Exploiting Petri nets Model for Performing Multithreaded Processors

Chin-Yung Chen[Ψ], Dong-Liang Lee[Ψ], Rifu Du[Φ],and Jerry Tang[¥]
Ψ, ¥: The Department of Information management
Φ: The Department of Electronic Engineering
¥: Department of Information management, Chihlee Institute of Commerce
St. John's & St. Mary's Institute of Technology
499, Sec. 4, Tam King Road, Tamsui, Taipei, Taiwan
{cychen, lee, tu} @mail.sjsmit.edu.tw, jerrytang@mail.taipeilink.net

Abstract

A Petri nets is an abstract and formal model of information flow. Petri nets provide a natural representation of systems where control and state information is distributed Petri nets can also be used to represent system hierarchically. Petri nets theory supply analysis techniques that are useful for verifying design before they are synthesized.

In this paper, we describes a Petri nets based methodology for modeling and evaluation multithreaded processors. A general purpose Petri nets simulator has been developed using trace-driven approach. Using this simulator, the execution of the Petri nets models of some sub-models have been simulated. In this paper we only concern on the scheme that makes the Petri nets to model the communication control of multithreaded processors, and to compare the performance and cost/performance ratio (CPR) of the multithreaded processors to single thread processor.

Keywords: Petri nets (PN), multithreaded processors (MTP), cost/performance ratio (CPR), and transaction.

1 Introduction

While signal chip processors become more powerful, the use of pipelining scheme to speed up instructions fetching, decoding, execution, and write back have become more prevalent. The most model processors are supported new features to improvement the performance such as instructions prefetching, I-cache and D-caches, parallel and distributed techniques, and multithreading scheme. To improve the performance gap between memory systems and processors, and reduce the memory latency incurred by access from memory. There are many methods of solving this latency problem, the latency hiding techniques such as coherent cache, prefetching, and relaxed memory models have been developed. Multithread is the attractive techniques to reduce processor idling time and to raise processor utilization by fast context switching to a read-to-run thread [6].

Multithreaded processors utilize the simple and efficient sequential execution technique of control flow combined with data flow like concurrency. This supports the conceptually simple but quite powerful idea of rescheduling rather than blocking when waiting for data transmission latencies. A multithreaded processor's efficiency is determined by four parameters: (1) the number of contents supported by the hardware, (2) the cost of switching between contents, (3) the number of cycles executed between context switching and (4) the characteristic latency of the operations that are to be hidden [8,11,13,15].

In processors design, understanding the detailed timing of such processors is extremely difficult and therefore understanding the bottlenecks in systems that use them is also difficult. The use of Petri nets provides convenient means of specifying concurrent systems at the behavioral level, their analysis and verification, as well as synthesis of their hardware implementations [19]. The Petri nets (PN) had been extend many kinds of Petri nets classification and developed hierarchical. This means that we can construct a large Petri net by relating smaller Petri nets to each other, in a well-defined way. The hierarchical construct of Petri nets play a role similar to that of submodels, procedures and models of main model.

The possibility of using both top-down and bottom-up approaches in the design processor means Petri nets are efficient tool for rapid prototyping. This paper addresses the needs for tools, which can permit rapid construction of faithful models of multithreaded processors.

Petri nets are becoming increasingly attractive as a formal model for hardware system design. The graphical nature of the Petri net notation makes it more attractive to circuit designers than algebraic notations.

Several multithreaded-processor architectures had recently been proposed and implemented of different multithreading methods [1,3,4,5,6,7,9,10]. In this paper, we design of simple multithreaded

processors to be developed future into a fully operational version. We demonstrate the design method that is used a Petri nets to model their modeling power. We also show how to use a trace-driven simulation tools to simulation, analyze and compare the performance of the proposed multithreaded processors.

For the performance analysis of the multithreaded processors, we will compare the cost/performance ratio (CPR) of multithreaded processors to the single thread processor architecture one's. Through this examine, we have verified that the multithreaded processors has higher CPR than single thread processor one's.

The remainder of this paper is organized as follows: Section 2 introduces the Petri nets and highlights the properties of the model. Section 3 makes the model of multithreaded processors using the Petri nets. In Section 4 illustrates the SES/workbench simulation tools to verify the multithreaded processors with Petri nets. The performances are analyzed in Section 5. Finally, we remark the conclusions and take into the future in Section 6.

