Biclustering of very large datasets with GPU tecnology using CUDA

Javier Arnedo-Fdez, Igor Zwir*, and Rocío Romero-Zaliz

Dpt. of Computer Science and Artificial Intelligence
University of Granada
Spain
{arnedo,igor,rocio}@decsai.ugr.es

Abstract. In this work we report our first research steps on using GPUs to accelerate biclustering of very large data sets, which are common in real world applications such as biomedical and biotechnological. The bicluster problem is NP-hard, thus, finding an optimal solution could be time consuming, especially when dealing with large data sets. We present a GPU-accelerated implementation of the biclustering probabilistic move-based algorithm called FLOC, which can efficiently and accurately approximate biclusters with low mean squared residues without the impact of random interference. Results show that when the size of the dataset increases, the GP-GPU version of FLOC solves the biclustering problem much faster than the CPU FLOC version running on a single CPU core.

Keywords: Data-mining, Bioinformatics, Biclustering, Parallel algorithm, GPU, CUDA, GP-GPU.

1 Introduction

There has been a substantial interest in scientific and engineering computing community to speed up the CPU-intensive tasks on graphical processing units (GPUs) with the development of General Purpose GPU (GP-GPU) systems, since GPUs have a very large memory bandwidth and computational power. Cluster analysis is a widely used technique for grouping a set of objects into classes of similar objects and commonly used in many fields such as data mining, pattern recognition and bioinformatics [1, 2] and a suitable application for the GPUs intensive power of calculation. A special case of clustering is *biclustering* where there is a simultaneously grouping of rows and columns to uncover submatrices of a given data matrix that optimize a desired objective function [3].

There are many biological applications of biclustering algorithms mainly focused on DNA microarray studies and ranges from gene sample classification, genetic pathways identifcation, gene co-regulation study, transcriptional regulatory modules identification, biomarkers discovery, drug design, single nucleotide

* This work is supported by University of Granada - GREIB.PT.2011.20 - GREIB.AL.2011.06.

polymorphism (SNP) analysis, and genetic interactions identification [4]. Eventhough microarray studies are beign replaced by newer and sophisticated methods, like next generation sequencing (NGS), biclustering techniques are still a useful tool for their analysis [5, 6].

Although there is a need to build faster biclustering algorithms that can deal with very large datasets, there is, to our best knowledge, only one biclustering GP-GPU implementation published [7].

2 Background

2.1 Parallel architectures

While conventional CPU clusters still dominate the High-Performance Computing (HPC) market, GPUs are gaining popularity as cost-effective HPC accelerators. GPUs provide a huge amount of fine-grain parallelism, since thousands of threads may be running concurrently [7].

GP-GPU systems have become increasingly popular in recent years as a means of delivering large computational power to the desktop market. Such systems consist of a host CPU with the GPU connected through a PCI-Express link. GPUs support high computational rates (in terms of floating point operations per second) and have a high bandwidth to memory on the GPU board. This makes such systems ideal for throughput oriented applications [8].

Special libraries and packages were developed for building GP-GPU system, like the API extension the C programming language called CUDA [9] (Compute Unified Device Architecture) for NVIDIA cards and OpenCL [10]. In this work we will use CUDA to code normal C functions and run them on the GPU's stream processors, thus taking advantage of a GPU's ability to operate on large matrices in parallel, while still making use of the CPU when appropriate.

2.2 Biclustering

In gene expression analysis a bicluster is defined as a submatrix spanned by a set of genes and a set of samples. Alternatively, a bicluster may be defined as the corresponding gene and sample subsets [11].

The concept of *bicluster* was introduced by Cheng and Church [12] to capture the coherence of a subset of genes and a subset of conditions. Unlike previous methods that treat similarity as a function of pairs of genes or pairs of conditions, the bicluster model measures coherence within the subset of genes and condition. The coherence score is defined as a symmetric function of genes and conditions involved and thereby the biclustering is a process of simultaneous grouping of genes and conditions. The so called mean squared residue was employed and applied to expression data transformed by a logarithm and augmented by the additive inverse. While the mean squared residue represents the variance of the selected genes and conditions with respect to the coherence, the goal of biclustering is to find biclusters with low mean squared residue [13]. It has been proven that the problem of finding biclusters satisfying these criteria is NP-hard in general. Therefore, a set of heuristic algorithms were designed by Cheng and Church [12] to either find one bicluster or a set of biclusters which consist of iterations of masking null values and discovered biclusters, coarse and fine node deletion, node addition, and the inclusion of inverted data.

