PLOS ONE

RESEARCH ARTICLE

FastGCN: A GPU Accelerated Tool for Fast Gene Co-Expression Networks

Meimei Liang, Futao Zhang, Gulei Jin, Jun Zhu*

Institute of Bioinformatics, Zhejiang University, Hangzhou, Zhejiang, RP China, 310058

* jzhu@zju.edu.cn

Abstract

Gene co-expression networks comprise one type of valuable biological networks. Many methods and tools have been published to construct gene co-expression networks; however, most of these tools and methods are inconvenient and time consuming for large datasets. We have developed a user-friendly, accelerated and optimized tool for constructing gene co-expression networks that can fully harness the parallel nature of GPU (Graphic Processing Unit) architectures. Genetic entropies were exploited to filter out genes with no or small expression changes in the raw data preprocessing step. Pearson correlation coefficients were then calculated. After that, we normalized these coefficients and employed the False Discovery Rate to control the multiple tests. At last, modules identification was conducted to construct the co-expression networks. All of these calculations were implemented on a GPU. We also compressed the coefficient matrix to save space. We compared the performance of the GPU implementation with those of multi-core CPU implementations with 16 CPU threads, single-thread C/C++ implementation and single-thread R implementation. Our results show that GPU implementation largely outperforms single-thread C/C++ implementation and single-thread R implementation, and GPU implementation outperforms multi-core CPU implementation when the number of genes increases. With the test dataset containing 16,000 genes and 590 individuals, we can achieve greater than 63 times the speed using a GPU implementation compared with a single-thread R implementation when 50 percent of genes were filtered out and about 80 times the speed when no genes were filtered out.

Citation: Liang M, Zhang F, Jin G, Zhu J (2015) FastGCN: A GPU Accelerated Tool for Fast Gene Co-Expression Networks. PLoS ONE 10(1): e0116776. doi:10.1371/journal.pone.0116776

Academic Editor: Junwen Wang, The University of Hong Kong, HONG KONG

Received: August 26, 2014

Accepted: December 8, 2014

Published: January 20, 2015

Data Availability Statement: Documentation and source code for FastGCN are available at https://github.com/DrLiang/FastGCN.

Funding: This research is supported by grant from the National Natural Science Foundation of China (30470916). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

Introduction

Gene co-expression analysis provides a comprehensive way to identify the interactions between and the functions of certain genes. Additionally, co-expression analysis is an important step in bioinformatics analyses of gene expression data [1]. Currently, the majority of gene expression data are generated from microarray and RNA-seq methods. Microarray technologies have been widely used over the last few decades. With the emergence of high-throughput next-generation sequencing technologies, RNA-seq technology has been introduced. RNA-seq has several advantages over microarray technologies and may fully replace microarrays when several issues are overcome [2–4].

Gene co-expression networks comprise one type of valuable biological networks. One network can cover all human genes [5]. In gene co-expression networks, a gene is represented by vertexes and the edges that correspond to the pairwise correlations between expressions. Many methods have been published to construct gene co-expression networks [6–9]. The Pearson correlation coefficient is widely used. However, this method requires a substantial amount of time and space for large datasets [10–12]. Recently, Graphics Processing Units (GPUs) provide a promising tool to process tasks in parallel. GPUs are designed to have hundreds or even thousands of cores with small caches and logic control units. Therefore, the nature of GPU architecture makes it a powerful device for parallel computing rather than for data caching or branch control. The gene co-expression analysis for Pearson correlation coefficient computing is pairwise and easy to divide into small tasks. These tasks are independent and can be assigned to many-core GPUs for parallel processing. In the future, GPU computing should be used to construct the co-expression networks for large datasets [7].

CUDA (Compute United Device Architecture) C is a C-extended language developed by NVIDIA. This language is used to develop programs that can run on CUDA GPUs. The work pattern of GPU computing is SIMT (Single Instruction Multiple Thread). The code segment that starts a GPU is called Kernel. When Kernel runs, the thread space should be defined. The GPU threads are organized in Blocks. The thread number in Block is limited. No more than three dimensions are available for address space in Block. In an identical manner, Blocks are organized into a Grid, for which the address space should have no more than three dimensions. When the GPU starts computing, Blocks are assigned to SMX (prior to Kepler architecture, SMX was called SM) and running in a SIMT pattern. The number of running Blocks in SM depends on the Register number that one thread needs and the size of SMEM (Shared Memory). Threads should be busy for as much time as possible. The Block size and the resources that each thread claims leverage the entire resource.

