An Analytical Solution for a Markov Chain Modeling Multithreaded Execution†

Rafael H. Saavedra-Barrera‡
David E. Culler‡

ABSTRACT

Multithreading is an architectural technique aimed at maintaining high processor utilization in the presence of large memory or interprocessor communication latency. While waiting for a remote reference to complete, the processor switches to another execution thread. Several realizations of this concept have been proposed, but little data is available on the actual costs and benefits. This paper presents an analytical model of multithreaded execution, which may serve to guide and explain empirical studies. The model is based on three key parameters: thread run-length, switch cost, and latency. A closed-form expression for processor utilization is obtained for deterministic and stochastic run-lengths. The derivation involves identifying specific patterns in the very large set of equations forming the Markov chain. Using this result, three operating regimes are identified for a multithreaded processor subject to long latencies: *linear*, where utilization is proportional to the number of threads per processor, *saturation*, where utilization is determined only by the run-length and switch cost, and *transition* between the other regimes. The model can be used to estimate the effects of several architectural variations.

1. Introduction

In a parallel machine based on conventional processors, the impact of communication delays between the CPUs and the memory modules increases with the number of processors and eventually limits processor utilization. The interval between sending a request to memory and receiving the result is called memory latency, and it is consider one of the fundamental problems in parallel computing [Arvi87]. The use of cache memories, that have proved to be effective in eliminating latency in uniprocessors, do not appear to provide a satisfactory solution in systems containing a large number of processors. In addition, caches in multiprocessors have to be kept coherent and it is not clear that present coherency protocols scale well.

† The material presented here is based on research supported in part by NASA under consortium agreement NCA2-128 and cooperative agreement NCC2-550, and by the National Science Foundation under grant PYI 90-58432. Computing resources were provided in part by the National Science Foundation through the UCB Mammoth project under grant CDA-8722788.

‡ Computer Science Division, EECS Department, University of California, Berkeley, California 94720.

An alternative solution to the latency problem is to tolerate the latency that caches and local memory do not eliminate. Multithreading is one of the promising architectural mechanisms for tolerating latency. It consists in switching very rapidly between ready-to-execute threads when the processor, on behalf of a thread, attempts to execute a long-latency operation. By switching execution to a different thread the processor avoids the effects of memory latency by executing instructions from other threads. However, this requires that a sufficient number of threads, either from a single task or from different tasks, be present in the processor. Assuming that it is possible to 'saturate' the processor with ready-to-execute threads, the pernicious effects of very large memory latencies are alleviated.

Multithreading has been used or proposed in some parallel architectures [Hals88, Smit78], such that the processor switches between threads on every cycle independent on the behavior of the thread. Although this approach simplifies the design it may increase the turnaround time of a task, as each thread executes one instruction every s cycles, where s is the depth of the execution pipeline. In more recent proposals the switching between threads is caused by a load instruction, a cache miss [Agar90], or a synchronization event [Iann88]. These proposals appear to offer the same level of utilization, but without overly increasing the turnaround time of a task.

Even though multithreading offers to improve processor utilization, there have been few studies that actually attempt to evaluate its potential benefits [Webe89, Agar89]. In our work we have tried to reverse this situation by proposing a model of multithreading based on a small number of significant parameters and studying their effect. We want to use the model to answer some of the most interesting questions confronting multithreading, such as: what is the magnitude of memory latency that justifies the use of multithreading? How many threads are needed to keep the processor busy most of the time? How sensitive is processor utilization to context switch overhead, i.e., the time it takes the processor to preempt a thread and start a new one? How do multiple contexts affect cache miss rates and network delays?

In a previous paper [Saav90], we presented a performance analysis of multithreading based on several parameters including: the number of contexts in the processor, the magnitude of the memory latency, the context switch overhead, and the average run length. Different variations of the model incorporated several embellishments to make the model more realistic. In the simplest model, memory latency and average run lengths were assumed constant. Further extensions to the model included incorporating a stochastic behavior of run lengths and assuming a dependency between the number of contexts and the cache miss rates. In a switch-on-miss multithreaded processor, variations in the number of cache misses have a corresponding effect in the run length of a thread. Our results showed that as the number of context increases processor utilization increases until it reaches some maximum and then starts to fall. This decrease in utilization is a direct result of a higher number of cache misses, as more contexts start competing for cache resources. In addition, we showed that there exists a close agreement between our results and trace-driven simulations when network congestion is moderate[1].

Other recent studies of multithreading [Agar89] have also derived expressions for processor utilization under the assumption that latency and run lengths are constant for a fixed number of

[1] Our model did include an explicit dependency between the number of contexts and network delays.

contexts. The assumption that latency only depends on the number of contexts is not unreasonable, considering that near to saturation a significant variation in latency have only a small impact on processor utilization. However, the thread run length is the most important factor affecting the effectiveness of multithreading and assuming it is constant is unrealistic. The novelty in our approach resides in considering that run lengths behave as a stochastic process having a geometric distribution. Experimental studies of run lengths between cache misses seem to support this assumption [SoZe88]. Incorporating this into the model results in a large Markov Chain, but one that has an analytical solution. In [Saav90] we presented the solution, but without providing a detailed derivation. The main purpose of this paper is to show how the solution was obtained.

In Section 2, we present some of the definitions used in the paper and derive expressions for processor utilization when run length is considered constant. Later we improve this by assuming that they behave as a stochastic process characterize by a geometric distribution. This condition gives rise to a large Markov chain. At the beginning of Section 3 we obtain the solution to a concrete example that illustrates some of the important steps in the derivation. We then present the set of equations in the general case and derive their solution. One of the steps of the derivation involves counting the number of states representing all the configurations of memory residual times under a given condition. The solution to this subproblem can be found by studying a related problem in combinatorics. This is presented in Appendix A. We end the paper by giving a small summary.

2. A Simple Multithreading Performance Model

A multithreaded processor consists of all the normal elements of a processor plus support for multiple active threads, also called contexts, fast context switching, and additional hardware to deal with pending memory requests. A context is embodied by its register set, program counter, and program status word. The main factors affecting the utilization of a multithreaded processor are the number of contexts in the processor (N); the magnitude of the memory latency (L_{mem} cycles); the overhead involved in switching contexts (C cycles); and the average run length between context switches (R cycles). Note that latency and run lengths are a function of the number of contexts; supporting more contexts means that each has less access to shared resources. In the following analysis we will assume that even when latency and run lengths are a function of the number of contexts, these are constant for a fixed value of N. Later we will drop this assumption in the case of run lengths.

Suppose that there is one context executing in the processor. Also assume that contexts are switched every time they attempt to execute an instruction that requires a remote reference to a memory module. As long as the context executes instructions that do not require a large latency it continues to have control of the processor. When the processor is forced to make a remote reference the context is preempted and another context is selected. A context is ready to execute when it has no outstanding memory requests pending in the network. If there are no ready-to-execute contexts, then the processor has to wait for the first memory request to complete before continuing doing useful work. This means that, at any cycle, the processor can be in one of the three following states: *Busy*, when a context is executing; *Switching*, while preempting a context and selecting a new one; and *Idle*, when all contexts are waiting for memory requests. Now,

if we focus on a context, instead of the processor, we see that it iterates through the following four states: *Running*, *Leaving*, *Blocked*, and *Ready*.

The above description assumes that there is at most one outstanding remote request per context. This is not an essential condition and can be replaced by assuming that contexts can make some number of remote requests before being blocked. A similar feature has being proposed for the Horizon machine [This88]. Obviously, the value of the average run length and memory latency will be affected by having more requests floating in the network.

The behavior of a multithreaded processor is conveniently represented by a Petri net diagram as in Figure 1. The circles represent *places*, and the boxes *transitions*. Four of the five places correspond directly to the four context states; place A is used to enforce the constraint that only one context can be running or leaving. Note that transition E is immediate, while transitions R, C and L are deterministic (black rectangles).

