동시 다중 쓰레딩 마이크로프로세서를 위한 스케쥴링 알고리즘의 성능 평가

이 정 훈[†] · 김 진 석[††]

요 약

최근 많은 프로세서 제작업체들이 프로세서의 효율을 높이기 위한 방법으로 독립적인 쓰레드들을 한 프로세서 사이클에 동시에 실행시킬 수 있는 동시다중 쓰레딩 기술을 구현하고 있으며 그 예의 하나가 하이퍼쓰레딩이다. 물리 프로세서 안에 여러 개의 논리 프로세서를 가질 수 있는 하이퍼쓰레딩 기술은 기존의 여러 개의 독립적인 프로세서들을 갖춘 멀티 프로세싱 환경과는 차이가 있으며, 하이퍼쓰레딩 환경에 알맞은 특정한 작업 할당 방법이 필요하다.

따라서, 본 논문에서는 하이퍼쓰레딩 기술에 적합한 스케쥴링 알고리즘을 제안하고 그 성능을 다양한 방법으로 측정해 봄으로서 하이퍼쓰레딩 시스템을 올바르게 인식하고 적절하게 관리하여 효율적인 성능을 기대할 수 있게 되었다.

Performance Evaluation of a New Scheduling Algorithm for the Simultaneous MultiThreading Microprocessor

Jung-Hoon Lee[†] · Jin Suk Kim[††]

ABSTRACT

Recently, many processor manufacturers have implemented simultaneous multi threading technology, which can simultaneously execute independent threads in one processor cycle, as a way of increasing processor efficiency, and, one particular example is Hyper Threading. Hyper Threading technology, which enables many logical processors to reside a physical processor, differs from the current multiprocessing environment, which has many independent processors, and calls for a particular work assignment method optimized for Hyper Threading environment. Thus, in this paper, We have proposed a scheduling algorithm compatible with Hyper Threading technology and analyzed its performance using various methods. As a result, we shall expect its efficient performance by properly understanding and managing Hyper Threading system.

키워드 : 동시(simultaneous), 다중 쓰레딩(multi threading), 성능(performance)

1. Introduction

SMT (Simultaneous MultiThreading) [1] technology, proposed by Dean Tullsen in 1995 to decrease the waste of processor resources and increase efficiency, has been actually implemented on a real processor under the name of Hyper Threading [2]. Hyper Threading technology is designed to process different threads simultaneously by enabling one physical processor to have two logical processors[3]. Hyper Threading technology allows each logical processor to share resources such as execution engine, cache, system BUS interface, etc. in a physical processor [4].Thus, from the application's perspective, one processor may be seen as multiple processors, and Hyper Threading technology runs many independent threads at the same time, thereby maximizing the parallelism of the processor, and as a result, increases the performance of the processor entirely. The characteristics of such Hyper Threading system differ from the current multi-processor environment, which possesses independent processor resources, and, its scheduling policy should be approached in a different way for efficient resource utilization and processor performance increase. Therefore, in this paper, Wehave proposed a scheduling algorithm and applied it on the Linux operating system, so that people can expect efficient performance based on this research by properly

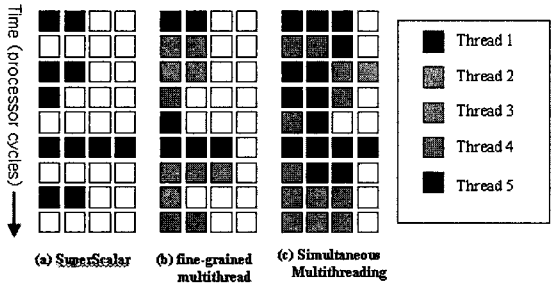
※ 이 논문은 2003년도 서울시립대학교 학술연구조성비에 의하여 연구되었으며, 이에 감사드립니다.
† 정 회 원 : 하이닉스 반도체 사원
†† 정 회 원 : 서울시립대학교 컴퓨터과학부 부교수, Corresponding Author
논문접수 : 2004년 12월 8일, 심사완료 : 2005년 3월 3일

understanding the system supporting Hyper Threading technology and managing logical processors.

Later in this paper, the following topics are discussed. In section 2, we will examine the characteristics and problems of Hyper Threading system, as compared in the traditional SMP system. In section 3, we will examine the related researches covering the solutions to the problems, and in section 4, We will explain the algorithm proposed in this paper. In section 5, We will compare and analyze the performance of the proposed algorithm, and finally, in section 6, We will draw a conclusion and suggest directions for future researches.

