High Performance Data Mining Using R on Heterogeneous Platforms

Prabhat Kumar, Berkin Ozisikyilmaz, Wei-Keng Liao, Gokhan Memik, Alok Choudhary
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL, USA
{*pku649, boz283, wkliao, memik, choudhar*}*@ece.northwestern.edu*

***Abstract*—The exponential increase in the generation and collection of data has led us in a new era of data analysis and information extraction. Conventional systems based on general-purpose processors are unable to keep pace with the heavy computational requirements of data mining techniques. High performance co-processors like GPUs and FPGAs have the potential to handle large computational workloads. In this paper, we present a scalable framework aimed at providing a platform for developing and using high performance data mining applications on heterogeneous platforms. The framework incorporates a software infrastructure and a library of high performance kernels. Furthermore, it includes a variety of optimizations which increase the throughput of applications. The framework spans multiple technologies including R, GPUs, multi-core CPUs, MPI, and parallel-netCDF harnessing their capabilities for high-performance computations. This paper also introduces the concept of interleaving GPU kernels from multiple applications providing significant performance gain. Thus, in comparison to other tools available for data mining, our framework provides an easy-to-use and scalable environment both for application development and execution. The framework is available as a software package which can be easily integrated in the R programming environment.**

***Keywords*-R; GPU; Data Mining; MPI; K-Means; Fuzzy K-Means; PCA; Parallel-netCDF;**

I. Introduction

Knowledge driven decisions are a key to success in today's world. Business corporations, financial institutions, government departments, research and development organizations collect huge amounts of data with a view to gain a deeper insight in their respective fields. Social networks such as Facebook and micro-blogging website Twitter generate enormous amounts of data which can provide useful information about the latest trends in the society. Sifting through such vast collection of data and discovering unknown patterns is not a trivial task, especially when the data sizes are of the order of exabytes and petabytes. Data mining presents a pool of automated analysis techniques which can discover hidden knowledge and predict new trends and behaviors.

Analyzing large quantities of data requires computational resources. Recent times have seen the emergence of many high performance architectures like GPGPUs, Cell, Multi-cores, FPGAs, etc., each presenting its own unique benefits. The paradigm of homogenous computing, where all the nodes have the same architecture, is transforming itself to heterogeneous computing, where each task is allocated to the architecture that suits its properties best. Since data mining kernels are characterized as being computationally intensive, the new generation of architectures can provide a significant boost to their performance. Furthermore, storing and retrieving large quantities of data adds to the complexity of data mining applications.

Exploring hidden patterns and trends require a collection of data mining techniques. Tools such as Clementine[1] and WEKA[2] provide a rich collection of algorithms. However, they lack the capability to utilize the benefits of co-processors and do not have scalable I/O capabilities. This limits their usability as a high performance data analytics tool. This paper describes a scalable framework for developing parallel applications on a heterogeneous computational backbone. It incorporates a library of compute-intensive kernels and explores performance optimization techniques

1530-2075 2011

DOI 10.1109/IPDPS.2011.329

to increase the throughput of applications. In our framework, an application is written as a script which is composed of modules (e.g., commonly used kernels). The framework provides a middleware which deploys these modules on a cluster of heterogeneous hardware platforms. Further, processing huge amounts of data requires reading and writing to storage devices, like disk drives, SSDs etc. I/O presents a significant bottleneck in the overall performance of data mining applications as a poor read/write interface can hinder any benefit obtained from parallel architectures. To alleviate this problem, our framework incorporates a parallel I/O interface. Thus, the framework discussed in this paper provides parallelism both for I/O and computations while still being simple and flexible.

Besides the above mentioned features, the proposed framework outlines a new optimization technique aimed for GPU architectures. This technique involves interleaving kernels from different applications to improve their throughput. The optimization relies on the domain specific knowledge that it is not always known apripori, what is the best algorithm to mine raw data for useful information. In such situations the data is explored using multiple algorithms. Since all the algorithms work on the same dataset, they can run in close coordination to improve the overall performance. Overall, the major contributions of the paper are as follows:

1) A scalable framework for writing high performance applications on heterogeneous platforms.
2) A high performance library of commonly used kernels for data exploration.
3) An interface to parallel I/O functionality
4) Various optimizations to increase the throughput of applications

The paper is organized as follows. Section II presents the related work. Section III presents the implementation overview of our framework. Section IV describes how applications can be written for the framework. Section V presents a discussion of the results. We conclude the paper in Section VI with directions to future work.

II. Related Work

R[3] is a widely used programming language for statistics and data manipulation. Given that huge statistical problems have become commonplace today, a number of parallel R packages have been developed. A few such packages for explicit parallelism are mentioned below.

The Rmpi[4] package provides an interface from R to MPI. The SNOW[5] package runs on top of Rmpi (or directly via sockets), allowing the programmer to express the parallel disposition of work more conveniently. Rdsm package[6] gives the R programmer a shared memory view, but the objects are not physically shared. Instead, they are stored in a server, and accessed through network sockets, thus enabling a threads-like view. Parallel-R[7] and pR[8] enable the statistical analysis routines available in R to be deployed on high performance architecture.