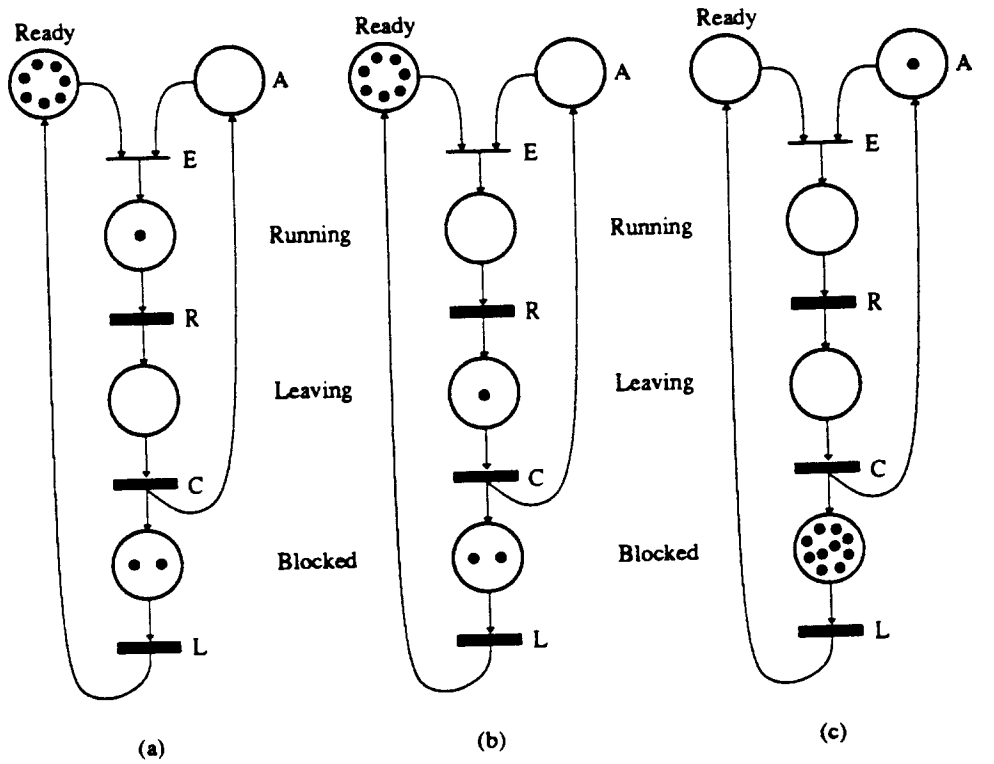

Figure 1: A Petri net representing the three states of the multithreaded processor. In (a) the processor is *Busy*; in (b) the processor is *Switching*; and in (c) the processor is *Idle*. The states for the contexts are represented by the places labeled *Ready*, *Running*, *Leaving*, and *Blocked*.

Multithreading attempts to use multiple contexts in order to increase the utilization of the processor. We will define processor utilization as the fraction of time the processor stays in the *Busy* state

$$\epsilon = \frac{Busy}{Busy + Switching + Idle}. \quad (1)$$

The basic idea behind a multithreaded machine is to interleave the execution of several contexts in order to dramatically reduce the value of *Idle*, without overly increasing the magnitude of *Switching*.

2.1. Utilization of a Single-Threaded Processor

A single-threaded conventional processor executes its only context until a remote reference is issued (R cycles) and then stays idle until the reference completes (L_{mem} cycles), before being allowed to resume execution. There is no context switch and, obviously, no switch overhead. We can model this behavior as an alternating renewal process having a cycle of $R + L_{mem}$. In terms of eq. (1), R and L_{mem} correspond to the amount of time during a cycle that the processor is *Busy* and *Idle*, respectively. Thus, the efficiency of the single-threaded machine is given by

$$\varepsilon_1 = \frac{R}{R + L_{mem}} = \frac{1}{1 + L_{mem}/R}. \tag{2}$$

In figure 2 we show how efficiency degrades as a function of L_{mem} and $1/R$. For the average run length of 50 cycles between memory requests and a latency of 10 cycles the efficiency is 83%. If memory latency increases to 75 cycles the efficiency drops to 40%. A 75-cycle latency is not an unreasonable value on a parallel machine.

2.2. Utilization of a Multithreaded Processor

With multiple contexts, memory latency can be hidden by switching to a new context, but we now must assume that switching takes C cycles of overhead. Assuming the run length between switches is constant, with a sufficient number of contexts there is always a context ready to execute when a switch occurs, so the processor is never idle. In this case, we say the processor is *saturated*. The cycle of the renewal process in this case is $R + C$, the time to execute a process and switch to the next, and the efficiency is simply

$$\varepsilon_{sat} = \frac{R}{R + C} = \frac{1}{1 + C/R}. \tag{3}$$

Observe, that the efficiency in saturation is independent of the latency and also does not change with a further increase in the number of contexts. Saturation is achieved when the time the processor spends servicing the other threads exceeds the time to process a request, i.e., when $(N-1)(R+C) \geq L_{mem} - C = L$. This gives the saturation point, under constant run length, as

$$N_d = \frac{L}{R + C} + 1. \tag{4}$$

Where L represents the residual time of the memory latency after context switching ($L_{mem} = L + C$). When the number of contexts is below the saturation point, there may be no ready contexts after a context switch, so the processor will experience idle cycles. The time to switch to a ready context, execute it until a remote reference is issued, and process the reference is equal to $R + C + L$. Assuming N is below the saturation point, during this time all the other contexts have a turn in the processor. Thus, the efficiency is given by

$$\varepsilon_{lin} = \frac{NR}{R + C + L} = \frac{N}{1 + L_{mem}/R}. \tag{5}$$

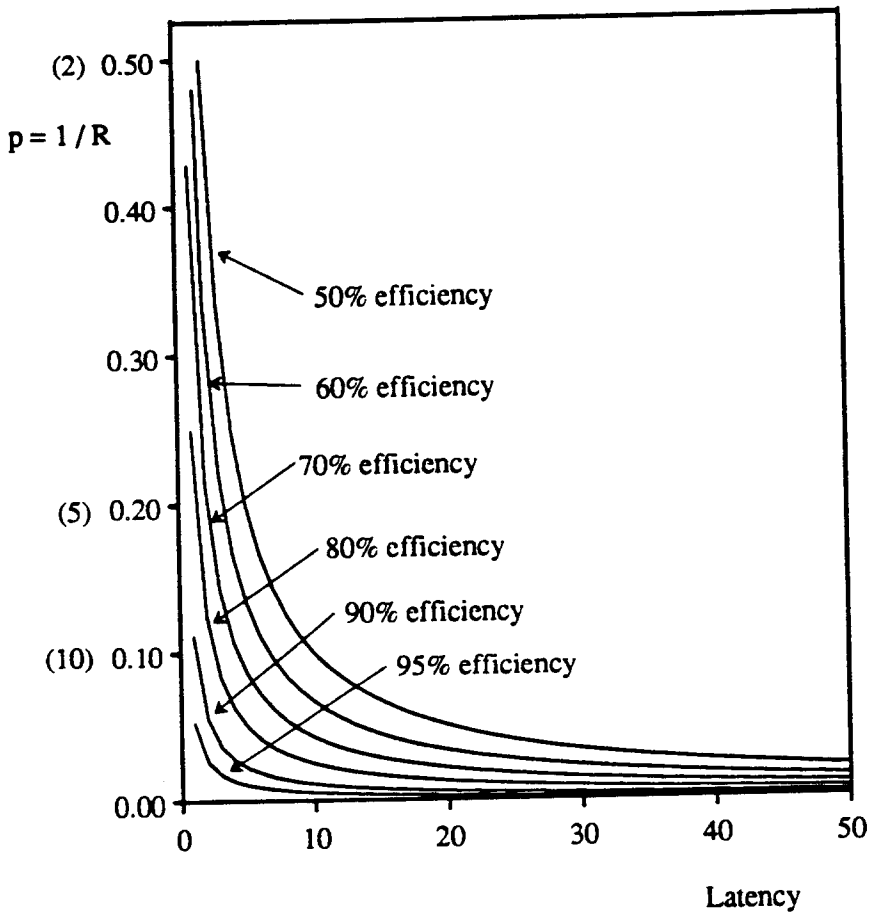

Figure 2: Curves of constant efficiency a function of context switch probability (p = 1/R) and memory latency. Efficiency is equal to the limiting probability of the processor being in a busy state.

Observe, that efficiency increases linearly with the number of contexts until the saturation point is reached and beyond that remains constant. The equation for ε_{sat} gives the fundamental limit on the efficiency of a multithreaded processor and underlines the importance of the ratio C/R. Unless the context switch is extremely cheap, the remote reference rate must be kept low.

3. Analysis with Stochastic Run Lengths

In this section, we improve the basic model by assuming the run length of a context (R) is a random variable having a geometric distribution; i.e., the probability of executing an instruction that will trigger a context switch is $p = 1/R$.

Before presenting the solution to the Petri Net, we will use a concrete example to show the relevant issues present in the derivation of the general solution. Consider a multithreaded processor supporting only 3 contexts ($N = 3$), and having a context switch time (C) of only 1 cycle. The

memory latency (L_{mem}) for a remote request is 6 cycles. Recall that the residual latency (L) is equal to the memory latency minus the context switch time, so this gives a value for L of 5 cycles. We can label this system as a $(3, 1, 5)$–multithreaded processor. The probability of context switching is modeled as a geometric distribution; all contexts in execution have, on every cycle, the same probability ($0 \leq p \leq 1$) of making a remote request.

