시퀀트 멀티프로세서의 시뮬레이션에 관한 연구

임청규*, 이정열*

요약

본 논문은 메모리 공유 기반의 시퀀트 멀티프로세서 시스템에 관한 시뮬레이션 연구이다. 이 시스템은 단일 프로세서의 한계성을 극복하기 위하여 도입된 것이며 또한 심메트리 시스템으로 버클리 유닉스 시스템과 시스템 V 계열 명령어들 모두 운용이 가능한 유닉스 버전이다. 이 시스템의 특징과 구현시 사용되어지는 마이크로 시스템 루틴들이 소개된다. 그리고 이 시스템의 4개의 멀티 프로세서를 활용하여 하나의 병열 프로그램이 구현되어진다. 결과로는, 시뮬레이션 할 때 사용한 4개의 프로세서 성능을 실행 시간과 스피드 업 현상으로 분석한다.

A Simulation Study of Sequent Multiprocessor system

Chung-Kyu Lim*, Jeong-Yeol Lee*

ABSTRACT

In this thesis, the Sequent multiprocessor system based on memory sharing are simulated and discussed.. This system is to overcome the limits of single processor, and is shared memory multiprocessors running version of UNIX and supporting both the Berkeley UNIX and UNIX system V command sets on the symmetry. The characteristics and microtasking routines of the system are shown. A parallel algorithm is implemented on the Sequent system with a certain number of processors, from 1 to 4, respectively. The execution time and the speed-up are used to measure the performance per processors. In conclusion, computational results are cited in the thesis.

Key Words : sequent system, multiprocessor, parallel, simulation, implementation

* 전북과학대학 인터넷정보계열(✉lckyoo@jbsc.ac.kr)

· 제1저자(First Author) : 임청규 · 교신저자(Correspondent Author) : 임청규
· 접수일(2011년 11월 2일), 수정일(1차 : 2011년 12월 2일), 게재확정일(2011년 12월 5일)

I. Introduction

The latest advances in very large scale integration (VLSI) technology have brought about a new generation of parallel computers-which are able to perform several tasks concurrently.

A parallel algorithm offers computations faster than they can be done with just single processor, because problems can be decomposed into subprograms, with each subprogram being solved concurrently. Ideally, this allows a speed-up in the computation proportional to the number of processors employed. The parallel algorithms that we use belong under an SIMD model based on memory sharing, one of four classes of computer[1, 2 ,3].

We implement the parallel sorting-and-searching algorithms that contain the major variants on the Sequent system. The main purpose of this thesis is to study the parallel sorting-and-searching algorithms based on the memory-sharing model, to implement the parallel algorithms on the Sequent system, and to check the execution time and the speed-up of the algorithms.

The remainder of this thesis is organized as follows. In Chapter 2, we survey the Sequent system and its family, the Symmetry system. And then we introduce how to provide proper data flow between loop iterations for parallel execution, and how to analyze data flow within a loop. Also, This Chapter is devoted to odd-even transposition sort and the complexity on the Sequent system. In Chapter 3, we will give the implemented results of parallel algorithms on the Sequent system. Chapter 4 concludes this thesis.

II. The Sequent System

The Sequent system [4] as the MIMD-TC model is tightly coupled multiprocessors. There are two families: the Symmetry system and the Balance system. These two systems are very similar in their structure, configuration, OS, and user software. The primary difference between them is the microprocessor used. The Symmetry system uses the Intel 80386 while the Balance system uses the National Semiconductor NS32032.

2.1 Symmetry System

The Symmetry system is shared memory multiprocessors running the DYNTX operation system, version of UNIX 4.2BSD and supporting both the Berkeley UNIX and UNIX system V command sets[2]. The CPUs of Symmetry system are general-purpose, 32-bit microprocessors. The system typically incorporates from 2 to 30 microprocessors and a shared memory.

A Symmetry system is configured around the Sequent System Bys (SSB), Which links the system's CPUs, memory, and I/O subsystems.

표 1. 심메트리 시스템 특징
Table 1. Characteristics of the Symmetry system

True Multiprocessor
Tightly Coupled
Common Bus
Symmetric
Transparent
Dynamic Load Balancing
Shared Memory
Hardware support for mutual

Peripheral interface boards on the system bys enable peripheral devices to access system memory via the SCSI (Small Computer System Interface) bus and via MULTTBUS. The characteristics of the Sequent system are shown in Table 1.

