MSA-GPU: Exact Multiple Sequence Alignment Using GPU

Daniel Sundfeld and Alba C.M.A. de Melo

Department of Computer Science, University of Brasília, Brasília, Brazil
{sund,alves}@unb.br

Abstract. In this paper, we propose and evaluate MSA-GPU, a solution to implement the exact Multiple Sequence Alignment algorithm in Graphics Processing Units (GPUs). In our solution, we use the Carrillo-Lipman upper and lower bounds to reduce the amount of computation. We propose a fine-grained strategy to explore the search space by using 2D projections. The results were obtained with a GTX 580 NVidia GPU comparing sets of 3 sequences (real and synthetic). We show that, for sequences with medium/low similarity, our GPU approach is able to outperform the MSA 2.0 CPU program, achieving a speedup of 8.6x.

1 Introduction

Bioinformatics is an interdisciplinary field that involves computer science, biology, mathematics and statistics [12]. One of its main goals is to analyze biological sequence data and genome content in order to obtain the function/structure of the sequences as well as evolutionary information.

Once a new biological sequence is discovered, its functional/structural characteristics must be established. In order to do that, the newly discovered sequence is compared against the sequences that compose genomic databases, in search of similarities. Sequence comparison is, therefore, one of the most basic operations in Bioinformatics. A sequence can be compared to another sequence (Pairwise Comparison), to a profile that describes a family of sequences (Sequence-Profile Comparison) or to a set of sequences (Multiple Sequence Alignment).

In a Multiple Sequence Alignment (MSA), similar characters among a set of k sequences ($k > 2$) are aligned together. Multiple Sequence Alignments are often used as a building block to solve important and complex problems in Molecular Biology, such as the identification of conserved motifs in a family of proteins, definition of phylogenetic relationships and 3D homology modeling, among others. In all these cases, the quality of the solutions relies heavily on the quality of the underlying multiple alignment. MSAs are often scored with the Sum-of-Pairs (SP) objective function and the exact SP MSA problem is known to be NP-complete [18]. Therefore, heuristic methods are often used to solve this problem, even when the number of sequences is small.

A great number of heuristic methods were proposed to tackle the Multiple Sequence Alignment problem. In a general way, they fall into two categories:

progressive and iterative. A progressive MSA method initially generates all pairwise alignments and ranks them. The closest sequences are aligned first and then an MSA is built by adding the other sequences, in order of relevance. ClustalW [3] is an example of a progressive method. Iterative methods create an initial MSA of groups of sequences and then modify it, until a reasonable result is attained. DIALIGN [11] is an example of a deterministic iterative method. More recently, statistical methods that take into account evolutionary information such as Prank [7] and StatAlign [15] have also been proposed. This creates a great number of MSA heuristic tools, making it difficult to compare results and to determine the quality of a given MSA.

On the other hand, there do exist exact methods that are able to obtain the optimal Multiple Sequence Alignment [12]. The so-called naive exact method is a generalization of the exact algorithm based on dynamic programming that obtains optimal pairwise alignments [17]. It has time complexity $O(n^k)$, where n is the size of the sequences and k is the number of sequences.

Carrillo-Lipman [2] made an important contribution in the area of exact Multiple Sequence Alignment by showing that it is not necessary to explore the whole search space in order to obtain the optimal alignment. They showed that an heuristic alignment can be used to obtain an upper bound to the optimal alignment and all possible pairwise combinations can be used to obtain a lower bound. It is proven that the cells that fall outside these bounds do not contribute to the optimal alignment and, thus, these cells do not need to be calculated. Even with this, time complexity remains exponential.

Many efforts have been made to reduce the execution time of Multiple Sequence Alignment algorithms. Parallel versions were proposed to accelerate heuristic methods in clusters [5], [9], FPGAs (Field Programmable Gate Arrays) [16] and GPUs (Graphics Processing Units) [6], [1]. A few efforts were also made to implement exact Multiple Sequence Alignment algorithms in clusters [4] and FPGAs [8]. As far as we know, there are no implementations of the exact MSA in GPUs.