The solution to the Petri Net can be found by obtaining its equivalent Markov chain, computing the limiting probabilities, and using these to compute the *Busy*, *Switching*, and *Idle* probabilities. The states of the Markov chain are exactly those in the reachability set of the Petri Net. The reachability set consists of all possible configurations of the Petri Net, and associated with each configuration there is a corresponding state in the Markov chain. We name each state in the Markov chain by appending the individual states of the contexts that are relevant in determining a unique configuration on the Petri Net. Contexts that are ready to execute are not relevant, so we ignore them. For example, if one context is running, one ready to execute, and another has been waiting for 2 cycles for its memory request to finish, we label this state as RB_3. Note, that here we need not indicate that there is one context in the ready state. With respect to the blocked context, the subscript gives the residual time for the memory request to finish. If in the next cycle the running context blocks, the Markov chain may make a transition to state C_1B_2. This state indicates that one contexts is in its first cycle of context switching and the blocked context has a residual time of two cycles. Note, that with respect to context switching, we give the time since the beginning of the transition and not the residual time, as in the case of a blocking context. This inconsistency between context switching time and residual blocking time does not affect the solution and it makes the analysis easier.

Not all combinations of residual blocking times and context switching times represent valid states in the Markov chain. Given the conditions for the behavior of a multithreaded processor, some configurations cannot happen. For example, RC_1 and RB_2B_1 are invalid. The first configuration cannot occur because the processor cannot be executing one context and context switching another in the same cycle. The second configuration is invalid because in the case of a $(3, 1, 5)$–multithreaded processor, requests to memory are separated in time by a distance of at least two cycles; the smallest interval between two consecutive memory requests is not less than two cycles, since every context executes for at least one cycle plus the time it takes to context switch. In the general case, the conditions that determine which configurations represent valid states are the following:

1) There is at most one context running or leaving.
2) The distance between any two residual blocking times is always greater than C, i.e., the time it takes to context switch.
3) If a context is being switched, its distance to any other blocked context must be greater than C.

Condition 3 is condition 2, but in the special case when one context is being context switched.

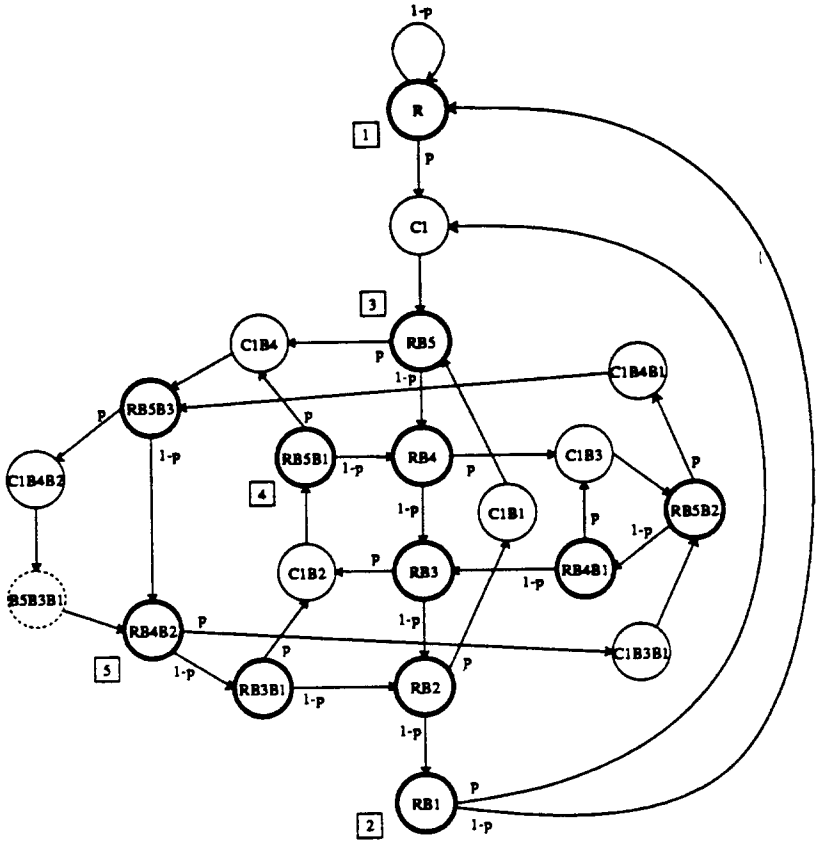

Figure 3: Markov chain for a (3, 1, 5)–multithreaded processor. The name of each state indicates a particular configuration of the Petri Net. Each stochastic transitions is labeled by its associated probability. The line width of a circle indicates its type: running states are shown in bold line, switching states in normal line, and idle states in broken line.

3.1. Solution to the Markov Chain for a (3, 1, 5)–Multithreaded Processor

Figure 3 depicts the Markov chain for the (3, 1, 5)–multithreaded processor with the name of all the states and the probabilities in the case of stochastic transitions. There are 21 states; in 12 of these states the processor is busy; in 8 context switching; and in one the processor is idle. In the figure, running states are indicated by a circle in bold line; switching states by a circle in normal line; and idle states by a circle in broken line. We can see how the Markov chain behaves by considering the set of transitions when the first four run lengths are 3, 2, 1, 1 cycles:

$$R \rightarrow R \rightarrow R \rightarrow C_1 \rightarrow RB_5 \rightarrow RB_4 \rightarrow C_1B_3 \rightarrow RB_5B_2 \rightarrow C_1B_4B_1 \rightarrow RB_5B_3 \rightarrow C_1B_4B_2 \rightarrow B_5B_3B_1.$$

In 7 of the 12 cycles the processor is busy, in four it is context switching, and in the last cycle it is idle.

From the Markov chain we can obtain the set of equations for the limiting probabilities. The equations for the busy states are:

$$R = (1-p)R + (1-p)RB_1$$
$$RB_4 = (1-p)RB_5 + (1-p)RB_5B_1$$
$$RB_3 = (1-p)RB_4 + (1-p)RB_4B_1$$
$$RB_2 = (1-p)RB_3 + (1-p)RB_3B_1$$
$$RB_1 = (1-p)RB_2$$
$$RB_4B_1 = (1-p)RB_5B_2$$
$$RB_3B_1 = (1-p)RB_5B_2$$
$$RB_5 = C_1 + C_1B_1$$
$$RB_5B_3 = C_1B_4 + C_1B_4B_1$$
$$RB_5B_2 = C_1B_3 + C_1B_3B_1$$
$$RB_5B_1 = C_1B_2$$
$$RB_4B_2 = (1-p)RB_5B_3 + B_5B_3B_1.$$

The first equation indicates that the state that corresponds to one context running and two contexts ready, can be entered from the same state with a probability $1-p$ (the running context does not block) or from a similar state where there is one ready context instead of two, plus one blocked context with one cycle left before receiving its memory request. This blocked context will make a transition into the ready state in the next cycle.

We can identify five patterns in the previous equations which represent all the different ways in which the system can make a transition into a running state. Specifically, a running state can be reached from, 1) two running states, 2) one running state, 3) two switching states, 4) one switching state, or 5) one running and one blocked state. Each pattern has a particular domain of validity. Furthermore, in the general case these same patterns describe all the possible equations for running states. The difference between patterns 1 and 2 (3 and 4) is a consequence of enforcing condition 2. In figure 4 some of the states in the chain have small squares with labels to illustrate the five patterns.

Similarly, the equations for the limiting probabilities for context switch and idle states are:

$$C_1 = p\,R + p\,RB_1$$
$$C_1B_4 = p\,RB_5 + p\,RB_5B_1$$
$$C_1B_3 = p\,RB_4 + p\,RB_4B_1$$
$$C_1B_2 = p\,RB_3 + p\,RB_3B_1$$
$$C_1B_1 = p\,RB_2.$$
$$C_1B_4B_2 = p\,RB_5B_3$$
$$C_1B_4B_1 = p\,RB_5B_2$$
$$C_1B_3B_1 = p\,RB_4B_2$$
$$B_5B_3B_1 = C_1B_4B_2$$

We see two distinct patterns for the context switch states and one pattern for idle states. In the general case there are two more patterns for the former and one for the later. The reason is that in this example we have a one-cycle context switch time and a single idle state. In the general case, when C is greater than one there are two more patterns modeling transitions between context switching states. And when the number of idle states is greater than one there is an additional equation.

