THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-02-01

Performance Analysis of KDD Applications using Hardware Event Counters

Peter Christen and Adam Czezowski

February 2002

Joint Computer Science Technical Report Series

Department of Computer Science
Faculty of Engineering and Information Technology

Computer Sciences Laboratory
Research School of Information Sciences and Engineering

This technical report series is published jointly by the Department of Computer Science, Faculty of Engineering and Information Technology, and the Computer Sciences Laboratory, Research School of Information Sciences and Engineering, The Australian National University.

Please direct correspondence regarding this series to:

Technical Reports
Department of Computer Science
Faculty of Engineering and Information Technology
The Australian National University
Canberra ACT 0200
Australia

or send email to:

`Technical.Reports@cs.anu.edu.au`

A list of technical reports, including some abstracts and copies of some full reports may be found at:

`http://cs.anu.edu.au/techreports/`

Recent reports in this series:

TR-CS-01-02 Jeremy E. Dawson and Rajeev Gore. *Mechanising cut-elimination for display logic*. October 2001.

TR-CS-01-01 Stephen Roberts Peter Christen, Markus Hegland and Irfan Altas. *A scalable parallel fem surface fitting algorithm for data mining*. October 2001.

TR-CS-00-02 Peter Strazdins. *A survey of simulation tools for cap project phase iii*. October 2000.

TR-CS-00-03 Bill Clarke. *Sparc v9 instruction set specification*. October 2000.

TR-CS-00-01 Jens Gustedt, Ole A. Maehle, and Jan Arne Telle. *Java programs do not have bounded treewidth*. February 2000.

TR-CS-99-02 Samuel Taylor. *A distributed visualisation tool for digital terrain models*. July 1999.

Performance Analysis of KDD Applications using Hardware Event Counters

Peter Christen* and Adam Czezowski

CAP Research Group, Department of Computer Science
Australian National University, Canberra
ACT 0200, Australia

URL: http://cap.anu.edu.au/cap/projects/KDDMemPerf/

Abstract. Modern processors and computer systems are designed to be efficient and achieve high performance with applications that have regular memory access patterns. For example, dense linear algebra routines can be implemented to achieve near peak performance. While such routines have traditionally formed the core of many scientific and engineering applications, commercial workloads like database and web servers, or decision support systems (data warehouses and data mining) are one of the fastest growing segments in the high-performance computing market. Many of these commercial applications are characterised by complex codes and irregular memory access patterns, which often result in a decreased performance. Due to their complexity and the lack of source code, performance analysis of commercial applications is not an easy task. Hardware performance counters allow acquisition of low level, reliable data, necessary to perform detailed analysis of program behaviour. In this paper we describe experiments and present first results conducted with various KDD applications on an UltraSPARC III platform.

1 Introduction

Commercial applications like database and web servers, or decision support systems (data warehouses and data mining) represent one of the most rapidly growing segments in the high-performance computing market. While modern processors and memory systems are designed to be efficient and achieve high performance with applications that have regular memory access patterns (like dense linear algebra software) they often perform poorly when running commercial applications. Such applications are characterised by complex codes, irregular memory access patterns and large dynamic data structures.

A rapidly growing market segment is KDD (Knowledge Discovery in Databases) or Data Mining [11], which deals with the analysis of large and complex data sets. KDD applications combine techniques from machine learning, statistics, databases and high-performance computing. Tasks involved are data exploration and preprocessing, clustering, predictive modelling, association rules, decision

* Corresponding author, E-Mail: Peter.Christen@anu.edu.au

tree induction, and others. A common characteristic of these tasks is that they are (1) compute intensive, (2) operate on large and complex data sets, and (3) involve irregular memory access patterns due to their dynamic and often recursive data structures (like hash tables, trees or linked lists). The first two characteristics makes them attractive for implementation on high-performance platforms, especially for shared memory multiprocessors (where all CPUs have access to one memory system), while the last characteristic is an obstacle for efficient system utilisation and high performance. Traditional compiler optimisation techniques based on spatial and temporal locality - which proved to be successful for many scientific and engineering applications - cannot successfully be applied for KDD and related applications.