In this paper, we propose and evaluate MSA-GPU, a GPU solution to implement the exact Multiple Sequence Alignment algorithm. In our solution, we use the Carrillo-Lipman upper and lower bounds to reduce the amount of computation. We propose a fine-grained strategy to explore the search space by using 2D projections as in [8]. The results were obtained with a GTX 580 NVidia GPU comparing sets of 3 sequences (real and synthetic). We show that, for sequences with medium/low similarity, our GPU approach is able to outperform the MSA 2.0 CPU program, achieving a speedup of 8.6x.

The rest of this paper is organized as follows. We present an overview of the Multiple Sequence Alignment problem in Section 2. Section 3 discusses related work in the area of Exact MSA. In Section 4, the design of our GPU strategy for exact MSA is presented and the experimental results are shown in Section 5. Finally, we conclude the paper and give the future work directions in Section 6.

2 Multiple Sequence Alignment (MSA)

To compare two sequences, we search the best alignment between them, which amounts to place one sequence above the other making clear the correspondence between similar characters or substrings from the sequences [12].

A global Multiple Sequence Alignment (MSA) of $k > 2$ sequences $S = S_1, S_2, ..., S_k$ is obtained in such a way that spaces (gaps) are inserted into each of the n sequences so that the resulting sequences have the same length n. Then, the sequences are arranged in k rows of n columns each, so that each character or space of each sequence is in a unique column [12]. Figure 1 shows an example of one pairwise aligment and one MSA of 3 DNA sequences.

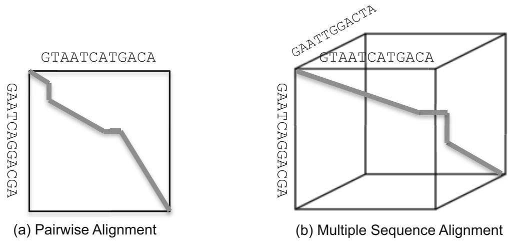


Fig. 1. Pairwise alignment and MSA with 3 sequences. The gray line represents one possible alignment.

Usually, MSAs are scored with the Sum-of-Pairs (SP) function and the exact SP MSA problem is known to be NP-hard [18]. In SP, every pair of bases is scored with the pairwise scoring function and the final score is the addition of all these values [12]. For instance, considering that the punctuation for matches (similar characters), mismatches (different characters) and gaps are 0, +1 and +1, respectively, the score generated by pairwise comparison of sequences S_1 and S_2 (Figure 2) is $0 + 1 + 1 + 0 + 0 + 0 + 0 + 1 + 1 + 0 + 0 + 0 + 1 + 0 = 5$. The SP score of the MSA in this example is 16. When comparing proteins, a substitution matrix is used to score matches/mismatches. The most common substitution matrices are PAM and BLOSUM [12].

2.1 Heuristic Methods

Many heuristic methods have been proposed in the literature to solve the MSA problem. ClustalW [3] is a progressive heuristic method that aligns k sequences in three phases. In the first phase, all $(k * (k - 1)/2)$ pairwise alignments are computed and scores are obtained. These scores are used to generate a distance matrix that indicates the similarity between the sequences. In the second phase, a guide tree is generated from the distance matrix, using the neighbor-joining method. The guide tree is used in the third phase to progressively generate the MSA, starting with the most closely related sequences.

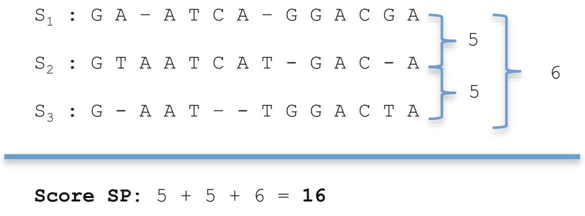

S1 : G A - A T C A - G G A C G A
S2 : G T A A T C A T - G A C - A
S3 : G - A A T - - T G G A C T A
5
5
6
Score SP: 5 + 5 + 6 = 16

