Multi-Program 벤치마크를 이용한 대칭구조 Multiprocessor의 성능평가와 분석

정 태 경*

Performance Evaluation and Analysis of Symmetric Multiprocessor using Multi-Program Benchmarks

Taikyeong Jeong*

요 약

본 논문은 컴퓨터 시스템의 성능평가와 분석을 대칭구조의 멀티프로세서를 실행할 수 있는 시뮬레이터를 사용하여 살펴보았으며 또한 시스템 분석을 하는데 있어서 멀티프로세서를 위한 멀티프로그램 벤치마크의 집합체인 SPLASH-2를 이행하여 대칭구조의 운영체제 IRIX 5.3 탑재한 멀티프로세서의 행위범위의 연구를 수행하기위하여 멀티프로세서의 시스템 분석을 실시하였다.

또한 대칭구조의 멀티프로세서의 구조와 평가방법을 보다 유효하게하기 위해서 멀티프로세서의 확장성을 functionality-based 소프트웨어인 SimOS를 가지고 증명하였으며 본 논문을 통하여 멀티프로그램 벤치마크인 RADIX 정렬 알고리즘이나 Cholesky 인수분해 알고리즘을 이용하여 로칼 인스트럭션과 로칼 데이터 사이에서의 멀티프로세서의 Cache miss의 수 와 Stall 시간을 동시에 검사하였다.

ABSTRACT

This paper discusses computer system performance evaluation and analysis by employing a simulator which able to execute a symmetric multiprocessor in machine simulation environment. We also perform a multiprocessor system analysis using SPLASH-2, which is a suite of multi-program benchmarks for multiprocessors, to perform the behavior study of the symmetric multiprocessor OS kernel, IRIX 5.3.

To validate the scalability of symmetric multiprocessor system, we demonstrate structure and evaluation methods for symmetric multiprocessor as well as a functionality-based software simulator, SimOS. In this paper, we examine cache miss count and stall time on the symmetric multiprocessor between the local instruction and local data, using the multi-program benchmarks such as RADIX sorting algorithm and Cholesky factorization.

키워드

Multiprocessors, Performance Evaluation, Machine simulator, Multi-program benchmarks

I. INTRODUCTION

The major goal of performance evaluation is to collect and disseminate information relative to performance aspects. Many researchers now consider some benchmarks and workloads for performance measurements that can be used to

* Department of Electrical and Computer Engineering, the University of Texas at Austin, Austin, TX 78712.

접수일자 : 2005. 11. 28

compare compute-intensive workloads on different type of high-performance computer systems including uni-processors and multiprocessors.

In this case, we will focus on the quantitative and analytical characterization of processors and applications for general-purpose and scientific computer system. Several papers from recent performance evaluation, computer architecture, and workload characterization-related conferences have studied and verified these analysis methods and algorithms [1]

The pervasive nature of performance evaluation for computer architecture extends it to virtually all sciences. The use of complex algorithms to compare different families of computers is correct since computer efficiency depends not only upon the concrete computer architecture but also upon the operating system version which controls the user's tasks.

Therefore, this paper will examine that symmetric multiprocessor structure and evaluation methods for design effectiveness using multi-program benchmarks. We also observe a cache miss count and a stall time to validate a symmetric multiprocessor performance with multi-program workloads.

Ⅱ. SIMULATION ENVIRONMENTS

In this section, first we derive our simulation environment with a software simulator. Subsequently, we discuss way to adapt it to more realistic organizations.

2.1 Experimental Platform

A workload for computer architecture evaluation is important in the area of programming and system design to estimate the efficiency of computer processor operation. SimOS is a functionality-based software simulator for the FLASH (Flexible Architecture for SHared memory) project [2]. In this case, the FLASH multiprocessor will be a scalable multiprocessor able to support a variety of communication models, including shared memory and message passing protocols, through the use of a programmable node controller. This processor structure has a symmetric structure since memory is shared, and it provides the architectural support necessary to use FLASH as a traditional "standalone" supercomputer, a computer server, a robust multi-user system, or a distributed system.

Therefore, SimOS has been widely used in real system designs, such as the Stanford FLASH machine [6], Hive operating system, and SUIF parallel compiler as well as research projects such as Tornado project at University of Toronto [5] and IRAM project at University of California at Berkeley.

