Code optimization of multiple sequence alignment software tool MSA_BG on GPU-accelerated computing infrastructures

Cite as: AIP Conference Proceedings **2172**, 020006 (2019); https://doi.org/10.1063/1.5133488
Published Online: 13 November 2019

Plamenka Borovska, Maria Marinova, and Vasil Tsanov

Code Optimization of Multiple Sequence Alignment Software Tool MSA_BG on GPU-Accelerated Computing Infrastructures

Plamenka Borovska [a)], Maria Marinova [b)] and Vasil Tsanov [c)]

Technical University of Sofia, 8 boul. Kliment Ohridsky, 1000 Sofia, Bulgaria
Faculty of Applied Mathematics and Informatics, Informatics Department

[a)]pborovska@tu-sofia.bg
[b)]m_marinova@tu-sofia.bg
[c)]vtsanov@tu-sofia.bg

Abstract. Multiple biological sequences alignment is one of the fundamental tasks in computational biology. The problem is NP-hard and has been subjected to intense research during the last 10 years. Exact MSA algorithms such as Clustawl have considerable serial sections (for building up the guide tree) which limit the efficiency of code parallelization and optimization. In the era of Big data, the Big genomic data ecosystem has accumulated huge amounts of genomic data provoking the challenge for innovative massively parallel algorithmic paradigms targeted for the efficient exploitation of the abandon parallel hardware resources within the high performance computing infrastructure. The goal of our investigation is to design and implement software tool for massively parallel multiple sequence alignment based on our randomized method for massively parallel multiple sequence alignment targeted for GPU accelerated computing infrastructures. Parallel performance evaluation analysis shows efficient scalability in respect to data size and machine size.

INTRODUCTION

In biology, multiple sequence alignment (MSA) of DNA, RNA or protein sequences is one of the fundamental and central tasks that should be accomplished before phylogenic reconstruction, modeling of protein structure, or gene annotation [1,2]. An increasing number of biological modeling methods depend on the precise assembly of multiple sequencing sequences. Multiple sequence alignment is a fundamental approach for research in several areas of molecular biology and bioinformatics, such virulence studies, drug and vaccine design, phylogenetic tree reconstruction, 3D structure protein prediction, and conservative domain identification. A wide spectrum of algorithms have been developed as part of the attempt to improve accuracy and effectiveness of the alignment, but there is not yet a single MSA method that can generate accurate and relevant alignments for all types of biological tests within an acceptable computational time. In the era of big data the number of the deployed genomic databases and their capacity is exponentially growing. The accumulated genomic data has been doubled up every 5 months within the last decade. The major sources of genomic data acquisition within the big genomic data ecosystem [3] (next generation DNA sequencing technologies, "omics" data generation, in silico experimental data, genome databases, cancer genome databases, and the related technologies Internet of medical Things (IomT) and cloud technologies) have accumulated huge amounts of genomic data provoking challenges for discovering innovative methods and designing and implementing fast parallel software tools for processing and analyses.

Proceedings of the 45th International Conference on Application of Mathematics in Engineering and Economics (AMEE'19)
AIP Conf. Proc. 2172, 020006-1–020006-10; https://doi.org/10.1063/1.5133488
Published by AIP Publishing. 978-0-7354-1919-3/$30.00

Exact MSA algorithms such as Clustawl [4,5] have considerable serial sections (for building up the guide tree) which limit the efficiency of code parallelization and optimization. Fast processing of MSA requires scalable parallel computing platforms and innovative massively parallel algorithmic paradigms targeted for the efficient exploitation of the abandon parallel hardware resources within the high performance computing infrastructure.

In [6,7] we have suggested massively parallel randomized method for MSA and have implemented and deployed the software tool MSA_BG_ABC based on the metaphor and algorithmic framework of the metaheuristics ABC (Artificial Bee Colony) algorithm exploiting the principles of diversification and intensification. Based on the concept of architectural and algorithmic spaces correlation we have optimized the code building up hybrid parallel MPI/OpenMP implementation that was ported on the German supercomputer JUQUEEN. Experimental results have shown speedup up to 70 times on the supercomputer JUQUEEN for machine size of 512 computing cores.

The goal of our investigation is to explore the potential parallelism of the our randomized method for massively parallel MSA in respect to GPGPU code optimization, upgrade our software tool MSA_BG and evaluate the efficiency, speed up and scalability of the optimized code on GPU-accelerated infrastructure.