Fig. 2. The Sum-of-Pairs scoring function

Like ClustalW, DIALIGN (DIagonal ALIGNment) [11] is an heuristic method for MSA. In order to align sequences, DIALIGN looks for ungapped fragments (or diagonals) and aligns them. Thus, in DIALIGN, an alignment is defined to be a chain of diagonals. The algorithm is executed in three phases. In the first phase, all DIALIGN pairwise alignments are computed, i.e., there are $k*(k-1)/2$ chains of diagonals, one for each pairwise alignment, where k is the number of sequences [10]. In the second phase, the diagonals that compose the pairwise alignments are sorted by their score and the degree of overlap with other diagonals. This sorted list is used to obtain an MSA with a greedy algorithm, generating alignment A. In the last phase, the alignment A is completed with an iterative procedure where the parts of the sequences that are not yet aligned with A are realigned by executing phase 2 again, in such a way that consistent non-aligned diagonals are included in A. This phase is repeated until no diagonal with a positive weight can be included in A.

In addition to ClustalW and DIALIGN, there are many other methods to calculate heuristic MSAs such as SAGA [14] and T-Coffee [13]. Even though the heuristic methods are able to provide good solutions, it is not guaranteed that the optimal MSA will be obtained.

2.2 Exact MSA

The optimal multiple sequence alignment among k sequences can be calculated by extending the exact dynamic programming algorithm for pairwise comparison [12]. Without loss of generality, assume that there are 3 sequences S_1, S_2 and S_3, of sizes n_1, n_2 and n_3, respectively.

The goal is to obtain the optimal score, which indicates the alignment distance (minimum number of insertions, deletions and substitutions). In order to do that, we calculate a 3-dimensional dynamic programming matrix D, where $D(i, j, k)$ is the optimal alignment of prefixes $S_1[1..i]$, $S_2[1..j]$ and $S_3[1..k]$.

The naive algorithm is depicted in Algorithm 1. There are three *for* loops (lines 1 to 3), one loop for each sequence. The scores for matches and mismatches are calculated in lines 4 to 6. After that (lines 7 to 13), seven values are calculated which correspond to the seven neighbor cells of $D(i, j, k)$. In line 14, $D(i, j, k)$ receives the minimum value of those calculated in lines 7 to 13. If there are k

Algorithm 1. `Naive exact Multiple Sequence Alignment`

```
1: for $i = 1 \rightarrow n_1$ do
2:   for $j = 1 \rightarrow n_2$ do
3:     for $k = 1 \rightarrow n_3$ do
4:       $c_{ij} = AssignMatchMismatchPunctuation(S_1(i), S_2(j));$
5:       $c_{ik} = AssignMatchMismatchPunctuation(S_1(i), S_3(k));$
6:       $c_{jk} = AssignMatchMismatchPunctuation(S_2(k), S_3(j));$
7:       $d_1 = D(i-1, j-1, k-1) + c_{ij} + c_{ik} + c_{jk}$
8:       $d_2 = D(i-1, j-1, k) + c_{ij} + gap$
9:       $d_3 = D(i-1, j, k-1) + c_{ik} + gap$
10:      $d_4 = D(i, j-1, k-1) + c_{jk} + gap$
11:      $d_5 = D(i-1, j, k) + 2 * gap$
12:      $d_6 = D(i, j-1, k) + 2 * gap$
13:      $d_7 = D(i, j, k-1) + 2 * gap$
14:      $D(i, j, k) = Min[d_1, d_2, d_3, d_4, d_5, d_6, d_7]$
15:    end for
16:  end for
17: end for
```

sequences to be compared, there will be k loops in this algorithm and $2^k - 1$ cells will be used to calculate each cell of matrix D.

Carrillo-Lipman Bound. Carrillo-Lipman [2] showed that it is not necessary to explore the whole search space in order to obtain the optimal Multiple Sequence Alignment. They defined a lower and an upper bound which confine the region that contains the optimal alignment and thus restrict the area of the n-dimensional matrix to be calculated.

The lower (L) and upper (U) bounds are calculated as explained in the following paragraphs.

