Performance Characterization of Data Mining Benchmarks

Vineeth Mekkat[⋆], Ragavendra Natarajan[⋆], Wei-Chung Hsu[†], Antonia Zhai[⋆]

[⋆]*Department of Computer Science & Engineering*
University of Minnesota - Twin Cities
Minneapolis, MN 55455
{*mekkat,natar,zhai*}*@cs.umn.edu*

[†] *Department of Computer Science*
National Chiao Tung University
Hsinchu, Taiwan
hsu@cs.nctu.edu.tw

ABSTRACT

Explosive growth in the availability of various kinds of data in both commercial and scientific domains have resulted in an unprecedented need to develop novel data-driven, knowledge discovery techniques. Data mining is one such data-centric application. It consists of methods to discover interesting, nontrivial, and useful patterns hidden within massive amounts of data. Researchers from both academia and industry have recognized that the challenges of data mining applications will help shape the future of multi-core processor and parallelizing compiler designs. However, relatively little has been done to understand the performance characteristics of these applications on modern multi-core processors.

The exponential growth of on-chip resources make it critical to exploit parallelism at all granularities for improving the performance of data mining applications. In this paper, we examine the instruction-level, memory-level and thread-level parallelism available in data mining applications. We observe that (i) data mining applications have a slightly different instruction mix from SPEC integer applications, and this difference can potentially lead to different ILP extraction; ii) although many data mining applications suffer from data cache miss penalty, similar to SPEC integer applications, different techniques must be developed to enable effective prefetching due to the existance of complex and irregular data structures, such as hash tables; (iii) although data mining applications have large amount of thread-level parallelism, efficient extraction of such parallelism depends on on-chip cache performance; and (iv) the performance characteristics of data mining applications can vary at runtime, and thus techniques that dynamically tune the applications to adapt to such variations are desired.

Categories and Subject Descriptors: C.1.0 [Processor Architectures]: General
General Terms: Thread-level parallelism, Instruction-level parallelism, Cache Performance, Multi-core
Keywords: Data Mining, Performance Characterization, Parelleliza-tion

1. INTRODUCTION

The phenomenal growth of computer technologies over much of the past five decades has been fueled by the enormous demand of scientific applications for raw computing power. However, the current generation of modern applications, both commercial and scientific, is data-centric. Explosive growth in the availability of various kinds of data in both commercial and scientific domains has resulted in an unprecedented need to develop novel data-driven, knowledge discovery techniques. Data mining is one such data-centric application. It consists of methods to discover interesting, nontrivial, and useful patterns hidden within massive amounts of data [13, 32]. Recently, the rate of performance improvement of data mining applications is much slower than the rate in which the amount of data collected is increasing [23], thus creating an urgent need for building efficient data mining applications. Researchers from both academia and industry have recognized that the challenges of data mining applications will help shape the future of multi-core processor and parallelizing compiler designs. For instance, Intel [9] describes *recognition, mining* and *synthesis* as the three key components of future computation infrastructures that handle Tera bytes of data. In addition, many data mining applications use dynamic programming, as well as backtrack/branch+bound algorithms in their computation kernel, and these algorithms have been identified as two of the six key challenges in future parallel computation systems by Asanovic *et al* [5].

There have been numerous studies on improving the performance of data mining applications. Much of these efforts focus on characterizing the performance of specific applications on specific workloads [6, 7, 14]; or on optimizing specific data mining algorithms [16, 34, 24, 35, 12, 10, 30, 36, 17, 31, 28]. However, relatively little has been done to characterize the performance of diverse data mining applications on different workloads to examine whether existing architecture features and compilation optimization technologies are suitable for emerging data mining applications; and how to design future microprocessors that can be efficiently utilized by this emerging workloads.

Recently, as the exponential growth of clock frequency came to a halt, microprocessor designers face the challenges of upholding the exponential performance improvement without an increasing clock rate, thus exploiting parallelism at all granularities becomes increasingly critical. Intuitively, data mining applications exhibits parallelism of different forms, including ILP (instruction-level parallelism), DLP (data-level parallelism, e.g. SSE on IA32), TLP (thread-level parallelism) and MLP (memory-level parallelism). However, although multi-core environment, with large amount of resources and efficient inter-thread communication capability, seems to be the ideal environment for extracting such parallelism, the complex interaction between threads and limited off-chip bandwidth complicates the parallelism extraction. In this paper, we study the performance characteristics of data mining applications on existing modern multi-core systems using the NU-MineBench [22] benchmark suite. With respect to this data mining benchmark suite we make the following observations:

- Comparing the sequential performance of SPEC Integer and NU-MineBench applications reveals that the average IPCs of these two benchmark suites are comparable. However, data mining applications show potentials for more ILP since there are fewer store instruction and branch instruction; and these instructions often hinders the extraction of ILP.

- Comparing the memory performance of the SPEC Integer and NU-MineBench applications reveals that improving cache performance is key for both benchmark suites. Of the thirteen benchmarks examined, in NU-MineBench, nine of them spent more than 40% of the CPU cycles stalling for cache misses. Interestingly, we find that the large input sets are not the source of the poor memory performance as accesses to input sets have spatial locality. However, these applications often construct large and irregular auxiliary data structures, such as hash tables; and accesses to these data structures exhibit poor spatial and temporal locality, leading to cache misses. Although the compiler inserts prefetch instructions in these programs, these instructions are often ineffective and in some cases detrimental.

- While quite a few applications have large amount of TLP, not all applications demonstrated linear scalability on multi-core processors. The performance of the last-level cache (LLC) often have significant impact on the scalability . However, different applications have contradicting requirements on the organization of LLC as shared or private.

- The performance characteristics of many data mining applications exhibit variation at runtime. We have not only observed phase behaviors, where the application behave differently as they enter different functions; but also performance variation based on the characteristics of the input data. More interestingly, we have observed that the performance characteristics of the same code segment can change over time as the application making progress in mining the input data.

It is also worth pointing out that, there also exists a large amount of DLP in data mining applications. For example, the dot product operation over large vectors, seen in many applications, can potentially benefit from hardware support for vector operations. However, in this paper, we will not provide in depth discussion on DLP.

