# Improving MSAProbs Algorithm performanceand Parallel Computing using GPU

Sally Zaki El-hadary
Faculty of Computer and Information
Menofya University

Sara A. Shehab
Faculty of computers and artificial intelligence
Sadat City

Hatem Said Ahmed
Faculty of Computer and Information
Menofya University

## ABSTRACT

MSA Probs is a parallel algorithm developed to align multiple sequence alignment using a central processing unit (CPU). Whereas the CPU has some limitations, such as the inability to parallelize tasks in the processor (latency-oriented). To overcome these limitations, this paper proposes an improved version of MSA Probs that is compatible with a graphical Processing Unit (GPU). This idea helps in enhancing the performance of our algorithm (MSAprobs). To parallelize the sequential algorithm, Compute Unified Device Architecture (CUDA) or OpenCL is commonly used on GPUs. The NIVIDIA API is used to investigate the GPU's computing power. The results of using CPU only in MSAprobs versus the CPU and GPU are compared using two data sets from the Bali Base and OX Bench. The evaluation of the CPU and GPU is done using Threads 1,2 and 4. The results showed that by combining CPU and GPU, performance is improved and execution time is reduced.

## Keywords

Multiple sequence alignment, parallel processing, Latency Oriented, GPU, CPU

## 1. INTRODUCTION

MSA, or multiple sequence alignment, is crucial for bioinformatics in general. It helps a lot in bioinformatics studies. MSA has evolved into one of the essential tools for bioinformatics. It comes as a first step in the analysis that is crucial in many fields, including the explanation of the life tree, epidemiology and virulence investigations, medication development, and human genetics. the results of the MSA provide biologists with helpful data on a variety of topics. Sequence alignments in general are important in computational biology. For instance which parts of a gene are mutable can be observed and learn about the evolution of the organism (or its derived protein). Allowing for the substitution of one residue for another without changing function allows us to investigate homologous genes, find paralogs, and find orthologous genes that are evolutionarily related [10].. Multiple Sequence Alignment (MSA) methods are a group of computational techniques that, under specific circumstances, align sequences that are related to evolution while taking into consideration events like mutations, insertions, deletions, and rearrangements [11]. MSA algorithms are essentially algorithms designed to find patterns in DNA, RNA, and protein sequences. The precise approach to achieve optimal pair-wise alignment, which is used to locate MSAs, is dynamic programming (DP).

There are a number of heuristic techniques, Consequently, methods for progressive alignment, such as ClustalW [5], T-Coffee (Tree-based Consistency Objective Function for Alignment Evaluation) [6], MAFFT (Multiple Alignment using Fast Fourier Transform) [7], DIALIGN ((DNA or protein) alignment program) [8], and PRALINE (Profile Alignment) [9], have been suggested to speed up the generation of MSAs. However, the MSAs calculated by these methods do not meet the demands of biologists. Being MSAProbs based-on pair Hidden Markov Models make it from the most accurate MSA tool. Input of MSAProbs is molecular data (big data) that means that it takes long-time in running, so distance computation is one of the disadvantages of MSAProbs, it necessitates fast computation, here in our paper an idea of mixing the power of MSAProbs(accuracy)with the mechanism of GPU (parallelism) is cleared here.The contribution is to overcome the poor performance due to program latency.

This paper is organized as follows:- Section 2 Related work, Section 3 describes the Proposed Work, Section 4 discuss the experimental results and the paper is concluded in section 5.

