Latency Tolerance: A Metric for Performance Analysis of Multithreaded Architectures

Shashank S. Nemawarkar and Guang R. Gao*
School of Computer Science
McGill University, Montreal, Quebec H3A 2A7 Canada

Abstract

Multithreaded multiprocessor systems (MMS) have been proposed to tolerate long latencies for communication. This paper provides an analytical framework based on closed queueing networks to quantify and analyze the latency tolerance of multithreaded systems. We introduce a new metric, called the tolerance index, which quantifies the closeness of performance of the system to that of an ideal system. We characterize the latency tolerance with the changes in the architectural and program workload parameters. We show how an analysis of the latency tolerance provides an insight to the performance optimizations of fine grain parallel program workloads.

1 Introduction

A multithreaded multiprocessor system (MMS) like TERA [4] and Alewife [3], tolerates long latencies for communication by rapidly switching context to another computation thread, when a long latency is encountered. Multiple outstanding requests for multiple threads at a processor increase the latencies. An informal notion of latency tolerance is that if the processor utilization is high due to multithreading, then the latencies are tolerated [3, 5]. However, there is no clear understanding of the latency tolerance.

Performance of multithreaded architectures has been studied using analytical models [2, 1], and simulations of single- and multiple-processor systems [10, 9, 3]. Kurihara et al.[6] show how the memory access costs are reduced with 2 threads. Our conjecture, however, is that the memory access cost is not a direct indicator of how well the latency is tolerated.

The objectives of this paper are, to quantify the latency tolerance, to analyze the latency tolerance of the multithreading technique, and to show the usefulness of latency tolerance in performance optimizations. An analysis of the latency tolerance helps a user or architect of an MMS to narrow the focus to tune the architectural and workload parameters which have a large effect on the performance. Further, at a time, one or more subsystems can be systematically analyzed and optimized using the latency tolerance.

Intuitively, we say that a latency is tolerated, when a latency does not affect the performance of computation, i.e. the processor utilization is not affected. The latency tolerance is quantified using the tolerance index for a latency, and indicates how close the performance of the system is to that of an ideal system. An ideal system assumes the value of the latency to be zero.

To compute the tolerance index, we develop an analytical model based on closed queueing networks. Our solution technique uses the mean value analysis (MVA) [8]. Inputs to our model are workload parameters (e.g., number of threads, thread runlengths, remote access pattern, etc.) and architectural parameters (e.g., memory access time, network switch delay, etc.). The model predicts the tolerance index, processor utilization, network latency and message rate to the network. Analytical results are obtained for an MMS with a 2-dimensional mesh. The framework is general, and has been applied to analyze the EARTH system[7].

Our analysis of the latency tolerance of an MMS shows the following. First, in an MMS, the latencies incurred by individual accesses are much longer than their no-load values. However, the latency tolerance depends on the rate at which the subsystems can respond to remote messages, similar to vector computers. Second, to ensure a high processor performance, it is necessary that both the network and memory latencies are tolerated. Finally, with suitable locality, switches with non-zero delays act as pipeline stages to messages and relieve contentions at memories, thereby yielding better performance than even an ideal (very fast) network.

Next section describes our multithreaded program execution model and the analytical framework. Section 3 defines the tolerance index. Sections 5 to Section 7, report the analytical results. Finally, we present the conclusions.

*The first author is with IBM, Fishkill, and the second author is with University of Delaware.
2 The Analytical Model

This section outlines the analytical model. [7] reports the details.

The application program is a set of partially ordered threads. A thread is a sequence of instructions followed by a memory access or synchronization. A thread repeatedly goes through the following sequence of states—execution at the processor, suspension after issuing a memory access, and ready for execution after arrival of the response. Threads interact through accesses to memory locations.

The Multithreaded Multiprocessor System (MMS):

Our MMS consists of processing elements (PE) connected through a 2-dimensional torus. Each PE contains following three subsystems. A connection exists between each pair of these subsystems.