The M0BNI Microarray Lab in University of Michigan has provided a gputools[9] package for R which constitutes a handful of common statistical algorithms that are used in the biomedical research community. Another work in this area is the RGPU package which enables parallel evaluation of linear algebra expressions, as well as access to some of the function provided in CUDA SDK[10]. The magma[11] package provides an interface to the hybrid Matrix Algebra on GPU and Multicore Architectures implementation.

There has been multiple works in using cluster of GPUs in parallel. DisMaRC, a distributed GPGPU based MapReduce framework is presented in [12]. In another work by Lawlor[13], the author analyses two new communication libraries, cudaMPI and glMPI, that provide an MPI-like message passing interface to communicate data stored on the graphics cards of a distributed-memory parallel computer. Then there are numerous examples of single applications that are ported on cluster of GPUs [14], [15], [16], [17]. In the data mining domain GPUs have been used extensively. Some implementations of K-Means on GPUs can be found in [18], [19], [20], [21], [22], [23]. Our work focuses on low-level and

micro-level performance optimizations which are explored using a library of customized kernels incorporated in a scalable framework. In this work we focus on a bottoms-up approach of developing high performance applications using simpler kernels. This is in contract to the various works mentioned above which follow a top-down approach. Furthermore, we stress on the overall throughput of applications as opposed to scaling independent applications.

III. Implementation Overview

The framework presented in the paper spans across different domains to harness their capabilities in an attempt to provide a scalable system for using data mining applications for knowledge discovery. This section gives a detailed description of the different components used in the framework. The front end of the framework is the widely used R-statistical tool. MPI is used for communication between the cluster nodes and parallel I/O is achieved using MPI-IO and Parallel-netCDF[24] interface. The computation intensive tasks are handled by multi-core CPUs and multi-threaded GPUs [25], [10]. Figure 1 shows the different components of the system.

Figure 2 shows the dataflow in our framework. The framework is launched on all the nodes in a master-slave configuration. The application is written as an R script. The script calls the high-performance I/O interface to read the data from platform independent netCDF[26] file in parallel.

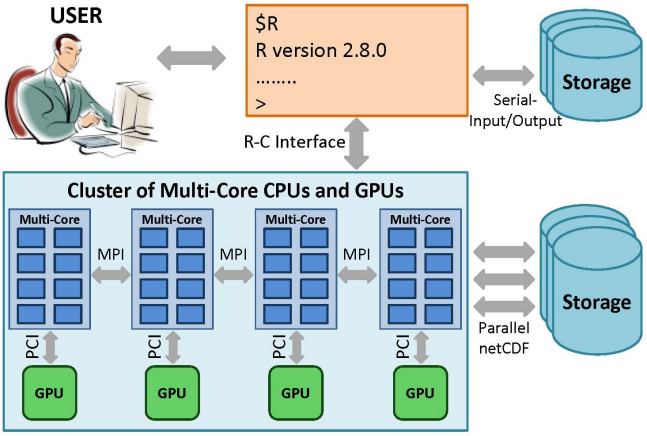

Figure 1: Overview of the Framework

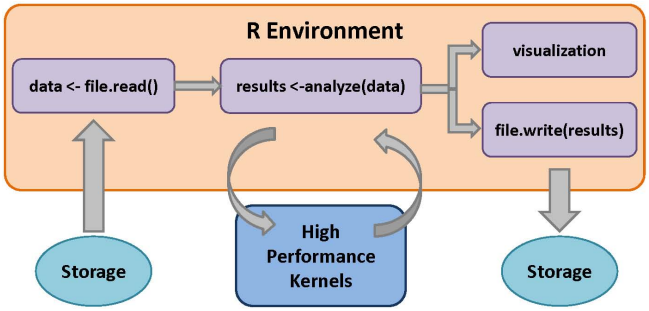

Figure 2: Dataflow in the framework

Once the data is read, the script invokes high-performance kernels, through an efficient R-C interface, to analyze the data. The MPI communication enables the nodes to interact with each other. The results, which are communicated to the R environment, can take advantage of the rich analysis and/or visualization tools available in R. The programming model consists of - (1) a programming infrastructure and (2) a library of high performance kernels. The former provides tools and methodologies for developing scalable applications while the latter provides a collection of commonly used data mining kernels to accelerate application development.

A. Programming Infrastructure

The programming infrastructure provides a software platform which presents different methods for managing the various components with a view of scalable implementation and a high performance scripting interface for an easy-to-use front end to write various applications.

