# Aligning Multi Sequences on GPUs

Hong Phong Pham[1], Huu Duc Nguyen[1], and Thanh Thuy Nguyen[2]

[1] Department of Information System, Hanoi University of Science and Technology, NOT is High Performance Computing Center
[2] Department of Computer Science, VNU - University of Engineering and Technology
{phongph.hut,ducnh.hut}@gmail.com,
nguyenthanhthuy@vnu.edu.vn

**Abstract.** Implementing Multi Sequence Alignment (MSA) problem using the method of progressive alignment is not feasible on common computing systems; it takes several hours or even days for aligning thousands of sequences if we use sequential versions of the most popular MSA algorithm - Clustal. In this paper, we present our parallel algorithm called CUDAClustal, a MSA parallel program. We have paralleled the first stage of the algorithm Clustal and achieved a significant speedup when compared to the sequential program running on a computer of Pentium 4 3.0 GHz processor. Our tests were performed on one GPU Geforce GTX 295 and they gave a great computing performance: the running time of CUDAClustal is smaller approximately 30 times than Clustal for the first stage. This shows the large benefit of GPU for solving the MSA problem and its high applicability in bioinformatics.

**Keywords:** Multi sequence alignment, Clustal, CUDA, GPU.

## 1 Introduction

Bioinformatics is an important field which affects almost sides of people life, it mainly go along with genetics and the science of researching genes. This paper solves problem of multi sequence alignment (MSA) problem [2]. The challenge here is to find out how to align thousands of ADN, ARN or protein sequences to identify similar residues and regions between them. There are methods of resolving this problem such as dynamic programming: Needleman-Wunsch [3], Smith-Waterman [1], progressive alignment methods: Clustal [4], T-Coffee [10].

However, those proposed methods face computational performance problem because the computational complexity of algorithms is very large. In the case of the algorithm Clustal, the complexity is $O(n^2 \times l^2)$, in which $n$ is the number of sequences and $l$ is the average length of sequences. This means that to align multi sequences with a dataset of 1000 sequences with the average length of 500, it takes several hours if using common CPUs. Therefore in this paper, we took advantage of the great computing power of GPU to grow computational performance. Results have shown a significant increase of computational performance when compared to the sequential program, demonstrating GPU's high applicability in bioinformatics field.

P.C. Vinh et al. (Eds.): ICCASA 2012, LNICST 109, pp. 300–309, 2013.

### 1.1 Pairwise Sequence Alignment Problem

In bioinformatics fields, one of the most important problems is sequence alignment problem, including two main problems: pairwise sequence alignment (PSA) [15] and multi sequence alignment (MSA) [2]. Firstly, we examine the problem PSA; it is the foundation for the MSA problem. Supposing that we have a pair of sequences {A,B} satisfying the following properties:

- $A = a_1 a_2 \dots a_{l_a},\ B = b_1 b_2 \dots b_{l_b}$
- $a_i, b_j \in R\ (1 \le i \le l_a, 1 \le j \le l_b)$
- $R$ is a set of given characters
- $R \not\ni$ '-' (gap)

An alignment of the pair of sequences {A,B} results in another sequence pair of {A',B'} satisfying all the following properties:

- $A' = a_1' a_2' \dots a_{l'}',\ B' = b_1' b_2' \dots b_{l'}' (l' \ge l_a, l_b)$
- $a_i', b_j' \in R \cup \{\text{'-'}\}\ (1 \le i, j \le l')$
- If removing some gaps, $A'$ will become $A$ and $B'$ will be $B$
- $\nexists i: a_i' = b_i' =$ '-' $(1 \le i \le l')$

In the pairwise sequence alignment problem, the sum of scores of all character pairs is defined as the score of alignment. The optimal alignment is the one with the highest score. The score of this optimal alignment is called the similarity of the two given sequences. A popular ranking method is using a weight matrix; in this paper, we use the popular weight matrix BLOSUM.

