NU-MineBench: Understanding the Performance and Scalability Characteristics of Data Mining Algorithms

Jayaprakash Pisharath, Ying Liu, Wei-keng Liao, Gokhan Memik, Alok Choudhary
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL – 60208.
{ jay, yingliu, wkliao, memik, choudhar}@ece.northwestern.edu

Pradeep Dubey
Architecture Research Labs
Intel Corporation
Santa Clara. CA - 95052
pradeep.dubey@intel.com

Abstract

Data mining has become one of the most essential tools for various businesses as well as researchers in diverse fields. The surge in the operational speed of computing systems, and also the emergence of compact, low-cost, high-performance parallel and distributed systems have provided abundant venues for improving the performance of data mining algorithms. However, in recent years, there has also been a tremendous increase in the size of data that is collected and also the complexity of data mining algorithms themselves. The rate of this growth exceeds the rate of performance improvements in computing systems, thus widening the performance gap between data mining systems and algorithms. In this paper, our goal is to narrow this gap by enabling designers to build systems that are tuned in accordance with the requirements and developments of data mining algorithms. We achieve this by performing a detailed characterization of a set of representative data mining programs from both the hardware and software perspectives. We first study several widely-used data mining algorithms from multiple categories and, then, use them to design NU-MineBench, a benchmarking suite containing representative data mining applications. MineBench suite includes two classification, two association rule mining, and four clustering applications. We evaluate the NU-MineBench applications on an 8-way shared memory parallel machine and analyze important performance characteristics of the applications. We believe that this information can aid designers of future systems as well as programmers of new data mining algorithms to achieve better system and algorithmic performance. Moreover, current trends indicate that systems are moving towards the deployment of multiple processing cores as a way to increase their total computation power. Hence, scalability is an inevitable factor expected in future algorithms and systems. With this in mind, we vary both the input data sets and the number of processors used in our evaluation process. We present the results based on various characteristics that span from the software level to the hardware level, such as I/O complexity, fraction of time spent in the OS mode, breakdown of execution cycles, memory hierarchy behavior, communication/synchronization overhead.

1. Introduction

With the enhanced features in recent computer systems, increasingly larger amounts of data are being accumulated in various fields. Without any sophisticated analysis tools, this data is useless. Especially, as the

CUCIS TECHNICAL REPORT # CUCIS-2004-05-001
CUCIS, Northwestern University, Evanston. IL.

data sizes are exponentially increasing, the need to use automated tools to extract information from the collected data becomes clear. Therefore, data mining programs have become essential tools in many domains including marketing, customer relationship management, scoring and risk management, recommendation systems, and fraud detection. In addition, data mining techniques have been adopted by various scientists to analyze the vast amounts of data representing real-world systems. For example, data mining techniques have been utilized in cosmology simulation, climate modeling, bioinformatics, drug discovery, and intrusion detection.

As the amount and dimensions of data collected increases, we will need to utilize even more sophisticated data mining applications. However, one important obstacle that has to be addressed is the fact that the performance of computer systems is improving at a slower rate compared to the increase in the requirements of data mining applications. Recent trends suggest that the system performance (data based on memory and I/O bound workloads like TPC-H) has been improving at a rate of 10-15% per year, whereas, the volume of data that is collected doubles every year. Also, existing data mining tools are not able to run efficiently on existing systems. Researchers have focused on efficient implementations of different data mining algorithms by proposing numerous algorithmic optimizations and by proposing parallel and distributed versions of these algorithms. However, it is clear that even such optimized versions of algorithms have long run times. We believe that in order to close this gap, the key is to develop an understanding of the characteristics of data mining applications and to identify the way these applications get mapped on to existing computing systems. This is the goal of this paper. This information in turn can be utilized during the implementation of the algorithms and the design/setup of the computing systems. Understanding the architectural bottlenecks is essential not only for processor designers to adapt their architectures to data mining applications, but also for programmers to adapt their algorithms to the revised requirements of applications and architectures.

Data mining is a relatively new application area and it involves algorithms and computations from different domains such as mathematics, machine learning, statistics, and databases. For this reason, very little is known in terms of the characteristics of the underlying computations and data manipulation, and their impact on computer systems. We address this issue in this paper by trying to investigate data mining applications and their characteristics for a sequential processor as well as for a representative parallel architecture. We first establish a benchmarking suite of applications that encompasses algorithms commonly used in data mining. Then, we analyze the architectural properties of these applications in detail to investigate the bottlenecks associated with them. Specifically, in this paper we make the following contributions:

1) We introduce NU-MineBench, a benchmarking suite that includes popular data mining applications from various categories,

2) We analyze the architectural properties of the applications on a sequential processor and highlight important performance bottlenecks, and

3) We analyze the scalability of these applications in terms of data and parallelization.

Benchmarks play a major role in all domains. SPEC [23] benchmarks have been well accepted and used by several processor manufacturers and researchers to measure the effectiveness of their design. Other fields have popular benchmarking suites designed for the specific application domain: TPC [24] for database systems, SPLASH [26] for parallel machine architectures, MediaBench [15] for media and communication processors. Benchmarks do not only play a role in measuring the relative performance of different systems. They also aid programmers in the specific domain in various ways. For example, a programmer implementing a new data mining application can compare the performance (in terms of output quality, scalability, and execution time) of the new application to the applications in the benchmarking suite. In addition, the programmer can use certain types of algorithms and programming styles from the applications in the existing suite.

Although there has been previous work analyzing individual data mining applications [3, 14], analyzing the behavior of a complete benchmarking suite will certainly give a better understanding of the underlying bottlenecks for data mining applications. We analyze each application in our benchmark on a sequential processor and present the key characteristics of each algorithm. Using our experimental results, we show that data mining applications are highly computation intensive, where the OS, I/O and synchronization overheads usually constitute a small fraction of the overall execution times.

Another important aspect of our study is implementing and analyzing parallel versions of our benchmark applications. As the size of the available datasets and their dimensionality grow, parallel computers are becoming essential platforms to execute the data mining applications. In fact, data mining is rapidly becoming the most widely executed application category for supercomputers [11]. Some of the existing data mining applications have already been parallelized. However, the parallelization is usually made in an ad-hoc manner. We analyze the characteristics of the applications in shared-memory Multi-Processor (SMP) machine. Despite their limited scalability, SMPs have become the most common parallel computing type in the industry due to their simplicity. By analyzing the application characteristics in this representative multiprocessor system, we provide an insight into the parallel applications, which can be potentially helpful when developing parallel data mining algorithms on SMPs.