2. The Properties of Colored Petri nets

The concept of the Petri net is a graphical and mathematical tool; it had origin by Carl Adam Petri's dissertation in 1962[20]. There are many researchers have proposed extensions to the Petri net notation for the accurate modeling of circuit properties such as timing information, Timed Petri nets [24], Stochastic Petri nets (SPN) [16,18] that is model associated with each transition an exponential firing time, and Colored Petri nets [14,26]. The Colored Petri nets (CPN of CP-net) are a formalism, which extends ordinary Petri nets with data types and modularity. This makes them a very efficient tool for designing systems dealing with parallel and distributed, or multithreaded processors. The Colored Petri net provides a framework for the design, specification, validation, and verification of system. Colored Petri nets are promising tool for describing and studying information processing system. They are characterized as being concurrent, asynchronous, distributed and parallel, non-deterministic, and/or stochastic Colored Petri nets can be used as a visual communication and similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and current activities of systems.

A Petri net is combined of a particular kind of directed graph, which is together with an initial state called the initial marking and a bipartite graph that is consisted of two kinds of nodes, places (P) represents the state of a Petri net and transitions (T) represents the actions of a Petri net. Between the places and transitions, where arcs are either from a place to a transition or from a transition to a place.

In graphical representation, places are drawn as circles, transitions as bars or boxes, and arcs are labeled with their weights, where a K-weighted arc can be interpreted as the set of K parallel arcs. The basic Petri net is usually defined as a system composed of a finite, nonempty set of places, transitions and directed arcs. Usually the set of places connected by directed arcs to a transition is called the input set of a transition, and the set of placed connected by directed arcs outgoing from a transition, its output set [17,19,22].

In modeling of all kinds of Petri nets, are using the concept of conditions and events, places represent conditions and transitions represent events. Some typical interpretations of transitions and their input places and output places are shown in Table 1. A formal definition of a Petri nets are given in Table 2.

The behavior of many systems can be described in terms of system states and their changes. In order to simulate the dynamic behavior of a system, a state or marking a Petri nets is changed according to the following transition rule [2,19]:

Table 1. Some type interpretation of transitions and places

Input Places	Transition	Output Places
Preconditins	Event	Postconditions
Input data	Computaion	Output data
Input signals	Signal processor	Output signals
Resources needed	Task or job	Resources released
Conditions	Cause in logic	Conclusions
Buffers	Processor	Buffer

Table 2. Formal definition of a Petri nets

A Petri nets (PN) is a sample, CPN (P,T,F,W,M_0) where:

$P = \{P1, P2, ...,Pn\}$ is a finite set of places,
$T = \{T1, T2, ...,Tn\}$ is a finite set of transitions,
$F \subset (P*T) \cup (T*P)$ is a set of arcs (flow relation),
$W: F \rightarrow \{1,2,3,...\}$ is a weight function,
$M_0: P \rightarrow \{0,1,2,3.....\}$ is the initial marking,
$P \cap T = \Phi$ and $P \cup T \neq \Phi$.
A Petri nets structure $N=(P,T,F,W)$ without any specific initial marking is denoted by N.
A PN with the given initial marking is denoted by (N, M_0)

1) A transition is said to enabled if each input place of transition is marked with at least w(p,t) tokens, where w(p,t)is the weight of the arc from place to transition.

2) An enabled transition may or may not fire that depends on whether or not the event actually takes place.
3) A firing of an enabled transition removes w(p,t) tokens from each input place to transition, and adds w(t,p) tokens to each output place of transition, where w(t,p) is the weight of the arc from transition to place.

3. Implement Multithreaded Processors with Petri Nets

We follow the top-down design methodology in which after deciding upon the top-level specification. We refine the specification until we reach the implemental level. The main objective of the first design stage is to produce a labeled Petri nets that have transitions labeled only when actions of the corresponding modules. During the second stage, we transform this high-level labeled Petri net into one that contains explicit transactions of control elements, and can therefore be translated into a circuit.