### 1.2 Multi Sequence Alignment Problem

Supposing that we have $n$ sequences $\{A_1, A_2, \dots, A_n\}\ (n \ge 3)$satisfying all the following conditions:

- $A_i = a_{i,1} a_{i,2} \dots a_{i,l_i}\ (1 \le i \le n)$
- $a_{i,j} \in R\ (1 \le i \le n,\ 1 \le j \le l_i)$
- $R$ is a set of given characters
- $R \not\ni$ '-' (Gap)

A solution of aligning multi sequences $\{A_1, A_2, \dots, A_n\}$ is a set of sequences $\{A_1', A_2', \dots, A_n'\}$ satisfying all the following conditions:

- $A_i' = a_{i,1}' a_{i,2}' \dots a_{i,l'}'\ (l' \ge l_1, l_2, \dots, l_n)$
- $a_{i,j}' \in R \cup \{\text{'-'}\}\ (1 \le i \le n, 1 \le j \le l')$
- If removing gaps then $A_i'$ will become $A_i\ (1 \le i \le n)$
- $\nexists j: a_{1,j}' = a_{2,j}' = \dots = a_{n,j}' =$ '-' $(1 \le i \le l')$

```
TTGACATG CCGGGG---A AACCG
TTGACATG CCGGTG--GT AAGCC
TTGACATG -CTAGG---A ACGCG
TTGACATG -CTAGGGAAC ACGCG
TTGACATC -CTCTG---A ACGCG
```

**Fig. 1.** An example of aligning multi sequences.

Methods of grading for MSA alignments have the same principles with PSA. Major approaches of solving the MSA problem include:

- The dynamic programming method: this method is not used to directly solve the MSA problem because the size of required memory have the exponential increase.
- The progressive alignment method: the most used method for MSA. This heuristic method uses the approach of "progressive": aligning sequences which are "close" to each other and then adding progressively further sequences into the current alignment. The most used method in progressive alignment is the algorithm **Clustal** [13]. This algorithm includes three steps which are presented in below section. Steps of calculating distances and aligning multi sequences require the large computing power which common computing systems do not meet within an accept time, therefore we implemented these steps on GPU using the parallel programming language CUDA and achieved a very great performance.

## 2 GPU and Programming Model CUDA

In recent years, the computing power of GPU graphics processors has increased significantly compared to CPU. Until June 2008, NVIDIA's GPU GT200 generation has reached the threshold of 933 GFLOPS, more than 10 times over dual-core processor the Intel Xeon 3.2 GHz at the same time. The computing performance of GPUs is only seen in problems that one certain task can be executed on a lot of independent data concurrently and then each processing core of GPU is assigned to perform one task on a set of data. CUDA [11] is popular software which supports to develop applications on multi cores GPU. A CUDA program includes one or a few special pieces of code, called parallel *kernels*. These kernels can be executed in parallel on the large number of threads on GPUs. Threads are divided into small groups which are executed on the same streaming multiprocessor, called thread *blocks*, these blocks are also designed to a *grid*. GPU's memory is hierarchically organized for effective usage:

- Main memory: the memory area for CPU code. Only this code can access and modify information here.
- Global memory: the memory area that all GPU threads can access to it. Programmers can move data from main memory to global memory by using functions from a CUDA basic library. This memory is often used to store inputs and outputs for parallel threads on GPUs.

- Shared memory: the memory area that only threads in one same block can access. This memory is integrated on-chip; so the speed of accessing data on it is much higher than on global memory. This memory is often used to store temporary shared data among threads in a block to speed up the process of memory usage.
- Local memory: the memory area allocated to local variables of each thread and one GPU thread cannot access to those from others.

With the ability to perform data parallelism on such a lot of threads, GPU is an appropriate choice to solve the multi sequence alignment problem, in which threads can calculate one cell on sub-diagonals of the similarity matrix in parallel or calculate distances between sequences concurrently, as presented in the below section.

