Performance Optimization of Clustering On GPU

Vijal D. Patel[1] Prof. Sumitra Menaria[2]
[1, 2] C.S.E. Dept., Parul Institute of Engineering & Technology

Abstract--In today's digital world, Data sets are increasing exponentially. Statistical analysis using clustering in various scientific and engineering applications become very challenging issue for such large data set. Clustering on huge data set and its performance are two major factors demand for optimization. Parallelization is well-known approach to optimize performance. It has been observed from recent research work that GPU based parallelization help to achieve high degree of performance. Hence, this thesis focuses on optimizing hierarchical clustering algorithms using parallelization. It covers implementation of optimized algorithm on various parallel environments using Open MP on multi-core architecture and using CUDA on may-core architecture.

I. INTRODUCTION

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of explorative data mining, and a common technique for statistical data analysis used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other, for example a hierarchy of clusters embedded in each other. Clustering can be roughly distinguished in:

1) Hard clustering: each object belongs to a cluster or not
2) Soft clustering (fuzzy clustering): each object belongs to each cluster to a certain degree (e.g. a likelihood of belonging to the cluster)

A. Classification of Clustering

Traditionally clustering techniques are broadly divided in [1]:

1) Hierarchical methods
2) Partitioning Methods
3) Density Based Algorithms
4) Grid Based Clustering

B. Hierarchical Methods

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Hierarchical clustering generally falls into two types:

1) Agglomerative Algorithm
2) Division Algorithm

In hierarchical clustering the data are not partitioned into a particular cluster in a single step. Instead, a series of partitions takes place, which may run from a single cluster containing all objects to n clusters each containing a single object. Hierarchical Clustering is subdivided into agglomerative methods, which proceed by series of fusions of the n objects into groups, and divisive methods, which separate n objects successively into finer groupings. Hierarchical clustering may be represented by a two dimensional diagram known as dendrogram which illustrates the fusions or divisions made at each successive stage of analysis.

C. Partitioning Methods

The partitioning methods generally result in a set of M clusters, each object belonging to one cluster. Each cluster may be represented by a centroid or a cluster representative; this is some sort of summary description of all the objects contained in a cluster. The precise form of this description will depend on the type of the object which is being clustered. In case where real-valued data is available, the arithmetic mean of the attribute vectors for all objects within a cluster provides an appropriate representative; alternative types of centroid may be required in other cases, e.g., a cluster of documents can be represented by a list of those keywords that occur in some minimum number of documents within a cluster. If the number of the clusters is large, the centroids can be further clustered to produces hierarchy within a dataset.

D. Density-Based Algorithms

Density-based algorithms are capable of discovering clusters of arbitrary shapes. Also this provides a natural protection against outliers. These algorithms group objects according to specific density objective functions. Density is usually defined as the number of objects in a particular neighborhood of a data objects. In these approaches a given cluster continues growing as long as the number of objects in the neighborhood exceeds some parameter.

E. Grid Based Clustering

These focus on spatial data i.e. the data that model the geometric structure of objects in the space, their relationships, properties and operations. This technique quantizes the data set into a no. of cells and then work with objects belonging to these cells. They do not relocate points but ratter builds several hierarchical levels of groups of objects. The merging of grids and consequently clusters, does not depend on a distance measure .It is determined by a predefined parameter.

1) Mode of Performance Optimization

Performance optimization is foremost objective when we are performing many computations on processor.

So following techniques are used for performance optimization:

1) Efficient Resources (Devices)
2) Optimized computing Algorithm
3) Best utilization of resources with less cost (Parallel processing)

This paper proposes a mechanism for High Performance Computing.

II. GPU ARCHITECTURE

GPU (Graphics Processing Unit) have been of large interest in last decade to achieve high performance computing through massive data parallelism model. GPU has hundreds of core processors. GPUs are designed for graphics. GPU is used to process similar operation on independent vertices & pixels in parallel. GPU is more suitable for stream computations like SIMD (single Instruction multiple data) programming model. For higher efficiency, GPU processes many data elements in parallel with same program module.

Last decade, to satisfy demand of high performance computing a tremendous rise in computing power has been observed. Although CPU has been constantly increasing their performance, the growth has been over shadowed by the improvement in performance of GPU. This trend has led to draw the interest of researchers with high computational requirements to look for GPUs as a possible solution. This has led to the emergence of a new term GPGPU (General Purpose computing using Graphical Processing Units).

