Quantization Error and Accuracy-Performance Tradeoffs for Embedded Data Mining Workloads

Ramanathan Narayanan, Berkin Özışıkyılmaz, Gokhan Memik,
Alok Choudhary, and Joseph Zambreno

Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208, USA
{ran310,boz283,memik,choudhar}@eecs.northwestern.edu

Abstract. Data mining is the process of automatically finding implicit, previously unknown and potentially useful information from large volumes of data. Embedded systems are increasingly used for sophisticated data mining algorithms to make intelligent decisions while storing and analyzing data. Since data mining applications are designed and implemented considering the resources available on a conventional computing platform, their performance degrades when executed on an embedded system. In this paper, we analyze the bottlenecks faced in implementing these algorithms in an embedded environment and explore their portability to the embedded systems domain. Particularly, we analyze the floating point computation in these applications and convert them into fixed point operations. Our results reveal that the execution time of five representative applications can be reduced by as much as 11.5× and 5.2× on average, without a significant impact on accuracy.

1 Introduction

Data mining algorithms have been successfully applied to predict trends in a variety of fields including marketing, biotechnology, multimedia, security, combinatorial chemistry, and remote sensing. As application-specific architectures become increasingly available, there is an urgent need to port data mining applications to embedded systems. The increased availability of embedded devices has led to a rapid increase in their usage in various fields. The level of intelligence demanded of these embedded systems require them to use complex and expensive data mining techniques. For example, a distributed traffic sensor system providing real-time information may consist of embedded devices with access to streaming data.

Data mining applications and algorithms are designed keeping in mind the ample computing power available on conventional systems. As a result, their performance on embedded systems is greatly hindered. In this paper, we study the amount of floating point calculations used by several data mining applications, and identify these as a major cause of poor performance of these algorithms on embedded environments. Further, we propose a solution to this problem by

Y. Shi et al. (Eds.): ICCS 2007, Part III, LNCS 4489, pp. 734–741, 2007.

replacing the floating point calculations with fixed point arithmetic. By doing so, we are sacrificing the high precision offered by floating point operations for the high implementation efficiency of fixed point computation. As data mining applications are used in critical sectors like healthcare and traffic sensors, it is imperative that we study the effects of our optimization techniques on the accuracy of these algorithms.

The remainder of this paper is organized as follows. In the following section, we present a brief overview of related work in this area. In Sect. 3, we present our methodology to convert a data mining application to use fixed point computations. A brief description of the data mining applications analyzed is provided in Sect. 4. The conversion procedure is described for each application in detail in Sect. 5, after which we provide the quantization and error analysis results. The paper is concluded in Sect. 6 with a look towards some planned future efforts.

2 Related Work

Our approach in this paper is similar to work done in the Digital Signal Processing [1,2] domain. In [2], the authors have used MATLAB to semi-automate conversion of floating point MATLAB programs into fixed point programs, to be mapped onto FPGA hardware. Currently our fixed point conversion is done manually. However, we support varying precisions and do not perform input scaling transformations. In [3], an implementation of sensory stream data mining using fixed point arithmetic has been described. The authors in [4] have used fixed point arithmetic with pre-scaling to obtain decent speedups for artificial neural networks used in natural language processing. Several data mining algorithms have been previously implemented on FPGAs [3,5,6]. In [5], the Apriori algorithm, which is nearly pure integer arithmetic, has been implemented on hardware. In [6], algorithmic transformations on K-Means clustering have been studied for reconfigurable logic.

3 Fixed Point Arithmetic

Fixed point representation uses a fixed number of digits to represent the integer and fractional parts of real numbers. We use the notation $Q.i.f$ to represent a fixed point variable of size $i + f$, with i digits used to represent the integer part and f digits used to represent the fractional part. The major stumbling blocks associated with fixed point arithmetic are *Overflow* and *Underflow*. Overflow occurs when a number is too large to be represented using the $Q.i.f$ format. The integer part of the fixed point number then wraps around and changes sign. Underflow, on the other hand, occurs when a number is too small to be represented using a fixed point notation, causing it to become zero.