**Processor:** Each processor executes a set of \( n_i \) threads. The time to execute the computation in a thread is the runlength, \( R_i \), of a thread. The context switch time is \( C \).

**Memory:** The processor issues a shared memory access to a remote memory module with probability \( p_{\text{remote}} \). The memory latency, \( L \), is the time to access the local memory (without queueing delay) and observed memory latency, \( L_{\text{obs}} \), is the latency with queueing delay at the memory.

**IN Switch:** The IN is a 2-dimensional torus with \( k \) PEs along each dimension. A PE is interfaced to the IN through an inbound switch and an outbound switch. The inbound switch accepts messages from the IN and forwards them to the local processor or towards their destination PE. An outbound switch sends messages from a PE to the IN. A message from a PE enters the IN only through an outbound switch.

The Closed Queueing Network Model:

The closed queueing network (CQN) model of the MMS is shown in Figure 1. Nodes in the CQN model represent the components of a PE and edges represent their interactions. We model access contentsions. P, M and Sw represent the processor, memory and switch nodes, respectively.

All nodes in the performance model are single servers, with First Come First Served (FCFS) discipline. The service times are exponentially distributed. The service rates for P, M, and Sw nodes are \( \frac{1}{R} \), \( \frac{1}{L} \) and \( \frac{1}{S} \).

For requests from a thread at processor \( i \) to the memory at node \( j \), \( \text{em}_{i,j} \) is the visit ratio.\(^1\) The \( \text{em}_{i,j} \) depends on the distribution of remote memory accesses across the memory modules, geometric or uniform.

The geometric distribution is characterized by a locality parameter, \( p_{\text{sw}} \). \( \text{em}_{i,j} \) for a remote memory module at a distance of \( h \) hops is \( p_{\text{sw}} \frac{1}{h} \), where \( a \) is \( \sum_{k=1}^{\infty} p_{\text{sw}}^k \), and \( d_{\text{sw}} \) is the maximum distance between two PEs. A low \( p_{\text{sw}} \) shows a higher locality in memory accesses.

\(^1\)The visit ratio for a subsystem like the memory at a node \( j \) for a thread on processing node \( i \), is the number of times the thread requests an access to memory at node \( j \) between two consecutive executions on processor \( i \).

The average distance traveled by a remote access is:

\[
d_{\text{av}} = \sum_{m=1}^{d_{\text{sw}}} \frac{p_{\text{sw}}^m}{a} \times h.
\]

For uniform distribution over \( P \) nodes, \( \text{em}_{i,j} \) is \( \frac{1}{P - 1} \).

The switch is modeled as two separate nodes, inbound and outbound, each with a mean service time of \( S \) time units. The network switches are not pipelined.

A switch node interfaces its local PE with four neighboring switch nodes (in a mesh). The visit ratio \( e_{i,j} \) at inbound switch \( j \) is the sum of the remote accesses, which pass through the switch \( j \). The visit ratio \( e_{o_{i,j}} \) for the outbound switch is same as \( \text{em}_{i,j} \).

Figure 1: Queueing Network Model of a PE.

Solution Technique:

The state space for above CQN model is extremely large, and grows rapidly with the number of threads or number of processors. Since the above CQN model is a product-form network, we use an efficient technique Approximate Mean Value Analysis (AMVA) [8].

The core approximate MVA algorithm iterates over statistics for the population vectors \( N = (n_i, ..., n_i) \) and \( N - 1 \) representing the number of threads on each processor. With \( n_i \) threads on each processor, for each class \( i \) of threads and at each node \( m \), the AMVA computes: (i) the arrival rate \( \lambda_i \) of threads at the processor \( i \); (ii) the waiting time \( w_{i,m} \); and (iii) the queue length \( n_{i,m}^* \).

Using AMVA, we compute the following measures.