The variables defined in Kernel are allocated in Registers automatically. Several shared variables or data structures are allocated in SMEM. SMEM and Registers are in the GPU chip; therefore, these two components have high access bandwidth. Other data are allocated in GMEM (Global Memory). To hide the GPU latency, the number of running warps (32 threads a warp) on the SM (Stream Multiprocessor) should be as many as possible. The GPU thread space is created by the syntax ⋘ grid, block⋙. In general, the size of the grid should be at least three times the number of SM. Moreover, there should be more than four warps in a Block.

In recent years, several tools and methods were introduced to harness the GPU power for increasing processing speeds in gene expression network analyses [13–17]. Here we propose a computationally efficient tool, FastGCN, to construct gene co-expression networks. The analysis procedure mainly contains four steps: preprocessing the input data with genetic information entropy, computing the Pearson correlation coefficient for the gene pair, transforming the coefficients to a normal distribution and identifying modules. GPU computing technology was exploited in all four steps. A sparse matrix compression was used in the correlation coefficient matrix. Our method outperformed the traditional gene co-expression analysis methods. This method is a powerful tool to assist the construction of gene co-expression networks. The source code and the executable binaries for this software are available at https://github.com/DrLiang/FastGCN.

Methods

Data Preprocessing with Genetic Information Entropy

Information entropy was proposed by Shannon [18] and has been widely used in industry to measure the uncertainty of a random variable. After the initial description, information

entropy was heavily used in the life sciences [19–26]. In the present study, we used genetic information entropy to reduce the input data. For one dataset with n individuals and m genes, the genetic information entropy of the i-th gene can be defined as the following:

$$H_i = -\sum_{j=1}^{n} p_{ij} \log p_{ij} \tag{1}$$

where $p_{ij} = \frac{|x_{ij}|}{\sum_{k=1}^{n} |x_{ik}|}$, x_{ij} is the expression abundance of the i-th gene of the j-th individual. When p_{ij} reaches an equilibrium distribution (the i-th gene has no differential expression among the individuals), the genetic information entropy H_i attains its maximum value. Therefore, we can filter a portion of the non-differentially expressed genes using the genetic information entropy.

Pearson Correlation Analysis

For two genes x and y, the Pearson correlation coefficient is the following:

$$\rho_{xy} = \left(\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i \right) / n \right) / \sqrt{\left(\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2 / n \right) \left(\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i \right)^2 / n \right)} \tag{2}$$

where x_i and y_i are the expression abundances of genes x and y of the i-th individual, respectively.

Normalization of Pearson Correlation Coefficients

For k genes, we are able to write $t = k(k-1)/2$ for the Pearson correlation coefficients. We normalized these coefficients to a test statistic, z, to test for the co-expression between two genes,

$$z_i = (\rho_i - \bar{\rho}) / S \tag{3}$$

Where $S = \sqrt{\sum_{i=1}^{k} (\rho_i - \bar{\rho})^2 / (k-1)}$ is the standard deviation and $\bar{\rho}$ is an overall mean. Therefore, z is asymptotically distributed in a standard normal distribution. We can then transform the z-value to a P-value. Finally, we used the False Discovery Rate to control the multiple tests.

Module Identification

With the normalized z as the evaluation criteria, we can easily transform coefficient matrix to adjacency matrix. The adjacency function is defined as:

$$a_{ij} = signum(q_{ij}, \tau) = \begin{cases} 1 & q_{ij} \le \tau \\ 0 & q_{ij} > \tau \end{cases} \tag{4}$$

where q_{ij} is the q-value for False Discovery Rate control. τ is threshold parameter. When the adjacency matrix is defined, the node connectivity can be got by:

$$c_i = \sum_{j} a_{ij} \tag{5}$$

To select the relevant genes and reduce the computational burden, the genes with $c_i = 0$ are removed.

In gene co-expression network, modules are always defined as clusters of highly correlated genes [8,27–29]. We implemented module identification analysis based on topological overlap measure with a node similarity clustering method. The similarity matrix was defined as:

$$s_{ij} = \frac{h_{ij} + a_{ij}}{c_i + c_j - h_{ij} - a_{ij}} \quad (6)$$

where $h_{ij} = \sum_u a_{iu} a_{uj}$ is the number of shared nodes that connect to both node i and node j. c_i and c_j are the connectivity of node i and node j, respectively. The similarity matrix is also symmetric. These similar genes can form hot blocks which can be identified as modules along the diagonal.

Implementation

We implemented the algorithms of these four steps using GPU technology with CUDA. The GPU works in a SIMT manner. When the GPU issues one instruction, many threads initiate on different data. These algorithms are able to exploit the parallel GPU computing power because each task was independent and easily assigned to one GPU thread.