Equation 1 calculates the lower bound, based on the sum-of-pairs score. In this equation, $scale(S_i, S_j)$ is the weight of each pairwise aligment, usually choosen via an evolutionary tree of N sequences. $d(S_i, S_j)$ is the score of the optimal pairwise alignment of S_i and S_j .

$$L = \sum_{i<j} d(S_i, S_j) \cdot scale(S_i, S_j) \qquad (1)$$

L is a lower bound for the following reason. Since $d(S_i, S_j)$ is the optimal score between sequences S_i and S_j, this is the lowest possible score for (S_i, S_j). Since the score of the Multiple Sequence Alignment is a sum-of-pairs, i.e., an addition of all pairwise scores, and the scores are non-negative, therefore the optimal Multiple Sequence Alignment score must be greater or equal to the sum of all optimal pairwise scores.

Consider that A^o is the optimal alignment, $c(A^o)$ is the score of the optimal alignment, $scale(S_i, S_j)$ is 1, and $c(A^o_{i,j})$ is the score of the pairwise alignment of i and j. The Carrillo-Lipman Bound is given by Inequation 2:

$$c(A^o_{i,j}) \leq d(S_i, S_j) + U - L \tag{2}$$

$U - L$ can be obtained by a heuristic alignment A^h. This alignment induces a score c in the sum-of-pairs i and j, and so $U - L$ is obtained by:

$$U - L = \sum_{i<j} [c(A^h_{i,j}) - d(S_i, S_j)] \tag{3}$$

Equation 3 shows that the lower and upper bounds have a value which is based on the projection of a heuristic alignment subtracted from the scores of the pairwise alignments. It is guaranteed that the cells that are outside those bounds do not contribute to the calculation of the optimal Multiple Sequence Alignment [2]. Frequently, the difference $U - L$ is called δ_{msa}.

3 Related Work

Even though there are many works in the literature that implement heuristic methods in GPUs [6], [1], FPGAs [16] and clusters [5], [9], there are very few works that implement exact MSA methods in high performace computing platforms.

Helal et al. [4] propose the use of a master-slave architecture to execute the exact MSA algorithm in a cluster. In order to reduce the search space, the authors use geometrical relations over hyper-diagonals and hyper-lattices. They were able to compare sets of 3, 4 and 5 sequences of small size in a cluster with 8 cores. The MSA comparison of 5 sequences took more than 2 days.

Masuno et al. [8] implemented the exact MSA with the Carrillo-Lipman bound in FPGA. In their proposal, the n-dimensional dynamic programming matrix is transversed in windows defined over 2D-projections, where i, j vary, whereas the other dimensions remain fixed. In order to implement the Carrillo-Lipman bound algorithm, a heuristic MSA is calculated in CPU and its score is transferred to the FPGA. Before calculating a window, the algorithm tests if it is inside the Carrillo-Lipman bound. If not, the window is not calculated. Two different circuits are proposed to calculate MSAs of 4 and 5 sequences, respectively. The computation of an MSA of 5 sequences took about 5 minutes in the FPGA.

4 Design of MSA-GPU

In order to execute the exact MSA algorithm, we opted to use GPUs since they provide massive SIMD (Single Instruction Multiple Data) parallelism for large-scale problems. For MSA-GPU, we designed two strategies, which are described in Sections 4.1 and 4.2.

4.1 Coarse-Grained Strategy

In the coarse-grained strategy, we used a multidimensional wavefront calculated by one GPU block and, inside the block, each GPU thread calculates one cell of the dynamic programming matrix D (Algorithm 1).

Figure 3 (coarse) illustrates the coarse-grained strategy, showing the tasks executed in the CPU and in the GPU. First, the user provides the sequences and the Carrillo-Lipman δ_{msa} (*delta_ cl*) (Section 2.2). Then, the data structures in the GPU are initialized. After that, the kernels in GPU are executed for each multidimensional diagonal with n threads. The wavefront indexes are calculated in GPU and represent the coordinates of the cells that can be calculated in parallel. At the end of each kernel computation, the variable *bound_ reached* is used to discard unnecessary multidimensional diagonals. At the very end of the computation, the optimal score is obtained and given as output to the user.