Petri nets model of a hardware system consists of a description of a set of possible events in the system. Each event has pre-conditions that must be true in order for the event to occur and post-conditions that become true once the event has occurred in multithreaded processors. Typical events include: initial thread dispatch, initiate instruction prefetch, initiate instruction dispatch OPcode to function units, initiate operand fetch, the storing of results, needed data read, result data commit, and the handshaking events of communication unit among threads.

We start with the initial specification, initialize transition, of Petri nets shown in Figure 1. This follows the most abstract specification of the multithreaded processor's operation: it alternates between threads context switch modes through communication unit. Thus, the initial specification is simply a labeled Petri nets with transitions representing those modes.

In this multithreaded execution model, a program is a collection of partially ordered threads, and a thread consists of a sequence of instructions, which are executed, in the traditional pipeline processor model. The executions of threads for thread slots are model by thread issue transition. The executions of instructions are model by the conventional pipelined processors and fired by the program counter (PC) transition. Each thread slot is a logical pipeline processor. Typical pipeline architecture has five stages, including instruction fetch, instructions decode, operand fetch, execution, and results write back. We extend the data memory pipeline stage and embed a communication unit to transfer the needed data via handshaking protocols.

While the instructions decode stage, the needed data accessed and the Opcode decoded can be operated on simultaneously. Thus the thread slot must decide where read the needed data, which may be obtained from general-purpose register (GPR), D-cache, or other thread slot. We use the data selection place to present the decided and to obtain the needed data from the one of above three data source.

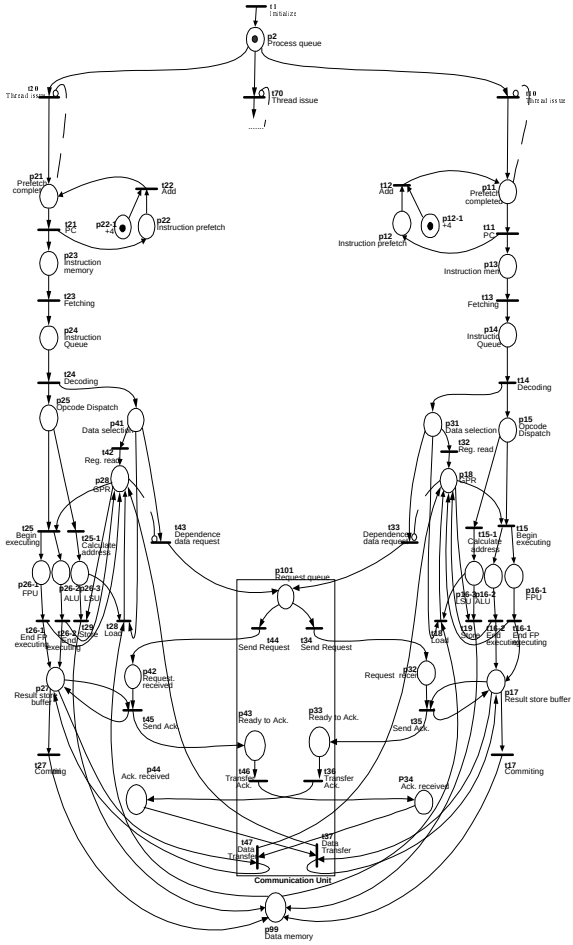

Fig. 1. The Petri net model for multithreaded processors.

The context switch among threads is executed of a communication unit. The communication handshaking is started the transition of dependence data request. Through the communication protocols (shown in Figure 2), the needed data are transferred from one thread slot to other thread slot. First, the requesting thread slot sends the dependence data request signal to the request queue of communication unit, then the queuing request is sent to the requested threaded slot from communication unit. If the needed data is resident in the requested thread slot and it is needed for the requesting thread slot, then the requested thread slot sends a acknowledge signal to

the communication unit, immediately. Thus, the acknowledge signal is through the communication unit transmitted to the requesting thread slot. Finally, upon the requesting thread slot receiving the needed data, the requesting thread slot send a signal to excite the data transfer transition of communication unit to active. Thus, the needed data are transferred from the requested thread slot to the requesting thread slot via the communication unit.