1. **Observed Network Latency:** The network latency \( S_{\text{obs}} \) for an access, is the sum of waiting time at a switch node (weighted by the visit ratio of a class \( i \) thread to that switch node) over all \( P \) switches in the IN:

\[
S_{\text{obs}} = \sum_{j=1}^{P} (w_{i,j} \times e_{i,j} + w_{i,j} \times e_{o_{i,j}})
\]

2. **Message Rate to the Network:** \( \lambda_{\text{net}} = \lambda_i \times p_{\text{remote}} \).
3. **Processor Utilization:** \( U_i = \lambda_i \times R_i \).

We use the above model to analyze our MMS. We verified the analytical performance predictions using Stochastic Timed Petri Net (STPN) simulations [7]. We have also applied the model to the EARTH system.
3 Tolerance Index

In this section, we discuss the latency tolerance and quantify it using the tolerance index.

When a processor requests a memory access, the access may be directed to its local memory or a remote memory. If the processor utilization is not affected by the latency at a subsystem, then the latency is tolerated. Thus, either the subsystem does not pose any latency to an access, or the processor progresses on additional work during this access. In general, however, the latency to access a subsystem delays the computation, and the processor utilization may drop. For comparison, we define a system ideal when its performance is unaffected by the response of an ideal subsystem under consideration, e.g., memory.

**Definition 3.1 Ideal Subsystem:** A subsystem which offers zero delay to service a request is called an ideal subsystem.

**Definition 3.2 Tolerance Index (for a latency):** Tolerance index, \( \text{tol}_{\text{subsystem}} \), is the ratio of \( U_{p, \text{subsystem}} \) in the presence of a subsystem with a non-zero delay to \( U_{p, \text{ideal subsystem}} \) in the presence of an ideal subsystem. In other words, \( \text{tol}_{\text{subsystem}} = \frac{U_{p, \text{subsystem}}}{U_{p, \text{ideal subsystem}}} \).

The choices for an ideal subsystem are: a zero delay subsystem or a contention-less subsystem. The former choice ensures that (for the network latency tolerance in an ideal system), the performance of a processor is not affected by changes in either the system size or a placement strategy for remote data. Further, we can also analyze the latency tolerance for more than one subsystem at a time.

A tolerance index of one implies that the latency is tolerated. Thus, the system performance does not degrade from that of an ideal system. We define that the latency is:

- tolerated if \( \text{tol}_{\text{subsystem}} \geq 0.8 \);
- partially tolerated, if \( 0.8 > \text{tol}_{\text{subsystem}} \geq 0.5 \);
- not tolerated, if \( 0.5 > \text{tol}_{\text{subsystem}} \).

The choice of 0.8 and 0.5 is somewhat arbitrary.

To compute \( \text{tol}_{\text{subsystem}} \), say for network, there are two analytical ways to obtain the performance of an ideal system. The latter can be measured on existing systems like EARTH [7]. First, let the switches on the IN have zero delays, then the performance can be computed without altering the remote access pattern. Second, let \( \text{tol}_{\text{remote}} \) be zero, then the ideal performance for an SPMD-like model of computation is computed without the effect of the network latency. The disadvantage is that the remote access pattern needs to be altered.

4 Outline of Results

We analyze the MMS described in Section 2 as a case study with default values of parameters in the Table 1. Architecture parameters are chosen to match the thread run-length \( R \). Our results show how high \( S_{\text{obs}} \) values rise with respect to its unloaded value 27.33 time units, under multi-threaded execution, and how to tolerate these long latencies.

In Section 5 we analyze the impact of workload parameters on the network latency tolerance. Section 6 reports an analysis of the memory latency tolerance. Section 7 analyzes how the tolerance index varies with scaling the number of processors from 4 to 100, i.e., \( k \) varies from 2 to 10.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_t )</td>
<td>( p_{\text{remote}} )</td>
</tr>
<tr>
<td>8</td>
<td>0.2, 0.4</td>
</tr>
</tbody>
</table>

Table 1: Default Settings for Parameters.

5 Network Latency Tolerance

In this section, we show the impact of workload parameters on the network latency tolerance.

