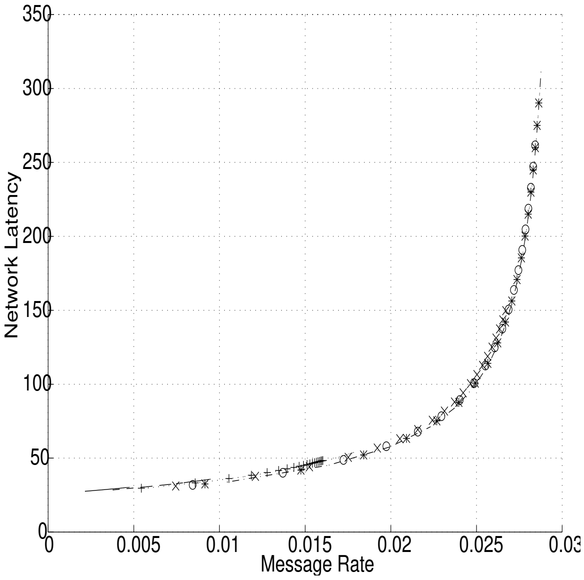
8.2 Robustness of Processor Performance Prediction

We observed in preceding sections (6 and 7) that the *feedback* effect plays an important role in performance of an MMS, and in Section 8.1 that open system model fails to capture this effect. Implications of these results for a user (compiler writer or programmer) are as follows:

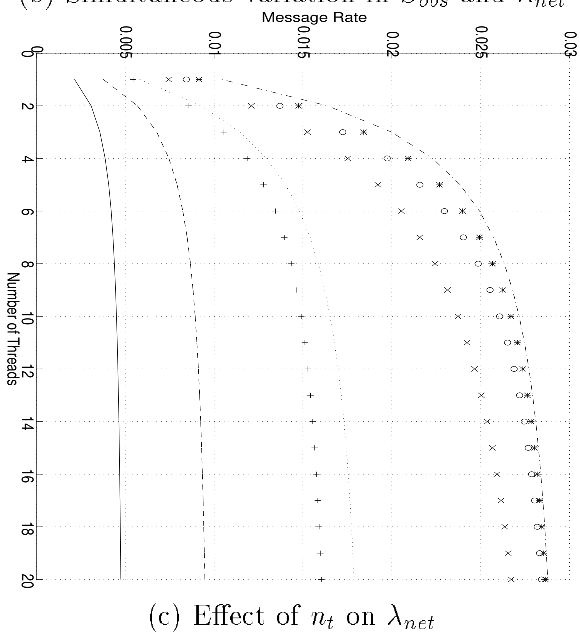
> What is the difference in the processor performance prediction using *closed* and *open* system models? And in case of a significant difference in the performance prediction:
>
> Given an open system characteristics of the network, can we obtain the performance of an MMS by coupling it with a model for the processing node of an MMS?