We believe that our study is unique in nature. While data mining algorithms are typically characterized by people who propose them, we believe that our study highlights some of the major issues that have been assumed to be true during such studies. It should be noted that we do not study just a single data mining algorithm. Instead, we consider applications (not algorithms) from various data mining domains. Moreover, our atypical approach, which is a bottom-up approach to understanding data mining algorithms, identifies the real performance bottlenecks. The results are from real system evaluation as against a simulation setup. This provides new insights and hence, new venues for optimizations.

The rest of the paper is organized as follows. In the next section, we overview the related work. In Section 3, we discuss the data mining applications that are included in our benchmarking suite. Section 4 presents the evaluation methodology. The characteristics of our benchmark applications are presented in Section 5. Section 6 summarizes the results.

2. Related work

Since fast-growing, tremendous amount of data, collected and stored in large and numerous databases, has far exceeded our human ability for comprehension without powerful tools, data mining technologies, which can perform fast data analysis and uncover important data patterns, have attracted a great deal of attention in various domains in the recent years [7, 18]. However, the increase in application requirements far exceeds the increase in computing power of systems, which suggests parallel computing as a potential solution to meet the requirements. In the past decade, most research on parallel data mining [7, 12, 28] has been focused on distributed-memory parallel machines due to its capability for massive parallelism. However, share-memory parallel machines are becoming the dominant types of parallel machines in industry because of its simplicity and low to medium degree of parallelism besides its nominal price. A few parallel algorithms on SMPs have been proposed in [27, 28].

We include some of the commonly used data mining algorithms as representatives of each category into our NU-MineBench, and we perform evaluation on shared-memory parallel machines at the architecture level. Similar performance characterization work of database workloads is seen in [8, 13], and specifically targeted for SMPs in [22, 25]. Performance characterization of individual data mining algorithm has been done in [3, 14], where they focus on the memory and cache behaviors of a decision tree induction program. However, we believe that analyzing the behaviors of a complete data mining benchmarking suite will certainly give a better understanding of the underlying bottlenecks for data mining applications.

3. NU-MineBench Application Suite

Data mining applications are broadly classified into classification, clustering, association rule mining, sequence mining, similarity search, text mining, multimedia mining, and other categories based on the nature of their algorithms [7]. We first establish NU-MineBench, a benchmarking suite containing data mining applications. The selection of categories as well as the applications in each category is based on how commonly these applications are used in industry and how likely to be used in the future, thereby achieving a realistic representation of the existing applications. Another concern of the algorithm selection is the scalability when executing on parallel or distributed systems. For instance, the flow and the set of operations performed by algorithms in our suite can be seen in various data mining tools currently available from the industry, like Clementine [31], IBM Intelligent Data Miner [32] and SAS Enterprise Miner [33]. NU-MineBench has 8 applications from three of the categories listed above: classification, association rule mining (ARM), and clustering. We parallelize 5 out of the 8 applications because they show good scalability and performance on very large-scale databases in literature. The applications as well as important characteristics of the applications are listed in Table 1, which presents the applications, the category they belong to, a short description of the applications, and the programming language used to implement it. In the following sections, we discuss each application in detail according to the category they belong to.

3.1 Classification Programs

A classification problem has an input dataset called the training set which consists of example records with a number of attributes. The objective of a classification algorithm is to use this training dataset to build a model such that the model can be used to assign unclassified records into one of the defined classes. Classification has applications in diverse fields such as retail marketing, fraud detection, and design of telecommunication service plans [7]. Representative algorithms include decision tree, Bayesian classification, backpropagation, and neural networks.

ScalParC is an efficient and scalable variation of decision tree classification [12]. The decision tree model is built by recursively splitting the training dataset based on an optimal criterion until all records belonging to each of the partitions bear the same class label. Decision trees can easily be converted to classification rules [7, 12]. Among many classification methods proposed over the years, decision trees are particularly suited for data mining, since they can be built relatively fast compared to other methods, especially when database is large. They are also easy to interpret [21]. Decision tree classifiers obtain similar, and often better, accuracy compared to other methods [19].

Bayesian classifiers are statistical classifiers. They predict the probability that a record belongs to a particular class. It is based on Bayes' Theorem. A simple Bayesian classifier, called Naive Bayesian classifier [5], is comparable in performance to decision trees and exhibits high accuracy and speed when applied to large databases.

3.2 Clustering programs

Clustering is the process of discovering the groups of similar objects from a database to characterize the underlying data distribution. It has wide applications in market or customer segmentation, pattern recognition, biological studies, and spatial data analysis [7]. Generally, clustering algorithms can be classified into four categories: partitioning-based, hierarchical-based, density-based, and grid-based.

The first clustering application in NU-MineBench is **K-means** [16]. K-means is a partition-based method and is arguably the most commonly used clustering technique. K-means represents a cluster by the mean value of all objects contained in it. Given the user-provided parameter k, the initial k cluster centers are randomly selected from the database. Then, K-means assigns each object to its nearest cluster center based on the similarity function. For example, for spatial clustering, usually the Euclid distance is used to measure the closeness of two objects. Once the assignments are completed, new centers are found by finding the mean of

Table 1. MineBench applications. Category that the application belongs to, a short description of the application, and the programming language used to implement it.

Algorithms	Category	Description	Lang.
ScalParC	Classification	Decision tree classifier	C
Naïve Bayesian	Classification	Statistical classifier based on class conditional independence	C++
K-means	Clustering	Partitioning method	C
Fuzzy K-means	Clustering	Fuzzy logic based K-means	C
BIRCH	Clustering	Hierarchical method	C++
HOP	Clustering	Density-based method	C
Apriori	ARM	Horizontal database, level-wise mining based on Apriori property	C/C++
Eclat	ARM	Vertical database, break large search space into equivalence class	C++

all the objects in each cluster. This process is repeated until two consecutive iterations generate the same cluster assignment.