2.2 The Microtasking Routines

The routines constitute the Parallel Programming Library, which supports microtasking in programs. For information on microtasking programming models, refer to [5].

The routines described here support microtasking. The microtasking routines allow us to fork a set of child processes, assign the processes to execute loop iterations in parallel, and synchronize the processes as necessary to provide proper data flow between loop iterations. Table 2 lists the microtasking routines.

표 2. 마이크로테스팅 루틴
Table 2. Microtasking routines

m_fork	Execute a subprogram in parallel.
m_get_myid	Return process identification number.
m_get_numprocs	Return number of child processes.
m_kill_procs	Terminate child processes.
m_lock	Lock a lock.
m_multi	End single-process code section.
m_next	Increment global counter.
m_park_procs	Suspend child process execution.
m_rele_procs	Resume child process execution.
m_set_procs	Set number of child processes.
m_single	Begin single-process code section.
m_sync	Check in at barrier.
m_unlock	Unlock a lock.

The m_fork routine assings a subprogram to child processes, which cooperate in executing the subprogram in parallel. The number of child processes used by the m_fork call can e set with a previous call to m_set_proce. If desired, the number given to m_set_procs can be dependent on the number of actual CPUs available, which is determined by the function CPUS_online, a data-partitioning routine. After execution of the loop, the program returns from the m_fork call and the child processes spin, waiting for more work.

2.3 Parallel Simulation Algorithm

In this chapter, we will present parallel simulation algorithm that is designed to run on the Sequent system. The algorithm are written in pseudocode and the examples of those will be shown in Fig 1.

2.3.1 Odd-Even Transposition Sort

The Odd-Even Transposition sort can be considered as an algorithm for sorting n elements using N processors in O (n) them. The sequence S={x1, x2 , ⋯ , xn} can be sorted by t he following two processes. . In step1, we call and odd_process procedure. For i=0, 1,⋯, (n-1) / 2, N processors compare odd-numbered elements x2i+1 and x2i+2, and if x2i+1>x2i+2, the two elements are exchanged. In the step2, we call and even_process procedure. For I = 0, 1, ⋯ , (n/2)-1, the similar comparison exchanges are executed with even-numbered elements x2i+2 and x2i+3. These two steps are executed in parallel [n/2] times. We are write the Odd-Even transposition sort in pseudo code as shown in Algorithm 1[6].

```
-------------------------------------------
/* main program *
for j = 1 to[n/2]
    call an odd_process procedure;
    call an even_process procedure;
end for;

step1 : /*odd-numbered elements are compared
*/
odd_process procedure;
for i = 0 to (n-1)/2 /* in parallel */
    if x2i+1>x2i+2 then
    [x2i+2<->x2i+3;]/*exchange*/
    end if;
end_for;
step2 : /*even-numbered elements are compared
*/
even_process procedure;
for i= 0 to (n/2)-1
    if x2i+2> x2i+3 then
    [x2i+2<-> x2i+3;] /* exchange*/
    end_if;
end_for;
-------------------------------------------
```

알고리즘 1. 홀수 짝수 정렬
Algorithm 1. Odd-Even transposition sort

Let S={8, 7, 6, 5, 4, 3, 2, 1}. The Odd-Even transposition for this input is illustrated in Figure 1. We use 4 processors P0, P1, P2, P3 to sort the sequence S. In step 1, each P0, P1, P2, P3 calls an odd-processor procedure, and the odd-numbered elements. In step 2, each processor P0, P1, P2, P3 calls an even-processor procedure, and sorts rhe even-numbered elements. After ⌈n/2⌉ times, the sequence S is sorted.