RANDOMIZED METHOD FOR MASSIVELY PARALLEL MULTIPLE BIOLOGICAL SEQUENCE ALIGNMENT

The method is iterative and comprises the following steps (Fig.1):

1. Align the lengths of the biological sequences in respect to the longest ones +1 by inserting the required number of blank positions (gaps) for each of the shorter sequences so that a matrix of sequences of the same length is obtained;
2. Generate independent random multiple alignments of the input set by applying a pair of operations "insert - delete" of gaps to each sequence such that:
   The set of sequences within a working set is denoted by SEQ,
   $SEQ = SEQ_1 \cup SEQ_2 \cup ... SEQ_n$
   Where, SEQ_i is the set of symbols within a sequence i, and i is the number of the sequence within the working set of all sequences SEQ;
   The set of positions of letters (A, C, G, T - nucleotides) within a given sequence is denoted by L_i, while the set of positions of gaps within a sequence SEQ_i is denoted by G_i, $SEQ_i = L_i \cup G_i$ and $L_i = SEQ_i - G_i$, where L_i is the set of positions of letters in SEQ_i, and G_i is the set of positions of the gaps in SEQ_i
   The position to insert a gap is denoted by INS_i and the position to delete a gap - by DEL_i, obligatory $INS_i \in L_i$ and $DEL_i \in G_i$;
   a) Obligatory the position of DEL_i is different from INS_i, INS≠DEL, where DEL_i is a position of a gap within the sequence SEQ_i, and INS_i is obligatory a position holding a symbol, not a gap;
   b) In case $DEL_i > INS_i$, the symbols (letters and gaps) between positions INS_i and DEL_i are shifted one position to the right (fig.1);
   c) In case $DEL_i < INS_i$, the symbols (letters and gaps) between positions INS_i and DEL_i are shifted one position to the left (fig.2);
3. For each aligned working set the global grade of the quality of solution is evaluated as follows:
   a) For the working set of the aligned sequences for each column the number of identical symbols is counted up and for each column the symbol of maximum rate of occurrence is determined ;
   b) The maximum similarity level of each column is determined as the ratio of the number of the symbol of maximum occurrence rate to the number of the sequences within the working set (%);
   c) The global grade of the alignment is computed as the average of the maximum similarity levels of the columns.
4. The sequence of maximum similarity (sequence-favorite) is created as an ordered sequence of the column symbols of maximum occurrence rate (fig.3);
5. For each sequence of the aligned working set a counter is setup that evaluates the local similarity level as follows (fig.4):

$$Count_i = \sum_1^k S_k \qquad S_k = \begin{cases} 1\ if\ P_k = F_k \\ 0\ if\ P_k = GAP \\ -1\ if\ P_k \neq F_k \end{cases} \qquad (1)$$

Where P_k is the symbol in position k of sequence i, and F_k is the symbol in position k of the sequence-favorite.

6. Select the best alignment. The global grades of the random alignments are compared and the alignment of the highest global grade is selected as the next initial working set.
7. In case stop criteria is not satisfied – to step 2.
8. Terminate parallel computation and output results.

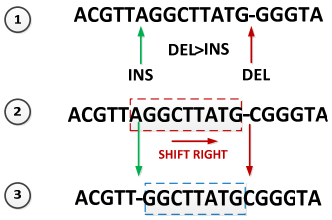

Figure 1. In case $DEL_i > INS_i$, the symbols (letters and gaps) between positions INS_i and DEL_i are shifted one position to the right

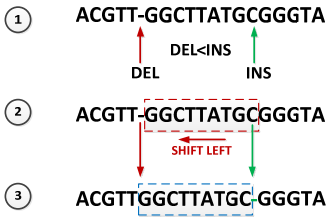

Figure 2. In case $DEL_i < INS_i$, the symbols (letters and gaps) between positions INS_i and DEL_i are shifted one position to the left

SEQUENCE OF MAXIMUM SIMILARITY (SEQUENCE – FAVORITE)

A	C	G	C	T	T	A	G	A	G	C	T	G	C	A	T	T	C	A	T	
COLUMN MAXIMUM SIMILARITY LEVEL (%) / COLUMN NUMBER																				**LOCAL SIMILARITY (COUNTERS)**
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
80	**60**	**100**	**60**	**60**	**60**	**60**	**40**	**60**	**40**	**60**	**80**	**60**	**80**	**80**	**40**	**80**	**60**	**60**	**80**	
A	**C**	**G**	**G**	**T**	**T**	**A**	**G**	**A**	**G**	**C**	**T**	**G**	**C**	**G**	**T**	**A**	**G**	**C**	**T**	**15-5=10**
-	**T**	**G**	**C**	**T**	**G**	**A**	**-**	**A**	**C**	**G**	**T**	**G**	**C**	**A**	**-**	**T**	**C**	**T**	**T**	**12+3.0-5=7**
A	**G**	**G**	**T**	**-**	**G**	**A**	**T**	**-**	**A**	**C**	**T**	**C**	**-**	**A**	**A**	**T**	**C**	**A**	**A**	**9+3.0-8=4**
A	**C**	**G**	**C**	**T**	**T**	**C**	**-**	**G**	**-**	**C**	**A**	**G**	**C**	**A**	**-**	**T**	**C**	**A**	**T**	**14+3.0-3=11**
A	**C**	**G**	**C**	**-**	**T**	**C**	**G**	**A**	**G**	**-**	**T**	**-**	**C**	**A**	**T**	**T**	**G**	**A**	**T**	**15+3.0-2=13**

