An Approach for Fast Hierarchical Agglomerative Clustering Using Graphics Processors with CUDA

S.A. Arul Shalom[1], Manoranjan Dash[1], and Minh Tue[2]

[1] School of Computer Engineering, Nanyang Technological University
50 Nanyang Avenue, Singapore
{sall0001,asmdash}@ntu.edu.sg
[2] NUS High School of Mathematics and Science
20 Clementi Avenue 1, Singapore
{h0630082}@nus.edu.sg

Abstract. Graphics Processing Units in today's desktops can well be thought of as a high performance parallel processor. Each single processor within the GPU is able to execute different tasks independently but concurrently. Such computational capabilities of the GPU are being exploited in the domain of Data mining. Two types of Hierarchical clustering algorithms are realized on GPU using CUDA. Speed gains from 15 times up to about 90 times have been realized. The challenges involved in invoking Graphical hardware for such Data mining algorithms and effects of CUDA blocks are discussed. It is interesting to note that block size of 8 is optimal for GPU with 128 internal processors.

Keywords: CUDA, Hierarchical clustering, High performance Computing, Computations using Graphics hardware, complete linkage.

1 Introduction

High performance computing on various multi-core processors remains as a challenge in the software industry. Amidst the presence and growth of CPU based parallel and distributed computing, the field of General Purpose Computation on Graphics Processing Unit (GPGPU) has shown tremendous achievements in increasing the computational speed by few folds. [1, 2, 9, 10, 11].

1.1 Computing Trends and Challenges Using GPU

The launch of NVIDIA's Compute Unified Device Architecture (CUDA) technology is a catalyst to the phenomenal growth of the application of Graphics Processing Units to various scientific and data mining related computations. The skills and techniques needed in invoking the internal parallel processors of a GPU should be viable to Data mining programmers who might not be expert Graphics Programmers. The intension of this work is to implement Hierarchical Agglomerative Clustering (HAC) algorithms using CUDA and demonstrate the speed gains.

M.J. Zaki et al. (Eds.): PAKDD 2010, Part II, LNAI 6119, pp. 35–42, 2010.

1.2 Graphics Processors and CUDA Technology

The GPU is designed to perform computations with extreme speed. The raw computational power of GPU can be expressed and compared with that of the CPU in terms of 'Peak floating-point operations per second' and 'Memory Bandwidth'. Fig. 1 shows the growth trend of computational power of the GPU and the CPU in terms of Peak Giga Flops (GFlops). The NVIDIA 8800 GTS GPU has a Peak performance of 624 GFlops and Memory Bandwidth of 62 Giga Bytes per second whereas a 3.0GHz Dual Core Intel CPU has a Peak Floating point rate of 12 GFlops and Memory Bandwidth of 12.8 Giga Bytes per second. Such form of raw power of the Graphics hardware is thus available to be utilized for non-image processing related computations [4, 8, 10].

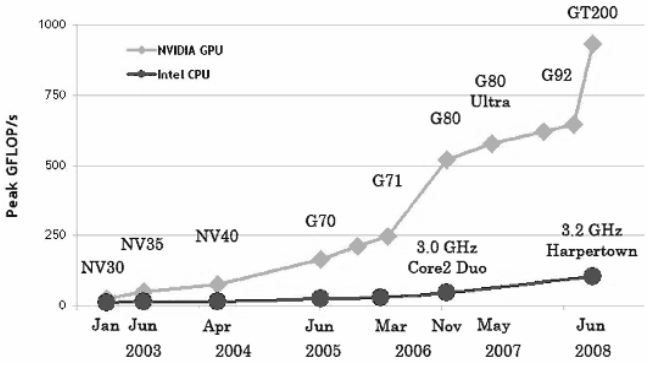

Fig. 1. Growth trend of NVIDIA GPU vs. CPU (Courtesy: NVIDIA)