2. Characteristics of the Hyper Threading System

Researches on increasing processor performance is divided into two broad categories, one that increases ILP (Instruction-Level Parallelism), and another that increases TLP (Thread-Level Parallelism). ILP increments the number of instructions that can be executed within one processor cycle, and TLP executes many threads on many processors simultaneously. Recently, however, SMT technology, which can efficiently utilize the wasted resources of a processor and eventually, increase the performance of a processor as a while, has been proposed. In SMT technology, one physical processor is made up of many logical processor, sharing physical processor resources, and SMT aims to increase the performance of a processor by simultaneous multi-threading within one processor cycle. Simultaneous multi threading method is a mixture of wide-issue superscalar and multithreaded processor. In other words, as shown in Fig. 1, similar to superscalar, multiple instructions can be run in each cycle, and similar to multithreaded processor, it supports multiple threading method.



(Fig. 1) Simultaneous Multiple Threading Method [9]

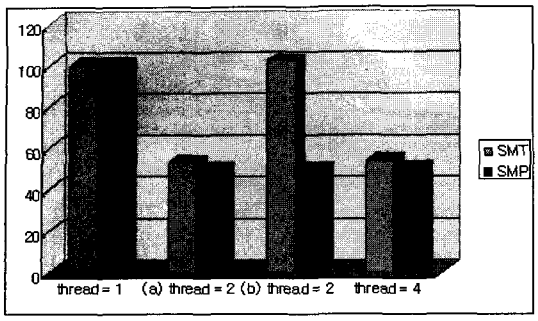
Superscalar increments ILP because it can execute multiple instructions within a cycle, and fine-grained multithread or traditional multithreaded processor increases TLP, for it has many hardware contexts and can independently processes different threads during multiple cycles. But, SMT increases both TLP and ILP, for it is designed to process many hardware contexts simultaneously, and decreases vertical issue slot waste and horizontal issue slot waste at the same time. This SMT technology, under the name Hyper Threading, has actually been implemented on real microprocessors. Hyper Threading technology possesses multiple independent architectural state registers and is designed to function independently to other logical processors, so each can be stopped, interrupted, or process threads [2]. Such differences in the processor designing stage, traditional SMP systems, which independently possesses cache, system BUS, execution engine, etc, differs greatly, and effects system performance on application stage in various ways.

If that is the case, we need to examine the characteristics of Hyper Threading technology on application stage, and why the current operating system needs to adopt different scheduling policy, compared to the scheduling policy of existing SMP environment. To do that, in the paper, we have conducted experiments using dual Intel Xeon 2.4GHz processor, which supports Hyper Threading technology, Linux kernel 2.4.17, Intel C++ complier, and OpenMP version 2.0, and used Intel VTune analyzer to analyze the results. Each experiment has been conducted by analyzing the completed result obtained from a routine calculating the value of by running parallel code obtained using OpenMP, and the routine executed by each thread is experimented in two categories, homogeneous codes made up of integer operations and heterogeneous codes made up of integer operation and float operation.

Fig. 3 displays the result of running the thread code of Fig. 2 according to the changes in the number of thread in seconds. To compare Hyper Threading / SMT system

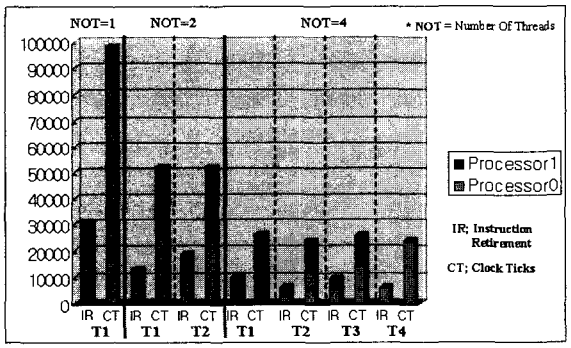
```
int compute_pi_by_Thread() {
  int i;
  int x, pi, sum=0, step;
  int myid;
  printf ("Entering compute_pi 1.");
  step = 1/MAX;
  myid = omp_get_thread_num();
  for (i=1; i < MAX; i++) {
  x = (i-1)*step;
  sum = sum + 4/(1+x*x); }
  pi = sum*step;
  printf("%d, %d thread finished.", pi, myid);
return 0;
```