The clusters produced by the K-means algorithm are sometimes called "hard" clusters, since any data object either is or is not a member of a particular cluster. The **Fuzzy K-means** algorithm [2] relaxes this condition by assuming that a data object can have a degree of membership in each cluster. The Fuzzy K-means assigns each pair of object and cluster a probability. For each object, the sum of the probabilities to all clusters equals to 1. Compared to the Euclid distance used in K-means, the calculation for the fuzzy membership results in higher computational cost. However, the flexibility of assigning objects to multiple clusters might be necessary to generate better clustering qualities.

BIRCH [29] is one of the hierarchical clustering methods that employ a hierarchical tree to represent the closeness of data objects. BIRCH first scans the database to build a clustering-feature (CF) tree to summarize the cluster representation. Then, a selected clustering algorithm, such as K-means, is applied to the leaf nodes of the CF tree. For a large database, BIRCH can achieve good performance and scalability. It is also effective for incremental clustering of incoming data objects.

Density-based methods grow clusters according to the density of neighboring objects or according to some other density function. **HOP** [6], originally proposed in astrophysics, is a typical density-based clustering method. After assigning an estimation of its density for each particle, HOP associates each particle with its densest neighbor. The assignment process continues until the densest neighbor of a particle is itself. All particles reaching this state are clustered as a group. HOP is highly scalable when applied to large databases [30]. HOP can be applied in diverse applications in molecular biology, geology, and astronomy.

3.3 Association Rule Mining(ARM) programs

Association rule mining is to find the set of all subsets of items or attributes that frequently occur in database records. In addition, ARM programs extract rules on how a subset of items influence the presence of another subset [7, 28]. ARM can discover interesting association relationships among large number of business transaction records. This can aid business decision-making processes, such as catalog design, cross-marketing, and loss-leader analysis [7].

Apriori [1] is arguably the most influential ARM algorithm. It explores the level-wise mining of Apriori property: all nonempty subsets of a frequent itemset must also be frequent. At the kth iteration (for $k > 1$), it forms frequent (k+1)th-itemset candidates based on the frequent k-itemsets and scans the database to find the complete set of frequent (k+1)th-itemsets, L_{k+1}. To improve the efficiency, a hash-based technique is used to reduce the size of the candidate k-itemsets.

Eclat [28] uses a vertical database format. It can determine the support of any k-itemset by simply intersecting the id-list of the first two (k-1)-length subsets that share a common prefix. It breaks the search space into small, independent, and manageable chunks. Efficient lattice traversal techniques are used to identify all the true maximal frequent itemsets.

3.4 Parallel implementation

As mentioned in the earlier sections, part of our goal is to study the scalability of data mining applications on SMPs. Hence, parallel versions of our benchmark applications are also provided. Parallel experimental results have been provided for 5 applications out of the 8 benchmark applications: ScalParC (Classification), K-means, Fuzzy K-means, HOP (Clustering), and Apriori (ARM). We chose these applications not only because these parallel algorithms are commonly found in the literature, but also because they demonstrate good performance and scalability when applied to large-scale databases. ScalParC is parallelized on SMPs using the scheme presented in [27]. Simple data parallelism is exploited to parallelize K-means, Fuzzy K-means, and HOP. We implement parallel Apriori based on the Common Candidate Partitioned Database (CCPD) strategy described in [28].

3.5 Discussion

In this section, we have presented eight applications that are included in NU-MineBench. All of these applications are complete applications as opposed to kernels or procedures, i.e. each application can be executed independently without affecting the others. This is an important property for the benchmarks. Our goal is not to study a small portion of the tool, but rather to look at the whole application and investigate bottlenecks that might be caused by I/O, OS, or the application itself. Typically algorithm developers evaluate their algorithms based on their optimizations and also based on a fixed computing system. We believe that when applications are implemented based on these algorithms, there are various factors that affect the overall performance of the algorithm (for instance, the operating system overheads). Another important aspect is the relation between the implementation and the algorithm. It is natural to question whether a bottleneck arises because of the inefficiency of the implementation or because of the inherent nature of the algorithm. Therefore, we have rigorously optimized the applications for our system. Hence, any characteristic we observe during the evaluation process is likely to be inherent in the algorithm. However, due to the nature of our study, we emphasize on the trends and relative results rather on the exact performance numbers. Such trends and relative results are usually independent of the specific implementation.

4. Evaluation Methodology

Benchmarks are used to evaluate architectures, methodologies, implementations and application algorithms. Hence, applications in a benchmark need to have distinct characteristics. In the next section, we consider the applications from our NU-MineBench suite, and distinguish the characteristics that make each application unique by studying it both from the algorithmic and the system perspective. For this, we first consider a parallel environment and port our applications to that system. Then, each application is evaluated for performance and scalability by varying the number of processors and data sizes. Routines within each application are analyzed in detail both from the functional and architectural granularity, to identify the key parameters in each algorithm.

In the following part, we present the parallel setup that forms the basis for our experiments. Subsequently, we elaborate on the software tools that we used for parallelization, algorithm evaluation and also for studying the architectural performance. The input data set considered in our experiments is discussed in Section 4.3.

4.1. Hardware setup

We chose an Intel IA-32 multiprocessor platform for evaluation purposes. Our setup consists of an Intel Xeon 8-way Shared Memory Parallel (SMP) machine running Red Hat Linux Advanced Server 2.1 operating system. The system has a 4GB shared memory and 1024 KB L2 cache for each processor. Each processor has 16KB non-blocking, integrated L1 instruction and data caches.

4.2. Software tools

In all the experiments, we use VTune Performance Analyzer [10] for profiling the functions within our application, and for measuring their execution times. To trace subroutine calls, we use VTune calling graph, which presents a hierarchical decomposition of the execution time. VTune counter monitor provides a wide assortment of metrics. We look at different characteristics of the applications: execution time, fraction of time spent in the OS space, communica-tion/synchronization complexity, I/O complexity, memory behavior, and CPI behavior. Related VTune parameters are used to collect data for these properties.

In parallel implementations of the applications, we use OpenMP pragmas [20]. OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism. Due to its simplicity, OpenMP is quickly becoming one of the most widely used programming styles for SMPs. In SMPs, processors communicate through shared variables in the single memory space. Synchronization is used to coordinate processes. Similar to other parameters, VTune provides the aggregate time spent on different types of pragmas that are used for job parallelization and synchronization (includes individual loops and routines as well). This way we can accurately measure the time spent on synchronization, and other relevant contentions. For compiling applications, we use the parallel Intel C++ compiler, version 7.1 for Linux.

