Performance Evaluation with Petri Nets of a Bus-based Multithreaded Multiprocessor

Marcelo H. Cintra
Laboratório de Sistemas Integráveis
e-mail: mcintra@lsi.usp.br

Wilson V. Ruggiero
Laboratório de Arquitetura e Redes de Computador
e-mail: wilson@larc.usp.br

Universidade de São Paulo

Abstract

Multithreaded architectures have been actively investigated in recent years to build large-scale multiprocessors that are more tolerant to intra-context instruction dependencies and large synchronization and memory access latencies. In this paper we develop a multilevel Petri net model of a bus-based multiprocessor system and use this model to verify the impact of multithreading in processor utilization and network latency. Using a multilevel modeling methodology for Petri nets we show that multithreaded architectures have higher processor utilizations, but offer a higher load to the interconnection network and the memory system.

1 Introduction

With the advances in current microprocessor technology, we observe an increase in the difference between the processor's clock cycle time and memory and network latencies; and an increase in the level of on-chip hardware parallelism (multiple functional units and multiple issue). To achieve the highest performance possible with the available resources, it is important to maintain a high processor utilization. However, memory access latencies and pipeline dependencies reduce the actual processor utilization in current systems.

Several techniques have been proposed in the literature to cope with long latency operations and instruction dependencies, such as relaxed memory consistency models, prefetching, fine-grain synchronization, and compile-time optimizations to increase the scheduling scope.

Multithreaded architectures can potentially cope with both long-latency operations and instruction dependencies. The basic idea in multithreading is to allow the processor to run other threads while some threads are blocked due to pipeline dependencies or long-latency operations, thus increasing the processor utilization at the system level and

reducing execution time of parallel applications. Software approaches to multithreading require little hardware support and rely on the compiler and the operating system to maintain and schedule threads [6], while hardware-based approaches rely almost entirely on multiple-context processors [1, 4, 7, 9, 12, 13].

The usual approach currently used to evaluate the performance of multiprocessor architectures is to build customized program-driven simulators, which can accurately model the behavior of both the hardware architecture and the application, but require much time to build and to debug the custom simulation code. Another approach to evaluate the performance of computer systems is to use discrete event system models such as Petri nets [11] and queueing systems [8]. As these models have a limited number of object types, simple and strict operating semantics and useful graphical representations, the time required to create and debug the computer architecture model is much shorter.

As in a previous work [5], in this work we have chosen to use Petri nets to model and evaluate the performance of a multiprocessor computer architecture. This technique has proven itself adequate to model both parallelism and conflict [3], two important characteristics present in modern computer systems. Extensions such as timed and stochastic Petri nets make these nets a powerful technique for analyzing the performance of systems.

Modeling large and complex computer systems leads to complex Petri net models that are difficult to understand, debug and solve. A possible approach to deal with complex models is to partition the model in submodels that are hierarquicaly related to each other. The advantage of this approach is that only part of the data from the solution of the inferior model may be needed to solve the hierarquically superior models, leading to more efficient solutions of complex models.

In this paper we use a multilevel Petri net model to evaluate the performance of multithreaded processors in a bus-based multiprocessor system. In [2], analytical models for a blocked multithreaded processor, a point-to-point interconnection network and a cache system are derived. These models, however, represent a limited multithreading scheme and cannot be easily adapted to different configurations of the network. Petri nets were used to model a blocked-multithreaded processor [10], but the model presented implements a simple memory model, and does not take into account the conflicts for the bus. Our model is a detailed model for an interleaved-multithreaded processor and the memory model accurately represents the contention for the bus and the memory modules.

The organization of this paper is as follows: in section 2 we discuss multithreading techniques and in section 3 with present a multilevel modeling methodology. In section 4 we present a Petri net model for a bus-based multiprocessor and in section 5 we evaluate the performance of a multithreaded architecture. Finally, section 6 concludes the paper and presents some future works. For a review of Petri net theory refer to Peterson's book [11].

