A Tool for Modeling and Simulation of Computer Architectures Using[1] Petri Nets

Marcelo H. Cintra[I]
Wilson V. Ruggiero[II]

[I]Laboratório de Sistemas Integráveis
e-mail: mcintra@lsi.usp.br
[II]Laboratório de Arquiteturas e Redes de Computador
e-mail: wilson@larc.usp.br

Universidade de São Paulo

Abstract

The developments in the field of computer architecture, especially parallel systems, lead to the design of even more complex architectures, making it difficult to take decisions that would increase the performance of the system. In order to analyze objectively the advantages of different architectural choices, it is important to have modeling and analysis techniques and tools that can efficiently acquire data about the system's performance.

Petri nets have been used successfully as a modeling tool for computer architectures. However, the analysis of the complex nets needed to model real systems has become a limiting factor for using Petri nets. To efficiently use Petri nets for modeling these complex systems, one needs powerful computer simulation tools.

In this paper we present the program RP_SIM, an object oriented tool for the simulation of Petri nets. We use this simulator to analyze a simple computer architecture model, showing the viability of the use of Petri nets, together with the tool presented, to model general computer architectures.

1 Introduction

In the pursuit of more powerful computers, many different architectures have been proposed. Those computers have great differences with respect to the number and type of processors, memory configuration and hierarchy, interconnection network, synchronization support, and many other factors that affect the overall performance of the computer system.

To make an objective analysis of the advantages of a given architecture over other proposals, both to help in the design and to compare existing machines, it is convenient to use models and simulations. With this approach one can accelerate the design of new machines and reduce the need for experiments with real systems, which are usually expensive and time-consuming.

[1] This research was supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq - Brazil)

Petri nets have been used successfully to model computer architectures. This technique has proven itself adequate to model both parallelism and conflict[1], two important characteristics present in modern computer systems. Besides the capacity of the original net to model the flow of control and data, extensions such as timed and stochastic Petri nets are also powerful techniques for analyzing the performance of systems. Zuberek[16] used timed Petri nets to get performance indices of some computer architectures. Shaefer[14] used basic Petri nets to model the control of massively parallel computers. Petri nets can also be used to model a microprocessor's internal operation, as was done by Razouk[11].

The complexity of the analysis of the Petri nets created to model real computer systems requires the use of computer simulation tools. Many tools for that purpose have been proposed in the literature, as in [2] [3] [6] [9]. These tools were developed to run in different platforms and have distinct features and graphical user interfaces. We believe that an important feature that will prove to be extremely useful is the possibility of step-by-step simulation, as is done in [6]. By doing step-by-step simulation, one can analyze very complex Petri nets without the need to use traditional analytical methods that involve the computation of the net's reachability set, whose complexity tends to grow exponentially with the problem size.

In this paper we present the simulator RP_SIM as an object oriented tool that can solve complex Petri nets, step-by-step, in non-expensive personal computers. The organization of this paper is as follows: in section 2 we present a short introduction to Petri nets, in section 3 we present an overview of the simulator and its features, and in section 4 we use the simulator to analyze a queuing system M/M/2/B. In section 5 we use the simulator in a simple model of a computer architecture, mostly to demonstrate the potential of the use of Petri net modeling and the tool presented. In section 6 we present some conclusions and final remarks and we discuss some desired improvements in the program, which shall be implemented in the future.

2 Petri Nets

A basic Petri net may be defined as a graph created with three sets: a set of *places* P, a set of *transitions* T and a set of directed *arcs* A. Arcs may link places to transitions or transitions to places. A formal definition of a basic Petri net would be:

$PN = \{P, T, A\}$

$P = \{p_1, p_2, p_3, ..., p_n\}$

$T = \{t_1, t_2, t_3, ..., t_m\}$

$A \subset \{TxP\} \cup \{PxT\}$

A place p_i is said to be an input place of a given transition t_j if there is an arc directed from p_i to t_j. Similarly a place p_i is said to be an output place of t_j if there is a directed arc from t_j to p_i.