4.3. Dataset Characteristics

Input data is an integral part of the data mining applications. The data considered for our experiments are either real data got from various fields or widely-accepted synthetic data generated using existing tools that are used in scientific and statistical simulations. During evaluation, we use multiple data sizes to investigate the characteristics of the NU-MineBench applications. Particularly, for each application we generate input data in 3 different sizes: Small, Medium, and Large. For ScalParC and Naïve Bayesian, we use three synthetic datasets (see Table 2 – "Classification") generated by the IBM Quest data generator [9]. The notation **F**x-**A**y-**D**zK denotes a dataset with **F**unction x, **A**ttribute size y, and **D**ata comprising of z*1000 records. Function 26 is a relatively complex function and produces large trees. For Apriori and Eclat, we also use three synthetic datasets from IBM Quest data generator (see Table 2 – "ARM"). D denotes the number of transactions, T is the average transaction size, and I is the average size of the maximal potentially large itemsets. In Table 2, the number of items is 1000 and the number of maximal potentially large itemsets is 2000. For HOP and BIRCH, we use three sets of real data from a cosmology application, ENZO [4], each having 61440 particles, 491520 particles and 3932160 particles. We use a section of the real image database distributed by Corel Corporation for K-means and Fuzzy K-means. This database consists of 17695 scenery pictures. Each picture is represented by two features: color and edge. The color feature is a vector of 9 floating points while the edge feature is a vector of size 18. Both K-means implementations use Euclid distance as the similarity function and execute it for the two features separately. Since the clustering quality of K-means methods highly depends on the input parameter k, we perform both K-means with ten different k values ranging from 4 to 13. The timing results provided in this paper are the accumulated time for the ten runs.

Table 2. Classification and Association Rule Mining Dataset Characteristics. (Dataset size in MB)

Dataset	Classification		ARM	
	Parameter	Size	Parameter	Size
Small	F26-A32-D125K	27	T10-I4-D1000K	47
Medium	F26-A32-D250K	54	T20-I6-D2000K	175
Large	F26-A64-D250K	108	T20-I6-D4000K	350

5. Program characteristics

In this section, we analyze several characteristics of NU-MineBench programs. For each characteristic, we analyze how the results vary when we change the input data and when we change the number of processors used in the execution. During the experiments, we investigate the performance scalability of the applications from NU-MineBench on the SMP machine. The benefits and drawbacks of using a shared memory model for our data mining algorithms are also discussed. Our measures of interest include the overall program execution time, the operating system overheads, I/O times and synchronization times. The numbers presented in this paper are for the entire application, and in relevant contexts, we present the individual processor breakups as well.

5.1. Execution time

Table 3 shows the application execution times on 1 processor and speedups with respect to 1 processor case. For all benchmarks, the data size is varied from Small (S), to Medium (M), and finally to Large (L) based on the parameters of Section 4.3. We measure the scalability of the parallel applications by executing them on 1, 4 and 8 processors. The performance numbers for the 2-processor case is not presented in our paper due to the fact that there is minimal (or in some cases, none) improvement in performance when the application is executed on 2 processors.

For our parallel applications, the best speedup (improvement in execution time with respect to the 1 processor case) is seen in the decision tree algorithm (ScalParC). A speedup of 6.19 for 8 processors arises due to the balanced partitioning of data on to processors. This avoids concurrent read-write operations to the shared variables, which minimizes the contention during memory access (note: ours is a shared memory model). If data is evenly distributed, each processor is able to work independently (faster) by accessing only its respective data block in the memory without requiring access to memory blocks of other processors. HOP follows ScalParC in terms of the achieved speedups. Apriori has limitations when extended to SMPs. This is due to the significant amount of atomic access to the shared hash-tree structure and the nature of unbalanced transaction data. Overall, it is evident from Table 3 that data mining algorithms are scalable. Care should be taken to make sure shared operations are minimal in applications. That way applications can be efficiently hosted on multiple processing cores to exploit parallelism, and thus, to achieve high speedups. Typically, in data mining algorithms, the computation kernels are significant and constitute majority of the total execution times. It will be evident from further sections that the data retrieval is fast but data reuse is not very efficient, which we attribute to the nature of data mining algorithms.

Table 3. Execution times for NU-MineBench applications. S is the small data set, M is medium data set and L is the large data set, except for K-means, in which case, S is color and M is edge data. P1, P4, P8 represent 1, 4 and 8 processor cases. The values shown under the column for P1 are the actual execution times in seconds, while the columns P4 and P8 show the speeds attained with respect to the 1 processor case.

Program	Data set = S			Data set = M			Data set = L		
	P1	P4	P8	P1	P4	P8	P1	P4	P8
HOP	6.3	3.5	5.25	52.7	1.92	6.06	435.3	3.4	5.34
K-means	5.7	2.85	4.38	12.9	3.9	4.96	-	-	-
Fuzzy K-means	164.1	3	6.02	146.8	3.44	5.42	-	-	-
BIRCH	3.5	-	-	31.7	-	-	172.6	-	-
ScalParC	51.0	3.78	4.9	110.6	3.88	5.12	225.9	3.9	6.19
Bayesian	12.6	-	-	25.1	-	-	51.5	-	-
Apriori	6.1	2.03	2.35	102.7	2.66	3.36	200.2	2.76	3.18
Eclat	11.8	-	-	81.5	-	-	127.8	-	-

5.2. Operating system overhead

For any program, the CPU utilization is split into operating system (OS) and user space. The OS overheads in a program include factors like system calls (for process/thread management, invoking locks, handling hardware interrupts), and allocation of intermediate system buffers during program execution. In Figure 1, we present the OS component (as a percentage of total execution time) of each individual application. When the number of processors equals to one, the operating system overheads are minimal. The OS overheads that arise in the single processor case are primarily from due to intermediate buffer allocations. The maximum overhead (1.7%) is seen for BIRCH. When the number of processors deployed is increased, the OS component increases drastically due to the parallelization overheads. Under the OpenMP programming environment, each OpenMP (_omp) directive adds extra cycles of overhead. These directives include the ones used during program initialization, thread spawning, barrier controls and also program loop hints. The individual program locks (which are basically system locks) used during parallelization also contribute to the OS overheads. Collectively, it is seen that when the code is parallelized to more processors, the OS overheads increase. Among the applications, K-means has the worst overhead. The OS overheads can help explain the poor performance of a given code. For instance, K-means shows an average speed up of 4.96 for 8 processors. This is as a result of the 40% OS overhead (Figure 1) from the omp directives and locks during the parallelization of K-means. The percentages of user space are similar for all data sizes and hence are not presented.