SimOS contains the whole machine simulation environment required for studying both hardware and software of modern complex high-performance computer systems. It usually provides the ability to simulate the whole computer system instead of just a couple of functional units. Various hardware models, ranging from CPUs to consoles have been validated and enclosed in SimOS simulation environments. Additionally, SimOS describes the hardware system in such detail that a commercial OS can be booted directly on it with little or no modification endeavor.

In the meantime, SimOS employs a couple of mechanisms for trading off between simulation speed and accuracy [2]. This characteristic makes our simulation results of complex multiprocessor systems become both more efficient and practical. Moreover, SimOS contains a sophisticated but flexible translation scheme which re-maps the huge amount, but less informative, low-level hardware activities, i.e., cache line fill bus transactions, translation look-aside buffer (TLB) misses, cache-to-cache transfers, and inter-processor interrupts back to understandable semantics such as remote misses.

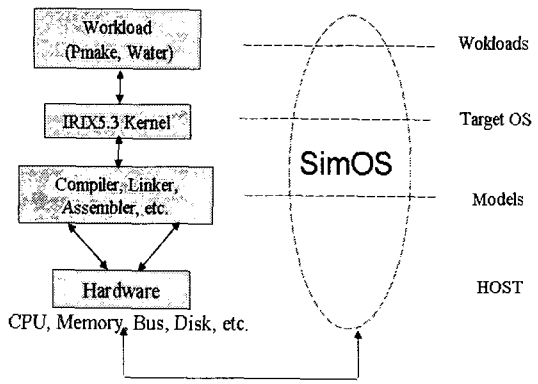

Fig 1. An overview of SimOS simulation environment.
그림 1. SimOS 시뮬레이션 환경의 개괄도

Figure 1 shows an overview of SimOS simulation environment. Currently, SimOS can run on several platforms - Host - including MIPS R400 with IRIX 5.3 (also included IRIX 6.4), Alpha with digital UNIX platform, as well as X86 on Linux machine.

Furthermore, SimOS supports the booting of the OS and execution of real workloads by providing detailed yet comprehensive hardware models, e.g., CPU, memory, bus, disk, ethernet, and console of completed computer systems. We also observed that a pre-commercial version of symmetric multiprocessor OS kernel IRIX 5.3 kernel has been modified and ported to SimOS.

Many realistic and complex workloads, such as the GNU C++ compiler, Sybase, and SPEC 95 benchmark suites can run on the top of the target OS directly or with little modification. Therefore, SimOS provides the potential ability to investigate the performance of a wide range of real world workloads on high-performance computer system architecture.

2.2 Multi-programmed Workload

We have more information about the characteristics of the multi-programmed benchmarks, which we will discuss it in detail. In order to provide access to this information so that researchers do not need to perform costly simulations to get the same information that we already have, we reported a set of experimental results along with a validation methodology and an interactive drawing that allows symmetric multiprocessor architecture to be viewed in a variety of ways.

A multi-programmed benchmark, Splash, was initially released by Stanford University to perform multiprocessor performance studies in 1992 [11]. The first version of the Splash suite was written for the bus-based multiprocessors with uniformly accessible shared memory, but it was not implemented for optimal interaction with modern computer system characteristics such a long cache lines, high latencies, and physically distributed memory.

The Splash-2 [6] was explored and employed as the workload to perform a case study of the multiprocessor OS kernel. We have examined Splash-2 as a multi-programmed benchmark suite for performance evaluation and analysis.

Therefore, we observed a simulation time, although cycle-by-cycle functional simulation does provide detailed and accurate results, is quite unexpected. It is almost impossible to use only one simulation mode to support the whole simulation, i.e., booting the operating system, mounting workload disk, performing the cache warm up, past initialization, while collecting statistics.

The other problem that we might have is tool common language (TCL) scripts can be used to control and monitor the whole picture of hardware activities, (e.g., cache miss, pipeline stalls and exception) that occur during the simulation execution, they also contribute to the simulation delay. As a result, only source those TCL scripts that are really required for the performance study.

Ⅲ. METHODOLOGY

In this section, we present a performance evaluation methods and analysis of symmetric multiprocessor using multi-programmed benchmarks and some algorithms such as the RADIX sort algorithm and Cholesky factorization.