GPGPU is defined as General Purpose computation using GPU and graphics API in applications other than 3D graphics. In order to achieve higher efficiency, GPGPU is designed as a combination between hardware components and software that allows the use of a traditional GPU (Graphics processor) to perform non-graphical computing tasks with higher processing power. [2]

Fig. 1 describes GPU architecture. Each GPU has number of Streaming Multiprocessors(SM) i.e.30 SMs and each streaming multiprocessor has eight streaming core processor. This architecture has characteristics to perform same operation on collection of records (Data parallelism). Hence, GPU uses streaming processor as core processor to achieve massive data parallelism. Each NVIDIA GPU has a hundreds of parallel cores and within each core has floating point unit, logic unit (add, sub, mul), move and compare unit, and a branch unit.

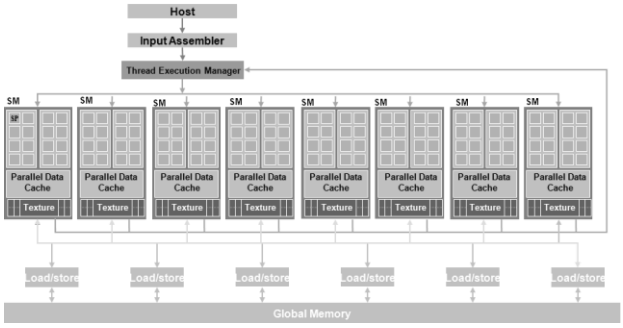

Figure (1): GPU Architecture [2]

Hence, GPU uses streaming processor as core processor to achieve massive data parallelism. Each NVIDIA GPU has a hundreds of parallel cores and within each core has floating point unit, logic unit (add, sub, mul), move and compare unit, and a branch unit.

Each core is managed by a thread manager. A thread manager can spawn and manage hundreds of threads per core. A very important feature is that it uses lightweight threads that have little creation overhead and zero-overhead scheduling. The GPU is highly parallel compute coprocessor that has its own device memory with high bandwidth.

III. CUDA

Compute Unified Device Architecture (CUDA) is a Programming model and the APIs required for performing computation tasks on NVIDIA GPUs. CUDA is a scalable highly parallel and multithreaded API to execute a program on any number of processor without recompiling. CUDA allow a program written for a platform to be executed in other platform without recompilation. CUDA provides efficient programming environment for regular applications. CUDA is designed to support many programming languages as shown in Fig.2. It provides extension to programming languages for performing computation on NVIDIA GPUs. CUDA C is extension to C language with CUDA API to operate on NVIDIA GPU. It allows a programmer to write a program for one thread, and it instantiate for multiple parallel threads.

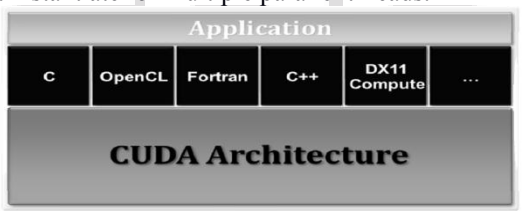

Figure (2): CUDA Interface [2]

A. *CUDA Programming Execution model*

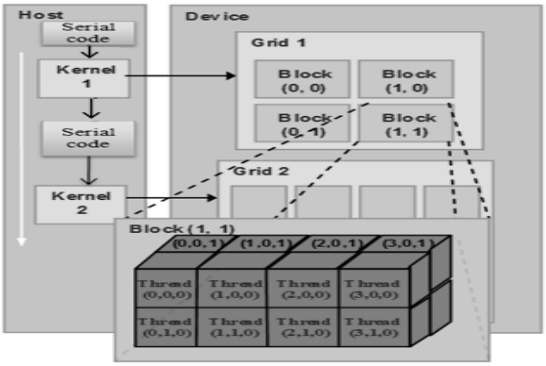

Figure (3): CUDA execution model [2]

Programming model provided for designing and executing general purpose computation kernel on NVIDIA GPU is named as Compute Unified Device Architecture (CUDA).CUDA programming modal is available for most known operating systems Windows, and Linux. CUDA programming modal allows programmer to use GPU for parallel programming without knowledge of graphics. Fig.3 describes CUDA computational model to be

executed on CPU and GPU both. A CUDA program is sequential host program and one or many kernel code as shown in Fig.3.Serial code will be executed by a CPU thread, and Kernel code will be executed by GPU threads grouped in a block.

