# CUDA-Parttree: A Multiple Sequence Alignment Parallel Strategy in GPU

**Caina Razzolini**[1], **Alba Cristina Magalhaes Alves de Melo**[1]

[1]Department of Computer Science, University of Brasília (UnB)

cainarazzolini@aluno.unb.br, alves@unb.br

***Abstract.*** *In this paper, we propose and evaluate CUDA-Parttree, a parallel strategy that executes the first phase of the MAFFT Parttree Multiple Sequence Alignment tool (distance matrix calculation with 6mers) on GPU. When compared to Parttree, CUDA-Parttree obtained a speedup of* $6.10x$ *on the distance matrix calculation for the Cyclodex_gly_tran (*$50,280$ *sequences) set, reducing the execution time from* $33.94s$ *to* $5.57s$*. Including data conversion and movement to/from the GPU, the speedup was* $2.59x$*. With the sequence set Syn_100000 (*$100,000$ *sequences), a speedup of* $4.46x$ *was attained, reducing execution time from* $209.54s$ *to* $47.00s$*.*

## 1. Introduction

Bioinformatics is an interdisciplinary field that seeks to create algorithms and tools to help biologists analyze data, in order to understand the function and structure of biological sequences and the evolution of organisms [Mount 2013]. Sequence alignment is a basic Bioinformatics operation used to identify the similarity of sequences and can be done between two (pairwise alignment) or more (Multiple Sequences Alignment - MSA). MSA is a computationally challenging problem, proved to be NP-Complete [Wang and Jiang 1994]. Because of that, it is common to use heuristic algorithms.

MSA algorithms receive a set of sequences and return a score and a multiple alignment, which identifies regions of similarity and difference between the sequences. A heuristic MSA algorithm usually can be divided in three phases: (a) similarity matrix (distance matrix) calculation; (b) construction of a guide tree and (c) progressive alignment.

The evolution of sequencing methods have been generating increasingly larger genomic databases. In this scenario, heuristic MSA algorithms have difficulty aligning sets with tens of thousands of sequences in a timely manner. Even algorithms specifically designed for tens of thousands of sequences may take hours to obtain results [Deorowicz et al. 2016].

In order to shorten execution times, many Bioinformatics applications have been using the great parallel capacity of GPUs (Graphics Processing Units). There are MSA tools using GPU, but most are focused on sequence smaller than 20,000 sequences.

MAFFT [Katoh et al. 2002] is an MSA package that uses FFT (Fast Fourier Transform) in the progressive alignment phase. One of the algorithms offered by MAFFT is Parttree, which uses, by default, shared 6mer (subsequences of 6 characters) counting

to calculate the distance between sequences, and is capable of aligning over 100,000 sequences. The sequential version of Parttree has been used in many different works like [Mi et al. 2012], [Nam-phuong et al. 2015] and [Lamnidis et al. 2018], but its execution may take hours. At this moment, there is no version of Parttree for GPU.

In this paper, we propose and evaluate CUDA-Parttree, a parallel strategy using GPU to calculate the MSA of tens of thousands of sequences using Parttree. This strategy uses the GPU to calculate the number of shared 6mers betweens sequences, because not only is it a computationally demanding operation, but also because 6mer counting (or more generically k-mer counting) is also used on other Bioinformatics problems, like DNA assembly [Ghosh et al. 2019].

Using this strategy, we were able to reduce the execution time of the distance calculation between all sequences compared to Parttree. CUDA-Parttree was used to align 6 real sequence sets, ranging from $25,534$ to $151,443$ sequences, and 4 synthetic sets, ranging from 10,000 to 100,000 sequences. CUDA-Parttree obtained a speedup of $6.10x$ on the distance matrix calculation for the *Cyclodex_gly_tran* ($50,280$ sequences) set, reducing the execution time from $33.94s$ to $5.57s$. Including the data transformation required for the GPU calculation and to return the distance matrix, CUDA-Parttree obtained a speedup of $2.59x$. With the sequence set *Syn_100000* ($100,000$ sequences), a speedup of 4.46x was attained, reducing execution time from $209.54s$ to $47.00s$.

The rest of this paper is organized as follows. We present an overview of the MSA problem in Section 2. Section 3 discusses related work in the area of MSA using High Performance platforms. In Section 4, we present the design of CUDA-Parttree and the experimental results are shown in Section 5. Finally, we conclude the paper in Section 6.