1) Software Platform: As mentioned above, we have four major components in our framework: the front end (or the scripting interface), the back end (managed by C/C++/CUDA), communication (which is MPI) and the I/O. Broadly defined, there are two different implementation methods for gluing these components to have a scalable platform while keeping it flexible enough to incorporate different kernels. These methods are described in the following. The first implementation, which is referred to as C-level parallelism (C-LP), is shown by dotted arrows in Figure 3. In this, the MPI communication is not visible in the R environment. Each of the nodes running R call

the corresponding C interface functions and all the MPI calls are handled at the C level. The other way of implementation, called R-level Parallelism (R-LP), in which the MPI communication is visible at the R environment, is shown by solid arrows in Figure 3. The R nodes call the C interfaced kernels and the communication among the nodes is handled in R. Notice that the C kernels are serial as opposed to MPI-enabled kernels in C-LP.

Both the implementations have pros and cons. R-LP has a higher overhead in sharing data among the nodes as the data needs to come to the R environment and then communicated to other nodes before finally filtering down to the C environment, as opposed to C-LP where the data can be shared at the same level i.e., across the C environment as shown in Figure 3. Secondly, C applications/kernels which are already written using MPI paradigm can be directly interfaced to the R environment with no or little modifications. On the contrary, R-LP requires application to be written as R script. The limitation faced by C-LP is that it requires all the development to be done within Rmpi package, i.e., all the code needs to be compiled with the Rmpi code. The reason lays in the fact that MPI initialization can be done only once for whole system which makes it impossible for packages not compiled within Rmi to use

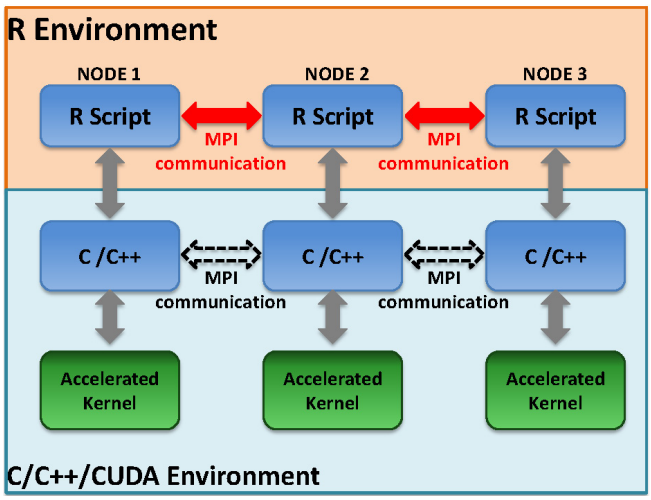

Figure 3: C-level Parallelism (C-LP) - Parallelism embedded at the C environment (dotted arrows) and R-level Parallelism (R-LP) Parallelism exposed to the R environment (solid arrows)

MPI function calls. R-LP is more flexible in this regard as high performance library packages can be developed independent of the Rmpi package. Both these approaches have been followed in the framework for different components. As discussed in later sections, parallel I/O interface is built upon C-LP, while the kernel libraries and application development follow the R-LP methodology.

2) High-Performance R: The programming infrastructure of our framework includes a high-performance scripting language capable of being used in a distributed computing environment. This scripting language interface is based on the widely used statistical tool R. However, since R is not good for heavy lifting, an interface to high level languages like C/C++/Fortran, known for their computational capabilities, is provided. Furthermore, since all the accelerator/coprocessors have an interface to high-level languages an efficient R-C interface is necessary for true high performance scripting capabilities. R serves as a front-end interface to the user. Compiled C functions can be invoked in the R environment using .C or .Call interface functions. With .C, the R objects are copied to C data structure before being passed to the C code, and copied again to a R list object when the compiled code returns. On the contrary .Call does not copy arguments. Since data mining algorithms process huge amounts of data, copying of arguments can severely hamper the performance of applications. Our framework uses .Call function to provide the C interface to R. Also, we havent noticed any degradation in the execution of the C functions using .Call interface. Besides lower amount of copied data, other advantages of using .Call function include:

- The ability to dimension the answer in C code
- Access to the attributes of the vectors
- And, access to other types, e.g., expressions and raw type

These advantages come at the cost of increased complexity in writing the interface functions.

B. High-Performance library of Kernels

The second component of our frameworks programming model is the library of optimized and