2 Choice of HAC Algorithms and Implementation

2.1 HAC Algorithms

HAC is a common and important algorithm used in Data mining in the domains of micro array analysis, genome clustering, image processing and web page clustering. Hierarchical clustering seeks to build up a hierarchy of clusters. In the agglomerative approach each data vector is considered as a cluster in its own and pairs of such clusters are merged and the hierarchy moves up [3]. A measure of similarity between the clusters is required to decide which clusters should be merged. We use the Euclidean distance as the metric to determine the similarities between pair of clusters. [7, 8] where the vector norms a_i and b_i can be calculated using Equation (1), where n is the number of cluster vectors to be merged. Selection of a pair of clusters to merge depends on the linkage criteria. There are two commonly used linkage criteria and the type of HAC depends on the linkage criteria used to merge the pair of clusters.

$$|| a - b ||_2 = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \quad . \tag{1}$$

The criteria used by the complete linkage clustering and the single linkage clustering are given in Equation (2) and Equation (3) respectively.

$$Maximum\{ dist(a,b) : a \in A, b \in B \} . \quad (2)$$

$$Minimum\{ dist(a,b) : a \in A, b \in B \} . \quad (3)$$

The pair of clusters selected based on a criteria are merged and a cluster vector is updated. The centroid of the individual cluster vectors within the merged pair is computed to replace the original cluster vectors. The centroid C_j of k cluster vectors to be merged can be computed using Equation (4).

$$Cj = \frac{a_1 + a_2 + ... + a_k}{k} \quad (4)$$

The resultant hierarchy of clusters can be presented as a dendrogram. In this research paper, we intend to implement and analyze the results of HAC based on complete linkage and the single linkage methods. The HAC single linkage method has been previously implemented using CUDA [3]. We find that the merging of clusters was not done by computing the centroid of all the individual clusters within the pair of clusters selected for the merge and that short coming is rectified.

2.2 HAC Implementations Using CUDA

The computational steps that can be made parallel are implemented on the Graphics processor. Table 1 summarizes the functions used in the implementation of HAC complete linkage method using CUDA on GPU. This implementation architecture is common for both the single linkage with Centroids and the complete linkage methods. Understanding the CUDA architecture is vital in effectively implementing computational algorithms on the GPU and is well explained. [1, 3, 5]

Table 1. CUDA functions in HAC Complete Linkage

Computational Steps in HAC	GPU CUDA functions	Kernel in GPU?
Compute distances	*calculateDistance();*	Yes
Compute Centroid	*updateArray0();*	Yes
Update similarity half matrix	*updateArray1();* All distances from j^{th} cluster are set at d.	Yes
Identify maximum distance vectors	*updateArray2();*	Yes
Update similarity half matrix	*updateArray3();* i^{th} cluster distances are recalculated.	Yes
Update the i^{th} and j^{th} cluster vectors	*updateArray4();*	Yes

3 HAC Implementation Results and Discussions

The CUDA implementations of the HAC Algorithms are executed and tested on a NVIDIA GeForce 8800 GTS GPU with a memory of 512MB. The corresponding CPU implementation is run on a desktop computer with Pentium Dual Core CPU, 1.8 GHz. 1.0GB RAM on MS Windows Professional 2002. Gene expressions of Human

Mammary Epithelial Cells (HMEC) with 60000 Genes and 31 features each were used to evaluate the performance compared to the CPU implementation. This microarray dataset has been obtained from experiments conducted on HMEC. The performance of the GPU over the CPU is expressed as computational '*Speed Gain*', which is simply the ratio between the CPU computational time and the GPU computational time. The GPU computational time includes the time taken to transfer the input vectors from the CPU to the GPU and transfer cluster results back to CPU.

3.1 Determination of Optimal Block Size Based on Speed Gains

One of the parameter that affects the computational performance of GPU is the Block size. Block size in CUDA determines the number of threads to be invoked during run time. A block size of 8 invokes 8 x 8 = 64 threads during runtime which could be run in parallel. Each thread independently operates on a vector and thus exploits the parallel computing power of the GPU. Fig. 2 shows the results obtained by implementing the complete linkage HAC algorithm on the GPU with various block sizes using 5000 genes versus different dimensions. Results show that the block size of 8 is optimal for any selected number of dimensions which was used further.