Rest of the paper is organized as follows: Section 2 discusses related work; Section 3 details our experimental infrastructure and briefly describes the various data mining algorithms; Section 4 compares SPEC and NU-MineBench from the perspectives of ILP; Section 5 analyzes cache performance of NU-MineBench to examine the availability of MLP; Section 6 presents the scalability of parallel data mining applications and discuss whether TLP in data mining applications can be effectively extracted. We present dynamic performance characteristic changes in NU-MineBench applications in Section 7. Finally, we present our conclusions in Section 8.

2. RELATED WORK

There have been a number of studies in characterizing data mining applications. Bradford and Fortes [6] analyzed performance and memory-access behavior of *decision tree induction* based data mining algorithms. Ghoting *et al* [11] analyzed *frequent itemset mining* and *clustering* algorithms for their performance and memory behaviors. Chen *et al* [8] conducted a performance scalability study for data mining workloads, but was limited to bioinformatics applications.

Most of the previous works either focus on one particular class of data mining applications like *classification, clustering* or is limited to one particular characteristic like memory hierarchy or scalability. In this paper, we provide a comprehensive study of data mining benchmarks from five different categories. In addition to memory hierarchy and scalability characteristics, we also discuss the instruction-level parallelism and dynamic(run-time) behaviors of these applications.

Liu *et al* [20] introduced and performed the initial analysis for MineBench, the predecessor to NU-MineBench. The analysis focused on the I/O overhead and synchronization costs. Berkin *et al* [23] show that data mining applications have distinct behaviors in comparison to SPEC [29], MediaBench [18] and TPC-H [4], but they provide relatively little details on the underlying architecture support.

Shaw [26], Li *et al* [19] present a detailed analysis of the working sets of data mining applications. They show that increasing cache size alone cannot improve performance and highlight the benefits of simple hardware prefetching. Our study analyzes the complex auxiliary data structures created by the data mining applications, their irregular access and observe that simple prefetching mechanisms will not be sufficient in this scenario. We point out the need for more sophisticated prefetching mechanisms. Jibaja and Shaw [15] analyze the applicability of existing CMP approaches to data mining applications. They show that the applications exhibit different levels of data sharing and this calls for flexible cache configurations. In this paper we confirmed these results on two off-the-shelf CMP processors with different last-level cache organizations.

There have been numerous efforts on adapting data mining algorithms to parallel platforms. These efforts have included parallelized algorithms for clustering, classification, and association rule mining; mostly for shared and distributed memory architectures. Examples include: Joshi *et al* [16], Zaki *et al* [34, 35], Parthasarathy *et al* [24], Han *et al* [12], Foti *et al* [10] and Stoffel *et al* [30]. In this paper, we study the scalability performance of data mining applications on shared memory multi-core processor systems, which is increasingly becoming the *defacto* standard for modern multi-processor systems.

3. EVALUATION INFRASTRUCTURE

We study the performance characteristics of data mining applications using the NU-MineBench [22] benchmark suite. We evaluated thirteen out of seventeen benchmarks in this suite, omitting BAYESIAN, BIRCH, SNP and GENENET due to difficulties encountered in compilation. All benchmarks are written in `C/C++`, and all, with the exception of AFI, GETI and ECLAT, are parallelized with OpenMP directives. The benchmarks are compiled with `-O3` using the Intel C/C++ compiler version 11.0. Taking advantage of the extensive hardware performance counters available on Itanium processors, these applications are evaluated on a 8-core Itanium-based CMP with dual-core Intel Itanium-2 [1] processors running at 1.6GHz. Each of the 8 cores have a 16KB L1I, a 16KB L1D cache, a 1MB L2I, a 256KB L2D cache and a 9MB unified private last-level cache. To compare the parallel performance of multi-core processors with shared and private last-level on-chip cache, we also evaluate the parallel execution of these applications on a 8-core x86-based CMP with quad-core Intel Xeon [2] processors running at 2.66 GHz. The x86 based machine has two processors, each containing two dual-core dies, and cores in the same die share a 4MB L2 cache. Each core has 32KB L1 cache.

The PerfMon [25] library is used to manage the hardware performance counters. Stalls in the Itanium pipeline back-end are divided into five mutually exclusive categories. Most of the stalls caused by execution units waiting for operands are due to cache misses. We refer to them as `Cache` stalls in our figures. Stalls due to recirculation of data access requests from L1D due to TLB or L2D OzQ being full is referred to here as `Recirculation` stalls. `Flush` corresponds stalls caused by pipeline flushes. In our case, almost all such stalls are caused by branch mis-predictions. `RSE` corresponds to stalls caused by the Register Stack Engine, which is negligible in all applications. `FE` corresponds to stalls caused by the pipeline front end. `Pin` [21], a binary instrumentation tool, is used to collect dynamic instruction profile.

The data mining applications we analyzed are divided into six categories: association rule mining, classification, clustering, optimization, structure learning, and error-tolerant itemset mining.

Association Rule Mining applications take a database of item transactions as input, and identify frequently-occurring itemsets in the transactions. Among the benchmarks evaluated, APRIORI, UTILITY MINE and ECLAT belong to this class. APRIORI identifies all frequently occurring itemsets based on the *apriori* principle. Candidate itemsets of length k are generated from itemsets of length k-1. Frequency of candidate itemsets are calculated and the ones which are infrequent are then pruned to generate frequent itemsets of length k. The implementation uses a hash tree data structure to store the generated candidate itemsets. UTILITY MINE algorithm is similar to APRIORI except that now each itemset is also given a ”utility” value and those with higher utility are identified. ECLAT represents transactions by vertical transaction ID lists and a candidate k itemset is generated by intersecting two (k-1)-itemsets that have a common prefix. The program partitions the search space into small manageable chunks thereby reducing the pressure on the memory system.