(Fig. 2) Thread Run Routine

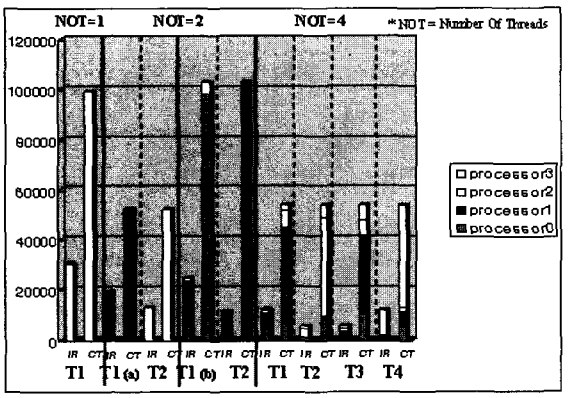


(Fig. 3) Result of Thread Routine Runs

with the existing SMP systems, we have continuously turned on and off the Hyper Threading function in each system and extracted the result value. When the number of threads is one or four, SMT and SMP have displayed similar performance. The reasons Hyper Threading system, which simulates four processors, does not perform much better than SMP system, which simulates two processors, are extremely efficient branching forecasting, cache utilization rate, and pipeline stalling, which are relatively small in numbers caused by the characteristics of thread routine, which is made up of simple for loops. One peculiar result of this experiment is that when the number of threads is two, two similar results (a, b) are repeatedly displayed. To investigate this phenomenon, we have analyzed it using VTune analyzer. The result is shown below:



(Fig. 4) Analysis of Test Runs on SMP System



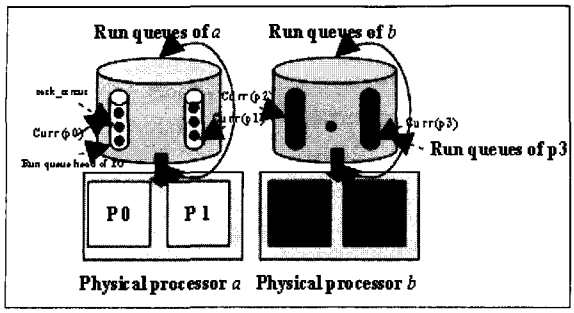
(Fig. 5) Analysis of Test Runs on Hyper Threading System

Fig. 4 shows the result of running the experiment shown in Fig. 3 on SMP, and Fig. 5 displays the result of running the same routine on a Hyper Threading system. IR designates the number of Instruction Retired, CT for the Clock Tick, NOT for the Number Of Thread, and T1, T2, T3, T4 for each Thread. In Fig. 4, we can observe that in case of SMP, two processors is performing proper scheduling, but the Hyper Threading system in Fig. 5, when NOT=2, operations are executed using improper resource allocation policies within the four logical processor resources. In other words, number 0 and 1 are the logical processors actually included in the same physical processor, and processor number 2 and 3, displays properly displays normal results, as shown in Fig. 3(a), when 2 threads are allocated as physically independent processors number 1 and 3 in a Hyper Threading system consisted of logical processors within the same physical processor as shown in Fig. 5(a). In contrast, in case of Fig. 5(b), two threads are each allocated to the two logical processors number 0 and 1 that belongs to the same physical processor, and as a result, bad results are obtained as shown in Fig. 3(b). This phenomenon occurs because in existing operating systems, the special characteristics of Hyper Threading systems are not recognized and scheduling policy implemented using the same method used in the existing multiprocessor environment. To overcome such shortcomings, we need a resource assignment policy appropriate for Hyper Threading technology.

3. Related Work

To recognize Hyper Threading system and properly manage processor resources, in Linux kernel 2.4, various features, such as 128-byte lock alignment, Spin-wait loop optimization, Non-execution based delay loops function, etc. are added. But in reality, we cannot say that these features have actually solved such shortcomings displayed in the experiments described above. To solve such problems, Molnar proposed a method that changes queue in Linux kernel 2.5.32, as shown in the figure below[5].