(a) Effect of n_t on S_{obs}

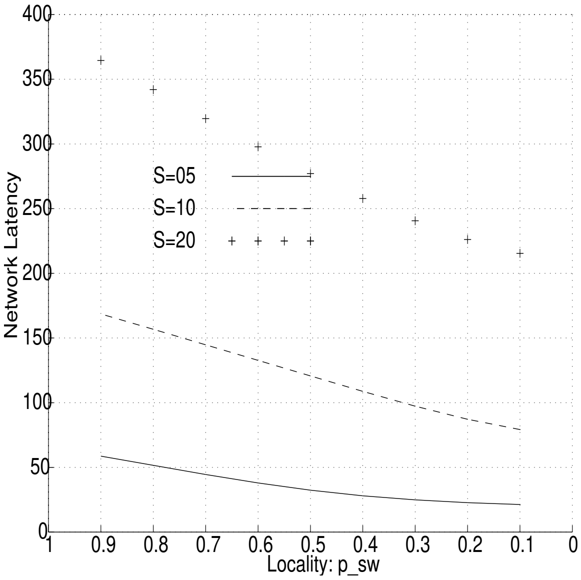


(b) Simultaneous variation in S_{obs} and λ_{net}

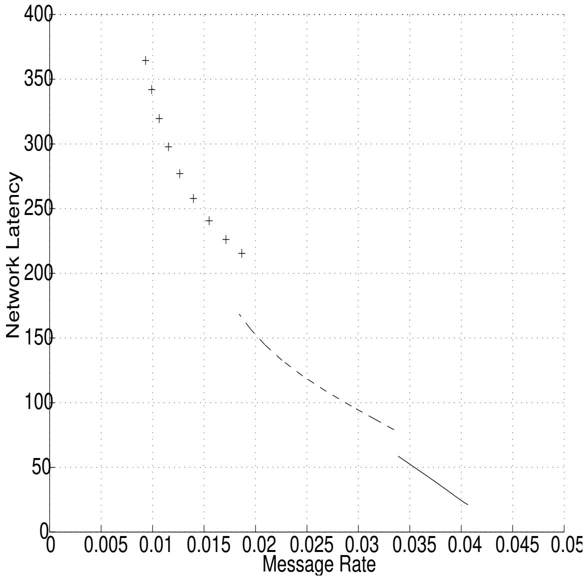


(c) Effect of n_t on λ_{net}

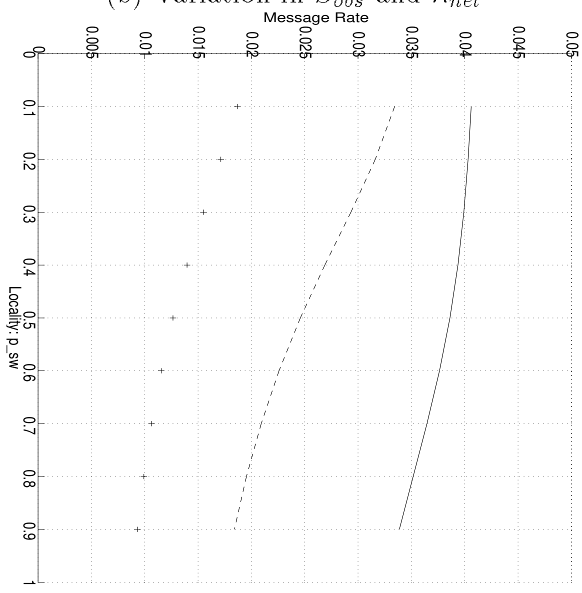
Figure 14: Operating Point in a Multiprocessor Network



(a) Effect of p_{sw} on S_{obs}



(b) Variation in S_{obs} and λ_{net}



(c) Effect of p_{sw} on λ_{net}

Figure 15: Impact of Locality on Operating Point

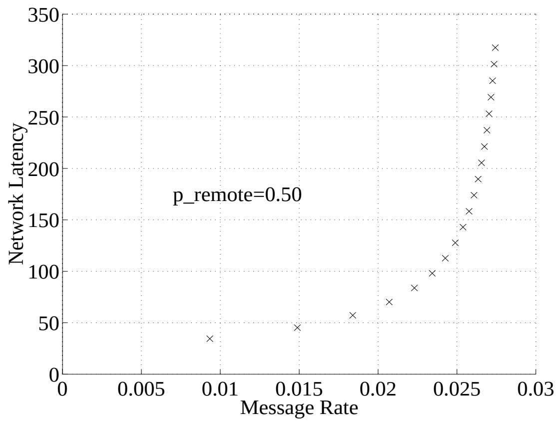


Figure 16: S_{obs} and λ_{net} variations at $p_{remote} = 0.5$

We consider three successively refined models to predict the processor performance, based on open system models, and compare their results with our closed system model. In all three cases, we assume that Figure 16, which is derived from Figure 14(b), is the representative open system characteristics of network in the MMS under study. These characteristics represent the function f, such that $S_{obs} = f(\lambda_{net})$.

- First model uses a naive approach that the network is moderately loaded. For this load (λ_{net}), S_{obs} is read from the open system characteristics. Using S_{obs}, we compute the time for each thread (access) to return to the processor as ($R + L + 2\ p_{remote}\ S_{obs}$), and obtain U_p ($= \frac{n_t\ R}{cycle\ time\ for\ a\ thread}$).
- Second model assumes initial values for λ_{net} and S_{obs}, and proceeds as the first model to compute new values for $\lambda_{net}(= \frac{n_t\ p_{remote}}{cycle\ time\ for\ a\ thread})$ and U_p. New λ_{net} is used to obtain S_{obs} from Figure 16. The process repeats till the values in successive iterations are close enough (see Figure 17).
- Third model uses open system models of the processing node, and the network subsystem. To capture the feedback effect, the output parameter of the processing node model, λ_{net}, is fed to the input parameter of the network model; and the output parameter of the network model, S_{obs}, is fed to the input parameter of the processing node model. Solving these models simultaneously yields the values of performance measures— λ_{net}, S_{obs}, and U_p.