Hardware performance counters are an easy to use instrument which can provide reliable data, necessary to perform detailed analysis of application performance behaviour at low level. Such counters are available on most modern microprocessors, including *UltraSPARC*, *Pentium* and *Alpha*. They can count various events, including number of loads, stores and floating-point operations, cache and TLB misses, branch mispredictions, or cycles per instruction. The number and types of events that can be counted differ widely on various hardware platforms. However, some core set of events such as memory references and instructions counts are available on most of the processors. Machine and operating system dependent libraries (like the *Solaris* `libcpc` [10]) provide access to hardware counters on a specific platform and operating system. Platform independent counter libraries are currently under development in several research projects. Two such libraries are the *Performance Counter Library (PCL)* [3] and the *Performance Application Programming Interface (PAPI)* [6]. Their aim is to provide a set of platform independent interfaces to the counters, that allow easy portability of programs instrumented with these libraries, and to allow inter-platform performance comparisons.

In the next section we present some work that has been done previously in the area of performance analysis of commercial applications, and in Section 3 we present our setup consisting of three freely available KDD applications and one platform optimised linear algebra routine. We also discuss the data set we were using. Experiments and first results are presented in Section 4, and finally we give an outlook on future plans in Section 5.

2 Related Research

There is much ongoing research in dedicated KDD and data mining algorithms and in their parallel implementations. In contrast, we are only aware of a small number of publications dealing with the performance analysis of KDD applications [4, 5, 12, 14]. More work has been done on analysing database servers and related commercial applications [2, 19]. To our knowledge, no performance analysis of KDD applications has been done using hardware counters. This can partly be explained by the lack of available source code for KDD applications and, in the past, lack of user-friendly software interfaces to access hardware counters.

The memory behaviour of a parallel association rule algorithm is discussed in [14]. The authors looked at custom memory placement scheme and found that simple schemes (like the different hash tree building blocks being allocated in a single memory region) can be quite efficient improving the execution time for some data sets up to a factor of two. They state that the data structures used by association rules algorithms (hash trees and lists) exhibit poor locality, and the arbitrary allocation of memory makes it difficult to detect and eliminate false sharing. A run time memory allocation library based on the Unix `malloc()` library is presented, which allows customised memory allocation.

Memory characteristics of a parallel implementation of the self-organising map (SOM) neural network model is discussed in [12]. Four characteristics were examined and compared. First, the working set size (temporal locality), second the spatial locality and memory block utilisation, third the communication characteristics and scalability, and fourth the TLB performance. The authors use a simulation tool adapted from the *Augmint* multiprocessor simulator. They conclude that the size of the working set is not sensitive to the number of input records.

In [4, 5] the popular decision tree induction algorithm *C4.5* [15] is analysed in its memory and parallelisation characteristics. The authors are using *RSIM* [13] to simulate three different instruction level parallelism (ILP) processors. One of their conclusions is that such an algorithm is limited by the memory latency and bandwidth, and cache size has a significant effect on performance as well. In [5] a parallel version of *C4.5* optimised for a *ccNUMA* architecture is presented and analysed. This parallel version puts significantly less pressure on the memory hierarchy, and has a larger working set.

The memory system characteristics of some commercial workloads is studied in [2]. The authors present detailed performance studies of three different important classes of workloads: Online transaction processing (OLTP), decision support systems (DSS) and Web index search. They use monitoring experiments and *SimOS* [16] to study the effects of architectural variations. One of their findings is that operating system activity and I/O latencies do not dominate the behaviour of well-tuned database workloads. For OLTP a large off-chip cache is in favour, while DSS and the Web index search are primarily sensitive to the size and latency of on-chip caches.

The performance analysis of the *TPC-C* benchmark on a four-processor Pentium based SMP is presented in [19]. The authors analytically model the performance and then validate their results with simulations (using *SimOS*) and hardware counter experiments. They conclude that experimentally based evaluation of complex commercial applications is time consuming, and that analytical modelling is a feasible alternative.

3 Applications and Data Sets

We choose three KDD applications and one vendor optimised linear algebra code for hardware counter performance analysis using `libcpc` [10] on a *Ultra-*

SPARC/Solaris platform. We only counted events in the core computation routines, i.e. without file in- or output. The programs we analysed use mainly text files (which is generally slow). Commercial versions of such programs would either read data from binary files or access them directly from a database server.

3.1 Decision Tree Induction – C4.5

The freely available popular decision tree induction program *C4.5*[1] [15] was chosen as a typical KDD application. This application has already been analysed in its memory access behaviour using a machine simulator [4, 5].

