Approximate Performance Evaluation of Multi–Threaded Distributed Memory Architectures

W.M. Zuberek

Abstract

The long–latency memory accesses and unpredictable synchronization delays in multithreaded distributed memory architectures are tolerated by context switching, i.e., by suspending the current thread and switching the processor to another thread waiting for execution. Simple queueing models of multithreaded processors and their interconnecting network are used for finding approximate performance measures for the boundaries of the space of model parameters. These approximate measures are compared with performance results obtained by simulation of a detailed model of the analyzed architecture.

1. Introduction

Multithreading is a technique of tolerating long–latency memory accesses and synchronization delays in distributed memory systems. The basic idea of multithreading is quite straightforward [BH95, BR92]: instead of waiting for the completion of long–latency memory accesses (which in distributed memory systems can require hundreds of processor cycles), the processor suspends the current thread, switches to another thread waiting for execution (provided such a thread is available), and executes it concurrently with the initiated long–latency memory access. Since the threads execute in the same address space (threads are also known as ‘lightweight processes’), switching from one thread to another (which is called ‘context switching’) can be done very efficiently, even in one processor cycle if the processor has different sets of registers for different threads.

Several multithreaded architectures have been proposed [AC90, BH95, GN95, Sm81] which differ in two basic aspects, in the number of instructions executed before switching to another thread (the possibilities are one, several, or as many instructions as possible), and the cause of switching (count–based, on long–latency memory access, on long–latency remote memory access). In fine–grain multithreading, context switching is performed after each instruction; consecutive instruction are thus issued from consecutive threads in a cyclic manner, and if the number of threads is equal to the number of pipeline stages, pipeline stalls are eliminated because consecutive instruction from the same thread are separated by a sufficient number of instructions from other threads, with no dependencies among instructions. This is the approach used in HEP [Sm81]. For a long–latency memory access, the thread becomes inactive (usually for a one or two full thread cycles), with a relatively small degradation of the pipeline performance. For fine–grain multithreading, however, the number of available threads

\[1\] Department of Computer Science, Memorial University of Newfoundland, St. John’s, Canada A1B 3X5; email: \texttt{wlodek@cs.mun.ca}
is critical because the processor’s performance for a single thread is rather poor. In the alternative approach (called blocked multithreading), instructions are issued from the same thread until a long-latency memory access occurs, at which point the context is switched and another thread continues its execution. If the time of context switching is significantly smaller than the memory access time, it is beneficial (from performance point of view) to perform context switching for long-latency accesses to remote as well as local memory. If, however, the context switching time is not less than the memory cycle time, context switching is performed only for accesses to remote memory.

It is assumed in this paper that context switching is performed very efficiently (in one processor cycle); consequently, context switching is assumed to occur on every long-latency memory access (local as well as remote).

In distributed memory architectures the processors communicate by exchanging messages that are passed from one node to another by links of the interconnecting network. Interconnecting networks can have different topologies and different properties depending upon topology. It is assumed in this paper that all messages are routed along the shortest paths, but in a nondeterministic way. That is, whenever there are multiple shortest paths between two nodes, any of these paths is equally likely to be used. The delay of each message is proportional to the number of hops between the source node and the destination node, and it also depends upon the traffic in the chosen path. The interface between the network and the processor is through a pair network interfaces.

The performance of such distributed memory architectures depends upon a number of parameters related to the architecture (e.g., memory latency time, context switching time, switch delay in the interconnecting network) and a number of application parameters, such as the runlength of a thread (i.e., the average number of thread instructions issued before a context switch takes place), the average number of available threads, the probability of long-latency access to local memory, etc. A number of analytical studies of multithreaded architectures has been reported in the literature [Ag92, AB91, KD92, NG93, SB90, ZG97].

This paper derives simple models of the multithreaded architectures and uses them to perform approximate performance evaluation of the system. In particular, the utilization of the processors is studied in greater detail although the same approach can be used to study the performance of the memory or the switches of the interconnecting network. Section 2 describes a general model of multithreaded processors and the interconnecting network. Approximate performance measures are derived in Section 3, and Section 4 contains a few concluding remarks.