## 2. Multiple Sequence Alignment (MSA)

Biological sequences are ordered set of amino acids, for proteins, or nucleotides, for DNA/RNA. DNAs e RNAs are represented by sequence of characters from alphabet $\{A, T, C, G\}$ and $\{A, U, C, G\}$, respectively, and protein by sequences of characters from alphabet $\{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ .

A Multiple Sequence Alignment is obtained by inserting gaps (spaces) into the $n$ sequences until all sequences have the same length ($l$). A scoring function is used in order to qualify an alignment. The most used scoring function is the Sum-of-Pairs, which defines the score (SP) of a MSA as the sum of the scores of the pairwise alignment of all possible pairs. The score of each pairwise alignment can be calculated in many ways. DNA/RNA alignments usually set a value for a match (equal characters) and another for a mismatch (different characters), and the total score of a pairwise alignment is given by the sum of the scores of each column. Protein alignments, on the other hands, use substitution matrices, like PAM250 [Dayhoff et al. 1978], in order to score each column.

The Weighted Sum-of-Pairs (WSP) is an adaptation of the SP scoring function that uses different weights for each pairwise alignment, chosen based on evolutionary distances between sequences.

There is an exact algorithm for obtaining the optimal alignment between a set of $N$ sequences [Mount 2013]. The algorithm has one loop for each sequence that covers the $L$ character of the sequences. A multi-dimensional dynamic programming matrix is

computed. The complexity of the algorithm is $O(L^N)$, which makes it unfeasible for even small sequence sets (30 sequences).

The MSA of a large number of sequences is of great interest to biologists since it allow the identification of subtle and disperse similarities. However, the MSA using SP was proved to NP-complete [Wang and Jiang 1994], as such, many heuristic methods were developed in order to obtain the MSA in a timely manner. Most methods can be classified in two categories: progressive alignment or iterative refinement.

Progressive alignment methods, such as Clustal W [Thompson et al. 1994], consists of 3 steps: (*a*) Distance matrix calculation; (*b*) Guide tree construction; and (*c*) Progressive alignment construction.

On the fist step (distance matrix calculation), the distance matrix is built using the scores of the pairwise alignment between all sequences or between a selected subset of the sequences and the remaining sequences, like the center-star strategy, where a single sequence is aligned to all others. There are many algorithms used for the pairwise alignment like [Smith et al. 1981], [Hirschberg 1977] or the number of shared kmers (subsequences of length $k$). The progressive alignment construction (step 3) is built following the order of the guide tree. It can be done adding the sequences to a single growing alignment or building temporary alignments that are eventually aligned themselves.

Iterative refinement algorithms seek to reduce the error introduced in the progressive alignment. They consist of 4 steps, the first 3 being the progressive alignment steps (*a*,*b* and *c*) and the fourth being (*d*) the iterative refinement step. The iterative refinement uses an initial alignment, obtained using a progressive method, and iteratively remove sequences from the alignment and realigns them, for as long as there is a gain in the overall accuracy higher than a defined value. This often results in more accurate alignments, but also in longer execution times.

MAFFT Parttree [Katoh and Toh 2006] is a progressive alignment algorithm for tens of thousands of sequences on the MAFFT package. It is a divisive clustering algorithm that builds a guide tree from unaligned sequences in $O(N \log N)$. It uses a recursive approach where on each level of recursion a piece of the guide tree is built from a subset of sequences and the superior level of recursion combines the trees. In order to build the tree, the distance between sequences is calculated using the number of shared 6mers (subsequences on 6 characters).

Initially all sequences are aligned to the longest sequence. Next, a subset of $n$ reference sequences is chosen, containing the longest sequence, the most distant sequence from the longest and $n - 2$ random sequences. The reference sequences are then aligned amongst themselves and a tree is built between them. Following, the $N - n$ other sequences (base sequences) are then aligned to the reference sequences and grouped to the closest one. A recursive call to Parttree in then made for each group and the resulting trees stores. Finally, the resulting trees are combined with the reference tree and the result returned.

MAFFT Parttree was able to align more than 60,000 sequences with an average loss of 3% accuracy compared to the golden standard of PFAM [Finn et al. 2006].