Data preprocessing on the GPU

For m genes (or exons) and n individuals in the raw input data, we created m GPU threads to compute the genetic information entropy. One entropy task was computed by one GPU thread. Using the traditional method, we should copy the data from the host memory to the GPU device memory to perform GPU computing. However, this method has two shortcomings in gene co-expression analysis. First, the raw data can be large but the GPU device memory is relatively small. Occasionally, the GPU device memory cannot accommodate all the input data, intermediate data and the result data. Second, transporting data from host memory to the GPU device memory belongs to I/O operations. I/O operations are inherently slow. To improve the performance, we should reduce the I/O operations as much as possible. To meet these challenges, Zero-copy technology was exploited. Zero-copy technology allows the GPU to use page-locked host memory. The GPU can directly access this part of the host memory. However, page-locked host memory is not page-able, and with overuse over a long time, this process can decrease the performance of the entire system. The asymptotic time complexity of entropy computation is $O(mn)$. With GPU computing, this procedure can finish in a short time. Then, the occupied page-locked memory can be released. S1a Fig. shows the GPU architecture for the genetic information entropy computation. After determining the entropies from the GPU, the CPU took the responsibility of filtering the genes accordingly.

Pearson Correlation Analysis on the GPU

The Pearson correlation analysis is a pairwise analysis. For k genes remaining after data preprocessing, $j = k(k - 1) / 2$ coefficients should be computed. The asymptotic time complexity of Pearson correlation analysis is $O(nk^2)$. This analysis is the most time consuming step in the entire analysis. Generally, the coefficients are stored in a $k \times k$ strictly lower triangular matrix or upper triangular matrix. To save space, we used a packed matrix storage model. The packed matrix stores only the coefficients in a vector. To minimize the latency and improve the performance, as many GPU threads should be created as possible. We created j GPU threads. Each thread computed the coefficient of one gene pair. To implement this step, the gene index must be calculated by the GPU thread identification; therefore, we mapped the one-dimensional GPU thread space onto a two-dimensional input matrix space and one-dimensional result

vector space. The mapping relationship is shown in S2a Fig. The GPU threads read the data from page-locked memory and stored the results in the GPU device memory. Then, the page-locked memory can be freed. S1b Fig. shows this pattern.

Normalization of the GPU

We created j GPU threads in this normalization. One thread normalized one coefficient. The standard deviation in Equation 3 was shared between all GPU threads. Having each thread calculate the standard deviation separately is not an efficient method. Therefore, we used the summation reduction to determine the standard deviation of the threads cooperatively (S2b Fig.). The first half of the threads was used to sum the two coefficients. Then, the active threads were used at a half of the last step, and the summation was repeated. When the active threads were reduced to one thread, we achieved the final result. In the GPU memory hierarchy, shared memory is on-chip memory. On-chip memory is much faster than GPU device memory. Therefore, we used shared memory to achieve a high performance. However, if two or more threads request access to the identical memory bank when using shared memory, a bank conflict occurs. The access would then be serialized. To avoid bank conflict, we used sequential addressing. In this step, GPU threads are not independent from one another. The threads should be synchronized. To ensure the correct result, several barriers should be added to control the synchronization. This architecture is shown in S1c Fig.

Module Identification on GPU

We implemented module identification with three GPU kernels. We also assume there are k genes. First, k GPU threads were created to get the adjacency matrix and the node connectivity vector. Then, we used CUBLAS, which is a CUDA library of basic linear algebra subroutines to calculate the matrix h defined in Equation 6. Finally, we used $k \times k$ GPU threads in two-dimensional thread space to compute the similarity matrix. We plotted S1d Fig. showing this architecture.

Results

We have developed four versions of our tool FastGCN: GPU version, Multi-core CPU version, Single-thread CPU version with C/C++ and Single-thread CPU version in R. The version using the R language was highly optimized; we eliminated the explicit loops in all the computing procedures. To evaluate the performance of FastGCN for co-expression network inference, we used the GPU workstation consisting of a NVIDIA Tesla K20c card running on an Intel Xeon E5-2690 (16 CPU cores) with a clock rate of 2.90GHz using 256GB DDR3 host memory. We conducted several experiments using different datasets to compare the performance of these four versions. These datasets are extracted from the expression data covering Breast Invasive Cancer (BRCA) from TCGA [30] (https://tcga-data.nci.nih.gov/tcga/) which contains 17,814 genes and 590 individuals. Each dataset contains 590 individuals but has a different number of genes. We implemented two test groups: one with fifty percent of gene cutoff at the data preprocessing stage and the other with no gene cutoff. We tested the multi-core CPU version on 16 CPU threads running on 16 CPU cores. The results of this comparison are summarized in Table 1 and Table 2, respectively. This comparison showed the obtained running obtained as the mean running time of 1000 simulations. We calculated the running time from entropy preprocessing, the Pearson correlation coefficients calculated using the z-score normalization and module identification whereas excluding the input procedure and the output procedure because we only compared the computational performance of this tool. From Table 1, we can see that by using the GPU, the entire computational procedure of the dataset with 16,000 genes

Table 1. Running time (in seconds) for the GPU implementation, Multi-core CPU implementation, Single-thread C/C++ implementation and Single-thread R implementation when 50% of the genes were filtered out during the data preprocessing stage (individual number = 590).

