JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 8, NO. 12, DECEMBER 2005(pp. 1605-1612)

Simulation-based Design Verification for High-performance Computing System

Taikyeong T. Jeong

ABSTRACT

This paper presents the knowledge and experience we obtained by employing multiprocessor systems as a computer simulation design verification to study high-performance computing system. This paper also describes a case study of symmetric multiprocessors (SMP) kernel on a 32 CPUs CC-NUMA architecture using an actual architecture. A small group of CPUs of CC-NUMA, high-performance computer system, is clustered into a processing node or cluster. By simulating the system design verification tools, we discussed SMP OS kernel on a CC-NUMA multiprocessor architecture performance which is 32% of the total execution time and remote memory access latency is occupied 43% of the OS time. In this paper, we demonstrated our simulation results for multiprocessor, high-performance computing system performance, using simulation-based design verification.

Keywords: High-performance computing, Multiprocessor, Simulation, Design verification

1. INTRODUCTION

In principle, methodologies for computer system performance studying fall into two categories: analytical modeling and quantitative techniques[1]. Analytical modeling, although fast and flexible, frequently makes some assumptions and simplifications when employed on real but complex systems. Sometimes these simplifications and approximations may nebula the realistic system performance picture and occasionally lead incorrect results. On the other hand, quantitative techniques provide alternatives in performing system performance studies in a more accurate and detailed style. One quantitative method is hardware prototyping: a technique that uses dedicated hardware to generate non-invasive data at high speeds[2].

Additionally, the special purpose design verification tools and system cannot be easily ported to those systems with different architecture. Simulation is another quantitative approach frequently exploited by today's computer architecture researcher to investigate ideas and potential designs[2]. Compared with its hardware counterpart, software simulation is slower, but more flexible. A lot of simulation tools have been designed to support various simulations ranging from simple trace-driven simulation based on a direct mapped cache to the cycle-by-cycle simulation of the pipeline execution of one superscalar microprocessor[3].

Unfortunately most of them focus on and can only deal with one functional unit simulation of a computer system, e.g., cache, pipeline, main memory unit, hard disk and interconnection network. Furthermore, these simulators are frequently driven by some kind of traces instead of the realistic workloads. As a result, these tools cannot be used to study complete computer system behaviors with complex workloads, such as OS-intensive commercial software tool, operating systems, and compilers. We demonstrated that SMP kernel on a 32 processors CC-NUMA architecture which is an example of

※ Corresponding Author : Taikyeong T. Jeong, Address : Texas at Austin, Austin, TX 78712 USA. And, now he is with the University of Delaware TEL : +1-512-786-6402, E-mail : ttjeong@alumni.utexas.net
Receipt date : Oct. 14, 2005, Approval date : Nov. 29, 2005

high-performance computer system. To show a simulation-based design verification method, we show an architectural overview and extend our simulation results through simulated machines.

The paper is organized as follows. Section II reviews preliminaries on architectural platform. A design methodology for simulation environment is in Section III. Section IV shows a detail result with a simulation-based design verification method. Section V summarizes this work.

2. ARCHITECTURES

In this section, we present simulation-based design verification for multi-core architecture as a case study.

2.1 SMP and CC-NUMA Architecture

We involve two kinds of shared memory multiprocessors architectures, namely symmetric multiprocessors (SMP) and CC-NUMA (Cache Coherence Non-Uniform Memory Access) multiprocessors.

In a SMP system, each processor (with its private cache) is connected with shared memory by system interconnection (usually bus or hierarchy multi-bus). Each CPU is symmetric to the shared memory from the perspective of memory accesses. Whenever a cache miss occurs, the corresponding memory request will be handled across the shared bus. Snoopy-based protocols are frequently used in SMP to maintain coherence between CPU private caches[4]. With the increase of CPU and cache pair numbers, the shared bus will be the bottleneck.

In general, the SMP system is not scalable and can support only a small number of CPUs. Fig. 1 shows an overview of SMP architecture.

In CC-NUMA architecture, a small group of CPUs is clustered into a processing node or cluster[5]. The global shared memory is physically node or cluster. Also, the global shared memory is physically distributed in each processing node. In fact, each processing node can be regarded as a SMP with a small CPU number. When a cache miss occurs, it can be satisfied in local shared memory (portion of shared memory within the same processing node) or in remote shared memory (portion of shared memory distributed in other processing nodes). There are two kinds of memory access modes. local memory access and remote memory access. In this case, remote memory access is more expensive since it will generate more traffic in both local bus and system interconnection.