**Table 1. State of the art MSA algorithm in HPC platforms.**

| Reference | Year | Distance algorithm | Maximum Number of sequences | Platform |
|---|---|---|---|---|
| CUDA Clustal W | 2015 | LCS | 1.000 | GPU |
| CUDA-Linsi | 2015 | Smith-Waterman | 200 | GPU |
| CSMA | 2017 | Center-star/ K-band | 500.000 | GPU |
| MSAProsbs-MPI | 2016 | HMM | 1.543 | MPI/OpenMP |
| FAMSA | 2016 | LCS | 415.519 | AVX ou GPU |

## 3. Related work

There are several works on the literature that uses HPC platforms for MSA. We have analyzed 5 recent works: CUDA Clustal W [Hung et al. 2015], MAFFT CUDA-Linsi [Zhu et al. 2015], CMSA [Chen et al. 2017], MSAProbs-MPI [González-Domínguez et al. 2016] and FAMSA [Deorowicz et al. 2016], that were published between 2015 and 2017. Four of them, CUDA Clustal W, MAFFT-Linsi, MSAProbs-MPI and FAMSA, compare all possible sequence pairs with quadratic dynamic programming algorithms (LCS,Smith-Waterman,HMM-Viterbi). CMSA compares only one sequence to all sequences, reducing considerably the number of comparisons, with a negative impact on the quality of the alignments. Only CMSA and FAMSA are able to compare more than $100,000$ sequences either in GPU or CPU with SIMD instructions (AVX). The best performance is given by FAMSA [Deorowicz et al. 2016] with the AVX implementation.

## 4. Design of CUDA-Parttree

### 4.1. Parttree analysis

CUDA-Parttree is proposed in this paper and it is a version of Parttree that executes part of the distance calculation using shared 6mers in GPU. The distance calculation step is of interest in regards to parallelization because it requires a large quantity of independent calculations. Each pairwise distance can be calculated independently from one another, which makes it very compatible with GPU processing.

The calculation of the distance matrix usually requires $\frac{N(N-1)}{2}$ comparisons. For sequence sets with tens of thousands of sequences, this means an order of billion of comparisons. Parttree only aligns a sequences to a subset of up to $n$ sequences (reference sequences), which reduces the number of comparisons to $\frac{n(N-n)}{2}$. Usually, $n << N$, as such Parttree is able to calculate the alignment much faster than most progressive alignment algorithms. Besides, Parttree uses shared 6mer counting to define distances between sequences, which is much faster than alignments using dynamic programming.

Parttree uses a recursive approach to build the guide tree, and in each level of recursion, three distance calculations are executed: (a) between all sequences and the largest $(1 \times N)$(Area 1 in Figure 1); (b) between the reference sequences $(n \times n)$(Area 2); and (c) among the reference sequences and the remaining ones (base sequences) $(n \times (N - n))$(Area 3). Since, for a huge set of sequences, Area 3 is much bigger than the other areas, we decided to use GPU to accelerate the calculation of shared 6mers only on the first level of recursion and only for Area 3.

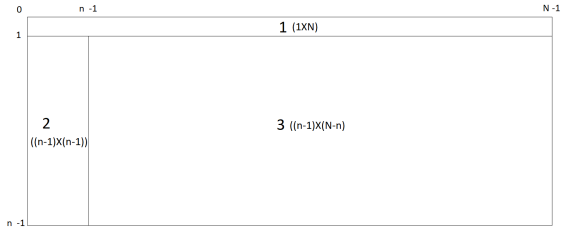



**Figure 1. One recursive level of the Distance matrix calculated by Parttree**

The distance calculation algorithm (*calculateSimilarity*) groups each amino acid in one of 6 groups, based on the values of PAM250 [Dayhoff et al. 1978], and represent them as a number between $0$ and $5$. Next, each 6mer (subsequence of 6 characters) is transformed into an integer number using $kmer = \sum_{i=0}^{5} 6^i * s[i]$, having a value between $0$ and $6^6 - 1$. This way, each sequence of amino acids is converted into a sequence of 6mers.

Algorihtm 1 presents the pseudo code for the *calculateSimilarity* function. Firstly, a vector called *refMap*, containing $6^6$ positions, where element $i$ represents the number of times that 6mers $i$ appeared in the sequence, is created (lines 2 to 8). Next, the *calculate6mers* function is called and for each 6mer $k$ on the base sequence it is checked if the value position $k$ on the *mirrorMap* is less than the value of position $k$ on the *refMap*. If positive, the number of shared 6mers and the position $k$ on the *mirrotMap* are incremented by 1. Otherwise, it means that 6mer $k$ appeared more times in the base sequence and no value is changed.