# 3 Parallel Clustal on GPU

## 3.1 Overview of the Algorithm Clustal

As mentioned above, in the approach of *progressive* alignment to solve the MSA problem, Clustal [13] is one of the most used algorithms. This algorithm includes three stages; the following is details of three stages:

1. *Align Pairwise Sequences*
At first step, we do alignment all pairs of sequences to calculate relative distances about evolution between all sequences. The common method is dynamic programming, in which the value of distance between two sequences is calculated as compensative value of point for these two sequences. The implementation of the algorithm Clustal in this paper uses a simple version of *Smith-Waterman* algorithm. Firstly, the algorithm does alignments for all pairs of sequences $\{A_i, A_j\}$ $(1 \leq i < j \leq n)$ which can be generated from n input sequences $\{A_1, A_2, \dots, A_n\}$ and using the score of the optimal alignment to calculate one distance value $dist(A_i, A_j)$ for those pair of sequences. The value of distance is calculated as follows: for a pair of sequences $A_x = a_{x,1}a_{x,2} \dots a_{x,l_x}$ and $A_y = a_{y,1}a_{y,2} \dots a_{y,l_y}$, we create a similarity matrix H with the size of $(l_x + 1) \times (l_y + 1)$ as follows:

- $H_{i,0} = H_{0,j} = 0$ $(0 \leq i \leq l_x, 0 \leq j \leq l_y)$ (sentries) (1)
- $$H_{i,j} = max \begin{pmatrix} 0, \\ H_{i-1,j} - 1, \\ H_{i,j-1} - 1, \\ H_{i-1,j-1} + sub(a_{x,i}, a_{y,j}) \end{pmatrix} \begin{pmatrix} 1 \leq i \leq l_x, \\ 1 \leq j \leq l_y \end{pmatrix} \quad (2)$$

After building the matrix H, the distance value is calculated as follows:

$$dist(A_x, A_y) = 1 - H_{l_x,l_y}/min(l_x, l_y) \quad (3)$$

2. *Build the phylogenetic tree*: In this step, distance values are used to create a binary phylogenetic tree by algorithms of clustering. Here we apply a simple version of the popular clustering algorithm *neighbor-joining*.

3. *Perform multi sequences alignment*: the phylogenetic tree is used to perform multi sequences alignment progressively. Aligning multi sequences is executed by using a version of the algorithm Needleman-Wunsch [3].

### 3.2 Parallel the Algorithm Clustal

In this paper, we show how to use the parallel computing technology CUDA on GPU to accelerate the algorithm by paralleling stages of the algorithm Clustal and evaluate the performance of the GPU-based algorithm.

#### 3.2.1 Basic Strategy of Parallelization

Parallelization strategy of the algorithm can be divided into two directions: paralleling the creation of similarity matrices and paralleling the calculation of distance values between sequences. To calculate distance values, we can see that distances of all pairs of sequences can be calculated independently; therefore we can calculate for all distances in parallel. In parallelization of building similarity matrices, from mathematical formulas of the algorithm Smith-Waterman in part 3.1, calculation of one element in the matrix only depends on values of its left, upper and upper-left neighbors. This is described as Fig.2.

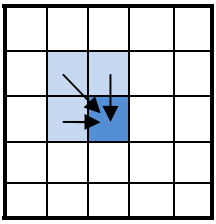

**Fig. 2.** The dependent relationship between values of elements in a similarity matrix

So all elements on the same diagonal of the matrix are directly or indirectly complete independent to each other. Therefore, calculation of these elements can be executed in parallel theoretically. This leads to basic parallel strategy as follows:

- Calculate all elements on the same diagonal concurrently.
- Diagonals are sequentially calculated in order of from upper-left to down right.

#### 3.2.2 Parallelization Using CUDA on GPU

One significantly modified parallel method from above basic parallel strategy to improve the performance is proposed in the program MSA-CUDA, one parallel implementation of the algorithm Clustal by Yongchao Liu and Maskell in [17]. MSA-CUDA shows two new parallel methods: intra-task parallelization and inter-task parallelization. Here, one task is defined as calculating the distance between a pair of sequences - calculating a similarity matrix of the pair of sequences. In intra-task parallelization, one task is assigned to one block of threads and all threads inside that block combine to perform the assigned task. For inter-task parallelization, one task is

assigned to one thread and threads inside a block do not combine to each other. Moreover, calculation of a similarity matrix in MSA-CUDA is also modified with the method of cell block for inter-task parallelization. In this method, each matrix is calculated by units of blocks of elements which are of sub-matrices of the size of n×n of the original matrix instead of every single element.