3.1 Methodology

Our methodology for converting a data mining application using floating point arithmetic, to a fixed point application is described in Fig. 1. The first step in our

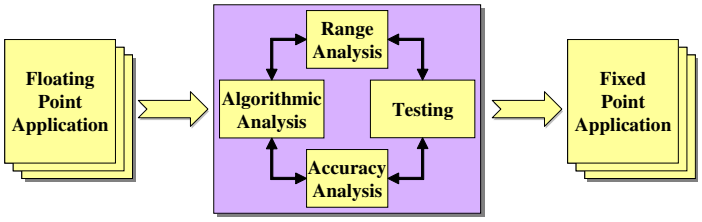

Fig. 1. Fixed Point Conversion Methodology

methodology is algorithmic analysis of the target application. In this step, we identify the functional blocks that are suitable for fixed point conversion. After a detailed algorithmic analysis and functional block identification, we apply a range analysis on the functional blocks. The purpose of range analysis is to determine the variables that may be susceptible to Overflow and Underflow errors. This step determines the various fixed point formats feasible and also identifies the various combinations of integer and fractional bits that are valid for the target application. In the accuracy analysis phase, we study the effects of differences between floating point operations and fixed point operations. We concentrate on the gradual loss of accuracy stemming from minor differences between fixed point and floating point operations. We may need to retain some of the critical components as floating point operations. In this phase, we may also have to reorder some of the calculations to optimize them with regard to fixed point calculations. This analysis procedure must be iterated several times until we obtain a fixed point representation that is *safe*, meaning that there are no critical errors. After the range and accuracy analysis are completed, we convert the data mining application to use fixed point operations.

4 Data Mining Applications

Data mining applications can be broadly classified into association rule mining, clustering, classification, sequence mining, similarity search, and text mining, among others. Each domain contains unique algorithmic features. In our study, we analyze applications belonging to four major domains: clustering, association rule mining, classification, and sequence mining. We have selected five applications from NU-MineBench, a data mining applications benchmark suite [7]. In our application selection, we have given priority to the applications that have the most floating point operations, since these are negatively affected while executing on embedded environments. Table 1 highlights the relative execution times on a conventional platform and an embedded system. The disparity in runtimes is due to the higher processor speed and dedicated hardware floating point unit available on the conventional (x86) platform (AMD Opteron, 2.4GHz), as compared to the embedded system (PowerPC). We also compute the fraction of floating point operations within the executed instructions [8], and surmise that there is significant scope for optimization by converting the

Table 1. Overview of the MineBench applications analyzed

Application	Inst Count (billions)	Floating Point Ops	Exec Time [x86] (s)	Exec Time [PPC] (s)
K-Means	53.77	19.87%	24.519	11145.89
Fuzzy	447.03	4.64%	443.7	57600.45
Utility	15.00	10.03%	9.506	482.21
ScalParC	5.47	9.61%	8.134	1553.82
PLSA	401.44	2.57%	136.67	2859.25

floating point operations to fixed point arithmetic. Detailed information about the K-Means, Fuzzy K-Means, Utility, ScalParC and PLSA applications and their corresponding datasets can be found in [7].

5 Conversion and Results

5.1 Experimental Setup

We performed our experiments on the Xilinx ML310, which is a Virtex-II Pro-based embedded development platform. It includes an XC2VP30 FPGA with two embedded PowerPC processors, DDR memory, PCI slots, ethernet, and standard I/O on an ATX board. We have 16KB separate, configurable, two-way set-associative instruction and data cache units. The operating frequency is 100MHz.

5.2 K-Means

Algorithmic analysis of `K-Means` reveals that a major fraction of floating point operations are due to Euclidean distance calculation. We performed a range analysis of the floating point operations, and determined the maximum and minimum values produced during computation. It is seen that at least 13 integer bits are required to avoid overflow, which generates negative values for distance and causes a critical error. Also, the input data requires precision of up to 10^{-3}, hence the binary representation of the number in fixed point notation must contain at least 12 fractional digits for accurate representation of the input data. Keeping this in mind, we find that the number of integer bits required by the fixed point representation for `K-Means` lies between 12 and 20.

The timing results for various fixed point implementation of `K-Means` are shown in Table 2. The results indicate that the fixed point versions run 9.1× to 11.6× faster than the floating point enabled version. The metric we use for accuracy analysis of `K-Means` is the 'membership' of each object to its cluster, as seen in Table 4. Here we study the percentage of points that change their cluster membership while varying the computation formats. The values obtained are well within reasonable error bounds for the `Q.16.16` and `Q.20.12` formats. The loss of precision is responsible for the larger error percentages in the `Q.24.8` case.