## 2. RELATED WORK

Dynamic programming is the base of many multiple sequence alignment schemes. Using this approach, the problem at hand is divided into overlapping subproblems that are resolved separately before being combined to offer a total of two solutions. Multiple sequence alignments make the assumption that the matched sequences are homologous [1]. Despite the fact that there are several algorithms for the process of multiple sequence alignment, the accurate and fast calculation of very precise multiple alignments are still a challenge. Despite being the Clustal and T-Coffee (Tree-Based Consistency Objective Function for Alignment Evaluation) methods have historically been the most extensively utilised algorithm for multiple sequence alignments have also been shown to be slow [4]. According to CLUSTAL algorithm, if sequences share only one region of homology; this region is aligned and the rest of the sequences is ignored. Although the Mafft technique, which has gained popularity recently, uses an iterative refinement method to provide a quick alignment algorithm, it has popularity at subset alignments only(semi-local). The fuel-efficient progressive alignment method MUSCLE (Multiple Sequence Comparison by Log Expectation) is used to align a large number of nucleic acid and protein sequences not the small. The well-known technique Probcons provides sequence alignment using Hidden Markov Models (HMMs Probalign), which employs a partition function strategy. PRANK is a phylogeny-aware alignment technique that distinguishes between alignment spaces created by insert and delete operations using phylogenetic knowledge. In order to find conserved functional areas in sequences that share only local homologies but are otherwise unrelated, DIALIGN algorithms construct multiple alignments from local pairwise sequence similarities. PicXAA, anon progressive, greedy technique that emphasizes local similarities, employs regions of high local similarity to create the initial alignment that may then be repeatedly refined. Although there are a lot of algorithms for multiple sequence alignments, each of them has an advantage and teers of disadvantages, the final algorithm should be Solvable, Scalable, Accessible, Independent and meaningful algorithm, ideal algorithm, and that is so difficult. To find this algorithm we can combine with a parallelized MSA algorithm for protein sequences based on progressive alignment such as MSAProbs with CUDA (Parallel Processing GPU technique) and this is the proposed algorithm. For use on multicore CPUs, the progressive alignment algorithm MSAProbs has been multithreaded and parallelized. A progressive alignment method for computing multiple protein sequence alignments is called MSAProbs. It functions as follows:

1. Using a pair-HMM and a partition function to compute all pairwise posterior probability matrices.
2. Using the posterior probability matrices to compute a pairwise distance matrix.
3. Using the pairwise distance matrix to build a guide tree and determine the sequence weights.
4. Giving each pairwise posterior probability matrix a weighted probabilistic consistency transformation.

utilising the altered posterior probability matrices to calculate a progressive alignment along the guide tree. As a stage's post-processing step, an additional iterative refinement is carried out to further increase alignment precision. The algorithm that has been mentioned up is the traditional one. The new approach rely on combining traditional algorithm with CUDA.so lets answer the question (what CUDA?).

## 2.1 CUDA overview

(Compute Unified Device Architecture) is what CUDA stands for. It can be described as an NVIDIA parallel computer architecture. Being the use of GPUs difficult it was necessary to use CUDA for non-graphics applications. CUDA support synchronization, atomic operations, and eased memory access by revealing a general- purpose parallel programming paradigm in a multithreaded system (see Fig.1).

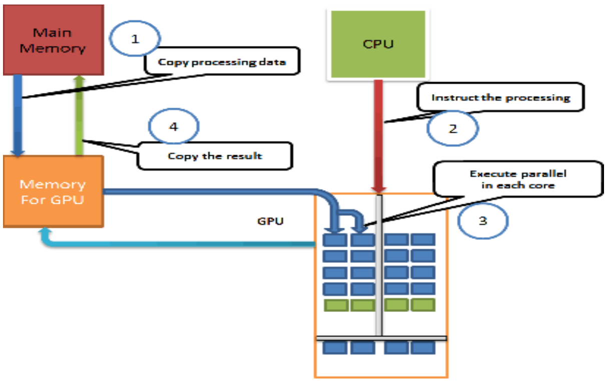


**Fig 1: CUDA Workflow that illustrate the process of CUDA**

## 3. Proposed Work

This paper proposed a Parallel version of MSAProbs algorithm, this version depends on merging traditional MSAProbs algorithm with GPU. The proposed algorithm divides the data sets between available cores using CDUA. The goal is to enhance the weak point of traditional MSAProbs algorithm (execution time) with Maintaining the rest of the features (including performance).

The CPU side will allocate data to the GPU after reading the input sequence which will be in fasta formate and it can be got from benchmark dataset tables (OX-bench and BaliBase). The alignment of MSA Sequences using a hidden Markov model which employs a partition function strategy and this make it very accurate, now is the time for guide tree Now Data will be copied from the GPU side to the CPU for printing the outcome. The alignment will be printed to the file if all data has been computed, which may be verified at the end. See fig2 for the proposed work:

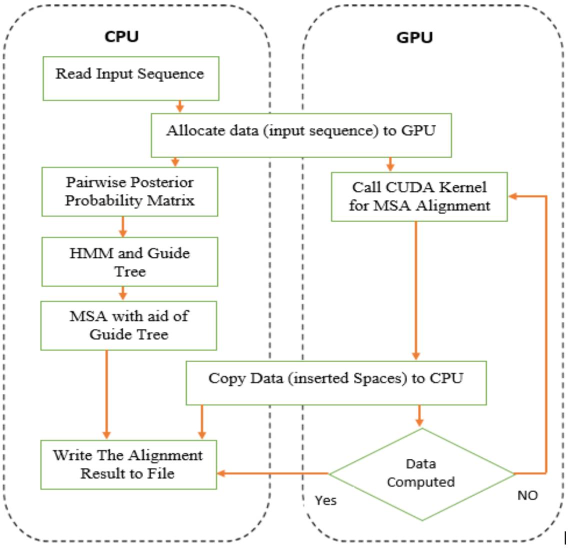

CPU
GPU
Read Input Sequence
Allocate data (input sequence) to GPU
Pairwise Posterior Probability Matrix
Call CUDA Kernel for MSA Alignment
HMM and Guide Tree
MSA with aid of Guide Tree
Copy Data (inserted Spaces) to CPU
Write The Alignment Result to File
Data Computed
Yes
NO

**Fig2: Proposed Work**

## 3.1 Model in Programming Mode

It is a single program multiple-data, and it is programming model that the GPU follow in which many items are handled concurrently by the same code. Elements of processing first read data(threads) from a shared global memory (the "pool" process) then be written back to (The "scatter" procedure). The global shared memory is divided into three random locations. This is the same code that the SIMD execution model's threads use. Co-processing mode is considered as execution mood of CUDA programs. There are two sections to the application: Sequential parts and Compute-intensive, parallel parts. Execution of application's Sequential parts can be done on the CPU (host) that is in charge of memory transfers, data management, and GPU execution setup. Implementation of kernel is done by SMs (streaming multiprocessors) and it has many light-weight threads that are running on the processing units. Because this thread's execution is organized, no matter how many parallel processors are used, the kernel scales well. Since GPUs have efficient thread management, programmers can expose much more parallelism than what is possible with hardware execution resources, at little to no cost.

Any size GPU with increased parallelism on GPUs with increased processor core and thread counts can be used to execute compiled CUDA program [5]. How much threads in a block, and how many grid blocks by altering the kernel's execution settings should be known.

Each block has a distinct identifier within a grid, and each thread has a distinct thread index within the block. Through special, built-in variables, threads can access those identifiers and dimensionality. As a rule, for processing multidimensional data, to simplify memory addressing and to branch decisions, every thread uses its own indices.

## 3.2 Memory Hierarchy

To achieve fast, parallel execution CUDA offers specific memory structure. Threads can access registers, local memory, shared memory, constant memory, texture memory, and global memory. All threads have access to sluggish, global device memory (DRAM). Data could be temporarily stored and reused in smaller memory Data could be temporarily stored and reused in smaller memory.as access to global memory is slow. The cost of accessing local memory is equal to that of accessing global memory [7]per- block shared memory is where block's threads could share data from. It is the responsibility of the programmer to move data into and out of the shared memory.

# 4. EXPERMINTAL RESULTS

Although writing a valid CUDA application that runs on any CUDA device is not difficult, it is true that performance varies significantly depending on the resource limitations of the specific device architecture.

## 4.1 Thread Scheduling and Execution

Thread blocks is what to call. Threads are executed in thread batches and that is done in a logic manner. Physically, on another side, wraps indicate to every thread block that runs on an SM in blocks of 32 threads. Blocks of thread can use their indices to partition it into wraps. Consecutive threads want it to remain in the same warp can be considered. Because the threads in a block can be executed in any sequence relative to one another, synchronization may be necessary to prevent race situations. global memory accesses and floating- point operations is a long latency operation to reduce latency, we can employ warps as an execution optimization method and this is done by all threads when selected for execution. when every thread in a warp takes the same route, SIMD style (Single instruction, multiple data) of execution works well. However, when threads within a warp diverge, there, a problem arises. since threads are allowed to follow different execution paths, all threads in a warp have their respective control pathways executed in a sequential manner and thus performance penalty what the programmer should do. Compared to using the hierarchy's other memory, global memory access is slow. We should consider that when GPU applications process enormous amounts of data, careful global memory accesses must be made to utilize the maximum bandwidth available on the GPUs. GPU memory is organized in banks to achieve high bandwidth utilization. Number of banks on NVIDIA Tesla GPU architecture is 16. one memory request per cycle can be served by each bank. Consecutive memory locations to allow simultaneous accesses should be locating. Peak bandwidth utilization, when we access global memory can be achieved. The next step is to combine memory accesses into a single memory transaction, enabling fast data delivery and memory coalescing refers to this. How can we set up memory accesses? They don't interfere with those helpful patterns? This query should be addressed by programmers.