Kernel here specifies portion of computational code to be executed on NVIDIA GPU device. CUDA kernel executes on a grid of thread blocks, where a thread block is a group of threads that work in SIMT(Single Instruction Multiple Thread) model. Threads of each block execute cooperates each other using shared memory and synchronizing its execution. Threads from different blocks operate independently. A grid can be of 1 or 2 dimensions, and a thread block can be of 1, 2, or 3 dimension as shown in Fig.3. A grid can have up to 65535 blocks of threads in each dimension and a thread block can support 768 or 1024 threads depending on GPU architecture. Each thread on a block and a block on grid has a unique identification.

IV. MCL ALGORITHM

MCL uses two simple algebraic operations, expansion and inflation, on the stochastic (Markov) matrix associated with a graph. The Markov matrix M associated with a graph G is defined by normalizing all columns of the adjacency matrix of G. The clustering process simulates random walks (or flow) within the graph using expansion operations, and then strengthens the flow where it is already strong and weakens it where it is weak using inflation operations. By continuously alternating these two processes, the underlying structure of the graph gradually becomes apparent, and there is

Converges to a result with regions with strong internal flow (clusters) separated by boundaries within which flow is absent.

1. // G is a graph, matrix M and a real number r > 1
2. // columns of M with power coefficient r is written Γr(M), and Γr is called the inflation operator with power coefficient r
3. // Γr(Mij) = Mijr / Σr,j(M)
4. add loops to G
5. set Γ to some value
6. set M_1 to be the matrix of random walks on G
7. while (change)
8. {
9. M_2 = M_1 * M_1 # expansion
10. M_1 = Γ(M_2) # inflation
11. change = difference(M_1, M_2)
12. }
13. set CLUSTERING as the components of M_1

CUDA MCL Algorithm

```
//************************************
//* chaos = max energy on each iteration.
//* energy = energy tolerant to stop MCL.
//* val,col,rL = the ELLPACK-R form of M.
//* maxL = max number of entries by rows
energy=ETOL; //default=1e-3.
chaos=1.0
SH=1; //SH=1 used shared memory
{copy sparse matrix M to device}
while (chaos>energy)
{ chaos=0.0 ;
  for (int k=0;k<N;k++)
  { CUDA_expansion_kernel(k);
    if (SH==1)
      {CUDA_inflation_SH_kernel(k);
       //* compute chaos
       CUDA_chaos_SH_kernel;}
    else
      {CUDA_inflation_GB_kernel(k);
       //* compute chaos
       CUDA_chaos_GB_kernel;}
   //** restore result into column-k M2
   CUDA_ellpackr_restore_kernel(k);
 }
 //** copy global chaos to host
 copy_chaos_to_host();
 //** copy matrix M2 into M in device
 copy_M2_to_M()
 //** synchronize threats
 cudaThreadSynchronize();
 iters++;
 if (iters>=MAX_ITERS) chaos=0.0;
}
{Interpret M as clusters}
```

Listing (1): MCL Algorithm [12]

V. CLUSTERING RESULTS AND ANALYSIS

In Fig 4 some of the results of our clusters analysis are shown, comparing number of clusters against cluster sizes, for the clusters output from data sets PPI1 (Bio GRID)

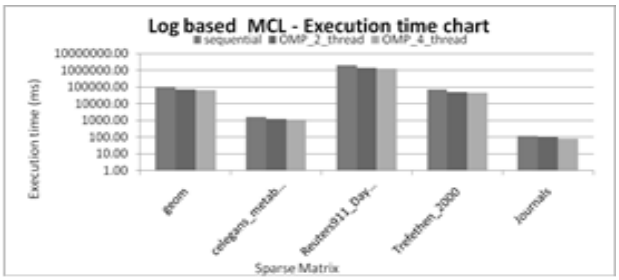

Figure (4): Comparison of MCL Execution time for SpMv using ELLAPCK-R format

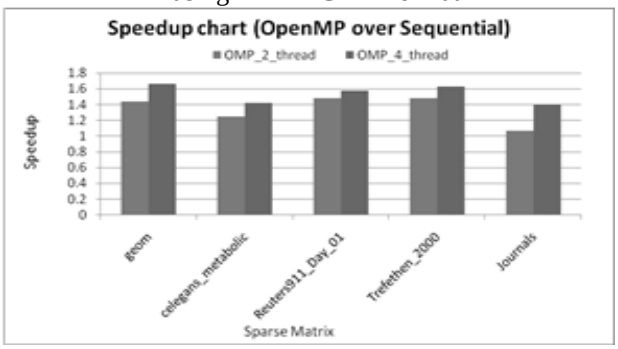

Figure (5): Speedup of MCL Execution time for Sparse-matrix using ELLAPCK-R format