$I(t) = \{p \mid (p, t) \in A\}$

$O(t) = \{p \mid (t, p) \in A\}$

Besides the sets defined above, a marked Petri net is also identified by a *marking* M. Tokens are assigned to places and the marking in a given state of the net is defined by the set of all tokens currently assigned to each place in the net:

$M_S = \{m_{s1}, m_{s2}, m_{s3}, ..., m_{sn}\}$

where m_{si} represents the number of tokens in place i in the marking M_s. M_0 represents the initial state of the net.

The execution of a Petri net is done by the *firing* of transitions. A transition may fire if it is *enabled*, a situation that happens when all its input places have at least one token. The firing of a transition involves removing a token from all the input places and putting a token in each output place, thus generating a new marking M. Based on this definition, one can see that the number of tokens in the net changes when $|I(tj)| \neq |O(tj)|$.

With respect to these basic transitions, the firing is immediate and in the case that two transitions be in conflict, i.e., both are enabled in a given marking M_s and the firing of one transition disables the other, the choice of which one to fire is non deterministic.

Many extensions have been proposed to increase the modeling power of the basic Petri nets. Some extensions are:

- Arcs with multiplicity k: in this case a transition is enabled only if the number of tokens in each of its input places is greater than or equal to the multiplicity of the arc linking the place to the transition. The firing of a transition, then, involves removing from the input places as many tokens as the multiplicity of the incoming arc, and assigning to the output places as many tokens as the multiplicity of the outgoing arc.
- Inhibitor arcs: inhibitor arcs indicate that the absence of tokens in an input place enables the transition and the presence of a token disables the transition. The firing of the transition follows the same rules of the basic net except that the input place connected to the inhibitor arc remains untouched.
- Colored nets: in this special net, the tokens may be assigned an identification (color) and the enabling of a transition may depend on the colors of the tokens in the input places. The firing of a transition may remove tokens of a given color from the input places and put tokens of a different color in the output places, thus changing the colors of the tokens as they move through the net.
- Timed nets: the time factor may be introduced in the places so that a new token is only available to a transition after some time delay. More commonly, one associates time to transitions in which case there are two possibilities: a transition may require a given enabling time, after which the firing is immediate; or the transition may fire as soon as it is enabled but the firing may take some time.
- Stochastic nets: in this case one associates a random time to the firing or enabling time of transitions. In the GSPN model, transitions can also be defined as immediate, and they have priority of firing over transitions with non zero time. There is also the DSPN net that is a GSPN net that can handle both random and deterministic firing times.

For more details about Petri nets the reader is referred to Peterson's book[10].

3 The Simulator RP_SIM

The simulator RP_SIM is a tool for simulation of Petri nets that is capable of dealing with deterministic and stochastic Petri nets (DSPN) and which also supports colored tokens and arcs with multiplicity. This program was initially developed by Sangiorgio[12] and is currently being improved and extended. Different from the majority of the existing tools, this simulator presents an open architecture, allowing the user to extend the tool by adding its own code, written in a language developed with some features offered by the object oriented programming language C++[15]. This particular

feature of the simulator makes it highly flexible, increasing, however, the complexity of its use.

Due to its open architecture, the simulator is not an executable program, but consists of codes written in C++ - RP_SIM.CPP and SIM_OBS.H - that must be compiled along with the user's code, called RP_SIM.H. The executable program generated may be run in PC-type computers, without demanding requirements of hardware and software.

This program can simulate the step-by-step execution of Petri nets, firing the transitions and generating the new markings. A diagram of the overall operation of the simulator is shown in figure 1. As can be seen in figure 1, the simulation is divided in well-defined phases. Because of the simulator's open architecture it is possible for the user to access each one of these phases, both for data acquisition and for taking decisions regarding the flow of the simulation. The access to the simulation phases is done by code added to the routines shown in figure 1. This code must be written in C++ and it may include some extensions created to describe and deal with Petri nets. These extensions were created with some features of the C++ language, such as classes, operator overload and member functions. The extensions help describe and manipulate Petri nets and are described in more detail in [4] [12] and [13].