C4.5 is written in ANSI C, it reads data from a text file and then builds a decision tree. Only the tree building routine, i.e. the function `BestTree()`, was analysed, with the complete primary data set (the table loaded from disk) stored in memory. This data is stored in an array with pointers to vectors, with each vector (one data record) having a length corresponding to the number of attributes. Every element in this vector consists of a `short` and a `float` variable. In the case of a categorical type attribute, the category number is stored as a `short`, while a continuous attribute (a real number) is stored as a `float`. Thus one of the two variables is always unused. The decision tree is a complex recursive data structure that is built dynamically in the tree building routine.

3.2 Association Rule Induction – APRIORI

Mining association rules is a popular data mining algorithm. It is for example used to analyse market basket data to find frequent item sets and extract rules like *'if a customer buys milk then he will most likely also buy cheese.'* For our performance analysis we use a freely available implementation[2] of the *APRIORI* algorithm [1]. An older version of this program is incorporated in the data mining tool *Clementine 5.0*. The program is written in C, it reads a text file with transactional data and writes the resulting rules either into a text file or prints them. The items in the input transactions are stored in a vector data structure as integer numbers. Once all data is loaded, the items are sorted with descending frequency, and a prefix tree is built, which is then modified and updated for item sets of increasing length. Besides pointers to parent and (variable number of) child nodes, the prefix tree contains a counter vector, stored as integer numbers. Once all frequent item sets are found, they are sorted and the extracted rules are printed or saved.

3.3 Additive Models – ADDFIT

The *ADDFIT* algorithm [7] was developed by the ANU Data Mining[3] group and implemented sequentially and on distributed memory platforms (using C/MPI) [8].

[1] `http://www.cse.unsw.edu.au/~quinlan/`
[2] `http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html`
[3] `http://datamining.anu.edu.au`

This algorithm builds an additive model of the data by assembling a dense symmetric linear system in a first step, which is then solved in a second step using either a sequential or parallel solver [8, 17]. For the performance analysis we are not interested in the first step, as it involves reading the data from (binary) files once and assembling each data record into a matrix at data dependent locations. The primary data structure (the records from disk) is loaded and then the assembly is started. Only the assembly routine is analysed. The data dependent assembly results in irregular memory access patterns. For each continuous attribute in a data record four non-zero values are added into the matrix, while for a categorical attribute only one value is added. The locations where these values are added are data dependent and can be anywhere in the matrix. As the assembled linear system is symmetric, only a dense upper triangular matrix with corresponding vector is allocated, whereby each entry is a `double` sized floating-point value. The size of this linear system is determined by the number of categories for categorical and the resolution of the model for continuous attributes, but it is completely independent from the size i.e., number of records, in the primary input data.

3.4 Dense Matrix-Matrix Multiplication – BLAS (SUNPERF)

To allow a comparison with a platform optimised application with regular memory access patterns we also instrumented a dense matrix-matrix multiplication (the *BLAS* routine `dgemm()` as implemented in Sun's *SUNPERF* library) with calls to the `libcpc` hardware counter library. The `dgemm()` routine is typically used in the core of various scientific and engineering applications. For the performance analysis two *Hilbert* matrices were created and dense matrix-matrix multiplications of two such matrices were performed and analysed.

3.5 Data Sets and Data Structures

For *C4.5* and *ADDFIT* we used the *Census-Income* data set which is freely available from the *UCI KDD Archive*[4]. This data consists of a training file which contains 199 523 records and a test set with 99 762 records. For our purpose we concatenated both files into one to get large enough test data. The *Census-Income* data set contains 5 continuous and 37 categorical attributes.

For *APRIORI* we created synthetic data sets of various size and complexity as described in [1]. For the tests we then chose a smaller data set with 10 000 records and a larger one with one million records.

The *primary* data structures used by the KDD applications hold the input data. They are mainly arrays or vectors, and their size and dimension usually increases linearly with the size of the input data set. In the case of *ADDFIT*, this data is only used once, i.e., each data record is access once, but for *C4.5* and *APRIORI* usually several iterations over the input data are needed, each accessing the primary data structure.

