Parallel Biological Sequence Comparison in Linear Space with Multiple Adjustable Bands

Gabriel H.G. Silva, Edans F.O. Sandes, George Teodoro, and Alba C.M.A. Melo(✉)

Department of Computer Science, University of Brasilia (UnB), Brasilia, Brazil
alves@unb.br

Abstract. In this paper, we propose and evaluate Fickett-MM, a parallel strategy that combines the algorithms Fickett and Myers-Miller, splitting a pairwise sequence comparison into multiple comparisons of subsequences and calculating an appropriate Fickett band to each subsequence comparison (block). With this approach, we potentially reduce the number of cells calculated in the dynamic programming matrix when compared to Fickett, which uses a unique band to the whole comparison. Our adjustable multi-block strategy was integrated to the stage 4 of CUDAlign, a state-of-the-art parallel tool for optimal biological sequence comparison. Fickett-MM was used to compare real DNA sequences whose sizes ranged from 10KBP (Thousands of Base Pairs) to 47MBP (Millions of Base Pairs), reaching a speedup of 59.60× in the 10MBP × 10MBP comparison when compared to CUDAlign stage 4.

Keywords: Parallel biological sequence comparison · Multiple adjustable bands

1 Introduction

Pairwise biological sequence comparison is a widely used operation in Bioinformatics. It produces as output a score, which represents the similarity between the sequences, and an alignment [1]. The optimal global alignment with linear gap can be obtained with the Needleman-Wunsh (NW) algorithm [9], which is based on dynamic programming (DP) and has $O(mn)$ time and space complexity, where m and n are the lengths of the sequences. Smith-Waterman (SW) [14] proposed a DP-based algorithm that computes optimal local alignments with linear gap with the same time and space complexity.

Gotoh [3] modified the NW algorithm, calculating optimal alignments with the affine gap model. Since gaps tend to occur together in nature, the affine gap model is more appropriate for realistic scenarios. Hirschberg [4] proposed a variant of the NW algorithm that retrieves optimal alignments in linear space ($O(m + n)$) with linear gap. This variant was further modified by Myers-Miller (MM) [8] in order to use the affine gap model. Fickett [2] proposed an algorithm that retrieves the optimal global alignment by calculating only a k-band of the

D. Figueiredo et al. (Eds.): AlCoB 2017, LNBI 10252, pp. 167–179, 2017.
DOI: 10.1007/978-3-319-58163-7_12

DP matrix near the main diagonal, where k is the number of diagonals computed, executing thus in $O(kn)$ time and space. If the alignment does not fall into the k-band, the band is enlarged and the DP matrix is iteratively re-computed until the alignment can be retrieved.

The use of the NW and SW algorithms and its variants to compare long DNA sequences or a protein sequence to a huge genomic database can lead to very high execution times and, for this reason, parallel solutions are usually employed. In the literature, there are many proposals that execute NW or SW variants in parallel architectures such as clusters [7,12], FPGAs (Field Programmable Gate Arrays) [13,16], GPUs (Graphics Processing Units) [6,10] and Intel Xeon Phis [5,15], among others. CUDAlign 4.0 [10] is a state-of-the-art tool which computes optimal local alignments between long DNA sequences in linear space using 5 stages. Stage 1 executes phase 1 of the Gotoh algorithm (score calculation) with affine-gap and stages 2 to 5 execute phase 2 (traceback), with an adapted version of the MM algorithm.

In this paper, we propose and evaluate Fickett-MM, a parallel strategy which combines MM with a variant of Fickett's [2]. Unlike the original Fickett algorithm, we divided the alignment problem into several parts and computed a different k-band for each part of the problem, which is adjusted to its alignment characteristics. Fickett-MM was implemented in C++/pthreads and integrated to the stage 4 of CUDAlign. The results obtained with real DNA sequences whose sizes varied from 10 KBP to 47 MBP show that our strategy is able to achieve a speedup of up to 59.60× in stage 4 of CUDAlign, when compared to the original implementation. In the longest comparison, the execution time of CUDAlign stage 4 was reduced from 2 min and 54 s to 30 s.