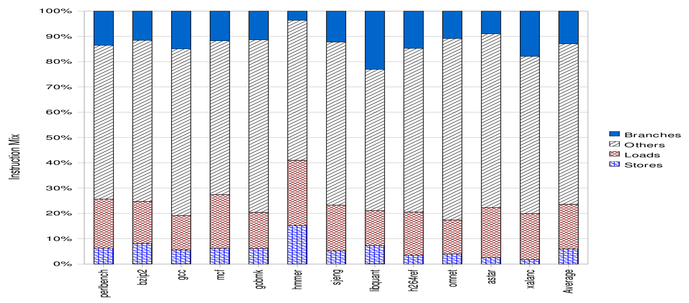
Classification workloads take as input a training dataset which is used to develop a predictive model which is then used to classify unlabelled records. SCALPARC is a parallel and scalable version of the decision tree classification algorithm. The data is classified by recursively splitting the records on the attribute which leads to the best split. The process is continued until all records in a partition fall under the same class. The implementation builds a tree based decision model and uses a hash table to make the computation more efficient. RSEARCH is a RNA sequencing program which finds homologous RNA sequences by searching a gene database. The RNA sequence being searched for is built using a context-free grammar and a local alignment algorithm is used to find homologous RNAs in the database. Support Vector Machines- Recursive Feature Elimination (SVM-RFE) classifies records by selecting specific features. Records are eliminated recursively from a set of active variables based on some support criteria.

Clustering aims at discovering groups of similar objects from a database to expose the underlying data distribution. K-MEANS divides the input data by assigning them to their closest cluster center. The kernel of the program is the Euclid distance computation function. The K-means algorithm assigns a point in the database to a single cluster. But FUZZY K-MEANS relaxes the condition by allowing a point to be a member of more than one cluster. The third program in this category is HOP, a density based clustering algorithm. The algorithm forms clusters by assigning particles to its densest neighbor.

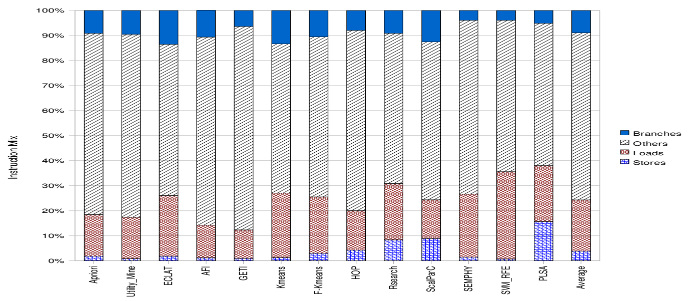
Error Tolerant Itemset Mining finds sets of items where, informally, most of the items commonly occur together. These algorithms are useful for finding patterns that are a relaxation of frequent itemsets or for finding frequent itemsets in noisy data. This category includes AFI and GETI. The algorithms used in these programs are similar to the ones used in association rule mining.

SEMPHY is the only **Structure Learning** workload evaluated by us. It is used to find species that are genetically related. Phylogenetic trees are used to represent genetic relationships in species, where closely related animals are in nearby branches. SEMPHY searches for maximum likelihood phylogenetic trees using the structural expectation maximization algorithm.

Optimization programs identify similar regions in DNA, RNA and protein sequences. PLSA, the only benchmark in this category, uses the dynamic programming paradigm based on the Smith and Waterman algorithm to find similar regions between two sequences.



(a) SPEC CPU2006 Integer Benchmarks



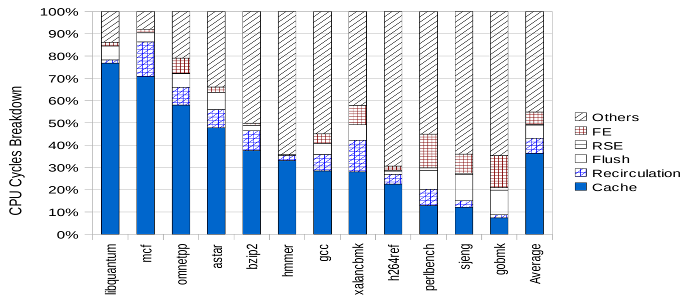
(b) NU-MineBench Programs

Figure 1: Dynamic instruction mix comparison.

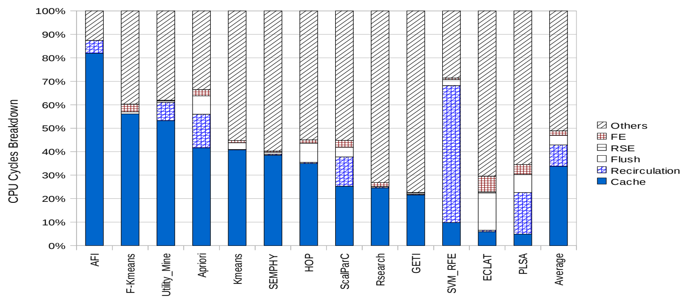
4. PERFORMANCE CHARACTERISTICS: SPEC VS NU-MINEBENCH

For years, the SPEC CPU [29] has been widely accepted by computer architects as the benchmark for measuring CPU performance. Many architectural features and compiler optimization techniques that aim to improve instruction-level parallelism (ILP) are designed with the performance characteristics of SPEC CPU in mind. In this section, we analyze the differences between the SPEC Integer [29] benchmarks with data mining benchmarks [22] in terms of dynamic instruction mix, cach misses, effect of software prefetching and execution stall cycle breakdown. NU-MineBench programs are mostly control intensive like SPEC Integer as hence we compare them, rather than with SPEC FP which is loop intensive.