2.3 FLOC

Cheng and Church original heuristics suffer from some serious drawback that produce the masking of null values and discovered biclusters with random numbers that may result in the phenomenon of *random interference* which in turn impacts the discovery of high quality biclusters. To address this issue and to further accelerate the biclustering process, a probabilistic move-based algorithm called FLOC [13] was developed for generalizing the model of bicluster to incorporate null values that can discover a set of possibly overlapping biclusters simultaneously.

The data is represented in the form of a matrix where the rows correspond to the genes and the columns correspond to the conditions. The FLOC biclustering algorithm starts from a set of seeds (initial biclusters) and carries out an iterative process to improve the overall quality of the biclustering. At each iteration, each row and column is moved among biclusters to produce a better biclustering in terms of lower mean squared residues. The best biclustering obtained during each iteration will serve as the initial biclustering for the next iteration. The algorithm terminates when the current iteration fails to improve the overall biclustering quality [13].

3 GP-GPU implementation of FLOC

The FLOC algorithm is implemented inside a Bioconductor package called BicARE [14]. The complexity of this FLOC algorithm implementation is $O((N + M)^2 \times k \times p)$, where N and M are the number of rows and columns of the original data matrix D, while k is the number of the biclusters to search for and p is the number of iterations till termination [13]. Although the FLOC biclustering method is faster than the original Cheng and Church approach, when N and M are large the FLOC algorithm performance is quite slow for real world applications where the number of genes and conditions can be very high.

To accelerate the FLOC algorithm we decided to implement a GPU version of the original algorithm based on the CUDA programming model from NVIDIA [9].

To detect which function or piece of code were the most time consuming, we perfomed a profiling of the FLOC algorithm. The funtion which calculates of the *residue* of a bicluster (see Definitions 1, 2, 3) is called several times during the execution of the algorithm, each time performing $O((M + N) \times k)$ operations.

Definition 1. *The residue of an entry d_{ij} of data matrix D in a bicluster (I, J), where $I \subseteq \{1, \ldots, M\}$ subset of genes, $J \subseteq \{1, \ldots, N\}$ subset of condition, is*

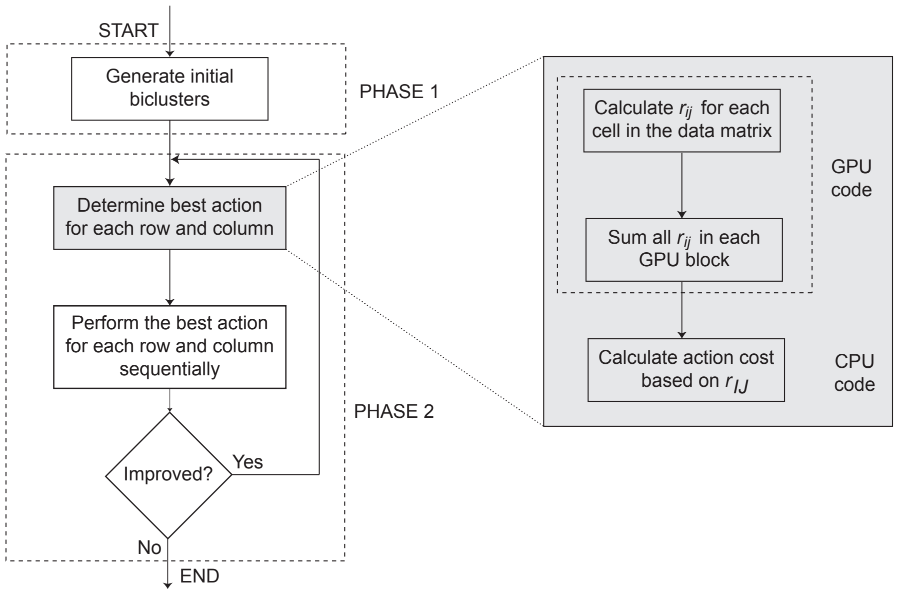

START
Generate initial biclusters
PHASE 1
Determine best action for each row and column
Perform the best action for each row and column sequentially
Improved?
Yes
No
PHASE 2
END
Calculate r_{ij} for each cell in the data matrix
Sum all r_{ij} in each GPU block
GPU code
Calculate action cost based on r_{IJ}
CPU code

Fig. 1. GP-GPU FLOC flowchart. In gray we show the section of the flowchart that is parallelized using GP-GPU.

$r_{ij} = d_{ij} - d_{iJ} + d_{IJ}$ if d_{ij} is specified in the bicluster, else $r_{ij} = 0$. d_{iJ} stands for the sum of all d_{ij} in J, while d_{IJ} stands for the sum of all d_{ij} for all I and J.