2 Multithreaded Architectures

Multithreading techniques allow the processor to switch among resident contexts and issue instructions from different threads to maximize processor utilization when some threads are unable to fill the issue slots. So, the basic issue in multithreading models

is how to schedule threads and their instructions in the available issue slots. The multithreading models described next differ basically on when a context switch occurs, and a new thread is allowed to issue more instructions. For superscalar processors, the models also differ on how the available issue slots are filled with instructions from non-blocked threads.

- Issue instructions from a single thread and switch on long-latency operations

  In this model a thread is allowed to issue instructions until it reaches a long-latency operation, such as a cache miss, at which point the thread is considered blocked and a context switch occurs. In this model, which has been called "blocked" or "coarse grain multithreading" [1, 14], the thread is allowed to fully utilize the available issue slots and functional units. Thus, this model offers a single-thread performance similar to a single context processor, but is also subject to all pipeline dependencies found in single context processors.

- Issue instructions from a single thread and switch on every cycle

  In this model a context switch occurs every clock cycle, that is, at each clock cycle the instruction is issued from one of the available threads, in a round-robin or priority scheme. Some variations of this model have been studied in the literature, such as the "fine-grain" multithreading scheme, exemplified by the HEP multiprocessor [12] and the MASA architecture [7], and the "interleaving" scheme [9], which is the model that we use in our multithreaded multiprocessor system, described in section 5. This model can also be used in multiple-issue processors, as in the Tera multiprocessor [4], which issues 3 instructions from the same thread, in a VLIW scheme, and switches to a different context every cycle.

- Issue instructions from multiple threads and switch on every cycle

  As in the previous model a context switch occurs every clock cycle, and instructions can be issued from different contexts on each new cycle. However, in this model the issue slots, in a superscalar or VLIW processor, can be used by different threads in the same cycle. This scheme was proposed in [13] and was called "Simultaneous Multithreading". Several variations of this model can be implemented, with varying degrees of complexity and flexibility. The most general variant, called "Full Simultaneous Multithreading", allows all active threads to compete each cycle for all available issue slots, but has a very complex implementation.

3 A Multilevel Modeling Methodology

Detailed Petri net models of complex computer systems are difficult to debug and maintain and require large ammounts of computational resources to be solved. Partitioning the model in submodels leads to small Petri net modules that are easier to debug and maintain. In addition, creating submodels that are hierarchically related, i.e. one submodel is a detailed description of part of the higher level model, can make the solution process less expensive.

The basic procedure in creating multilevel models is to represent a certain portion of the system as a high level event (a transition in Petri nets) in the upper level and

as a detailed submodel in the lower level. Solving the lower level submodel we can then obtain enough information on the timing behavior of the subsystem and, using a special methodology, we can accurately transfer this information to the upper level model.

In our proposed methodology, we represent the subsystem in the high level model as a single transition, whose firing delay depends on the number of tokens in its input place, which we call the *load* place. The number of tokens in this transition, which we will call the *hierarchical* transition, represents the load offered to the lower level subsystem. To solve the higher level model, we then need a table mapping the number of tokens in the input place of the hierarchical transition to the actual firing time of the transition.

Following this methodology, we need only to obtain the mapping from the load on the lower level submodel to the effective time involved in the event represented by the subsystem. This can be accomplished by solving the lower level submodel for different loads and obtaining the average time required by the subsystem to complete its service. In practice, the subsytem model is an open Petri net model, and the time involved in the operation of the subsystem is the time required by a token to flow through the model and exit. Thus, to solve the model and obtain the average time required by the subsystem, we need to close the model, which can be accomplished by feeding back all the exits from the submodel to a single entrance place. This entrance place is then the place in the interface of both models, and we call it the *interface* place. Figure 1 pictures the relationship between the submodels in our methodology.

