Neurocomputing 134 (2014) 239–246

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

GPU-based biclustering for microarray data analysis in neurocomputing

Benben Liu, Yao Xin, Ray C.C. Cheung *, Hong Yan

Department of Electronic Engineering, City University of Hong Kong, Hong Kong SAR, China

ARTICLE INFO

Article history:
Received 11 March 2013
Received in revised form
6 June 2013
Accepted 25 June 2013
Available online 14 January 2014

Keywords:
Biclustering
Microarray
High Performance Computing (HPC)
Graphics Processing Unit (GPU)

ABSTRACT

Biclustering is one of the important techniques in neurocomputing and bioinformatics. Geometric Biclustering (GBC) algorithm is used to find the common patterns in given microarray data for neural processing. A microarray can produce a massive amount of data and require high computational power for data analysis. With intrinsic parallel architecture and appropriate mapping technique Graphical Processing Unit (GPU) has the advantage of processing large number of threads and data compared to CPU. This paper analyzes the parallelism and data reuse of the GBC algorithm, and presents three different efficient implementations using five benchmarks from real world. The proposed GPU-based GBC program achieves significant speedup over highly optimized CPU program. By comparing implementation results, the paper studies how to design a scalable architecture for mapping the GBC and other similar algorithms that deal with microarray data analysis. The paper also explores how GPU-based GBC is affected by the input data size.

1. Introduction

Neurocomputing is a kind of computing using the models that simulate nerve system, such as artificial neural networks and machine learning systems. The basic component of a human nerves system is a neuron. Typically humans have billions of neurons working in parallel, asynchronous and distributed fashion. By using these models, brain-like computing can be achieved to fulfill aim of embedding intelligence into machines such as computers and microprocessor chips, so it is becoming increasingly important in various areas.

Clustering is a data classification process that partitions the data according to a number of criteria in order to discover the patterns that exist in the dataset. Biclustering is a simultaneous clustering technique to identify patterns on both row and column dimensions of the data matrix based on both properties and features [1]. It has been widely used in many different application domains such as biomedical research [2], finance [3] and image processing [4].

Biclustering is an essential area in neurocomputing applications. For example, some applications of genetic algorithms (GAs) in microarray deal with biclustering. In [5] GAs are employed by integrating a greedy algorithm as a local search in order to improve the quality of biclustering. In [6] Mitra and Banka analyze the relationship between gene expression level variation over time of a transcription factor and that of its target in the framework of evolutionary biclusters. A simple and novel correlation-based approach is employed to automatically extract gene interaction networks from biclusters in microarray data [7].

In addition, biclustering is an important technique for neural information processing [8] and biological dataset analysis [2]. It has been widely studied in research [9–12]. Besides, biclustering is employed to investigate gene data related to many diseases. For example, cancer can be identified based on tissue classification [13]. Genes with similar expression patterns may result in close biological behaviors, so a lot of research has been devoted into this interesting area. Therefore, it is of great importance to develop efficient algorithms for solving biclustering problems. However, there are usually a huge number of data involved, and biclustering is an NP-complete problem which requires either large computation power or heuristic method to reduce computational complexity [14].

Geometric biclustering (GBC) algorithm [15] is developed to reduce the computational complexity which identifies the linearities of microarray in a high dimensional space of biclustering algorithm. It provides results with higher accuracy than other methods. In order to further speedup the GBC algorithm, a hypergraph partitioning method [16] uses a software partition tool called hMetis [17] to reduce the size of matrix in each operation. It is still quite time consuming to identify patterns in a large microarray. Therefore, High Performance Computing (HPC) system is essential to accelerate this process.

Nowadays, Graphic Processing Unit (GPU) is commonly used in HPC systems. It is able to process a large amount of data in parallel to reduce the total running time. In this system General-Purpose GPU (GPGPU) works as a coprocessor to accelerate algorithms

* Corresponding author.
E-mail addresses: benben.liu@my.cityu.edu.hk (B. Liu), yaoxin2-c@my.cityu.edu.hk (Y. Xin), r.cheung@cityu.edu.hk (R.C.C. Cheung), h.yan@cityu.edu.hk (H. Yan).