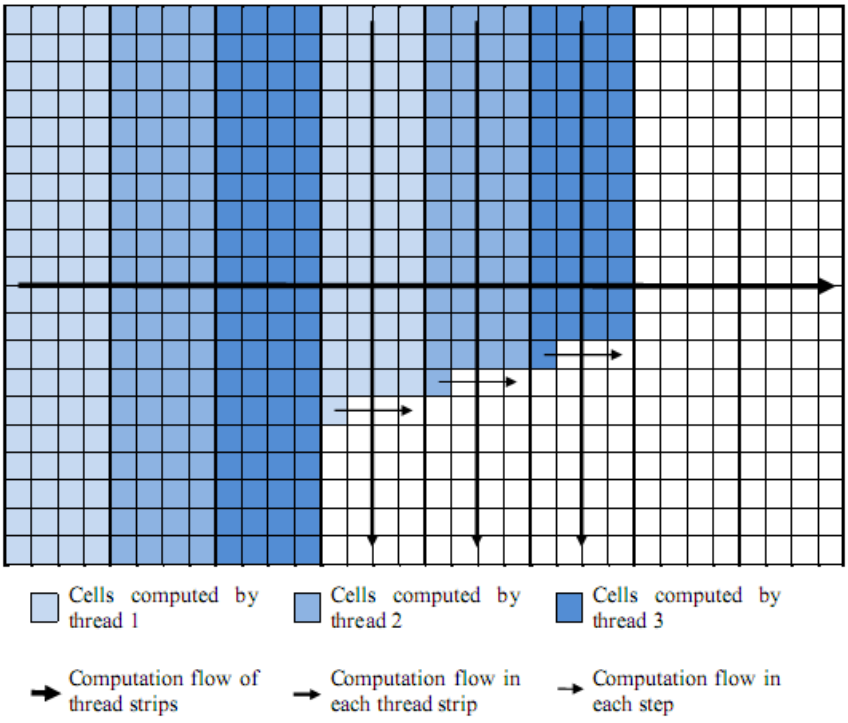


**Fig. 3.** Illustration of calculating the matrix by intra-task parallelization of strip-wise

### 3.2.3 Parallelization for Stage 1

To parallelize stage 1 of the algorithm Clustal, we utilized an original "strip-wise parallelization" approach on GPUs, based on the idea of "cell block" found in MSA-CUDA and Hirschberg's algorithm [5] for sequence alignment with linear memory. Similar to MSA-CUDA, our approach is split into two flavors: intra-task parallelization and inter-task parallelization.

***a) Intra-task parallelization***

In intra-task parallelization, each similarity matrix is assigned to a whole block, and all threads in the block cooperate to compute the matrix in question. The similarity matrix is split into vertical "block strips" with fixed horizontal size (about 4-16 cells) and vertical size equal to the matrix's vertical size. Each block strip will be computed almost simultaneously by all threads in the block (except the last strip).

Next, each block strip is spilt into a fixed number of thread strips, each of which corresponds to a thread in the block. Hence, the number of thread strips in each block strip (except for the last) is equal to the number of threads in the block. Each thread strip, in turn, is split into steps. Each step consists of one row of cells in the thread strip.

Since the computation of each cell depends on the cells to its left, above and upper-left, the block strips will be computed from left to right, the steps will be computed from top to bottom and the cells in each step will be computed from left to right. Also, the thread strips will not start computing simultaneously, but the thread strips on the left will start first and work gradually to the right. This can be seen in the figure above. After each step is finished, all threads will be synchronized once before moving on to the next step.

In this way, the similarity matrix can be computed by every thread in the block (except for the beginning and the end of each block strip), which reduces computation time. However, the main advantage of strip-wise intra-task parallelization lies with the fact that the block strips and thread strips are of fixed widths. This means that one can utilize Hirschberg's algorithm to store all the intermediate results on shared memory or even registers, as seen in Fig. 4, without the worry of memory overflow due to their limited sizes.