The remainder of this paper is organized as follows. Section 2 presents algorithms for optimal biological sequence alignment and the CUDAlign tool is presented in Sect. 3. The design of Fickett-MM is explained in Sect. 4. In Sect. 5, experimental results are discussed and Sect. 6 concludes the paper.

2 Biological Sequence Comparison

2.1 Basic Algorithms - NW and SW

The Needleman-Wunsh (NW) [9] algorithm is based on DP and retrieves the optimal global alignment in $O(mn)$ space and time, executing in two phases: (a) calculate the DP matrix and (b) retrieve the alignment (traceback).

The first phase receives sequences S_0 and S_1, with lengths n and m, and computes the DP matrix H as follows. The first row and column are filled with $H_{i,0} = i * g$ and $H_{0,j} = j * g$, where g is the gap penalty and i and j represent the sizes of the prefixes of the sequences. The remaining cells are calculated with the recurrence relation expressed by Eq. (1) [9].

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + p(i,j) \\ H_{i,j-1} - g \\ H_{i-1,j} - g \end{cases} \quad (1)$$

	*	T	A	G	T	C
*	**0**	−2	−4	−6	−8	−10
T	−2	**1**	−1	−3	−5	−7
A	−4	−1	**2**	0	−2	−4
G	−6	−3	0	**3**	**1**	−1
C	−8	−5	−2	1	2	**2**

T A G T C
T A G – C

(a)

	*	T	A	G	T	C
*	**0**	0	0	0	0	0
T	0	**1**	0	0	1	0
A	0	0	**2**	0	0	0
G	0	0	0	**3**	1	0
C	0	0	0	1	2	2

T A G
T A G

(b)

Fig. 1. DP matrices and alignments for S_0 and S_1 (mi = −1, ma = +1, g = −2). (a) NW matrix; (b) SW matrix.

In this equation, if DNA or RNA sequences are compared, $p(i, j)$ is the match punctuation (ma), if $S_0[i] = S_1[j]$, or the mismatch punctuation (mi), otherwise. If amino acid sequences are compared, $p(i, j)$ is given by a given 20×20 substitution matrix [1]. Each cell $H_{i,j}$ keeps an indication of which cell ($H_{i-1,j-1}$), ($H_{i,j-1}$) or ($H_{i-1,j}$) was used to produce its value (arrows in Fig. 1). The optimal score is in cell $H_{m,n}$. In order to produce the alignment, phase 2 (traceback) is executed from the bottom right cell in the DP matrix, following the indications until the top left cell is attained. Figure 1(a) illustrates the DP matrix calculated by NW. The optimal score is 2 and the optimal global alignment, obtained in the traceback phase, is shown below the DP matrix.

When the biologists are interested in calculating how similar the fragments of the sequences are, local alignment is usually applied and the Smith-Waterman (SW) algorithm is used. The SW uses DP, has the same complexity of NW and executes in two phases. Nevertheless, NW and SW are distinct in three ways. First, differently from NW, the elements of the first row and column of the SW matrix are set to zero. Second, the SW recurrence relation is slightly different from the NW recurrence relation since no negative values are allowed in SW (Eq. (2)) [14]. Finally, the cell that contains the optimal local score is the cell $H_{i,j}$ which has the highest value in H. In the traceback phase, SW starts from cell $H_{i,j}$, following the arrows until a cell whose value is zero is found. Figure 1(b) illustrates the DP matrix calculated by SW. In this figure, the optimal score is 3 and the optimal local alignment is shown below the DP matrix.