In addition, potential processing node communications, such as cache coherence maintenance operations, may also be caused by remote access. Some dedicated hardware is used to handle remote memory access. A directory- based cache coherence protocol is frequently used in CC-NUMA architecture to maintain coherency caches among processing nodes. Fig. 2 shows the block diagram of CC-NUMA multiprocessors.

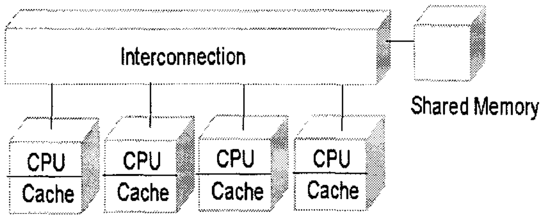

Fig. 1. An Overview of SMP Architecture.

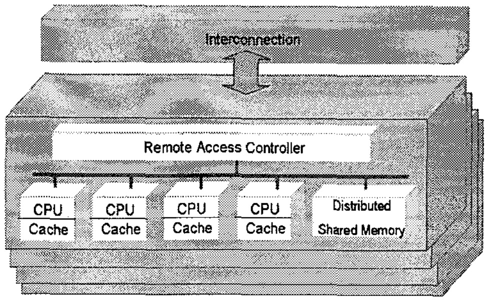

Fig. 2. A Block Diagram of CC-NUMA Architecture.

2.2 SMP OS Kernel Architectures

A SimOS-ported SMP OS kernel was used to perform a case study of architectural behavior under multi-programmed workloads. In this case, IRIX 5.3 kernel is executed for SMP OS kernel which is originally from SGI's implementation of NIX SVR4. A user process can be dynamically trapped into the kernel via system call. The IRIX 5.3 kernel supports multiple thread control. In the kernel, all control flows are treated and scheduled as threads[6].

In the user space, one or several user threads created by a single UNIX process is scheduled and executed as a lightweight process (LWP). In the OS kernel, a corresponding kernel thread is generated for each user LWP. This kernel thread takes the responsibility to execute kernel code when a system call in the corresponding LWP occurs. Some kernel threads are created for special purpose, e.g., handling interrupts and providing network file system (NFS) services.

Consequently, these kernel threads have no LWP associated with them. Therefore, kernel threads are selected from a ready queue and dispatched in priority order on the pool of available processors by the scheduler.

In an IRIX 5.3 ported SMP OS kernel, we assumed that the scheduler assumes all processors are equivalent and all threads are executed as a logic processor and/or LWP. Fig. 3 shows the architectural view of SMP OS kernel.

In the multiprocessor architecture, memory operations issued by one processor may be delayed or reordered when observed by other processors within the same system. In this case, synchronization primitives should be used to protect memory access to shared data. Like their counterparts, kernel threads synchronize via a variety of synchronization primitives, such as mutual exclusion locks, condition variables, and/or counting semaphores. SGI has already tuned IRIX 5.3 to run effectively on machines

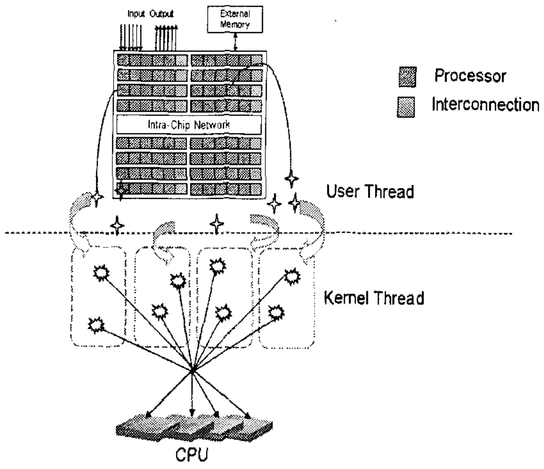

Fig. 3. SMP OS Kernel Architectures.

with as many as 36 processors (SGI Challenge machines)[7].