Figure 1 shows the dynamic instruction mix of the SPEC Integer benchmark suite and NU-MineBench benchmark suite. These two benchmark suites show similar trends, but with the following distinctions: SPEC Integer applications have 54% more store instructions than NU-MineBench programs, but have 16% fewer load instructions. Data mining applications typically use the input data to build temporary data structures. These data structures are then traversed many times with a small number of updates. This access pattern leads to more loads but fewer stores. SPEC programs also

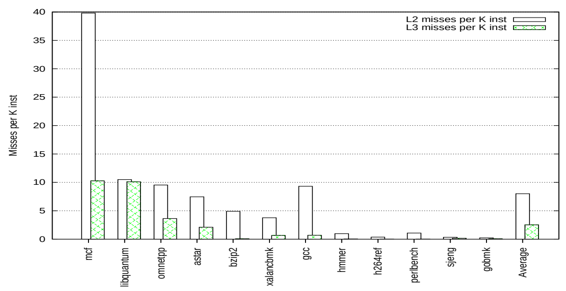


(a) SPEC CPU2006 Integer Benchmarks

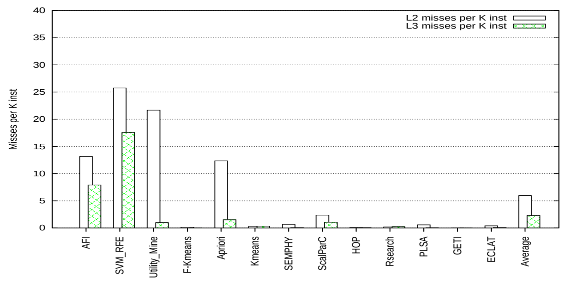


(b) NU-MineBench Programs

Figure 2: CPU Cycles breakdown.



(a) SPEC CPU2006 Integer Benchmarks



(b) NU-MineBench Programs

Figure 3: Cache Miss Rate comparison.

have 45% more branch instructions as compared to NU-MineBench applications.

Figure 2 shows that both SPEC Integer and NU-MineBench have similar characteristics on the average stall cycle distribution, despite the differences in instruction mix and in cache miss rate. Nine out of the thirteen data mining programs spend more than 40% time stalling on cache misses. We discuss the memory behavior of data mining workloads in detail in Section 5.

The number of L2 and L3 misses per thousand instructions for SPEC and NU-MineBench programs are summarized in Figure 3. The average number of cache misses per thousand instructions for the two benchmark suites are similar: L2 cache misses per thousand instructions is 34% higher for SPEC Integer; and L3 cache miss per thousand instructions is same for both benchmark suites. In NU-MineBench, four benchmarks have a large number of L2 cache miss.

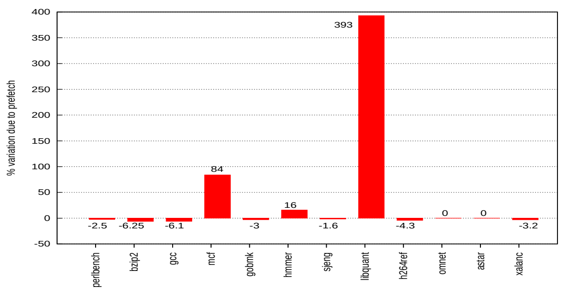
Although many data mining applications process large input sets, access to input data typically exhibits good locality and does not cause large number of cache misses. Data mining benchmarks generate large, complex auxiliary data structures such as hash trees. Accesses to these structures are very irregular and difficult to prefetch and cause frequent cache misses.

4.1 Discussion

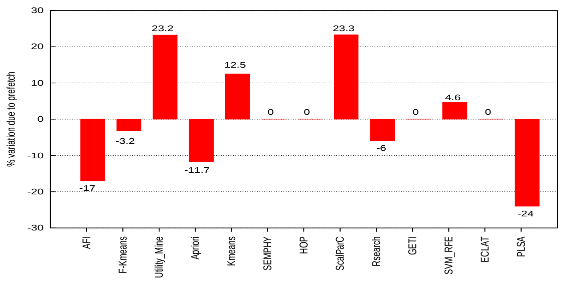
We compare NU-MineBench to SPEC Integer benchmarks since both applications contain branch intensive codes and have a large number of irregular memory accesses. Although there is a difference in the dynamic instruction mix between the suites, the cache miss rates and stall distributions are similar. Hence existing processors built with SPEC workloads in mind should be able to perform well on data mining workloads. Data mining applications have fewer store instructions than SPEC programs, and hence there are more instruction scheduling opportunities resulting in more ILP that can be exploited. Therefore, existing processors built to exploit ILP in SPEC CPU benchmarks should be able to exploit more ILP from data mining applications. But there are some key differences in data mining applications that present challenges and novel opportunities in processor design. Processors built for data mining applications can have smaller load-store queues than current generation processors, since there are fewer store instructions to commit in data mining applications. This can potentially improve the power efficiency of processors as load-store queues typically consume a lot of power. Processors can also afford to have smaller write buffers for data mining applications for the same reason. Data mining applications generate and work with temporary data structures which are deleted during the course of the program. Hence any writes made to them need not be committed to memory as it will be useless. Identifying such writes and not committing to memory would increase memory performance significantly. By eliminating unnecessary writes, fewer instructions will be executed leading to faster execution times.

5. MEMORY HIERARCHY PERFORMANCE

Improving cache performance is key for many data mining applications, since nine out of the thirteen benchmarks we examined spent over 40% of total execution cycles stalling as a result of cache misses. Our analysis reveals that, although many data mining applications process large input sets, accesses to the input data usually exhibit spatial locality and do not cause cache misses. However, these applications often construct large and irregular auxiliary data structures, such as hash trees [32]. Accesses to these data structures exhibit poor locality due to the use of indirect accesses and pointers. Although software-controlled prefetch can potentially reduce cache misses by fetching data into cache before it is used, for many data mining applications, either no prefetch instructions are generated by the compiler or the compiler-inserted prefetch instructions are ineffective, even with aggressive optimizations. This is not only because the underlying data structure is complex and pointer-based, but also because accesses to these data structures are embedded in complex control flow.



(a) SPEC CPU2006 Integer Benchmarks



(b) NU-MineBench Programs

Figure 4: Effect of compiler inserted prefetch instructions.

Figure 4 shows the impact of compiler-inserted prefetch instructions on NU-MineBench and SPEC Integer applications. The graph shows the percentage change in total execution cycles for executions compiled with compiler inserted prefetch instructions over the executions without. In SPEC, a few applications, such as MCF, LIBQUANTUM and HMMER, benefit significantly from compiler-inserted prefetch. while others are not significantly affected. In NU-MineBench, on the other hand, the effect of compiler inserted prefetch instructions is more dramatic. While some applications, such as UTILITY-MINE, K-MEANS and SCALPARC, benefit significantly from prefetch instructions, many suffer significant performance degradation. These variations indicate that there is a need for better static and dynamic optimization techniques for generating compiler-inserted prefetches. In the rest of this section, we provide a case-by-case analysis of the cache performance of each application.

