Optimizing Data Mining Workloads using Hardware Accelerators

Alok Choudhary Ramanathan Narayanan Berkin Özışıkyılmaz Gokhan Memik
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208
{*choudhar, ran310, boz283, memik*}*@eecs.northwestern.edu*

Joseph Zambreno
Dept. of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011
zambreno@iastate.edu

Jayaprakash Pisharath
Architecture Performance and Projections Group
Intel Corporation
Santa Clara, CA 95054
jayaprakash.pisharath@intel.com

Abstract

Data mining is the process of finding useful and actionable patterns in large data sets. Data mining algorithms have become vital to researchers in science, engineering, medicine, business, search and security domains. In recent years, there has been a tremendous increase in the size of the data being collected and analyzed. Data mining algorithms have been unable to scale up to these vast amounts of data, leading to significant performance degradation. Also, the enhancements in processor and system designs do not necessarily aid data mining workloads. In our previous work, we demonstrated that computational characteristics as well as data access requirements for data mining workloads are quite different than those of other common workloads. Therefore, there is a need to specifically address the limitations of accelerating data mining workloads. In this paper, we present a brief overview of the major challenges faced in data mining systems design. We first highlight important characteristics of these workloads. Then, we describe some initial designs and results for accelerating data mining algorithms using programmable hardware. Our results show that tremendous performance gains can be obtained by accelerating these workloads when compared to using traditional systems.

1. Introduction

Data mining is a powerful technology that converts raw data into an understandable and actionable form, which can then be used to predict future trends or provide meaning to historical events. Originally limited to scientific research and medical diagnosis, these techniques are becoming central to a variety of fields including marketing and business intelligence, biotechnology, multimedia, and security. As a result, data mining algorithms have become increasingly complex, incorporating more functionality than in the past.

According to a recent survey, the use of digital technologies is fueling data growth, which is doubling every two years akin to Moore's law for data [5]. This growth has posed several challenges to conventional data mining techniques. Users request more information to be extracted from their data sets, which requires increasingly complicated algorithms. Also, in many cases, the analysis needs to be done in real time to reap the actual benefits. For instance, a security expert would strive for real-time analysis of the streaming video and audio data in conjunction. Performing run-time analysis on such data sets appears to be the next big challenge in computing. On the other hand, recent computing trends suggest that the system performance, 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 two years. The important obstacle is the fact that the performance of computer systems is improving at a slower rate when compared to the increase in the data and the requirements of data analysis. Hence, there is a need to redesign and customize systems with respect to data mining applications.

There are mainly two types of enhancements to achieve this goal: evolutionary changes to the general-purpose systems and applications to increase their efficiency or revolu-

Table 1. Top three kernels of applications in MineBench and their contribution to the total execution time

Application	Top 3 Kernels (%)			Sum[%]
	Kernel 1 (%)	Kernel 2 (%)	Kernel 3 (%)	
k-Means	distance (68%)	clustering (21%)	minDist (10%)	99
Fuzzy k-Means	clustering (58%)	distance (39%)	fuzzySum (1%)	98
BIRCH	distance (54%)	variance (22%)	redistribution (10%)	86
HOP	density (39%)	search (30%)	gather (23%)	92
Naive Bayesian	probCal (49%)	varience (38%)	dataRead(10%)	97
ScalParC	classify (37%)	giniCalc (36%)	compare (24%)	97
Apriori	subset (58%)	dataRead (14%)	increment (8%)	80
Utility	dataRead (46%)	subsequence (29%)	Main (23%)	98
SNP	compScore (68%)	updateScore (20%)	familyScore (2%)	90
GeneNet	condProb (55%)	updateScore (31%)	familyScore (9%)	95
SEMPHY	bestBrnchLen (59%)	expectation (39%)	lenOpt(1%)	99
Rsearch	covariance (90%)	histogram (6%)	dbRead (3%)	99
SVM-RFE	quotMatrx (57%)	quadGrad (38%)	quotUpdate (2%)	97
PLSA	pathGridAssgn (51%)	fillGridCache (34%)	backPathFind (14%)	99