Table I: Kernels for large data using CUDA Streams

Functionality	Kernel Interface in R	Description
Summation	*gpu.stats.sum*	*Input*: vector of elements; *Output*: sum
Minimum	*gpu.stats.min*	*Input*: vector of elements; *Output*: min
Maximum	*gpu.stats.max*	*Input*: vector of elements; *Output*: max
Variance	*gpu.stats.var*	*Input*: vector of elements; *Output*: variance
Histogram	*gpu.stats.hist*	*Input*: vector of elements, bins; *Output*: histogram
Distance Computation	*gpu.distance_compute*	*Input*: N records, K points; *Output*: Membership of each record
Cluster Update	*gpu.cluster_update*	*Input*: N records and membership; *Output*: Location of all centres
Weighted Cluster Update	*gpu.w_cluster_update*	*Input*: N records and membership; *Output*: Weighted location of all centres
Eigenvalue Computation	*gpu.eigenvalue*	*Input*: NxD Matrix; *Output*: Eigenvalue (using Householder and Bisection)
Eigenvector Computation	*gpu.eigenvector*	*Input*: NxD Matrix, eigenvalues; *Output*: Eigenvectors (using Inverse Iteration)
Parallel netCDF file read	*ncmpi.file.read*	Read data from netcdf file parallely
Parallel netCDF file write	*ncmpi.file.write*	Write data to newcdf file parallely
MPI-IO file read	*mpi_io.file.read*	Read data from a binary file using MPI
MPI-IO file write	*mpi_io.file.write*	Write data to a binary file using MPI

high performance kernels which can be embedded in R scripts. The library provides a collection of commonly used data mining kernels implemented for different architectures. Apart from this, intra-node and inter-node optimizations are also included. To keep the development process simple and flexible, we follow the R-LP approach (refer to Figure 3). Decoupling the high performance kernels from applications gives us the opportunity to develop new applications. Furthermore, kernels implemented on different architectures enable us to explore the design space to achieve the best performance.

1) Computational Intensive Kernels: We have implemented the kernels both for CPU and GPU. CPU kernels are used for hybrid-execution on a heterogeneous cluster comprising of GPUs and CPUs. For CPU, some kernels are already available in R. The implementation is done keeping in view that the kernels can easily scale in a cluster environment. For the GPU implementations, the input data is first shipped to the GPU device memory and then kernels are launched which process the input data in the device memory. The results are subsequently shipped back to the host (CPU) memory. Table I shows a list of kernels and their corresponding interface functions for R.

2) Kernel optimization for GPUs: The above mentioned kernels work well when the input data fits entirely into the GPU device memory. However, since data mining deals with huge amounts of data, typically, the entire data will not fit into the GPU device memory. Transferring data to and from the GPU device will result in significant performance degradation. This requires out-of-core implementation using CUDA Streams. The framework, however, uses the multi-threaded kernels (mentioned above) and schedules them to overlap with host/device data transfers. This limits the need of developing new out-of-core kernels. Figure 4 shows an example where the input dataset is divided into smaller tiles and assigned to two different streams. Each data transfer on *Stream 1* is overlapped with a kernel execution on *Stream 2* resulting in reduced overhead.

3) Communication + I/O: In our framework the communication among the nodes of a cluster is handled by MPI though the Rmpi package. However, besides sharing the data during computations, large amounts of data need to be accessed from storage devices. Absence of a parallel I/O interface will severely affect any performance gain achieved using multi-core multi-threaded kernels. We, therefore, enhace the capabilities of Rmpi package to provide MPI-IO interface to R for parallel read/write capability. Further, parallel-netCDF is built on top of MPI-IO. We have implemented a parallel-netCDF interface for R which

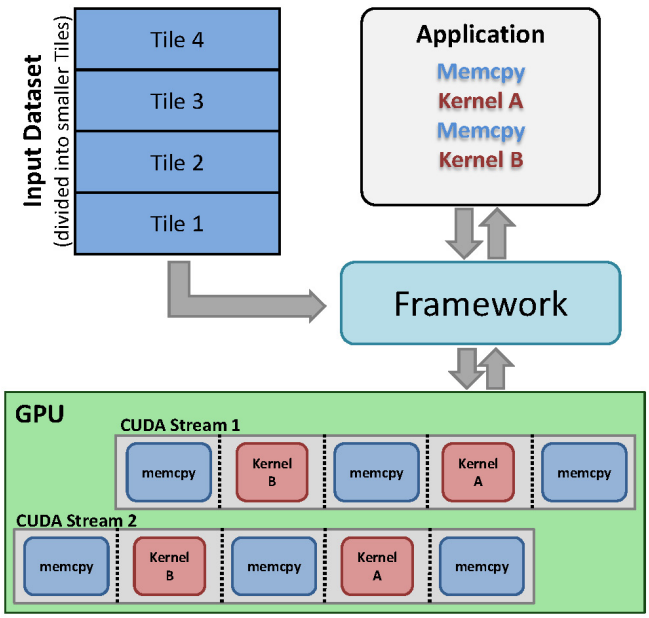

Figure 4: Kernels for large data using CUDA Streams

provides the capability of reading/writing netcdf file format to all the nodes in the R-cluster. Table I gives a list of interface functions for parallel read/write.

IV. Application Development Using the Framework

Previous section has given a detailed description of the different components that make our framework. In this section, we present how applications can be written using all these components. We divide this section into three subsections discussing about the implementation of algorithms using the kernels, the optimizations offered by the framework, and how to scale the applications to a cluster of nodes. Notice that application development is done on the front end in R script and the kernels and I/O functions are called only when necessary.