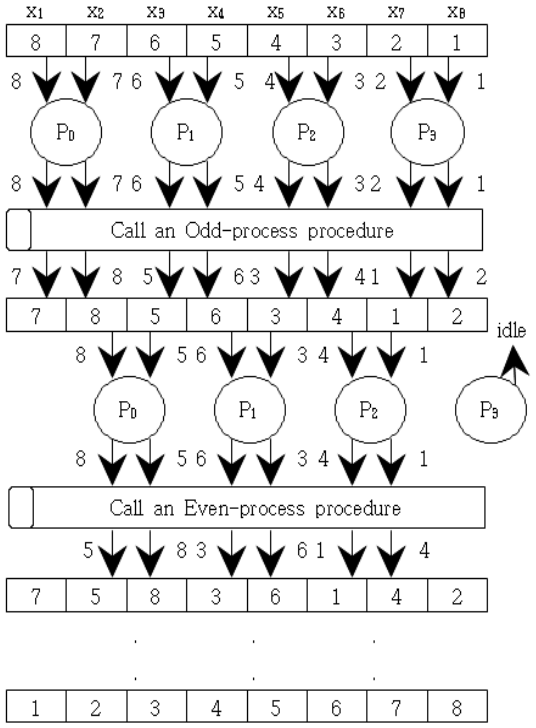

그림 1. 홀수 짝수 정렬 과정
Fig 1. Odd-Even transposition sort

2.4 Computational Complexity

We survey the fundamental concepts of computational complexity in section 2.4.1. In section 2.4.2 we summarize the computational complexity of the algorithm discussed in sections 2.3.1.

2.4.1 The Fundamental Concepts

The fundamental concepts to evaluate a parallel algorithm are followed as: running time, the number of processors used, cost, and speed-up.

2.4.1.1 Running Time

Running time is defined as the time taken by algorithm, that is, the time elapsed from the moment the algorithm start to the moment it terminates.

Running time is usually obtained by counting two kinds of steps executed by the algorithm:

· Routing steps. Data travel from one processor to another through the communication net work or via shared memory.

· Computational step. This step is an arithmetic or logic operation performed on data within a processor.

For a problem of size n, the parallel worst-case running time of an algorithm, a function of n, will be denoted by t(n).

2.4.1.2 Number of Processor

Another criterion for accessing the value of a parallel algorithm is the number of processors it requires so solve a problem. For a problem of size n, a function of n, will be denoted by p(n).

2.4.1.3 Cost

For a problem of size n, the cost of a parallel algorithm, a function of n will be denoted by c(n). The cost of a parallel algorithm is defined as the product of the previous two measures: hence c(n) = t(n) * p(n).

2.4.1.4 Speed-up

The major advantage of multiprocessor systems, as compared to the single processor systems, is the "speed-up" achieved with the increment of number of processors. The speed up is used to measure performance. The speed-up S(N) is computed by dividing T(1) by T(N), where T(1) is the execution time for the sequential algorithm obtained by removing the micotasking routines and Shared and Private in declaration statements, and T(N) is the execution time of the parallel algorithm using N processors.

2.4.2 Computational Complexity

We summarize the a symptom bounds of the algorithm in terms of the number of processors uti9lized and execution time (the latter being estimated as the number of parallel comparison steps required by the algorithms). The computational complexities of the parallel sorting-and-searching algorithms are given in Table 3.

표 3. 프로세서 수와 실행시간
Table 3: Number of Processors and Running Time

Algorithm	Processors P(n)	Time T(n)
Odd-Even Transposition sort	N	$O(n^2/N)$

· Odd-Even Transposition sort. With respect to a quick_sort algorithm, it achieves speed-up. Since p(n) = N, the algorithm's cost is given by c(n) = O(n2). Its cost is unreasonable.

III. Implementation Results

The parallel algorithm discussed in the previous chapter is implemented on the Sequent system with a certain number of processors, from 1 to 4, respectively. For the implementation of the algorithm, the system call gettimeofday is available. The methods to measure implemented results are the following; (1) the execution time is represented in the way of the number of processor per second and (2) speed-up is represented in the way of the number of

processors pet second.

· Odd-Even transposition sort. Figure 2 shows that the simulation algorithm can reduce the execution time while increasing the number of processors compared to the single processor. But some part in not linear in 3 processors. The Delay occurs.

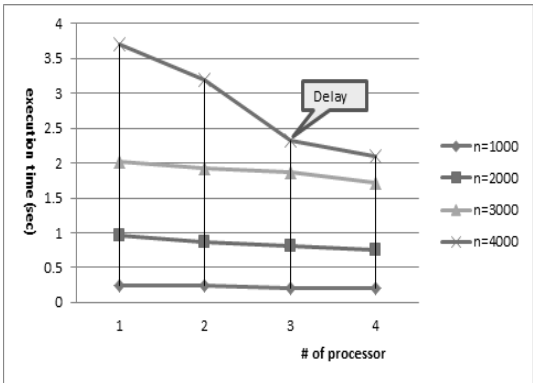

그림 2. 프로세서별 실행시간
Fig. 2. execution time per processors

Figure 3 shows the speed-up occurs with the number of processors greater than 1. The Delay occurs.

