Parallel Implementation of the Wu-Manber Algorithm Using the OpenCL Framework

Themistoklis K. Pyrgiotis, Charalampos S. Kouzinopoulos, Konstantinos G. Margaritis

Parallel Implementation of the Wu-Manber Algorithm Using the OpenCL Framework

Themistoklis K. Pyrgiotis, Charalampos S. Kouzinopoulos, and Konstantinos G. Margaritis

Parallel and Distributed Processing Laboratory
Department of Applied Informatics, University of Macedonia
156 Egnatia str., P.O. Box 1591, 54006 Thessaloniki, Greece
t.pirgiot@gmail.com
{ckouz,kmarg}@uom.gr

Abstract. One of the most significant issues of the computational biology is the multiple pattern matching for locating nucleotides and amino acid sequence patterns into biological databases. Sequential implementations for these processes have become inadequate, due to an increasing demand for more computational power. Graphic cards offer a high parallelism computational power improving the performance of applications. This paper evaluates the performance of the Wu-Manber algorithm implemented with the OpenCL framework, by presenting the running time of the experiments compared with the corresponding sequential time.

Keywords: Multiple pattern matching, OpenCL, Wu-Manber algorithm, Biological sequence databases.

1 Introduction

During the last few years manufacturers of Central Processing Units (CPUs) have turned into new architectures by embracing the parallelism. This trend also affected the Graphics Processing Units (GPUs), that they have been altered from fixed function rendering devices into programmable parallel processors. Nowadays, GPUs are used instead of CPUs, for general purpose computations (GPGPU) with lots of APIs being introduced, like CUDA [3] and OpenCL [9].

The multiple pattern matching is a basic issue in computer science and especially to the area of computational biology, where the objective is to locate any nucleotides and amino acid sequence patterns into biological databases. According to [2], the multiple pattern matching problem can be defined as follows. "Given a sequence database or input string $T = t_1 t_2 ... t_n$ of length n and a finite set of r patterns $P = p^1, p^2, ..., p^r$, where each p^i is a string $pi = p_1^i p_2^i ... p_m^i$ of length m over a finite character set Σ and the total size of all patterns is denoted as $|P^r|$, the task is to find all occurrences of the patterns in the sequence database".

In this paper, the Wu-Manber (WM) [11] algorithm for multiple pattern matching problems has been implemented using the OpenCL framework, in order to succeed better performance than the corresponding sequential version of

the algorithm. The data set on which the algorithm was executed consists of biological sequence databases.

The paper is organized as follows. Section 2 presents related GPU-based for multiple pattern matching algorithms. Section 3 provides the background by presenting the functionality of the WM algorithm. Our parallel GPU-based version of the WM algorithm is introduced in Section 4. Experimental methodology and results are given in Sections 5 and 6 respectively. Finally, the conclusion of the paper is in Section 7.

2 Related Work

Multiple pattern matching algorithms executed on GPUs have been the object of study of many researchers, mostly as far as bioinformatics and intrusion detection systems are concerned.

In [7] a group of researchers suggests the creation of a WM-like multiple pattern matching algorithm for the network intrusion detected systems. The proposed algorithm is executed on a commodity graphic card, the NVIDIA GeForce 7600 GT, by using the highly parallelism computation power to inspect the packet content in parallel. OpenGL was the graphics API that has been used to execute the algorithm on the card. There have been realized experiments with random and real traffic data and has been concluded that the proposed WM algorithm is up to two times faster than the already existing modified WM used in Snort [12].

Reference [6] also refers to network intrusion detected systems, with the suggestion of a novel parallel algorithm to speedup the execution on GPUs. The new algorithm called Parallel Failureless-AC Algorithm (PFAC) is a variation of the well known Aho-Corasick (AC) [1] for multiple pattern matching. CUDA was used for the parallel implementation on GPUs. For the evaluation of the new algorithm the three implementations have created, and observed that the AC algorithm on GPU was 6.4 times faster than the sequential AC algorithm on CPU, while the corresponding proposed algorithm PFAC was 4000 times faster.