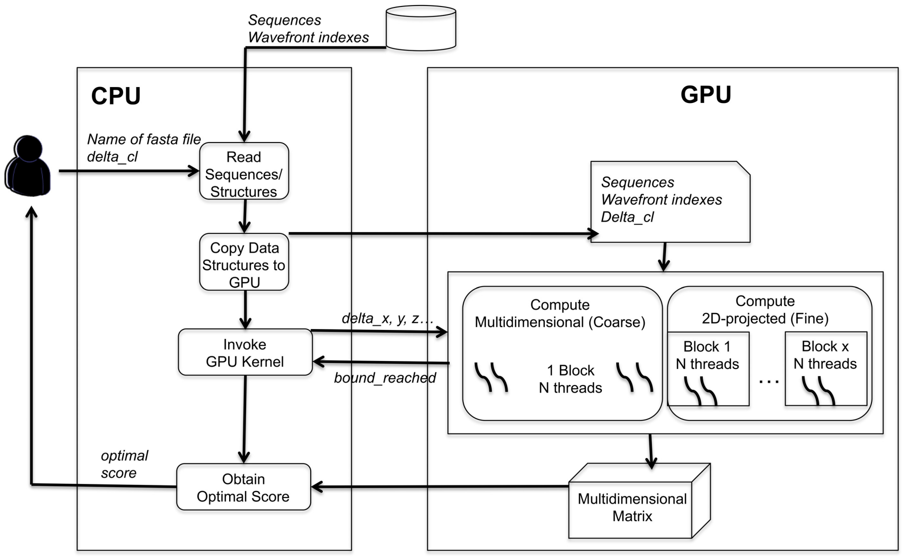
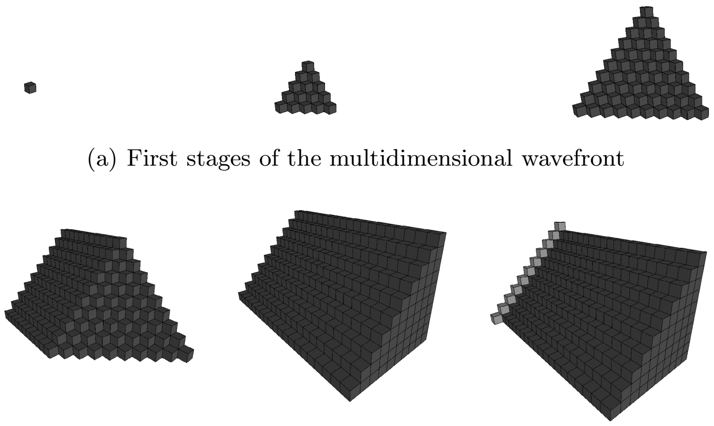


Fig. 3. Overview of the coarse-grained and fine-grained strategies

When using this coarse-grained strategy, we observed that the parallelism in the multidimensional wavefront grows a lot. But when the algorithm execute the last stages, some cells that might be calculated in paralell do not have the initial shape of a multidimensional wavefront. In this case, using the same wavefront shape would imply in the recalculation of some cells, reducing the throughput. The shape of the multidimensional wavefront processing is shown in Figure 4(a) and 4(b). In the left-corner picture in Figure 4(b) we can observe the effect of the cells without the wavefront shape.

4.2 Fine-Grained Strategy

With this strategy, we intended to augment the parallelism by using more than one GPU block. This was possible because we chose the wavefront indexes to



(a) First stages of the multidimensional wavefront

(b) Last stages of the multidimensional wavefront

Fig. 4. Multidimensional wavefront in the coarse-grained strategy

calculate the cells of the DP matrix using 2D projections. This led to a more regular dependency pattern and, thus, multiple blocks could be used.