Figure 3. The sequence of maximum similarity (sequence-favorite) is created as an ordered sequence of the column symbols of maximum occurrence rate (Steps 3 and steps 4 of the method)

No	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**S**
1	1	1	1	-1	1	1	1	1	1	1	1	1	1	1	-1	1	-1	-1	-1	1	**10**
2	0	-1	1	1	1	-1	1	0	1	-1	-1	1	1	1	1	0	1	1	-1	1	**7**
3	1	-1	1	-1	0	-1	1	-1	0	-1	1	1	-1	0	1	-1	1	1	1	-1	**4**
4	1	1	1	1	1	1	-1	0	-1	0	1	-1	1	1	1	0	1	1	1	1	**11**
5	1	1	1	1	0	1	-1	1	1	1	0	1	0	1	1	1	1	-1	1	1	**13**

Figure 4. For each sequence of the aligned working set a counter is setup that evaluates the local similarity level (step 5 of the method)

DESIGN AND IMPLEMENTATION OF SOFTWARE TOOL MSA_BG_CUDA FOR MULTIPLE BIOLOGICAL SEQUENCE ALIGNMENT

The software tool MSA_BG_CUDA has been designed and implemented based on the massively parallel randomized method for multiple biological sequence alignment and applying GPGPU computing in order to exploit software parallelism of the application and parallel hardware resources of the GPU-accelerated computing infrastructure. The software tool MSA_BG_CUDA is written in CUDA C++, which is efficient in creating massively parallel applications with CUDA and facilitates the deployment of high performance algorithms accelerated by thousands of parallel threads running on GPUs. The optimized code of the software tool MSA_BG_CUDA comprises kernel function that is divided into 4 blocks – each containing 256 threads (Fig.5). The multithreaded program is partitioned into blocks of threads that execute independently from each other. CUDA programs contain a sequential part, called a kernel. The kernel represents the operations to be formed by a single thread and is invoked as a set of concurrently executing threads. Kernel functions cannot be recursive, they cannot have a variable number of arguments, and they must return void. They are marked with the __global__ specifier and are called by the host.

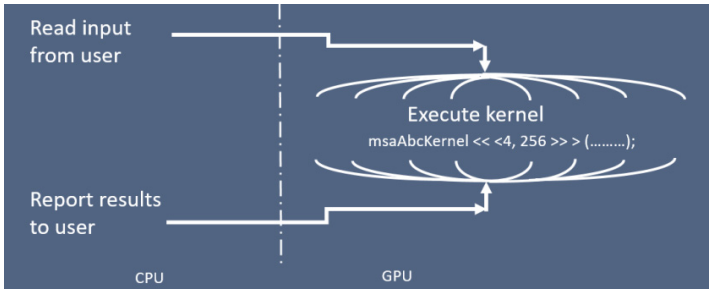

Figure 5. Example of running of kernel function msaAbcKernel on GPU – accelerated computing infrastructure

The organization of blocks and grids has impact on thread communication and synchronization. Threads within a thread block can communicate through a *per-block shared memory* (PBSM) and may synchronize using barriers. However, threads located in different blocks cannot communicate or synchronize directly. Besides the PBSM, there are four types of memory: per-thread private local memory, global memory for data shared by all threads, texture memory and constant memory. Texture memory and constant memory can be regarded as fast read-only caches.

The parallelization approach of the software tool MSA_BG for GPGPU processing (CUDA-based implementation) is shown in Fig.6.

EXPERIMENTAL FRAMEWORK

In order to verify and validate as well as to evaluate the performance and efficiency of the software tool MSA_BG we have conducted a series of experiments on two GPU-accelerated computing infrastructures (Tabl.1) and be able to make a comparative performance analysis. The nVidia GTX1080ti – based platform comprises 28 streaming multiprocessors (SMP), each of it with 128 cores, total 3584 GPU cores and six-core CPU (12 threads). The nVidia GTX1060 – based platform includes total 1280 GPU cores. The experimental GPU-accelerated infrastructures are actually highly parallel multi-core systems giving the opportunity to exploit efficiently massive data parallelism of MSA algorithms. The operating system is Scientific Linux.

The parallel computing platform is CUDA in order to use the computational power of the CUDA-enabled GPU's General-Purpose computing on Graphics Processing Units. The CUDA platform is a software layer that gives direct access to the parallel computational units for the execution of compute kernels. CUDA 10 is used as powerful software development platform for building GPU-accelerated applications providing all the components needed to build applications for NVIDIA's platforms and high performance computing (HPC) workloads [13].