Using the CUDA API on a GeForce GTX285, in [4] researchers have developed an implementation of agrep algorithm [10], for approximate nucleotide sequent matching. For the evaluation, an OpenMP implementation of the algorithm has been developed. The experiments were executed on subsequences from 17 large genomes, and achieved 70-fold and 36-fold speedups over the multi threaded CPU version of the algorithm, for length of patterns 30 and 60 respectively.

Finally, an extension of the bit-parallel WM algorithm is presented in [13], for approximate searching. Comparing with the original, the presented algorithm has less operations. A GPU-based implementation with the OpenCL on a NVIDIA GeForce 480 has been developed and achieved a speedup of 62 over the corresponding sequential implementation.

3 Background

WM algorithm is a simple variant of the Boyer-Moore algorithm that uses the bad-character shift, for multiple pattern matching. To improve the performance, the algorithm considers the characters of both pattern and text as blocks of size B instead of single characters. As recommended in [11], the suggested value for B is $\log_{|\Sigma|} 2P$, although in practice values 2 and 3 are being used.

The operational process of the WM algorithm includes two phases. The first one is the preprocessing, where three tables are constructed by the patterns, the SHIFT, the PREFIX and the HASH tables. The SHIFT table stores the shift values of the block characters, that determine the safe shifting of characters during the searching phase. If a block of B characters does not occur in any pattern, then the shift value for that block assigns to the maximum value, which is $m - B + 1$. The HASH table stores hashed values (h) of B characters suffix of each pattern while the PREFIX stores hashed values (h') of B' characters prefix of a list of patterns that they have the same suffix.

The second part of the algorithm is the searching phase. During this phase, the algorithm is searching for the occurrences of all patterns in the input text with the assistant of the three tables that have been created by the previous state. Firstly, a hash value (h) for the block of B characters is calculated into the current search window and the shift value for that is checked (SHIFT[h]) . If the shift value is greater than zero, then the current search window is shifted by SHIFT[h] positions, or else there is a potential matching and the tables HASH and PREFIX should be considered in order to validate the matching. Further details about the WM algorithm are presented in the technical report [11].

4 OpenCL Implementation of the WM Algorithm

OpenCL is an open royalty-free standard for developing general purpose parallel programs that are executed across heterogeneous platforms consisting of CPUs, GPUs, and other parallel computing devices. It is composed of a programming language, the OpenCL C for writing the kernels, and an API for defining and controlling the platforms. The parallel execution of the OpenCL applications is taking place in the OpenCL devices during the kernel execution. Each instance of a kernel execution is called work-item (thread) and is identified uniquely into the index space also known as NDRange. Additionally, work-items can be organized into work-groups providing a coarse-grained decomposition of the index space. As fas as the OpenCL memory model is concerned, it is composed by four distinct regions, the global, the constant, the local and the private. Global and constant regions allow access to all work-items of the index space in contrast with local and private regions that allow access per work-group and per individual work-item respectively. At last, it could be said that OpenCL supports both data and task parallel programming models, with the data parallel to be the primary one [5].

For the parallel implementation of the WM algorithm the OpenCL framework was used. The basic idea to parallelize the algorithm is to cut into chunks the

input text, in our case the biological sequence, and assign each one to a work-item. Considering that the input text is of one dimension, the index space that will be used for addressing the work-items and the work-groups should be of one dimension too.

During the kernel execution, the input text and the patterns are loaded from the main memory to the global memory of the device. Each thread has two indices that indicate the working area into the input text and each one is responsible for finding all occurrences of the patterns in it. For the total occurrences of all threads, an auxiliary table is used. The size of the table is equal to the sum of all threads, where each one returns the matches for its chunk to the corresponding cell. As soon as the kernel returns, the host program is responsible for collecting and present the results of the algorithm.