Figure 2 show \( U_p, S_{\text{obs}}, \lambda_{\text{net}} \) and \( \text{tol}_{\text{network}} \) for \( R=10 \). Figure 3 shows \( \text{tol}_{\text{network}} \) for \( R=20 \).

While the absolute value of \( U_p \) is critical to achieve a high performance, the tolerance index signifies whether the latency of a subsystem is a performance bottleneck. Figures 2 and 3 show the tolerance index (\( \text{tol}_{\text{network}} \)) for the network latency at \( R=10 \) and 20, respectively. Horizontal planes at \( \text{tol}_{\text{network}} = 0.8 \) and 0.5 divide the processor performance in three regions: \( S_{\text{obs}} \) is tolerated; partially tolerated; and not tolerated.

In Figure 2, below critical \( p_{\text{remote}} \) of 0.3, on average, a processor receives a response before it runs out of work. Thus, even at a small \( n_t \) of 5, \( \text{tol}_{\text{network}} \) as high as 0.86 (Figure 2). Beyond \( p_{\text{remote}} \) of 0.3, \( \text{tol}_{\text{network}} \) drops to 0.75. A higher value of \( R \) increases the critical value of \( p_{\text{remote}} \) to 0.6 (see Figure 3).

Consider \( \text{tol}_{\text{network}} \) values for performance points with similar \( S_{\text{obs}} \) values (shown in Table 2). At \( R=10 \), \( n_t=8 \) tolerates an \( S_{\text{obs}} \) of 53 time units, while \( n_t=3 \) does not. For the same architectural parameters, different combinations of \( n_t, R \) and \( p_{\text{remote}} \) can yield the same \( S_{\text{obs}} \) but different \( \text{tol}_{\text{network}} \).

To improve \( \text{tol}_{\text{network}} \), first, with low \( p_{\text{remote}} \), more work is performed locally in the PE (e.g., \( p_{\text{remote}}=0.2 \), \( n_t=8 \), and \( R=20 \)), and hence \( \text{tol}_{\text{network}} \) value is higher. Second, an increase in \( n_t \) increases \( \text{tol}_{\text{network}} \), but increases the contentions and latencies of network and memories. Third, an increase in \( R \) reduces the number of messages to IN and local memory. Thus, \( S_{\text{obs}} \) and \( L_{\text{obs}} \) decrease and \( \text{tol}_{\text{network}} \) increases. The critical \( p_{\text{remote}} \) is also improved.

<table>
<thead>
<tr>
<th>( R )</th>
<th>( n_t )</th>
<th>( p_r )</th>
<th>( L_{\text{obs}} )</th>
<th>( S_{\text{obs}} )</th>
<th>( \lambda_{\text{net}} )</th>
<th>( U_p )</th>
<th>( \text{tol}_{\text{net}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8</td>
<td>0.2</td>
<td>40.7</td>
<td>52.7</td>
<td>0.0164</td>
<td>81.94</td>
<td>0.929</td>
</tr>
<tr>
<td>4</td>
<td>0.3</td>
<td>20.0</td>
<td>53.4</td>
<td>0.0170</td>
<td>56.80</td>
<td>0.710</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>14.8</td>
<td>54.7</td>
<td>0.0177</td>
<td>35.45</td>
<td>0.473</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>0.4</td>
<td>17.0</td>
<td>56.1</td>
<td>0.0175</td>
<td>87.55</td>
<td>0.899</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>14.9</td>
<td>55.2</td>
<td>0.0175</td>
<td>70.18</td>
<td>0.741</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.7</td>
<td>13.1</td>
<td>53.6</td>
<td>0.0174</td>
<td>49.69</td>
<td>0.543</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: \( \text{tol}_{\text{network}} \) at \( R=10 \) and \( R=20 \).
Figure 2: Effect of Workload Parameters at $R = 10$.

Figure 3: $tol_{network}$ at $R = 20$.