**Algorithm 1** calculateSimilarity(refSeq,baseSeq)

```
1:  function CALCULATESIMILARITY(refSeq, baseSeq)
2:      distMtx[n * (N − n)] = [0, 0...0]
3:      for r ∈ refSeq do
4:          refMap[6^6] = [0, 0...0]
5:          for i = 0 → length(r) do
6:              k ← r[i]
7:              refMap[k] + +
8:          end for
9:          calculate6mers(refMap, baseSeqs, dstMtx)
10:     end for
11:     return distMtx
12: end function
13: function CALCULATE6MERS(refMap, baseSeq, dstMtx)
14:     for s ∈ baseSeqs do
15:         mirrorMap[6^6] = [0, 0...0]
16:         for i = 0 → length(s) do
17:             k ← s[i]
18:             if refMap[k] > mirrorMap[k] then
19:                 distMtx[r][s] + +
20:                 mirrorMap[k] + +
21:             end if
22:         end for
23:     end for
24: end function
```

## 4.2. CUDA-Parttree

The objective of CUDA-Parttree is to speedup the calculation of the distance matrix of the last call to *calculateSimilarity* on the first level of recursion (Figure 1 Area 3) when compared to Parttree.

The calculation of the distance matrix in CUDA-Parttree was divided in 3 phases: (a) Data conversion (Input); (b) GPU computation and (c) Data retrieval (Output). Figure 2 illustrates our design.

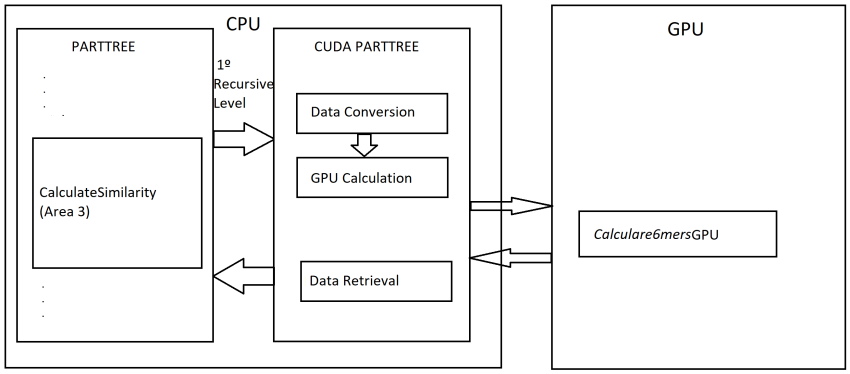



**Figure 2. CUDA-Parttree organization**

The Data Conversion (Input) phase receive the base sequences, makes the necessary transformations for execution on GPU and then sends them to GPU. The GPU Calculation phase generates the *refMaps*, sends them to GPU, execute the GPU shared 6mer calculation and moves the distance matrix from GPU to CPU. Finally, the Data Retrieval phase applies a length based correction to the value of shared 6mers calculated and returns the distance matrix back to the Parttree algorithm.

In order to achieve good performance on GPU, some transformations were made to the data. First, for each pair of reference sequence and base sequence, Parttree used a mirror vector in order to avoid over counting 6mer. Since the original code is sequential, it was possible for Parttree to always use the same vector, only requiring the values to be reset for each new pair. In CUDA-Partree however, the calculations are done in parallel, requiring multiple vectors, each with $6^6$ integers, which means about $182KB$ per pair.

In order to reduce the amount of memory necessary without affecting the parallelism, a new way of representing the sequences is proposed in CUDA-Parttree. In Parttree, sequences are represented as a vector of integers representing 6mers, on the order in which they appear. The new way of representing the sequences uses a vector of structs, (2 integers): 6mer and the number of times it appears in the sequence (Figure 3). In this new representation, the mirror vector is no longer necessary and the 6mer counting is still done in $O(L)$, where $L$ is the length of the sequence.

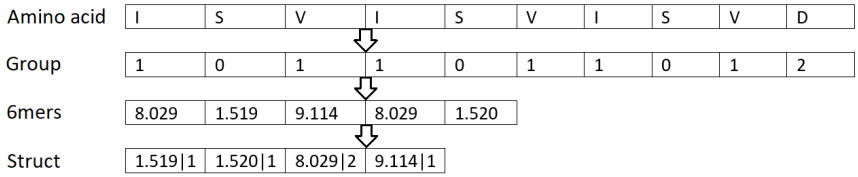



**Figure 3. CUDA-Parttree data conversion**

Once the sequences are converted, they are sorted according to length, setting the $positionMap$ vector with the original positions of the sequences. Next, paddings are added to the sequences so that each group of 32 sequences (section) has the same length,

which will help reduce the time threads remain idle on GPU. Finally, the sequences are transfered to a linearized vector and copied to GPU. During this phase, two additional vectors are created to keep the length of the sequences of each section and the initial positions of each section in the linearized vector.