The local memory of the device is much faster than the global memory, but it has a size restriction. The overall performance of the algorithm could be improved when it is possible to use the local memory instead of global. In addition, tables SHIFT, HASH and PREFIX are the most used during the searching phase. According to the two previous facts, there has been an effort of loading the three tables from the preprocessing phase to the local memory in order to achieve better performance. Initially, none of these table was able to fit into the local memory, which was in our case 16KB. Because of the limitted alphabet of the data set, 4 and 20, there has been a switching of the ASCII characters values to new ones, in order to compress the tables. Eventually, due to that procedure only the SHIFT table was able to fit into the local memory.

5 Experimental Methodology

In order to evaluate the performance of the parallel WM algorithm, the practical running time has been compared with the corresponding running time of the sequential implementation. Practical running time is the total time in seconds an algorithm needs to find all occurrences of all patterns in a text. Since the preprocessing phase has been implemented in both cases sequentially, the practical running time does not involve the first phase of the preprocessing, but only the second phase of searching.

The computer system in which the experiments were executed is composed by an Intel Core2 Duo (E8400) CPU with a 3.00 GHz clock speed, 64 KB L1 cache and 6 MB L2 cache and 3 GB of main memory. For the parallel implementation, a commodity GPU card with 1.3 compute capability was used, the NVIDIA GTX 280 with 1GB of global memory, 30 computes units and 240 scalar processors [8]. The Ubuntu Linux 10.04 was the operating system that was used and the implementations of the algorithm were developed using ANSI C programming language and version 1.0 of the OpenCL platform. Finally, for time measuring a CPU timer was used, more specifically the MPI_Wtime function of the Message Passing Interface, since it has better resolution than the standard clock function. The data set was similar to the ones used in [2]. It is consisted of:

- The genome of Escherichia coli from the Large Canterbury Corpus with size of n = 4.638.690 characters and the FASTA Nucleic Acid (FNA) of the A-thaliana genome with size of n = 116.237.486 characters. In both cases, the alphabet Σ = {a, c, g, t} is consisted of four nucleotides.
- The SWISS-PROT Amino Acid sequence database with a size of n = 182.083. 608 and the FASTA Amino Acid (FAA) of the A-thaliana genome with a size of n = 11.102.141 characters respectively. The alphabet Σ = { a, c, d, e, f , g, h, i, k, l, m, n, p, q, r, s, t, v, w, y} used by the databases is consisted of 20 different characters.

The pattern set for each execution is consisted of 100, 1.000 and 10.000 patterns respectively. All patterns have been generated randomly and each one can have a size of m = 8 and m = 32 characters. The number of work-items that were used in each execution is the maximum possible, depending on the size of the input text and the length of the patterns. Finally, for the execution of the experiments work-groups of size 512 were used.

6 Experimental Results

This section evaluates the performance of the parallel WM algorithm according to the execution time for the different biological databases. There have been carried out four experimental executions. As can generally be observed from the figures below, by varying parameters such as the size of the text, the size of the alphabet, the length of the patterns and the number of them, the performance of the algorithm can be affected.

Figures 1 and 2 present the running time of the sequential (CPU) as well as the parallel implementation (GPU) for the different executions for a pattern length of m = 8 and m = 32, and for 100, 1.000 and 10.000 patterns respectively. It should be noted that the presentation of the running time on y-axis is on logarithmic scale in contrast with the x-axis which is on linear scale. In addition, Table 1 indicates the speedups of the different executions. Speedup represents how much the parallel implementation is faster than the corresponding sequential one.

Table 1. Speedup of the WM algorithm for all the executions.

	text / **patterns**	E.Coli	FNA	FAA	SWISS-PROT
m=8	**100**	16.91	31.27	7.02	9.57
	1.000	10.37	15.89	10.29	13.54
	10.000	8.58	13.26	17.40	21.85
m=32	**100**	14.56	21.41	2.47	4.35
	1.000	9.97	13.96	7.21	13.29
	10.000	7.90	10.71	10.30	18.37

In any case, it can be observed that the performance of the parallel implementation is much better than the sequential one. It can be remarked that in the case of E. Coli and FNA biological databases, where the alphabet is of size 4, the speedup accomplished is higher than the speedup of the FAA and SWISS-PROT databases, where the alphabet is of size 20. In particular, the best values of speedup achieved are 31.27 and 21.85, for size of alphabet 4 and 20 respectively.