## 4.2 Shared Memory

achieving The GPUs' great performance requires several calculations in between two global memories are being accessed. Threads cooperation between blocks in shared memory, makes reuse of data is better and helps in improving arithmetic intensity also. Ordinarily data set should be partitioned into subsets to fit in the shared memory. Subsets are loaded by thread block computation on the elements after moving data from shared memory to the global memory. Temporary data in shared memory can be stored.

Results from shared memory to global memory are written by thread block. In order to take advantage of memory coalescing, data should be loaded and stored coherently.

## 4.3 Resource Limitation

Every streaming multiprocessor in the GPU has a limited number of execution resources, such as shared memory, registers, thread block slots, and thread slots. These all threads should be dynamically partitioned among threads during the execution. You have two options first, you can handle fewer threads that use plenty of registers, Alternatively, several threads that just need a few registers. Although dynamic resource partitioning gives programmers and compilers more flexibility, it results in underutilization of resources, which lowers performance.

## 4.4 Streams

Through the use of the kernel, massive data parallelism also offers another way for tasks to overlap. The principle of streams, which is revealed by CUDA, presents a form of task-parallelism between the CPU and the GPU [8]. CPU and GPU memory copies and kernel execution are concurrent and that can help in accelerating GPU applications.This can be considered one of the tasks of streams [8] and [9].

Steps of MSAProbs:

1. calculate all pairwise posterior probability matrices and this is done by using a partition function and a pair-HMM together
2. Use pairwise posterior probability matrices to calculate a pairwise sequence distance matrix from
3. From the pairwise sequence distance matrix, create a guide tree.
4. Perform aweighted pairwise posterior probability matrices are transformed using the probabilistic consistency method.;
5. Using the altered posterior probability matrices, a profile-profile pro-gressive global alignment along the guide tree is computed.

## 4.5 Platform

The tests that were executed are run on a platform with a heterogeneous CPU and GPU and 4 GB of RAM, an 64-bit operating system,x-64 based processor and an NVIDIA GeForce GT 525M graphic card. Windows 10 pro is installed and CUDA Toolkit CUDA 11.6 is used to build the software. There are two cores in the CPU. The precise requirements for an NVIDIA GeForce GT 525M is shown in Table 1.

**Table 1 Specification of NVIDIA GetForce**

| **cuda version** | 11.6 |
|---|---|
| **GPU Compute Capability** | 2.1 |
| **cuda cores** | 96 |
| **gpu clock rate** | 600 MHz |
| **Processor clock:** | 1200 MHz |
| **Total amount of global memory** | 1536 MB |
| **memory bandwidth** | 28.8 GB/s |

| | |
|---|---|
| **Open CL support:** | 1.1 |
| **OpenGL support:** | 4.2 |
| **Pixel fill rate:** | 2.4<br>Gigapixels/s |
| **Texture fill rate:** | 9.6<br>Gigatexels/s |
| **Maximum digital resolution:** | 2560 x 1600 |
| **Maximum VGA resolution:** | 536 |

## 4.6 Dataset Used

In this paper two main data set used Bali-Base data set with roughly more than 100 sequence. Ox-bench are also used with maximum number of sequences.