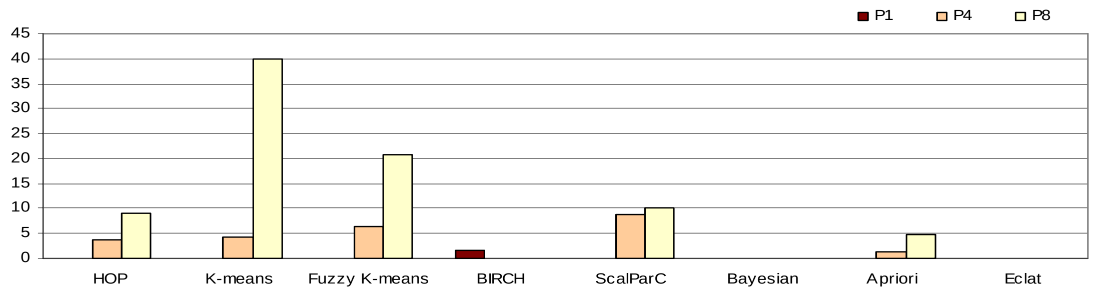

Figure 1. OS overheads of NU-MineBench applications as a percentage of the total execution time. Data size is medium (M) for each application.

5.3 I/O time

As mentioned earlier, the external overheads of a program could affect the overall performance drastically. In general, I/O is a key component that could affect the overall performance of a system. Figure 2 shows the time for performing I/O as a percentage of the overall execution time. It is clear that the overheads arising from I/O operations generated during the data retrieval process in our applications are typically small except for Bayesian. For Bayesian, data is read as ASCII characters one by one, whereas for ScalParC (another classification algorithm), data is read in bulk string mode (less read operation overheads). This indicates that bulk loading of data could help. Considering the growth in memory technology (more storage, compact and partitioned layouts, faster access and cheaper cost), such mechanisms must be easy to implement. Data mining algorithms are yet to take full advantage of such technology advancements. To study how the I/O scales with respect to increasing data sizes, we varied the input data sets (S, M, L). Figure 2 shows that for a few applications, the I/O scales in an orderly fashion. For instance, in ScalParC and Apriori, on increasing the data sizes from S to M to L, the corresponding I/O percentages reduce. When the system reads more data (implies more I/O), the CPU gets more data to "mine". Thus, CPU computations outperform the I/O

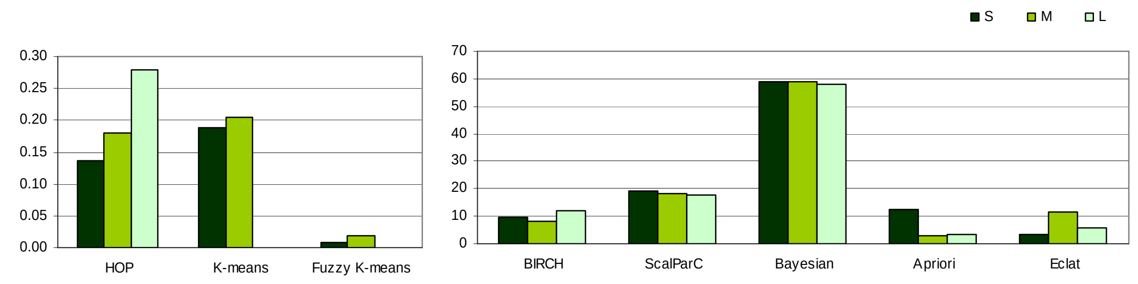

Figure 2. Percentage of I/O time with respect to the overall execution times.

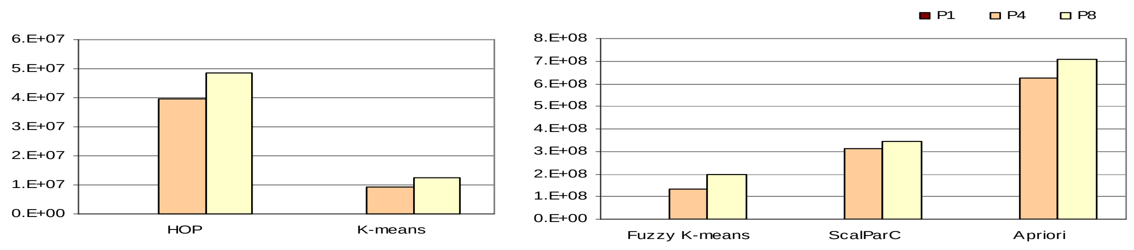

Figure 3. Synchronization time in CPU cycles for all applications. The synchronization time increases when computation is scaled to multiple processor.

operations, which implies, the CPU is well utilized. Overall, these results highlight the computation-intensive nature of our benchmark.

5.4 Communication/Synchronization Overhead

In a shared memory model, the inter-processor communication is achieved by accessing shared memory addresses. To access a shared variable (which in turn is a shared location in the memory), the processors have to pay a penalty. Processors request read permission to the shared variables and then “wait”. This could be a considerable bottleneck if the shared variable is locked by another processor, in which case the requesting processor must wait until the lock is released. Moreover, during parallel execution, there are execution breakpoints where all processors need to synchronize their data values for all their local/shared variables. This again, could be another bottleneck. All such inter-processor communication overheads are reflected in the synchronization measurement of our benchmark. The synchronization costs are shown in Figure 3 for 1, 4 and 8 processor case. When using one processor, the synchronization overheads are negligible due to no inter-processor communication. When more processors are involved, shared and private variables arise. In our case, the synchronization overheads increase as more processors are brought into the system. We found that for all parallel applications, the average synchronization time is just 0.14% of the overall execution time. This implies that the idle time spent in synchronization is very less and the CPU is very well utilized for mining information from the input data.