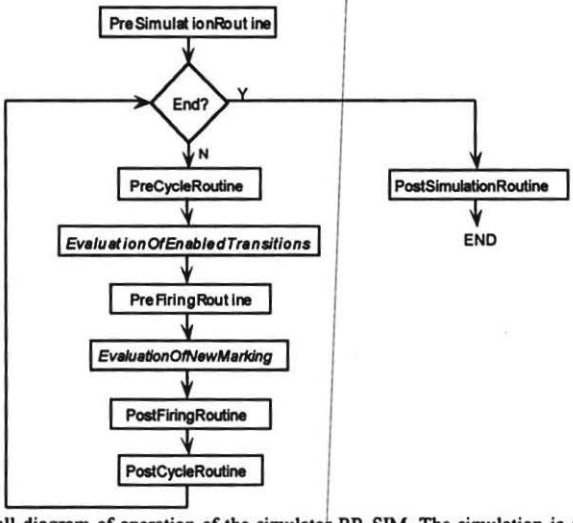

Figure 1 overall diagram of operation of the simulator RP_SIM. The simulation is divided in many phases that are represented by functions that can be accessed by the user. These phases correspond to the beginning and end of the simulation and the beginning and end of each simulation cycle

The simulator does not use traditional analytical methods to get Petri net parameters such as net invariants, boundedness, liveness, conservativeness and safeness. However, it can obtain many other important parameters in an iterative way, along the simulation, which is a feature that, as mentioned above, does not require the computation of the reachability set. Figure 2 shows the parameters related to places and transitions obtained by the simulator in a given simulation. Besides these values, the simulator also displays de total simulation time and the number of simulation cycles.

Place	Transition
mean number of tokens	time in firing state
mean holding time of a token	number of firings
number of cycles without tokens	mean firing rate
	occupation factor

Figure 2 parameters related to places and transitions given by the simulator. The mean firing rate is the ratio between the number of firings and the simulation time. The occupation factor is the ratio between the time in firing state and the total simulation time

Before starting the simulation, the program asks the user to specify the desired number of simulation cycles. Each simulation cycle corresponds to the firing of one transition and the number of cycles in a given run is an important parameter for the correctness of the results. Insufficient simulation cycles may lead to results that do not express the normal operation of the net in steady state.

The user has also the option of asking for the interruption of the simulation after the occurrence of some predefined event. In these cases, the simulation is paused and the partial results for the parameters in figure 2 are shown. The simulation can be interrupted in the following situations:

- only at the end of simulation (no interruption)
- at every N cycles
- when there are N tokens in a given place
- after the firing of a given transition

After the displaying of the partial results, the user has the option to terminate the simulation or continue it until the next occurrence of the event or until the number of cycles reaches the limit, whichever happens first.

Because of the simulator's open architecture and the object oriented programming in C++, the user can create his or her own kinds of places and transitions, derived from the basic types offered by the simulator. The user can also get data and statistics beyond the ones mentioned in figure 2. This can be done by implementing particular pre and post-firing and pre and post-cycle routines. In these routines the user can, for instance, perform input and output of data specific to his or her own model. These procedures may also reflect the user's net own semantics, plus the user can even alter the standard operation of the Petri net, if so desired.

4 A Queuing System

To test the effectiveness of the simulator RP_SIM, we developed a Petri net model of a queuing system of the type M/M/2/B. Figure 3 shows the model. This net was then extensively simulated and some important parameters were measured.