Gene Number	GPU	Multi-core CPU	Single-thread C/C++	Single-thread WGCNA	Single-thread R
2k	0.624	0.094	0.375	1.622	1.295
4k	0.671	0.218	1.138	5.020	5.101
6k	0.811	0.39	2.403	14.102	15.163
8k	0.967	0.655	4.447	25.032	25.740
10k	1.202	0.889	7.301	41.769	42.260
12k	1.388	1.419	10.811	69.685	69.420
14k	1.747	1.965	14.633	94.731	95.220
16k	2.122	3.010	20.369	130.627	134.100

Multi-core CPU version ran on 16 CPU threads running on 16 CPU cores.

doi:10.1371/journal.pone.0116776.t001

finished in about 2 seconds. However, 134.1 seconds were required for the single-thread R version, and 20.369 seconds were required for the single-thread C/C++ version. Therefore, the GPU implementation was approximately 63 times faster than the traditional serial CPU implementation in the R language and approximately 10 times faster than the CPU implementation with the C/C++ language. From Table 2, the GPU version can achieve speed increases of almost 80 times that of the single-thread R version. WGCNA[8] is a promising R package for correlation network analysis. The performance of WGCNA was better than the single-thread R implementation but worse than others. For the sake of immediacy, we also plotted Fig. 1A and Fig. 1B to show the speedup curves. These curves show that when the dataset is small, the multi-core version running on 16 CPU threads outperforms the other processes. However, as the size of dataset increases, the GPU version can surpass the multi-core version in terms of performance.

To illustrate the application of FastGCN in the construction of a co-expression network of gene expression data, FastGCN was applied to the Breast Invasive Cancer (BRCA) data from TCGA. The genes in the HTLV-I infection pathway from the KEGG database (http://www.genome.jp/kegg/pathway.html) were included in the analysis.

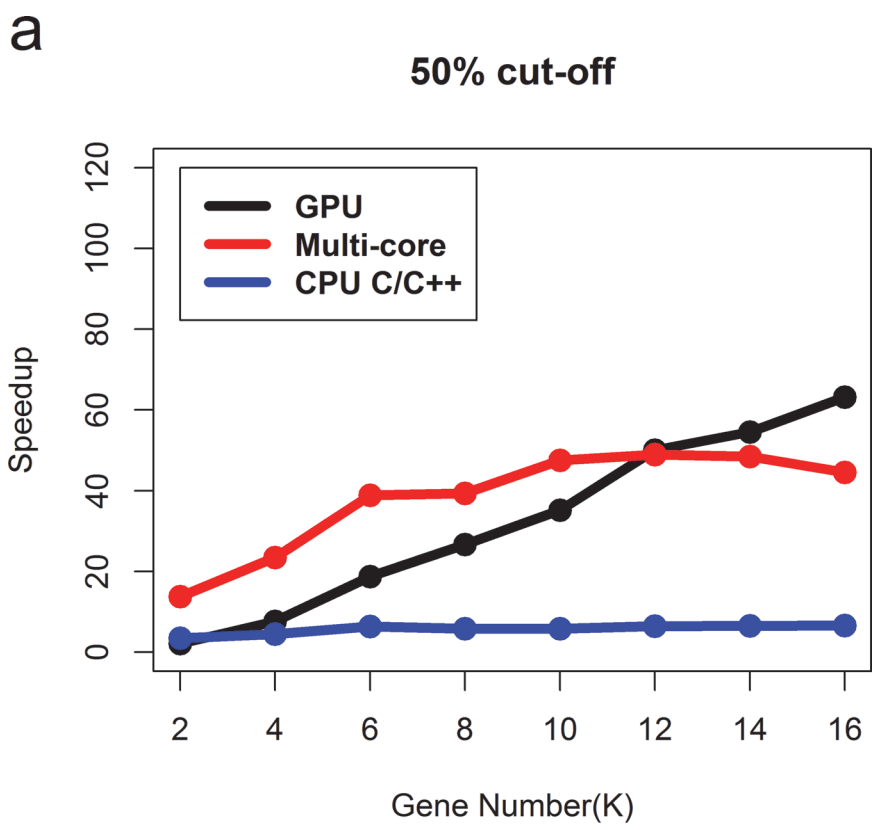
The co-expression network constructed by FastGCN is shown in Fig. 2. We identified three hub genes that are heavily connected: *CRTC1, CD3D* and *WNT16*. These hub genes were of functional importance. These hub genes have been reported to be associated with mortality [31–34]. Previous studies have also reported that *CD3D, LCK* and *ZAP70* are relatively expressed [35].