GPU computing involves the following phases: (1) Copy data from main memory to GPU memory; (2) parallel computation by all cores; (3) copy the result data from GPU memory to main memory.

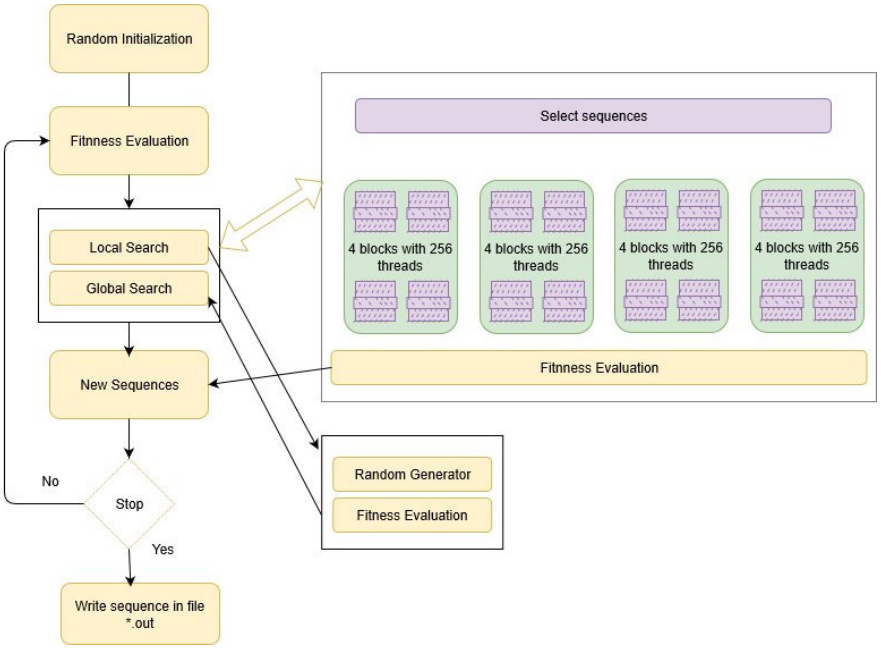

Figure 6. The parallelization approach for GPGPU processing of MSA

Table 1 Parameters of the experimental GPU-accelerated infrastructures

Parameters	**I7 GTX 1080Ti**	**I7 GTX1060**
CUDA cores/MP	SMP – 128 / 3584 CUDA cores	SMP-128 / 1280 CUDA cores
Max number of threads per MP	2048	1024
Max number of threads per block	1024	512
Memory	11GB	6GB

The benchmark datasets used to conduct the experiments comprise three groups of biological sequence datasets. The first group of datasets encompasses sequences of the Swine Flu (N1H1) viral RNA's, the second group comprises human breast cancer associated genes sequences (BRAC1 and BRAC2) and third group - test protein sequences, downloaded from NCBI [12]. The last group sequences can be classified into eight test sets: (1) 100 sequences with length of 97, (2) 100 sequences with length of 498, (3) 100 sequences with length of 1523, (5) 1000 sequences with length of 97, (6) 1000 sequences with length of 498, (7) 1000 sequences with length of 1002, and (8) 1000 sequences with length of 1523.

The goal of the experiments is to make performance comparison of performance parameters on both computing platforms (specified in Tabl.1) of the software tool MSA_BG_ABC (multithreaded version - C++, OpenMP), that is based on the massively parallel randomized method for multiple biological sequence alignment (presented in Section 1) which applies the metaphor and algorithmic framework of ABC metaheuristics and is parallelized in respect to multithreading, on the one hand, and the software tool MSA_BG_CUDA implementing the same method with GPGPU – oriented code optimization (nVidia CUDA).

EXPERIMENTAL RESULTS AND ANALYSIS

In order to make comparative analysis of the scalability in respect to machine size and data size of the software tool MSA_BG_CUDA and software tool MSA_BG_ABC we have conducted experiments with the 3 groups of benchmark datasets on the two computing platforms. The execution times for the case study of the 3 benchmark datasets groups scaling the number of iterations – 1000, 10000, 100000 are shown in Fig.7.

Execution times of the software tools MSA_BG_ABC (scaled up to 12 threads) and MSA_BG_CUDA for the case study of short sequences (100 elements – H1N1 swine virus) is shown in Fig. 8, and for the case study of sequences of length 1000 elements (swine virus H1N1) – in Fig. 9.