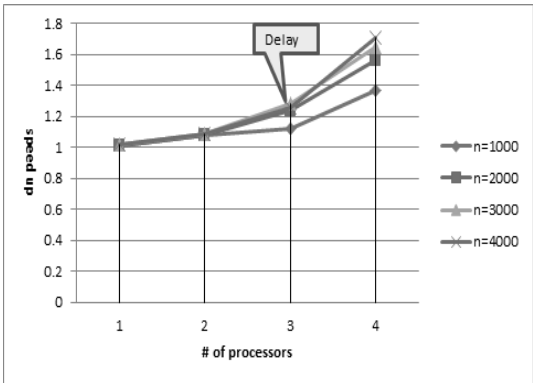

그림 3. 프로세서별 스피드 업
Fig. 3. speed-up per processors

IV. Conclusion

On the Sequent system, we have implemented the odd-even transposition algorithm. From the implemented results, we know the following.

· Speed-up linear. Most algorithms that have a speed-up are linear in the number of processors. In other words, the performance of a N processor system is N times that of a single processor.

· Variable analysis. A analysis of the loop variables with the help of a variable classification scheme discussed in Chapter 2 can be a great aid to parallelization and can improve the performance. The variables in the loop can affect the performance of the algorithm.

· Communication overhead. Because of the communication overhead between N processors, the actual performance of the algorithms is not proportional to N, the data receiving and sending through the bus affect the performance.

The odd-even transposition algorithm implemented on Sequent system have brought some performance. But in the future, it is necessary to study the following issues for better performance.

· A bout inner loop. From the implementation result, we know that executing the inner loop in parallel affects the speed-up.

· Load balanced. In processing a loop in parallel, it is the study of balancing the load. The loop iterations may be distributed among the processes in order that each iteration involves the same of computation.

· Resident size set. It is the study of adjusting resident se size. A process's resident set size determines the amount of text and data it can maintain in physical memory. Once the resident set

size is determined, pages of text and data are brought into memory as needed until the resident set is filled. At that point, one page must be traded out of the resident set for every page that is brought in. If the pages containing locks are traded out, delaying processes occur because the locks are not immediately accessible. This can interfere with the performance of the parallel algorithms.

· Bus bottleneck. The Symmetry system has an architecture based on single-bus. So, as the number of processors scale up, the bus will eventually become the bottleneck of the system shown in the delays of Fig 2 and Fig 3. It is worth studying about the bottleneck for different numbers of processors.

In conclusion, we present that an efficient parallel simulation algorithm can overcome the limits of single processor by making use of odd-even simulation scheme. Also, we look forward to putting more efforts on those issues.

참고문헌

[1] MICHAEL J.QUINN. Designing Efficient Algorithms for parallel computers. McGRAW-HILL INTERNATIONAL EDITIONS., pp.23-49. 1987

[2] FLYNN, M. J. "Very high-speed computing systems.", Proc, IEEE 54, pp.1901-1909. Dec., 1966.

[3] K. G. Jones and S. R. Das. Parallel execution of a sequential network simulator. In 2000 Winter Simulation Conference Proceedings, pages 418 - 424, December 2000.[1] Sequent Computer System. SEQUENT GUIDE TO PARALLEL PROGRAMMING, 1987

[4] Sequent Computer Systems, SEQUENT(TM) TECHNICAL SUMMARY, 1987..

[5] Sequent Computer Systems, DYNIX PROGRAMMER'S MANUAL, BOLUME1, SECTIONS 2-3, 1987.

[6] MONOJ KUMAR. and DANIEL S. HIRSCHBERG. "An Efficient Implementation of Batcher's Odd-Even Merge Algorithm and Its Application in Parallel Sorting Schemes.", IEEE TRANS. Comput., 1983

저자소개

임청규(Chung-Kyu Lim)

1986년 한남대학교 전자 계산공학과 졸업
1990년 自由中國(臺灣) 國立交通大學校 資訊情報工學科 工學碩士
1998년 전북대학교 전자공학과 박사과정 수료
1995년~현재 : 전북과학대학교 인터넷정보계열 교수
※ 관심분야 : 정보통신, 보안, 알고리즘

이정열(Lee Jeong Yeal)

1986년 8월 동국대학교 경영대학원 경영학석사
1994년 2월 강원대학교 대학원 이학석사
1998년 8월 전북대학교 전산통계학과 박사과정 수료
1995년~현재 : 전북과학대학교 인터넷정보계열 교수
※ 관심분야: 소프트웨어공학, 멀티미디어, 모바일컴퓨팅