Moreover, in the SMP OS kernel, the virtual memory management routines divide an address space into a set of segments. Each segment presents a chunk of virtual memory. Page faults associate segments with the physical page in the following way: If the requested page is backed by *vnode* (data structure used to memory-map the file in the OS), the *vnode* operation will hit page cache and feed it to the segment. Otherwise, *vnode* allocates an empty entry in the page cache and starts I/O operations to load the appropriate page and call OS routine to set up a memory *module unit translation*, such as TLB replacement, TLB refill and etc. Since the SMP treats the shared memory space symmetrically, the kernel does not ensure that processes scheduled on a particular processor are placed in a particular location of shared memory that is faster to access from that processor.

3. SIMULATION ENVIRONMENT

Considering a set of different architectures, we present a simulation environment of simulation-based design verification methods for multi-core processors

3.1 Architecture of Simulated Machine

SimOS is configured as a 32-processor CC-NUMA machine. Our simulated machine can be regarded as a prototype of DASH (4CPU per memory). Table 1 summarizes the hardware configuration for our simulation.

The NUMA memory model of SimOS is customized for simulating the memory system of a CC-NUMA multiprocessor with invalid-based directory scheme. The number of memory/directory controllers can be configured to model different memory configurations ranging from SMP to CC-NUMA. Having only one memory/directory controller would make it similar to a bus-based machine. A number of memory/directory controllers which is smaller than the number of CPUs would result in a memory system similar to the DASH. Having the number of memory/directory controllers equal to the number of CPUs would result in architecture similar to the FLASH[9] or MIT Alewife[8] machine.

Table 2 shows the major parameters of CC-NUMA simulated machine model used in the simulation. A L2 cache miss which can be satisfied by local memory has a 225*ns* delay. In this case, the remote access has two situations: if the remote access requires a memory block that is clean in the home node's memory, the latency is 725*ns*. Otherwise, an additional access is required to retrieve the exclusive copy from the third processor's cache which has the dirty copy of that cache line, and will have 975*ns* delay. Therefore, the CC-NUMA multiprocessor which contains 32-processor can be simulated by this design verification tools and verification methods.

Table 3 also compares the local and remote access time of several commercial systems with the CC-NUMA multiprocessor model. In this case, the latency ratio of local and remote access of our CC-NUMA model is similar to most commercial CC-NUMA systems. Because the comparison results of local memory and remote

Table 1. Hardware Configuration for Simulation.

Hardware Type	Performance Characteristics
32 Processors	MIPS R4000 at 200 MHz
32 1-level instruction caches	32KB, 2-way, 64byte block size
32 1-level data caches	32KB, 2-way, 64byte block size
32 2-level unified caches	1MB, 2-way, 128byte block size
256 MB Memory	32MB for each cluster
8 memory/Directory Controllers	Accurate NUMA model(225 ns min latency, 725 ns of remote node)
1 Disk	Accurate model of HP97560 SCSI-I disk

Table 2. Parameters of NUMA model.

Type of Access	Latency
L2 cache hit	50 (ns)
Bus Operation from processor to local directory controller	75 (ns)
Occupancy of directory controller on outgoing remote miss	100 (ns)
Occupancy of local directory controller on remote miss	25 (ns)
Occupancy of remote-directory on remote miss	350 (ns)
Occupancy of local directory controller on incoming remotes	25 (ns)
Latency for the directory controller to fetch a cache line from local memory	50 (ns)
Fixed latency for going between directory controller (across the network)	150 (ns)
Dirty penalty	250 (ns)

memory are indicated quite similar range, we could utilize these data as a simulation-based design verification tools.

3.2 Software Environment

We use a release of IRIX 5.3 multiprocessor that has been ported to the SimOS simulation environment to perform this case study. IRIX 5.3 kernel is developed and well tuned for SGI Power 4D/340 and Challenge machines, two kinds of bus-based multiprocessors. Due to the limitation of this SMP OS system, two assumptions have been made: Firstly, all I/O devices are connected at *cluster 0*. In our simulated machines, a SCSI-I (HP97560) disk services all of the 32 processors. Secondly, unlike some NUMA-like operating system that distributes *n* data and code in a round robin way across clusters, IRIX 5.3 allocates contiguous physical pages for its code and data from low memory address. Therefore, most OS code and data structures are allocated on *cluster 0*.

4. SIMULATION RESUTLS

In this section, we discuss an implementation of a multiprocessor system as it relates to the OS and kernel.