Table 2. Running time (in seconds) for the GPU implementation, Multi-core CPU implementation, Single-thread C/C++ implementation and Single-thread R implementation when no genes were filtered out during data preprocessing stage (individual number = 590).

Gene Number	GPU	Multi-core CPU	Single-thread C/C++	Single-thread WGCNA	Single-thread R
2k	0.655	0.218	1.295	4.529	3.947
4k	0.858	0.592	4.384	24.250	24.020
6k	1.295	1.591	10.311	63.971	65.400
8k	1.950	2.902	19.906	127.037	131.148
10k	3.089	4.68	31.31	220.112	221.040
12k	4.321	8.236	46.863	322.876	326.34
14k	6.771	10.968	72.758	480.106	484.56
16k	8.003	15.600	85.597	618.084	632.04

Multi-core CPU version ran on 16 CPU threads running on 16 CPU cores.

doi:10.1371/journal.pone.0116776.t002


a
50% cut-off
GPU
Multi-core
CPU C/C++
Speedup
0
20
40
60
80
100
120
2
4
6
8
10
12
14
16
Gene Number(K)

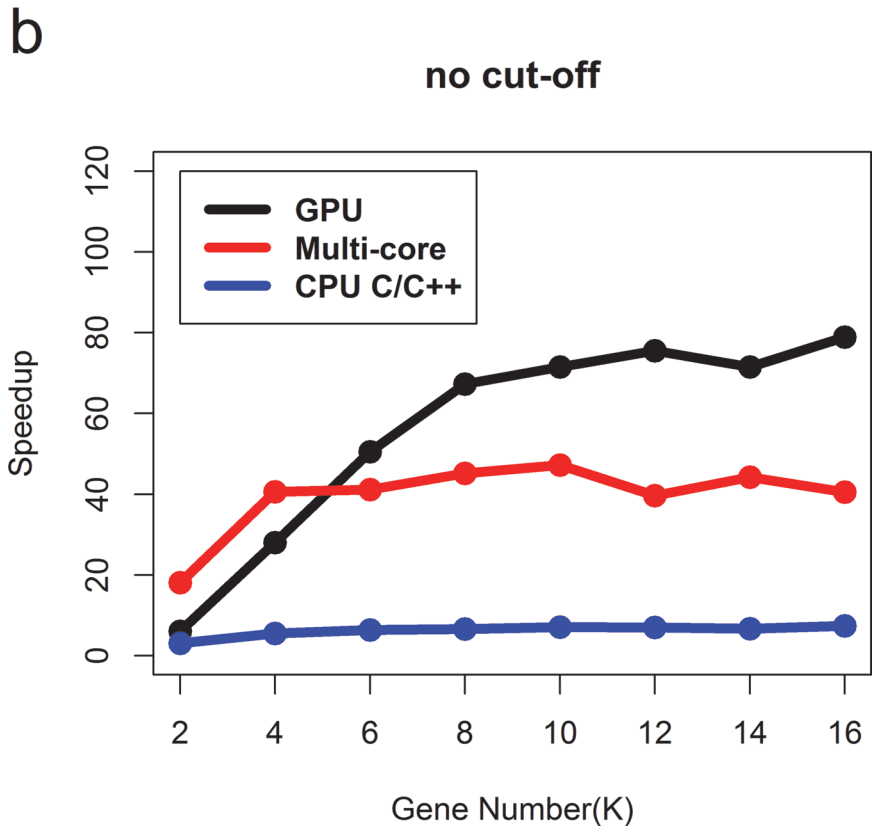

b
no cut-off
GPU
Multi-core
CPU C/C++
Speedup
0
20
40
60
80
100
120
2
4
6
8
10
12
14
16
Gene Number(K)

Figure 1. Curves of speedups against Single-thread R CPU implementation. (a) Speedup curves when 50% genes were filtered out at data preprocessing stage. (b) Speedup curves when no gene was filtered out at data preprocessing stage.

doi:10.1371/journal.pone.0116776.g001

Discussion

Numerous tools have been proposed for analyzing gene co-expression. However, the majority of these tools are not popular because they are inconvenient to use and have prohibitive computational time because of the extremely large number of tests. To address these problems, we have presented a user-friendly, accelerated and optimized tool for gene co-expression analysis. We used a novel entropy based method for data preprocessing. According to the entropy of each gene, we can make the reduction to cutoff genes with no or small expression changes. From Table 1 and Table 2, this method was shown to alleviate the computational burden. Because the gene number was reduced, the multiple tests could also be partially controlled.