In the first model, we assume a load on the network (λ_{net}), and obtain S_{obs} from Figure 16. We compute the cycle time for a thread (access) to return to the processor as ($R + L + 2\ p_{remote}\ S_{obs}$), and obtain U_p ($= \frac{n_t\ R}{cycle\ time\ for\ a\ thread}$). Similar approach of an assumed, fixed network load is used by Boothe [10], and Thekkath [25], to study various aspects in multithreading. This naive method works well when $n_t = 1$. Assume $S_{obs} = 27.33$, the un-loaded value. So U_p is 21.1%, whereas closed system model yields 17.67%. At higher n_t values, predictions differ widely. For example, at $n_t = 6$, let us assume λ_{net} to be 0.025, i.e. $S_{obs} = 135$. So U_p is 38.7%. Predictions of closed system model for λ_{net}, S_{obs}, and U_p are 0.023, 92.5, and 46.0%, respectively. Different choice of λ_{net} (e.g., 0.02 and 0.015), leads to widely different U_p values (69.3% and 94%, respectively). The better the prediction of network performance, i.e. λ_{net} and S_{obs}, the greater the accuracy of U_p. The most important weakness of this model is an absence of a feedback to correct the assumption about the network load (λ_{net}). A caution for the use of this simple model is that the assumed load should be higher than actual, otherwise predicted value of U_p may *wrongly* exceed 100%.

In the second model, we provide a feedback through changes in S_{obs} and λ_{net} values to correct the error in performance prediction. Figure 17 shows the steps in this model. Starting with a no-load value of S_{obs}, λ_{net} is computed as $\frac{n_t}{(R + L + 2\ p_{remote}\ S_{obs})}$, and U_p as $\frac{n_t\ R}{(R + L + 2\ p_{remote}\ S_{obs})}$. Using new λ_{net}, S_{obs} is obtained as $S_{obs} = f(\lambda_{net})$, the function depicted in Figure 16. Iterative process repeats till the new S_{obs} value is close to its old value. In our examples, this method converged up to $n_t = 2$, when linear interpolation is used for performance points in between those shown in Figure 16. Predictions at $n_t = 2$, for S_{obs}, λ_{net}, and U_p, are 46.1, 0.0149, and 29.89%, while the closed system model predicts 43.8, 0.0142, and 28.44%. We note that:

- The prediction of U_p value is within 5% of that using closed system model, when network is unsaturated.
- For higher values of n_t, the iterative process fails to converge because of the steep slope of open system characteristics (see *almost* vertical line at $\lambda_{net} \approx 0.029$, which prevents accurate computation of the network operating point (S_{obs})).
- It is difficult to study the effect of network related parameters (like the study using S and p_{sw} in Section 8.1) with only the knowledge of open system characteristics in Figure 16.

Given an open system characteristics for the interconnection network, $S_{obs} = f(\lambda_{net})$

0- **Assume** S_{obs} to be a no-load value, say S_j, where j is the iteration number.
1- **Compute** λ_j and U_p, based on n_t, p_{remote}, and S_j.
2- For new λ_j, **obtain** $S_j = f(\lambda_j)$.
3- **If** $((S_j - S_{j-1}) > tolerance)$
then go to **step 1**;
else exit.

Figure 17: Feedback Algorithm

Third model uses analytical expression for the network performance, and works better than above simplistic feedback algorithm. Johnson [16] reports this approach. Validity of this approach is based on the property of *aggregation* for queuing network [12], which states that any subnetwork (of a product-form queuing network) can be exactly aggregated into a single station with a load-dependent service rate. When remote access pattern is independent of processing nodes in networks like k-ary n-cube, the analytical expression for the network performance can be obtained.