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + p(i,j) \\ H_{i,j-1} - g \\ H_{i-1,j} - g \\ 0 \end{cases} \tag{2}$$

2.2 NW and SW Variants

To produce more biologically relevant results, Gotoh [3] proposed an algorithm that implements the affine-gap model, with two different gap penalties:

one to initiate a sequence of gaps (G_{first}) and another to extend it (G_{ext}). Gotoh calculates three DP matrices: H, E and F, where H keeps track of matches/mismatches and E and F keep track of gaps in each sequence (Eqs. 3, 4 and 5).

$$H_{i,j} = \max \begin{cases} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} - p(i,j) \end{cases} \tag{3}$$

$$E_{i,j} = \max \begin{cases} E_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{first} \end{cases} \tag{4}$$

$$F_{i,j} = \max \begin{cases} F_{i-1,j} - G_{ext} \\ H_{i-1,j} - G_{first} \end{cases} \tag{5}$$

When long sequences are compared, linear space algorithms should be used. One of the first linear space algorithms for sequence comparison is the one proposed by Hirschberg [4]. First, the DP matrix is calculated from the beginning to the middle row ($i*$), storing only the last row calculated. After that, the DP matrix is calculated from the end to $i*$, over the reverses of the sequences. At this point, there are two middle rows, one calculated with the original sequences and another one calculated with the reverses of the sequences. Hirschberg proved that the position where the addition of the corresponding values in these two middle rows is maximum belongs to the optimal alignment [4]. This point is called crosspoint and it divides the problem into two smaller subproblems, which are processed recursively, until trivial solutions are found. Myers-Miller (MM) [8] adapted Hirschberg to the Gotoh algorithm by using two additional vectors. The first and second recursions of the MM algorithm are shown in Fig. 2.

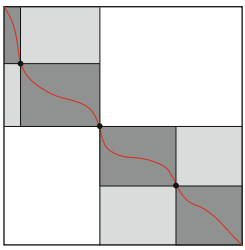


Fig. 2. Myers-Miller (MM) algorithm. The black circles represent the crosspoints.

Fickett [2] proposed an algorithm that can be executed quickly if the sequences compared are very similar. In this case, the alignment between the sequences is confined in a small region near the main diagonal of the DP matrix.

Thus, Fickett only calculates and stores a small set of diagonals near the main diagonal (k-band), with time and space complexity $O(kn)$. The k-band is estimated with a heuristic measurement of the similarity of the sequences. The optimal score is contained in cell $H_{n,m}$ and it is used to do the traceback over the band. If the k-band was underestimated, the alignment cannot be retrieved (Fig. 3a). In this case, the algorithm enlarges iteratively the k-band and the DP matrix is calculated for the new k-band, until the whole alignment is obtained (Fig. 3b).

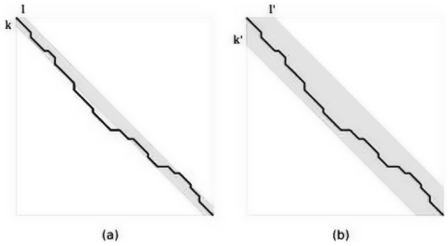
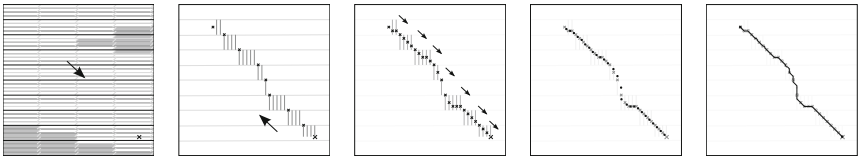


Fig. 3. Fickett's algorithm. The gray area represents the k-band.

Although algorithms MM and Fickett have been proposed to the global alignment problem, they can be easily adapted to the local alignment case as follows. First, the DP matrix is processed with SW, giving as output the highest score. Second, the matrix is recalculated from the position where the optimal score occurs over the reverses of the sequences until the position where the optimal local alignment begins is found. With these two positions, the problem is transformed into a global alignment problem and MM or Fickett can be readily applied.