Definition 2. *The volume v_{IJ} of a bicluster (I, J), where $I \subseteq \{1, \ldots, M\}$ subset of genes, $J \subseteq \{1, \ldots, N\}$ subset of condition, is defined as the number of specified entries d_{ij} such that $i \in I$ and $j \in J$.*

Definition 3. *The residue of a bicluster (I, J) is $r_{I,J} = \frac{\sum_{i \in I, j \ in J} r_{ij}^2}{v_{IJ}}$, where $I \subseteq \{1, \ldots, M\}$ subset of genes, $J \subseteq \{1, \ldots, N\}$ subset of conditions, r_{ij} is the residue of the entry d_{ij} and v_{ij} is the volume of the bicluster.*

To accelerate the calculus of the residue fuction we created a function to be executed in each core of a GPU producing the calculation for a specific cell of the data matrix and its posterior sum. Each cell (i, j) of the data matrix calculates r_{ij}^2. Afterwards, the sum of all cell residues are performed in the same GPU device for each block independently, avoiding the overhead of transmitting all data from device to CPU. Finally, in the CPU the sum of every block is calculated and divided by the volume of the bicluster v_{IJ} (Figure 1).

4 Experiments and Results

Several experiments were performed to analyze the performance of our GPU-FLOC implementation.

First, we wanted to see if the size of the data matrices used for biclustering was actually an issue as we thought it would be. Therefore, we fixed all FLOC parameters but the size of the data matrices and run the original and improved FLOC algorithms. We created randomized matrices for sizes 10×10, 50×50, 100×100, 200×200, 300×300, 500×500, 1000×1000 and 2000×2000. For each size we created 5 different random matrices, obtaining 40 matrices in total. Figure 2 shows the results obtained using this synthetic dataset. Each point in the plot represents the mean time spent in the biclustering calculation for each matrix using different sizes. We can infer from this experiment that for small matrices (Figure 2(a)) the CPU FLOC version is faster, but when the size of the dataset is above aproximately 250×250 the GPU-FLOC version is much more efficient. All the previous experiments using the GPU-FLOC algorithm were run using 256 threads in each block of a GPU device.

(a) CPU FLOC

Matrix size	Time consumption (s)	
	Mean	*Standard Deviation*
10×10	0.01	1.79e-03
50×50	1.32	2.43e-02
100×100	10.06	4.47e-03
200×200	79.10	2.53e-01
300×300	266.34	6.79e-01
500×500	1233.60	1.41e+00
1000×1000	9859.20	8.47e+00
2000×2000	80152.90	1.56e+02

(b) GP-GPU FLOC

Matrix size	Time consumption (s)	
	Mean	*Standard Deviation*
10×10	2.40	6.98e-02
50×50	15.36	1.23e-01
100×100	29.47	2.18e-01
200×200	100.88	2.54e-01
300×300	170.40	4.34e-01
500×500	483.94	8.48e-01
1000×1000	2050.00	3.15e+00
2000×2000	10449.70	1.47e+01

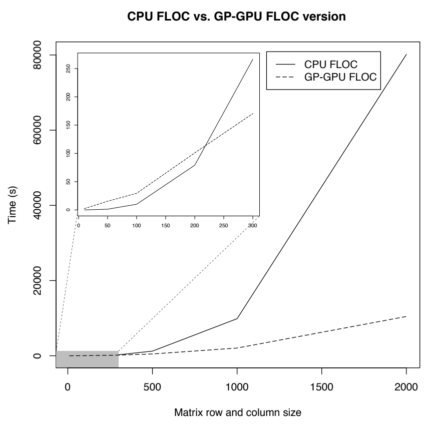
Table 1. Statistics for the performed experiments.

Second, we wanted to test which thread and block configuration was the best option and whether changing these parameters made a substantial difference in performance. The experimental framework used fixed all FLOC parameters but the number of threads per block. Figure 2(b) shows the results. As we expected, when the number of threads per block increases the performace of the GPU implementation is faster, but when the number of threads is quite high the gain is negligible. Nevertheless, using the slowest configuration of the GP-GPU FLOC (i.e., 16 threads per block), its performance is much better than the CPU FLOC algorithm.