The synchronization times increase when more data is brought into the system. This is due to the increase in the amount of data that is shared between processors, which in turn increases the need for timely synchronizations. This is shown in Figure 4. It should be noted that in case of K-means, S is color dataset and M is the edge dataset, which are independent datasets. This is the reason for the drop in synchronization cycles as we move from S to M.

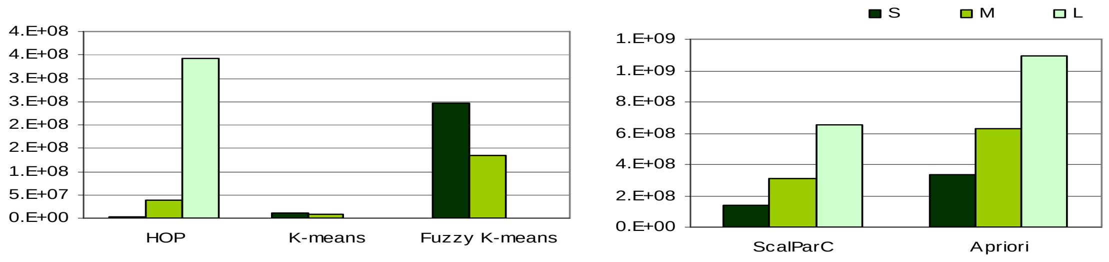

Figure 4. Synchronization time in CPU cycles for all applications for different datasets. The synchronization time increases when data size is increased.

5.5 Memory Analysis

Studies have indicated that memory hierarchy is a significant performance bottleneck in modern computing systems [17]. This is more relevant in our case due to the low I/O overhead, as seen in the pervious section. When data is read, it is brought to the memory, which implies, understanding the program characteristics from the memory hierarchy is essential to improve the overall performance. Here, we present the performance characteristics of our programs with respect to the L1, L2 data caches and memory.

5.5.1 L1 D-Cache

Figure 5 shows the L1 cache miss ratio (percentages) when our applications are executed on 1, 4 and 8 processors. It is clear that the applications are drastically different in their L1 cache behaviour. The applications can be categorized into two: one that has less cache misses (<0.6%) and those having more than 2% cache misses. The maximum cache miss ratio (system-wide) is less than 7%, which is less considering the amount of data that is processed in our applications. We also measure the misses that occur on individual processors. It is not presented here as the trends are similar to the overall application trends except that the master processor incurs supplementary misses due to its additional task of managing and coordinating tasks that run on other processors. We also varied the data sizes to study the effect of increased data processing on L1 caches. Figure 6 shows the results. In general, the misses increase when data sizes are increased. This is due to the limited capacity of L1 cache (16KB).

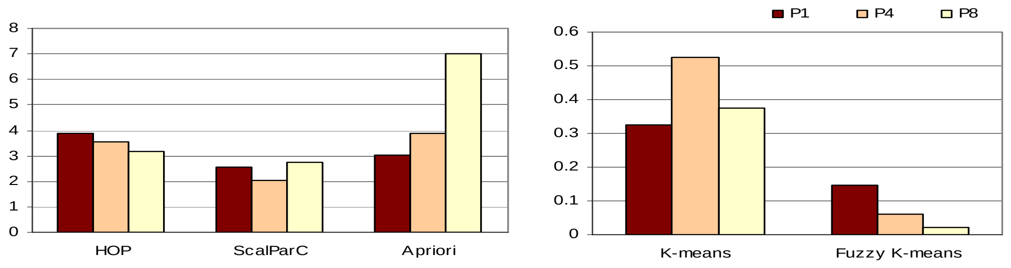

Figure 5. L1 miss ratio (percentages) for NU-MineBench applications on 1, 4 and 8 processors.

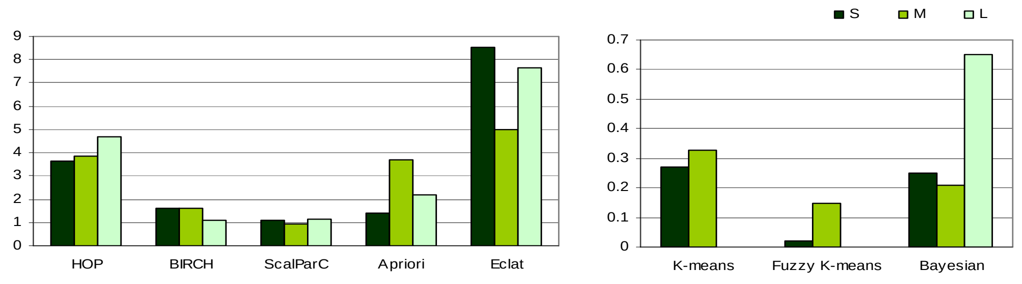

Figure 6. L1 miss ratio (percentages) for NU-MineBench applications with S, M and L datasets for the single processor case.

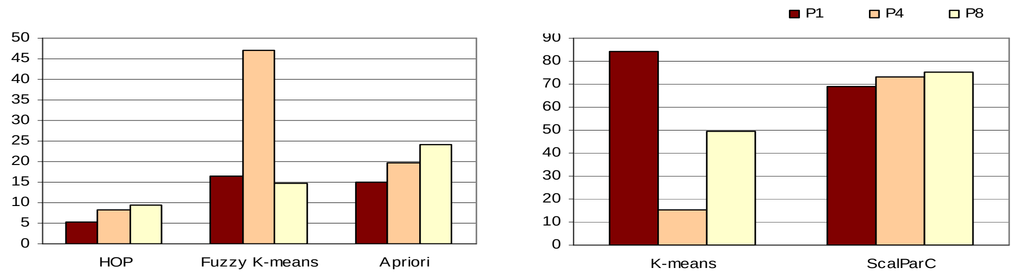

Figure 7. L2 miss ratio (percentages) for NU-MineBench applications on 1, 4 and 8 processors.