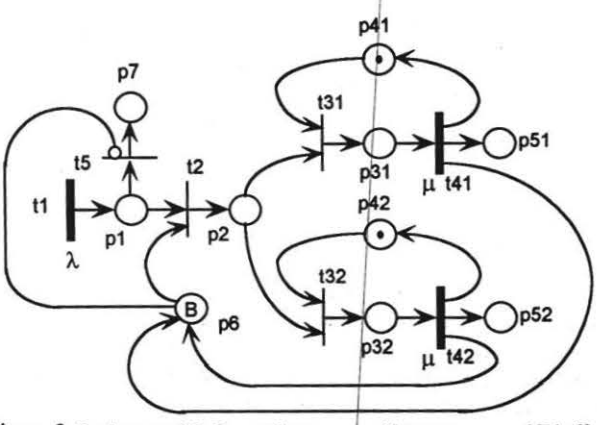

Figure 3 Petri net model of a queuing system with two servers and B buffers

In the model of figure 3, place p2 represents the client waiting on the line, and places p31 and p32 represent the client being serviced by one of the servers. The maximum number of clients in the system (buffer and servers) is limited by the number of tokens in p6, which is limited to B. Place p7 serves the purpose of collecting data about the services refused by the system. Since the simulator currently does not support inhibitor arcs, transition t5 is explicitly disabled if there are tokens in p6. Places p51 and p52 collect data about the services completed by each server. Transitions t1, t41 and t42 have an enabling time exponentially distributed with mean $1/\lambda$, $1/\mu$ and $1/\mu$ respectively. Thus, the arrival rate of requests is λ and the total service rate is $2*\mu$. The load in such a queuing system is defined as $\rho=\lambda/(2*\mu)$. Jain[8] presents a more detailed description of queuing systems and presents some analytical formulas for the system's performance that will be used in the following analysis.

Some simulations were performed to obtain the mean waiting time in the queue (E[w]), the mean number of clients in the queue (E[n]), and the loss rate (γ). The waiting time is the time spent by a client waiting to be serviced and the loss rate is the ratio between the number of clients refused because the buffers are full and the total simulation time. Figure 4a shows the simulation results for some values of ρ and figure 4b shows the values evaluated with the analytical expressions presented in [8].

	E[w]	E[n]	γ
0.1	1.04×10^{-3}	2.06×10^{-3}	0
0.2	3.91×10^{-3}	15.5×10^{-3}	0
0.3	9.81×10^{-3}	59.0×10^{-3}	0
0.4	18.4×10^{-3}	0.146	0
0.5	31.6×10^{-3}	0.315	0
0.6	60.7×10^{-3}	0.727	0
0.7	102×10^{-3}	1.44	0
0.8	202×10^{-3}	3.21	0
0.9	410×10^{-3}	7.39	3.60×10^{-3}

(a)

	E[w]	E[n]	γ
0.1	1.01×10^{-3}	2.02×10^{-3}	$O(10^{-66})$
0.2	4.17×10^{-3}	16.7×10^{-3}	$O(10^{-46})$
0.3	9.89×10^{-3}	59.3×10^{-3}	$O(10^{-34})$
0.4	19.0×10^{-3}	0.152	$O(10^{-26})$
0.5	33.3×10^{-3}	0.333	$O(10^{-20})$
0.6	56.2×10^{-3}	0.675	$O(10^{-14})$
0.7	96.1×10^{-3}	1.35	$O(10^{-10})$
0.8	178×10^{-3}	2.84	1.42×10^{-6}
0.9	422×10^{-3}	7.60	1.81×10^{-3}

(b)

Figure 4 mean waiting time (E[w]), mean number of services in the queue (E[n]) and loss rate (γ) (a) from simulation (b) from analytical solution. For the simulation: B=66, μ=10, N=2 and the number of simulation cycles was 100000