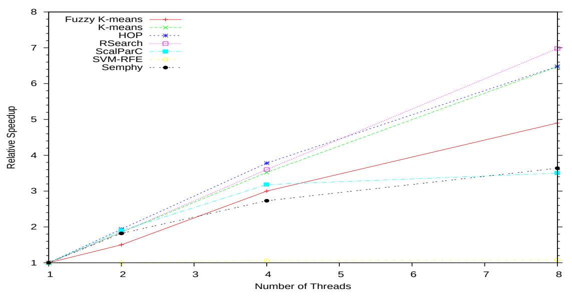
SCALPARC, a classification benchmark, stalls on 57% of CPU cycles due to cache misses. The program has a memory footprint that is twice its input data size to store a model based on the training data. The model is constructed as a large, complex tree-based data structure. Model construction is where most of the time is spent as classifying the input dataset is not computationally intensive. During the model building phase, the program builds lists for each attribute of the input data sorted by the attribute value. At each node of the decision tree, a hash table is used to partition these attribute lists between the child nodes. The hash table is similar in size with the number of records, containing millions of entries, and does not fit in the cache. By examining the addresses accessed by the program it is clear that accesses to the hash table are very irregular. Since the hash table is very large, and accesses to it are highly irregular, it leads to frequent cache misses. When the program is parallelized using MPI, each processor gets a part of the hash table which fits in the cache, but when OpenMP is used to parallelize the program all threads read from a single copy of the hash table which does not fit in cache leading to poor memory performance. The compiler does not insert prefetch instructions for this program as the element of the hash table accessed cannot be predicted.

SVM-RFE, another classification benchmark, shows the highest stall percentage. It stalls for 69% of total CPU Cycles, which is almost entirely due to recirculation stalls. The algorithm represents the data set as points in an n-dimensional plane and splits the data points into different classes by searching for an optimal splitting hyper-plane. The optimization kernel makes use of Math Kernel Library, to compute vector dot product, where each vector is a one dimensional floating point array representing the coordinates of the point. The program stalls when loading data from the vectors for dot product computation. The assembly code generated for the library, revealed that the prefetch instructions generated by the compiler were not very effective.

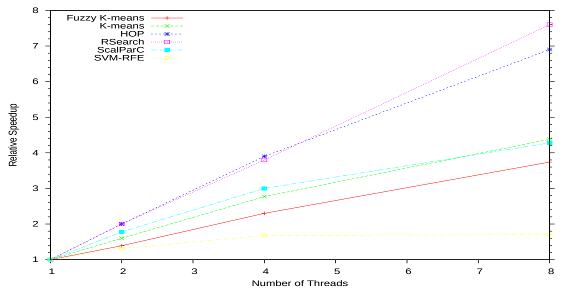
APRIORI and UTILITY MINE are association rule mining programs which have similar behavior. APRIORI stalls for 66% of total cycles, the majority of which are caused by cache misses which account for 53% of total CPU cycles. UTILITY MINE stalls for 62% of the total cycles, almost all of which are cause due to cache misses. The programs try to find items that occur together frequently in a large group of item transactions. Both programs use a hash tree to store subsets of the input transactions. The tree is then traversed to count the number of occurrences of each subset in the transaction data. However, the hash tree is too large to fit into the cache and exhibits poor locality. Hence, repeated accesses of the hash tree data structure generate a large number of cache misses. Also, because of the complex control flow in the *hot-spot*, compiler does not insert prefetch instructions which further degrades the performance.

AFI, an error tolerant itemset mining algorithm, is very similar to Apriori and finds frequent itemsets in noisy data. The program has a load instruction embedded in a double nested loop, which causes a large number of cache misses. The static compiler inserted a prefetch instruction in the inner loop to hide this latency. However, the inner loop has a small iteration count; and thus this prefetch is not only ineffective, but actually causes performance degradation. Moving the prefetch instruction to the outer loop improves the performance of this application by more than 30%. It is difficult for the compiler to identify this optimization opportunity since the outer loop contains conditional branches.

HOP, a clustering algorithm stalls for 45% of total CPU cycles, of which cache misses account for 35% of the CPU cycles. In this application, the instructions were scheduled poorly which could not effectively hide the latency of load instructions. The compiler failed to unroll certain loops which were inside control statements. Since the compiler was not sure if the loop would be executed at run time it was conservative and did not unroll these loops. By manually unrolling them, the performance of the program improved by 14%. K-means and Fuzzy K-means, both clustering applications, also stalled for 45% and 60% of total CPU cycles respectively. Cache miss penalty accounted for 41% of total cycles in K-means, while it accounted for 56% of total cycles in Fuzzy K-means.



(a) 8-core Itanium system



(b) 8-core x86 system

Figure 5: Scalability of NU-MineBench on multi-core systems.

PLSA, a gene sequence alignment algorithm, contains compiler-inserted prefetch instructions, but removing these prefetch instructions actually improves performance by 24%. In this application, the compiler identified linear accesses to large array-based data structures and decided to insert prefetching instructions. However, due to complex branch behaviors, at runtime the application only accesses a small fraction of the array. Thus most of the prefetched data is useless, and the associated overhead degrades program performance. These instructions would have been effective, if the program traversed through a large portion of the array. Unfortunately, the compiler is unable to determine the correct access pattern and make the proper decisions statically.

6. EXTRACTING THREAD-LEVEL PARALLELISM ON MULTI-CORE PROCESSORS

Prior works [16] have shown that data mining applications are able to scale to 128 processing element on distributed-memory multiprocessor systems using MPI. For applications in the NU-MineBench suite, ten out of the thirteen benchmarks that we have evaluated have been explicitly parallelized using OpenMP directives [23], and most of these applications are able to scale linearly on shared-memory multi-processors systems with up to 8 processors. While it is clear that these applications exhibits high level of TLP, it is not clear whether such parallelism can be extracted on multi-core processors that are becoming increasingly common. While multi-core processors have abundant resources and efficient inter-thread communication, we must face the following challenges when extracting parallelism on multi-core processors: (i) sharing of on-chip resources may cause contention among threads. For example, shared caches may lead to complex thrashing/prefetching behaviors; (ii) limited *off-chip bandwidth* may become the new performance bottleneck. While the computing power of modern multi-core processors increases dramatically with each generation of technology, the increase in off-chip bandwidth cannot scale at the same rate. This constraint may become the new performance bottleneck.