There are two difficulties in the third model:

- *First,* when the remote access pattern (or the program workload) for all the processing nodes is not the same, an analytical expression for the network is difficult to obtain.

- *Second,* when no single representative link can be found to analytically capture the performance behavior of all the links in the network, the performance model can be difficult to solve. This typically happens in an hierarchical network, for example, interconnection network of the MANNA system [15] which consists of an hierarchy of crossbar switches to achieve large configurations. In such a case, more than one representative links may be present. So a set of simultaneous equations (one for each representative link, and one for the processing node) are needed to obtain the performance measures of the MMS. To our knowledge, such a study has not been conducted.

Another drawback specific to Johnson's model is that for program workload exhibiting high locality (i.e. distance traveled per dimension, $k_d < 1$), the analytical expression is incorrect— a fact also observed by other researchers [22]. For this locality, the model assumes that there is *no* contention. Let us consider the parameters in Table 1, for which $d_{avg} = 1.733$ i.e. $k_d = 0.866$. Our results show that with $n_t = 8$, S_{obs} and U_p are 120.6 and 49.18% respectively. However, like Johnson's model, if no contention is considered on the network (i.e. $S_{obs} = 27.33$), then U_p value significantly rises to 78.85%, an error of 60%. Johnson's assumption that the network is unsaturated, so the contention is less, appears reasonable only when n_t is small (say, 1). In this case, U_p values with and without contention are 17.67% and 20.37%, respectively.

8.3 Summary

In summary, we have shown the usefulness of closed system model for performance prediction of an MMS, over that by an open system model:

1. It is robust in network saturation, and input parameters are predictable (and familiar) to users– compiler writers, programmers and system architects. Robustness in prediction of network latency (and the queue lengths at network switches) helps to ascertain the buffer requirements at the network switches. A small error in arrival rate near saturation can lead to significant error in buffer requirements at the switches.

2. It provides a broader perspective by capturing the feedback effect, e.g. an increased locality in remote accesses decreases S_{obs}, but increases λ_{net}.

3. Comparison with three techniques based on open system models, showed that feedback is essential for accurate processor performance prediction.

   - A simplistic feedback approach for accounting the impact of the network performance on the performance of processing nodes, works well only when the network is unsaturated.
   - We point out that through a use of *aggregation* property of queuing networks, the processor performance can be computed accurately, provided the network performance can be accurately described by an analytical expression. However, such a performance model is difficult to solve in the following cases:
     (1)- when remote memory access patterns of individual, or groups of processing nodes are different;
     (2)- when the network is hierarchical, so more than one network links are needed to analytically capture the performance behavior of all the network links.

9 An Example for Workload Optimization

Now we show through a specific example (see Table 2 excerpted from Figure 5, Figure 8 and Figure 11) how a compiler writer can progressively optimize the workload characteristics to achieve high processor utilization. Our proposed order of steps is in the order of their difficulty in implementation, however a compiler writer may prefer other orders suitable for a particular application.

As an example, consider the values marked '*b*' in Table 2 (Section 7), and are *emphasized.* An increase in n_t from 1 to 4, increases U_p up to 2.3 times (e.g., at p_{remote}=0.2 and R=10). λ_{net} also increases up to 2.3 times (at p_{remote}=0.2). However, a further increase of n_t to 8 yields only a small increase in U_p, when λ_{net} is close to saturation (also observe very high S_{obs} values).

At higher p_{remote} values, U_p values are low. A change of data layout (by placing the data close to computation) can improve the utilization. Table 2 captures this effect by sending less accesses to remote memory (i.e. a decrease in p_{remote}). At each value of n_t, a decrease in p_{remote} yields a higher U_p, while S_{obs} and λ_{net} decrease. However, a certain amount of data sharing is necessary, in most applications, which places a bound on the performance gains achieved by this technique.

If the processor utilization is still low, then thread runlength should be increased. We need techniques such as aggregating the network requests to the same memory from multiple threads (works well for array accesses), increasing the number of instructions between successive memory requests by merging two or more dependent threads (at times, at the cost of parallelism), and using sophisticated register allocation/instruction scheduling techniques (which reduce the register spills). These extra efforts pay off because for higher thread runlength, the memory accesses are fewer, so the contention at the memory and the network are less (see L_{obs} and S_{obs} values). Resulting U_p values are significantly high. Note that $n_t = 4$ at $R = 20$ yields higher U_p values than $n_t = 8$ at $R = 10$. Table 2 also shows that for the application with $p_{remote} = 0.8$ and $R = 20$, the network is still a bottleneck for high performance. Such an application is a test case for system architects for tuning the architectural parameters.