The GPU calculation phase consists of the calculation of the *refMaps*, sending them to GPU, executing the calculation of the distance matrix (6mer counting) in GPU, retrieving the resulting matrix to CPU and freeing the GPU memory.

Algorithm 2 presents the algorithm for counting shared 6mers in GPU. Each GPU thread is responsible for calculating the number of shared 6mers between one base sequence and all reference sequences. The padding added ensures that all threads in a warp execute over vector of the same length, improving the parallelism.

Algorithm 2 starts by calculating the id of the base sequence that the thread will be responsible (line 1) and checking that it is a valid id (line 2). Next, the id of the section to which the sequence belongs is calculated (line 3), followed by its position within the section (line 4). In line 5 the initial position of the section is obtained, followe by the length of the sequences on the section (line 6) and then the position of the sequence on the vector (line 7). The loop in line 8 is responsible for iterating over the reference sequences and in line 9 the position of the refMap is calculated. The loop on line 11 iterates over the vector of 6mer. In line 12, the number of times that the 6mer appeared on the base sequence is recovered, followed by he number of times it appeared on reference sequence (line 13). Next the number shared 6mers is updated with the lowest value (lines 14 to 18). Finally, in lines 20 and 21 the total value obtained is moved to distance matrix.

**Algorithm 2** calculate6mersGPU(sectionLengthVec,sectionAddrVec,N,n,refMap, baseSeqs,distMtx)

```
1:  seqId = blockIdx.x * blockDim.x + threadIdx.x
2:  if seqId < N then
3:      sectionId = ⌊seqId/WARPSIZE⌋
4:      offset = sectionId%WARPSIZE
5:      sectionAddr = sectionAddrVec[seqId]
6:      sectionLength = sectionLengthVec[seqId]
7:      sequenceAddr = sectionAddr + offset * sectionLength
8:      for refSeqId = 0 → n do
9:          mapAddress = refMap + (refSeqId * MAPSIZE)
10:         distance = 0
11:         for j = 0 → sectionLength do
12:             baseCount = baseSeq[sequenceAddr + j].count
13:             refCount = mapAddress[baseSeq[sequenceAddr + j].kmer]
14:             if baseCount <= refCount then
15:                 distance += baseCount
16:             else
17:                 distance += refCount
18:             end if
19:         end for
20:         index = N * refSeqId + seqId
21:         distMtx[index] = distance
22:     end for
23: end if
```

This phase ends after the resulting distance matrix is recovered from GPU and the GPU memory is freed.

The Data Retrieval phase is responsible for moving the values calculated in GPU to the correct positions, according to the original sequence order before the sorting on

phase 1 and for applying a length based correction to the distances calculated.

## 5. Experimental Results

### 5.1. Experimental Setup

CUDA-Parttree was written in CUDA C and tested in the test environment described in Table 2. It was executed using GPU blocks with $256$ threads each ($T = 256$), and as many blocks as necessary to cover all sequences ($B = N/T$). These values were chosen empirically. The maximum number of reference sequences used for these experiments was $n = 5.000$, the highest value our execution environment was able to complete. The section size was set to $32$, the same as the GPU's warp size. All other parameters used the Parttree default values.

**Table 2. Test Environment.**

| CPU | GPU | Software |
|---|---|---|
| Intel Core i7-3770 CPU, 3.40GHz<br>4 Cores<br>128KB L1 Cache<br>8GB RAM<br>1 TB Disco | GeForce GTX 980 Ti<br>2816 CUDA cores<br>1.19 GHz<br>6GB RAM | CentOS Linux 7<br>nvcc release 8.0<br>gcc version 4.8.5<br>CUDA Toolkit 8.0 |

We used 6 real sequence sets, obtained from the extHomFam benchmark [Gudys 2016] and 4 synthetic sequence sets generated randomly. Tables 3 present the sequence sets used.