Taking advantage of the parallel nature of multi-core GPUs, the parallel version of this tool can highly reduce the computational time compared to the traditional single-thread version. GPUs and CPUs have different architectures. GPUs are designed to have a higher memory bandwidth and more transistors for data parallel processing but less ability for data caching and branch control than CPUs[36]. Therefore, GPU computing is suitable for a program that has few branches and is independently executed on many data elements. We accelerated the entropy calculation, correlation calculation, z-score transformation and module identification on the GPU because these four procedures are independent and parallel and have minimal branches. Using the datasets illustrated in Table 1 and Table 2, the computation of these four procedures can be completed in a few seconds on a GPU. Additionally, we achieved promising speedups over a single-thread R or C/C++ implementation on a CPU. For example, using a dataset with 16,000 genes and 590 individuals, the entire computational procedure can be finished about 2 seconds on a GPU, which is 63 times the speed of the R implementation and 10 times the speed of the C/C++ implementation on a CPU (when we filter 50 percent of the genes in the data preprocessing step). If no genes were filtered, then the entropy calculation would not be invoked. Using the identical dataset, the computational procedure finished in 8 seconds on a GPU. This completion time was approximately 80 times the speed of the R implementation and more than 10 times the speed of the C/C++ implementation on a CPU.

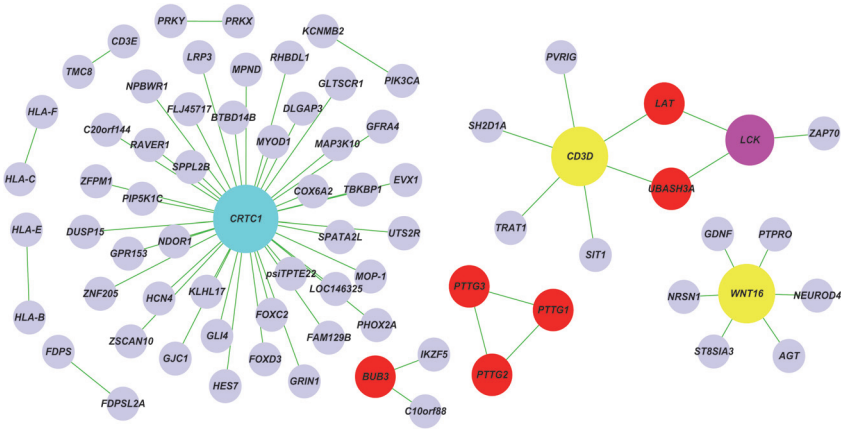


Figure 2. The co-expression network reconstructed by FastGCN using BRCA expression data with HTLV-I infection pathway.

doi:10.1371/journal.pone.0116776.g002

The GPU cannot reach its maximum potential with small datasets. The multi-core program is faster than the GPU program. However, the GPU overtakes the multi-core CPU when the dataset is greater than 6000 without a gene filter or than 12,000 genes when half of the genes are filtered. The reasons for this higher efficiency are mainly because of 1) the overhead of the schedule of the GPU kernels, 2) the overhead of the data transfer from the GPU memory to host memory, 3) and small GPU thread space created for a small dataset. More precisely, a GPU can provide a benefit only when the reduction in the computing time on the GPU exceeds the cost of the overhead. To improve the performance of the GPU, as many GPU threads should be created as possible to offset the overhead. For a small dataset, the computational complexity may not be large enough to offset the cost of the overhead. As the dataset increases, the computational workloads will increase and additional GPU threads will be created. Therefore, the GPU performance improves and eventually overcomes the multi-core CPU as shown in Fig. 1. For analyzing small datasets, we also provide the multi-core version in our source code packages. To achieve high performance on the GPU, FastGCN was designed to avoid branches, to exploit matrix compression and to store gene expression data in major columns to coalesce GPU global memory. The data fetched by one global memory access can serve as many threads as possible in a warp.

The main purpose of this article is to emphasize that the entropy reduction method and the new GPU parallel computing technology can collaboratively contribute to gene co-expression analysis. Therefore, we hope the tool we developed will be wildly used in gene expression analyses.