3.1 RADIX Sort Algorithm

A sorting algorithm is an algorithm that puts elements of a list in a certain order. Efficient sorting is important to optimizing the use of other algorithms, such as *search* and *merge* algorithms, which require sorted lists to work correctly; it is also often useful for canonical order data and for producing readable output. Therefore, the RADIX sorting algorithm is used in many dataset systems and aerospace applications. This radix program [4] requires bulk data to be moved among processors. At the beginning, take the least significant digit (or group of bits, both being examples of radices) of each key, and the algorithm can sort the list of elements based on that digit, but keep the order of elements with the same digit (this is the definition of a stable sort). Finally, repeat the sort with each more significant digit.

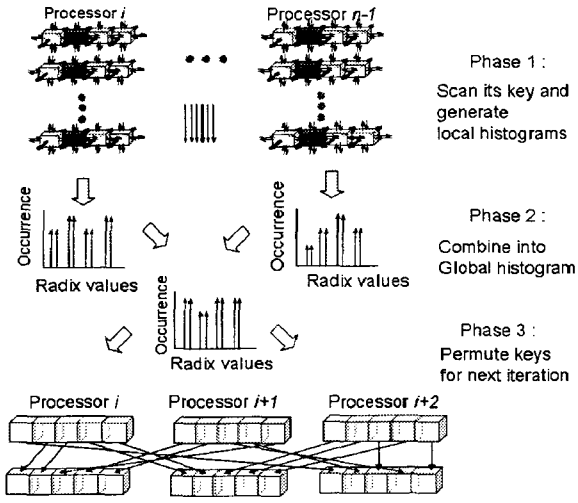

Fig 2. The RADIX sort algorithm
그림 2. RADIX 정렬알고리즘

The integer RADIX sort algorithm, which is shown in Figure 2, is based on the method described by Belloch [9]. The whole RADIX algorithm is composed of several iterations. Each iteration can further be divided into phases. During the first phase, each processor generates its own histogram of the key. And then, the local histograms are combined into global histogram which represents the distribution of RADIX digits of all the keys. After that, each processor (from processor *o* to processor *n-1*) uses the global histogram to permutate their partition of keys. The keys are written into the destination array through processor writes.

An important factor that may affect the performance of RADIX is the choice of radix *r* for sorting. The algorithm performs iteration for each radix *r* digit in the keys. Consequently, less iteration is required when a large radix value is used. However, more storage is required for the local and global histogram and performance will also be reduced when radix *r* increases. The choice of radix *r*, the number of processors, the number of keys to be sorted, and the number of keys that fit in the cache can all affect the sorting performance.

3.2 Cholesky Factorization

Another multi-programmed workload is the Cholesky factorization algorithm and it performs the factorization of a sparse positive definite matrix.

In parallel architecture and distributed computing, Cholesky factorization is often the most complicated step in numerically solving a positive definite linear system of equations. For a system with *n* equations and n unknowns, the computational complexity is $O(n^3)$ floating point operations. In many large-scale optimization algorithms, such as sequential quadratic programming, the system of equations is augmented with additional data and the Cholesky factorization must be re-evaluated. To validate the multi-programmed workload, we describe an algorithm for showing the Cholesky factorization of an *n* x *n* positive definite matrix when it has been augmented with *m* additional rows and columns. In this case, the computational complexity is $O(m^3) + O(m^{2n}) + O(mn^2)$.

We investigated a use of this complicated algorithm for a performance evaluation of symmetric multiprocessor. When given a positive definite matrix *A*, this algorithm finds a lower triangular matrix *L*, such that $A = LL^T$. This algorithm is frequently used in structural analysis, device and process simulation, and electric power network problems. The Choslesky factorization algorithm can be divided into a preprocessing phase and a factorization phase.

Figure 3 shows the pseudo-code of sequential algorithm for the Cholesky algorithm. In the preprocessing phase, the matrix is decomposed into blocks that have non-zero elements and blocks that have all-zero elements. Blocks are created by reordering matrix columns so that columns with similar non-zero structures can be put together adjacent to each other.

```
1 :  for k=1 to n do
2 :     Lkk = factor (Lkk)
3 :     for i = k+1 to n with Lik ≠ 0 do
4 :        Lik = Lik L^-1kk
5 :     for j = k+1 to n with Ljk ≠ 0 do
6 :        for i =j to n with Lik ≠ 0 do
7 :           Lij = Lij - Lik L^T jk
```

Fig 3. Pseudo-code of sequential algorithm
그림 3. 순차 알고리즘의 유사코드 구현

In a Cholesky program, it is important to allocate blocks of similar computation to each processor. Otherwise, the computational load will be unbalanced and some processors will have too much idle time.