A. Algorithms

Using the framework, we have developed different data mining algorithms. Due to limited space we give only a brief description of three of them: K-Means[27], Fuzzy K-Means[28], [29] and PCA[30], [31], [32], [33]. K-Means is a widely used clustering algorithm which attempts to find K partitions of the input dataset by minimizing the squared error within each partition. The K-Means algorithm can be implemented using the Distance Computation, Cluster Update and Histogram kernels as mentioned in Table I. A variation of K-Means algorithm called Bisection K-Means can also be implemented similarly. Fuzzy K-Means is a superset of K-Means algorithm with the distinction that it allows each record in the data set to have a degree of membership to each partition. Principal component Analysis (PCA) aims at finding the principal components which are representative of the input dataset and can be implemented using Eigenvalue and Eigenvector kernels.

B. Scheduling Optimizations using the Framework

Besides the above mentioned kernels, our framework provides a number of optimizations which can help increase the speedup of the applications. We present a couple of optimizations here. Notice that these optimizations are currently specific to the hardware but as new hardware devices are introduced, new optimizations for that particular hardware are easy to integrate in the current system.

1) Hybrid Implementation: Hybrid implementation refers to harnessing the capabilities of both

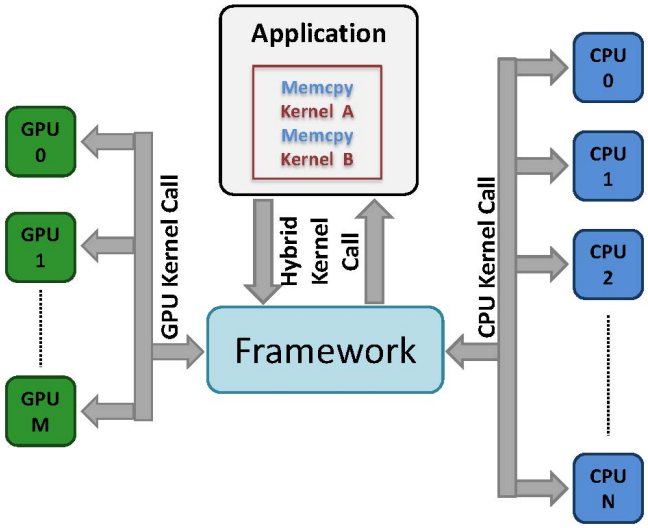

Figure 5: Distribution of tasks in a Heterogeneous environment

GPUs and CPUs simultaneously. Consider a situation when we have a GPU and a multi-core CPU in the system. It would be desirable to distribute the tasks between the GPU and the CPU cores. Since the computational power of GPU is significantly higher than that of the CPUs, the data need to be distributed such that the work remains balanced. Our framework provides the functionality to run an application in the hybrid mode. In this mode, the data will be distributed among the nodes and the corresponding CPU or GPU kernels will be launched. Figure 5 shows how a hybrid kernel call gets broken down into architecture specific kernel calls using the framework.

2) Multiple Kernel Optimization: This optimization is specific to the CUDA implementation. We notice that data mining kernels process huge amounts of data and it is not always possible to fit the entire data in the GPU device memory. As mentioned in Section III-B2, this will require the usage of CUDA Streams to lower the overhead caused by copying data from host memory to the device memory. We further notice in our experiments that kernel execution time is smaller

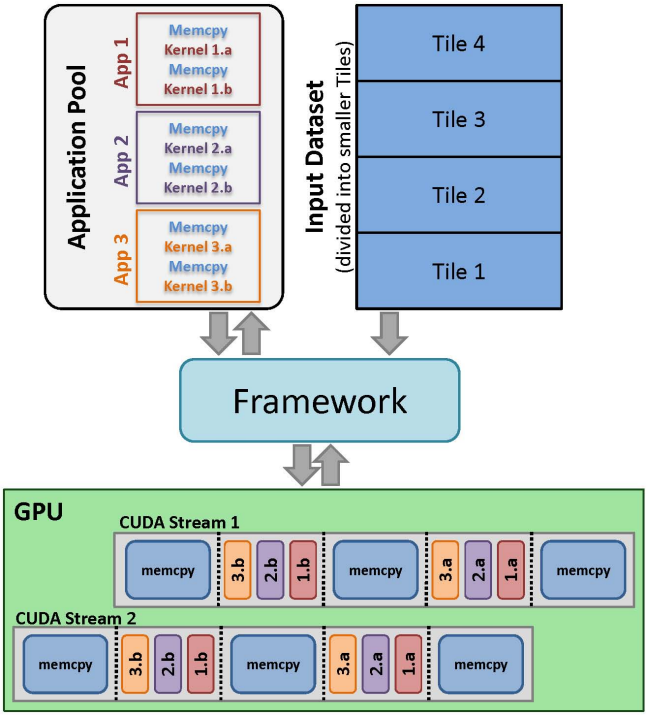

Figure 6: Concept of Interleaved Kernel Optimization

than the time it takes to copy smaller tiles of data to the GPU device memory. This presents us a unique opportunity to leverage the time difference. In practical situations, a number of different data mining algorithms are used on a given dataset. We propose to run kernels from different applications on the dataset while it is in the device memory so as to reduce the overhead of memory copy as much as possible. Figure 6 shows the idea behind this optimization. As an example, three different applications App1, App2, and App3 are shown in the figure. For each memory transfer call put on a CUDA Stream, one kernel call from each of the applications (1.a, 2.a, and 3.a) is allocated on that particular stream as shown. This can be viewed as a single kernel whose execution time is close to the combined execution time of the same kernels running separately. The kernel execution and host-device memory copy times can be used to predict the number of applications which can be interleaved in the above fashion.