The set of 21 equations contains only 20 independent equations, so we can obtain a parametric solution to this system in terms of one of the variables. The parametric solution in terms of state R is given by

$$RB_1 = C_1 = \frac{p}{1-p}R$$

$$RB_2 = RB_3 = RB_4 = RB_5 = \frac{p}{(1-p)^2}R$$

$$C_1B_1 = \frac{p^2}{(1-p)^2}R$$

$$RB_3B_1 = RB_4B_1 = RB_5B_1 = C_1B_2 = C_1B_3 = C_1B_4 = \frac{p^2}{(1-p)^3}R$$

$$RB_4B_2 = RB_5B_2 = RB_5B_3 = \frac{p^2}{(1-p)^4}R$$

$$B_5B_3B_1 = C_1B_3B_1 = C_1B_4B_1 = C_1B_4B_2 = \frac{p^3}{(1-p)^4}R$$

Using the additional condition that the sum of all limiting probabilities is equal to one, allows us to find the value of the limiting probability for state R

$$R = \frac{(1-p)^4}{2p^3+2p^2+2p+1}.$$

By adding all contributions for the states corresponding to a *Busy*, *Switching*, or *Idle* processor gives us the following limiting probabilities

$$Busy = \frac{p^2+p+1}{2p^3+2p^2+2p+1}; \quad Switching = \frac{p^3+p^2+p}{2p^3+2p^2+2p+1}; \quad Idle = \frac{p^3}{2p^3+2p^2+2p+1}.$$

These equations give the proportion of time, as a function of the context switching probability, that the processor will be in each state.

We can plot these three equations and see the efficiency of the processor as a function of the average run length of a context; Figure 4 shows this. When the run length is infinite the processor has a maximum efficiency of 1, and as the run length decreases the efficiency diminishes, with a minimum value of .4273, when every context executes for only one cycle. When the probability of context switch is 1, the run length is deterministic and the model is in the linear regime. In this case the efficiency is equal to 3/7 and is the same as that given by equation (5).

3.2. General Solution to the Markov Chain

We now present the solution for the Markov Chain in the general case. The derivation follows the same steps as the example we presented in the last subsection. The steps are: 1) we give the set of equations for all the running, switching, and idle states. 2) We obtain the parametric solution. 3) We then add all contributions of the limiting probabilities.

Extending the notation we used in the last section, we will represent a running state with k $(0 \le k \le N-1)$ contexts waiting for memory request to finish as $RB_{i_1} \cdots B_{i_k}$, where B_{i_j} indicates that the j blocked context has a residual waiting time of i_j cycles. Recall that the difference between any two residual times is greater than C. A particular configuration of the multithreaded processor when it is context switching is given by $C_l B_{i_1} \cdots B_{i_k}$, and the configuration when the processor is idle is given by $B_{i_1} \cdots B_{i_N}$.

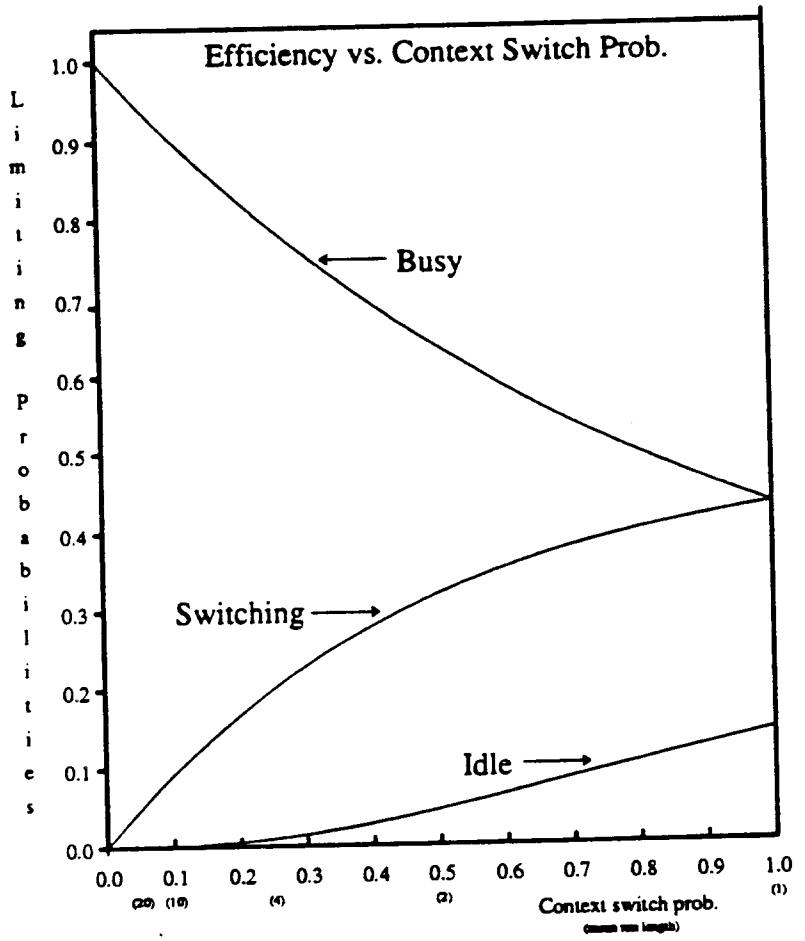

Figure 4: Efficiency versus context switch probability. Efficiency is equal to the limiting probability of the processor being in a busy state.

3.2.1. The Set of Equations

As we mentioned in section 3.1 there are only five different patterns present in the set of equations modeling the limiting probabilities when the processor is running. These are given by

$$RB_{i_1}\cdots B_{i_k} = \begin{cases} (1-p)RB_{i_1+1}\cdots B_{i_k+1} + (1-p)RB_{i_1+1}\cdots B_{i_k+1}B_1 & i_k > C;\ i_1 < L;\ 0 \le k \le N-2 \\ (1-p)RB_{i_1+1}\cdots B_{i_k+1} - 1 & i_k \le C;\ i_1 = L;\ 1 \le k \le N-1 \\ C_C B_{i_1+1}\cdots B_{i_k+1} + C_C B_{i_1+1}\cdots B_{i_k+1}B_1 & i_k > C;\ i_1 = L;\ 1 \le k \le N \\ C_C B_{i_1+1}\cdots B_{i_k+1} & i_k \le C;\ i_1 < L;\ 1 \le k \le N-1 \\ (1-p)RB_{i_1+1}\cdots B_{i_k+1} + B_{i_1+1}\cdots B_{i_k+1}B_1 & i_k > C;\ i_1 < L;\ k = N-1 \end{cases}$$

There are different equations depending on the type of states that can make a transition to a running configuration. For example, the first equation indicates that a running configuration where the largest residual time is less than L and the smallest residual time is larger than C can be

reached from two other running configurations each with a transition probability of $(1-p)$. The second state differs from the first by having an additional blocked context with a 1-cycle residual time. Note that this equation is the generalization of first four equations on the (3, 1, 5)–multithreaded processor example. The other equations represent different portions of the configuration space and enforce conditions 1-3 in section 3.

In the same way, we can obtain the set of equations modeling the limiting probabilities for switching configurations

$$C_l B_{i_1} \cdots B_{i_k} = \begin{cases} p\,RB_{i_1+1} \cdots B_{i_k+1} + p\,RB_{i_1+1} \cdots B_{i_k+1} B_1 & i_k > C;\ l = 1;\ 0 \le k \le N-2 \\ p\,RB_{i_1+1} \cdots B_{i_k+1} & i_k \le C;\ l = 1;\ 1 \le k \le N-1 \\ p\,RB_{i_1+1} \cdots B_{i_k+1} & i_k > C;\ l > 1;\ 0 \le k \le N-2 \\ C_{l-1} B_{i_1+1} \cdots B_{i_k+1} + C_{l-1} B_{i_1+1} \ldots B_{i_k+1} B_1 & i_k > C;\ l = 1;\ k = N-1 \\ C_{l-1} B_{i_1+1} \cdots B_{i_k+1} & i_k > C;\ l > 1;\ k = N-1 \\ C_{l-1} B_{i_1+1} \cdots B_{i_k+1} & i_k \le C;\ l > 1;\ 1 \le k \le N-1 \end{cases}$$

Note that there are only four different patterns present in the six equations. Finally, the equations for the idle configurations are:

$$B_{i_1} \cdots B_{i_w} = \begin{cases} C_C B_{i_1+1} \cdots B_{i_w+1} & i_1 = L \\ B_{i_1+1} \cdots B_{i_w+1} & i_1 < L. \end{cases}$$