3 Design of CUDAlign 4.0

CUDAlign [10] is a tool that obtains the optimal local alignment between two long DNA sequences in GPU, using adapted versions of the Gotoh and MM algorithms (Sect. 2.2). CUDAlign executes in 5 stages, as shown in Fig. 4.



(a) Stage 1 (b) Stage 2 (c) Stage 3 (d) Stage 4 (e) Stage 5

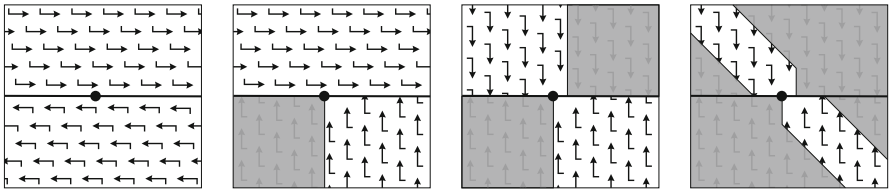
Fig. 4. General overview of CUDAlign

Stage 1 corresponds to the first phase of the Gotoh algorithm and executes in GPU in linear space, giving as output the highest score and its position in the matrix. Stage 1 uses mainly two optimizations. The optimization *cells delegation* processes the Gotoh matrices in multiple blocks, in a parallelogram wavefront shape, allowing maximum parallelism most of the time. The optimization *block pruning*, shown in gray in Fig. 4(a), does not compute DP cells which certainly do not contribute to the optimal alignment. In order to accelerate the further stages, some rows of the DP matrices (special rows) are stored. Stages 2, 3, 4 and 5 implement phase 2 (traceback).

In stage 2, a variant of MM is used in GPU to retrieve the midpoints that cross the special rows (crosspoints), which belong to the optimal alignment. Unlike MM, the special rows contain information about the maximum score and can be used to accelerate the computation. So, it is sufficient to find the position in the special row where the addition is equal to the (already known) maximum score. With this observation, Stage 2 starts from the position in the DP matrix in which the optimal score occurs and processes over the reverses of the sequences, calculating the area column by column (instead of row by row, as in the original MM) and finding midpoints until the beginning of the optimal local alignment is found. In stage 3, the beginning and end of the optimal local alignment are received as input. Moreover, the special columns saved to disk in stage 2 are used. Stage 3 starts from the beginning of the alignment and uses the special columns to retrieve more crosspoints in GPU.

Stage 4 executes in CPU using a modified MM algorithm between each successive pair of crosspoints (partition) found on stage 3, with multiple threads. The goal of stage 4 is to increase the number of crosspoints until the distance between any successive pair of crosspoints is smaller than a given limit (e.g. 16×16). Figure 4(d) shows the additional crosspoints obtained in stage 4.

Figure 5 presents four ways to process a partition in stage 4, where the gray areas represent the regions that do not need to be processed after the crosspoint is found. The conventional MM algorithm processes both halves of the partition entirely (Fig. 5(a)). CUDAlign 2.0 introduced an optimization called Orthogonal Execution, which processes the top half of the partition over rows and the bottom half of the partition over columns (Fig. 5(b)). CUDAlign 4.0 extended this idea processing both halves of the partition over columns, alternating columns from



(a) Conventional (b) Orthogonal (c) Optimized (d) Ficket-MM

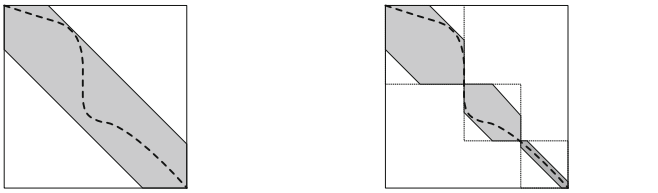
Fig. 5. DP submatrix computed in stage 4. Area in gray are not processed.

each half (Fig. 5(c)). The Ficket-MM algorithm proposed in this paper (Sect. 4) reduces further the area processed (Fig. 5(d)).