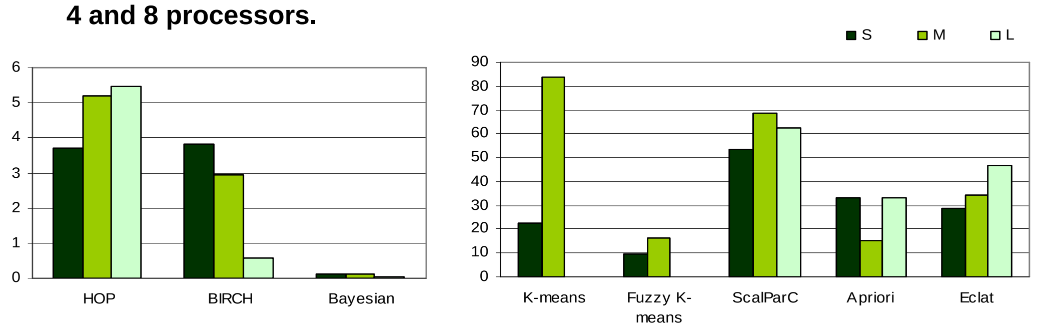

Figure 8. L2 miss ratio (percentages) for NU-MineBench applications with S, M and L datasets for the single processor case.

5.5.2 L2 Cache

We also performed an analysis of the L2 cache behavior as well. Figure 7 shows the L2 miss ratios (percentages) for the NU-MineBench applications when executed on 1, 4 and 8 processors. Figure 8 shows the L2 miss ratio for varying data sizes. It is evident that L2 cache performance is dissimilar across applications, for both the cases when processor sizes and data sets are increased. In certain cases, the L2 cache misses increase, while in others it decreases. One reason for this behavior is that the data distribution is random as we use dynamic scheduling for parallelization of our applications. In dynamic schemes, the processor gets assigned a new block of data in a random fashion as it becomes available. Hence, the data gets distributed to multiple caches in a random fashion, which increases the likelihood of not finding "spatial" (nearer items in space) or "temporal" (nearer items in time) data. The scenario changes when the data size is increased. For a single processor case, the cache is not able to accommodate all of the requested data. Hence, as requests increase, misses also increase. But in the case of multiple processors (with large input data sizes), each processor is able to accommodate more data (note: each processor has a 1024KB cache). Each cache is able to accommodate more temporal and spatial data (each processor requests it). In our case, the misses reduce when 4 processors are used instead of 1 (graphs not presented here due to space restrictions). The misses resume to increase once data is extended to 8 processors, which implies the 4 processor case is the best alternative for the large data sets and L2 cache size.

Overall, the cache misses are high for these applications bearing in mind the fact that data has already been loaded efficiently into the memory (as seen before, I/O times are less for the benchmark applications). This implies that there is poor data reuse in applications. We found that a set of applications that have high miss rates incur nearly 0.013 misses per instruction. This indicates that an instruction is bound to miss more than 1% of the time. Considering the fact that there are applications in the suite with lesser misses per instruction (in the order of 0.0001), it is evident that there is more room for cache optimizations in data mining algorithms.

5.5.3 Memory Accesses

To understand the effects of caching, we study the number of memory accesses that go out of the cache to the memory. The access cycles spent in requesting and receiving memory data is presented in Figure 9 for 1, 4 and 8 processor cases. Memory contention has a bad effect on the overall performance as can be seen in the case of Apriori. There are a lot of memory cycles spent during data accesses (arises from wait/synchronization modes), which results in poor speedups. ScalParC has a speedup of 3.8 on 4 processors, but 4.9 on 8 processors. This non-uniform trend in speedups (failure to attain linear speedups) is also attributed to memory contention as well. The memory access times increase when more processors are used due to the repeated data accesses (arising from cache misses, multiple data reads) and the increased time spent in memory idle cycles (during synchronization, waiting for data).

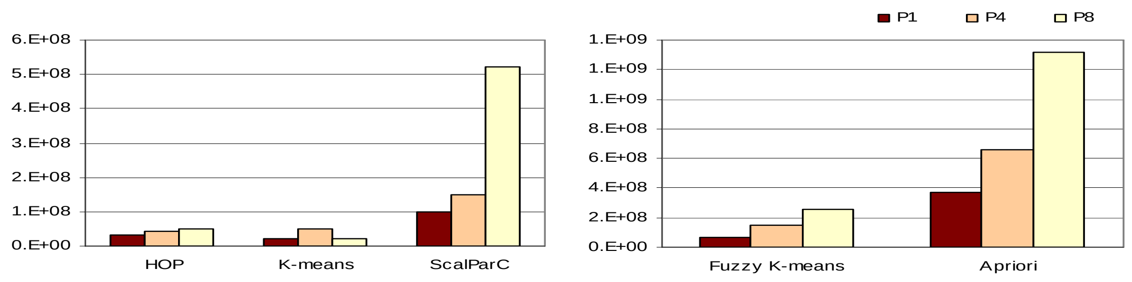

Figure 9. Memory access cycles corresponding to requests going to the memory from cache, for 1, 4 and 8 processors. This includes all memory read/write requests that miss the cache and access the memory.

5.6 CPI behavior

To understand the efficiency of our applications, we studied the Cycles Per Instruction (CPI). CPI is the ratio of the total execution cycles to the number of instructions successfully handled by a processor. We show the CPI for our applications, and with multiple processors in Table 4. The results shown are for the M dataset. For other dataset, the trends were similar. Even though, the CPI for ScalParC and HOP are comparatively less, they outperform other applications by avoiding repeated synchronizations. That is, the CPI is consistent throughout the program execution for ScalParC, whereas for other applications (like Apriori), there is a huge variation of CPI during program execution due to the presence and absence of synchronizations. This disparity in behavior highlights the fact that data mining applications need to better utilize the processor computational resources by hiding the overheads that arise from other non-computational components. Memory latency hiding techniques (like prefetching) should be useful.

Table 4. CPI for applications from NU-MineBench suite. P1, P4, P8 represent 1, 4 and 8 processor cases.

Programs	P1	P4	P8
HOP	1.53	1.36	1.45
K-means	1.82	1.56	1.72
Fuzzy K-means	1.36	1.53	1.61
BIRCH	1.29	-	-
ScalParC	2.96	2.63	2.61
Bayesian	1.20	-	-
Apriori	3.83	2.66	3.44
Eclat	9.59	-	-

6. Conclusions

In this paper, we introduce and evaluate NU-MineBench, a benchmarking suite for data mining applications. NU-MineBench can be efficiently used by system designers as well as programmers for new data mining applications. It contains 8 representative applications: two association rule mining algorithms, two classification algorithms, and four clustering algorithms. We have studied important characteristics of the applications when executed on an 8-way SMP machine. While our results do highlight some existing trends in data mining algorithms, it also identified the major bottlenecks from a different perspective. This fresh outlook has also given us the opportunity to highlight trends that have never been studied before. The following summarizes the broad trends seen in our benchmark applications, and also suggests newer vistas for system and algorithm optimizations.