We studied the parallel performance of seven out of ten benchmarks that have OpenMP directives in the benchmark suite. APRIORI, UTILITY MINE and PLSA are omitted due to compilation errors. Our evaluation is conducted on two distinct CMP machines as described in Section 3. The implementation of SVM-RFE in NU-MineBench is overly synchronized, thus is unable to scale at all. However, previous work [8] has shown that other implementations of SVM-RFE can potentially scale better. Thus, we exclude this benchmark from further discussions. SEMPHY scales linearly only up to 2 threads and shows small performance improvement with 4 and 8 threads. Close examination reveals that SEMPHY has unbalanced workload between the threads. In many segments of execution, only one thread is active. The complete results are available in the Figure 5.

We use two platforms to evaluate scalability, as described in Section 3. The most notable difference between these two architectures is the organization of the LLC: Itanium-based multi-core processor has private on-chip LLC, while on the Intel Xeon-based multi-core processor, two cores that are located on the same die share the same LLC, but cores on different chips do not share cache.

In terms of scalability, the two architectures show similar trends: HOP and RSEARCH are able to scale linearly upto 8 threads; FUZZY K-MEANS, SCALPARC and SEMPHY are able to scale only up to 4 threads; K-MEANS is able to scale linearly upto 8 cores only on the Itanium-based system. For SCALPARC the scalability issue is mainly affected by cache performance. On the 8-core Itanium-based system, as the number of threads increases, the percentage of total execution cycle stalls due to cache miss also increases significantly. When number of threads increases from four to eight, no performance improvement is observed, but stall cycles due to cache miss penalty increases by 50%. When the number of threads changes from 1 to 2 to 4, and finally to 8, the total percentage of stall cycles increases from 46% to 48% to 54% and to 74%. The major contributor to this is the EXE stall cycles, mainly due to cache misses, which increases from 27% to 27% to 31% and finally to 48%. Stalls due to recirculation also contribute to stall cycle increase, but is less significant. It only increases from 13% to 13% to 15% and finally to 20%. The most significant increase occurs when the number of threads increases from four to eight. The number of cache misses also increases correspondingly. While not scaling linearly, SCALPARC is able to scale better on the Intel Xeon-based multi-core system. In particular, the number of cache misses stays the same even as the number of threads increases. K-MEANS, on the other hand, shows the opposite behavior as SCALPARC. On the Itanium-based multi-core system, when the number of threads increases from four to eight, the cache miss rate decreases by 36%. However, on the Intel Xeon-based multi-core system, the cache miss rate remains constant.

The key difference between SCALPARC and K-MEANS is that there is a large amount of shared data between threads in SCALPARC, but very little in K-MEANS. Therefore, K-MEANS is able to scale better on systems with private last-level cache; while SCALPARC is able to scale better on systems with shared last-level cache. For K-MEANS, on shared LLC, data brought in by one thread can potentially over-write data brought in by other threads, but cannot be used by other threads; on private LLC, data brought in by one thread does not interfere with data brought in by another thread. For SCALPARC, on private LLC, data brought in by one thread cannot be used by other threads; thus shared data must be brought into the cache multiple times causing the cache miss rate to increase. On shared LLC, shared data brought in by one thread can be used by others, thus cache miss rate does not increase with the number of threads.

7. DYNAMIC PERFORMANCE CHARACTERISTICS VARIATIONS

The results presented in this paper so far make two simplifications. First, it is assumed that the behaviors of the program stay the same throughout the entire duration of the execution, thus we present average performance measurement. Second, it is assumed that the performance characteristics of data mining remain the same over all input sets, thus we only present the performance characteristics for one input set. While these two simplifications do not affect the accuracy of the results for most applications, we have observed that the performance characteristics change for some applications as (i) the program enters different code segments; (ii) different input sets are used; and (iii) the same code segment make progress in mining the data.

Phases behaviors [27], often refer to the fact that the program behaviors change as the program enters different segments of code, have been reported in SPEC Integer applications.

While phase behavior is common in both SPEC and NU-MineBench, data mining applications demonstrated additional dynamic performance characteristics variations that we will be focusing for the rest of this section.

7.1 Input Dependent Performance Variations

We refer to the fact that some applications behave differently when different input sets are given as input sensitive behaviors. Association rule mining applications demonstrated such input sensitivity. We evaluated the performance characteristics of APRIORI with three different input sets: (i) the input data set released with NU-MineBench [22]; (ii) anonymized traffic accident data; and (iii) data generated from the IBM Quest data generator [3]. The latter two input sets are from the University of Helsinki [33]. We have not only observed different code segments emerge as the performance hot spots, but also changes in the stall cycle breakdown as the input sets change. The total percentage of execution time stalling, for the three input sets are 67%, 50% and 88%, respectively. A detailed examination of the breakdowns of stall cycles reveals even more differences: stalls due to cache misses correspond to 42%, 16% and 79% of total cycles, recirculation stalls correspond to 14%, 22% and 5% of total cycles, and stalls due to branch misprediction correspond to 8%, 10% and 2% of total cycles. One reason for such performance variation is the fact that different segments of code emerge as the *hot-spots* with different input sets. APRIORI has two phases of execution: candidate generation and support counting. For the first two input sets, support counting for the candidate itemsets is the *hot-spot*, whereas, for the third input set, candidate itemset generation is the *hot-spot*.

