Bystrov Alexander Sergeevich
Student
Ural Federal University
Russia, Ekaterinburg
Academic supervisor: Kovaleva Aleksandra Georgievna

GRAPHICAL PROCESSORS USE IN NON-RELATED GRAPHICS TASKS

Abstract. *Graphical Processing Units are made to speed up calculations of computer graphics. Modern GPU`s architecture allows not only calculating graphics, but also working on tasks for general computing. The main target of this paper is to analyze GPU`s performance in different tasks and to compare GPU`s CUDA architecture with standard CPU`s x86-64 architecture.*

Keywords: *GPU, CPU, algorithms, CUDA, x64.*

Быстров Александр Сергеевич
Студент
Уральский федеральный университет имени первого
Президента России Б.Н. Ельцина
Россия, г. Екатеринбург
Научный руководитель: Ковалева Александра Георгиевна

ИСПОЛЬЗОВАНИЕ ГРАФИЧЕСКИХ ПРОЦЕССОРОВ В ЗАДАЧАХ, НЕ СВЯЗАННЫХ С ГРАФИКОЙ

Аннотация. *Графические процессоры были созданы для того, чтобы ускорять обработку и отображение компьютерной графики. Современная архитектура графических процессоров не только позволяет быстрее рассчитывать графику, но также эффективно взаимодействовать с большими объёмами данных. Целью данной работы является проведение анализа работы* *графических ускорителей в различных задачах, а также проведение сравнительного анализа архитектур графических процессоров на примере CUDA с центральными процессорами на примере x64.*

Ключевые слова: *Графические ускорители, CUDA, Центральные процессоры, Алгоритмы, x64.*

Introduction

The GPU devices are designed to accelerate graphics in computing polygons, textures, shaders, etc. Due to its architecture the scope of application is quite narrow and GPU`s are not intended for general purpose processing. But in June 23, 2007 Nvidia Corporation introduced the Compute Unified Device Architecture or CUDA which is a parallel computing platform and application programming interface (API), that allows software developers and software engineers to use a CUDA-enabled graphics processing unit for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units).

The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. The CUDA API is designed to work with programming languages such as C and C++. The GPU`s design is more effective than general-purpose central processing unit (CPUs) for algorithms in situations where processing large blocks of data is done in parallel, such as fast sorting algorithms of large lists and machine learning.

The purpose of this study is to make a comparison and analysis of CPU and GPU algorithms.

The first task is Biclustering which is special grouping technique able to perform simultaneous row-column clustering (method to obtain groups of elements that share a set of common properties). Biclustering was originally introduced in the 1970s by Hartigan and it can be a NP-hard problem in which the search space is composed by all the possible overlapping subsets of elements that share a common subsets of properties. Although most of the biclustering approaches proposed in the literature are based on efficient heuristics, recently big efforts have been carried out in order to adapt them to the current scientific and technological environment, in which huge volumes of complex data that may come from different sources are generated. To deal with this challenge, High Performance Computing (HPC) techniques are used to take advantage of all available hardware and software resources to create parallel and distributed computing strategies, with the aim of solving problems that involve huge volumes of data with a high computational cost. Besides, general-purpose computing on Graphics Processing Units (GPU) and its most common programming model, CUDA, is one of the most used HPC models for massive data processing.

The second task is a bitmap quering. Efficient querying of massive data repositories relies on advanced indexing techniques that can make full use of modern computing hardware. Though many indexing options exist, bitmap indices in particular are commonly used for read-only scientific data. A bitmap index produces a coarse representation of the data in the form of a binary matrix. This representation has two significant advantages: it can be compressed using run-length encoding and it can be queried directly using fast primitive CPU logic operations. A bitmap index is created by discretizing a relation's attributes into a series of bins that represent either value ranges or distinct values. Each row in the bitmap represents a tuple from the relation. The specific bit pattern of each row in the bitmap is generated by analyzing the attributes of the corresponding tuple. A value of 1 is placed in the bin that encodes that value and a value of 0 is placed in the remaining bins for that attribute for each attribute in a tuple. One major benefit of bitmap indices is that they can be queried directly, greatly reducing the number of tuples that must be retrieved from disk.

Algorithms representation

The main problem in this field is to develop scalable high-parallel algorithm which takes full advantages of Graphical Processing Units architecture and deal with special problems like warp synchronization in CUDA based GPU`s.

This part is fully devoted to implementation of algorithms in tasks of biclustering and range-querying (bitmapping).