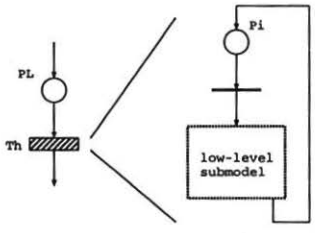

Figure 1: Pictorial representation of the Petri net submodels in the multilevel modeling methodology. Place p_L is a load place and place p_i is an interface place. Transition t_h is a hierarchical transition.

Once we have the closed model of the subsystem, we must determine the range of load offered by the higher level model, i.e. the range of the number of tokens that can appear in the interface and load places. With this range, we then simulate the submodel and obtain a table of number of tokens in the interface place versus average time required by a token to circulate in the submodel and return to the interface place. Because such simulation results are subject to stochastic behavior, and the difference in the cycle time can be very small for consecutive numbers of tokens in the interface place, we may observe some oscilation in the function obtained from this table. To remove this oscilation and obtain a smooth function relating load versus cycle time, one can increase the number of cycles or run more simulations and average the results across these simulations. If some oscilation persists at some points, one can also use a simple linear interpolation to correct the values of the points.

The table created by the methodology described above can then be used to dynamically calculate, at each simulation cycle, the firing delay of the hierarchical transition in the higher level model. This methodology has two main advantages over solving a complete model of the system. First, once the mapping table is created, it is possible to simulate the higher level model without having to simulate the lower level submodel. The second advantage is that because the only information required from the lower level submodel is the mapping table, one can easily substitute the lower level model by a different model to either increase the detail of the submodel or represent a different configuration.

The proposed multilevel modeling methodology for Petri nets can be summarized as follows:

- Partition the complete system in hierarchically related submodels, following the scheme shown in figure 1.
- Close the lower level submodel by feeding back the exits from the submodel to a common input place.
- Determine the range of the load offered by the higher level portion of the system to the lower level subsystem.
- Obtain the cycle time for all the range of load in the subsystem
- Eliminate the eventual oscilations in the function relating the load to the cycle time.
- Solve the higher level submodel dynamically calculating the firing time of the hierarchical transition using the table obtained in the previous phase.

4 A Bus-based Multiprocessor Model

To evaluate the performance of a multithreaded architecture we must accurately evaluate the long-latencies observed in the system. In this work we consider a popular configuration of a bus-based multiprocessor with N processors and M memory modules, connected by a common bus. In this system, the contexts requesting a memory access are put in a queue to access the bus, which can only service one request at a time. Once the bus is granted to a context, a data request packet is sent to one of the memory modules and the request for data is enqueued at this module. All memory modules service only one request at a time and to send the responses back to the processors they must compete to access the bus. Figure 2 shows a Petri net model for such a system.

In the Petri net model of figure 2, place $p3$ represents the shared bus and place p5 represents the memory pool with M modules. For simplicity, in this model we consider that the memory requests are uniformily distributed across the memory modules. Place $p1$ represents the contexts requesting memory access and the firing of transition $t6$ removes a context from the memory system after its request is completed. Transitions $t1$ and $t5$ represent the contention for using the bus for the data request and reply, respectively. Transitions $t2$, $t4$ and $t6$ have fixed firing delays and represent the packet transmission time ($t2$ and $t6$) and the actual memory access time ($t4$). The values used

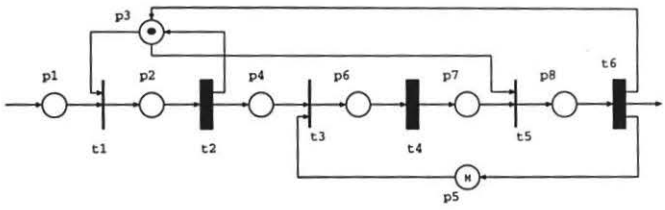

Figure 2: Petri net model of a bus-based multiprocessor with M memory modules.

for these times (in processor cycles) are typical of a modern multiprocessor system: $t2 = t6 = 5$ and $t4 = 20$.