V. Experimental Results

In this section, we evaluate the performance of the applications developed using the framework as well as various proposed optimizations. On the hardware side, we have a cluster of 4 nodes. The host CPU on each node is an Intel Quad Core 2.4 GHz processor with 4 GB of main memory. The co-processor on each node is NVIDIAs GeForce 8800GT Graphics Processing Unit with 112 processing cores and 512 MB of device memory. On the GPU, the grid of thread blocks can have a maximum of 65535 blocks on each dimension, with a maximum of 512 threads per block. Each multiprocessor has 16 KB of shared memory and can run 8 thread blocks concurrently. Each node has two of these GPUs. The software setup includes R version 2.8.0. MPICH2 version 1.2.1 is used for providing the MPI communication. To provide parallel-netCDF functionality Pnetcdf library version 1.2 is used. The GPU kernels are compiled using the CUDA compiler driver, NVCC, release 2.0. The entire software framework is compiled using GCC version 4.4.2.

A. *Performance of Applications*

Figure 7 shows the performance of clustering algorithms like K-Means and Fuzzy K-Means for 20 clusters and different sizes of the input data set ranging from 10K to 1 million records. We notice that as the data size is increased, there is an initial improvement in the speedup, which eventually saturates at around 40 for K-Means and 70 for Fuzzy K-Means. The performance difference between the two algorithms occurs because of the computationally intensive membership calculation resulting in higher speedups for larger workloads. Due to limited space we cannot provide performance charts for other applications. However, for basic statistical kernels we obtain a speedup of up to 30x and for PCA the speedups achieved are of the order of 35x when compared to a single CPU implementation.

Figure 8 shows the performance results for larger datasets when the entire dataset does not fit into the GPU device memory. Data is divided into smaller tiles of size 768K records and the number of CUDA Streams used is 4. We notice that the speedup in this case is somewhat smaller as compared to K-Means in Figure 7. This can be attributed to two reasons. First, since the data transfer time (between CPU and GPU) is not negligible, copying data for all iterations incurs extra overhead lowering the performance gains. Second, since the kernels take less time compared to the memory transfer, this overhead cannot be eliminated even using CUDA Streams. Hence, the speedup saturates around 25 compared to a speedup of 40 in Figure 7.

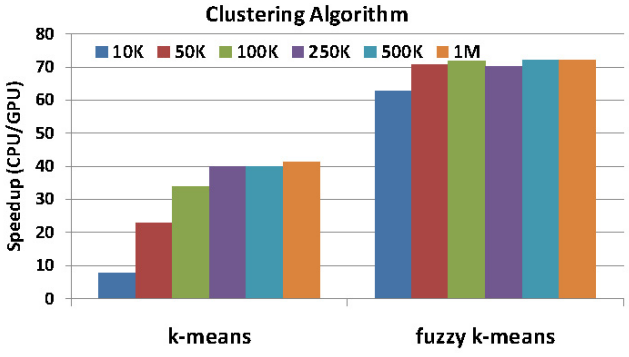

Figure 7: Performance improvement for K-Means and Fuzzy K-Means for K=20

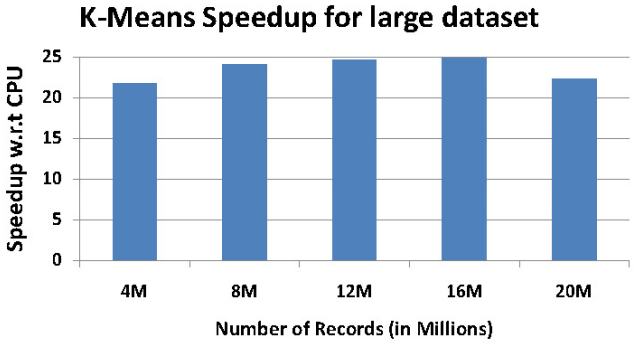

Figure 8: K-Means implementation for large dataset

B. *Effects of Scheduling Optimizations*

In Figure 9, we show the performance gain achieved by interleaving different kernels. As an example, we consider K-Means with different number of clusters (which is a parameter for K-Means) as different applications. The results show the speedups obtained for 2 to 7 applications relative to a single application for data sizes ranging from 4 million to 20 million records. We notice that as more kernels are interleaved for the same amount of data transferred to the device memory, the speedup increases. The speedup increases from 1.6x for two applications and saturates around 2x as the number of applications are increased beyond 4. It should be noted that the speedup of 2x w.r.t. a single application amounts to an overall performance gain of 50x when compared against a single threaded CPU implementation. The saturation is attributed to the fact that memory copy time is completely hidden by the kernel execution time and any further addition of kernels will not result in a performance gain.