**Table 3. Description of the sequence sets.**

| ID | Number of sequences | Length shortest sequence | Length longest sequence | Average length | Dataset size(MB) |
|---|---|---|---|---|---|
| bac_luciferase | 25534 | 30 | 405 | 294 | 8.4 |
| Peptidase_M24 | 25574 | 23 | 354 | 218 | 6.4 |
| fn3 | 49584 | 21 | 157 | 84 | 5.7 |
| Cyclodex_gly_tran | 50280 | 18 | 686 | 190 | 11.2 |
| mdd | 104692 | 24 | 387 | 120 | 15.7 |
| HATPase_c | 151443 | 31 | 243 | 115 | 21.9 |
| syn_10000 | 10000 | 130 | 149 | 140 | 1.5 |
| syn_25000 | 25000 | 130 | 149 | 140 | 3.8 |
| syn_50000 | 50000 | 130 | 149 | 140 | 7.6 |
| syn_100000 | 100000 | 130 | 149 | 140 | 15.2 |

### 5.2. Number of reference sequences

Because Parttree choses a subset of $n - 2$ random sequences as reference sequences and then filters them so that if sequences too similar only the longest is kept we measured how many reference sequences remained after the filtration step of Parttrre. Table 4 presents for each sequence set, the total number of sequences, the number of reference sequences after the filtration step on the first recursion level, the number cells on the calculated distance matrix and the size it occupies.

These same measurements were made for the synthetic sequence sets. However, because the sequences in these sets were generate randomly, in all sequence sets the filtration step did not find any sequences that were too similar to one another, as such the number of reference sequences used on the alignment was $5,000$. This means that for the largest sequence set, *Syn_100000*, the distance matrix calculated had 500 million cells ($5,000 \times 100,000$).

**Table 4. Number of reference sequences after filtration (the real sequence sets).**

| ID | Number of sequences | # reference sequences | # distMtx cells | distMtx size(MB) |
|---|---|---|---|---|
| bac_luciferase | 25.534 | 906 | 23.133.804 | 88.25 |
| Peptidase_M24 | 25.574 | 835 | 21.354.290 | 81.46 |
| fn3 | 49.584 | 2.278 | 112.952.352 | 430.88 |
| Cyclodex_gly_tran | 50.280 | 1.291 | 64.911.480 | 247.62 |
| mdd | 104.692 | 1.446 | 151.384.632 | 577.59 |
| HATPase_c | 151.443 | 2.281 | 345.441.483 | 1317.75 |

## 5.3. CUDA-Parttree execution time

In order to evaluate the speedup obtained with CUDA-Parttree, we measured the execution times of the distance matrix calculation in both CUDA-Parttree and Parttree. In CUDA-Parttree, this counts all three phases (data conversion, GPU calculation and data retrieval). Tables 5 and 6 present the results of our measurements. The execution times are those for the whole CUDA-Parttree, including Data conversion, GPU comparison and Data Retrieval.

**Table 5. Execution time for Parttree and CUDA-Parttree (real sequences).**

| ID | Parttree(s) | CUDA Parttree(s) | Speedup |
|---|---|---|---|
| bac_luciferase | 15.47 | 6.48 | 2.39x |
| Peptidase_M24 | 10.29 | 5.64 | 1.83x |
| fn3 | 27.17 | 13.47 | 2.02x |
| Cyclodex_gly_tran | 33.94 | 13.18 | 2.58x |
| mdd | 54.02 | 24.96 | 2.16x |
| HATPase_c | 121.56 | 68.65 | 1.77x |

**Table 6. Execution time for Parttree and CUDA-Parttree (synthetic sequences).**

| ID | Parttree(s) | CUDA Parttree(s) | Speedup |
|---|---|---|---|
| Syn_10000 | 9.50 | 4.73 | 2.01x |
| Syn_25000 | 41.22 | 14.86 | 2.77x |
| Syn_50000 | 98.96 | 34.54 | 2.87x |
| Syn_100000 | 209.54 | 70.40 | 2.98x |

CUDA-Parttree was able to achieve a maximum speedup of $2.58x$ on the real sequence set *Cyclodex_gly_tran*, reducing the execution from $33.94s$ to $13.18s$. On the synthetic sequences, CUDA-Parttree obtained a speedup of $2,98x$ on set *Syn_100000*, reducing execution time from $209.54s$ to $70.40s$.

## 5.4. Profiling of CUDA-Parttree

The speedups obtained, despite being good, were below our expectations. Because of that, we made a profiling of the execution times of each step of CUDA-Parttree: (a) sequence_setup; (b) creation and movement of reference maps (Calc/Send refMaps); (c) GPU calculation; (d) movement of the distance matrix from GPU to CPU; and (e) return to Parttree. Step (a) is equivalent to the Data conversion (Input) phase of Parttree, steps (b), (c) and (d) belong to the GPU calculation phase (Compute) and step (e) is equivalent to the Data Retrieval phase (Output).