So we propose following optimizations and summarize their impact:

1. Increase n_t to achieve a high U_p. Note that λ_{net}, S_{obs} also increase.
2. For $n_t \geq 5$, λ_{net} is close to saturation, so the performance gain due to a high n_t diminishes even though S_{obs} may not be high. So increase the locality to get a higher U_p.
3. Finally, increase R to improve p_{remote} for which high U_p can be obtained. Simultaneously U_p value also increases. A high thread runlength has more impact on U_p than n_t.

Note that the order of difficulty in each of the above steps changes with the application as well as the compiler. Here we showed benefits of these steps, without discussing the costs associated with their implementations.

10 Related Work

A number of analytical and simulation studies on the performance of multithreaded architectures have been reported in the literature. First group of analytical studies focuses on the processor performance only (like Saavedra-Barrera et al. [23], Alkalaj et al. [6], and Agarwal [4]), while second group studied interaction of various subsystems in an MMS (like Johnson [16], Nemawarkar et al. [19], and Adve et al. [2]).

Saavedra-Barrera et al.[23] and Alkalaj et al. [6] study multithreaded systems using Petri Nets. They analyze the state space of PNs to evaluate the performance. Since contentions at memory and network increase the state space of their models tremendously, the contentions are not studied.

In [4], Agarwal presents a detailed analytical model for processor performance based on cache parameters, thus complementing our study. The model does not consider the feedback of network on the cache miss rates, so the model is applicable where communication latency and cache miss rates are independent, e.g. either latency or cache miss rates are low.

Interconnection network performance has been extensively analyzed using open queuing network (OQN) model in the literature [1, 3, 13]. We showed that an OQN model does not faithfully capture the subsystem interactions, like feedback effect, hence the network performance predictions of these studies were not directly usable for analyzing MMS. However note that OQN models provide an insight to the IN performance, independent of the system, and the program execution model.

The performance model for an IN can be effectively combined with models for other subsystems to capture their interactions. Three studies using closed system models follow this approach [16, 19, 2]. In [16], Johnson develops a closed system model to account for the feedback effect of the IN, and predict the effect of locality in an MMS. He adds an application model (to reflect the computation on a processor) to Agarwal's network model [3]. Johnson's model assumes an unsaturated network and does not take into account the effects of the memory subsystem. His model does not capture the network behavior correctly, if the locality in the remote access pattern is high (specifically $k_d \leq 1$)– a fact also observed by Saavedra [22]. His results show benefits of tuning the workload parameters to exploit the locality, which we have confirmed in this paper.

In our earlier work [19] using a CQN model, we focused mainly on evaluating the subsystem utilizations. In this paper, we have presented a detailed model of the MMS and focused on analyzing the network latency, network capacity, workload distribution and their impact on processor performance. We also studied the robustness of the prediction of performance measures, and compared the benefits of using a closed system model over the techniques employing open system models. Further, we presented these results to serve as a guideline to tune the workload parameters for an MMS.

A recent, independent work by Adve and Vernon [2] also uses a CQN model to analyze the performance of a mesh interconnection network in an MMS. Similarities and differences with our work are as follows:

1. Both models are quite similar (which we expect because of the standard nature of modeling large systems using CQN [21, 18]).

2. Adve and Vernon concentrate on an IN with pipelining and virtual channels. To capture the detailed network model faithfully, they use a different set of approximations for applying AMVA. The result is an increased complexity of their model (albeit a greater accuracy).

   We use a simple network model, which reduces the complexity of the model, and use a well known approximation to apply AMVA.

3. They model the remote memory access pattern using a uniform distribution. In contrast, we use a geometric distribution which captures the effect of locality in remote memory accesses.

4. They focus on effects of architectural features (like pipelining and virtual channels in the IN) on performance, and provide guidelines for their use. They also study the scaling of the number of processors in an MMS.