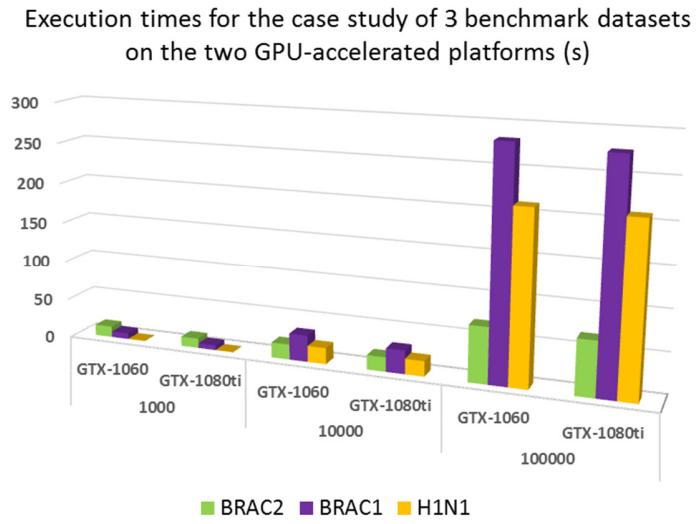

Figure 7. Scalability of MSA_BG_CUDA on GPU-accelerated infrastructures for the case study of the 3 benchmark data sets groups scaling the number of iterations – 1000, 10000, 100000

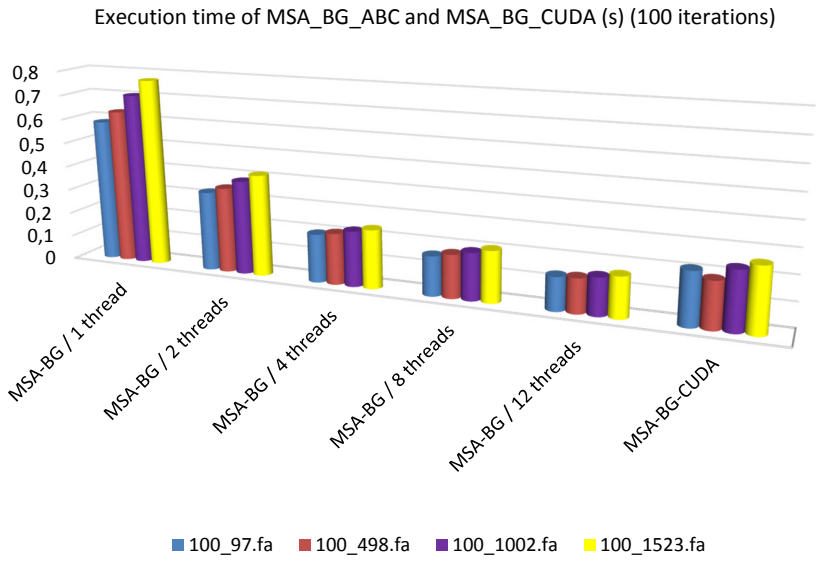

Figure 8. Execution times of MSA_BG_ABC and MSA_BG_CUDA on GPU-accelerated infrastructures for the case study of swine virus data set H1N1 for 100 iterations

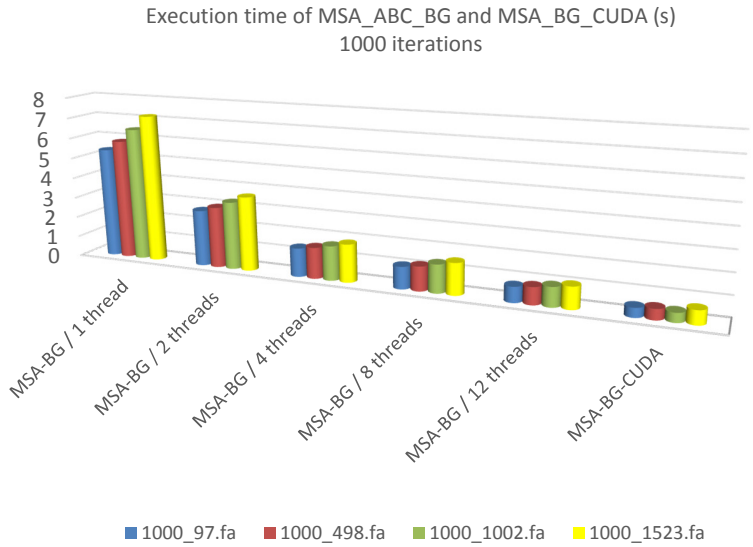

Figure 9. Execution times of MSA_BG_ABC and MSA_BG_CUDA on GPU-accelerated infrastructures for the case study of the swine virus dataset (H1N1) for 1000 iterations

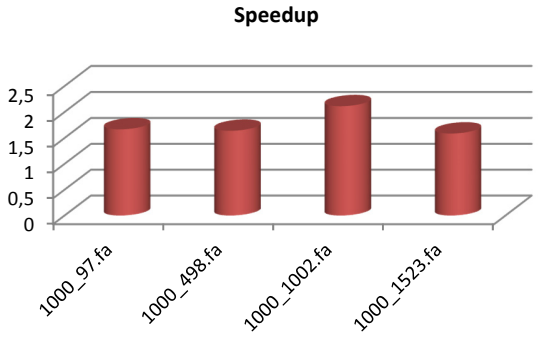

Figure 10. Speedup of MSA_BG_CUDA versus MSA_BG_ABC (12 threads) on GPU-accelerated infrastructure for the case study of the swine virus dataset (H1N1) for 1000 iterations

The estimated speedup of MSA_BG_CUDA versus MSA_BG_ABC (12 threads) on GPU-accelerated infrastructure for the case study of the swine virus dataset (H1N1) for 1000 iterations is presented in Fig.10.