These performance characteristic variations are not simply related to the input set sizes, rather, other factors like number of items in the input set, average transaction size, number of transactions etc., also have significant influence. For instance, input sets with more items tend to generate more candidate patterns, especially in the initial stages of the algorithm where itemsets of relatively small size are being evaluated. Input set characteristics are not the only input parameter that influence the behavior of the program. The *support* level in association rule mining applications defines the minimum number of times an itemset has to occur in the set of transactions to be considered frequent. The support level also influences the program behavior. In APRIORI, at higher support values, branch misprediction becomes a significant issue, contributing to more than 25% of stalls, whereas at lower support value, although the number of stalls due to branch misprediction remain the same, cache miss stalls become a more dominant bottleneck. The number of itemsets generated by the algorithm increases drastically at low support values, increasing the size of the hash tree used to store the itemsets. Hence with lowering support the hash tree generated by the program can no longer fit in the cache which increases the number of cache misses.

More interestingly, we observe that some compiler optimizations also behave differently as the input set changes. In APRIORI, prefetch instructions are inserted by the compiler in the support counting routine. However, these prefetch instructions are only effective for data set 1, but rather degrade performance for data set 2. These types of behaviors suggest that there may not exist a single optimal compilation strategy, rather dynamic optimizers that observe the behaviors of the application at runtime and re-optimize the code can potentially outperform static ones.

7.2 Time Dependent Performance Variations

Another form of dynamic performance variation is observed in data mining applications: the performance characteristics of the same code segment changes over time while operating on the same input set. APRIORI, for example, exhibits time dependent performance variation, but only on one of the three input sets, the input set released as part of the NU-MineBench benchmark suite. The dynamic execution cycle breakdown on this input set is shown in Figure 6. While the program executes the same code segment, in the first segment of the execution, LLC miss penalty is the key performance bottleneck, but during later segment of execution, recirculation stalls become the new performance bottleneck. APRIORI searches frequent itemsets in an input set, and the lengths of the searched itemsets increase as the application makes progress. The algorithm is able to eliminate some possible candidate itemsets, and eventually the search space fits in the LLC and cache miss penalty ceases to be the bottleneck.

8. CONCLUSIONS

Increasing emphasis on novel data-driven, knowledge discovery techniques have called for a re-evaluation of the architectural features and software optimization techniques employed in current computing systems. In this paper, we conduct a thorough performance analysis of NU-MineBench, a data mining benchmark suite that includes applications from several important categories of data mining such clustering, classification, association rule mining etc. We characterize the performance characteristics in terms of different forms of parallelism, such as ILP, MLP and TLP and found the following:

- the IPC of data mining applications is comparable to SPEC Integer applications. However, data mining applications show potential for more ILP since with fewer store and branch instructions, data mining applications have more potential for instruction scheduling.

- extracting memory-level parallelism is key for improving the performance of data mining applications, since nine out of the thirteen benchmarks examined spent more than 40% of the CPU cycles stalling for cache misses. However, data mining applications build large and irregular auxiliary data structures, such as hash tables to store runtime information; and existing prefetching techniques are inadequate with such data structures. Thus, developing more sophisticated prefetching techniques targeting such auxiliary data structures is key for improving MLP.

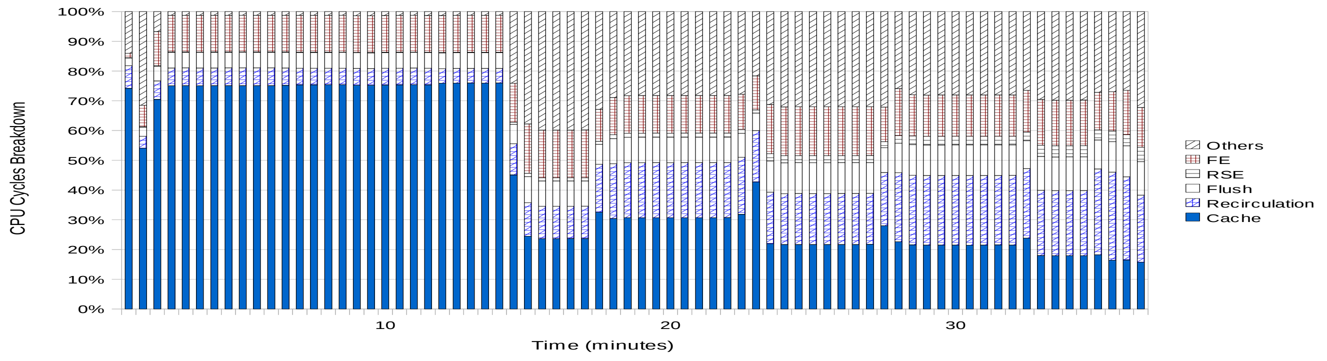


Figure 6: Run-Time Profile of APRIORI. Each bar represents the cycle breakdown for a 30-second interval.

- While quite a few applications have large amount of TLP, not all applications demonstrated linear scalability on multi-core processors. Our experiments suggest that the performance of the last-level on-chip cache is key. However, different applications have contradicting requirements for the organization of the LLC: applications with shared data prefer shared LLC, while applications with little shared data prefer private LLC. Our observation suggests that dynamic cache partition techniques may be desirable.

- Several different types of performance characteristics variation exist: the performance characteristics of an application vary as the program enters different code segments and as the input set changes. Furthermore, performance characteristics of the same code segment can also change over time. More interesting, the effectiveness of some compiler optimizations change as the performance characteristics vary. Thus, dynamic optimizers, that observe application performance characteristics and re-optimize the application to adapt to such variation are desirable.

9. REFERENCES

[1] Intel Itanium-2 processor. http://www.intel.com/products/processor/itanium/index.htm.

[2] Intel Xeon processor. http://ark.intel.com/Product.aspx?id=28035.

[3] Quest synthetic data generation code. http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html.

[4] TPC-H benchmark revision 2.0.0. http://www.tpc.org.

[5] ASANOVIC, K., BODIK, R., CATANZARO, B. C., GEBIS, J. J., HUSBANDS, P., KEUTZER, K., PATTERSON, D. A., PLISHKER, W. L., SHALF, J., WILLIAMS, S. W., AND YELICK, K. A. The landscape of parallel computing research: A view from berkeley, Dec 2006.

[6] BRADFORD, J. P., AND FORTES, J. Performance and memory-access characterization of data mining applications. *Annual IEEE International Workshop on Workload Characterization* (1998).