3.2.2. Parametric Limiting Probabilities

We are now in a position to give the parametric solution to the limiting probabilities. We express all solutions in terms of the state represented by R. The unique solution will be obtained later by adding all contributions and normalizing them to one. The solution for the running configurations are:

$$RB_{i_1} \cdots B_{i_k} = \begin{cases} \dfrac{p^k}{(1-p)^{(C+1)k}} R & i_k > C;\ 0 \le k \le N-1 \\ \dfrac{p^k}{(1-p)^{(C+1)(k-1)+i_k}} R & i_k \le C;\ 1 \le k \le N-1 \end{cases} \tag{6}$$

In the case of the switching configuration, the solutions are:

$$C_l B_{i_1} \cdots B_{i_k} = \begin{cases} \dfrac{p^{k+1}}{(1-p)^{(C+1)k+l}} R & i_k > C;\ 0 \le k \le N-2 \\ \dfrac{p^N}{(1-p)^{(C+1)(N-1)}} R & i_k > C;\ k = N-1 \\ \dfrac{p^{k+1}}{(1-p)^{(C+1)(k-1)+l+i_k}} R & i_k \le C;\ 1 \le k \le N-2 \\ \dfrac{p^N}{(1-p)^{(C+1)(N-2)+l+i_k}} R & i_k \le C;\ k = N-1;\ l+i_k \le C \\ \dfrac{p^N}{(1-p)^{(C+1)(N-1)}} R & i_k \le C;\ k = N-1;\ l+i_k > C \end{cases} \tag{7}$$

The solutions in the case of idle configurations are:

$$B_{i_1} \cdots B_{i_w} = \frac{p^N}{(1-p)^{(C+1)(N-1)}} R. \tag{8}$$

We can easily prove that these solutions satisfy the the transition equations by making a case analysis. We will illustrate this by proving it for one of the equations The uppermost of the running transition equations is

$$RB_{i_1} \cdots B_{i_w} = (1-p) RB_{i_1+1} \cdots B_{i_w+1} + (1-p) RB_{i_1+1} \cdots B_{i_w+1} B_1$$

with conditions:

$$i_k > C; \; i_1 < L; \; 0 \le k \le N-2.$$

We will substitute the solutions from eq. (6) on the right hand side of the equal sign and arrive at an identity. Given that $i_k > C$ and $i_{k+1} = 1$ (B_1), we have to use the upper solution of eq. (6) for the first term and the lower solution of the same equation for the second term. So the limiting probabilities for the two configurations are

$$RB_{i_1+1} \cdots B_{i_w+1} = \frac{p^k}{(1-p)^{(C+1)k}} R; \qquad RB_{i_1+1} \cdots B_{i_w+1} B_1 = \frac{p^{k+1}}{(1-p)^{(C+1)k+1}} R.$$

By replacing the solutions and using elementary algebra we obtain the desired result.

$$RB_{i_1} \cdots B_{i_w} = \frac{(1-p)p^k}{(1-p)^{(C+1)k}} R + \frac{(1-p)p^{k+1}}{(1-p)^{(C+1)k+1}} R = \frac{(1-p)p^k + p^{k+1}}{(1-p)^{(C+1)k}} R,$$

$$RB_{i_1} \cdots B_{i_w} = \frac{p^k}{(1-p)^{(C+1)k}} R.$$

This is the correct result, and all the other solutions can be verified in the same way.

3.2.3. Adding All Contributions

The next step in the derivation consists in adding all contributions to each of the three processor states. We have to count the number of different states in the Markov chain that share the same solution. To do this, we need to count the number of configurations for a each solution satisfying its associated conditions. This is easily done by introducing a new notation representing the number of elements in each set and using the results given in Appendix A about the number of (L, C, k)-subsets.

Let $\#(XB_{i_1} \cdots B_{i_k}, L)_D$ represent the number of different configurations of residual blocked times associated with pattern $XB_{i_1} \cdots B_{i_k}$, where X is R, C_l, or empty, and subject to condition D. Using this notation and assuming that the probability of state R is one we can express the three unnormalized limiting probabilities as

$$\Pi_{Busy} = \sum_{k=0}^{N-1} \#(RB_{i_1} \cdots B_{i_w}, L)_{i_w > j} \frac{p^k}{(1-p)^{(C+1)k}} + \sum_{k=1}^{N-1} \sum_{j=1}^{C} \#(RB_{i_1} \cdots B_{i_w}, L)_{i_w = C} \frac{p^k}{(1-p)^{(C+1)(k-1)+j}}$$

$$\Pi_{Switching} = \sum_{k=0}^{N-2} \sum_{l=1}^{C} \#(C_l B_{i_1} \cdots B_{i_w}, L)_{i_w > C} \frac{p^{k+1}}{(1-p)^{(C+1)k+l}} + \sum_{k=1}^{N-2} \sum_{l=1}^{C} \sum_{j=1}^{C} \#(C_l B_{i_1} \cdots B_{i_w}, L)_{i_w = j} \frac{p^{k+1}}{(1-p)^{(C+1)k+l+j}} +$$

$$\sum_{l=1}^{C} \#(C_l B_{i_1} \cdots B_{i_{w-1}}, L)_{i_{w-1} > C-l} \frac{p^N}{(1-p)^{(C+1)(N-1)}} + \sum_{l=1}^{C} \sum_{j=1}^{C-l} \#(C_l B_{i_1} \cdots B_{i_{w-1}}, L)_{i_{w-1} = j} \frac{p^N}{(1-p)^{(C+1)(N-2)+l+j}}$$

$$\Pi_{Idle} = \#(B_{i_1} \cdots B_{i_w}, L) \frac{p^N}{(1-p)^{(C+1)(N-1)}}.$$

Assuming that R is one does not invalidate the derivation and will be corrected later by normalizing the solution. The equations are obtained from equations (6)-(8) by summing over the appropriate domains and introducing the corresponding condition D. For example, the two terms represented in the equation for the *Busy* state are obtained from equation (6) in the following way. The upper term is valid for all running states in the Markov chain having any number of blocked contexts in the interval $0 \leq k \leq N-1$ with a minimum residual time greater than C. This translates in a summation, where each term gives the number of states in the Markov chain that have k blocked contexts subject to condition $i_k \geq C$, since the smallest residual time must be larger than C. The lower term in (6) also translates into a summation over the number of blocked contexts plus another sum representing condition $i_k \leq C$; one term for each values of i_k. The same approach is used to obtain the terms for the *Switching* and *Idle* equations. What we have to do next is to compute an explicit expressions for the number of blocked configurations satisfying a given condition.

3.2.4. From Processor Configurations to (L, C, k)-Subsets

We are almost in the position to use the results in Appendix A to count valid configurations. These results will allows us to count how many valid states exist in the Markov chain that have k blocked contexts and subject to condition 2, i.e, that the distance between two residual times should be greater than C. However, these results do not take into account that the number of valid configurations is smaller when the processor is switching between contexts. If the processor has been context switching for l cycles the residual times of all contexts should be less than $L-l$. We will take this into account in counting states.

It is easy to apply the results from Appendix A when the processor is running a context. The number of valid states is just the number of different (L, C, k)-subsets. The following two derivations for running configurations are obtained directly from the two corollaries given in Appendix A.

$$\#(RB_{i_1} \cdots B_{i_k}, L)_{i_k > C} = \#(B_{i_1} \cdots B_{i_k}, L)_{i_k > C} = S_c(L-C, k) = \binom{L-kC}{k}$$

$$\#(RB_{i_1} \cdots B_{i_k}, L)_{i_k = j} = \#(B_{i_1} \cdots B_{i_k}, L)_{i_k = j} = S_c(L-C-j, k-1) = \binom{L-(k-1)C-j}{k-1}$$

The first derivation is explained as follows. When one context is being executed the set of residual times for the blocked contexts is not affected. Thus, the term after the first equal sign represents all the states that can be formed from k contexts having a maximum residual time of L cycles, and subject to the condition that the smallest residual time should be greater than C. By Corollary 2 of the Appendix, we know that these states correspond to all the $(L-C, C, k)$-subsets. The combinatorial term represents all the possible states for k blocked contexts.