   In contrast, we show the effect of workload parameters to tolerate long latencies and achieve high processor utilization on an MMS. In particular, we have identified the role of network capacity on the network latency and processor utilization. Our results show how a compiler writer can systematically tune the application workload on an MMS.

In summary, our work, and Adve and Vernon's work complement each other in reinforcing the claim that a CQN faithfully captures various aspects of a large scale MMS.

Three simulation studies also report the performance benefits of multithreading [27, 10, 24]. An early study by Weber [27] shows the differences in performance gains due to multithreading, because of variations in bus traffic. In [24], the effectiveness of multithreading in presence of cache is studied. Their results indicate the need for tuning of workload characteristics –locality, number of threads– to obtain performance gains using multithreading, which is validated by our results. In [10], Boothe shows the benefits of various techniques manipulating the network messages, adjusting the thread runlengths, for multithreading. Our results confirm that these trends are indeed observed. We show that their assumption of a constant network latency does not realistically capture the network performance, so the degree to which the performance benefits are obtained, using the proposed techniques, changes substantially.

11 Conclusions

In this paper, we proposed a performance model for analyzing a multithreaded multiprocessor system. Our integrated system model, based on closed queuing networks, takes into account the behavior of processors, memories and interconnection network, and the interaction between them under various program workload.

We showed that given program workload parameters and architectural parameters, how to derive the key performance measures– processor utilization (U_p), message rate to the network (λ_{net}), and observed network latency (S_{obs}). We applied our model to provide a quantitative characterization of their variations with model parameters, to understand system bottlenecks, and to provide insight to the impact of performance related optimizations.

Our analysis brought out the importance of *feedback* effect of network performance on the processor utilization. Concurrently analyzing the network and processor performance we showed how significant is the *network capacity* to tune the workload characteristics to achieve a high processor utilization. For example, U_p increases with an increase in the number of threads (n_t) as long as the capacity of the IN is not reached, even when S_{obs} is large. Also, the higher the capacity, the higher can be the achieved U_p value.

We demonstrated the usefulness of closed system performance models to users (compiler writers, programmers, and system architects)– they can work directly with the program workload parameters and architectural parameters which are familiar. Added advantage is that near the network saturation, unlike open system models, problems of dealing with high sensitivity of performance to input parameters, do not arise. We also showed that the robustness of our model is helpful in processor performance prediction, in comparison to three scenarios based on open system models to estimate the processor performance.

We believe that our methodology, and the results of our analysis will be helpful for users in tuning workload and architectural parameters to achieve high performance on an MMS.

12 Acknowledgment

This work was supported by MICRONET, Network Centers of Excellence, Canada. We acknowledge the discussions with Prof. R.Govindarajan, and Prof. V.K.Agarwal during the early part of this research. We also acknowledge Prof. A.Chien for his comments and suggestions on various aspects of this work.

References

[1] S. Abraham and K. Padmanabhan. Performance of the direct binary n-cube network for multiprocessors. *IEEE Transactions on Computers*, C-38(7):1000–1011, July 1989.

[2] V.S. Adve and M.K. Vernon. Performance analysis of mesh interconnection networks with deterministic routing. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):225–247, March 1994.

[3] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[4] A. Agarwal. Performace tradeoffs in multithreaded processors. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):525–539, September 1992.

[5] A. Agarwal, B.H. Lim, D. Kranz, and J. Kubiatowicz. April: A processor architecture for multiprocessing. In *Proc. of the 17th Int'l. Symp. on Computer Architecture*, pages 104–114, 1990.

[6] L. Alkalaj and R.V. Bopanna. Performance of multi-threaded execution in a shared-memory multiprocessor. In *Proc. of 3rd Ann. IEEE Symp. on Parallel and Distributed Processing*, pages 330–333, Dallas, USA, December 1991. IEEE.

[7] R. Alverson et al. The Tera computer system. In *Proc. of the 1990 Int. Conf. on Supercomputing*, pages 1–6, Amsterdam, Netherlands, June 1990. ACM.

[8] Arvind and R.A. Iannucci. Two fundamental issues in multiprocessing. Computation Structures Group Memo 226, Laboratory for Computer Science, MIT, 1987.

[9] F. Baskett, K. Mani Chandy, R.R. Muntz, and F.G. Palacios. Open, closed, and mixed network of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, April 1975.