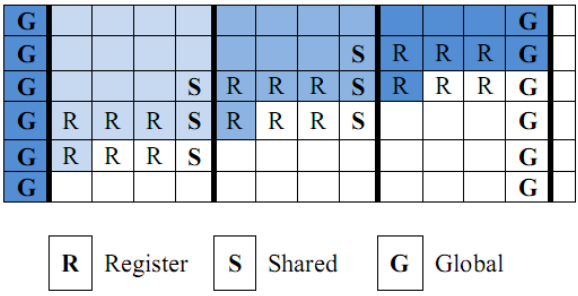


**Fig. 4.** Usage of memory in intra-task parallelization

Here only the cells computed in the last step and the cells being computed in the current step are stored. These cells can be stored in register, or in shared memory to take advantage of array structure. The cells on the border with the next thread strip on the right must be stored in shared memory to allow the next thread strip to use the results for the computation of its border cells. Only the cells bordering the next block strip are stored in global memory to be used later. The right-most columns of cells in the matrix will not required global memory as only the cell at the bottom-most does matter at that point.

This way of memory usage should sharply reduce memory latency compared to relying extensively on global memory to store intermediate results (eg. In MSA-CUDA's intra-task parallelization). In the Fig.4, global memory is accessed only once for the computation of 12 cells. And in the actual software the rate is 1:512. This means that for matrix with width of less than or equal to 512 cells, global memory

will not be used at all in the computation of the matrix itself, though it is still used to create sentry at the start of the computation in the actual software.

***b) Inter-task parallelization***

Essentially, inter-task parallelization is a watered-down version of intra-task parallelization, with each similarity matrix being assigned to a thread instead of a whole block. The computation are carried out pretty much the same as intra-task parallelization, albeit with only one thread the "block strip" and "thread strip" become one.

Inter-task parallelization suffers greatly in performance improvement for the computation of a single similarity matrix compared to intra-task parallelization, having all the cells in the similarity matrix computed consequentially, and a higher rate of global memory usage. Nonetheless, it makes up for it in computing more similarity matrices in parallel than intra-task parallelization, since now each thread computes one matrix instead of each block.

Inter-task parallelization, therefore, is more suitable for cases involving larger number of very short sequences. In these cases, because the size of the similarity matrices may be much smaller than the fixed size of one single block strip, many threads will go idle during the computation and are wasted. Intra-task parallelization should be used in cases with small number of very long sequences, in which the huge "jump" of a whole block strip can be fully exploited.

### 3.2.4 Parallelization for Stage 3

Theoretically, strip-wise parallelization approach can also be applied to the computation of similarity matrices in Stage 3 of the Clustal algorithm. However, due to time constraints, this has not been implemented in the actual software as of yet.

Apart from the parallelization of similarity matrix computation, multiple progressive alignments in multiple internal nodes can be carried out in parallel, as long as these internal nodes are not independent in computation from each other. This has been implemented successfully in MSA-CUDA, so we would not dig further into this.

## 4 Experiments and Evaluation

Our parallel algorithm - CUDAClustal in this paper were implemented and tested on one GPU Geforce GTX 295 in a PC running the Linux OS. In our tests, we extract data from the database UniProtKB/Swiss-Prot for testing. UniProtKB is a database containing a large amount of biological information about proteins; in which Swiss-Prot is the part evaluated and edited by hand. The algorithm is paralleled on four different datasets, with two cases: a large number of short sequences and a small number of long sequences. Details of data are shown in the Table 1.

CUDAClustal shows a significant improvement of performance when compared to sequential versions of the algorithm Clustal. If evaluating the time of running the entire steps, CUDAClustal is faster two times than Clustal when working with datasets with the large number of sequences (test1.fasta, test2.fasta) and is faster three times than Clustal in the case of datasets with large average lengths of sequences (test3.fasta, test4.fasta). These results are described in Table 1.