As far as the size of the text is concerned, it can be noticed that the larger the text, the better the performance reached. For instance, in the case of E. Coli text, where the number of the patterns is 1.000 and their length is m = 8, the speedup is 10.37. Whereas, in the case of FNA, which is 25 times larger than the E. Coli, the speedup is 15.89, while keeping all the previous parameters steady.

What can be ascertained in all experimental executions, is that assuming the number of patterns increases, the running time of both sequential and parallel executions increases too. This can be explained, as in a potential match during the searching phase, the algorithm will have to examine more patterns, as a result more computation power will be needed. The performance of the algorithm in the parallel executions has two different behaviors depending on the size of the alphabet in use, as shown in the table 1. In the case of the alphabet of size 4, as the number of the patterns increase the speedup decreases, in contrast in the case of the alphabet of size 20, as the number of the patterns increase the speedup increases too.

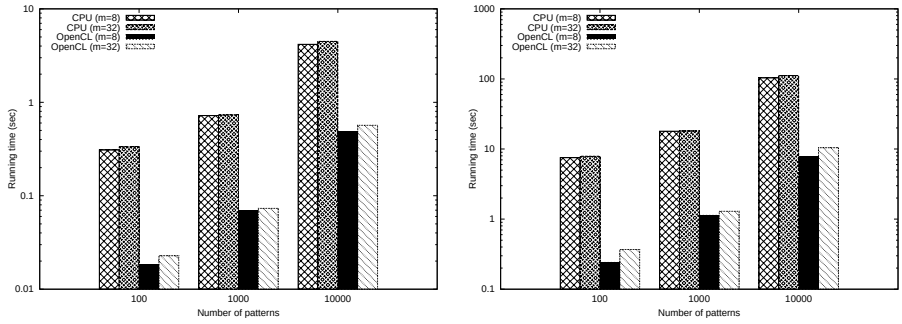

Fig. 1. Running time of the WM algoritm for the E. Coli (left) and FASTA Nucleic Acid (right) databases.

From the experiments that have been executed it is found that shift values tend to zero as long as the size of the alphabet remains small. This happens because it is possible that fragments of the same text of size B will be repeated in the patterns. As for an alphabet of larger size the shift values are greater than zero in most cases as long as the number of patterns remains small. Finally, the parameter of the size of patterns does not have a significant impact on the performance of the algorithm when the shift value is zero. The previous

observation results from the figures except for the cases where the number of patterns is 100 and 1.000 and the size of the alphabet is 20 (Fig. 2).

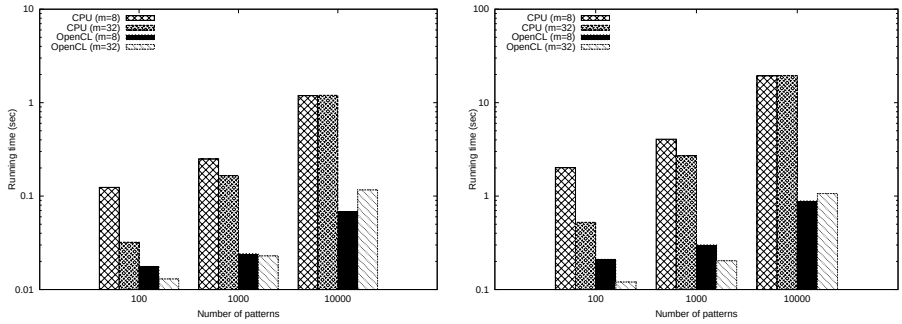

Fig. 2. Running time of the WM algoritm for the FASTA Amino Acid (left) and SWISS-PROT Amino Acid (right) databases.