Impact of a Thread Partitioning Strategy

A thread partitioning strategy strives to minimize communication overheads and to maximize the exposed parallelism [5]. Let us assume that our thread partitioning strategy varies $n_t$ and adjusts $R$ such that $n_t \times R$ is constant.\footnote{Figure 4 shows $tol_{network}$ with respect to $n_t$ and $R$. We highlight certain values of $n_t \times R$ from Figure 4 in Table 3 and Figure 5. Table 3 shows that at a fixed value of $p_{remote}$ (say, 0.2), $tol_{network}$ is fairly constant, because $U_p$ and $U_{p, ideal network}$ increase in almost the same proportion with $R$. For $R \leq L (= 10)$, $L_{obs}$ is relatively high and degrades $U_p$ values. Since $U_{p, ideal network}$ is also affected, $tol_{network}$ is surprisingly high.

When $R \leq L$, Figure 5 shows a convergence of $n_t \times R$ lines, because the memory subsystem has more effect on $tol_{network}$. For $R \geq L$, the $tol_{network}$ (and $U_p$) value is close to maximum at $n_t = 2$. Further, a high value of $n_t \times R$ exposes more computation at a time, so $tol_{network}$ is high.}
6 Memory Latency Tolerance

In this section, we discuss the tolerance of memory latency using workload parameters. Figure 6 shows $tol_{memory}$ for $L = 20$, when $p_{remote} = 0.2$. Table 4 focuses on sample points for which $n_t \times R$ is constant. The data for $L = 10$ from Tables 3 and 4 indicates that a high $tol_{memory}$ means that a system is not a bottleneck, but $U_p$ is low, unless the latencies of all subsystems are tolerated. (When $R \geq L$, $U_p$ is proportional to $tol_{memory} \times tol_{network}$.)

At low $p_{remote}$, $L_{obs}$ increases almost linearly with $n_t$. For $R \leq L$, memory subsystem dominates the performance. An increase in $L$ from 10 to 20 increases $L_{obs}$ by 2.5 times.

A high $R$ improves $tol_{memory}$ and $U_p$, since the processor is busy for longer duration. A side effect is a lower contention at the memory. Further, in the thread partitioning strategy with $n_t \times R = constant$, the contentions are further reduced, due to decrease in $n_t$.

We also note that depending on the workload characteristics, the same value of $L_{obs}$ can result, when the MMS is operating in any of three tolerance regions.

<table>
<thead>
<tr>
<th>$p_r$</th>
<th>$n_t$</th>
<th>$R$</th>
<th>$L_{obs}$</th>
<th>$S_{obs}$</th>
<th>$\lambda_{net}$</th>
<th>$U_p$</th>
<th>$tol_{network}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>2</td>
<td>20</td>
<td>13.1</td>
<td>33.2</td>
<td>0.0069</td>
<td>69.74</td>
<td>0.825</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10</td>
<td>22.7</td>
<td>43.5</td>
<td>0.0133</td>
<td>66.29</td>
<td>0.829</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8</td>
<td>30.0</td>
<td>48.6</td>
<td>0.0153</td>
<td>61.02</td>
<td>0.843</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>47.1</td>
<td>55.7</td>
<td>0.0173</td>
<td>51.96</td>
<td>0.891</td>
</tr>
<tr>
<td>0.4</td>
<td>2</td>
<td>20</td>
<td>12.9</td>
<td>38.4</td>
<td>0.0111</td>
<td>55.45</td>
<td>0.656</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10</td>
<td>17.6</td>
<td>61.3</td>
<td>0.0190</td>
<td>47.64</td>
<td>0.596</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8</td>
<td>19.7</td>
<td>75.0</td>
<td>0.0212</td>
<td>42.49</td>
<td>0.587</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>22.8</td>
<td>103.7</td>
<td>0.0237</td>
<td>35.60</td>
<td>0.610</td>
</tr>
</tbody>
</table>

Table 3: Effect of Thread Partitioning Strategy.