The first algorithm is gBiBit biclustering algorithm. The algorithm gBiBit may be divided into four steps. In first step the size of the input dataset should be reduced. The second step deals with the calculation of patterns for every pair of rows from the output dataset of the previous step. This process is suitable to be parallelized since the generation of each pattern is an independent task. In the third step duplicated patterns should be removed. This step is executed completely in CPU since this part of the code is not parallelizable. The last step which requires a huge computational cost, consists of adding rows to each valid potential bicluster to create a final result. Taking into account that the processing of every bicluster is an independent task, this part is suitable to be parallelized through the CUDA architecture.

The second group of algorithms are range-querying algorithms. All represented GPU-based range query algorithms rely on the identical preparations stage. In this stage, the CPU sends compressed columns to the GPU. After that the GPU obtains the compressed columns, it decompresses them in parallel. Once decompressed, the bit vectors involved in the query are word-aligned. This alignment makes the bitwise operation on two bit vectors an excellent fit for the massively parallel nature of GPUs. GPU range query execution strategies: column-oriented access (COA), row-oriented access (ROA), hybrid, and ideal hybrid access approaches. These approaches are analogous to structure-of-arrays, array-of-structures, and a blend thereof. Structure-of-arrays and array-of-structures approaches have been used successfully to accelerate scientific simulations on GPUs, but differ in the how data is organized and accessed which can impact GPU efficiency.

Results and Performance evaluation

Comparison of the performance of the gBiBit implementation with the BiBit original algorithm and the CUBiBit version has been presented for biclustering. Also, another version of BiBit, called ParBiBit is considered for this experimentation.

The measured execution times takes into account only the results of generation process, so the time needed for their visualization and storage is not considered. The four versions of the algorithm receive the same input parameters: a dataset, the minimum number of rows required (mnr) and the minimum number of columns required (mnc) for the final biclusters. These two last parameters have been set to 2, being the least restrictive value in order to obtain the highest number of results and, consequently, to test the performance with the maximum number of resources required. Figure 1 demonstrates the evolution of the execution times in seconds registered for BiBit, ParBiBit, CUBiBit and gBiBit, the last two executed with a single GPU device. For more details, all the execution times have been collected in Table 1.

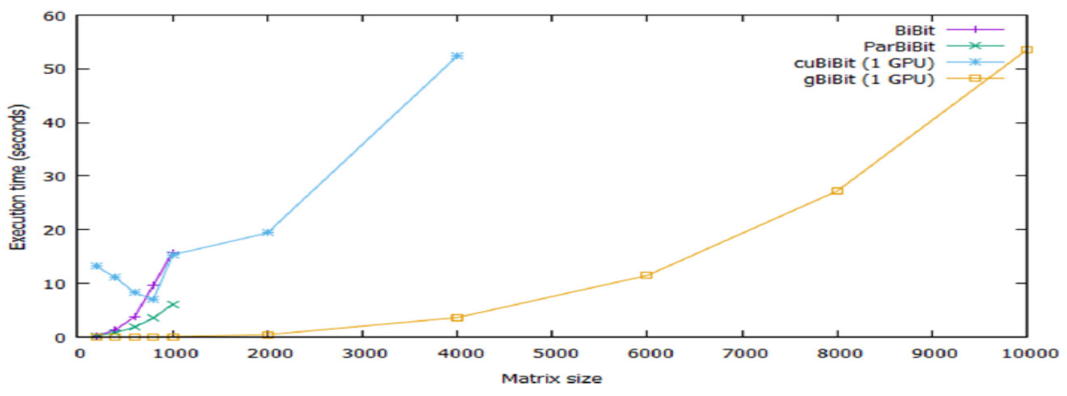

Figure 1. - Execution time in seconds for square synthetic datasets sizes from 200 × 200 to 10 000 × 10 000.

Figure 1 also shows that gBiBit presents the best execution times. The greatest difference is registered for the dataset with size 200 × 200. In this case, gBiBit is near 1900 times faster than CUBiBit, while the smallest difference is registered for the 4000 × 4000 dataset, for which gBiBit is 14 times faster than CUBiBit.

Table 1. Execution times in seconds for square synthetic datasets of various sizes.

Dataset	BiBit, s	ParBiBit, s	CUBiBit, s	gBiBit, s
200 × 200	0.219	0.21	13.16	0.007
400 × 400	1.362	0.83	11.10	0.011
600 × 600	3.754	1.84	8.28	0.020
800 × 800	9.664	3.55	7.01	0.038
1000 × 1000	15.807	6.11	15.33	0.066
2000 × 2000	n/a	n/a	19.41	0.446
4000 × 4000	n/a	n/a	52.37	3.637
6000 × 6000	n/a	n/a	n/a	11.475
8000 × 8000	n/a	n/a	n/a	27.203
10000 × 10000	n/a	n/a	n/a	53.555

Also, according to the Table 1 the gBiBit algorithm outperform ParBiBit algorithm by 17.66 times in average. The comparison of the performance of the COA and ROA implementation with the CPU algorithms with different number of cores has been represented for data-quering. Also, Hybrid and Ideal-Hybrid versions are considered for this experimentation.