Comparing the data from figures 4a and 4b, we can verify that the values evaluated by the simulator are very close to the ones evaluated by analytical means. The error for the mean waiting time was on the average 5.5% and the differences were all between 0.8% and 13.5%. The error for the mean number of clients in the queue was on the average 5.6% and the differences varied from 0.5% to 13%. As the seed for the generation of pseudo-random values for the simulation is generated at the beginning of the simulation, if we increase the number of simulations performed for the same value of ρ it is possible to further reduce the errors. In figure 4 we can also observe that the number of services refused was null for the majority of the cases. This happened because the number of cycles of simulation is finite and not enough cycles were simulated to reach the situation of firing of t5. On the other hand, the values from the analytical expressions are for an ideal situation with an infinite observation time. From the analytical values we can see that the loss rate was extremely small, requiring a huge number of simulation cycles to achieve the expected results.

In addition, we also conducted another set of simulations to get the number of buffers required to keep the loss rate below 0.01 ($\gamma < 0.01$). The simulations were performed for the same queuing system and the results for the simulation and the analytical expressions are shown in figure 5.

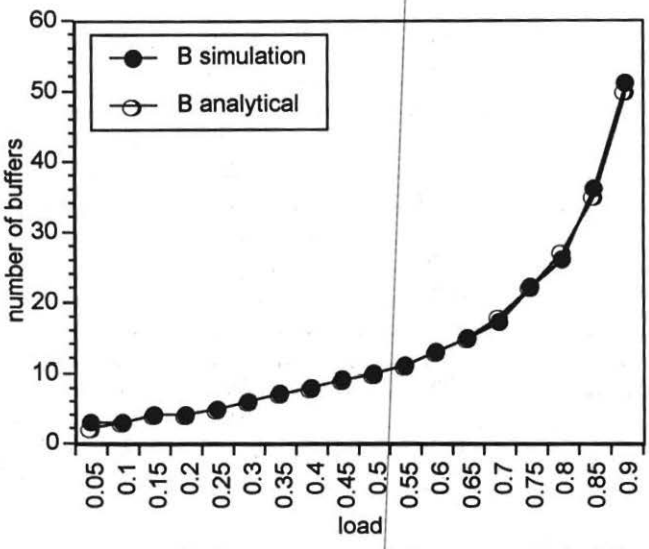

Figure 5 number of buffers required to keep the loss rate bellow 0.01. For the simulation: μ=10, N=2 and the number of simulation cycles was 100000

Looking at figure 5 we can see that the simulation results were extremely close to the expected values. In all cases the relative errors were very low, 5.9% in the worst case. Moreover, one would expect the relative errors to be high because the number of buffers is discrete. So, for example, if B=2 is not sufficient to keep γ within the desired limits, even for a little amount, the next possible value for B would be 3, generating a relative error of 50%. This peculiarity could increase the simulation errors, but even under this consideration the simulator reached almost exact results.

5 A Computer Architecture Model

Using Petri nets, we developed a model to represent, in a simplified manner, a massively parallel computer architecture. This model represents a MIMD computer with N processors connected through any static point-to-point interconnection network. This computer has four levels of data access: register, cache memory, local memory and remote memory (through distributed shared memory or message passing). The operation of the computer under average conditions or under a specific algorithm can be modeled through the probabilities of access to data at each one of these levels. Figure 6 shows the Petri net model.