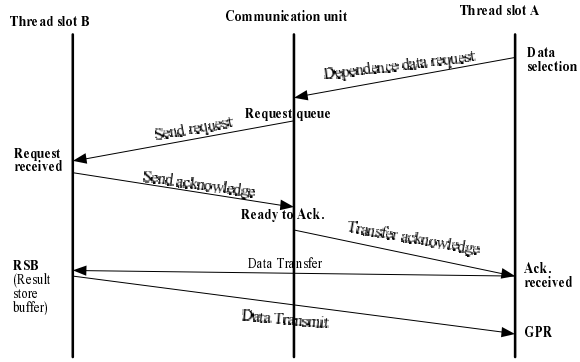

Fig. 2. Communication protocol among thread slots.

4. Simulation Model

Our simulation model is done by using SES/workbench simulation environment on a SUN SPARC workstation. SES/workbench is a design specification, hierarchical, modeling and simulation tool. It can be used to construct and evaluate proposed system designs, and to analyze the performance of implemented systems. The SES/workbench®, which is an integrated collection of software tools for specifying and evaluating system model design hierarchically [23]. The simulator simply generates a trace, which is simply the description of the initial states of the system, followed by a series of state deltas describing how the state of the system changes over time

A statistical simulation uses populations and utilization to characterize the workload parameters. Populations are the number of transactions present at the node, which in a SES/workbench model represents the manipulation of a physical or logical resource or some other processing step, in a model or sub-model. Utilization is the number of elements (servers or resources) of use on a node of model or sub-model. SES/workbench is an integrated set of software tools that specify and evaluate system designs. The major components of SES/workbench are: SES/design — a graphically oriented design-interface module, for specifying, or capturing, a system design— and SES/sim— a transactions, which is similar to token, and simulation module for converting the design specification into an executable simulation model.

We use SES/workbench to evaluate the performance and correctness of a design of a Petri nets system. We can specify arbitrary C declarations, expressions, statements, and functions almost anywhere in a SES/workbench model. Performance is evaluated by simulating the operation of the model. Correctness is evaluated by executing, during the simulation assertions called consistency constraints, attached to any component in the specification of a design.

Performance measurements are reported based on total execution time and as a percentage speedup over a multithread machine with different instructions size and different thread slots. The system model is implemented as a hierarchy of nested sub-models (structured modeling approach). This method keeps the model simple and easy to understand, because it represents a block of a function as a sub-model without including all the implementation details for that block or function. Different higher-level sub-models of the system can call the same lower level sub-model. In this case the type of the transaction entering the lower level sub-model indicates the feature of the service requested by the higher-level sub-models.

The main graphical part of the PN model of multithreaded processor is described as SES/design. The main flow of data takes place from the top to bottom. At first glance, the model shows the main stage of multithreaded processor. The main model of this multithreaded processor is constructed of two parts, thread-level-parallelism and instruction-level-parallelism. The thread dispatcher_submodel consists of several thread slots to simulate the thread concurrent execution and thread context switching when instruction suspends, dispatch instructions to each thread slots, and the needed data communication between threads.

The thread slot submodel is used to simulate the thread slot in which the block-instruction is executed. The thread transactions entering the thread slot submodel request the necessary number of blocks for the thread dispatcher to generate the proper number of new transactions to the corresponded thread slot. If the needed data is lack for the corresponded thread slot, then a dependence data request signal is sent from the corresponded thread slot to the communication unit.

The instruction-pipeline submodel is constructed as logical pipeline architecture's, which consists of the program counter, instruction register, decoder, register file, function unit, write back and the committed of the execution results to memory. When the instructions load into this submodel, which are processed by the logical pipeline processor.

The communication unit submodel, which

transfers the data among threads when data dependency occurs. Thus the needed data is transferred from the result store unit of requested thread slot to the function unit of the requesting thread slot via the handshaking control.

5. Analyzing the Performances

When the multithreaded processor model is simulated, a transaction is generated amongst other things, contains the timing data that we are expected. The simulator takes binaries compiled with gcc for the hierarchical model. A statistical simulation uses populations and utilization to characterize the workload parameters:

Throughput is defined to as follows, a total amount of transactions done in a give simulated time, which is defined as the average number of transactions executed by the submodel per second (TPS).