Tables 7 and 8 present the execution times of each step of CUDA-Parttree. The first column presents the ID of the sequence sets and the next five present the execution

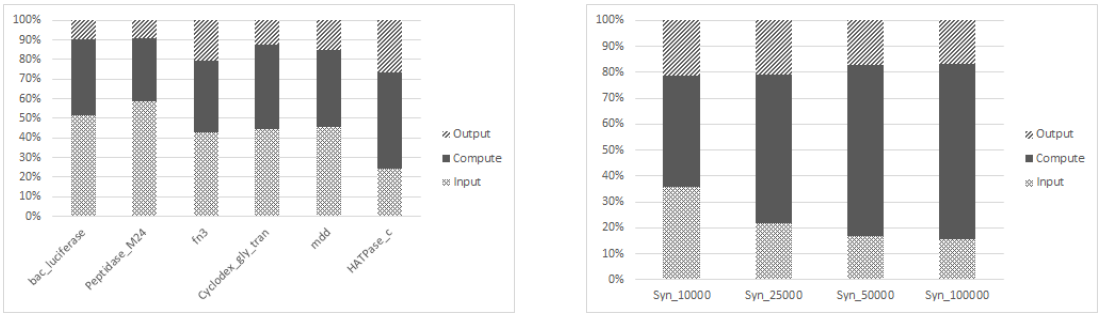
**Table 7. Profiling of CUDA-Parttree using real sequence sets.**

| ID | Sequence Setup(s) | Calc & send(s) RefMaps(s) | GPU calc(s) | distMtx Retrieve(s) | Parttree return(s) |
|---|---|---|---|---|---|
| | Input | Compute | | | Output |
| bac_luciferase | 3.342 | 0.029 | 2.466 | 0.022 | 0.619 |
| Peptidase_M24 | 3.319 | 0.024 | 1.769 | 0.020 | 0.506 |
| fn3 | 5.792 | 0.086 | 4.670 | 0.177 | 2.739 |
| Cyclodex_gly_tran | 5.890 | 0.037 | 5.529 | 0.074 | 1.644 |
| mdd | 11.417 | 0.037 | 9.580 | 0.210 | 3.719 |
| HATPase_c | 16.531 | 0.120 | 22.411 | 10.936 | 18.230 |

**Table 8. Profiling of CUDA-Parttree using synthetic sequence sets.**

| ID | Sequence Setup(s) | Calc & send(s) RefMaps(s) | GPU calc(s) | distMtx Retrieve(s) | Parttree return(s) |
|---|---|---|---|---|---|
| | Input | Compute | | | Output |
| Syn_10000 | 1.68 | 0.13 | 1.85 | 0.05 | 1.01 |
| Syn_25000 | 3.23 | 0.13 | 8.30 | 0.12 | 3.09 |
| Syn_50000 | 5.74 | 0.13 | 22.56 | 0.23 | 5.87 |
| Syn_100000 | 10.90 | 0.12 | 46.88 | 0.69 | 11.80 |

times of each step in seconds. The profiling allowed us to identify that a significant amount of time is spent on the Input and Output phases (Figure 4). As the size of the sets increase, so does the proportion of time spent in the Compute phase, reaching $42.75\%$ on the largest set. The synthetic sequence sets spent proportionally more time on the Compute phase than the real sequence sets since more reference sequences remained after the filtration phase for synthetic sequences.




Real sequence sets Synthetic sequence sets

**Figure 4. Distribution of time on each phase of CUDA-Parttree**

## 5.5. Comparison with FAMSA

FAMSA [Deorowicz et al. 2016] was the multiple sequence alignment algorithm for tens of thousands of sequences that presented the best results during our review of the state of the art, providing better accuracy and shorter times than the other algorithms, including Parttree. In order to evaluate the performance of CUDA-Parttree, we compared the total execution times of FAMSA and CUDA-Parttree using the same parameters used by [Deorowicz et al. 2016] on the 6 real sequence sets (Table 9). FAMSA obtained the best execution times on the smaller sequence sets. However, for the larger sets CUDA-Parttree had better execution times, achieving a speedup of $1.34x$ for sequence set *HATPase_C*.