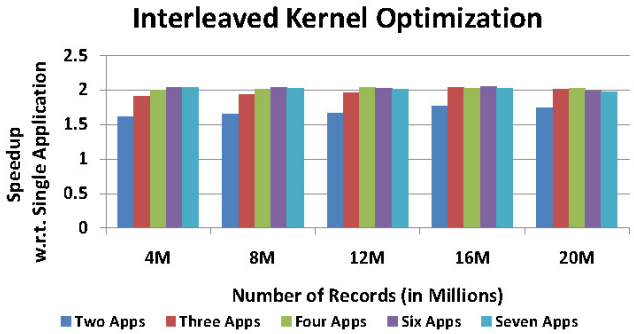

Figure 9: Performance evaluation with interleaving kernels

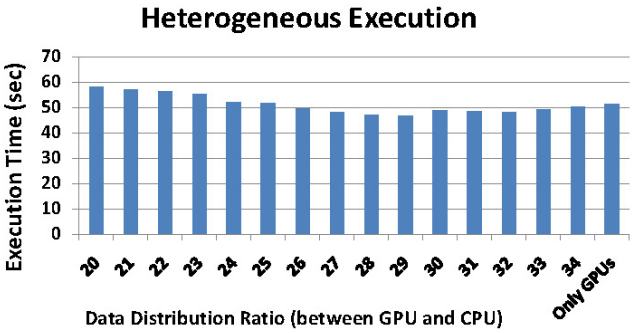

Figure 10: K-Means on heterogeneous platform (GPU+CPU)

C. Scalability

We evaluate the performance of the scalability infrastructure provided by our framework by scaling the above applications on a homogeneous and heterogeneous cluster of machines. Our heterogeneous environment consists of GPUs and CPUs. Figure 10 shows the execution times for different ratios of data distribution between the CPUs and GPUs for K-Means clustering algorithm. The hybrid middleware invokes CPU kernels and GPU kernels optimized for large datasets for distance computation and cluster update. We vary the data distribution ratio from 20 to 34 and notice that we achieve the best performance around the ratio of 29. This heterogeneous implementation results in a performance gain of around 9% compared to GPU-only implementation.

Figure 11 shows the scalability of K-Means algorithm on a homogeneous cluster of GPUs using our framework. The framework achieves 7.2x speedup when the number of GPUs increases from 1 to 8.

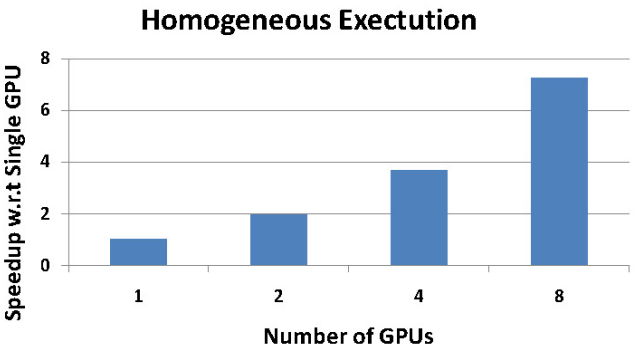

Figure 11: Scalability results for cluster of GPUs

VI. Conclusions and Future Work

In this paper we have presented a scalable framework for developing and using data mining algorithms. Our framework spans across different technologies to harness their capabilities. Data mining techniques require heavy computations and deal with huge amounts of data. To provide a scalable environment we have provided an efficient interface to handle compute-intensive tasks as well as a parallel I/O interface for optimized read/write of the data. We have further described optimizations which can be easily implemented using the framework. We introduce the concept of multiple kernel optimizations, which runs kernels from different applications for each data transfer. We also present a middleware for heterogeneous computations enabling both GPUs and CPUs work together. In future, other architectures and more data mining applications can also be easily integrated. Our framework provides the flexibility of integrating newer kernels and optimizations easily in the framework. It provides a library of (highly optimized) high performance kernels which are commonly used in data mining algorithms. The results show that we can achieve significant speedup with our optimizations. We also present, through case studies, how the framework can be used to write scalable applications.

Acknowledgment

This work is supported in part by NSF award numbers: CCF-0621443, SDCI OCI-0724599, CNS-0551639, IIS-0536994, and HECURA-0938000. This work is also partially supported by DOE grants DE-FC02-07ER25808, DE-SC0005309, DE-SC0005340, and DE-FG02-08ER25848.

References

[1] *Clementine ver. 12*, SPSS Corporation, http://www.spss.com/clementine.