All results are presented in this section. Those are obtained by simulation of the corresponding net models. Figure 3 shows the throughput of the multithreaded processor as a function of 4 thread slots and a one-thread slot, which is similar to a traditional pipeline structure of the uni-processor.

Referring to Figure 3, though the multithreaded processor has higher data and instructions access frequencies (transactions) from D-cache and I-cache, respectively. The multithreaded processor has lower instruction fetch from instruction queue and data result buffer (DSR) than the one threaded slot. This results cause that the instructions dispatched by the thread dispatcher unit and the needed data of each thread are directly access from the owner's register or request to other thread-slot's DSR via the communication unit. Thus the multithreaded processor (MTP) has higher transactions transfer than single thread processor for data and instruction access of D-cache and I-cache, program counter, and transactions communication of communication unit. But the transaction numbers for each transitions of pipeline stage, such as instruction fetch, results writeback and commit, and execute of multithreaded processor, are lower than the single thread processor.

Comparing the transactions transmit ratio (TTR) of data of the multithreaded processors to single thread processor, the TTR is defined as follows:

TTR = (The transaction of DSR * the transactions of communication unit)/the transactions of D-cache.

where, we set the TPS of the single thread processor is 1. Thus, we obtain the TTR of multithreaded processor is 15.04, and the TTR of the single thread processor is 1.85.

The performance of multithreaded processors and single thread processor are 15.04 and 1.85, respectively. The cost/performance ratio of both is 3.85 (15.04 is divided 4 thread-slots) and 1.85 (1.85 is divided 1 thread slot) for multithreaded processors and single thread processor, respectively.

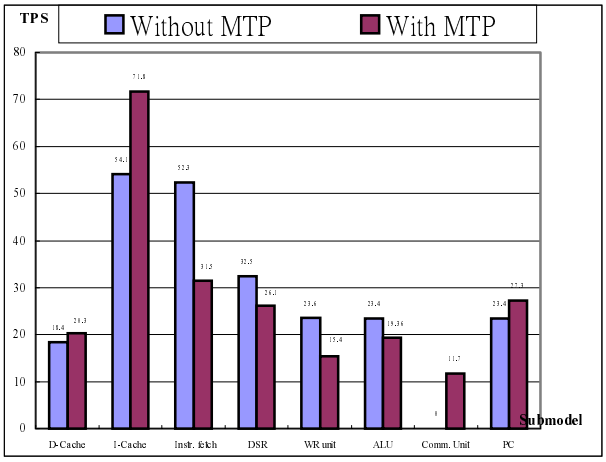

Fig. 3. The throughput between the proposed multithreaded processors and the traditional pipeline one.

6. Conclusions

In this paper, the use of Petri nets to model multithreaded processors is demonstrated. Exploiting model based analysis is helpful in the design multithreaded processors. Petri nets can be modeled a hierarchical model and be used to describe and analyze multithreaded processors system. In this past, various high performance processors are described with single-threaded processors and use finite state machine (FSM). In this paper, we attempt to exploit Petri nets to model the multithreaded processors and use the trace-driven simulation tool, SES/workbench, to construct the hierarchical level. The SES/workbench is effective to build the Petri nets hierarchical level specification.

In this paper we only concern on the scheme that makes the Petri nets to model the communication control of multithreaded processors. We also have verified that the multithreaded processors have the higher cost/performance ratio than the single thread processor, the CPR is 3.85 and 1.85, respectively. As for, the Petri nets model of instructions branch and interrupt control for pipeline or multithreaded processors. We level these research topics to our future works.