4.1 Design Verification for Multiprocessors

Cycle-by-cycle functional simulation can provide accurate test results. However, the speed of this kind of simulation is very slow, especially for large and complex multiprocessor systems with real workloads. Thus, how to make trade offs between simulation speed and accuracy becomes an important issue of this study. Our simulation technology, which employs speed-accuracy trade-off mechanisms, is described here. The first step of our simulation is to configure the simulated machine and customize statistics collection and reporting mechanisms. TCL scripts can be used to customize simulated machine architecture and to map low-level machine behavior to higher-level abstraction. SimOS emulation mode is used to boot the OS and mount workload disks from the host machine file system to the simulated machine by taking the advantage of the high-speed of this mode. Rough characterization mode is then used to perform cache '*warm up*' and to past all uninterested initialization part of the multi-programmed workloads. After the steady state of a workload has arrived, detailed characterization mode is used to perform cycle-by-cycle functional simulation and to collect execution statistics.

Therefore, our hierarchy simulation-based design verification technique enables us to get accurate simulation results while studying large complex multiprocessor systems with real workloads within reasonable time.

4.2 OS Kernel Execution Time Breakdown

Figure 4 shows the average execution time breakdown for RADIX sort algorithm on 32 CPU CC-NUMA machines. The data shown here is derived from three samples from different execution periods of the workload on IRIX 5.3

Table 3. Comparison of Local Remote Access Time on Different Machines.

System	Local Memory	Remote Memory (Non-Scaled)	Remote Memory (Scaled)
Hal S-1	240	710	1365
SGI Origin	200	710	805
HP-Examplar	450	1315	1955
Simulated machine	225	725	975

ported SMP kernels (See Fig 4). The total execution time can be divided into idle time, user execution time, and kernel execution time.

In particular, user execution time is defined as the period in which CPU is occupied by the instructions from user processes. Similarly, kernel execution time refers as the time spent on executing kernel instructions.

The idle time can be regarded as the transitive state between kernel and user. During idle time, OS is still active but simply executes a waiting loop for the ready processes or threads, either from user space or kernel space. The idle time can be triggered by synchronization of either user or kernel threads or I/O activities. During the idle time, OS performance is generally un important to overall system performance. For this reason, non-idle time rather than overall machine time is used here to generate the metric for this simulation.

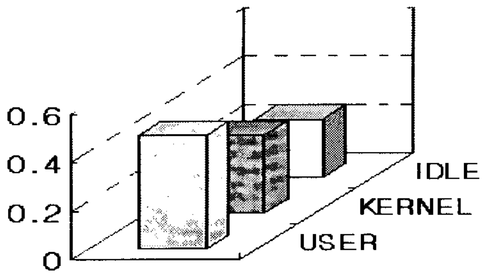

Fig. 4. Execution Time Breakdown.

4.3 Kernel Routines Execution Time

We find that the distribution of kernel service time is highly concentrated in this simulated machine: i.e., 80% kernel service time go to approximately 20~30 routines.

Table 4 summaries the functionality of these routines. Generally speaking, these frequently called routines fall into four categories: namely, virtual memory handlers, scheduling handler, frequently invoked services, e.g., interrupts and clock dispatching, and the file system interface

Table 4. Kernel Routines with the Most Execution Time.

Kernel Routine	Functionality
Fork	System call
Demand-Zero	VM fault handlers
Cpuintr	Dispach Interrupt
Pfault	Page fault handler
Runqproc	Scheduler
Clock	Clock Interrupt Handler
UTlb	TLB refill handler
BSD	File System
Vfault	Vnode fault handler

routines. High-performance computing system included multi-core processor and applications must often manage time-critical responses. Therefore, kernel routine execution time is very important issue in order to validate design verification both architectural approaches and simulation-based validation.

Figure 5 show the execution time breakdown by OS kernel services. The most frequent kernel routine *utlb, vfault* and *pfault* are used to perform virtual memory management and map virtual memory to physical memory (See Fig 5). These kernel routines will be called whenever a fault of either *vnode* or page table occurs.