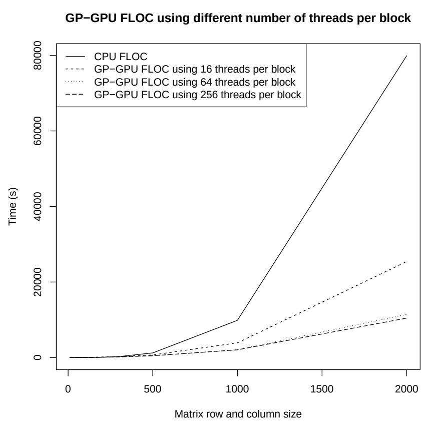
All experiments were run in an Intel i7 980 machine with 16 GB of RAM and Gainward GeForce GTX 480 video cards with 1.5 GB of RAM each.

5 Discussion

Preliminary results show that the use of GPU acceleration can substantially improve the performance of biclustering methods. This improvement will help



(a) CPU FLOC vs. GP-GPU FLOC version.



(b) GP-GPU FLOC using different number of threads per block.

Fig. 2. CPU FLOC vs. GP-GPU FLOC version. Matrix size ranges from 10 × 10 to 2000 × 2000. The number of biclusters searched were 10 and 50 iterations were performed, for both versions.

bioinformatic software to cope with the large amount of data that NGS technology is providing.

Memory transfers from the host CPU to the GPU devices over the PCI-Express bus is the main issue when programming GPU applications. The bandwidth of PCI-Express is much lower compared to the on-board memory bandwidth. This can then become the bottleneck of the system, especially if large amounts of data need to be transferred over the bus [9]. It is therefore critical to minimize the total data that is sent back and forth from CPU to GPU memory. Also, there is a limit in the number of threads per block and blocks per grid that has to be considered. Not all algorithms are suitable for GPU acceleration, whatsmore a wrong implementation can cause the GP-GPU algorithm to be even slower than the CPU version.

Future work will be devoted to test all possible GPU parameter's configuration including the use of more than one GPU, and to compare them with other parallel architectures like MPI or PVM [15, 16].

References

1. Yildirim, A.A., Ozdoğan, C.: Parallel wavelet-based clustering algorithm on gpus using cuda. Procedia Computer Science **3**(0) (2011) 396 – 400
2. Petros, X., Nikita, B., Neng, F., Panos M, P. In: Biclustering: Algorithms and Application in Data Mining. John Wiley & Sons, Inc. (2010)
3. Busygin, S.: Biclustering in data mining. Computers & Operations Research **35**(9) (2008) 2964–2987

4. Liu, L., Wei, D., Li, Y.: Handbook of Research on Computational and Systems Biology: Interdisciplinary Applications. Igi Global (2011)
5. Wang, J., Tan, A., Tian, T.: Next Generation Microarray Bioinformatics: Methods and Protocols. Methods in Molecular Biology. Springer Verlag (2011)
6. Huang, Q., Wu, L.Y., Qu, J.B., Zhang, X.S.: Analyzing time-course gene expression data using profile-state hidden Markov model. In: IEEE International Conference on Systems Biology. (2011)
7. Mejía-Roa, E., García, C., Gómez, J., Prieto-Matías, M., Nogales-Cadenas, R., Tirado, F., Pascual-Montano, A.D.: Biclustering and classification analysis in gene expression using nonnegative matrix factorization on multi-gpu systems. In: 11th International Conference on Intelligent Systems Design and Applications. (2011)
8. Satish, N., Sundaram, N., Keutzer, K.: Optimizing the use of gpu memory in applications with large data sets. In: HiPC. (2009) 408–418
9. Kirk, D.B., Hwu, W.m.W.: Programming Massively Parallel Processors: A Hands-on Approach. 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2010)
10. Khronos OpenCL Working Group: The OpenCL Specification, version 1.0.29. (8 December 2008)
11. Tanay, A., Sharan, R., Shamir, R.: Biclustering Algorithms: A Survey. Handbook of Computational Molecular Biology (2004)
12. Cheng, Y., Church, G.M.: Biclustering of expression data. Proceedings / ... International Conference on Intelligent Systems for Molecular Biology ; ISMB. International Conference on Intelligent Systems for Molecular Biology **8** (2000) 93–103
13. Yang, J., Wang, H., Wang, W., Yu, P., Ibm, U., Chapel, U., Ibm, H., Watson, T.J., Watson, T.J.: Enhanced biclustering on expression data. In: Proc. of 3rd IEEE Symposium on BioInformatics and BioEngineering (BIBE03. (2003) 321–327
14. Gestraud, P., Brito, I., Barillot, E.: Bicare: Biclustering analysis and results exploration (2010)
15. Pacheco, P.S.: Parallel programming with MPI. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
16. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V.S.: PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing. Scientific and engineering computation. (1994)