[4] `http://kdd.ics.uci.edu/`

Table 1. Program and Test Characteristics

Program	BLAS (SUNPERF)			ADDFIT	
	small	medium	large	small	large
Data	209 × 209 matrices	660 × 660 matrices	2090 × 2090 matrices	Census with 10,4858 records	Census with 209,715 records
Run time	0.075 sec	1.255 sec	44.5 sec	1.3 sec	7.4 sec
Iterations	100	10	1	10	10
Heap size	1,024 KB	10,240 KB	102,400 KB	10,024 KB	90,408 KB
User code	98.54%	99.16%	98.90%	99.89%	98.85%

Program	APRIORI		C4.5	
	small	large	small	large
Data	T5I4D10K with 10,000 records	T10I8D1000K with 1,000,000 records	Census with 8,322 records	Census with 266,305 records
Run time	3.1 sec	42 sec	3.2 sec	423 sec
Iterations	10	1	5	1
Heap size	19,776 KB	70,512 KB	3,960 KB	62,152 KB
User code	97.98%	98.53%	99.61%	76.24%

The *secondary* data structures built by the KDD applications, – the decision tree in *C4.5*, the prefix tree in *APRIORI*, and the dense matrix used by *ADDFIT* – are not directly proportional to the size of the input data. Rather, they are data dependent (e.g. a *C4.5* decision tree) or their size is determined by some parameters (e.g. model resolution in *ADDFIT*). The size of the prefix tree built by *APRIORI* depends both on the data as well as on parameters like *support* and *confidence*[1], which have to be set by the user. It is therefore very hard to specify the amount of memory a KDD application will use.

4 Results

The presented experiments were conducted on a *Sun Blade 1000* workstation with two *UltraSPARC III* processors running at 750 MHz, and having 2 GBytes of main memory. The Level-2 cache (E-Cache) of the machine is 8 MBytes, while the Level-1 Data cache (D-Cache) is 64 KBytes and the Level-1 Instruction cache (I-Cache) 32 KBytes. The measurements were performed on a single processor which was fully reserved for our application. Other processes, including system related processes, were scheduled to run on the remaining processor. Table 1 shows the characteristics of the test programs and data sets we used. All results for a given program/data set pair presented here are averaged over the number of iterations as listed in the table.

We have conducted measurements in the user, system and combined domains. As only a very small percentage of the execution time is spent in the system mode we show only user (usr) and combined (all) results.

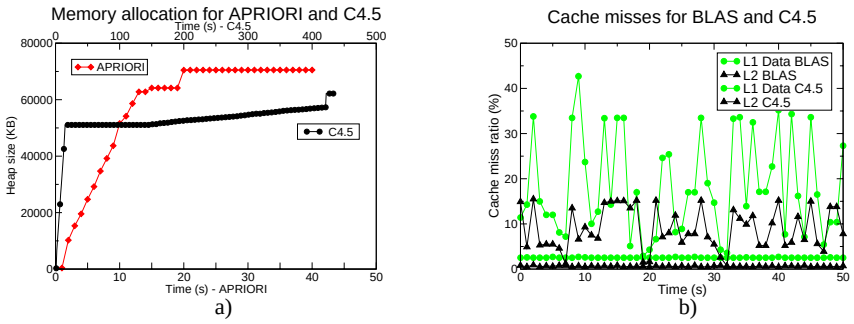

Fig. 1. (a) Dynamic Memory Allocation for APRIORI and C4.5 (b) Level-1 and Level-2 cache miss rate for BLAS and C4.5

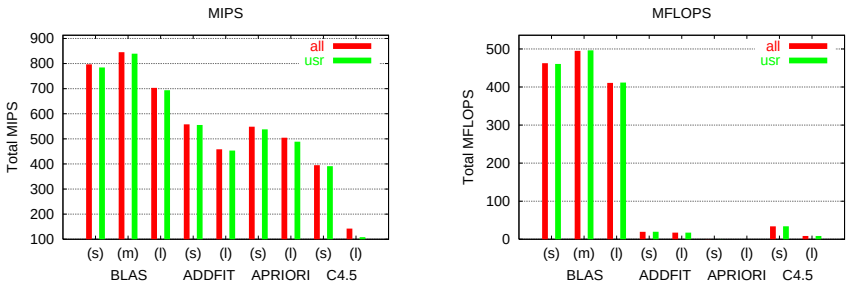

Fig. 2. MIPS and MFLOPS

4.1 Memory Allocation and Overall Performance

The maximal allocated memory for data (the heap size) is listed in Table 1. While *BLAS* and *ADDFIT* allocate all the memory they need (basically the dense matrices plus some buffers for input and temporary data) at the beginning, both *APRIORI* and *C4.5* dynamically allocate smaller memory blocks at run time. Figure 1a shows the heap sizes of *C4.5* and *APRIORI* as measured with the Unix command `pmap` for the larger test data sets. Two phases are clearly distinguishable, the first is loading the input data (steeper slope of the graphs), while the second is computing the decision tree or the frequent item sets (prefix tree), respectively. Such dynamic memory allocation and re-allocation results in many system calls and also prevents good data locality.