Fig. 6 displays the scheduling method proposed by Molnar. A common process queue is generated in a physical processor consisted of two logical processors, and processor queue for each logical processor are formed in lower stages. Molnar's proposal precisely recognizes and manages Hyper Threading system, and effectively



(Fig. 6) Scheduling Algorithm Method Proposed by Molnar

deals with the particular characteristics of the Hyper Threading system displayed in the experiment. At the same time, it is too complex, not extensible, and cannot be applied in ordinary SMP systems or uni-processor environment. Also, newly generated common queue has other potential side effects. Therefore, in this paper, We have developed a scheduling algorithm, which can perform resource management more suited for Hyper Threading system using a relatively simple policy, rather than a complex design structure and implementation such as Molnar's.

4. A Scheduling Algorithm for Hyper Threading System

Considering that each process allocated to a logical processor in a Hyper Threading system should be assigned to different independent processors preferably, HT-scheduling algorithm must detect logical processors included in different physical processors, and, based on the information, assign processes to other physical processors when assigning processes in Linux operating system. To do that, we must know the relationship between the physical processor and the logical processor. For example, in Intel processors, they are classified by APIC (Advanced Programmable Interrupt Controller) ID[6], and based on such information, we can define the number and relationship information of logical processors included in each physical processor[7]. Based on the information, HT-scheduling algorithm reads the number of logical processors included in a physical processor, and takes an approaching method to assign incoming processes to physical processors. If we assume that Ti task is assigned to a system with n physical processors and m logical processors, a simple equation for this task can be expressed as shown in Fig. 7.

Also, if we restart a process which has already been executed in a particular logical processor, we can maximize the efficiency of shared resources by applying the same accumulated value to all other logical processors included in the same physical processor. Such method, recognizing the relationship between logical processors and physical processors in a Hyper Threading system and enabling the scheduling that considers the characteristics displayed in the previous experiments, maximizes the efficiency of the Hyper Threading system. Also, due to the relatively simple design, it has high extensibility and portability, and, can be easily applied in the current multiprocessor environment.

Algorithm HT−scheduler

Input : $T_i,\ i \geq 0$
Output : $P_j(C_k),\ 0 \leq j \leq n-1$ and $0 \leq k \leq m-1$
1. *find* $j = i \bmod n$
2. *find* $k = (i / n) \bmod m$
3. *return* $P_j(C_k)$

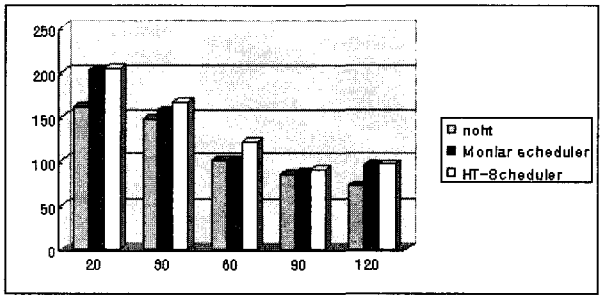
(Fig. 7) HT-scheduling Algorithm

5. Performance Analysis

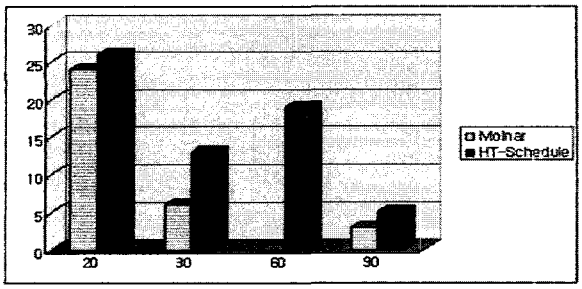
To analyze the performance of HT-scheduling algorithm proposed in this paper, We have used dual Intel Xeon 2.4GHz processor, which supports Hyper Threading, Linux kernel 2.5.3, and various benchmarking toolkits such as dbench benchmarking toolkit. This environment is slightly different from previous one at section 2 because of the linux kernel version limit which can work properly on the Hyper-Threading systems.

Firstly, the dbench [8] benchmarking toolkit is primarily used for the measurement of the performance of file server, which allows multiple file access on a network. It generates a great amount of loads, runs processor operations, and has a high number of accessed or generated files. Thus, it is known to be an appropriate tool for Hyper Threading system performance measurement.