3.2 Speed Gain Profile Using the Gene Expression Data Set

Fig. 3 shows the Speed Gain versus the number of Genes with 31 features for both the single linkage with Centroids and the complete linkage method. It can be noted that the single linkage method can be about 44 times faster than the CPU implementation when there are 10000 Genes to be clustered, whereas the complete linkage method reaches only about 20 times the speed of the CPU implementation above which the CPU took too long to complete, hence aborted.

3.3 Effect of Gene Size and Dimensions on Speed Gain in HAC Single Linkage

Significant resources of the GPU are utilized for the calculation of half distance matrices. For this experiment with single linkage HAC algorithm, the number of dimensions is artificially increased and the computational time taken is measured for 10000 Genes. Fig. 4 shows the effect of increase in dimensions of Genes on computational Speed Gains and the % of time taken to compute the half similarity matrices. It can be noticed that speed up to about 90 times is gained at low dimensions and it drops as the dimensions increase to about 13 to 15 times. Fig. 5 contrasts the performance with 6 and 31 dimensions.

4 Research Issues with Clustering Algorithms on CUDA

Data mining algorithms are often computationally intense and repetitive in nature which exhibit rich amounts of data parallelism. Data parallelism is a characteristic of a computational program whereby arithmetic operations can be performed on data vectors simultaneously. The inherent parallelism in the graphics hardware is invoked by CUDA features. Fig. 6 shows the CUDA architecture and the hardware resources which are used in the invocation of HAC computations on the GPU [5].

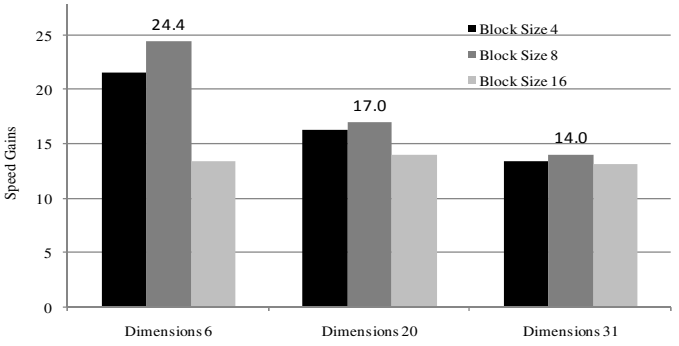

Fig. 2. HAC Complete linkage Speed Gains vs. Dimensions and CUDA Block sizes

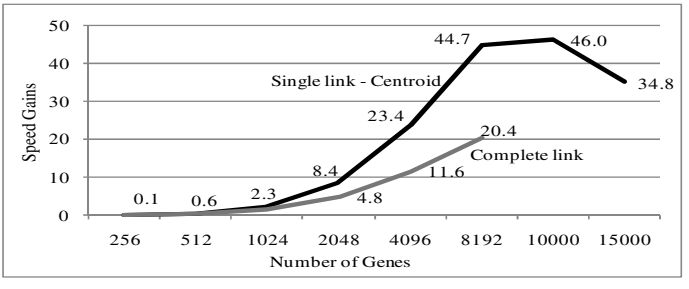

Fig. 3. GPU Performance: Single link – Centroid and Complete link vs. number of Genes

4.1 CUDA Process Block Size and Threads

Each CUDA processing block is run on one Multiprocessor (MP). In an 8800 GTS GPU there are 16 such MPs with 8 internal processors each that makes the total number of internal processors to 128. Thus while using a block size of 8, the use of 8 * 8 = 64 threads is referred to. Each thread is associated with an internal processor during runtime. Internal processors which do not belong to a block cannot be accessed by that block. So there will be a maximum of 8 execution cycles to process 64 threads while the block size is 8. If 8 such blocks can be used simultaneously, then all the 128 internal processors can be used simultaneously, thus fully harnessing the power of the GPU. This also explains why the Speed Gains with block size of 8 is high.