References

1. Haitham Akkary and Michael A. Driscoll, "A

Dynamic Multithread Processor," The proceedings of the 31st Micro-architecture ACM/IEEE Annual international Symposium, 1998, pp. 226-236.
2. L. Anneberg and M. Singh," Petri Net Approach to Software Development Under Pipeline and Parallel Processing Architecture," IEEE 1990, pp. 653-656.
3. F. P. Burns, A. M. Koelmans, and A. V. Yakovlev, " Analyzing Superscalar Processor Architectures with Coloured Petri Nets," The Journey of STTT, Feb. 1998, pp.182-191.
4. Gregory T. Byrd and Mark A. Holliday, "Multithreaded Processor Architecture," IEEE SPECTRUM, August 1995, pp. 38-46.
5. R. S. Chappell, J. Stark et al, "Simultaneous Subordinate Microthread (SSMT)," The proceedings of the 26th Computer architecture international Symposium, 1999, pp. 186-195.
6. Y-K Chong and K Hwang, "Performance Analysis of Four Memory Consistency Models for Multithreaded Multiprocessors," IEEE Transactions on Parallel and Distributed System, Vol. 6, No. 10, Oct. 1995, pp. 1085-1099.
7. Marco Fillo, Stephen W. Keckler and Willian J. Dally, "The M-Machine Multicomputer," The Proceedings of the 28th Annual international Symposium on Micr- architecture, Mon. 29-Dec. 1, 1995, pp. 146-156.
8. Chris Jesshope and Bing Luo," Micro-thread A New Approach to Future RISC," The proceedings of the 5th Computer Architecture Conference, Australasiar, 1999, pp. 34-41.
9. K. M. Kavi, H. S. Kim et al., "A Decoupled Scheduled Dataflow Multithreaded Architecture," The proceedings of 4th International Symposium Parallel Architectures, Algorithms, and Networks, 1999. (I-SPAN '99), pp. 138 –143.
10. Y.-H. Kim, S. H. Kim et al., " Exploiting the Locality of Data Structure in Multithreaded," In Proceedings of Parallel and Distributed Systems, 1996, pp. 352 –358.
11. K. Kimura, H. Hirata et al., "Evaluation Method of Micro architecture for multithreaded Processor," In proceedings ISIE'94, 1994, pp. 53-58.
12. K. Khare, "Modeling of Various Addressing Schemes for Microprocessors Using Petri Net," IEEE 1988, pp. 680-684.
13. Hari Krishna and R. Govindarajan, "Classification and Performance Evaluation of Simultaneous multithreaded Architecture," The proceeding of 4th International Conference on High-Performance Computing, Dec. 18-21, 1997, pp. 34-39.
14. L. M. Kristensen, and S. Christensen," The practitioner's Guide to Coloured Petri Nets," The Journal of STTT, Feb. 1998, pp. 98-132.
15. Z. Li and J.-Y. Tsai et al., "Compiler Techniques for Concurrent Multithread with Hardware Speculation Support," In Processing of the 9th Workshop on Language and Compilers for Parallel Computing, LNCS, August 1996, pp. 175-191.
16. M. A. Marsan and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multithreaded Systems," ACM Transactions on Computer Systems, Vol. 2, No. 2, May 1984, pp. 93-122.
17. K. P. Mikkilineni, Y. -C. Chow and Y. W. Su, " Petri Net-Based Modeling and Evaluation of Pipelined Processing of Concurrent Database Queries," IEEE Transaction on Software Engineering, Vol. 14, No. Nov. 1988, pp. 1656-1667.
18. M. K. Molloy, "Performance Analysis Using Stochastic Petri Nets," IEEE Transaction on Computer, Vol. C-39, No. 9, Sept. 1982, pp. 913-917.
19. T. Murata, "Petri Nets: Properties, Analysis and Applications," Proceedings of the IEEE, Vol. 77, No. 4, Aril 1989, pp. 541-580.
20. C. A. Petri, "Communication with Automate", New York: Griffiss Air Force Base, Tech. Rep. RADC-TR-65-377, Vol. 1, and September 1996.
21. R. R. Razouk, " The Use of Petri Nets for Modeling Pipelined Processors," On Proceeding of the 5th ACM/IEEE Design Automation Conference, 1988, pp. 548-553.
22. Semenov, A. M. Kolemans, L. Lloyd, and A. Yakovlev, " Designing an Asynchronous Processor Using Petri Nets," IEEE Micro, March/April 1997, pp. 54-64.
23. SES/Workbench, "SES/Workbench User's Menu," Scientific and Engineering Software, Inc., 1996.
24. Zimmermann, J. Freiheit, and G. Hommel, " Discrete Time Stochastic Petri Nets for Modeling and Evaluation of Real-Time Systems," On Proceedings 15th International Parallel and Distributed Processing Symposium. 2001, pp.1069 -1074.