Table 2. Timing and speedup for K-Means

Type	Total
Floating point	11145.89s
Q16.16	9.06x
Q20.12	8.80x
Q24.8	11.59x

Table 3. Timing and speedup for Fuzzy

Type	Total
Floating point	3404.497s
Q12.20	1.46x
Q16.16	1.94x
Q20.12	3.19x
Q24.8	8.86x

Table 4. Relative error for K-Means

Num Clusters	Membership Error		
	Q16.16	Q20.12	Q24.8
5	1.52%	1.83%	2.44%
7	1.53%	1.58%	2.43%
9	1.55%	1.55%	2.09%
11	1.54%	1.62%	18.71%
13	1.61%	1.72%	4.65%

Table 5. Relative error for Fuzzy

Num Clusters	Membership			
	Q12.20	Q16.16	Q20.12	Q24.8
5	35.57%	1.69%	4.23%	4.85%
7	20.50%	0.35%	0.97%	1.87%
9	8.30%	0.17%	0.49%	1.18%
11	0.00%	0.12%	0.52%	3.33%
13	0.00%	0.08%	2.05%	2.89%

Considering various factors, it is seen that the Q.16.16 fixed point representation offers the best tradeoff between performance and accuracy. We also analyzed the difference in the cluster centers generated between the fixed point and floating point versions of the K-Means application. We notice that as the number of fractional bits increases from 8 to 16, the error in cluster centers decreases from 8% to 0.9%, for $k = 13$. In summary, K-Means can be executed using several fixed point representation formats to achieve significant speedups with minimal loss of accuracy.

5.3 Fuzzy K-Means

The major floating point computation intensive part in Fuzzy K-Means is the Euclidean distance calculation. Another important computation is that of the 'fuzzy membership' value and 'fuzzy validity' criterion. The Euclidean distance kernel discussed above, generates values that require 12 or more integer bits to avoid Overflow . The calculation of the 'fuzzy membership' value requires a fractional exponentiation computation, which is expensive to implement using fixed point computation. Hence, we make a design choice to compute this value using floating point variables. The choice of fractional bits for Fuzzy K-Means is slightly more flexible than the K-Means algorithm. Therefore the number of fractional bits needs only to be more than 10 in order to achieve reasonable results.

The timing results for Fuzzy K-Means, shown in Table 3, indicate significant speedups for the fixed point computation enabled versions. The speedup in execution time peaks at 8.86x for the Q.24.8 fixed point representation. To evaluate accuracy of the results, we analyze the percentage variation in the

Table 6. Timing and speedup for `Utility`

Support Value	Floating point	Q23.9	Q24.8
0.002	2863.02033s	1.27x	3.38x
0.004	1003.537s	1.18x	1.21x
0.008	571.711s	1.25x	1.32x
0.01	482.2119s	1.27x	1.59x
0.02	280.631s	1.38x	1.37x
0.03	367.3135s	1.37x	1.38x

Table 7. Average Relative Error for the total utility values of the points for various support values

Type/support	0.002	0.004	0.008	0.01	0.02	0.03
Q23.9	0.00147%	0.01100%	0.00314%	0.00314%	0%	N/A
Q24.8	0.02831%	0.02715%	0.00772%	0.00772%	1%	N/A

fuzzy-membership value. This value indicates the degree of membership of each object to its cluster. This value is shown in Table 5. We also analyze the differences in the cluster centers produced by the fixed point formats, and notice between 63% (for `Q.24.8`) and 0.73% (for `Q.12.20`) variation. Since the cluster centers are a derived attribute, we may compute them using higher precision floating point values. Therefore it can be seen that the optimum configuration for Fuzzy K-Means is the `Q.16.16` representation, which achieves significant speedup, with minimal loss of accuracy.

5.4 Utility Mining

Algorithmic analysis of `Utility mining` yields the calculation of 'Utility' value as the major floating point computation kernel. The range of Utility values generated using the dataset indicated that at least 23 integer bits would be required in the fixed point representation. Prevention of overflow is critical to the application and hence we are forced to choose fewer than 9 fractional bits. Fortunately, we have observed that other modules require accuracy of up to 10^{-2}, thus necessitating at least 8 fractional digits for successful termination of the algorithm. Consequently, we decided to use the `Q.23.9` or `Q.24.8` fixed point representation.