0925-2312/$ - see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2013.06.049

which are computationally intensive. By using GPU we can achieve higher performance per unit die size. At the same time, compared with traditional programming mapping algorithms to GPU requires sophisticated technique in order to obtain high speed. In this paper, we propose a GPU-based GBC algorithm that can be also applied to other algorithms for microarray data analysis. The proposed algorithm is scalable with the size of benchmarks and GPU resources, and can be efficiently implemented in parallel. The algorithm is implemented on a CUDA-enabled GPU and it achieves significant speedup as well as high memory bandwidth. Specifically, the key contributions of this paper are

1. analyzing the parallelism and data reuse feature of the geometric biclustering algorithm;
2. proposing three efficient implementations of the geometric biclustering (GBC) algorithm on GPU platform using five benchmarks from real world; and
3. exploring the relationship between memory access, control flow, input data size, CPU-GPU communication, and acceleration over CPU.

This paper is organized as follows. Section 2 describes the background and related work. Section 3 introduces the geometric biclustering algorithm and analyzes its parallelism and data reuse possibility. Section 4 presents the three efficient implementations on GPU. Section 5 shows the results and comparison of the implementations. Section 6 discusses the important consideration that limits the speedup. Section 7 concludes the paper and proposes the future work.

2. Background

In data analysis, response patterns are often identified from microarray data matrix [18] in order to detect disease subphenotypes, predict disease progression and activities of new compounds [13]. Biclustering is a simultaneous clustering technique on both row and column dimensions of the data matrix [1]. Biclustering is a data mining technique that can extract patterns from microarray data matrix according to certain criteria. Besides, it can be also formulated in multidimensional data space [2]. Biclustering can detect five coherent patterns [14] including (a) constant value in the entire pattern, (b) constant values in columns, (c) constant values in rows, (d) additive coherent values, and (e) multiplicative coherent values. These five patterns are shown in Fig. 1.

Since biclustering is an NP-complete problem, it is very slow to compare over tens of thousands of data patterns. For small simulated benchmark such as a 100 by 100 matrix, it will take more than 2 min to identify the biclusters using the hypergraph partitioning method processed with a Matlab program which runs on a PC with an Intel Core i7-920 2.66 GHz processor and DDR3 3GB memory according to our experimental test. A normal gene matrix, however, typically has more than 10,000 rows or columns, so it is necessary to explore the use of high performance computing (HPC) platforms to accelerate the GBC algorithms.

GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads simultaneously, rather than executing a single thread very quickly. Compute Unified Device Architecture (CUDA) is a parallel computing architecture developed by NVIDIA, which is accessible to software developers through variants of industry standard languages like C/C++. It has a great advantage over shading language when performing general-purpose computation. The architecture of CUDA is shown in Fig. 2. We can find from the figure that CUDA consists of a number of streaming multiprocessors (SM) and each SM is composed of many streaming processors (SP) which is also known as CUDA cores. The device C/C++ code is first compiled into PTX code, which is then allocated by thread execution manager to SMs, further to SPs. Each SM owns fast on-chip shared memory for all the SPs inside and all the SMs share data through huge global memory. Given one or more thread blocks to execute, an SM partitions them into groups of 32 parallel threads called warps which get scheduled by a warp scheduler for execution. Besides,

a

C1	C2	C3	C4
6	6	6	6
6	6	6	6
6	6	6	6
6	6	6	6
6	6	6	6

Constant pattern
(Ci = Cj = constant)

b

C1	C2	C3	C4
4	5	6	7
4	5	6	7
4	5	6	7
4	5	6	7
4	5	6	7

Constants columns
(Ci = constant_1,
Cj = constant_2)

c

C1	C2	C3	C4
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7

Constant rows
(Ci = Cj)

d

C1	C2	C3	C4
4	5	1	2
5	6	2	3
6	7	3	4
7	8	4	5
8	9	5	6

Additive coherent values
(Ci = Cj + constant)

e

C1	C2	C3	C4
1	2	4	8
2	4	8	16
3	6	12	24
4	8	16	32
5	10	20	40

Multiplicative coerent
(Ci = Cj * constant)

Fig. 1. The five types of patterns in biclustering.