tionary architectures/hardware designs targeting data mining. Either of these research directions require an understanding of the data mining applications. In our previous work, we have established a benchmarking suite of applications that we call MineBench, which incorporates algorithms commonly found in data mining [12]. We have analyzed the architectural properties of these applications to investigate the performance bottlenecks associated with them. Our studies show that data mining applications have a unique mix of high data rates combined together with high computation power requirements. We have observed that data mining applications regularly oscillate between computation-intensive and data-intensive phases. Our detailed study suggests that current processors and architectural optimizations need to be enhanced further in order to handle such unique data-intensive applications [8, 10, 14]. The gap between the expected performance for data mining applications and the delivered performance of processor architectures can be shortened if current computer architectures are optimized or redesigned to accommodate data mining applications.

Hardware acceleration of data mining algorithms is an attractive method to cope with the increase in execution times and can enable algorithms to scale with increasingly large and complex data sets. In this paper we describe a generic data mining system architecture which can be customized for specific applications. We also present designs and results for accelerating two sample applications using programmable hardware.

The remainder of this paper is organized as follows. In the following section we provide a brief overview of the related work in this area. In Section 3, we discuss a generic methodology for hardware acceleration of data mining algorithms. Section 4 presents our designs, implementations, and results for hardware acceleration of sample applications. Finally, the paper is concluded in Section 5 with a look towards some planned future efforts.

2. Related Work

There has been prior research on hardware implementations of data mining algorithms. In [4] and [13], k-Means clustering is implemented using reconfigurable hardware. Baker and Prasanna [2] use FPGAs to implement and accelerate the Apriori [1] algorithm, a popular association rule mining technique. They develop a scalable systolic array architecture to efficiently carry out the set operations, and use a 'systolic injection' method for efficiently reporting unpredicted results to a controller. In [3], the same authors use a bitmapped CAM architecture implementation on an FPGA platform to achieve significant speedups over software implementations of the Apriori algorithm. Compared to our designs, these implementations target different classes of data mining algorithms.

3. Generic Architecture for Data Mining Systems

In our previous work [14], we have observed that data mining applications stream in data at a high rate. However, they are different than pure data streaming applications, and