There is no direct control possible at block-level for the programmer and the block allocation to internal processors cannot be determined. When a grid is launched, CUDA automatically allocates blocks into the processors. There will be '$n*n/(2*k)$' threads created per block, where n is the number of observations and k is the possible number of threads per block. In Hierarchical clustering, '$n*n/2$' is the size of the half-similarity matrix, which is also the number of threads needed to simultaneously operate the entire matrix. In this HAC implementation only one block is used per grid. Hence only 1 MP is used and thus for block size 8, the number of threads invoked per block is 64. For the total number of threads generated to be invoked in the block, only

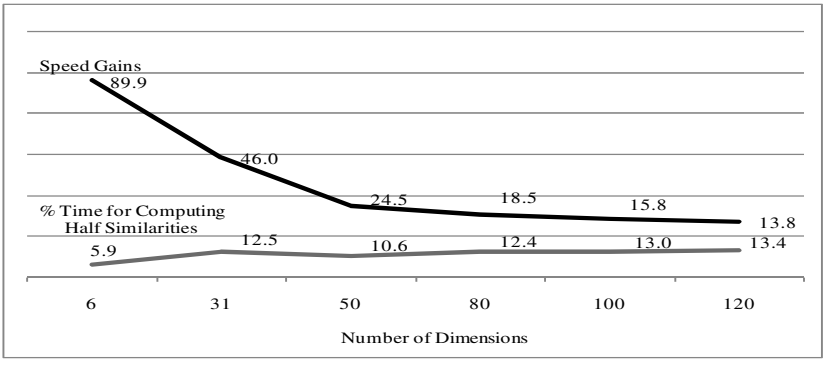

Fig. 4. HAC Single linkage method: Speed Gains with 10000 Genes vs. Dimensions and % of Time taken to compute half similarity matrices

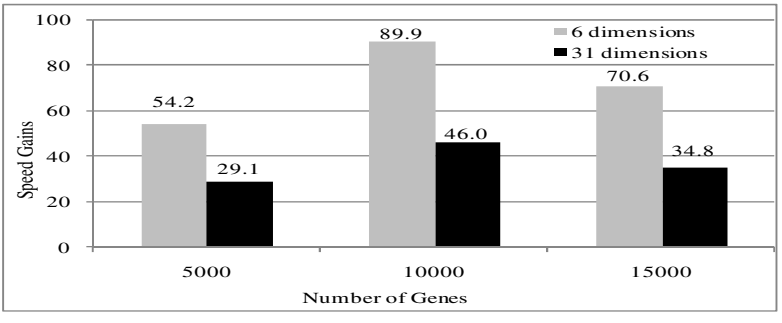

Fig. 5. HAC Single linkage Speed Gains: Gene Dimensions vs. number of Genes

'*k*number of blocks*' will be executed simultaneously. Though the total number of blocks required is '*n*n*/(2**k*)', there will be queuing while only one block is used. Within a grid, a number of blocks used will be queued up with threads and allocated to processors in a MP.

The CUDA program should use 16 or 12 or 4 blocks to fill the GPU depending on the number of internal processors in the GPU used. To be effective we need to use more blocks within the grid. When a grid is launched, the number of blocks processing is equal to '*number of MP used**8'. The number of block is designated as twice as the number of MP because computations in some of the blocks may finish earlier than the others. When a computational queue is complete, the processor will be idle and that is a waste. One way to overcome this issue is to use multiple blocks thus managing and utilizing the hardware resources of the GPU more effectively.

4.2 Analysis of Threads in CUDA for Data Parallelism

The number of threads invoked via a program is dependent on the algorithmic design. For example, for computing the vector distance of array *A* and array *B* of size *n*, there would be at least *n* threads operating on a pair of element (a_i, b_i), where $1 \leq i \leq n$. This