MIPS and MFLOPS are popular measures to show the overall performance mainly for scientific and engineering codes. In Figure 2 one can clearly see that only the *BLAS* dense matrix-matrix multiplication is actually dominated by floating-point operations. The three KDD programs do mainly integer and other operations, which is what one can expect from applications working on strings and integer numbers. It is also interesting to see that for all four applications the MIPS numbers decrease with larger input data sizes (except *BLAS* having a peak performance with a medium sized matrix).

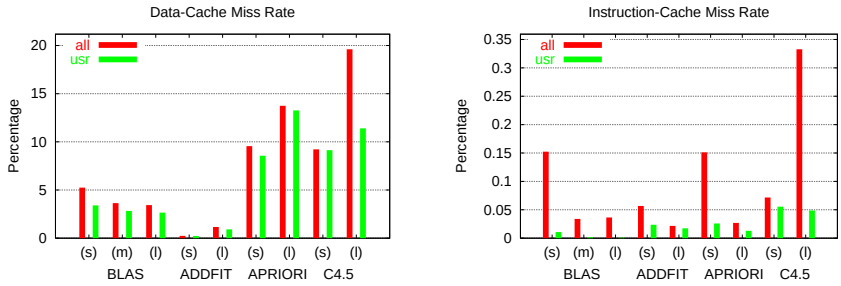

Fig. 3. Data- and Instruction-Cache miss rates

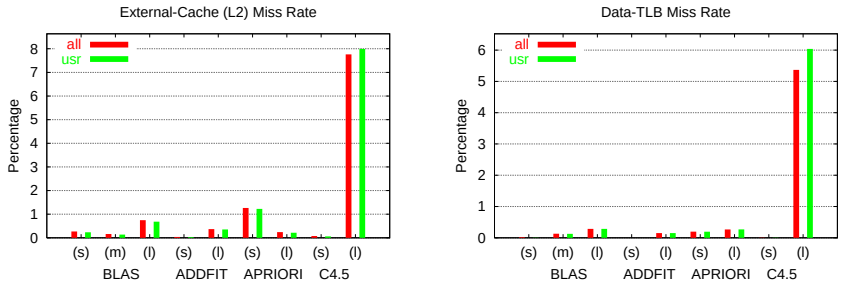

Fig. 4. Level-2 Cache and Data TLB miss rates

4.2 Cache Performance

Irregularity of the memory access patterns for the KDD applications is apparent when we compare Level-1 and Level-2 cache miss rates for *BLAS* and *C4.5* as presented in Figure 1b. For *BLAS* the average Level-1 and Level-2 miss rate is around 2.5% and 0.6% respectively and is maintained at such level through the duration of the computations. The same miss rates for *C4.5* exhibit very large fluctuations with some values for Level-1 cache misses being as high as 40%. This "fatal" cache utilisation is the major cause of such poor performance of all three KDD applications, presented in Figure 2, and *C4.5* in particular.

As can be seen form Figure 3 the data cache miss rate is approximately two orders of magnitude higher than the instruction cache miss rate. Small I-Cache miss ratio is typical for applications with low branch rate. The branch rate for *BLAS* is around 5 times less than for *C4.5* and this is clearly reflected in I-Cache miss rate measured in the user domain.

The highest data cache miss rates can be seen for *APRIORI* and *C4.5*. This is due to the erratic memory access patterns. While, for KDD applications, the data cache miss rate generally increases with larger input data sets, it remains nearly constant for *BLAS*.

The large discrepancies between user and combined (sys+usr) domains in I-Cache misses results, indicates, high I-Cache miss rate in the system domain.

Although, this has little bearing on the overall performance of applications being tested[5], we have established that infrequent system calls are the main reason for this.

The measurements for the Level-2 (external) cache as well as for the data translation look-aside buffer (TLB) in Figure 4 show again that the system Level-2 miss rate is much higher than the user miss rate. Interesting to see is the high rate both Level-2 as well as D-TLB misses for *C4.5* with the large data set. This has to be due to the sorting of entire categorical attributes (using recursive quick-sort) in *C4.5*, which results in almost no locality for data access. The instruction TLB miss rate (not shown here) is less than 0.02% for all applications.

5 Outlook

In this paper we presented first experiments of analysing KDD applications with hardware counters on an *UltraSPARC III* platform. To better understand the memory access characteristics of KDD applications further experiments with various data sets and other hardware event counters are needed. Running KDD applications on a machine simulator e.g., *SPARC Sulima* [9], will allow us to change parameters of the architecture and thus help to find bottlenecks in such applications. We are also planning to conduct similar experiments on a *Fujitsu Primepower SPARC server*. The *HAL SPARC* [18] processor in the *Primepower* allows simultaneous counting of more hardware events than the *SUN UltraSPARC-III* processor by dedicating one register to each event being counted.