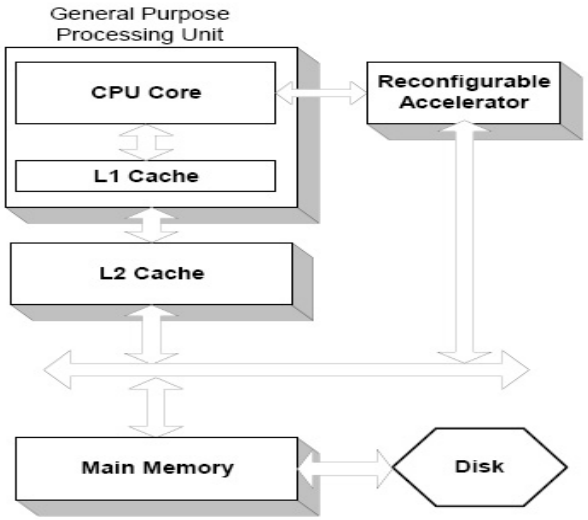

Figure 1. Data Mining Systems Architecture

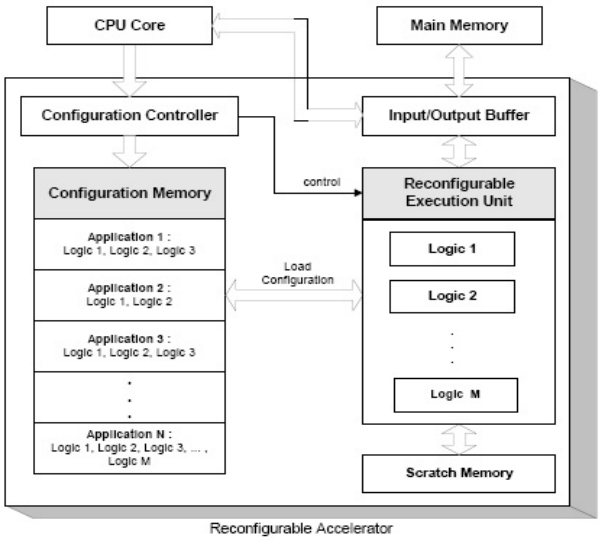

Figure 2. Design of the Reconfigurable Data Mining Kernel Accelerator

have phases. These phases contain important calculation steps, i.e., *kernels*. We have extracted the top 3 kernels for the applications in MineBench. The results can be seen in Table 1. For each application, the name of the kernel and the percentage of the system time spent executing the kernel are presented. In general, we see that most applications spend a majority of their time in small concise kernels. Identifying these kernels can lead to an understanding of the problems in the underlying hardware architectural components (i.e. processor, memory hierarchy, and other resources).

The results from our previous work [9] show that data mining applications have hit the performance wall of existing computing systems. In related areas of computing such as networks, graphics and physics processing, researchers have designed highly optimized architectures for their respective applications. Designing customized systems with high-speed data mining engines can help alleviate the performance degradation seen in conventional data mining applications.

In Figure 1, we present our generic design of the proposed data mining system architecture. In this system, we have the reconfigurable data mining accelerator as a co-processor that communicates with the general purpose processor. In this model, the processor can send the kernel operations to the accelerator (which executes the kernels faster than the processor) and the processor can continue with other non-kernel tasks. In Figure 2, we present details of the accelerator. In this model, when applications are loaded, their specific kernels should be loaded into the reconfigurable logic. Once the logics have been loaded, the execution unit hardly needs to be reprogrammed. This is due to the fact that the kernels remain the same for a given application. Only during an application change, the execution unit needs to be reprogrammed. The kernels for the applications are stored in the configuration memory, and their loading is triggered by the general purpose processor. Once the kernels are identified for each application, the hardware logic can be built and stored into the configuration memory by examining the underlying computations. The key to the efficient execution in this model is the implementation of the kernels. In the following section, we discuss a few examples where we design efficient architectures for these kernels.

4. Case Studies

4.1. K-Means and Fuzzy K-Means

K-Means is a clustering algorithm that 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, each object is assigned a nearest cluster based on a similarity function. Once the new assignments are completed, new centers are found by finding the mean of all the objects in each cluster. This process is repeated until some convergence criteria is met. In k-Means, the 'distance' kernel is responsible for calculating the Euclidean distance between two points and 'minDist' kernel calculates the minimum of the distances.

The 'clustering' kernel assigns the actual cluster and recalculates the centers (mean of points in a cluster) in each iteration. Fuzzy k-Means is closely related to k-Means, hence the distance calculation appears to be a prominent kernel for this application as well. In this case, the difference is that the clustering kernel is more time consuming than the distance calculation. This is because in fuzzy logic, the computations involved in performing the membership calculation (owing to multiple membership property) are more intense than the actual distance calculation. The 'fuzzySum' kernel is used during the convergence process. Figure 3 and Figure 4 show the hardware logic needed to implement the *distance* and *minimum* calculations respectively. The distance calculation logic, uses a level of N substractors followed by a set of N multipliers. The third level has a depth of log(N) and contains N-1 cumulative adders. The minimum computation involves a combination of multiplexers and comparator logic to compare and send the actual data item to the final level. In these designs the levels are tightly pipelined, allowing the results to be produced every cycle.

In the simulation of these designs, the accelerator has been attached to the overall processor, and we use an architecture-level cycle accurate simulator to measure the execution time. To enable the core processor to offload the kernel computations to the accelerator, markers have been attached in the actual application code. In our results, we have defined a new total cycle metric which contains the cycles spent by the core processor in non-kernel parts of the code including the handoff of computations to the accelerator plus the cycles the accelerator uses to calculate the results. We have tested our design with datasets of various sizes, and we have observed that as data set size increases, the speedups improve. This shows that the pipelined design becomes more effective when data set size increases, and shows that general purpose processors are not able to handle such streaming data efficiently. For k-Means and Fuzzy k-Means, we have seen speedups from 500x to 3600x and 400x to 1600x in the distance calculation kernel, respectively, and 600x to 1600x in minimum kernel in Fuzzy k-means. In the tests, the number of hardware resources have been varied, and it is clearly seen that application speedups scale well showing the applications exploit all the parallelism available to them. Overall, we have seen 11x to 80x speedup for k-Means and Fuzzy k-Means applications, respectively. The relatively lower speedups for the applications come from the fact that, when kernels are accelerated, the non-kernel parts of the applications become more dominant.