[10] B. Boothe and A. Ranade. Improved multithreading techniques for hiding communication latency in multiprocessor. In *Proc. of the 19th Int'l. Symp. on Computer Architecture*, 1992.

[11] D.E. Culler and Arvind. Resource requirements of dataflow programs. In *Proceedings of 15th Annual International Symposium on Computer Architecture*, pages 141–150. ACM, May 1988.

[12] Y. Dallery and X.-R. Cao. Operational analysis of stochastic closed queuing networks. *Performance Evaluation*, 14:43–61, 1992.

[13] W. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[14] J.B. Dennis and G.R. Gao. Evolution of multithreaded architectures. In Robert A. Iannucci, Guang R. Gao, Robert H. Halstead, Jr., and Burton Smith, editors, *Multithreaded Computer Architecture: A Summary of the State of the Art*, chapter 1. Kluwer, Norwell, MA, 1994.

[15] Gesellschaft für Mathematik und Datenverarbeitung mbH. *MANNA Hardware Reference Manual.* Berlin, Germany, 1993.

[16] K. Johnson. The impact of communication locality on large-scale multiprocessor performance. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 392–402. ACM, May 1992.

[17] C.P. Kruskal and M. Snir. The performance of multistage interconnection networks. *IEEE Transactions on Computers*, C-32(12):1091–1098, Jan 1983.

[18] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

[19] S.S. Nemawarkar, R. Govindarajan, G.R. Gao, and V.K. Agarwal. Analysis of multithreaded multiprocessors with distributed shared memory. In *Proceedings of the Fifth IEEE Symposium on Parallel and Distributed Processing*, pages 114–121, Dallas, Texas, December 1993.

[20] Q. Ning, V. van Dongen, and G.R. Gao. Automatic data and computation decomposition for distributed memory machines. ACAPS Technical Memo 82, School of Computer Science, McGill University, Montreal, Que., March 1994.

[21] M. Reiser and S. Lavenberg. Mean value analysis of closed multichain queueing networks. *Journal of ACM*, 27(2):313–322, April 1980.

[22] R.H. Saavedra, W. Mao, and K. Hwang. Performance and optimization of data prefetching strategies in scalable multiprocessors. *Journal of Parallel and Distributed Computing*, 22:427–448, 1994.

[23] R.H. Saavedra-Barrera, D.E. Culler, and T. von Eicken. Analysis of multithreaded architectures for parallel computing. In *Proc. of 2nd Ann. ACM Symp. on Parallel Algorithms and Architectures*, Crete, Greece, July 1990. ACM.

[24] R. Thekkath and S. Eggers. The effectiveness of multiple hardware contexts. In *Proc. of the 6th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 328–337. ACM, Oct. 1994.

[25] R. Thekkath and S. Eggers. Impact of sharing-based thread placement on multithreaded architectures. In *Proceedings of the 21st International Symposium on Computer Architecture*. ACM, April 1994.

[26] B. VanVoorst, S. Seidel, and E. Barszcz. Profiling the communication workload of an ipsc/860. In *Proceedings of the 8th Scalable High Performance Computing Conference*, pages 221–228. IEEE, May 1994.

[27] W.D. Weber and A. Gupta. Exploring the benefits of multiple contexts in a multiprocessor architecture: Preliminary results. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pages 273–280. ACM, 1989.

A Approximate Mean Value Analysis

We assume the same terminology and notation as described earlier in Section 2. For n_t threads on each of the P processors in the system, the AMVA[13] computes (see Table 3):
(i) the arrival rate λ_i for the threads belonging to each processor i; (ii) the waiting time $w^*_{i,m}$ at each node m; and (iii) the queue length $n^*_{i,m}$ for population vectors $\mathbf{N} = (n_t, \ ..., n_t)$ and $\mathbf{N} - \mathbf{1_i}$. Population vector $\mathbf{N} - \mathbf{1_i}$ indicates that there are $n_t - 1$ threads at processor i, and other processors have n_t threads each. Statistics for population vector $\mathbf{N} - \mathbf{1_i}$ are used to compute the queue lengths and waiting times for n_t-th thread added on to the processor i. Statistics for population vector $\mathbf{N}$ indicate the steady state performance measures when n_t threads are present at each processor.