**Table 1.** Comparison of runtime between CUDAClustal and Clustal

| Dataset | Number of sequences | Average length of sequences | Runtime | |
|---|---|---|---|---|
| | | | CUDAClustal | Clustal |
| test1.fasta | 200 | ~300 | 120,13s | 220,25s |
| test2.fasta | 300 | ~300 | 269,88s | 545,25s |
| test3.fasta | 100 | ~800 | 110,82 | 327,51 |
| test4.fasta | 50 | ~1600 | 121,65s | 332,2s |

Currently, CUDAClustal has been paralleled for only stage one of the entire algorithm, so to evaluate the computing performance accurately, we consider the runtime of each stages. Results show that for all datasets, CUDAClustal gives a great computational performance: the runtime for stage one of CUDAClustal is smaller approximately 30 times than the sequential Clustal. This result is shown in Table 2. So if only considering parts which have been paralleled then the effect of using GPU is very feasible. This means that if stage three of the algorithm is also paralleled, the total time of the program will be more significantly reduced.

**Table 2.** Comparison of runtime by stages between CUDAClustal and Clustal

| Datasets | Stage 1 | | | Stage 2 & 3 | |
|---|---|---|---|---|---|
| | CUDAClustal | | Clustal | CUDAClustal | Clustal |
| | Memory Operation | Runtime of kernel | | | |
| test1.fasta | 0,26s | 4,40s | 124,13s | 115,46s | 96,12s |
| test2.fasta | 0,35s | 12,16s | 304,28s | 257,37s | 287,88s |
| test3.fasta | 0,04s | 8,03s | 237,00s | 102,48s | 90,51s |
| test4.fasta | 0,001s | 6,51s | 233,05s | 115,14s | 99,15s |

## 5 Conclusion

The technology GPU shows the ability of improving computational performance for problems which can be paralleled. In this paper, we present our parallel algorithm – CUDAClustal to solve the MSA problem. We have paralleled the first stage of the algorithm and achieved a significant speedup when compared to the sequential program. Here, parallelization for stage one brings increase in performance which is approximately two times for the entire algorithm Clustal and 30 times for stage one. This proves a certain success level of paralleling the algorithm using CUDA on GPU, enabling to parallel the whole three steps. In the future work, we intend to modify our

program by parallelizing stage three by performing alignment at nodes with the same height from leaf nodes in parallel. Moreover, we will implement the algorithm on multi GPU and GPU Cluster to increase the computing performance.

## References


1. http://docencia.ac.upc.edu/master/AMPP/slides/ampp_sw_presentation.pdf
2. http://en.wikipedia.org/wiki/Multiple_sequence_alignment
3. http://en.wikipedia.org/wiki/Needleman-Wunsch_algorithm
4. http://gpgpu.org
5. http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Dynamic/Hirsch
6. Cheetham, J., et al.: Parallel ClustalW for PC clusters. In: International Conference on Computational Science and Its Applications (ICCSA), pp. 300–309 (2003)
7. Thompson, J.D., et al.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res. 22(22), 4673–4680 (1994)
8. Li, K.B.: Clustal-MPI: ClustalW analysis using parallel and distributed computing. Bioinformatics 19, 1585–1586 (2003)
9. Chaichoompu, K., Kittitornkun, et al.: MT ClustalW: multithreading multiple sequence alignment. In: International Parallel and Distributed Processing Symposium (2006)
10. Notredame, Higgins, Heringa: T-Coffee: A novel method for multiple sequence alignments. JMB 302, 205–217
11. NVIDIA, http://www.nvidia.com/object/cuda_home_new.htm
12. Duzlevski, O.: SMP version of ClustalW 1.82 (unpublished)
13. Sugawara, H., Chenna, R., et al.: Multiple sequence alignment with the Clustal series of programs. Nucleic Acids Res. 31, 3497–3500 (2003)
14. Feng, S., Tan, G., et al.: Parallel multiple sequences alignment in SMP cluster. In: International Conference on HPC in Asia Region, pp. 425–431 (2005)
15. Haque, W., Aravind, A., et al.: Pairwise sequence alignment algorithms: a survey. In: Conference on Information Science, Technology and Application, New York (2009)
16. Liu, W., et al.: Streaming algorithms for biological sequence alignment on GPUs. IEEE Transactions on Parallel and Distributed Systems 18, 1270–1281 (2007)
17. Liu, Y., Schmidt, B., Maskell, D.L.: MSA-CUDA: Multiple Sequence Alignment on Graphics Processing Units with CUDA. In: International Conference on Application-specific Systems, Architectures and Processors, USA (2009)