Parallel activities profiling of the software tool MSA_BG_CUDA has been performed for the two GPU accelerated infrastructures under investigation. In order to perform profiling we have used NVIDIA Visual Profiler [14]. This profiler is part of the CUDA Toolkit and actually is a cross-platform performance profiling tool providing statistical data for software developers for optimizing CUDA C/C++ applications.

Timeline View of the MSA_BG_CUDA runtime on the platform GTX 1060 for the case study of H1N1 benchmark data set is shown in Fig. 11, and Timeline View of the MSA_BG_CUDA runtime on the platform GTX 1080ti for the case study of H1N1 benchmark data set is shown in Fig. 12.

Time to copy data from main memory to GPU memory - Host/GPU and vice versa – time to copy results from GPU memory to main memory - GPU/Host are important parameters to estimate. We can inspect in detail these parameters with Visual Profiler, which provides an automated analysis engine to identify optimization opportunities.

Performance analysis of experimental results shows that the designed and implemented software tool MSA_BG_CUDA targeted for GPU-accelerated computing infrastructures shows efficiency in parallelizing MSA and scales up efficiently in respect to data size and machine size.

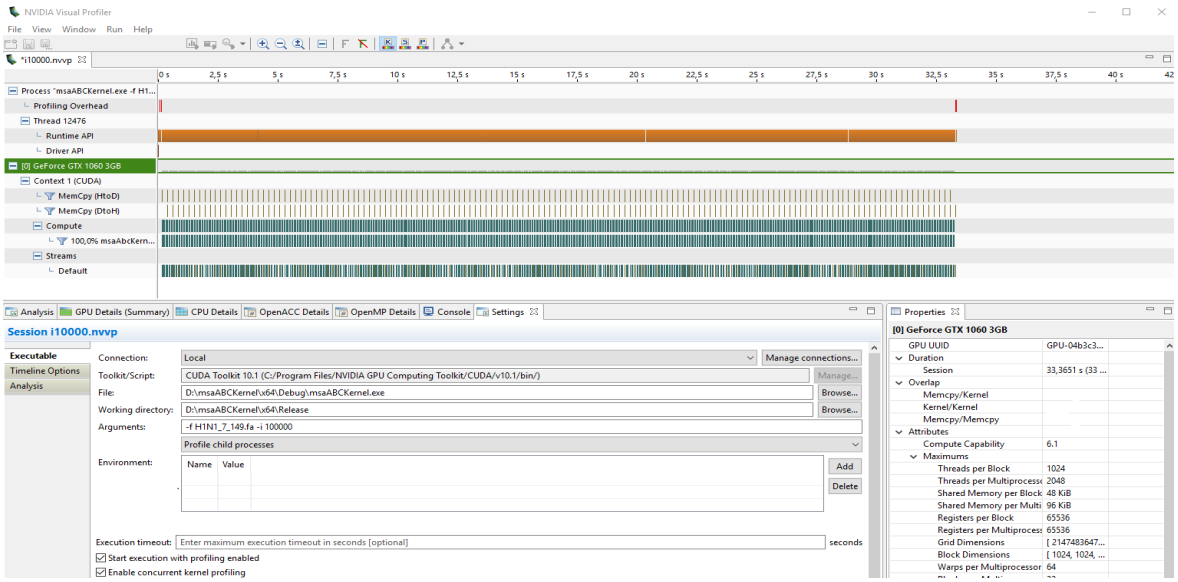

Figure 11. Profiling of MSA_BG_CUDA kernel activities on GPU-accelerated infrastructure GTX 1060 for the case study of H1N1 dataset benchmark.