According to the multilevel modeling methodology presented in section 3, to transfer the behavior of this memory system to the higher level system, we must compute the average overall access times for different number of contexts. We then closed the model presented in figure 2, by connecting transition $t6$ to place $p1$, and used the simulator RP_SIM [5] to compute the average access time for 1 to 32 tokens in place $p1$, representing 1 to 32 contexts requesting memory access. This data is then used in the multithreaded processor, presented in the next section.

We observed that modeling in detail the bus and memory system is highly desireable because the effective memory access time is very sensitive to the number of contexts contending for the bus. In our simulations, we found that the effective access time varies from 30 cycles, with only one context, to about 320 cycles with 32 contexts. Thus, a simple model of the multithreaded multiprocessor that approximates the memory access time by a single value for all loads will lead to misleading results.

5 A Multilevel Model of a Multithreaded Architecture

In this work we developed a Petri net model of an interleaved multithreaded architecture [9], which we believe to be the most flexible and better performing multithreading model for traditional scalar processors. In this model, an instruction from a different context is issued and executed at each cycle and if the instruction is a memory request the context is marked blocked, otherwise the context remains ready in the task pool. The Petri net model for this processor is shown in figure 3

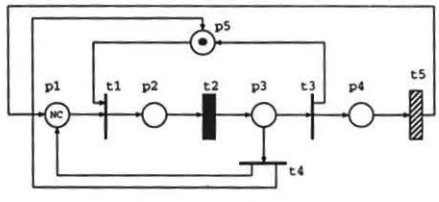

Figure 3: Petri net model of a multithreaded scalar processor with NC resident contexts and implementing the interleaving multithreading model.

In the model of figure 3, place $p1$ represents the task pool with NC ready contexts, $p5$ represents the single issue slot available each cycle and place $p4$ represents contexts requesting memory access. The system considered has an off-chip cache and the firing of transition $t4$ indicates that either the instruction was not a memory operation or the instruction was a memory operation that could be satisfied from the cache. Transition $t3$, on the other hand, represents a memory operation that could not be satisfied from the cache. Transitions $t3$ and $t4$ are a random switch and the probability associated with the firing of transition $t3$ is the cache-miss rate observed for a given application. To model a multiprocessor system with P processors, we need only replicate P times the Petri net model in figure 3 and use the sum of tokens in all places $p4$ as the total number of contexts contending for the memory. In this case the union of places $p4$ corresponds to the load place.

The times involved in the model of figure 3 are 1 cycle, fixed, for $t2$ and the memory access times for the bus-memory system, for $t5$. To model the timing behavior of the memory system, the firing delay of transition $t5$ is adjusted according to the number of tokens in all places $p4$ at each simulation cycle to reflect the memory access time obtained from the bus and memory system model.

Using the Petri net model of figure 3, along with the values obtained in section 4, we evaluated the processor utilization for multiprocessor systems of various sizes and different degrees of multithreading (the number of resident contexts allowed by the hardware). The configurations studied are representative of current shared-memory multiprocessors, with up to 32 processors. For all configurations studied, the number of memory modules used is the same as the number of processors in the system. The processors used have 1 (no multithreading), 2, 4 or 8 resident contexts.

To assess the gains obtained with multithreading, we then performed a series of simulations to compare multithreaded architectures with different degrees of multithreading but the same overall number of contexts. The cache miss rates used are similar to the rates observed for the MP3D and LocusRoute applications [14] from the SPLASH benchmarks: 7% and 0.6%, respectively. We have chosen these applications because they represent common parallel scientific applications with bad cache behaviors (MP3D) and good cache behaviors (LocusRoute).

The processor utilization versus number of contexts in the system for different degrees of multithreading is shown in figure 4. From this figure we verify that the increase in the level of multithreading increases the overall processor utilization. Also, for the same configuration, we observe that the processor utilization decreases as the size of the system is increased. This latter dependence reflects the bus and memory contention accurately modeled in the Petri net model of figure 2, and which is not accurately modeled in [10].