4.2. Decision Tree Classification

An important problem in data mining is Classification, which is the task of assigning objects to one of several predefined categories. A classification problem has an input dataset called the training set, which consists of a number of records, each possessing multiple attributes. Attributes may be categorical or continuous, depending on whether they have a discrete or continuous domain. The *classifying attribute* or *class ID* is a categorical attribute, whose value is known for records in the training dataset. A solution to the classification problem entails developing a model that allows prediction of the *class* of a record when the remaining attributes are known. Among existing solutions, Decision Tree Classification (DTC) is a popular method that yields high accuracy while handling large datasets. Poor scalability with increasingly large and complex data sets, as well as the existence of concise, well defined kernels make DTC a suitable candidate for hardware acceleration.

A decision tree model consists of internal nodes and leaves. Each of the internal nodes has a splitting decision and a splitting attribute associated with it. The leaves have a class label assigned to them. Building a decision tree model from a training dataset involves two phases. In the first phase, a splitting attribute and split index are chosen. The second phase uses this information to distribute records among the child nodes. This process is recursively continued until a stopping criterion is met. At this point, the decision tree can be used to predict the class of an incoming record, whose class ID is unknown. The prediction process is relatively straightforward: the classification process begins at the root, and a path to a leaf is traced by using the splitting decision at each internal node. The class label attached to the leaf is then assigned to the incoming record.

Determining the split attribute and the split index is a critical component of the decision tree induction process. In various optimized implementations of decision tree induction [11, 6], the splitting criteria used is to minimize the Gini index of the split. Previous work has shown that the largest fraction of the execution time of representative implementations is spent in the split determining phase [14]. For example, ScalParC [6], which uses a parallel hashing paradigm to efficiently map record IDs to nodes, spends over 40% of its time in the Gini calculation phase.

In our design of a hardware accelerator for DTC, we have chosen to accelerate the Gini score computation process. The Gini score is a mathematical measure of the inequality of a distribution. Computing the gini value for a particular split index requires computing the frequency of each class in each of the partitions. Therefore a linear search is made for the optimum split value, by evaluating the Gini score for all possible splits. This process is repeated for each attribute, and the optimum split index over all attributes is chosen. The total complexity of this operation is $O(|R| * |A|)$, where $|R|$ and $|A|$ represent the number of records and the number of attributes, respectively. Our architecture for acceleration DTC consists of several computation modules, referred to as 'Gini Units', that per-