Fig. 9 displays the performance comparison and analysis of a system not considering Hyper Threading, sched-



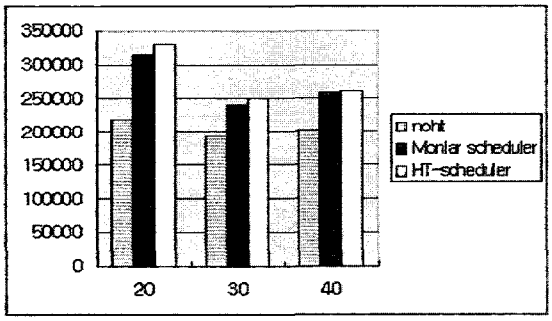
(Fig. 9) Result of Dbench Performance Analysis



(Fig. 10) Result of Dbench Speed Up Analysis

uler proposed by Molnar, and the scheduler proposed in this paper. The X axis is the parameter value of dbench, and as the parameter value increases, the number of client access, in other word, workload, increases. The Y axis displays throughput in MB/sec, and higher value means better performance.

Fig. 10 displays the Hyper Threading effect on the dbench throughput in Fig. 9, and based on the noht in Fig. 9, speed of Molar's scheduler and the HT-scheduler proposed in this paper is displayed in percentage. The x axis, same as Fig. 9, is the value of debench parameter, and the y axis is the percentage of speedup as compared to noht, higher value being better performance. In most cases, the HT-scheduler performs better.

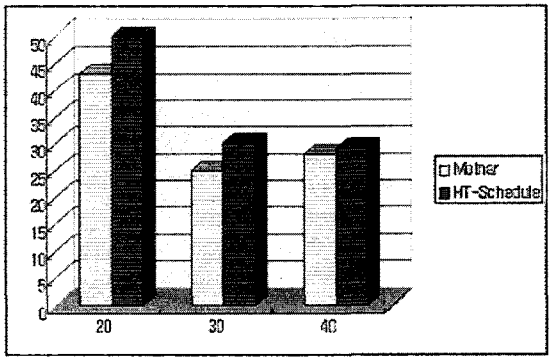


(Fig. 11) Result of Chat Benchmarking Performance Analysis

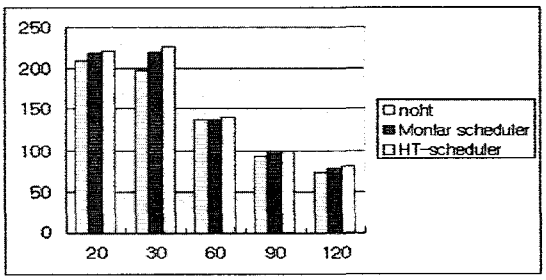
Fig. 11 shows the result of using a benchmarking toolkit [10] to measure the effect of Hyper Threading on a application stage, where real MultiThreading is used. The x-axis means the number of chat rooms, and the y-client is the number of messages transmitted from the chat client, bigger number meaning better performance. The chat benchmarking toolkit measures the performance of a MultiThreading application when the chat client program creates a connection to the chat server via TCP/IP, and sends and receives messages. This benchmarking opens the number of chat rooms designated when the toolkit starts, and each chat room receives 20 users, i.e., connections. Each connection generates two threads, one that sends messages and another that receives messages, and each client thread sends a specific number of messages to the server. The following table displays chat rooms and the related number of connections, threads, and messages.

〈Table 1〉 Relationship between the Number of Chat Rooms and Messages

chat room number	No. of Connections	No. of Threads	No. of Transmitted Messages	No. of Received Messages	Total No. of Messages
20	400	1,600	40,000	760,000	800,000
30	600	2,400	60,000	1,140,000	1,200,000
40	800	3,200	80,000	1,520,000	1,600,000



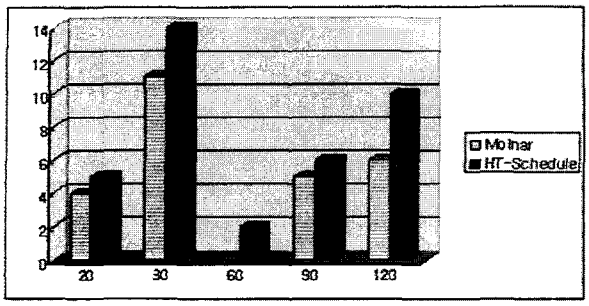
(Fig. 12) Result of Chat Benchmarking Speed Up Analysis



(Fig. 13) Result of Tbench Benchmarking Performance Analysis

Fig. 12 is the analysis of the improved speed up with noht as a standard in the chat benchmarking shown in Fig. 11, the x-axis is the number of chat rooms, and the y-axis is the improved speed up in percentage, compared to the noht.