Comparing the curves from figure 4(a) with the curves from figure 4(b), we notice that the performance improvements from multithreading are highly dependent on the cache behavior of the applications. For applications with good cache behaviors (figure 4(b)), the utilization of a single-context processor is already high, the utilization of a two-context processor is almost 1 and the incremental gains from multithreading degrees of more than 4 become marginal. Applications with bad cache behaviors (figure 4(a)), on the other hand, seem to require higher degrees of multithreading to achieve a high processor utilization.

A side-effect of using multithreading in multiprocessor systems is the increase in the

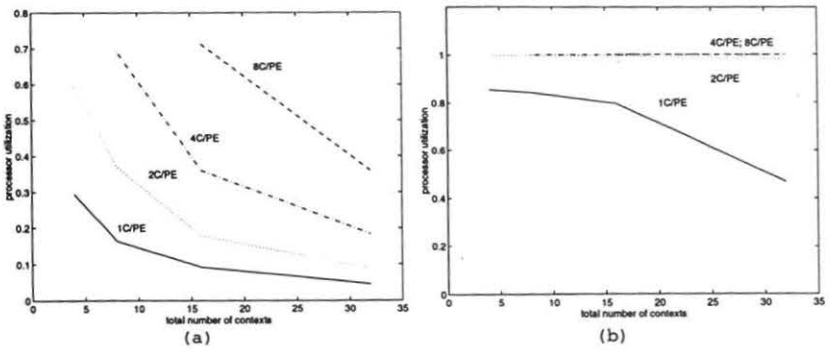

Figure 4: Processor utilization for different degrees of multithreading. The cache miss rates are similar to the rates observed in the (a)MP3D application and (b)LocusRoute application.

load offered to the network. This happens because processors with C resident contexts can have up to C outstanding memory requests, requiring larger buffers to hold the requests and a higher network and memory bandwidth to serve more requests in the same time. To verify this effect on the bus-memory system, we observed the effective memory access times for the configurations studied. The observed memory access times are shown in figures 5(a) and 5(b) for the MP3D and LocusRoute applications, respectively.

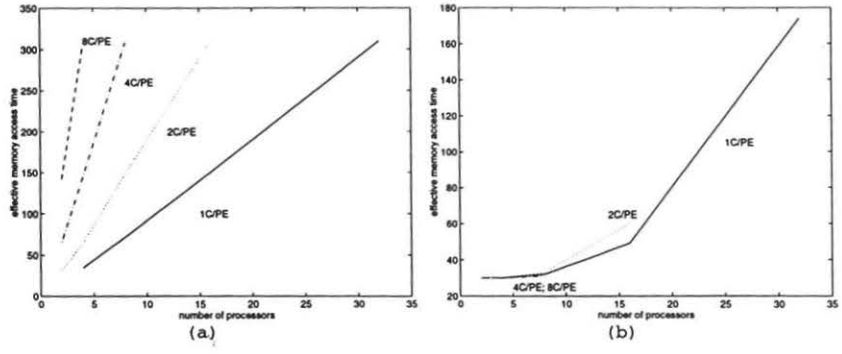

Figure 5: Effective memory access time in processor cycles for different degrees of multithreading. The cache miss rates are similar to the rates observed in the (a)MP3D application (b)LocusRoute application.

Analyzing figures 5(a) and 5(b), we observe that indeed systems with higher degrees of multithreading experience higher delays for the memory accesses. Again, we observe that applications with good cache behaviors tend to be less sensitive to effects of higher

degrees of multithreading. For a program with bad cache behavior (figure 5(a)) we notice that the effective memory access time is very close to the maximum observed time for the bus and memory subsystem with 32 contexts (see section 4). This result shows that for these applications multithreading, and thus multiple outstanding memory requests, together with a high cache miss, can lead the memory system to its maximum load.