After one processor finishes updating its block, the block becomes ready to be passed to other processors so that other blocks that depend on it can be updated also. The size of the matrix will affect the performance of the Cholesky factorization. If the matrix size is not big enough, many processors will be idle and the performance will be lower than expected.

Ⅳ. SYSTEM AND MEMORY PERFORMANCE ON OS KERNEL

We have discussed that system performance evaluation data based on multi-programmed benchmarks, e.g., RADIX sorting and Cholesky algorithms which make system reliability better since it has been evaluated already. It is an attempt to minimize cache miss count and stall time to local instruction to measure system performance.

Using multi-programmed workload, we have observed cache miss count and stall time on symmetric multiprocessor between the local instruction and local data. Figure 4 shows the execution breakdown based on non-idle time.

As depicted in Figure 4, a large fraction of non-idle time is occupied by memory system stall, 18 % of non-idle time is spent on OS memory system stall, compared to just 4 % spent on OS synchronization. In other words, 43 % of OS kernel time is spent stalled for cache misses. This fact indicates that a potential way to improve symmetric multiprocessor OS kernel (with IRIX 5.3 kernel), is to reduce memory system stall time. This result was tested and verified on symmetric multiprocessor, using CC-NUMA machine as our performance evaluation tool [7].

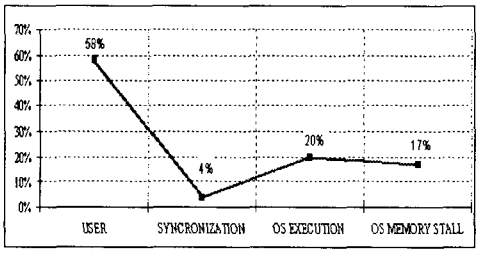

Fig 4. Non-idle execution breakdown on symmetric multiprocessor OS kernel
그림 4. 대칭구조 Multiprocessor의 비휴지기 실행구조분석

In this case, only 4 % of non-idle time is spent on symmetric multiprocessor OS kernel synchronization (i.e., waiting for spin locks). Synchronization is obviously not a performance bottleneck for RADIX sort algorithm because of the following reasons. First of all, SGI have already tuned the symmetric multiprocessor IRIX 5.3 OS kernel to run efficiently on machines with as many as 36 processors by providing a fine-grain synchronization kernel. The overhead of IRIX 5.3 kernel synchronization is relatively low even for a large multiprocessor system. Second of all, our multi-programmed benchmarks, RADIX sort algorithm uses limited I/O services for its algorithm.

Ⅴ. EXPERIMENTAL RESULTS

In this paper, we have explored two benchmarks, RADIX sort and Cholesky factorization from Splash-2 and use RADIX sort algorithm to perform this experimental result.

Most CPU's have first-level instruction and data caches on chip and many have second-level cache(s) that are bigger but somewhat slower. Memory accesses are much faster if the data is already loaded into the first-level cache. When some program accesses data that isn't in one of the caches, it gets a cache miss [8].

As discussed above, the OS spends approximately half of it time for cache misses. In a symmetric multiprocessor case, CC-NUMA multiprocessor machine, a cache miss can either lead to a local or remote memory access. Table 1 shows the contribution of these two types' memory accesses from the

Table 1. Memory performance of OS Kernel based on different memory access modes
표 1. 메모리 어세스형태에 따른OS Kernel의 메모리 성능비교

Type of Reference	% of Cache Miss Count	% of Cache Stall Time
Remote Instruction	0.11	0.13
Remote Data	0.65	0.81
Local Instruction	0.15	0.04
Local Data	0.09	0.02

perspective of both percentage of cache miss count and percentage of cache stall time.

Table 1 also indicates both cache miss count and stall time to the local instruction and local data are relatively low. This occurs because we configured a large (1Mbyte) L2 level cache for the simulated machine. The fact shows that the false sharing of RADIX sort algorithm is reasonable as we expected. Therefore, we derive that RADIX sort algorithm can be scaled to larger multiprocessor system with CPU numbers in excess of 36 processors.