The multi-block fine-grained strategy works as follows. Inside each block, the threads will calculate the same cell of the dynamic programming matrix. Therefore, lines 7 to 13 in Algorithm 1 will be parallelized, where each thread will calculate values d_1 to d_7 in parallel. Besides that, many cells that belong to the same projected diagonal will be calculated in parallel by different blocks. Figure 3 (fine) illustrates this strategy.

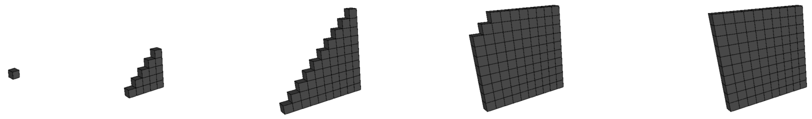
In this strategy, we traverse the search space using 2D projections, as shown in Figure 5(a) and 5(b). In this figure, it can be seen that a more regular pattern is obtained.

5 Experimental Results

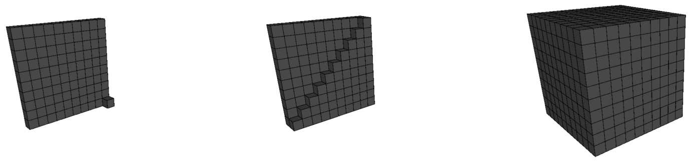
The strategies proposed in Section 4 were implemented in CUDA C, using the CUDA toolkit 4.1.21. The results were obtained with the NVidia GTX 580 GPU (512 cores and 1.5 GB RAM). This GPU was connected to an Intel Core i5 host machine, with 6GB RAM.

In our tests, we used real and synthetic sequences (Table 1). The real sequences were obtained from the PFAM (*pfam.sanger.ac.uk*) and Balibase 2.0 (*bips.u-strasbg.fr/fr/Products/Databases/BAliBASE2/*) databases. The synthetic sequences were constructed in order to reproduce easy, medium, hard and very hard MSA patterns.

First, we compared the sequences in Table 1 with the coarse (Section 4.1) and the fine granularity (Section 4.2) approaches. In the last case, one block and multiple blocks were used. The results are shown in Table 2. In this table, we can see that, when the sequences are small, the fine-grained strategy achieves better



(a) First stages of the projected wavefront



(b) Last stages of the projected wavefront

Fig. 5. 2D-projected wavefront in the fine-grained strategy

Table 1. Sequences used in the Tests

Name	Database	Reference	Sizes		
Seq15	PFAM	PF10550	15	15	14
Seq22	PFAM	PF08095	22	22	22
Seq42	PFAM	PF03855	41	44	42
Seq59	PFAM	PF08184	59	59	59
Seq110	PFAM	PF11513	106	111	110
Seq122	PFAM	PF06453	122	122	122
Seq143	PFAM	PF09155	143	143	143
Seq162	PFAM	PF03426	158	158	162
Seq373	Balibase2	1pedA	350	326	373
Seq446	Balibase2	1ad3	423	441	446
Seq416	Synthetic	synthetic_easy	231	416	363
Seq417.1	Synthetic	synthetic_medium	231	447	363
Seq417.2	Synthetic	synthetic_hard	231	447	423
Seq453	Synthetic	synthetic_veryhard	231	446	453

results than the coarse-grained strategy, even with only one block. When we augment the sizes of the sequences, the coarse-grained strategy surpasses the fine-grained strategy (one block) since the reduced number of threads is insufficient to deal with the parallelism. For all the sequences compared, the fine-grained strategy (multiple blocks) was able to achieve the best execution times, with a great improvement over the other approaches. For instance, when comparing sequences *Seq446*, the execution time was reduced from 8min55s (coarse-grained) to 6.62s (fine-grained multi-block).