- Data mining applications are favorably scalable, but special care must be taken when processors share the data, especially in a shared memory environment. Designers can optimize the system bus (interconnect) and network mechanisms, while the algorithm specialists can make sure their algorithms have minimal data sharing requirements.
- Typically, the OS overhead, the synchronization overhead, and the I/O times are usually small in NU-MineBench applications.
- The L1 cache miss rates are typically small. However, the L2 cache miss rates are considerably high, which is not seen in typical applications run on computing systems. We compared these miss rates with those of SPEC and TPC-H benchmarks. Results indicate that data mining applications are unique.
- The weak memory hierarchy performance might be attributed to the small instruction-level parallelism (measured in CPI). These results indicate that improvements in the performance of processors are likely to have a significant impact on the overall performance of data mining systems. In addition, techniques, like prefetching, should also improve the performance of the processor considerably. To improve the performance of their applications, the programmers can utilize this information and achieve better system performance.

Overall, our results indicate that there is ample scope for improvements in the performance of both data mining algorithms and systems. We believe our results could guide programmers and designers to achieve this goal with ease.

Reference

1. Agrawal, R., et al., Fast Discovery of Association Rules. In Advances in Knowledge Discovery and Data Mining. 1995, AAAI/MIT Press.

2. Bezdek, J.C., Pattern Recognition with Fuzzy Objective Function Algorithms. 1981: Plenum Press, New York.

3. Bradford, J.P. and J. Fortes. Performance and Memory-Access Characterization of Data Mining Applications. In Workload Characterization: Methodology and Case Studies. Nov. 1998. Dallas, TX.

4. Bryan, G., T. Abel, and M. Norman. Achieving Extreme Resolution in Numerical Cosmology Using Adaptive Mesh Refinement: Resolving Primordial Star Formation. In SuperComputing. Nov. 2001.

5. Domingos, P. and M. Pazzani. Beyond independence: Conditions for optimality of the simple Bayesian classifier. In 13th Internaltional Conference on Machine Learning. 1996.

6. Eisenstein, D.J. and P. Hut, HOP: A New Group Finding Alogrithm for N-Body Simulations. Journal of Astrophysics, 1998. 498: p. 137-142.

7. Han, J. and M. Kamber, Data Mining: Concepts and Techniques. 2001: Morgan Kaufmann.

8. Hankins, R., et al., Scaling and Charaterizing Database Workloads: Bridging the Gap between Research and Practice. In Intl. Symposium on Microarchitecture, 2003.

9. IBM, IBM synthetic data generation code. http://www.almaden.ibm.com/software/quest/Resources/index.shtml.

10. Intel, C., VTune Performance Analyzer. http://www.intel.com/software/products/vtune/.

11. Joshi, M.V., et al., Parallel Algorithms for Data Mining. CRPC Parallel Computing Handbook. 2000: Morgan Kaufmann.

12. Joshi, M.V., G. Karypis, and V. Kumar. ScalParC: A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets. In International Parallel Processing Symposium. 1998.

13. Keeton, K., et al., Performance Characterization of a Quad Pentium Pro SMP using OLTP Workloads. In Intl. Symposium on Computer Architecture, 1998.

14. Kim, J.-S., X. Qin, and Y. Hsu. Memory Characterization of a Parallel Data Mining Workload. In Workshop on Workload Characterization: Methodology and Case Studies. Nov. 1998. Dallas, TX.

15. Lee, C., M. Potkonjak, and W.H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In International Symposium on Microarchitecture. 1997.

16. MacQueen, J. Some Methods for Classification and Analysis of Multivariate Observations. In 5th Berkeley Symposium on Mathematical Statistics and Probability. 1967.

17. Wolf, Wm. A, and S.A. McKee, Hitting the Memory Wall: Implications of the Obvious. Computer Architecture News, 1995. 23(1): p. 20-24.

18. Michalski, R.S., I. Brakto, and M. Kubat, Machine Leaning and Data Mining: Methods and Applications. 1998, New York: John Wiley & Sons.

19. Michie, D., D.J. Spiegelhalter, and C.C. Taylor, Machine Learning, Neural and Statistical Classification. 1994: Ellis Horwood.

20. OpenMP, OpenMP: Simple, Portable, Scalable SMP Programming. http://www.openmp.org/.

21. Quinlan, J., C4.5 Programs for Machine Learning. 1993: Morgan Kaufmann.

22. Ranganathan, P., et al. Performance of database workloads on shared-memory system with out-of-order processors. In 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 1998. San Jose, CA.

23. Standard, P.E.C., Spec CPU2000: Performance Evaluation in the New Millennium, Version 1.1. December 27, 2000.

24. TPC, Transaction Processing Performance Council. http://www.tpc.org/.

25. Trancoso, P., et al. The memory performance of DSS commercial workloads in shared-memory multiprocessors. In Third International Symposium on High-Performance Computer Architecture (HPCA). Jan. 1997.

26. Woo, S.C., et al. The SPLASH-2 Programs: Characteriazation and methodological considerations. In International Symposium on Computer Architecture. June 1995.

27. Zaki, M., C.-T. Ho, and R. Agrawal. Scalable Parallel Classification for Data Mining on Shared-Memory Multiprocessors. In IEEE International Conference on Data Engineering. March 1999.

28. Zaki, M.J., Parallel and Distributed Association Mining: A Survey. IEEE Concurrency, Special Issue on Parallel Mechanisms for Data Mining, Dec. 1999. **7**(4): p. 14-25.

29. Zhang, T., R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In SIGMOD. June 1996.

30. Ying Liu, Wei-keng Liao, and Alok Choudhary, Design and Evaluation of a Parallel HOP Clustering Algorithm for Cosmological Simulation. In 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22-26 April 2003, Nice, France.

31. Clementine, SPSS Inc. ─ http://www.spss.com/clementine/.

32. IBM Intelligent Data Miner, IBM Corporation ─ http://www.software.ibm.com/data/iminer.

33. SAS Enterprise Miner, SAS Corporation ─ http://www.sas.com/technologies/analytics/datamining/miner.