Another group of routines that have high execution time is the kernel services with high occurrence frequency. These frequently invoked services, e.g., clock handler or interrupt dis-

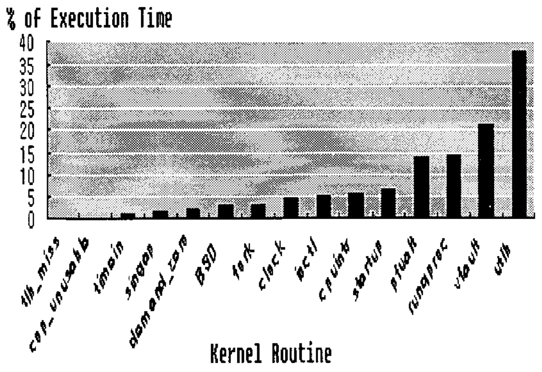

Fig. 5. Kernel Execution Time Breakdown based on Kernel Services.

patching routines, usually have small average execution time. However, their high frequency makes them also play important roles in kernel execution time.

Our simulation result indicates that a potential improvement of OS kernel can be made by enhancing the efficiency of those routines with the longest run time. On the other hand, the high concentrated characteristics of OS routines implies that a small amount of kernel code replication may help to reduce the kernel execution time by reducing remote misses at a much lower memory cost than full replication.

5. CONCLUSIONS

SimOS provides the ability to investigate both realistic workloads and operating systems by simulating the system design verification tools and system simulation. Meanwhile, trade-offs between speed and accuracy can be made while simulating large workloads on complex system. This paper has also presented a case study of IRIX 5.3 ported SMP kernel on a 32 CPUs CC-NUMA architecture.

Moreover, OS kernel execution time is highly concentrated and dominated by a small group of kernel routines, most of which are virtual memory management routines and frequency-invoked events handler.

Our simulation results show that approximately 32% of the total execution time involves the OS kernel. Also, memory system stalls and remote memory access latency is occupied almost 43% of the OS time. We have discussed that the IRIX 5.3 ported SMP kernel should be improved to show a good performance on CC-NUMA architecture by using.

6. REFERENCES

[1] M. Rosenblum, S.A. Herrod, E. Withchel, and A. Gupta, "Complete Computer System Simulation: the SimOS. Approach," *IEEE Parallel and Distributed Technology: System and Application*, vol 3, no., 4 pp. 34-43 winter 1995.

[2] S. A. Herrod, "Using Complete Machine Simulation to Understand Computer System Behavior," Ph.D. Thesis, *Stanford University*, Feb. 1998.

[3] N. Bowman, and et. al., "Evaluation of Existing Architecture in IRAM Systems," *International Symposium on Computer Architecture*, June 1997.

[4] M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod, "Using the SimOS Machine Simulator to Study Complex Computer Systems," *ACM TOMACS Special Issue on Computer Simulation*, 1997.

[5] E. Witchel and M. Rosenblum, "Embra: Fast and Flexible Machine Simulation," *Proc. of ACM SIGMETRICS '96: Conference on Measurement and Modeling of Computer Systems*, Philadelphia, PA, 1996.

[6] G.E. Blelloch and et. al., "A Comparison of Sorting Algorithms for the Connection Machine CM-2," *Proceedings of the Symposium on Parallel Algorithms and Architectures*, pp. 3-16, July 1991.

[7] A. Agawal, R. Bianchini, D. Chaikem, K.L. Johnson, D. Krnz, J. Kubiatowics, B. Lim, K. Machenzie, and D. Yeung, "The MIT Alewife Machine: Architecture and Performance," *Proc. of the 22nd International Symposium on Computer Architecture*," pp. 2-13, May 1995.

[8] J. Kuskin, and et.al., "The Stanford FLASH Multiprocessor," *Proceeding of the 21st International Symposium on Computer Architecture*, pp. 302-313, April 1994.

[9] S.C. Woo, and et.al., "The SPLASH-2 Program: Characterization and Methodological Consideration," *Annual International Symposium on Computer Architecture*, pp. 24-36 June 1995.

ACKNOWLEDGEMENT

The author would like to thanks the referees for their valuable comments.

Taikyeong Jeong

He received the Ph.D. degree from the Department of Electrical and Computer Engineering, the University of Texas at Austin in 2004. He performed research in the area of high performance circuit and power efficiency system design. He joined the University of Delaware, where he is now a research associate under the research grants of NASA (NNG05GJ38G), working on low power digital circuits for next generation space robotics devices and high performance computing. His research interests include VLSI design, computer architecture, logic design, and high performance computing. He is a member of IEE, IEICE and IEEE.