Fig. 13 shows the performance measurement of tbench [8], which is primarily used to measure file server workload, tbench performs socket calls through TCP/IP, but does not call file system, and is distributed with dbench benchmarking toolkit. In Fig. 13, the x-parameter is the value of tbench parameter, and the x-axis is the throughput in MB/sec, higher value meaning better performance, as in dbench benchmarking.



(Fig. 14) Result of Tbench Benchmarking Speed Up Analysis

Fig. 14 is the analysis of the improved speed up as compared to noht in tbench benchmarking. The x-axis is the parameter value of tbench benchmarking, same as Fig. 13, and the y-axis is the percentage of the improved speed up, as compared to noht. Taken as a whole, the HT-scheduler is always better when compared to noht, and, it is equal or slightly better when compared to Molar's scheduler. HT-scheduler is improved performance maximum 14% better than noht.

6. Conclusion and Directions for Future Researches

In this paper, We have proposed HT-scheduling algorithm to overcome the difference between Hyper Threading system and the current multiprocessor environment and through various experiments, demonstrated that the HT-scheduler algorithm performs excellently so we can expect efficient performance by understanding and managing systems that support Hyper Threading technology. HT-scheduler is improved more than maximum 14% performance than noht and 3~4% performance than Molnar's scheduler. In the future, we shall analyze the efficiency of Hyper Threading system related to the thread heterogeneousness, and continue to work on researches regarding more efficient scheduling algorithm and plans that satisfies many scheduling needs on the real applying stage based on the analysis.

References

[1] D. Tullsen. S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm, "Simultaneous MultiThreading: Maximizing On-Chip Parallelism," Proc. of the 22nd Annual International Symposium on Computer Architecture, June, 1995.

[2] Intel Corporation, "Introduction To Hyper-Threading Technology," Document number 250008-002, 2001.

[3] Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A. Koufaty, J. Alan Miller, and Michael Upton, "Hyper-Threading Technology Architecture and Microarchitecture," Intel Technology Journal, Q1, 2002.

[4] Intel corporation, "White Paper: Hyper-Threading Technology on the Intel Xeon Processor Family for Servers," 2002.

[5] http://lwn.net/Articles/8553/

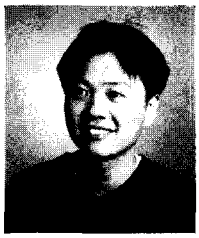
[6] Intel corporation, "IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture," 2001.

[7] Intel corporation, "Detecting Support for Hyper-Threading Technology Enabled Processors," Dec, 2001.

[8] http://samba.org/ftp/unpacked/dbench/README

[9] D. Tullsen et. al., "Simultaneous MultiThreading : a Platform for Next-Generation Processors," IEEE Micro, pp. 12-19, Oct, 1997.

[10] Linux Benchmark Suite Homepage, http://lbs.sourceforge.net/



이 정 훈

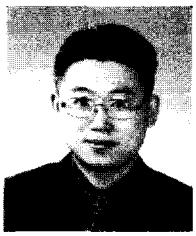
e-mail : jhlee94@sidae.uos.ac.kr

2001년 서울시립대학교 전산통계학과 졸업(학사)

2004년 서울시립대학교 컴퓨터통계학과 졸업(석사)

2004년~현재 하이닉스 반도체 사원

관심분야 : 클러스터링 시스템, 병렬처리, 공개소프트웨어



김 진 석

e-mail : jskim@venus.uos.ac.kr

1990년 과학기술대학 전산학과 졸업(학사)

1992년 KAIST 전산학과 졸업(석사)

1997년 KAIST 전산학과 졸업(박사)

1997년~1999년 KAIST 인공지능연구 센터 Postdoc 연구원

1997년~1998년 미국 M.I.T. Laboratory for Computer Science Postdoc Fellow

1998년~1999년 전자통신연구원(ETRI) 슈퍼컴퓨터센터 초빙 연구원

1999년~현재 서울시립대학교 컴퓨터과학부 부교수

2005년~현재 서울시립대학교 고성능컴퓨팅 센터(HPCRC) 센터장

관심분야 : 그리드 컴퓨팅, 금융공학, 병렬처리 시스템, 멀티미디어, 인터넷과 정보검색