**Table 9. Total execution times of FAMSA and CUDA-Parttree for the real sequence sets.**

| ID | FAMSA(s) | CUDA Parttree(s) | Speedup |
|---|---|---|---|
| bac_luciferase | 55.96 | 70.67 | 0.79x |
| Peptidase_M24 | 56.67 | 60,33 | 0.94x |
| fn3 | 91.35 | 118.33 | 0.77x |
| Cyclodex_gly_tran | 182,80 | 428,33 | 0.43x |
| mdd | 330.20 | 326.67 | 1.01x |
| HATPase_c | 707.78 | 528.00 | 1.34x |

## 6. Conclusion

In this paper, we proposed and evaluated CUDA-Parttree, parallel strategy for heuristic multiple sequence alignment of tens of thousands of sequences in GPU. CUDA-Parttree executes the counting of shared 6mers between sequences in GPU with sequences sorted by length, a new data structure for representing the sequences and paddings.

The results obtained from the experiments showed that CUDA-Parttree was able to calculate the distances between sequences much faster than Parttree. Considering Data Conversion and movement to/from the GPU, very good speedups were still obtained. Using a GTX980 Ti GPU we obtained a maximum speedup of $2.58x$, reducing the execution time from $33.94s$ to $13.18s$. We also observed that, despite setting the maximum number of reference sequences to $5000$, the real sequence sets used a lot less sequences than that, which also impacted the speedup. In the experiments with synthetic sets, we achieved a maximum speedup of $2.98x$ ($100,000$ synthetic sequences) on the distance matrix calculation, reducing the execution time from $209.54s$ to $47.00s$. The comparison with FAMSA showed we achieved a speedup of up to $1.34x$ on our test environment.

As future works, we intend to test CUDA-Parttree on more modern GPUs, specially of the Volta architecture. Additionally, we intend to implement techniques of asynchronous data movement between CPU and GPU and use CPU threads in order to increase the speedup of CUDA-Parttree. Finally, we intend to adapt our shared 6mer counting algorithm for use in other Bioinformatics applications.

### Acknowledgment

We would like to thank Prof. Kazutaka Katoh for his support with the Parttree tool. This work is partially supported by Capes/PROCAD n. 183794.

## References

Chen, X. et al. (2017). CMSA: a heterogeneous CPU/GPU computing system for multiple similar RNA/DNA sequence alignment. *BMC bioinformatics*, 18(1):315.

Dayhoff, M., Schwartz, R., and Orcutt, B. (1978). 22 A Model of Evolutionary Change in Proteins. In *Atlas of protein sequence and structure*, volume 5, pages 345–352. National Biomedical Research Foundation Silver Spring, MD.

Deorowicz, S., Debudaj-Grabysz, A., and Gudyś, A. (2016). FAMSA: Fast and accurate multiple sequence alignment of huge protein families. *Scientific reports*, 6.

Finn, R. D. et al. (2006). Pfam: clans, web tools and services. *Nucleic acids research*, 34(suppl_1):D247–D251.

Ghosh, P., Krishnamoorthy, S., and Kalyanaraman, A. (2019). Pakman: Scalable assembly of large genomes on distributed memory machines. *bioRxiv*.

González-Domínguez, J., Liu, Y., Touriño, J., and Schmidt, B. (2016). MSAProbs-MPI: parallel multiple sequence aligner for distributed-memory systems. *Bioinformatics*, 32(24):3826–3828.

Gudys, A. (2016). extHomFam benchmark.

Hirschberg, D. S. (1977). Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)*, 24(4):664–675.

Hung, C.-L. et al. (2015). CUDA ClustalW: An efficient parallel algorithm for progressive multiple sequence alignment on Multi-GPUs. *Computational biology and chemistry*, 58:62–68.

Katoh, K. et al. (2002). MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic acids research*, 30(14):3059–3066.

Katoh, K. and Toh, H. (2006). PartTree: an algorithm to build an approximate tree from a large number of unaligned sequences. *Bioinformatics*, 23(3):372–374.

Lamnidis, T. C. et al. (2018). Ancient fennoscandian genomes reveal origin and spread of siberian ancestry in europe. *Nature communications*, 9(1):5018.

Mi, H., Muruganujan, A., and Thomas, P. D. (2012). Panther in 2013: modeling the evolution of gene function, and other gene attributes, in the context of phylogenetic trees. *Nucleic acids research*, 41(D1):D377–D386.

Mount, D. (2013). *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, 2nd edition.

Nam-phuong, D. N. et al. (2015). Ultra-large alignments using phylogeny-aware profiles. *Genome biology*, 16(1):124.

Smith, T. F., Waterman, M. S., et al. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197.

Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic acids research*, 22(22):4673–4680.

Wang, L. and Jiang, T. (1994). On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348.

Zhu, X. et al. (2015). Parallel implementation of MAFFT on CUDA-enabled graphics hardware. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 12(1):205–218.