AMVA makes an initial guess for queuelengths at each node, for P classes of threads, assuming that the thread population in the system is $\mathbf{N}$. From Steps 2 to 5, the algorithm iteratively refines this guess till the maximum difference between queuelengths obtained from two successive iterations is within a *tolerance* value. Step 2 computes the queuelength $n^*_{i,m}$ at each node m, for a class i when the population is $\mathbf{N} - \mathbf{1_i}$. The *approximation* function used here, adds the contribution, to the queuelength for a class i, from a class j according to the number of threads in class j. So only when $i = j$, a scaling of $\frac{n_t - 1}{n_t}$ is required. This queuelength is the number of threads serviced at node m before a newly added thread in class i gets a service, so the waiting time for class i is computed accordingly. In Step 3, the cycle time for a thread in class i is the sum of waiting times at all nodes for a newly added thread in class i. Hence, the throughput for class i is $\frac{n_t}{cycle\ time\ for\ one\ thread}$. Step 4 computes new queuelengths at each node m, for class i threads when the population is $\mathbf{N}$, using the throughput for class i and the waiting time for a class i thread at node m. Finally Step 5 verifies whether the difference between the queuelengths from successive iterations are within the tolerance level.

1- Initialize $\quad n^*_{i,m}(\mathbf{N}) = \frac{N_i}{M}$

2- Calculate at each node m and for each class i

a- $\quad n^*_{i,m}(\mathbf{N} - \mathbf{1_i}) = 1 + \left[\frac{N_i - 1}{N_i} n^*_{i,m}(\mathbf{N})\right] + \sum_{j=1, j\neq i}^{P} n^*_{j,m}(\mathbf{N})$

b- $\quad w^*_{i,m}(\mathbf{N}) = \rho_{i,m}\left[n^*_{i,m}(\mathbf{N} - \mathbf{1_i})\right]$

3- Evaluate for each class i

$$\lambda_i(\mathbf{N}) = \frac{N_i}{\sum_{m=1}^{M} w^*_{i,m}}$$

4- Evaluate the new values for $n^*_{i,m}(\mathbf{N})$ at all nodes m and for all classes i

$$n^*_{i,m}(\mathbf{N}) = \lambda_i(\mathbf{N}) w^*_{i,m}(\mathbf{N})$$

5- If $difference(n^*_{i,m}(\mathbf{N})_{new},\ n^*_{i,m}(\mathbf{N})_{old}) > tolerance$
then go to *step 2* else *exit*

Table 3: AMVA Algorithm

[13] This AMVA algorithm has been excerpted from Lazowska *et al* [18], and suitably modified for our work.

B Symbols and Their Meanings

Parameter	Description
Workload Parameters	
n_t	Number of threads at each processor
R	Mean value of thread runlength
p_{remote}	Probability of accessing a remote memory module
p_{sw}	Probability for a message to travel an extra hop in geometric distribution
System Parameters	
S	Delay for a message at a switch node
C	Context switch overhead
L	Memory latency
k	Number of PEs in a row/column of the mesh
P	Number of PEs in the system
Output Parameters	
L_{obs}	Observed memory latency (with queuing delay)
S_{obs}	Observed network latency
λ_{net}	Rate at which a processor sends messages to the network
U_p	Processor Utilization
ρ	Switch Utilization
Analysis Related and Derived Parameters	
$em_{i,j}$	Visit ratio of thread from processor i to memory module j
$ei_{i,j}$	Visit ratio of thread from processor i to inbound switch at j
$eo_{i,j}$	Visit ratio of thread from processor i to outbound switch j
$\rho_{i,m}$	Visit ratio for thread from processor i at node m $\times$ service rate for class i at node m
M	Number of nodes in the system
N	Maximum thread population, n_t, at each processor in the system (for AMVA)
$N - 1_i$	Processor i having $n_t - 1$ threads while the rest have n_t threads
$n^*_{i,m}$	Queue length at node m, due to threads from processor i
λ_i	Message arrival rate at the processor i
$w^*_{i,m}$	Waiting time (including service time) for thread from processor i at node m
h	Number of hops traveled by a message from its source node
a	Normalization constant for geometric distribution
d_{avg}	Average distance (in hops) traveled by a message on the network

Table 4: Model Parameters