Our simulation results demonstrate remote memory accesses to both data and instructions dominate the cache misses of the OS, accounting for 76 % by count and 94 % by time. In this case, the symmetric multiprocessor IRIC 5.3 OS kernel is not related to CC-NUMAmachine largely or almost entirely in memory allocation and processor scheduling. Since the multiprocessor OS code and much of the OS data resides on cluster *0*, this memory layout will lead to 7 out of 8 cache misses being remote misses. The symmetric multiprocessor OS with IRIX 5.3 cannot detect and maintain the affinity between processors and the local shared memory distribute within the same cluster. Since events and system calls that access various multiprocessor OS kernel data may occur frequently on any CPU, the performance of this symmetric multiprocessor OS IRIX 5.3 kernel may be even worse than the performance of it on a small scale symmetric multiprocessor [10].

Furthermore, remote data cache misses incur more latency than remote instruction cache misses because it can be writeable with shared variables. SimOS is a complete machine simulation environment designed for the efficient and accurate study of both uni-processor and multiprocessor computer systems. Symmetric multiprocessor with computer software programmed simulator in enough detail to boot and run specialized multiprocessor operating systems using multi-programmed workloads.

VI. CONCLUSIONS

Benchmarking processor and computer system architectures has become extremely difficult due to the complexity of the processors and the complexity of the applications that run on the computers.

A considerable amount of workloads for computer architecture evaluation purposes have been done in the field of the programming systems for estimating the efficiency of computers operation on the basis of SimOS simulation environments and symmetric multiprocessor architecture. The given programming systems and analytical algorithms such as the RADIX sort algorithm, and the Cholesky factorization algorithm allow effective demonstration the statistical information collect by standard measuring systems, cache miss count and stall time, and calculation of the integral algorithm of computer efficiency (e.g., capacity, response time, load characteristics). One of the main problems in forecasting the efficiency algorithm of concrete symmetric multiprocessor architecture is the determination of the workload parameters.

As applied to the SimOS simulation environments and multiprocessor architecture, both general system measuring monitors (cache miss count and stall time) and specialized monitors realized through programs and hardware are used to collect statistics concerning the parameters of task classes.

참고문헌

[1] P. Bose and T. M. Conte, "Performance Analysis and Its Impact on Design," *IEEE Computer Society*, pp. 41-49, May 1998.

[2] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy, "The Stanford FLASH Multiprocessor," *Proc. of the 21st Int. Sym. on Computer Architecture*, pp. 302-313, Chicago IL, April 1994.

[3] 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.

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

[5] Tornado Operating System Project, Univ. of Toronto, http://www.eecg.toronto.edu/paralle/tornado-on-simos.html

[6] S.C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Program: Characterization and Methodological Consideration," *Annual Int. Sym. on Computer Architecture*, pp. 24 -36 June 1995

[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 Int. Sym. Computer Architecture*, pp. 2-13, May 1995.

[8] L. K. John. "More on finding a single number to indicate overall performance of a benchmark suite," *ACM SIGARCH Computer Architecture News*, Vol. 32, Issue 1, pp. 3-8, March 2004

[9] J. R. Mashey. "War of the benchmark means: time for a truce," *ACM SIGARCH Computer Architecture News*, Vol. 32, Issue 4, pp. 1-14, September 2004

[10] B. Black and J. P. Shen, "Calibration of Microprocessor Performance Models," *IEEE Computer Society*, pp. 59-65, May 1998.

[11] R. P. Weicker, "An Overview of Common Benchmarks," *IEEE Computer Society*, pp. 65-75, December 1990.

[12] L. John, P. Vasudevan and J. Sabarinathan, "Workload Characterization: Motivation, Goals and methodology," pp. 3-12 (also published in "Workload Characterization: Methodology and Case Studies," *IEEE Computer Society*, 1999).

저자소개

Taikyeong Jeong 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 design and power efficiency system design. He joined the University of Delaware, where he is now a research associate under the research grants of NASA (Grant No. NNG05GJ38G) in 2004, working on VLSI design for next generation space robotics devices and high performance circuit and system. His research interests include VLSI design, computer architecture, logic emulation, and high performance system design.