6 Acknowledgement

This research is funded by the ANU/Fujitsu CAP program. The authors would like to thank Alistair Rendell for sharing his *Sun Blade 1000* with us.

References

1. R. Agrawal and R. Srikant, *Fast Algorithms for Mining Association Rules*, in Proceedings of the 20th VLDB Conference, Santiago, Chile, 1994.
2. L. Barroso, K. Gharachorloo and F. Bugnion, *Memory System Characterization of Commercial Workloads*, Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA-98), 1998.
3. R. Berrendorf and B. Mohr, *PCL – The Performance Counter Library: A Common Interface to Access Hardware Performance Counters on Microprocessors (Version 2.0)*, Research Centre Juelich, Central Institute for Applied Mathematics, September 2000.
   `http://www.kfa-juelich.de/zam/PCL/`

[5] The system instructions constitute only small portion of the overall program size with the exception of *C4.5* with the large data set. For details see Table 1.

4. J.P. Bradford and J. Fortes, *Performance and Memory-Access Characterization of Data Mining Applications*, Workshop on Workload Characterization, 1998. Workshop held in conjunction with the 31st Annual International Symposium on Microarchitecture.
5. J.P. Bradford and J. Fortes, *Characterization and Parallelization of Decision Tree Induction*, School of Electrical and Computer Engineering, Purdue University, 1999.
6. S. Browne, J. Dongarra, N. Garner, K. London and P. Mucci, *A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters*, Proceedings SC'2000, November 2000.
`http://icl.cs.utk.edu/projects/papi/`
7. P. Christen, M. Hegland, O.M. Nielsen, S. Roberts, P.E. Strazdins and I. Altas, *Scalable Parallel Algorithms for Surface Fitting and Data Mining*, Elsevier Journal of Parallel Computing, special issue on Aspects of Parallel Computing for Linear Systems and Associated Problems, September 2001.
8. P. Christen, O.M. Nielsen, M. Hegland and P.E. Strazdins, *Parallel Data Mining on a Beowulf Cluster*, Accepted by the HPC Asia 2001 Conference, Gold Coast, Queensland, Australia, September 2001.
9. B. Clarke, A. Czezowski and P. Strazdins, *Implementation Aspects of Sparc V9 Complete Machine Simulator* , to appear in ACSAC-2002, the Australasian Computer Systems Architecture Conference, Melbourne, Australia, January 2002.
10. R. Garg and I. Sharapov, *Techniques for Optimizing Applications*, SUN Blueprints, Sun Microsystems Press, 2002.
11. J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2000.
12. J.S. Kim, X. Qin and Y. Hsu, *Memory characterization of a parallel data mining workload*, in Workload Characterization: Methodology and Case Studies. Based on the First Workshop on Workload Characterization. IEEE Comput. Soc, Los Alamitos, CA, USA, 1999.
13. V. Pai. P. Ranganathan and S. Adve, *RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors*, In Proceedings of the Third Workshop on Computer Architecture Education, February 1997.
14. S. Parthasarathy, M.J. Zaki and W. Li, *Custom Memory Placement for Parallel Data Mining*, Technical Report 653, University of Rochester, Computer Science Department, 1997.
15. J.R. Quinlan, *Programs for Machine Learning*, Morgan Kaufmann, 1993.
16. M. Rosenblum, S.A. Herrod, E. Witchel and A. Gupta, *Complete Computer System Simulation: The SimOS Approach*, IEEE parallel and distributed technology: Systems and applications, vol. 3, no. 4, 1995.
17. P.E. Strazdins and J.G. Lewis, *An Efficient and Stable Method for Parallel Factorization of Dense Symmetric Indefinite Matrices*, in Proceedings of the 5th International Conference and Exhibition on High-Performance Computing in the Asia-Pacific Region (HPC Asia 2001), Gold Coast, September 2001.
18. W.W. Wilcke, *Architectural Overview of HaL Systems*, in Proceedings of the 40th IEEE Computer Society International Conference, San Francisco, USA, 1995.
19. X. Zhang, Z. Zhu and X. Du, *Analysis of Commercial Workload on SMP Multiprocessors*, Proceedings of Performance'99, August, 1999.