2. Model of the system

In distributed memory architectures, each node has local, non-shared memory, and is connected to a number of neighboring nodes in a systematic way. A two-dimension torus, shown in Fig.1 for a 16-processor system, is one of popular interconnecting schemes. For a larger number of processors, the number of dimensions of the interconnection scheme increases in order to reduce the average number of hops required for the accesses to remote (i.e., non-local) memory modules.

It can be shown [ZG97] that, assuming a uniform distribution of memory accesses over the nodes of the network, the average number of hops, $n_h$, is ap-
proximately equal to $p/2$ for a $p \times p$ two-dimensional torus network; for the 16-processor system shown in Fig.1, $n_h$ is equal to 2.

![Fig.1. An outline of a 16-processor system with a 2-D torus network.](image1)

In $k$-dimensional hypercube interconnecting network, each node is connected to $k$ other nodes, and the total number of nodes is $2^k$. Fig.2 outlines a 16-processor system connected by a 4-dimensional hypercube network. Since, in this case, each node is also connected to four neighbors, the average number of hops is the same as for a two-dimensional torus network.

![Fig.2. An outline of a 16-processor system with a hypercube network.](image2)

Each node in the system contains a (multithreaded) processor, local memory, and two switches which connect the node with the interconnecting network; one switch handles the traffic incoming from the network, and the other the traffic outgoing from the node. It is assumed that the outgoing traffic is distributed equally to the adjacent nodes. An outline of a queueing model of a multithreaded processor and its connection with the interconnecting network is shown in Fig.3.

In the multithreaded execution model, a program is a collection of partially ordered threads, and each thread consists of a sequence of instructions which are executed in the conventional von Neumann model.

The ready threads wait for execution in the Ready Queue. When the thread executing in the Processor issues a (long-latency) memory operation, the thread becomes suspended, context switching initiates execution of another thread from the Ready Queue, and the memory operation request is directed to either local or remote memory. For local memory operations, the request enters the Memory Queue, and, when the memory becomes available, is serviced, after which the suspended thread becomes ready and joins the Ready Queue.

For remote memory accesses, the request joins the Outbound Queue and is forwarded (by the Outbound Switch) to one of neighboring nodes where it enters the node through Inbound Queue and Inbound Switch. If the request needs to be forwarded to yet another node (another ‘hop’), it is sent from the Inbound Switch directly to another node. If the request has reached its target node, it
enters the Memory Queue, and after accessing the Memory, the result is sent back to the request’s source through Outbound Queue and Outbound Switch. When the result returns to the source node, it changes the status of the suspended thread to ready, and the thread joins the Ready Queue.

![Queueing model of a single multithreaded processor.](image)

Fig.3. Queueing model of a single multithreaded processor.

The most important parameters which affect the performance of the model shown in Fig.3 include thread runlength (or the average service time of Processor), the Memory cycle time (i.e., the service time of Memory), the delay of Inbound Switch and Outbound Switch, and the probability of local (or remote) memory accesses (which characterizes the ‘locality’ of memory references).

The main parameters used in the model are:

<table>
<thead>
<tr>
<th>parameter</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of threads</td>
<td>$n_t$</td>
</tr>
<tr>
<td>thread runlength</td>
<td>$\ell_t$</td>
</tr>
<tr>
<td>processor cycle time</td>
<td>$t_p$</td>
</tr>
<tr>
<td>memory cycle time</td>
<td>$t_m$</td>
</tr>
<tr>
<td>switch delay</td>
<td>$t_s$</td>
</tr>
<tr>
<td>average number of hops (in one direction)</td>
<td>$n_h$</td>
</tr>
<tr>
<td>probability to access local memory</td>
<td>$p_L$</td>
</tr>
<tr>
<td>probability to access remote memory</td>
<td>$p_R = 1 - p_L$</td>
</tr>
</tbody>
</table>

In most cases, only relative values of temporal parameters (e.g., the processors cycle time, the switch delay) are needed, so it is convenient to express the values of other parameters in terms of a selected one; it is assumed that the processor cycle time is one unit, and all other temporal parameters are relative to the processor cycle time. For example, the memory cycle time is 10 (processor cycle times), and the switch delays are 5 or 10 (again, processor cycle times).