[2] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.

[3] *An Introduction to R*, http://cran.r-project.org/doc/manuals/R-intro.pdf.

[4] *Rmpi: Wrapper to MPI (Message Passing Interface)*, http://cran.r-project.org/web/packages/Rmpi/index.html.

[5] *SNOW: Simple Network of Workstatsions*, http://cran.r-project.org/web/packages/snow/index.html.

[6] *Rdsm: Threads-Like Environment for R*, http://cran.r-project.org/web/packages/Rdsm/index.html.

[7] N. F. Samatova, M. Branstetter, A. R. Ganguly, R. Hettich, A. Shoshani, and S. Yoginath, "High performance statistical computing with parallel r: Applications to biology and climate modelling," *Journal of Physics*, 2006.

[8] X. Ma, J. Li, and N. F. Samatova, "Automatic parallelization of scripting languages: Toward transparent desktop parallel computing," in *IEEE International Parallel and Distributed Processing Symposium*, Long Beach, CA, March 2007, pp. 1–6.

[9] J. Buckner, J. Wilson, M. Seligman, B. Athey, S. Watson, and F. Meng, "The gputools package enables gpu computing in r," *Bioinformatics*, vol. 26, no. 1, pp. 134–135, 2010.

[10] *NVIDIA CUDA SDK*, NVIDIA Corporation, http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html.

[11] *Magma: Matrix Algebra on GPU and Multicore Architectures*, http://cran.r-project.org/web/packages/magma/index.html.

[12] A. Mooley, K. Murthy, and H. Singh, "Dismarc: A distributed map reduce framework on cuda," University of Texas, Austin, Tech. Rep.

[13] O. Lawlor, "Message passing for gpgpu clusters: cudampi," in *Proceedings of the IEEE Cluster*, 2009.

[14] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover, "Gpu cluster for high performance computing," in *Proceedings of the IEEE Supercomputing Conference*, Pittsburgh, PA, November 2004.

[15] B. G. Aaby, K. S. Perumalla, and S. K. Seal, "Efficient simulation of agent-based models on multi-gpu and multi-core clusters," in *Proceedings of the SIMUTools Conference*, Torremolinos, Malaga, Spain, March 2010.

[16] D. A. Jacobsen, J. C. Thibault, and I. Senocak, "An mpi-cuda implementation for massively parallel incompressible flow computations on multi-gpu clusters," *48th AIAA Aerospace Sciences Meeting and Exhibit*, January 2010.

[17] M. Fatica, "Accelerating linpack with cuda on heterogenous clusters," in *Proceedings of the Workshop on General-Purpose Computation on Graphics Processing Units*, Washington, D.C., March 2009.

[18] R. Wu, B. Zhang, and M. Hsu, "Clustering billions of data points using gpus," *Unconventional High Performance Computing Workshop*, pp. 1–6, 2009.

[19] R. Farivar, D. Rebolledo, E. Chan, and R. H. Campbell, "A parallel implementation of k-means clustering on gpus," *International Conference on Parallel and Distributed Processing Techniques and Applications*, 2008.

[20] B. Hong-tao, H. Li-li, O. Dan-tong, L. Zhan-shan, and L. He, "K-means on commodity gpus with cuda," in *WRI World Congress on Computer Science and Information Engineering*, vol. 3, 2009, pp. 651–655.

[21] S. A. Shalom, M. Dash, and M. Tue, "Efficient k-means clustering using accelerated graphics processors," *International Conference on Data Warehousing and Knowledge Discovery*, pp. 166–175, 2008.

[22] R. Wu, B. Zhang, and M. Hsu, "Gpu accelerated large scala analytics," HP Labs, Tech. Rep. HPL-2009-38, 2009.

[23] J. D. Hall and J. C. Hart, "Gpu acceleration of iterative clustering," *The ACM Workshop on General Purpose Computing on Graphics Processors*, August 2004.

[24] J. Li, W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, and R. Latham, "Parallel netcdf: A scientific high-performance i/o interface," in *Processings of Supercomputing Conference*, November 2003.

[25] *NVIDIA CUDA Programming Guide*, NVIDIA Corporation, http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf.

[26] R. K. Rew and G. P. Davis, "Netcdf: An interface for scientific data access," *IEEE Computer Graphics and Applications*, July 1990.

[27] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[28] J. C. Bezdek, *Pattern recognition with fuzzy objective function algorithms*. Kluwer Academic Publishers, 1981.

[29] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, pp. 32–57, January 1974.

[30] I. T. Jolliffe, *Principal Component Analysis*. Springer-Verlag, 1986.

[31] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*. London: Oxford University Press, 1965.

[32] C. Lessig, *Eigenvalue Computation with CUDA*, NVIDIA Corporation, October 2007.

[33] J. E. V. Ness, "Inverse iteration method for finding eigenvectors," *IEEE Tansactions on Automatic Control*, pp. 63–66, February 1969.