[7] BRADFORD, J. P., AND FORTES, J. A. B. Characterization and parallelization of decision-tree induction. *Journal of Parallel and Distributed Computing 61*, 3 (2001), 322–349.

[8] CHEN, Y., DIAO, Q., DULONG, C., HU, W., LAI, C., LI, E., LI, W., WANG, T., AND ZHANG, Y. Performance scalability of data-mining workloads in bioinformatics. Tech. rep., Intel Technology Journal, 2005.

[9] DUBEY, P. A platform 2015 workload model recognition, mining and synthesis moves computers to the era of tera. In *Platform 1015*. 2005.

[10] FOTI, D., LIPARI, D., PIZZUTI, C., AND TALIA, D. Scalable parallel clustering for data mining on multicomputers. *Parallel and Distributed Processing* (2000), 390–398.

[11] GHOTING, A., BUEHRER, G., PARTHASARATHY, S., KIM, D., NGUYEN, A., CHEN, Y.-K., AND DUBEY, P. A characterization of data mining algorithms on a modern processor. In *DaMoN '05: Proceedings of the 1st international workshop on Data management on new hardware* (New York, NY, USA, 2005), ACM.

[12] HAN, E.-H., KARYPIS, G., AND KUMAR, V. Scalable parallel data mining for association rules. *SIGMOD Rec. 26*, 2 (1997), 277–288.

[13] HAN, J., ALTMAN, R. B., KUMAR, V., MANNILA, H., AND PREGIBON, D. Emerging scientific applications in data mining. *Comminucation of ACM 45*, 8 (2002), 54–58.

[14] JALEEL, A., MATTINA, M., AND JACOB, B. Last level cache (llc) performance of data mining workloads on a cmp - a case study of parallel bioinformatics workloads. In *The Twelfth International Symposium on High-Performance Computer Architecture* (2006), pp. 88–98.

[15] JIBAJA, I., AND SHAW, K. Understanding the applicability of CMP performance optimizations on data mining applications. In *the IEEE International Symposium on Workload Characterization (IISWC 2009)*. (October 2009).

[16] JOSHI, M., KARYPIS, G., AND KUMAR, V. Scalparc: A new scalable and efficient parallel classification algorithm for mining large datasets. *International Parallel Processing Symposium* (1998).

[17] JOSHI, M. V., HAN, E.-H. S., KARYPIS, G., AND KUMAR, V. Parallel algorithms for data mining. In *Sourcebook of Parallel Computing*, J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, Eds. Morgan Kaufmann, 2002.

[18] LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. H. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of MICRO 1997* (1997).

[19] LI, W., LI, E., JALEEL, A., SHAN, J., CHEN, Y., WANG, Q., IYER, R., ILLIKKAL, R., ZHANG, Y., LIU, D., LIAO, M., WEI, W., AND DU, J. Understanding the memory performance of data-mining workloads on small, medium, and large-scale cmps using hardware-software co-simulation. In *the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. (April 2007).

[20] LIU, Y., PISHARATH, J., LIAO, W., MEMIK, G., CHOUDHARY, A., AND DUBEY, P. Performance Evaluation and Characterization of Scalable Data Mining Algorithms. In *Proceedings of IASTED* (2004).

[21] LUK, C.-K., COHN, R., MUTH, R., PATIL, H., KLAUSER, A., LOWNEY, G., S.WALLACE, REDDI, V. J., AND HAZELWOOD, K. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of Programming Language Design and Implementation (PLDI)* (June 2005).

[22] NARAYANAN, R., OZISIKYILMAZ, B., ZAMBRENO, J., MEMIK, G., AND CHOUDHARY, A. Minebench: A benchmark suite for data mining workloads. In *the IEEE International Symposium on Workload Characterization (IISWC)* (October 2006).

[23] OZISIKYILMAZ, B., NARAYANAN, R., ZAMBRENO, J., MEMIK, G., AND CHOUDHARY, A. An architectural characterization study of data mining and bioinformatics workloads. In *the IEEE International Symposium on Workload Characterization (IISWC)* (October 2006).

[24] PARTHASARATHY, S., ZAKI, M. J., OGIHARA, M., AND LI, W. Parallel data mining for association rules on shared-memory systems. *Knowledge and Information Systems 3*, 1 (2001), 1–29.

[25] S. ERANIAN. Perfmon: Linux performance monitoring for IA64. http://perfmon2.sourceforge.net/.

[26] SHAW, K. Understanding the working sets of data mining applications. In *the Eleventh Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-11)* (2008).

[27] SHERWOOD, T., SAIR, S., AND CALDER, B. Phase tracking and prediction. In *Proc. of the 30th Intl. Symp. on Computer Architecture (ISCA)*. 2003.

[28] SKILLICORN, D. Strategies for parallel data mining. *IEEE Concurrency 7*, 4 (1999), 26–35.

[29] STANDARD PERFORMANCE EVALUATION CORPORATION. The SPEC Benchmark Suite. http://www.specbench.org.

[30] STOFFEL, K., AND BELKONIENE, A. Parallel k/h -means clustering for large data sets. *Euro-Par'99 Parallel Processing* (1999), 1451–1454.

[31] TALIA, D. Parallelism in knowledge discovery techniques. *Applied Parallel Computing* (2002), 758–758.

[32] TAN, P., STEINBACH, M., AND KUMAR, V. *Introduction to Data Mining*. Addison-Wesley, 2005.

[33] UNIVERSITY OF HELSINKI. Frequent itemset mining data repository. http://fimi.cs.helsinki.fi/data.

[34] ZAKI, M., HO, C.-T., AND AGRAWAL, R. Parallel classification for data mining on shared-memory multiprocessors. In *Proceedings on the 15th International Conference on Data Engineering* (1999), pp. 198–205.

[35] ZAKI, M., OGIHARA, M., PARTHASARATHY, S., AND LI, W. Parallel data mining for association rules on shared-memory multi-processors. In *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing* (1996), pp. 43–43.

[36] ZAKI, M. J., AND HO, C.-T., Eds. *Large-Scale Parallel Data Mining*, vol. 1759 of *LNCS/LNAI State-of-the-Art Survey*. Springer-Verlag, Heidelberg, Germany, 2000.