3. Approximate performance evaluation

If the probability of accessing local memory, $p_L$, is close to 1.0, the model can be simplified by neglecting the interconnecting network and its influence; in fact, in such a case each node can be considered in isolation, as shown in Fig.4.
For this simple cyclic model, if the number of threads, $n_t$, is small, there is practically no queueing, so the throughput (the same for processor and the memory) can be expressed as:

$$\frac{n_t}{\ell_t + t_m}$$

and then the utilization of the processor is:

$$u_p' = n_t \frac{\ell_t}{\ell_t + t_m}$$

while the utilization of the memory is:

$$u_m' = n_t \frac{t_m}{\ell_t + t_m}$$

(the utilization of switches is zero in this case).

If the number of threads is large, the system enters its saturation, and the throughput is limited by the element with the maximum service demand (the so called bottleneck). Since the visit rates for the processor and memory are the same (both equal to 1), the bottleneck is determined by the (average) service time of elements; if $\ell_t > t_m$, the processor is the bottleneck, its utilization is close to 100%, and the utilization of memory is:

$$u_m'' = \frac{t_m}{\ell_t}.$$ 

On the other hand, if $\ell_t < t_m$, the memory is the bottleneck, its queue contains most of the requests waiting for service, memory utilization is close to 100%, while the utilization of the processor is:

$$u_p'' = \frac{\ell_t}{t_m}.$$ 

If $\ell_t = t_m$, both the processor and memory are utilized approximately 100%.

If the probability of accessing local memory, $p_\ell$, is close to 0.0, only remote memory accesses should be considered, and then the simplified model can be as shown in Fig.5.

A very straightforward expansion of the loops on the inbound switches, based on the average number of hops, $n_h$, results in the following approximations [ZG98]:

![Fig.4. Simplified model for local memory accesses.](image-url)
– if the number of threads, \( n_t \), is small, the queueing can be ignored, and the throughput is:

\[
\frac{n_t}{\ell_t + t_m + 2 \cdot (1 + n_h) \cdot t_s}
\]

then the utilization of the processor is:

\[
u'_p = n_t \cdot \frac{\ell_t}{\ell_t + t_m + 2 \cdot (1 + n_h) \cdot t_s}
\]

the utilization of the memory is:

\[
u'_m = n_t \cdot \frac{t_m}{\ell_t + t_m + 2 \cdot (1 + n_h) \cdot t_s}
\]

and the utilization of the (inbound) switch is:

\[
u'_s = n_t \cdot \frac{2 \cdot n_h \cdot t_s}{\ell_t + t_m + 2 \cdot (1 + n_h) \cdot t_s}
\]

– if the number of threads, \( n_t \), is sufficiently large, the system enters the saturation region, in which the performance of the whole system is limited by the bottleneck; the three possibilities for the bottleneck are:

- the inbound switch (if \( 2 \cdot n_h \cdot t_s \geq \max(\ell_t, t_m) \)): the utilizations of the processor and the memory are:

\[
u''_p = \frac{\ell_t}{2 \cdot n_h \cdot t_s}, \quad u''_m = \frac{t_m}{2 \cdot n_h \cdot t_s};
\]

- the processor (if \( \ell_t \geq \max(2 \cdot n_h \cdot t_s, t_m) \)): the utilizations of the memory and the (inbound) switch are:

\[
u''_m = \frac{t_m}{\ell_t}, \quad u''_s = \frac{2 \cdot n_h \cdot t_s}{\ell_t};
\]
- the memory (if $t_m \geq \max(\ell_t, 2*n_h*t_s)$): the utilizations of the processor and the (inbound) switch are:

$$u''_p = \frac{\ell_t}{t_m}, \quad u''_s = \frac{2*n_h*t_s}{t_m}.$$ 

For the values of $p_\ell$ between 0 and 1, the model must take into account both local and remote accesses, as shown in Fig.6.