We also compared the execution times of the fine-grained multi-block GPU strategy with the MSA 2.0 CPU program. This program is publicly available at *www.ncbi.nim.nih.gov/CBBresearch/Schaffer/msa.html*, runs in CPU and is often used to obtain exact multiple sequence alignments. The execution times in our host machine (one core) and in the GPU are shown in Table 3. In this table,

Table 2. Comparison Between the Coarse-Grained (C-Grain) with 990 Threads, Fine-Grained One Block (F-Grain-1B) with 7 Threads, and the Fine-Grained Multiple Blocks (F-Grain-MB) with Variable Number of Threads

Name	C-Grain	F-Grain-1B	F-Grain-MB	
Seq15	1.09s	0.12s	0.05s	7 to 105 threads
Seq22	1.34s	0.27s	0.07s	7 to 154 threads
Seq42	3.61s	1.68s	0.10s	7 to 308 threads
Seq59	4.60s	4.28s	0.14s	7 to 413 threads
Seq110	16.51s	27.25s	0.35s	7 to 777 threads
Seq122	26.05s	37.71s	0.46s	7 to 854 threads
Seq143	29.84s	1min04s	0.47s	7 to 1001 threads
Seq162	41.07s	1min25s	0.74s	7 to 1106 threads
Seq373	5min34s	14min23s	4.04s	7 to 2282 threads
Seq416	4min16s	16min12s	3.69s	7 to 2912 threads
Seq446	8min55s	29min30s	6.62s	7 to 3087 threads
Seq447.2	5min11s	>30 min	6.49s	7 to 3129 threads

Table 3. Execution Times for the MSA 2.0 CPU program and the MSA-GPU fine-grained multi-block GPU strategy using Full Space Search (FSS) and reducing the Search Space with Carrillo-Lipman (CL)

Name	δ_{msa}	MSA 2.0 CPU (s)	MSA-GPU (FSS) (s)	MSA-GPU (CL) (s)
Seq15	15	0.001	0.051	0.051
Seq22	15	0.001	0.068	0.056
Seq42	15	0.002	0.102	0.096
Seq59	15	0.002	0.140	0.126
Seq110	15	0.005	0.353	0.313
Seq122	15	0.006	0.461	0.435
Seq143	15	0.007	0.473	0.423
Seq162	15	0.010	0.743	0.650
Seq373	137	0.180	4.041	3.324
Seq446	29	0.140	6.624	6.203
Seq416	502	1.652	3.693	2.611
Seq417.1	185	23.507	3.837	3.011
Seq417.2	484	28.711	6.478	4.764
Seq 453	387	31.078	6.948	3.612

the second column presents the δ_{msa} parameter given to the MSA-GPU program to use the Carrillo-Lipman bound. For the MSA 2.0 CPU program, the default parameters were used, whenever possible. For sequences *Seq447.1, Seq447.2* and *Seq453*, the MSA 2.0 program was not able to retrieve the score/alignment with the default parameters. Therefore, we had to augment the δ_{msa} parameter to 100, 100 and 1000, respectively. The third, forth and fifth columns show, respectively, the execution times for the MSA 2.0 CPU program, the MSA-GPU fine-grained multi-block strategy (Full Space Search) and the MSA-GPU fine-grained multi-block strategy (Carrillo-Lipman).

We can see that, for sequences *Seq15* to *Seq446*, the MSA 2.0 CPU program is able to execute very quickly, with much better execution times than the GPU program. This happens because the sequences have high similarity and, for this reason, the CPU program is able to prune efficiently the search space. Even though our GPU program also prunes the search space, these sequence sets do not have enough parallelism to surpass the CPU implementation.

Sequences*Seq447.1, Seq447.2* and *Seq453* are more complex cases. Therefore, the MSA 2.0 program was not able to execute with the default parameters and the δ_{msa} parameter was augemented. Augmenting the δ_{msa} parameter reduces the area pruned by the Carrillo-Lipman bound , thus augmenting the execution time. For these cases, the GPU program presents a speedup of 7.8x for the sequence set *Seq447.1* (medium similarity), 6.02x for *Seq447.2* (low similarity) and 8.60x for *Seq453* (very low similarity).

6 Conclusion and Future Work

In this paper, we proposed and evaluated MSA-GPU, a parallel tool that is able to calculate exact MSAs in GPUs. We proposed coarse-grained and fine-grained mechanisms to express the parallelism, with different strategies to compute the dynamic programming matrix (multidimensional and 2D-projected wavefronts).