7 Conclusions

In this paper the parallel implementation of the WM algorithm with OpenCL for multiple pattern matching has been presented. For the performance evaluation the practical running time of the searching phase has been compared with the corresponding sequential time. The experimental results show that the parallel implementation achieved a significant speedup of 31.27 for the best case.

It was concluded that the parallel implementation had better speedup when the alphabet of the biological data set remained small. Moreover, the algorithm had a better performance with bigger input texts than smaller ones. As far as the number of patterns is concerned, the performance of the algorithm was affected in two different ways. The speedup had a rising tendency as the patterns increased for FAA and SWISS-PROT databases where the alphabet is 20, in contrast with the Escherichia coli and FNA databases of alphabet 4, where the speedup had a decreasing tendency. Finally, regarding the length of patterns, it was shown that it did not have a significant impact on the performance when the shift values of the algorithm were close to zero.

As it was mentioned in Section 2, in [7] a parallel version of the WM algorithm has been implemented with the OpenGL and was achieved twice the performance of the corresponding WM algorithm used in Snort. Further, in [6] the AC multiple pattern matching algorithm has been implemented using CUDA, and a speedup of 6.4 has been achieved. The experimental results presented herein achieve a speedup of 31.27 in the best case, which compares favorably with the results presented in the previously mentioned references. On the other hand, two approximate string matching implementations, based on a modified

WM algorithm, [4], and the bit-parallel BPR algorithm, [13], achieve better results than those presented herein. The speedups reported are 76 and 30, in the former reference, and 62, in the latter. However, in all cases, direct comparison is not possible, since the experimental setup is not identical.

Further research efforts include the parallelization of the preprocessing phase of the WM algorithm, as well as the modification of the data structures used in the algorithm so that the GPU memory hierarchy is fully utilized, as proposed in [13].

References

1. A.V. Aho and M.J. Corasick: Efficient String Matching: An Aid to Bibliographic Search. Communications of the ACM, vol. 18, No. 6, pp. 333 –340, (1975).
2. C. S. Kouzinopoulos, P. D. Michailidis and K. G. Margaritis: Performance Study of Parallel Hybrid Multiple Pattern Matching Algorithms for Biological Sequences. In: International Conference on Bioinformatics - Models, Methods and Algorithms, pp.182–187. BIOINFORMATICS (2012)
3. CUDA Zone, `http://www.nvidia.com/object/cuda_home_new.html`
4. H. Li, B. Ni, M. Wong, and K. Leung: A fast CUDA implementation of agrep algorithm for approximate nucleotide sequence matching. In: 2011 IEEE 9th Symposium on Application Specific Processors, pp 74 – 77, (2011)
5. Khronos OpenCL Working Group: The OpenCL Specification, version 1.1 (2011), `http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf`
6. Lin, Cheng-hung and Tsai, Sheng-yu and Liu, Chen-hsiung and Chang, Shih-chieh and Shyu, Jyuo-min: Accelerating String Matching Using Multi-threaded Algorithm on GPU. pp. 1 – 5. Communications Society (2010)
7. N. F. Huang, H. W. Hung, S. H. Lai, Y. M. Chu, and W.Y. Tsai: A gpu-based multiple-pattern matching algorithm for network intrusion detection systems. In: 22nd International Conference on Advanced Information Networking and Applications (AINA). pp. 62–67, (2008)
8. Nvidia: OpenCL Programming Guide for the CUDA Architecture, version 4.0, (2011)
9. OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems, `http://www.khronos.org/opencl`
10. S. Wu and U. Manber: Agrep - A Fast Approximate Pattern-Matching Tool. In: Proceedings of USENIX Technical Conference, pp 153 – 162, (1992)
11. S. Wu and U. Manber: A fast algorithm for multi-pattern searching. Technical report TR-94-17, University of Arizona (1994)
12. Snort, `http://www.snort.org`
13. T. T. Tran, M. Giraud, and J. Varre: Bit-parallel multiple pattern matching. In: Parallel Processing and Applied Mathematics / Parallel Biocomputing Conference (PPAM / PBC 11), Torun (2011)