![Fig.6. Simplified model for memory accesses.](image)

If the number of threads is small, queueing can be neglected and then the processor throughput is:

$$\frac{n_t}{\ell_t + p_\ell * t_m + (1 - p_\ell) * (t_m + 2 * (1 + n_h) * t_s)}$$

so the utilization of the processor is:

$$u'_p = n_t * \frac{\ell_t}{\ell_t + t_m + 2 * (1 - p_\ell) * (1 + n_h) * t_s}$$

The utilizations of the other elements (the memory and the inbound switch) can easily be derived from the throughput, similarly to the case of $p_\ell = 0$.

It should be observed that for $p_\ell = 0$ or $p_\ell = 1$ this general case simplifies to the corresponding previous case.

If the number of threads is sufficiently large, the system enters its saturation, and:

- if the inbound switch is the bottleneck, its utilization is close to 100%, and the utilization of the processor can be obtained from the visiting ratios relating the throughputs of the switch and the processor; the throughput of the processor is equal to the throughput of the switch (i.e., $1/t_s$) divided by the visit ratio of the switch:

$$\frac{1}{2 * (1 - p_\ell) * n_h * t_s}$$

and then the utilization of the processor is:

$$u''_p = \frac{\ell_t}{2 * (1 - p_\ell) * n_h * t_s}$$
if the processor is the bottleneck, the processor’s utilization is (almost) 100%, and then the utilizations of the memory and the (inbound) switch are:

\[ u''_m = \frac{t_m}{\ell_t}, \quad u''_s = \frac{2 \ast (1 - p_t) \ast n_h \ast t_s}{\ell_t} \]

if the memory is the bottleneck, the memory is utilized in (almost) 100%, and the utilizations of the processor and the (inbound) switch are:

\[ u''_p = \frac{\ell_t}{t_m}, \quad u''_s = \frac{2 \ast (1 - p_t) \ast n_h \ast t_s}{t_m} \]

For example, if the thread runlength \( \ell_t \) is equal to the memory cycle time \( t_m \) and to the switch delay, the balanced 16–processor system corresponds to \( p_t = 0.25 \) [ZG97], and the approximate processor utilization is as follows:

<table>
<thead>
<tr>
<th>case</th>
<th>processor utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_t = 1, n_t \rightarrow 1 )</td>
<td>0.5 ( n_t )</td>
</tr>
<tr>
<td>( p_t = 1, n_t \rightarrow \infty )</td>
<td>1.0</td>
</tr>
<tr>
<td>( p_t = 0, n_t \rightarrow 1 )</td>
<td>0.125 ( n_t )</td>
</tr>
<tr>
<td>( p_t = 0, n_t \rightarrow \infty )</td>
<td>0.25</td>
</tr>
<tr>
<td>( 0 \leq p_t \leq 1, n_t = 1 )</td>
<td>0.125 + 0.375 ( p_t )</td>
</tr>
<tr>
<td>( 0 \leq p_t \leq 0.75, n_t \rightarrow \infty )</td>
<td>( 1/(4 \ast (1 - p_t)) )</td>
</tr>
<tr>
<td>( 0.75 \leq p_t \leq 1, n_t \rightarrow \infty )</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The approximate utilization is sketched in Fig.7 as the boundary lines for the utilization surface which is shown as a function of two variables, the probability of accessing local memory, \( p_t \), and the number of available threads, \( n_t \).

Fig.7. Approximate processor utilization (\( \ell_t = t_m = t_s = 10 \)).