The results obtained with real and synthetic sequence sets composed of 3 sequences show that the fine-grained multiblock strategy achieves better execution times than the coarse-grained strategy when the sequences have a reasonable size. When comparing the fine-grained multi-block MSA-GPU with the MSA 2.0 CPU program, we observed that the CPU program has very low execution times, when the sequences are similar. For sequences with medium/low similarity, the execution time augments a lot and, in these cases, MSA-GPU outperforms MSA 2.0, being able to reduce the execution time considerably.

As future work, we intend to extend MSA-GPU to compare up to 7 sequences. We also intend to investigate alternative bounds to the Carrillo-Lipman bound that guarantee that the optimal result will be produced by calculating a smaller area in the n-dimensional dynamic programming matrix.

References

1. Blazewicz, J., Frohmberg, W., Kierzynka, M., Wojciechowski, P.: G-MSA - A GPU-based, fast and accurate algorithm for multiple sequence alignment. Journal of Parallel and Distributed Computing 73(1), 32–41 (2013)
2. Carrillo, H., Lipman, D.: The Multiple Sequence Alignment Problem. SIAM Journal of Applied Math. 48, 1073–1082 (1988)
3. Higgins, D.G., Thompson, J.D., Gibson, T.J.: ClustalW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix. Nucleic Acids Research 22, 4673–4680 (1994)
4. Helal, M., Mullin, L.R., Potter, J., Sintchenko, V.: Search Space Reduction Technique for Distributed Multiple Sequence Alignment. In: NPC, pp. 219–226 (2009)

5. Li, K.B.: ClustalW-MPI: ClustalW analysis using distributed and parallel computing. Bioinformatics 19(12), 1585–1586 (2003)
6. Liu, Y., Schmidt, B., Maskell, D.L.: MSA-CUDA: Multiple Sequence Alignment on Graphics Processing Units with CUDA. In: ASAP, pp. 121–128 (2009)
7. Loytynoja, A., Goldman, N.: Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. Science 320, 1632–1635 (2008)
8. Masuno, S., Maruyama, T., Yamaguchi, Y., Konagaya, A.: An FPGA Implementation of Multiple Sequence Alignment Based on Carrillo-Lipman Method. In: Field Programmable Logic and Applications, pp. 489–492 (2007)
9. Macedo, E.A., Melo, A.C.M.A., Pfitscher, G.H., Boukerche, A.: Multiple biological sequence alignment in heterogeneous multicore clusters with user-selectable task allocation policies. The Journal of Supercomputing 63(3), 740–756 (2013)
10. Morgenstern, B., Dress, A., Werner, T.: Multiple DNA and protein sequence alignment based on segment-to-segment comparison. PNAS, USA, 12098–12103 (1996)
11. Morgenstern, B., Frech, K., Dress, A., Werner, T.: DIALIGN: Finding local similarities by multiple sequence alignment. Bioinformatics 14(3), 290–294 (1998)
12. Mount, D.W.: Bioinformatics: sequence and genome analysis. Cold Spring Harbor Laboratory Press (2004)
13. Notredame, C.: T-Coffee: a novel method for fast and accurate multiple sequence alignment. Journal of Molecular Biology 302, 205–217 (2000)
14. Notredame, C., Higgins, D.G.: SAGA: sequence alignment by genetic algorithm. Nucleic Acids Research 24, 1515–1524 (1996)
15. Novak, A., Miklos, I., Lyngso, R., Hein, J.: StatAlign: an extendable software package for joint Bayesian estimation of alignments and evolutionary trees. Bioinformatics 24(20), 2403–2404 (2008)
16. Oliver, T.F., Schmidt, B., Nathan, D., Clemens, R., Maskell, D.L.: Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW. Bioinformatics 21(16), 3431–3432 (2005)
17. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
18. Wang, L., Jiang, T.: On the complexity of multiple sequence alignment. Journal of Computational Biology 4, 337–348 (1994)