6 Conclusions and Future Work

Multithreaded architectures can potentially hide both memory access latencies and pipeline dependencies and thus increase the processor utilization. Hiding long-latency operations and increasing processor utilization will have greater impact in future systems than they have today, because we can expect larger systems, with more hardware parallelism, and an increase in the memory and communication latencies in processor cycles.

In this paper we have investigated two important aspects of multithreading in a bus-based shared-memory multiprocessor: processor utilization and contention in the shared bus. We have shown that multithreaded architectures with 2 to 8 contexts have better processor utilization than single-thread systems with the same overall number of contexts. This increase in the processor utilization can be exploited to speed up parallel applications and to increase the throughput of multiprogrammed multiuser systems.

Differently from previous work with analytical models, we have modeled in detail the behavior of the bus and the memory system, and the contention for their access. We have verified that using multithreaded processors increases the contention for the bus and the overall memory access time. Thus, when designing multithreaded architectures with high degrees of multithreading, we must account for the increase in the network traffic and design the memory and interconnection system with sufficient bandwidth.

In this work, we have used a multilevel modeling methodology for Petri nets that allowed us to efficiently and accurately solve the multithreaded multiprocessor model. The flexible approach for handling multiple-level models allows us to easily modify both models and even replace the subsystems and their models without having to change other parts of the model. Some extensions to the work presented in this paper are then to model different internonnection networks and memory systems and different multithreaded architectures, such as Simultaneous Multithreading, which is more likely to be used in the next generation of superscalar processors.

In all our simulations we used the Petri net simulator RP_SIM, which proved to be a very powerful and flexible tool for dealing with generic timed Petri net models. This simulator is currently being improved with the addition of a graphical user interface that will make this tool more accessible to a larger group of users.

References

[1] Agarwal, A., Lim, B. H., Kranz, D., and Kubiatowicz, J., APRIL: A Processor Architecture for Multiprocessing, In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 104–114, May 1990

[2] Agarwal, A., Performance Tradeoffs in Multithreaded Processors, *IEEE Transactions of Parallel and Distributed Systems*, vol. 3, no. 5, September 1992

[3] Agerwala, T., Putting Petri Nets to Work, *Computer*, pp. 85–94, December 1979

[4] Alverson, R., Callahan, D., Cummings, D., Koblenz, B., Porterfield, A., and Smith, B., The Tera Computer System, In *Proceedings of the ACM ICS*, pp. 1–6, June 1990

[5] Cintra, M. H. and Ruggiero, W. V., A Tool for Modeling and Simulation of Computer Architectures Using Petri Nets, In *Proceedings of the VII Brazilian Symposium on Computer Architecture*, August 1995

[6] Culler, D. E., Sah, A., Schauser, K. E., von Eicken T., Wawrzynek, J., Fine-grain Parallelism with Minimal Hardware Support: A Compiler-Controlled Threaded Abstract Machine, In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991

[7] Halstead, R. H. Jr. and Fujita, T., MASA: A Multithreaded Processor Architecture for Parallel Symbolic Computing, In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pp. 443–451, May 1988

[8] Jain, R., *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, John Wiley, 1992

[9] Laudon, M., Gupta, A., and Horowitz, M., Interleaving: A Multithreading Technique Targeting Multiprocessors and Workstations, In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 308–318, October 1994

[10] Nemawarkar, S. S., Govindarajan, R., Gao, G. R., Agarwal, V. K., Performance Evaluation of Latency Tolerant Architectures, In *Proceedings of the 4th International Conference on Computing and Information*, pp. 183–186, May 1992

[11] Peterson, J. L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981

[12] Smith, B. J., A Pipelined Shared Resource MIMD Computer, In *Proceedings of the 1978 International Conference on Parallel Processing*, pp. 6–8, August 1978

[13] Tullsen, D. M., Eggers, S. J., and Levy, H. M., Simultaneous Multithreading: Maximizing On-Chip Parallelism, In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 392–403, June 1995

[14] Weber W. D., and Gupta, A., Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture: Preliminary Results, In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pp. 273–280, June 1989