Supporting Information

S1 Fig. GPU architectures implemented in FastGCN. (a) GPU architecture of genetic information entropy computation. (b) GPU architecture of Pearson Correlation Coefficients computation. (c) GPU architecture of Z-score transformation. (d) GPU architecture of modules identification.
(TIF)

S2 Fig. Optimization strategies in GPU implementation. (a) mapping relationship among one-dimensional GPU thread space, two-dimensional input matrix space and one-dimensional coefficient vector space. (b) Summation reduction of the Pearson Correlation Coefficients in GPU.
(TIF)

Author Contributions

Conceived and designed the experiments: ML GJ JZ. Performed the experiments: ML FZ. Analyzed the data: ML FZ. Contributed reagents/materials/analysis tools: FZ ML. Wrote the paper: ML FZ.

References

1. Perkins AD, Langston MA (2009) Threshold selection in gene co-expression networks using spectral graph theory techniques. Bmc Bioinformatics 10. doi: 10.1186/1471-2105-10-S11-S4 PMID: 19811688
2. Guo Y, Sheng QH, Li J, Ye F, Samuels DC, et al. (2013) Large Scale Comparison of Gene Expression Levels by Microarrays and RNAseq Using TCGA Data. Plos One 8. doi: 10.1371/journal.pone.0071462 PMID: 23977046
3. Wang Z, Gerstein M, Snyder M (2009) RNA-Seq: a revolutionary tool for transcriptomics. Nat Rev Genet 10: 57–63. doi: 10.1038/nrg2484 PMID: 19015660

4. Zhao SR, Fung-Leung WP, Bittner A, Ngo K, Liu XJ (2014) Comparison of RNA-Seq and Microarray in Transcriptome Profiling of Activated T Cells. Plos One 9. doi: 10.1371/journal.pone.0078644 PMID: 24454679
5. Yang Y, Han L, Yuan Y, Li J, Hei N, et al. (2014) Gene co-expression network analysis reveals common system-level properties of prognostic genes across cancer types. Nat Commun 5: 3231. doi: 10.1038/ncomms4231 PMID: 24488081
6. Elo LL, Jarvenpaa H, Oresic M, Lahesmaa R, Aittokallio T (2007) Systematic construction of gene coexpression networks with applications to human T helper cell differentiation process. Bioinformatics 23: 2096–2103. doi: 10.1093/bioinformatics/btm309 PMID: 17553854
7. Gibson SM, Ficklin SP, Isaacson S, Luo F, Feltus FA, et al. (2013) Massive-scale gene co-expression network construction and robustness testing using random matrix theory. PLoS One 8: e55871. doi: 10.1371/journal.pone.0055871 PMID: 23409071
8. Langfelder P, Horvath S (2008) WGCNA: an R package for weighted correlation network analysis. BMC Bioinformatics 9: 559. doi: 10.1186/1471-2105-9-559 PMID: 19114008
9. Reverter A, Chan EK (2008) Combining partial correlation and an information theory approach to the reversed engineering of gene co-expression networks. Bioinformatics 24: 2491–2497. doi: 10.1093/bioinformatics/btn482 PMID: 18784117
10. Song L, Langfelder P, Horvath S (2012) Comparison of co-expression measures: mutual information, correlation, and model based indices. Bmc Bioinformatics 13. doi: 10.1186/1471-2105-13-328 PMID: 23217028
11. Wang L, Zheng W, Zhao HY, Deng MH (2013) Statistical Analysis Reveals Co-Expression Patterns of Many Pairs of Genes in Yeast Are Jointly Regulated by Interacting Loci. Plos Genetics 9. doi: 10.1371/journal.pgen.1003414 PMID: 23555313
12. Yim WC, Yu Y, Song K, Jang CS, Lee BM (2013) PLANEX: the plant co-expression database. Bmc Plant Biology 13. doi: 10.1186/1471-2229-13-83 PMID: 23688397
13. Arefin AS, Berretta R, Moscato P (2013) A GPU-based method for computing eigenvector centrality of gene-expression networks. Proceedings of the Eleventh Australasian Symposium on Parallel and Distributed Computing - Volume 140. Adelaide, Australia: Australian Computer Society, Inc. pp. 3–11.
14. McArt DG, Bankhead P, Dunne PD, Salto-Tellez M, Hamilton P, et al. (2013) cudaMap: a GPU accelerated program for gene expression connectivity mapping. BMC Bioinformatics 14: 305. doi: 10.1186/1471-2105-14-305 PMID: 24112435
15. Shi Z, Zhang B (2011) Fast network centrality analysis using GPUs. BMC Bioinformatics 12: 149. doi: 10.1186/1471-2105-12-149 PMID: 21569426
16. Zhang Q, Zhang YS (2006) Hierarchical clustering of gene expression profiles with graphics hardware acceleration. Pattern Recognition Letters 27: 676–681.
17. Borelli FF, de Camargo RY, Martins DC, Rozante LC (2013) Gene regulatory networks inference using a multi-GPU exhaustive search algorithm. BMC Bioinformatics 14 Suppl 18: S5. doi: 10.1186/1471-2105-14-S18-S5 PMID: 24564268
18. Shannon CE (1948) A Mathematical Theory of Communication. Bell System Technical Journal 27: 623–656.
19. Hasegawa M, Yano TA (1975) Entropy of the genetic information and evolution. Orig Life 6: 219–227. doi: 10.1007/BF01372408 PMID: 1153181
20. Kimura M (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. J Mol Evol 16: 111–120. doi: 10.1007/BF01731581
21. Lewontin RC (1972) The Apportionment of Human Diversity. Evolutionary Biology: 381–398. doi: 10.1007/978-1-4684-9063-3_14
22. Ohno Y, Narangajavana J, Yamamoto A, Hattori T, Kagaya Y, et al. (2011) Ectopic gene expression and organogenesis in Arabidopsis mutants missing BRU1 required for genome maintenance. Genetics 189: 83–95. doi: 10.1534/genetics.111.130062 PMID: 21705754
23. Teschendorff AE, Severini S (2010) Increased entropy of signal transduction in the cancer metastasis phenotype. BMC Syst Biol 4: 104. doi: 10.1186/1752-0509-4-104 PMID: 20673354
24. van Wieringen WN, van der Vaart AW (2011) Statistical analysis of the cancer cell's molecular entropy using high-throughput data. Bioinformatics 27: 556–563. doi: 10.1093/bioinformatics/btq704 PMID: 21172912
25. West J, Bianconi G, Severini S, Teschendorff AE (2012) Differential network entropy reveals cancer system hallmarks. Sci Rep 2: 802. doi: 10.1038/srep00802 PMID: 23150773
26. Hausser J, Strimmer K (2009) Entropy Inference and the James-Stein Estimator, with Application to Nonlinear Gene Association Networks. Journal of Machine Learning Research 10: 1469–1484.