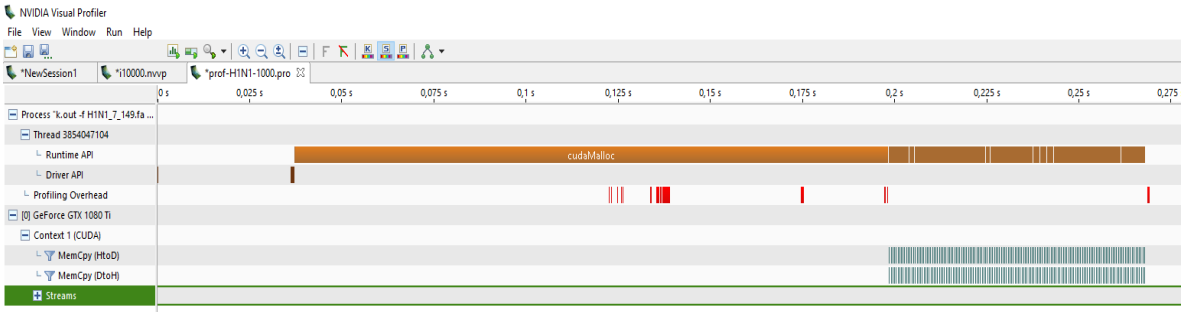

Figure 12. Profiling of MSA_BG_CUDA kernel activities on GPU-accelerated infrastructure GTX1080ti for the case study H1N1 dataset benchmark.

For small number of iterations the software tool MSA_ABC_BG outperforms the CUDA version due to the considerable time to copy data from main memory to GPU memory - Host/GPU and vice versa – time to copy results from GPU memory to main memory - GPU/Host comparable to computational time.

For large number of iterations, however, the software tool MSA_BG_CUDA outperforms MSA_ABC_BG because the portion of time to copy data from/to GPU memory/Host becomes less significant compared to the time for the sequences processing. The application of the tool MSA_BG_CUDA is targeted to processing Big genomic data and, therefore, we consider the experimental performance results satisfactory.

Scaling time consumed to copy data from main memory to GPU memory - Host/GPU and vice versa – time to copy results from GPU memory to main memory - GPU/Host on both GPU platforms for the case study of swine virus H1N1 benchmark data set in microseconds for different number of iterations is shown in Fig. 13. Scaling time of system overhead on both GPU platforms for the case study of 1000 iterations varying the length of the sequences (498, 1002 and 1523) in microseconds for 1000 iterations is shown in Fig. 14.

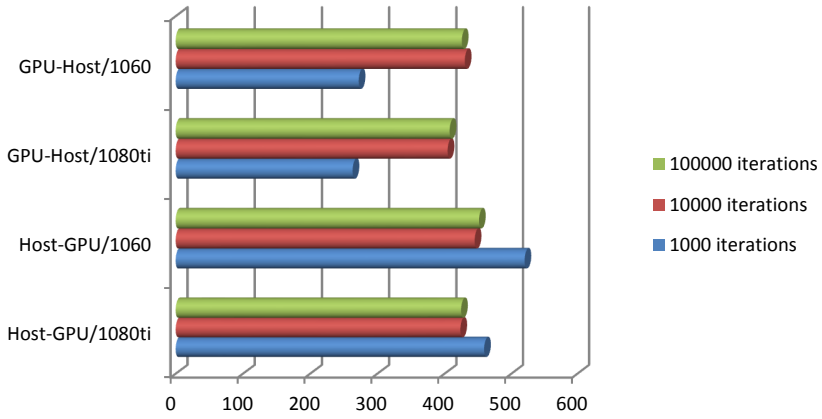

Figure 13. Scaling time consumed to copy data from main memory to GPU memory - Host/GPU and vice versa – time to copy results from GPU memory to main memory - GPU/Host on both GPU platforms for the case study of swine virus H1N1 benchmark data set in microseconds for various number of iterations (1000, 10000, 100000)

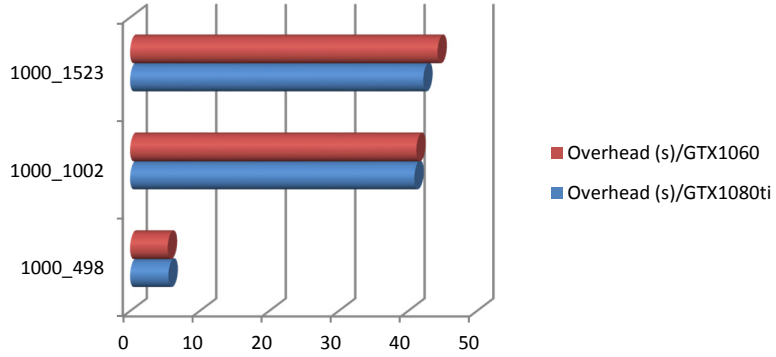

Figure 14. Scaling time of system overhead on both GPU platforms for the case study of 1000 iterations varying the length of the sequences (498, 1002 and 1523) in microseconds

Performance analysis of experimental results shows that the designed and implemented software tool MSA_BG_CUDA targeted for GPU-accelerated computing infrastructures shows efficiency in parallelizing MSA and scales up efficiently in respect to data size and machine size. For small number of iterations the software tool MSA_ABC_BG outperforms the CUDA version due to the considerable time to copy data from main memory to GPU memory - Host/GPU and vice versa – time to copy results from GPU memory to main memory - GPU/Host comparable to computational time. For large number of iterations, however, the software tool MSA_BG_CUDA outperforms MSA_ABC_BG because the portion of time to copy data from/to GPU memory/Host becomes less significant compared to the time for the sequences processing. The application of the tool MSA_BG_CUDA is targeted to processing Big genomic data and, therefore, we consider the experimental performance results satisfactory.