The number of switching configurations are obtained in a manner similar to that for running configurations. The only difference is that we must consider that when a context is been switched, the maximum value for the residual time is reduced by the number of cycles that the

processor has been context switching (condition 3). Therefore, the four terms representing all the context switching states are:

$$\#(C_l B_{i_1} \cdots B_{i_k}, L)_{i_k > C} = \#(B_{i_1} \cdots B_{i_k}, L-l)_{i_k > C} = S_c(L-C-l,k) = \begin{bmatrix} L-kC-l \\ k \end{bmatrix}$$

$$\#(C_l B_{i_1} \cdots B_{i_k}, L)_{i_k = j} = \#(B_{i_1} \cdots B_{i_k}, L-l)_{i_k = j} = S_c(L-C-l-j,k-1) = \begin{bmatrix} L-(k-1)C-l-j \\ k-1 \end{bmatrix}$$

$$\#(C_l B_{i_1} \cdots B_{i_{N-1}}, L)_{i_{N-1} > C-l} = \#(B_{i_1} \cdots B_{i_{N-1}}, L-l)_{i_{N-1} > C-l} = S_c(L-C,N-1) = \begin{bmatrix} L-(N-1)C \\ N-1 \end{bmatrix}$$

$$\#(C_l B_{i_1} \cdots B_{i_{N-1}}, L)_{i_{N-1} = j} = \#(B_{i_1} \cdots B_{i_{N-1}}, L-l)_{i_{N-1} = j} = S_c(L-C-l-j,N-1) = \begin{bmatrix} L-(N-2)C-l-j \\ N-2 \end{bmatrix}$$

The number of idle configurations is:

$$\#(B_{i_1} \cdots B_{i_N}, L) = S_c(L,N) = \begin{bmatrix} L-(N-1)C \\ N \end{bmatrix}.$$

Now we can replaced the above terms in the parametric equations.

3.2.5. Normalizing the Limiting Probabilities

To conclude the derivation of the general solution requires that we normalize the limiting probabilities. Remember that at this point we have the parametric solution in terms of the limiting probability of state R, when $R = 1$. By enforcing the condition that the sum of all limiting probabilities must be equal to one we obtain the actual solution by eliminating R.

$$Busy = \frac{\Pi_{Busy}}{\Pi_{Total}}; \qquad Switching = \frac{\Pi_{Switching}}{\Pi_{Total}}; \qquad Idle = \frac{\Pi_{Idle}}{\Pi_{Total}},$$

where

$$\Pi_{Total} = \Pi_{Busy} + \Pi_{Switching} + \Pi_{Idle},$$

and

$$\Pi_{Busy} = \sum_{k=0}^{N-1} \begin{bmatrix} L-kC \\ k \end{bmatrix} \frac{p^k}{(1-p)^{(C+1)k}} + \sum_{k=1}^{N-1} \sum_{j=1}^{C} \begin{bmatrix} L-(k-1)C-j \\ k-1 \end{bmatrix} \frac{p^k}{(1-p)^{(C+1)(k-1)+j}}$$

$$\Pi_{Switching} = \sum_{k=0}^{N-2} \sum_{l=1}^{C} \begin{bmatrix} L-kC-l \\ k \end{bmatrix} \frac{p^{k+1}}{(1-p)^{(C+1)k+l}} + \sum_{k=1}^{N-2} \sum_{l=1}^{C} \sum_{j=1}^{C} \begin{bmatrix} L-(k-1)C-l-j \\ k-1 \end{bmatrix} \frac{p^{k+1}}{(1-p)^{(C+1)k+l+j}} +$$

$$C \begin{bmatrix} L-(N-1)C \\ N-1 \end{bmatrix} \frac{p^N}{(1-p)^{(C+1)(N-1)}} + \sum_{l=1}^{C} \sum_{j=1}^{C-l} \begin{bmatrix} L-(N-2)C-l-j \\ (N-2) \end{bmatrix} \frac{p^N}{(1-p)^{(C+1)(N-2)+l+j}}$$

$$\Pi_{Idle} = \begin{bmatrix} L-(N-1)C \\ N \end{bmatrix} \frac{p^N}{(1-p)^{(C+1)(N-1)}}.$$

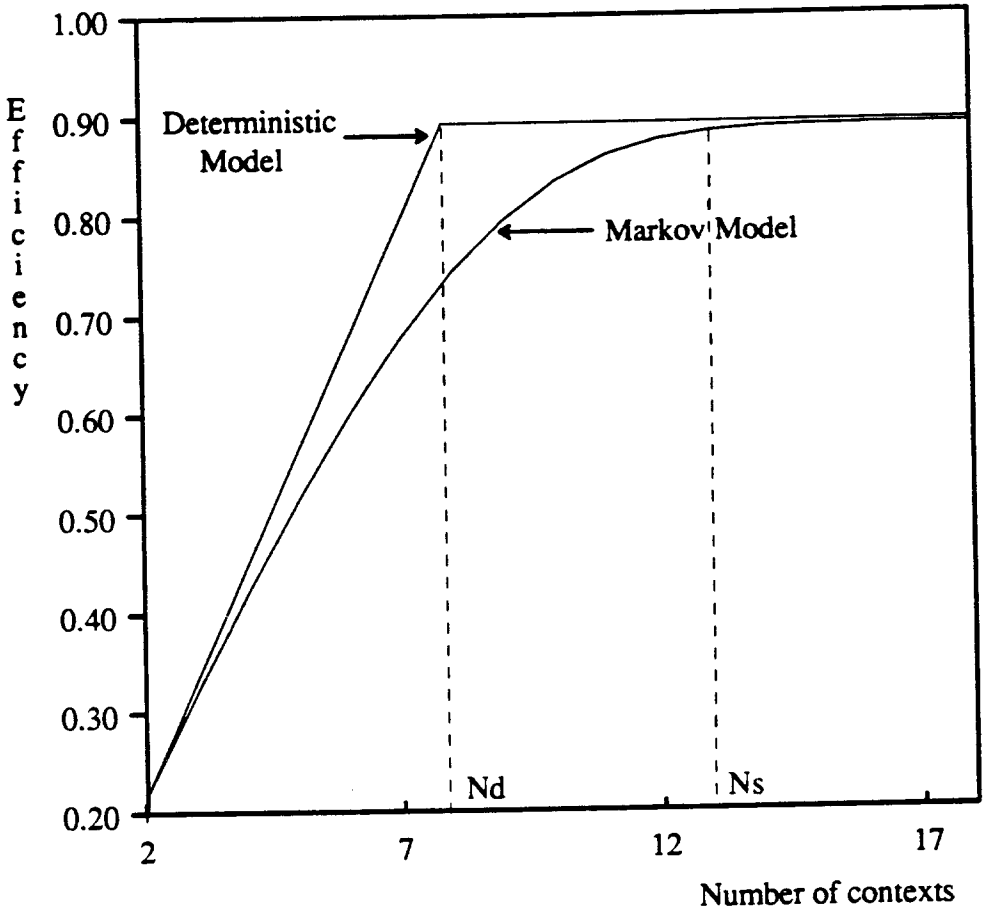

Figure 5: Efficiency as a function of the number of contexts: L = 128 cycles, R = 16 cycles, C = 2 cycles. The rightmost curve is obtained from the exact solution to the Markov chain.

3.3. Comparison Between the Deterministic and Stochastic Models

We will compare the solutions for the Markov chain and the deterministic model. Intuitively, we expect both solution to match well outside from the transition region with the maximum separation in the transition. In figure 5 we plot both solutions as a function of N. The latency is 128 cycles; the probability of context switching .0625 (R = 16); and the context switch time 2 cycles. In this case, the maximum efficiency of the processor is .888. The saturation point (N_d) predicted by the deterministic model is located at 8 contexts. This corresponds to a real efficiency of .743 which is only 84% of the actual maximum.

3.4. The Stochastic Saturation Point

In the deterministic model, efficiency is maximized when the number of contexts reaches the saturation point (N_d). In this situation the processor never enters the *Idle* state; it alternates only between the *Running* and *Switching* states. In the stochastic model, on the other hand,

efficiency approaches its maximum asymptotically and there is not a well defined saturation point, because there is always some probability, however small it may be, that the sum of all run lengths and contexts switching times will be smaller than the latency[2]. As we increase the number of contexts, however, we effectively reduce this probability. Hence, we can define a stochastic saturation point (N_s) as the number of contexts needed to reduce the probability of entering the idle state below some small constant.

Formally, we want to find the value of N_s that will satisfy the following condition

$$\Pr\left\{\sum_{i=1}^{N_s-1}(R_i+C)>L\right\}\geq\beta,$$

where R_i is the run length for context i. The equation is stated in terms of the probability of not entering the idle state. The solution to this equation is

$$N_s=\left\lceil\frac{\alpha R+((\alpha R)^2+4(R+C)L)^{1/2}}{2(R+C)}\right\rceil^2+1,$$

where $\alpha=\Phi^{-1}(\beta)$, and $\Phi^{-1}(x)$ is the inverse of the cumulative normal distribution. Details of the derivation are given in Appendix B.