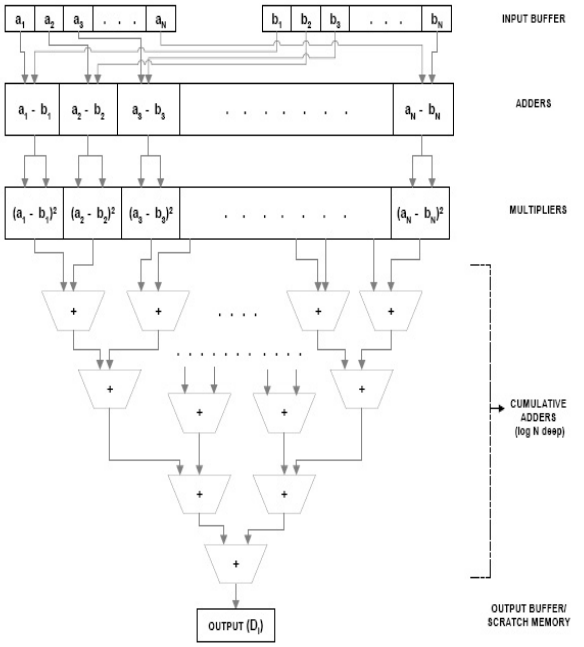

Figure 3. Distance calculation kernel

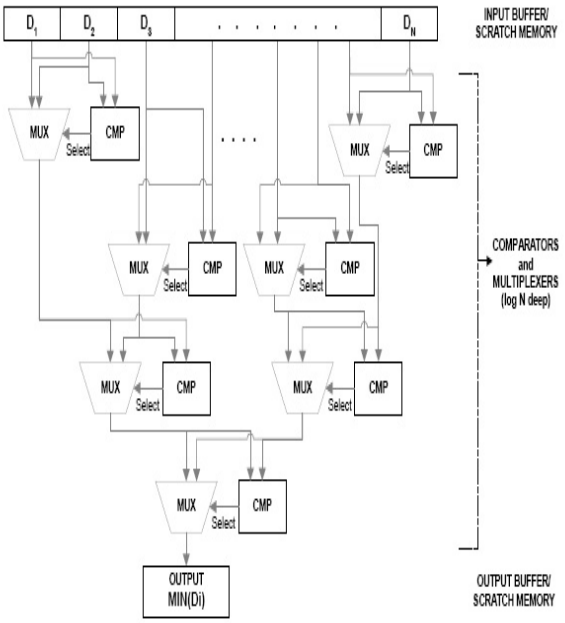

Figure 4. Minimum computation kernel

form Gini calculation for a single attribute. The high-level DTC architecture is presented in Figure 5. There is a DTC controller component that interfaces with the software and supplies the appropriate data and signals to the Gini units. The architecture functions as follows: when the software requests a Gini calculation, it supplies the appropriate initialization data to the DTC controller. The DTC controller then initializes the Gini units. The software then transmits the class ID information required to compute the Gini score in a streaming manner to the DTC controller. We apply a number of optimizations to make this hardware design efficient. Commonly, the class ID assumes only 2 values, '0' and '1'. Therefore, in hardware, only a single bit is sufficient to represent the class ID. This allows us to optimize the data transfer process to the Gini units. The class id information is stored in a bitmapped data structure which helps negate the bandwidth overhead generated while transmitting class IDs in the raw form. It is seen that this process of generating bitmaps can be done with very little overhead. Also, from a hardware perspective, we would like to minimize the number of computations and their complexity while calculating the Gini score. An implementation of the hardware in which the Gini score calculation is unaltered will be very complex and inefficient. A key observation is that the absolute value of the Gini score computed is irrelevant to the algorithm. It is only the split value and split attribute that are required. Therefore, we attempt to simplify the Gini computation to require minimal hardware resources, while generating the same value of split position and split attribute generated as earlier. We perform a series of manipulations to the Gini score calculation process itself, described in [7]. These changes dramatically reduce the complexity of an individual Gini unit, thus permitting a large number of attributes to be processed concurrently.

The DTC architecture was implemented on an Xilinx ML310 board [7], and its performance was compared with an optimized software implementation. Our architecture achieves a speedup of 5.58x over the software implementation when 16 gini units were used. The design also shows throughput scalability as the number of Gini units on board increases. We also measured the area occupied and clock frequency of our design. The experimental results strongly suggest that our system is scalable, and it will be possible to achieve higher speedups using larger-capacity FPGAs.

5. Conclusion

Data mining applications constitute a rapidly growing class of processor workloads. As data set sizes exponentially increase, conventional data mining applications are unable to scale up to the computational demands of these inputs. Hardware acceleration of data mining algorithms provides an attractive solution to this problem. In this paper, we propose a generic data mining system architecture using reconfigurable logic. Further, we design and implement hardware accelerators for two sample applications. The results indicate that our designs achieve significant speedups over software-only implementations, in addition to meeting area and bandwidth constraints. The success of these de-