CONCLUSION AND FUTURE WORK

In this paper we present the design and implementation of MSA_BG_CUDA software tool based on the proposed randomized method for massively parallel multiple sequence alignment. The code is optimized for target GPU-accelerated infrastructures. We have conducted a series of experiments for the purpose of parallel performance evaluation utilizing 3 benchmark data sets – Swine Flu (N1H1) viral RNA sequences, human breast cancer

associated genes sequences (BRAC1 and BRAC2) and test protein sequences from NCBI. Experimental analysis shows efficient scalability of the parallel system in respect to machine size and problem size. Furthermore, we have performed comparative analysis of the parameters of the software tool MAS_BG_CUDA versus our software tool MSA_ABC_BG developed on the basis of the metaphor and algorithmic framework of metaheuristics ABC algorithm and ported on the supercomputer JUQUEEN. Experimental results show that for greater problem size MAS_BG_CUDA outperforms MSA_ABC_BG. In the future, we intend to use this version of the software tool MAS_BG_CUDA to be used as a prototype and port it on GPU-accelerated supercomputer infrastructure.

ACKNOWLEDGMENTS

This paper presents the outcomes of research project "Intelligent Method for Adaptive In-silico Knowledge Discovery and Decision Making Based on Analysis of Big Data Streams for Scientific Research", contract ДН07/24, financed by the National Science Fund, Competition for Financial Support for Fundamental Research – 2016, Ministry of Education and Science, Bulgaria.

REFERENCES

1. H. Carrillo and D. Lipman, "The Multiple Sequence Alignment Problem in Biology," SIAM Journal of Applied Mathematics, vol. 48, no. 5, 1988, pp. 1073-1082.
2. L. Wang and T. Jiang, "On the complexity of multiple sequence alignment", Journal of Computational Biology, vol. 1, no. 4, 1994, pp. 337–348, doi:10.1089/cmb.1994.1.337.
3. V. Marx, The Big Challenges of Big data https://www.staff.ncl.ac.uk/alan.ward/Molecular_Microbiology/Lecture2/Data_Challenges.pdf
4. C.Hung, Y. Lin, and others "*CUDA ClustalW: An efficient parallel algorithm for progressive multiple sequence alignment on Multi-GPUs*", Journal of Computational Biology and Chemistry, Elsevier 2015
5. K. Li, "ClustalW-MPI: ClustalW Analysis Using Distributed and Parallel Computing," Journal of Bioinformatics, vol.19, no. 12, 2003, pp. 1585–1586.e
6. P. Borovska, V. Gancheva, "Massively Parallel Algorithm for Multiple Sequence Alignment Based on Artificial Bee Colony", PRACE Whitepaper, 2013, http://www.prace-ri.eu/IMG/pdf/wp114.pdf
7. Borovska, P., Gancheva, V., Georgiev, I., Ivanova, D., Hybrid parallel multiple sequence alignment based on artificial bee colony on the supercomputer JUQUEEN, Proceedings - 2017 European Conference on Electrical Engineering and Computer Science, EECS 2017; Bern, EECS 2017, pp. 47-51, Electronic ISBN: 978-1-5386-2085-4, DOI: 10.1109/EECS.2017.18
8. C.Hung, Y. Lin, et all, "*CUDA ClustalW: An efficient parallel algorithm for progressive multiple sequence alignment on Multi-GPUs*", Journal of Computational Biology and Chemistry, Elsevier 2015
9. D. Karaboga, "*An Idea Based On Honey Bee Swarm for Numeric Optimization*" Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005, mf.erciyes.edu.tr/abc/pub/tr06_2005.pdf.
10. V. Gancheva, "SOA based multi-agent approach for biological data searching and integration," 4th International Conference on Applied Mathematics and Computer Science AMACS 2019, in *International Journal of Biology and Biomedical Engineering*, ISSN: 1998-4510, Volume 13, 2019, pp. 32-37.
11. V. Gancheva and H. Stoev, "DNA sequence alignment method based on trilateration," In: Rojas I., Valenzuela O., Rojas F., Ortuño F. (eds) Bioinformatics and Biomedical Engineering, IWBBIO 2019, *Lecture Notes in Computer Science*, vol. 11466, Springer, Cham, pp. 271-283, https://doi.org/10.1007/978-3-030-17935-9_25.
12. NCBI home page, http://www.ncbi.nlm.nih.gov
13. CUDA Toolkit 10.1 https://developer.nvidia.com/cuda-downloads
14. NVIDIA Visual Profiler https://developer.nvidia.com/nvidia-visual-profiler