Results are shown for all GPU tests compared to the iterative CPU method, organized by data set (Figure 2). Iterative CPU range query performance typically improves with additional cores for every data set. The only exception is the BPA data set when transitioning from 8 to 16 cores. The GPU methods outperform the iterative CPU method in 96.2% of these tests with an average speedup of 14.50×. The GPU methods are capable of providing a maximum speedup of 54.14× over the iterative CPU method. On the average, the GPU methods provide 1.45×, 20.24×, 11.45×, 17.36×, 18.76×, and 17.72× speedup for the KDD, linkage, BPA, Zipf (skew = 0), Zipf (skew = 1), and Zipf (skew = 2) data sets, accordingly.

The GPU methods outperform the CPU reduction method (using 16 cores) in all tests. On the average, the GPU methods provide 2.16× speedup over the reduction CPU method when using 16 cores.

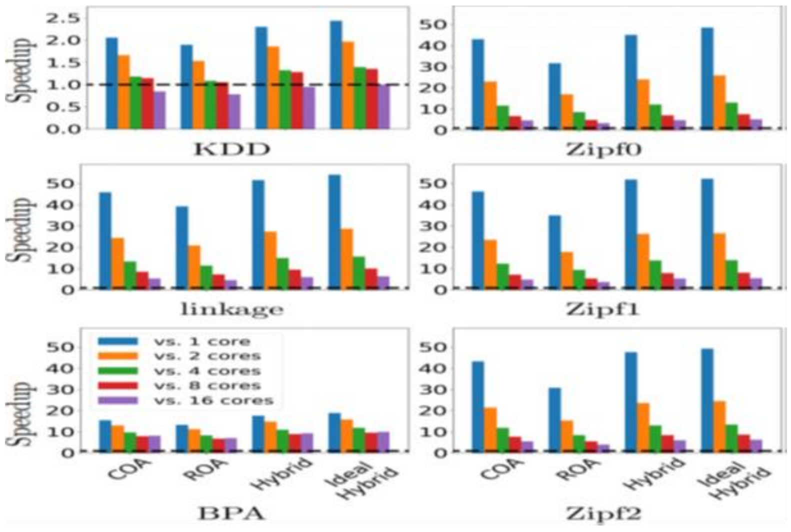

Figure 2 - Speedups (vertical axes) for the GPU methods compared to the iterative CPU method (using the number of cores shown in the legend) grouped by the GPU range query method (horizontal axes).

The Zipf data set skews are appended (e.g., Zipf0 is the Zipf data set with a skew of 0). The horizontal dashed line indicates a speedup of 1×. All plots share the same legend.

Conclusion

GPU technology and CUDA architecture are one of the most used options to adapt machine learning techniques to large and complex datasets. In the case of techniques, that are presented in this paper, they have been implemented to use GPU resources in parallel have improved their computational performance. However, this fact does not guarantee that these new algorithms can handle the processing of large datasets. There are some important issues that should be taken into account, like the data transfers between CPU and GPU memory or the balanced distribution of workload between the GPU resources. Nonetheless wide use of graphical processing units for general computing is limited to its API and programming languages, that may be used with.

REFERENCES

1. Aurelio Lopez-Fernandez, Domingo Rodriguez-Baena, Francisco Gomez-Vela, Federico Divina, Miguel Garcia-Torres// A multi-GPU biclustering algorithm for binary datasets/ Text: electronic. – 2020- URL: https://www.sciencedirect.com/science/article/abs/pii/S0743731520303701?via%3Dihub - (Reference date 25.12.2020).

2. Mitchell Nelson, Zachary Sorenson, JosephM. Myre, Jason Sawin, David Chiu2// Parallel acceleration of CPU and GPU range queries over large data sets //Pattern Recognition 2020 - Text: electronic. – URL: https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-020-00191-w -(Reference date 25.12.2020).

3. Jeremy A. Sauer, Domingo Muñoz-Esparza - The FastEddy® Resident-GPU Accelerated Large-Eddy Simulation Framework: Model Formulation, Dynamical-Core Validation and Performance Benchmarks //Text: electronic. – URL: https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2020MS002100-(Reference date 25.12.2020).