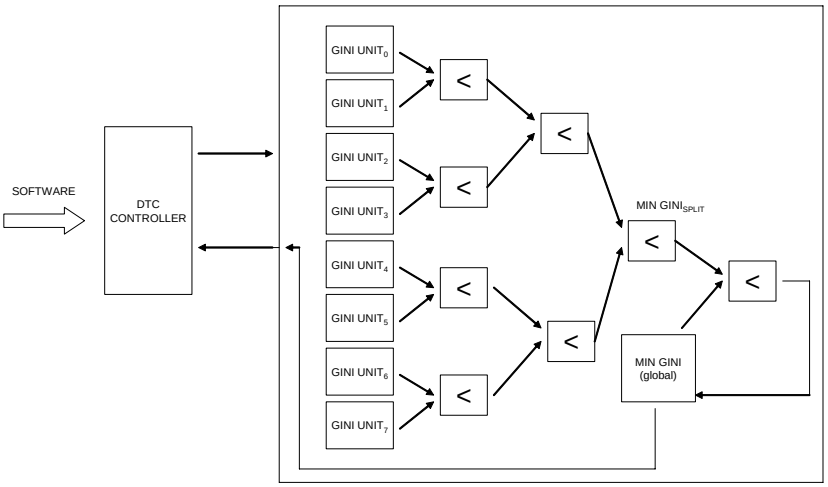

Figure 5. Architecture for Decision Tree Classification

signs make a strong case for further research on hardware accelerators for data mining applications.

Acknowledgments

This work was supported in part by NSF/CARP ST-HEC program under grant CCF-0444405, IIS-0536994, CNS-0551639, NSFs NGS program CNS-0406341, HECURA CCF-0621443, CCF-0546278, and in part by Intel Corporation.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast discovery of association rules. *Advances in Knowledge Discovery and Data Mining*, pages 307–328, 1996.

[2] Z. Baker and V. Prasanna. Efficient hardware data mining with the Apriori algorithm on FPGAs. In *Proceedings of the Thirteenth Annual IEEE Symposium on Field Programmable Custom Computing Machines (FCCM '05)*, 2005.

[3] Z. Baker and V. Prasanna. An architecture for efficient hardware data mining using reconfigurable computing system. In *Fourteenth Annual IEEE Symposium on Field Programmable Custom Computing Machines 2006 (FCCM '06)*, Apr. 2006.

[4] M. Estlick, M. Leeser, J. Szymanski, and J. Theiler. Algorithmic transformations in the implementation of k-means clusteringon reconfigurable hardware. In *Proceedings of the Ninth Annual IEEE Symposium on Field Programmable Custom Computing Machines(FCCM 01)*, 2001.

[5] Intel Corporation. Architecting the era of tera - technical white paper. Available at http://www.intel.com, 2005.

[6] M. Joshi, G. Karypis, and V. Kumar. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proceedings of the 11th International Parallel Processing Symposium (IPPS)*, 1998.

[7] R. Narayanan, D. Honbo, J. Zambreno, G. Memik, and A. Choudhary. An fpga implementation of decision tree classification. In *Proceedings of Design, Automation and Test in Europe (DATE)*, 2007.

[8] B. Ozisikyilmaz, R. Narayanan, J. Zambreno, G. Memik, and A. Choudhary. An architectural characterization study of data mining and bioinformatics workloads. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, Oct. 2006.

[9] J. Pisharath. *Design and Optimization of Architectures for Data Intensive Computing*. PhD thesis, Northwestern University, 2005.

[10] J. Pisharath and A. Choudhary. Design of a hardware accelerator for density based clustering applications. In *Proceedings of the International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2005.

[11] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. of the Int'l Conference on Very Large Databases (VLDB)*, 1996.

[12] The Center for Ultra-scale Computing and Information Security (CUCIS) at Northwestern University. NU-Minebench version 2.0. Available at http://cucis.ece.northwestern.edu, 2006.

[13] C. Wolinski, M. Gokhale, and K. McCabe. A reconfigurable computing fabric. In *Proceedings of the Engineering of Reconfigurable Systems and Algorithms (ERSA '02)*, 2004.

[14] J. Zambreno, B. Ozisikyilmaz, J. Pisharath, G. Memik, and A. Choudhary. Performance characterization of data mining applications using MineBench. In *Proceedings of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, Feb. 2006.