For a probability β of .93 ($\alpha=1.5$) the stochastic saturation point, for the parameters used in figure 5, is located at 13 contexts. The efficiency at this point is 99% of the maximum. The stochastic saturation point represents a much better approximation of where a multithreading processor is saturated.

3.5. Total Number of States in the Markov Chain

In [Saav90] we presented an equation for the number of states in the Markov Chain without saying how it was derived. Here we give the derivation. The computation is easy to obtain by using the main results of (N,C,L)–subsets. We know that the total number of states, is the sum of all the *Busy*, *Switching*, and *Idle* states

$$\#(States)=\#(Busy)+\#(Switching)+\#(Idle).$$

Using the notation introduced in § 3.2.3 we have that the number of states is

$$\#(States)=\sum_{k=0}^{N-1}\#(RB_{i_1}\cdots B_{i_k},L)+\sum_{k=0}^{N-1}\sum_{l=1}^{C}\#(C_lB_i\cdots B_{i_k},L)+\#(B_{i_1}\cdots B_{i_N},L).$$

We now translate each term to its corresponding expression in terms of (N,C,L)–subsets

$$\#(States)=\sum_{k=0}^{N-1}S_C(L,k)+\sum_{k=0}^{N-1}\sum_{l=1}^{C}S_C(L-l,k)+S_C(L,N).$$

Adding the contribution of the *Idle* states into the *Busy* summation gives

$$\#(States)=\sum_{k=0}^{N}S_C(L,k)+\sum_{k=0}^{N-1}\sum_{l=1}^{C}S_C(L-l,k).$$

---

[2] This is not completely true; there is a fixed saturation point when the number of contexts is greater than $1+L/(C+1)$. This value, however, is too large to be useful.

We now use the Theorem 1 on Appendix A to get

$$\#(States) = \sum_{k=0}^{N} \left[\begin{matrix} L-(k-1)C \\ k \end{matrix} \right] + \sum_{k=0}^{N-1} \sum_{l=1}^{C} \left[\begin{matrix} L-l-(k-1)C \\ k \end{matrix} \right].$$

The second term on the right hand side of the equation can be expressed as

$$\sum_{k=0}^{N-1} \sum_{l=1}^{C} \left[\begin{matrix} L-l-(k-1)C \\ k \end{matrix} \right] = \sum_{k=0}^{N-1} \left[\sum_{l=1}^{L-(k-1)C} \left[\begin{matrix} L-l-(k-1)C \\ k \end{matrix} \right] - \sum_{l=C+1}^{L-(k-1)C} \left[\begin{matrix} L-l-(k-1)C \\ k \end{matrix} \right] \right].$$

By making the following change of variable $u = L-l-(k-1)C$, we can rewrite the equation as

$$\sum_{k=0}^{N-1} \sum_{l=1}^{C} \left[\begin{matrix} L-l-(k-1)C \\ k \end{matrix} \right] = \sum_{k=0}^{N-1} \left[\sum_{u=0}^{L-(k-1)C-1} \left[\begin{matrix} u \\ k \end{matrix} \right] - \sum_{u=0}^{L-kC-1} \left[\begin{matrix} u \\ k \end{matrix} \right] \right].$$

that can be reduced with the help of the following identity [Knut68]

$$\sum_{i=0}^{m} \left[\begin{matrix} i \\ n \end{matrix} \right] = \left[\begin{matrix} 0 \\ n \end{matrix} \right] + \left[\begin{matrix} 1 \\ n \end{matrix} \right] + \cdots + \left[\begin{matrix} m \\ n \end{matrix} \right] = \left[\begin{matrix} m+1 \\ n+1 \end{matrix} \right],$$

into

$$\sum_{k=0}^{N-1} \sum_{l=1}^{C} \left[\begin{matrix} L-l-(k-1)C \\ k \end{matrix} \right] = \sum_{k=0}^{N-1} \left[\begin{matrix} L-(k-1)C \\ k+1 \end{matrix} \right] - \sum_{k=0}^{N-1} \left[\begin{matrix} L-kC \\ k+1 \end{matrix} \right].$$

Replacing the above expression in the original equation for the number of states and making a change of variable

$$\#(States) = \sum_{k=0}^{N} \left[\begin{matrix} L-(k-1)C \\ k \end{matrix} \right] + \sum_{k=1}^{N} \left[\begin{matrix} L-(k-2)C \\ k \end{matrix} \right] - \sum_{k=1}^{N} \left[\begin{matrix} L-(k-1)C \\ k \end{matrix} \right].$$

All the terms inside the first summation but one, cancel with the terms of the third summation. The only term left is the constant one, and it can be included as a term in the middle summation[3]

$$\#(States) = 1 + \sum_{k=1}^{N} \left[\begin{matrix} L-(k-2)C \\ k \end{matrix} \right] = \sum_{k=0}^{N} \left[\begin{matrix} L-(k-2)C \\ k \end{matrix} \right].$$

This is the total number of states in the Markov chain which also represents the number of linear equations modeling the limiting probabilities.

4. Summary

Multithreading appears to offer advantages over conventional processors for parallel computing in the presence of large memory latencies. In this paper, we have presented in detail the solution to a Markov Chain model of a multithreading processor. The model assumes that run lengths behave as a random variable having a geometric distribution with mean $1/p$. Memory

[3] In [Saav90] we gave the following expression

$$\#(States) = 1 + L + C + \sum_{k=2}^{N} \left[\begin{matrix} L-(k-2)C \\ k \end{matrix} \right],$$

both equations are equivalent.

latency is assumed to be constant for a particular number of contexts. This assumption seems reasonable, because when the processor is close to saturation variations in memory latency do not affect its utilization.

The large number of states in the Markov Chain makes it unfeasible to solve it numerically except for very small cases. Instead, we have found analytical expressions for the *Busy*, *Switching*, and *Idle* terms. We showed how the general solution is obtained and compare it against a simple deterministic model. A multithreaded processor presents two working regimes mainly determined by the number of contexts: linear and saturation. In the linear regime there is a small number of contexts, so the effect of the memory latency cannot be completely hidden. In this situation the processor suffers from low utilization which increases linearly with the number of contexts. On the other extreme, if the number of contexts is large, then the processor never has to wait for a blocked context to complete its memory requests. In this situation utilization is maximum and is determined by the ratio C/R. The deterministic and stochastic models agree with each other when the processor is saturated or at the beginning of the linear regime, but not in the transition. The deterministic model tends to overestimate utilization by almost 25 percent.

References

[Agar89] Agarwal, A., "Performance Tradeoffs in Multithreaded Processors", Laboratory for Computer Science, Massachusetts Institute of Technology, November 1989.

[Agar90] Agarwal, A., Lim, B.H., Kranz, D., and Kubiatowicz, J., "APRIL: A Processor Architecture for Multiprocessing", *Proc. of the 17th Annual Int. Symp. on Comp. Arch.*, Seattle, Washington, June 1990, pp..

[Arvi87] Arvind, and Ianucci, R.A., "Two Fundamental Issues in Multiprocessing". *Proc. of DFVLR - Conf. 1987 on Parallel Proc. in Sc. and Eng.*, West Germany, June 1987, pp. 61-88.

[Hals88] Halstead, R.H., Jr., and Fujita, T., "MASA: A Multithreaded Processor Architecture for Parallel Symbolic Computing". *Proc. of the 15th Annual Int. Symp. on Comp. Arch.*, Honolulu, Hawaii, June 1988, pp. 443-451.

[Ianu88] Ianucci, R.A., "Toward a Dataflow / von Neumann Hybrid Architecture". *Proc. of the 15th Annual Int. Symp. on Comp. Arch.*, Honolulu, Hawaii, June 1988, pp. 131-140.

[Knut68] Knuth, D.E., *The Art of Computer Programming:* Vol. 1, *Fundamental Algorithms*, Addison-Wesley, Reading, Mass, 1968.

[Saav90] Saavedra-Barrera, R.H., Culler, D., and von Eicken, T. "Analysis of Multithreaded Architectures for Parallel Computing", *2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, Crete, Greece, July 1990, pp. 169-178.

[Smit78] Smith, B.J., "A Pipelined, Shared Resource MIMD Computer". *1978 Int. Conf. on Parallel Proc.*, 1978, pp. 6-8.

[SoZe88] So, K., Zecca, V., "Program Locality of Vectorized Applications Running on the IBM 3090 with Vector Facility", *IBM Systems Journal*, Vol. 27, No. 4, 1988, pp. 436-452.