<table>
<thead>
<tr>
<th>$L$</th>
<th>$n_t$</th>
<th>$R$</th>
<th>$L_{obs}$</th>
<th>$S_{obs}$</th>
<th>$U_p$</th>
<th>$tol_{memory}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>20</td>
<td>13.1</td>
<td>33.2</td>
<td>69.74</td>
<td>0.843</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10</td>
<td>22.7</td>
<td>43.5</td>
<td>66.29</td>
<td>0.797</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8</td>
<td>30.0</td>
<td>48.6</td>
<td>61.02</td>
<td>0.763</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>47.1</td>
<td>55.7</td>
<td>51.96</td>
<td>0.729</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>20</td>
<td>32.3</td>
<td>31.7</td>
<td>55.01</td>
<td>0.665</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10</td>
<td>67.2</td>
<td>35.4</td>
<td>41.62</td>
<td>0.501</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8</td>
<td>87.5</td>
<td>36.2</td>
<td>35.28</td>
<td>0.441</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>128.0</td>
<td>37.0</td>
<td>27.82</td>
<td>0.390</td>
</tr>
</tbody>
</table>

Table 4: $tol_{memory}$ at $p_{remote} = 0.2$.

7 Scaling the System Size

In this section, we discuss how the latency tolerance changes when the number of PEs varies.

Figure 7 shows $tol_{network}$ when the number of processors, $P$, is varied from 4 to 100 (i.e. $k=2$ to 10 processors per dimension). We consider two distributions for remote access patterns, geometric and uniform. At $p_{remote} = 0.2$, $n_t$ is varied for two runlengths. First, for a uniform distribution, $d_{avg}$ increases rapidly (from 1.3 to 5.0) with the system size, and $S_{obs}$ is not tolerated. But for a geometric distribution, $d_{avg}$ asymptotically approaches $\frac{1}{1-p_{sw}}$ (≈ 2) with increase in $P$. The performance for the two distributions coincides at $k = 2$ for all $n_t$ values. Second, even a large system does not require a large $n_t$ to tolerate $S_{obs}$.

Note that at $R = 10$, and $k$ from 6 to 10, $tol_{network}$ increases up to 1.05 for a geometric distribution, i.e. the system performs better than with an ideal IN. Figure 8 shows the system throughput, when $n_t = 8$ and $R = 10$. A geometrically distributed access pattern has an almost linear increase in throughput (slightly better than the system with an ideal IN). Transit delay for all remote accesses on an ideal IN is zero. Accesses from all processors contend at a

---

2This is similar to a grouping of accesses to improve $R$, in [5].
memory module increasing the $L_{obs}$ (see Figure 8). Thus, $U_p,ideal$ network is affected. For a geometric distribution, the IN delays the remote accesses at each switch (similar to the stages in a pipeline), just enough to reduce $S_{obs}$ and $L_{obs}$. The local memory accesses are serviced faster, and $U_p$ values improve. A fast IN may increase the contention at local memory, and the performance suffers, if memory response time is not low. Prioritizing the local memory requests can improve the performance of a system with a fast IN.

8 Conclusions

In this paper, we have introduced a new metric called the tolerance index, $tol_{subsystem}$, to analyze the latency tolerance in an MMS. For a subsystem, $tol_{subsystem}$ indicates how close the performance of a system is to that of an ideal system. We provide an analytical framework based on closed queueing networks, to compute and characterize $tol_{subsystem}$.

Our results show that the latency tolerance depends on the values of workload parameters and inherent delays at the subsystems, rather than the latency for individual accesses. Further, the latency is better tolerated by increasing the thread runlength (coalescing the threads) than by increasing the number of threads. Finally, with suitable locality, non-zero delays on network switches help to reduce contentions at memories, thereby yielding almost linear performance. Thus, an analysis of the latency tolerance helps a user focus the performance optimizations on to the parameters which affect the performance the most.

9 Acknowledgment

We acknowledge the support of MICRONET, Network Centers of Excellence, Canada, and IBM, Fishkill, USA. We also thank Profs. Govindarajan, Bhatt, and A. Chien.

Figure 7: $tol_{network}$ with system sizes at $R = 10$.

Figure 8: System throughput and latencies.

References