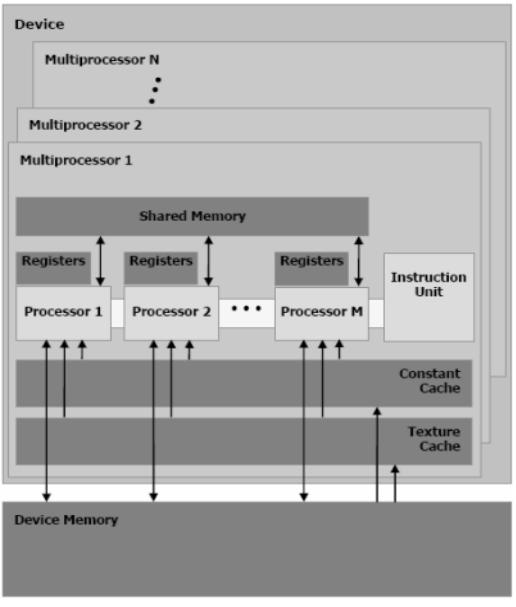

Fig. 6. GPU hardware model based on CUDA architecture (Courtesy: NVIDIA)

design theoretically would provide maximum parallelization. In the distance computations, *n* threads are arranged in a way to naturally group into blocks of similar size, satisfying the relation: '*n = number of threads* = (*number of blocks*) * (*number of threads per blocks*)'.

Each thread in a block is given a unique *thread-index*, in order to identify threads in different blocks. Therefore, to differentiate any two threads a *thread ID* can be conceived as follows: '*thread ID* = (*block-index, thread-index*)' where block-index is unique among any block. Blocks are organized into a grid. Thread-index and block-index may be formed of one, two or three dimensions. For the computations in HAC methods, it is found easier to conceive a one-dimensional grid [3].

5 Conclusion

We implemented single linkage centroid and the complete linkage HAC methods. Speed Gains about 15 to 90 times than the CPU have been achieved. The computational speed gain on HAC single linkage method is almost twice as obtained for the complete linkage method. This is due to the fact that the identifying maximum distance pair needs a custom developed function whereas the identification of minimum distance pair uses the built in CUDA library *(cublasIsamin)* function. The issues rising from the implementation of HAC methods using CUDA have been discussed and generalized. The optimal block size for CUDA processing on GPU with 128 internal processors should be 8. Maximum number of observations that can be currently clustered is limited by the size of distance matrix. Future plans include the use of 'Multiple Blocks' and implementing variants of HAC algorithm.

References

1. Halfhill, T.R.: Parallel Processing with CUDA. In: Nvidia's High-Performance Computing Platform Uses Massive Multithreading (2008)
2. Reuda, A., Ortega, L.: Geometric algorithms on CUDA. Journal of Virtual Reality and Broadcasting n(200n), Departamento de Informática Universidad de Jaén, Paraje Las Lagunillas s/n. 23071 Ja´en – Spain
3. Arul, S., Dash, M., Tue, M., Wilson, N.: Hierarchical Agglomerative Clustering Using Graphics Processor with Compute Unified Device Architecture. In: International Conference for Computer Applications, Singapore (2009)
4. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E.: A survey of general-purpose computation on graphics hardware. In: Proc. Eurographics, Eurographics Association, Eurographics'05, State of the Art Reports STAR, August, pp. 21–51 (2005)
5. NVIDIA Corportation. CUDA Programming Guide 2.0. NVIDIA CUDA developer zone (2008), `http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf` (retrived December 12, 2008)
6. Causton, H.C., Quackenbush, J.: Microarray Gene Expression Data Analysis. Blackwell Publishing, Malden (2003)
7. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons, Inc., New York (1990)
8. Chang, D., Nathaniel, A., Dazhuo, J., Ming, O.L.: Compute pairwise euclidean distances of data points with GPUs. In: Proc. IASTED International Symposium on Computational Biology and Bioinformatics (CBB), Orlando, Florida, USA (2008)
9. Wilson, J., Dai, M., Jakupovic, E., Watson, S.: Supercomputing with toys: Harnessing the power of NVDIA 8800GTX and Playstation 3 for bioinformatics problems. In: Proc. Conference Computational Systems Bioinformatics, University of California, San Diego, USA, pp. 387–390 (2007)
10. Govindaraju, N., Raghuvanshi, R., Manocha, D.: Fast approximate stream mining of quantiles and frequencies using graphics processors. In: Proc. ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, pp. 611–622 (2005)
11. Zhang, O., Zhang, Y.: Hierarchical clustering of gene expression profiles with graphics hardware acceleration. Pattern Recognition Letters 27, 676–681 (2006)