27. Shi Z, Derow CK, Zhang B (2010) Co-expression module analysis reveals biological processes, genomic gain, and regulatory mechanisms associated with breast cancer progression. BMC Syst Biol 4: 74. doi: 10.1186/1752-0509-4-74 PMID: 20507583

28. Wang Z, San Lucas FA, Qiu P, Liu Y (2014) Improving the sensitivity of sample clustering by leveraging gene co-expression networks in variable selection. BMC Bioinformatics 15: 153. doi: 10.1186/1471-2105-15-153 PMID: 24885641

29. Zhang B, Horvath S (2005) A general framework for weighted gene co-expression network analysis. Stat Appl Genet Mol Biol 4: Article17. PMID: 16646834

30. International Cancer Genome C, Hudson TJ, Anderson W, Artez A, Barker AD, et al. (2010) International network of cancer genome projects. Nature 464: 993–998. doi: 10.1038/nature08987 PMID: 20393554

31. Canettieri G, Coni S, Della Guardia M, Nocerino V, Antonucci L, et al. (2009) The coactivator CRTC1 promotes cell proliferation and transformation via AP-1. Proc Natl Acad Sci U S A 106: 1445–1450. doi: 10.1073/pnas.0808749106 PMID: 19164581

32. Gil J, Busto EM, Garcillan B, Chean C, Garcia-Rodriguez MC, et al. (2011) A leaky mutation in CD3D differentially affects alphabeta and gammadelta T cells and leads to a Talphabeta-Tgammadelta+B +NK+ human SCID. J Clin Invest 121: 3872–3876. doi: 10.1172/JCI44254 PMID: 21926461

33. Parra-Damas A, Valero J, Chen M, Espana J, Martin E, et al. (2014) Crtc1 activates a transcriptional program deregulated at early Alzheimer's disease-related stages. J Neurosci 34: 5776–5787. doi: 10.1523/JNEUROSCI.5288-13.2014 PMID: 24760838

34. Zheng HF, Tobias JH, Duncan E, Evans DM, Eriksson J, et al. (2012) WNT16 influences bone mineral density, cortical bone thickness, bone strength, and osteoporotic fracture risk. PLoS Genet 8: e1002745. doi: 10.1371/journal.pgen.1002745 PMID: 22792071

35. Joosten M, Seitz V, Zimmermann K, Sommerfeld A, Berg E, et al. (2013) Histone acetylation and DNA demethylation of T cells result in an anaplastic large cell lymphoma-like phenotype. Haematologica 98: 247–254. doi: 10.3324/haematol.2011.054619 PMID: 22899583

36. Owens JD, Luebke D, Govindaraju N, Harris M, Kruger J, et al. (2007) A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26: 80–113. doi: 10.1111/j.1467-8659.2007.01012.x