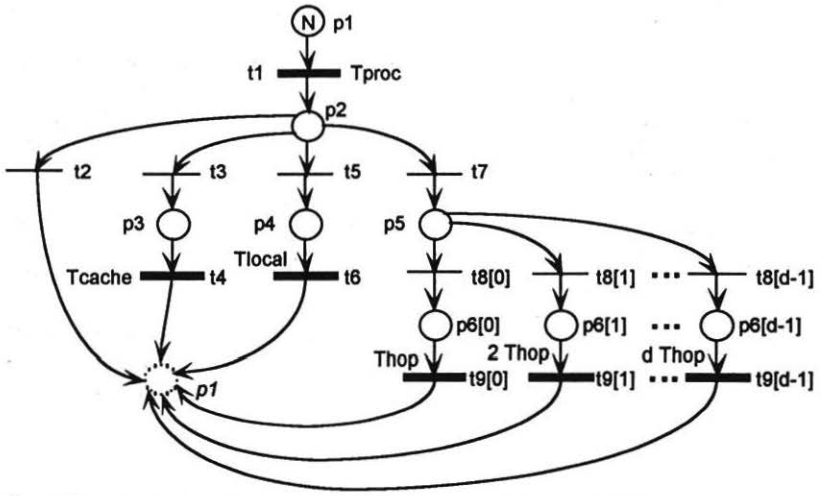

Figure 6 Petri net model of a massively parallel computer architecture with N processors and point-to-point static interconnection network

Figure 7a shows an example of interconnection network that can be modeled by the Petri net in figure 6: a 2D mesh with 16 processors. The internal organization of a processing node with microprocessor, cache and local memory is shown in figure 7b.

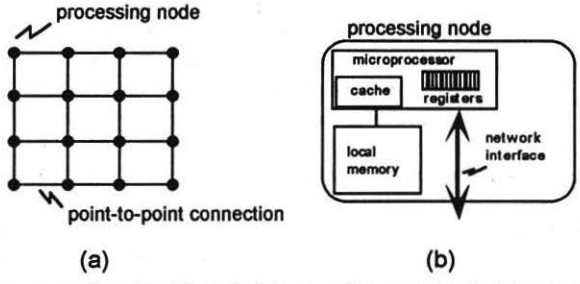

Figure 7 (a) representation of a 2D mesh interconnection network (b) internal organization of a processing node

In the model, the number of tokens in p1 indicates the number of processors that are executing instructions. The instruction fetch is assumed to take one clock cycle, and to execute the instruction the processor performs a data fetch. The data can be in the registers, in cache, in local memory or in remote memory. In the case of the data being in register, the access can be assumed to be immediate if a pipeline is used (actually the data fetch takes another cycle but the pipeline can hide this latency). In the other cases the data fetch imposes a delay during which the processor goes idle. This model does not currently implement latency hiding techniques present in some modern high performance computers. To model the parallel execution of instructions and data access by many processors, the firing rates of transitions t1, t4, t6 and t9[i] are adjusted according to the

number of tokens in their input places. So, for instance, the firing rate of t1 in a given simulation cycle is given by m(p1)/Tproc, where m(p1) is the number of tokens in p1. This technique for modeling parallel activities is not exact but is a good approximation for the ideal model, which would consist of many places and transitions connected in parallel and with fixed firing rates. The blocks formed by p6[i], t8[i] e t9[i] represent the access to a remote memory distant i+1 hops. The enabling times of transitions t9[i] have an exponential distribution and are proportional to the number of hops to the processing node and inversely proportional to m(p6[i]). Thus, for instance, the mean access time for data distant 5 hops is (5*Thop)/m(p6[4]), where Thop is the access time for data located in a neighboring node. With this Petri net we can model a system with an arbitrary number of processors and an interconnection network with diameter d, with a reasonably small number of places and transitions.

Initially, we used the above net to model a massively parallel computer with 512 processing nodes, with one processor at each node and with a 3D torus interconnection network. The diameter d of a torus with N nodes, dimension D and width w is given by:

$$w = \sqrt[D]{N}$$

$$d = D \cdot \left\lfloor \frac{w}{2} \right\rfloor$$

So, for N=512 and D=3 we have d=12. An example of a real computer architecture that is similar to the model described is the Cray T3D system, which has a 3D torus interconnection network with at most 1024 processing nodes with local memory and two processors per node[5]. To evaluate the probability of access to the different levels of the memory hierarchy we used the results presented in [7] for the instruction mix for a general RISC processor. The probabilities of access to the different levels are modeled by the random switch formed by t2, t3, t4 and t7, whose probabilities of firing are Preg, Pcache, Plocal and Prem, respectively. Thus, we assumed that for data located in the same processing node the probability that the data be in the register is 75%, i.e., Preg=(1-Prem)*0.75. If the data is in the node but is not in the register the probability of the data being in the cache is 95%, i.e., Pcache=(1-Prem)*0.25*0.95. The probability that the data be in the local memory is then 5%, i.e., Plocal=(1-Prem)*0.25*0.05. We performed a series of simulations for many values of Prem and we estimated the performance loss due to inter processor communication, using as reference the case N=512 and Prem=0, which represents the ideal case of no communication overhead. In this initial situation we considered that the remote data are equally distributed among the other processing nodes. So, we supposed that on the average this situation would be equivalent to accessing all remote data from a node d/2 hops away. For these simulations we used processing time Tproc=1, cache access time Tcache=3, local memory access time Tlocal=10 and remote memory access time Thop=1000 cycles. The results obtained are shown in figure 8.

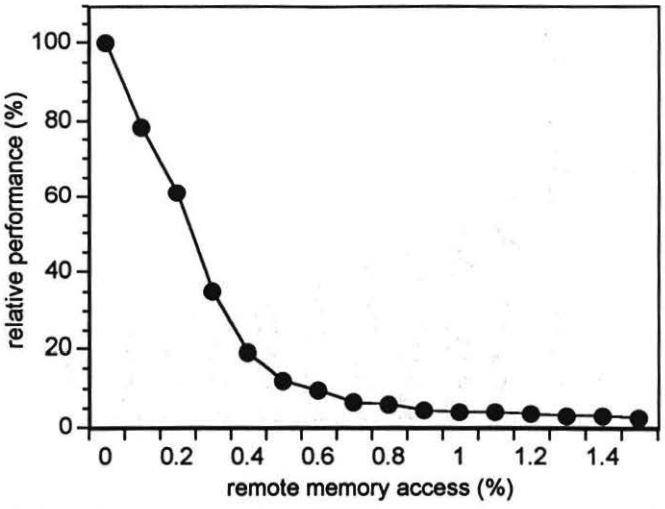

Figure 8 relative performance loss for the case of 512 processors, using as reference the ideal case of no inter processor communication overhead. The horizontal axis is the probability of access to data in the remote memory. The number of simulation cycles used was 400000

To calculate the performance loss we first considered the ideal case, with no remote access, and we obtained the number of instructions executed per unit time for that case (execution rate). To measure the number of instructions executed we added the number of firings of transitions t2, t4, t6 and t9[i]. The execution rate is the ratio between the number of instructions executed and the total simulation time. We then obtained the execution rate for the other cases and the relative performance loss for each case is the ratio between this execution rate and the sequencial execution ratc.

Based on figure 8 we can detect that the performance of the architecture is extremely sensitive to the data distribution. With a small fraction of the data being accessed remotely the achievable speedup may become very small. This matches the results found in real massively parallel systems, in which the actual performance in most applications is just a small fraction of the peak performance.

We then conducted an analysis of the sensibility of the architecture to the placement of the remote data. We considered in this case that the overall probability of access to remote data is 0.5%. To analyze the case of hotspots we considered that for remote data access the probability of the access being to a generic node k hops away is always 2%, except for the hotspot, whose probability of access given that the access is remote is 78%. Figure 9 shows the results of these simulations.