These approximate performance measures can be compared with the results obtained by simulation of the detailed model of distributed memory multiprocessor architecture [ZG97]. Fig.8 shows the utilization of the processor as a function of the same two variables, the probability of accessing local memory, \( p_t \), and the number of threads, \( n_t \), for \( \ell_t = t_m = t_s = 10 \).

If the delay of switches is reduced to 5 while \( \ell_t = t_m = 10 \), the balanced system corresponds to \( p_t = 0.5 \), and the approximate processor utilization is:
This approximate utilization is sketched in Fig.9, while Fig.10 shows the
case utilization of the processor obtained by a simulation study of a detailed model
of the system [ZG97].
'ridge' at \( p_\ell = 0.5 \) which is the balance boundary for this system; for \( p_\ell < 0.5 \) the inbound switch is the bottleneck while for \( p_\ell > 0.5 \) the processor and the memory become the bottlenecks (since \( l_t = t_m \)).

Since the inbound switch seems to be the most critical element in this system, its approximate utilization can be derived in a very similar way, using the simplified formulae for the same 16–processor system (with \( l_t = t_m = t_s \)):

<table>
<thead>
<tr>
<th>case</th>
<th>( p_\ell ) utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_\ell = 1 )</td>
<td>0.0</td>
</tr>
<tr>
<td>( p_\ell = 0, n_t \to 1 )</td>
<td>( 0.5 \times n_t )</td>
</tr>
<tr>
<td>( p_\ell = 0, n_t \to \infty )</td>
<td>1.0</td>
</tr>
<tr>
<td>( 0 \leq p_\ell \leq 1, n_t = 1 )</td>
<td>( 0.5 \times (1 - p_\ell) )</td>
</tr>
<tr>
<td>( 0 \leq p_\ell \leq 0.75, n_t \to \infty )</td>
<td>1.0</td>
</tr>
<tr>
<td>( 0.75 \leq p_\ell \leq 1, n_t \to \infty )</td>
<td>( 4 \times (1 - p_\ell) )</td>
</tr>
</tbody>
</table>

Fig.10. Processor utilization \((l_t = t_m = 10, t_s = 5)\).

Fig.11. Approximate switch utilization \((l_t = t_m = t_s = 10)\).
Fig. 11 outlines the approximation of the utilization surface as a function of the number of threads $n_t$ and the probability of accessing remote memory $p_r = 1 - p_c$; reverse orientation of $p_c$ is used in this case to improve the presentation.

![Figure 11: Deterministic Model: In Switch Utilization](image)

The approximation shown in Fig. 11 can be compared with simulation results shown in Fig. 12. Again, the agreement of simulation results with the derived approximation is quite good.

4. Concluding remarks

Simple formulae for approximate performance evaluation of multithreaded distributed memory architectures are derived by adjusting a general queueing model to specific boundary conditions. The approximate results are verified by comparison with results obtained by simulation of a detailed model; they appear to be quite close to the simulation results.

The derived measures can easily be generalized. For example, it was assumed that the distribution of memory accesses is uniform over the the nodes of the multiprocessor system. If this distribution is non-uniform and exhibits some locality effects, the average number of hops, $n_h$, should be adjusted accordingly (it should be reduced if the accesses to ‘close’ nodes are more likely than to the distant nodes), but otherwise the approach remains the same.

The proposed approach can easily be adapted to other models of multithreading. For example, context switching on remote loads (typically used in systems with ‘slow’ context switching), requires only a few modifications of the presented approach (in fact, the results for $p_r = 0$ remain the same; the other results need to be revised).

One of interesting conclusions from analysis of the results is that the multithreaded processors (with parameters similar to the presented ones) do not require large number of threads; typically about 5 threads is sufficient to reach the utilization that is not significantly different from the one obtained for a much larger number of threads.
It is hoped that a similar approach can be used for analysis of recent, more advanced ideas in multithreading techniques, which include simultaneous multithreading [Eg97,Se96], interleaving [La94], and other out-of-order instruction execution techniques [HS99] of advanced superscalar processors.

References