The speedup values for `Utility mining` shown in Table 6 reveal that a speed up of up to 3.38× is possible using the valid fixed point representations determined in Sect. 5. To measure the accuracy of the results, we compared the utility itemsets generated by the algorithm, for various values of minimum utility support. The results show that for lower values of minimum utility support, the `Q.24.8` format produces only 10 of 25 utility itemsets, whereas the `Q.23.9` fixed point format produces all the utility itemsets generated by the floating point version. However, as the minimum utility support increases, there is 100% corre-

Table 8. Timing and speedup for `ScalParC`

Type	I/O	Comp.	Total
Floating	74.70s	1458.82s	1553.52s
Q4.28	0.98x	1.27x	1.25x
Q8.24	0.98x	1.27x	1.25x
Q12.20	0.99x	1.26x	1.24x

Table 9. Timing and speedup for `PLSA`

Type	Total
Floating point	2859.25s
Q2.30	1.003x
Q1.31	1.014x
Q0.32	1.003x

spondence in the utility itemsets generated. Another measure of accuracy is the 'total utility' value of each utility itemset (Table 7). Percentage variation over the total utility value over the valid fixed point representation formats shows there there is insignificant error due to fixed point conversion.

5.5 ScalParC

`ScalParC` has a large percentage of floating point operations (9.61%), thus hindering performance on embedded systems. All the expensive floating point operations are done in the 'Calculate Gini' module, which is the most compute intensive module of `ScalParC`. By using a fixed point variable to store the 'Gini' value and simple reordering of the computations, we were able to avoid significant floating point data type casting overheards.

The timing results for various fixed point implementation of `ScalParC` are shown in Table 8. The fixed point operations achieves a execution time speedup of 1.25x and 1.24x for the Q12.20 and Q4.28 configurations, respectively. We have also compared the accuracy of various implementations by examining the number of nodes in the generated decision tree at each level and discovered that with 20 fractional bits, one out of 697 splits was not performed. Our results reveal that the accuracy increases with increasing precision, however, the output changes for the ScalParC are in general negligible.

5.6 PLSA

We also analyzed and converted the `PLSA` application, which has a relatively low percentage of floating point operations. The main floating point operation lies in the area calculation module. Because of the large values of area, we can have only 0-2 fractional bits. The algorithm has been modified so that the implementation does not need any fractional bits, and a multiplication with 0.1 has been replaced by division by 10.

The timing results for various fixed point implementations of `PLSA` are shown in Table 9. In general, the performance improvement of the fixed point conversion is small. Higher speed-ups may be achieved when using larger data-sets, because there will be a higher fraction of floation point operations. To analyze the accuracy of the converted application, two metrics have been considered. The first one is the sequence alignments. The alignments are exactly the same in all the runs, which shows that the application has executed correctly. Another

metric is the total area of the blocks solved in the dynamic programming approach. Our analysis has shown that the difference between these values is also negligible.

6 Conclusions

In this paper, we have described a method for implementation of data mining algorithms on embedded systems. Data mining algorithms are designed and implemented for conventional computing systems, and show poor performance while executing on an embedded system. We applied our methodology to several representative data mining applications and have shown that significant speedups can be achieved. We also quantized the error in each case and determined the optimum configurations for fixed point implementation. Particularly, with our fixed point conversion methodology in 4 out of the 5 applications, we achieve significant speedups, as much as 11.5x and 5.2x on average, with minimal loss of accuracy. As embedded data mining assumes importance in various key fields, our methodology will serve as a starting step towards efficient implementations.

References

1. Menard, D., Chillet, D., Charot, F., Sentieys, O.: Automatic floating-point to fixed-point conversion for DSP code generation. In: Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES). (October 2002)
2. Roy, S., Banerjee, P.: An algorithm for trading off quantization error with hardware resources for MATLAB-based FPGA design. IEEE Transactions on Computers **54**(7) (July 2005) 886–896
3. Cai, Y., Hu, Y.X.: Sensory steam data mining on chip. In: Second NASA Data Mining Workshop: Issues and Applications in Earth Science. (May 2006)
4. Ferrer, D., Gonzalez, R., Fleitas, R., Acle, J.P., Canetti, R.: NeuroFPGA - implementing artificial neural networks on programmable logic devices. In: Proceedings of Design, Automation and Test in Europe (DATE). (February 2004)
5. Baker, Z.K., Prasanna, V.P.: Efficient parallel data mining with the Apriori algorithm on FPGAs. In: Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM). (April 2005)
6. Estlick, M., Leeser, M., Theiler, J., Szymanski, J.J.: Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware. In: Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA). (February 2001)
7. Narayanan, R., Ozisikyilmaz, B., Zambreno, J., Memik, G., Choudhary, A.: MineBench: A benchmark suite for data mining workloads. In: Proceedings of the International Symposium on Workload Characterization (IISWC). (October 2006)
8. Zambreno, J., Ozisikyilmaz, B., Pisharath, J., Memik, G., Choudhary, A.: Performance characterization of data mining applications using MineBench. In: Proceedings of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW). (February 2006)