[This88] Thistle, M.R., and Smith, B.J., "A Processor Architecture for Horizon". *Supercomputing '88*, Florida, October 1988, pp. 35-40.

[Webe89] Weber, W., and Gupta A., "Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture: Preliminary Results". *Proc. of the 16th Annual Int. Symp. on Comp. Arch.*, Jerusalem, Israel, June 1989, pp. 273-280.

Appendix A

In order to obtain an expression for the different number of valid running, switching, or idle configurations subjected to some condition D, we need to study a related problem that deals with counting the number of different subsets of size k that can be formed from a set containing the first L positive integers. The name of this problem is (L, C, k)-subsets. Each subset must satisfy the property that the difference between any two elements is greater than C. For example given the set $\{x \mid 1 \le x \le 9\}$, there exists 10 (9, 2, 3)–subsets. The list of valid subsets are:

{9,6,3}	{9,6,1}	{9,5,1}	{8,5,2}	{8,4,1}
{9,6,2}	{9,5,2}	{9,4,1}	{8,5,1}	{7,4,1}

Note that each subsets represents a valid configuration of residual times, where $L = 9$, $C = 2$, and $k = 3$. Let $S_C(L,k)$ be the number of (L,C,k)-subsets. There is a one-to-one correspondence between configurations of valid residual times for k blocked contexts and (L,C,k)-subsets.

$$\#(B_{i_1} \cdots B_{i_k}, L) = S_C(L,k).$$

Now, the (L,C,k)-subsets can be partitioned in two subgroups. If a subset contains the number L we put the subset in the first group; otherwise we put it in the second group. If we delete element L from all subsets in the first group, we are left with all the $(L-C-1, C, k-1)$-subsets. The second group corresponds to all the $(L-1, C, k)$-subsets. From this we can obtain the following recurrence relation for $S_c(L,k)$:

$$S_C(L,k) = \begin{cases} S_C(L-C-1,k-1) + S_C(L-1,k) & L > (k-1)(C+1)+1 \\ 1 & L = (k-1)(C+1)+1 \\ 0 & L < (k-1)(C+1)+1 \end{cases}$$

It is easy to see in the above expression that if $C = 0$ we obtain the binomial recurrence relation.

Theorem 1. The number of (L,C,k)-subsets is equal to

$$S_C(L,k) = \left[{L-(k-1)C \atop k} \right]$$

Proof. We will prove the result by a double induction on k, and L.

Base Case: When $k = 1$ it is clear that the number of $(L,C,1)$-subsets is equal to L; there is one subset for each integer

$$S_C(L,1) = \left[{L \atop 1} \right] = L.$$

Induction Step: We will assume that the theorem is true for all $k' < k$, and will prove that it is valid for $S_C(L,k)$. We will do this by induction on L.

Base Subcase: Let $L = (k-1)(C+1)+1$. There is only one subset, namely, $\{(k-1)(C+1)+1, (k-2)(C+1)+1, \cdots, C+1, 1\}$, and this is the result we get using the binomial expression

$$S_C((k-1)(C+1)+1, k) = \left[{(k-1)(C+1)+1-(k-1)C \atop k} \right] = \left[{k \atop k} \right] = 1.$$

Induction Substep: Assume that the theorem is true for $k' \leq k$ and $L' < L$; we will prove that the relation is also valid for $S_C(L,k)$. From the recurrence relation we have

$$S_C(L,k) = S_C(L-C-1,k-1) + S_C(L-1,k) .$$

The induction hypothesis applies to the two right-hand side terms, so we can replace both terms and obtain

$$S_C(L,k) = \begin{bmatrix} L-(k-1)C-1 \\ k-1 \end{bmatrix} + \begin{bmatrix} L-(k-1)C-1 \\ k \end{bmatrix} = \begin{bmatrix} L-(k-1)C \\ k \end{bmatrix}$$

This completes the proof.

The theorem can now be used to compute the number of valid (L,C,k)-configurations $(\#(B_{i_1} \cdots B_{i_k},L))$. However, we still have to extend the result in the case when the configurations are subjected to some condition D. For our purposes we only need to consider two conditions: when the smallest residual time has to be greater than a constant and when it has to be equal. The following two corollaries provide the desired results.

Corollary 1. The number of (L,C,k)-configurations such that the smallest residual time is larger than j is

$$\#(B_{i_1} \cdots B_{i_k},L)_{i_k > j} = S_C(L-j,k) \tag{9}$$

Proof. The result follows from the fact that there is a 1-to-1 transformation between the restricted (L,C,k)-subsets and unrestricted $(L-j,C,k)$-subsets. From the conditions, we know that all elements in each restricted (L,C,k)-subset are greater than j, thus we can subtract j from all the elements and obtain a $(L-j,C,k)$-subset. In the other direction all we have to do is to add j to each element in each of the $(L-j,C,k)$-subsets to get (L,C,k)-subsets satisfying the condition that every element is greater than j (by construction).

Corollary 2. The number of (L,C,k)-configurations such that the smallest residual time is equal to j is given by

$$\#(B_{i_1} \cdots B_{i_k},L)_{i_k = j} = S_C(L-C-j,k-1) \tag{10}$$

Proof. There exist a 1-to-1 correspondence between the restricted (L,C,k)-subsets and the $(L-C-j,C,k-1)$-subsets. Given that all restricted (L,C,k)-subsets are by definition different and have j as their smallest element, we can delete j from each subset and still maintain the property that all subsets are different, otherwise two of the original subsets would had been equal. Every element in these new subsets is greater than $C+j$, so by subtracting $C+j$ everywhere, we obtain subsets of $k-1$ elements with the property that every element is less or equal to $L-C-j$. The inverse transformation on the $(L-C-j,C,k-1)$-subsets completes the proof.

Appendix B

In order to derive equation for the stochastic saturation point we will introduce the following notation. Let R_{N_s-1} be the random variable representing the sum of all run lengths and context switching times from the moment one particular contexts blocks, until it returns to the busy state. During this time, the $N_s - 1$ remaining contexts pass through the running and leaving states. We call this interval the *non-idle time* of a memory request because it represents the maximum amount of latency that the processor can hide when a context blocks. This random variable is given by

$$R_{N_s-1} = \sum_{i=1}^{N_s-1} (R_i + C),$$

and has the following mean and standard deviation

$$\mu = (N_s - 1)(R + C), \tag{11}$$

$$\sigma = ((N_s - 1)R(R-1))^{1/2} \leq (N_s - 1)^{1/2}R. \tag{12}$$

Our goal is to derive an expression for N_s such that the probability that the non-idle time is greater than the latency is quite large. In other words, with high probability all the latency can be hidden by additional useful work and context switching time. This condition is captured by the following equation

$$\Pr\{R_{N_s-1} > L\} \geq \beta. \tag{13}$$

Now, without loss of generality we can express the mean of R_{N_s-1} as a function of the memory latency

$$\mu = L + D \tag{14}$$

for some constant D.

Given that each R_i is an independent geometric random variable and knowing from the central limit theorem[4] that the distribution of a sum of iid random variables converges to the normal distribution, we obtain the following expression for the distribution of R_{N_s-1}

$$\Pr\{R_{N_s-1} + \alpha\sigma > \mu\} = \Phi(\alpha).$$

Replacing μ by the right side of eq. (14) and moving D to the left side of the inequality gives us

$$\Pr\{R_{N_s-1} + \alpha\sigma - D > L\} = \Phi(\alpha).$$

The above equation is very similar to equation (13) and the latter equation can be obtained, if the following conditions are satisfied

$$D = \alpha\sigma, \qquad \text{and} \qquad \alpha = \Phi^{-1}(\beta).$$

By replacing the first condition into eq. (13) we get

$$\mu - \alpha\sigma - L = 0. \tag{15}$$

[4] The central limit theorem is not essential in our argument and it can be replaced by Chebychev's inequality.

The solution to (15) gives us the number of contexts, as a function of the latency, that will satisfy eq. (13). All we need to do in order to solve (15) is to substitute eqs. (11) and (12)[5] to obtain

$$(R+C)(N_s-1)-\alpha R\,(N_s-1)^{1/2}-L=0;$$

and making the following substitution

$$U=(N_s-1)^{1/2}$$

gives a simple second degree equation

$$(R+C)U^2-\alpha R\,U-L=0$$

whose solution proves our result after we apply the inverse transformation.

$$N_s=\left[\frac{\alpha R+((\alpha R)^2+4(R+C)L)^{1/2}}{2(R+C)}\right]^2+1.$$

[5] We will use the right side of the inequality in order to obtain a simpler final expression. This is conservative and does not affect our derivation.