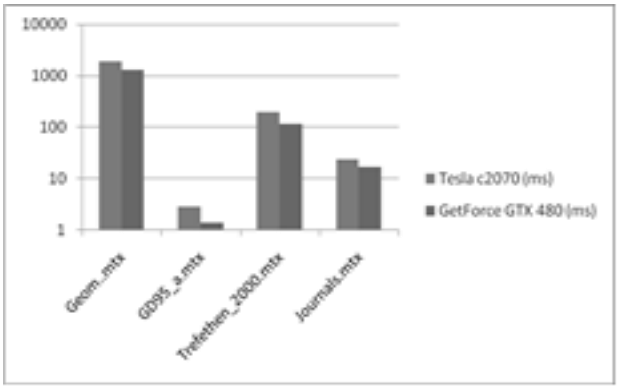

Figure (6): Comparison of MCL Execution time for SpMv matrix using ELLPACK-R format on GPU without transfer of data

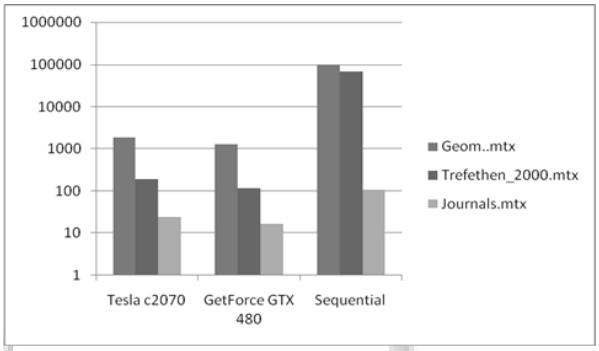

Figure (7): Comparison of MCL Execution time on CPU and GPU using ELLPACK-R Format

VI. CONCLUSION AND FUTURE WORK

1) As it has been Clustering Algorithm giving best performance on small data set so we will implement it on large data set and different GPU platform and optimize the Performance.
2) After Observation Parallel Computing is most important mode of Performance Optimization.
3) For high Performance we will implement Many Clustering Algorithm on Different
4) Platform of GPU and find out which platform is best according to Application.

REFERENCES

[1] Pradeep Rai, Shubha Singh, "A Survey of Clustering Techniques" in proceedings of the International journal of Computer Applications (0975-8887) Volume 7-No.12, October 2010.
[2] NVIDIA Corporation, NVIDIA Programming Guide, Version 3.1.1,July 2010.
[3] Osama Abu Abbas, "Comparison Between Data Clustering Algorithm" in proceedings of the International Arab Journal of Information Technology, Vol. 5, No. 3, July 2008.
[4] A.K. Jain, M.N. Murty, P.J. Flynn, "Data Clustering: A Review" ACM computing Surveys, Vol. 31, No. 3, September 1999.
[5] Wenbin Fang, Milan Lu, Xiangye Xiao, Bingsheng He, Qiong Luo, "Frequent Itemset Mining on Graphics Processors" in proceeding of the Fifth International Workshop on Data management on New Hardware, June 28, 2009.
[6] Balaji Dhanasekaran, Norman Rubin, "A new method for GPU based Irregular Reductions and Its Application to K-Means Clustering" in proceedings of the International Conference on GPGPU-4, March 2011.
[7] Feng Cao, A K.H. Tung, A Zhou, "Scalable Clustering Using Graphics Processors" School of Computing, National University of Singapore, 2004.
[8] Ren Wu, Bin Zhang, Meichun Hsu, "Clustering Billions of Data Points Using GPUs" UCHPC-MAW'09, May 2009, Ischia, Italy.
[9] Wenjing Ma, Gagan Agrawal, "A Translation System for Enabling Data Mining Applications On GPUs", ICS'09, June 2009, New York, USA, pp. 400-409.
[10] Hanaa M. Hussain, K Benkrid, A Erdogan, H Seker, "Highly Parameterized K-means Clustering on FPGAs: Comparative Results with GPPs and GPUs", in proceeding of the International Conference on Reconfigurable Computing and FPGAs, IEEE 2011.
[11] Dar-Jen Chang, M Kantardzic, M Ouyang, " Hierarchical clustering with CUDA/GPU", UCHPC-MAW, 2010.
[12] Alhadi Bustamam, Kevin Burrage, Nicholas A. Hamilton, "Fast Parallel Markov Clustering in Bioinformatics Using Massively Parallel Computing on GPU with CUDA and ELLPACK-R Sparse Format", in proceeding of the IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, VOL. 9, NO. 3, MAY/JUNE 2012.