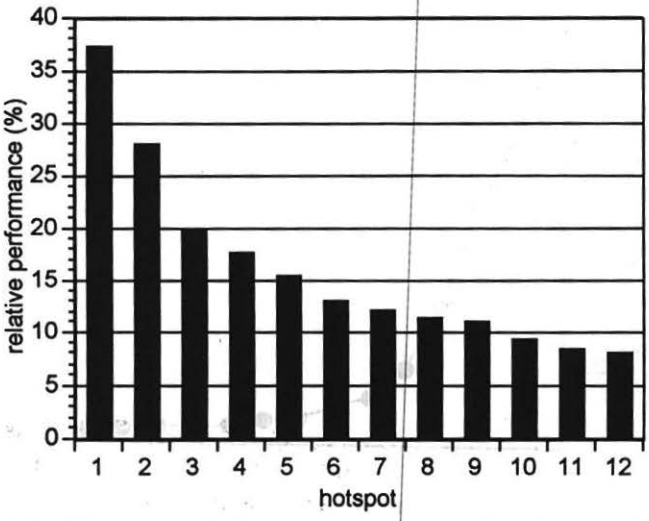

Figure 9 relative performance loss for the case of 512 processors and remote access with hotspot. In these cases the remote access probability is 0.5% and the number of simulation cycles is 400000

From figure 9 we can observe that the performance of the system suffers a significant degradation as the data required for the computation move to the more distant nodes. The impact of moving the data away is, however, bigger for the nodes closer to the processing node. This suggests that if an optimal data distribution is not possible, thus generating a high inter processor communication, then the impact of the actual remote access time is no longer so significant. Considering the preceding assertion, reducing the remote access time in machines with a large number of processors working together in a single problem is not as important as achieving a better data distribution and reducing the need for inter processor communication.

The model presented in this section, although simple, can model a general class of massively parallel computer systems, and many important performance indices can be obtained. We intend to extend the model, increasing the level of detail, making it an even better model of the real Cray T3D system. We also plan to embed existing algorithms in the model through the probabilities of access to the different levels of the memory hierarchy. With that, it will be possible to model both the computer hardware and the algorithm.

6 Conclusion and Future Work

The ability to analyze computer architectures through modeling and simulation is increasingly becoming an important issue in the design of new machines with reduced cost and development time. The analysis performed in this paper, though with a simple model, presented very promising results, showing that the joint use of Petri nets and the simulator RP_SIM can be used to analyze real computer architectures.

The simulator has a unique configuration, based on a step-by-step simulation and iterative evaluation of the relevant net parameters. We believe that this configuration can

solve nets with a higher degree of complexity than current Petri net tools can. Despite this unique configuration, the correctness of the simulator was verified through the analysis of a queuing system, whose analytical solution is known.

Though the simulator proved to be a powerful tool, its user interface still needs some refinements, specially the addition of a graphical user interface. The fact that the simulator uses an unusual method for solving the Petri net, does not discard the possibility of inclusion, in future versions, of traditional methods based on the reachability set. These additions would greatly improve the simulator's power and ease of use.

With the current features and future improvements we hope to offer a powerful tool both for teaching purposes and for helping in the development of new complex computer architectures.

References

[1] Agerwala, T., "Putting Petri Nets to Work", Computer, December 1979, pp. 85-94

[2] Atamna, Y., "RPTS: A Tool for Stochastic Timed Petri Nets", Proceedings of the 5th International Workshop on Petri Nets and Performance Models, 1993

[3] Ceska, M. and Skacel, M., "Petri Net Tool PESIM: the tool for Petri net drawing, simulation and analysis", Proceedings of the 5th International Workshop on Petri Nets and Performance Models, 1993

[4] Cintra, M. H., *Manual do Usuário do Programa RP_SIM - Simulador de Redes de Petri Interpretadas*, Technical Report, Universidade de São Paulo, 1994

[5] Cray, *Cray T3D System Architecture Overview Manual*, Cray Research, 1993

[6] Gellot, F., Carre-Menetrier, V., Lecolier, G. V., "PETRILAM: A Tool for Petri Nets Analysis and Simulation", Proceedings of the 5th International Workshop on Petri Nets and Performance Models, 1993

[7] Hennessy, J. and Patterson, D., *Computer Architecture a Quantitative Approach*, Morgan Kaufmann, 1990

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

[9] Lindemann, C., "DSPNexpress: A Software Package for the Efficient Solution of Deterministic and Stochastic Petri Nets", Proceedings of the 5th International Workshop on Petri Nets and Performance Models, 1993

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