### A. Impact of MSA-Prob Running Time Using CPU only and CPU+GPU (Bali-Base Data Set):-

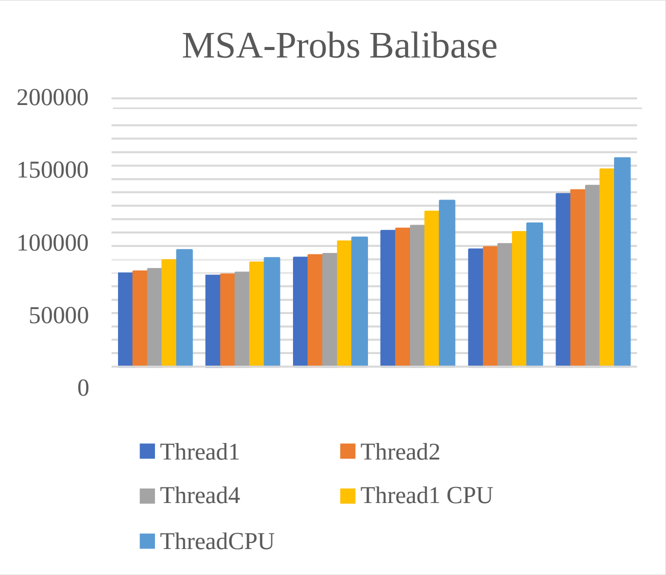
Table 2 lists the execution time of the MSA-Probs package over thread 1,2 Using CPU Only for Bali-Base and thread 1,2 and 4 using CPU+GPU.

**Table 2 Execution Time of MSA-probs using CPU only and CPU+GPU**

| | | CPU+GPU | | CPU Only | |
|---|---|---|---|---|---|
| Bali-Base | Thread 1 | Thread 2 | Thread 4 | Thread 1 | Thread 2 |
| BBS11018 | 4322.67 | 4441.25 | 4794.32 | 5486.63 | 5877.67 |
| BBS12034 | 174.246 | 190.258 | 215.485 | 273.87 | 413.27 |
| BB20004 | 178619 | 180997 | 186576 | 207140 | 222706 |
| BB30013 | 447231 | 456704 | 468439 | 530344 | 546485 |
| BB40002 | 141271 | 144007 | 149203 | 166232 | 181839 |
| BB50012 | 154421 | 157322 | 163696 | 182102 | 193533 |

### B. Impact of MSA-Prob Running Time Using CPU only and CPU+GPU (OX_BENCH Data Set):-

The graph of these result over the Bali-base and OX- bench Data sets shown in fig.3



## 5. CONCLUSION

At MSAprobs algorithm we find that the pairwise posterior probability matrix computation and the weighted probabilistic consistency transformation is the most time-consuming parts. In this case, we recommend the use of multiple threads on multi-core CPUs to accelerate the execution, considering the relatively good parallel scalability of our program. This paper proposed a parallel version of MSAProbs Algorithm using GPU computing technique especially CUDA. In the past MSAprobs was depend on CPU only in this paper the MSA-Probs is evaluated using both CPU and GPU. In MSA-CUDA , a dynamic scheduling parallelization has been proposed to parallelize the profile–profile progressive alignment stage and that result in improvement in speed.We intend to create an out-of-core pairwise probability.

## 6. REFERENCES

[1] Yongchao Liu 1, Bertil Schmidt, Douglas L Maskell,”MSAProbs: multiple sequence alignment based on pair hidden Markov models and partition function posterior probabilities.n”,2010.

[2] Budd, Aidan , "Multiple sequence alignment exercises and demonstrations". European Molecular Biology Laboratory. Archived from the original on 5 March 2012.

[3] Yongchao Liu and Bertil Schmidt ,”Multiple Protein Sequence Alignment with MSAProbs”,2014.

[4] Silberstein, Mark; Schuster, Assaf; Geiger, Dan; Patney, Anjul; Owens, John D. , "Efficient computation of sum-products on GPUs through software-managed cache" ,2008.

[5] Nickolls, J., Dally, W.J., GPU Computing era“, IEEE Micro, vol. 30, No. 2, 2010., pp. 56–69.

[6] Abi-Chahla, Fedy , "Nvidia's CUDA: The End of the CPU?". Tom's Hardware. Retrieved May 17, 2015.

[7] “Nvidia CUDA C Best Practices Guide“, version 4.0, NVIDIA Corporaton, 2011.

[8] Sanders, J., Kandrot, E.,”CUDA by Example: An ntroduction to General-Purpose GPU Programming“, Addison-Wesley, 2010.

[9] “ NVIDIA CUDA C Programming Guide“, v rsion 4.0,

NVIDIA Corporaton, 2011.

[10] An Overview of Multiple Sequence Alignment Systems Article February 2009 Fahad Saeed & Ashfaq A. Khokhar.

[11] Multiple sequence alignment modeling: methods and applications November 2015 Briefings in Bioinformatics 2015(6) Carsten Kemena , Jia-Ming Chang, Cedrik Magis&Maria Chatzou