Stage 5 aligns in CPU each partition found in stage 4 using NW. Then it concatenates all the results, giving as output the full optimal alignment (Fig. 4(e)). Stage 6 is an optional stage used only for visualization of the alignment.

4 Design of Fickett-MM

The main goal of Fickett-MM is to reduce the area computed in the alignment retrieval. In order to achieve this goal, we combine the well-known algorithms Fickett and MM (Sect. 2.2), creating the notion of adjustable bands. We assume that, as in MM, the computation is divided into blocks and the scores at the top left and bottom right corners of a block are known. With this information, we are able to define computation bands of different sizes, one for each block, in which the optimal alignment is guaranteed to be found.



(a) Fickett's band (b) Fickett-MM adjustable bands

Fig. 6. Bands in the Fickett algorithm and in the Fickett-MM algorithm

In Fig. 6, we illustrate the main difference between Fickett-MM and the original Fickett algorithm. In Fig. 6(a), Fickett's band (gray area) must encompass the whole alignment (dashed line), which has a considerable number of gaps in its beginning. For this reason, the size of the band is big, even though the alignment does not have many gaps in its end. On the other hand, Fickett-MM (Fig. 6(b)) defines three different bands (gray area), one for each block.

The efficiency of Fickett-MM is highly dependent on a good estimation of size of the computation bands. The scores at the upper left corner ($score_l$) and at the bottom-right corner ($score_r$) are known and a block is the rectangle defined by these two points.

The size of the band for each block is computed with Eq. 6 and it depends on four terms: PM, $score_d$, DPM and min_g, which are explained in the following paragraphs.

$$band = \left\lceil \frac{PM - score_d}{DPM} \right\rceil + min_g \tag{6}$$

The perfect match term (PM) computes the maximum score of the block in the best case, i.e., all the corresponding characters of the subsequences are

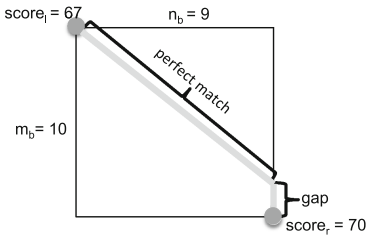


Fig. 7. Elements used in the perfect match (PM) computation.

the same (perfect match). Since the lengths of the subsequences may not be the same (Fig. 7), the length of the smaller subsequence is multiplied by the match punctuation and subtracted by the difference on the lengths of the subsequences multiplied by the gap penalty. In Eq. 7, ma is the punctuation for matches and gap is the punctuation for gap extension.

$$PM = min(m_b, n_b) * ma - (max(m_b, n_b) - min(m_b, n_b)) * |gap| \tag{7}$$

The difference between scores ($score_d$) in Eq. 6 is simply the difference between the score at the bottom right corner and the score at the upper left corner: $score_d = score_r - score_l$.

The deviation from the perfect match (DPM) term takes into account the fact that each time a gap is introduced, we need at least another gap to return to the perfect match case, and, since two gaps are introduced, one punctuation for match (ma) will not be counted. Equation 8 presents this computation.

$$DPM = |2 * gap| + ma \tag{8}$$

Finally, the term min_g calculates the difference between the sizes of the subsequences (m_b and n_b) since it indicates the minimum number of gaps needed for the band to contain the optimal alignment (Eq. 9).

$$min_g = max(m_b, n_b) - min(m_b, n_b) \tag{9}$$

The size of the band is computed by considering the worst case, i.e., gaps are introduced instead of mismatches. In addition, since we do not know in which sequence gaps will be introduced, we apply the same value of band for both sides of the perfect match case. With this, we guarantee that the band encompasses the optimal alignment even though in some cases it will be larger than necessary. In order to illustrate the computation of the size of the band, consider the values in Fig. 7 and assume that $ma = +1$, $gap = -2$. In this case, $PM = 7$, $score_d = 3$, $DPM = 5$ and $min_g = 1$, giving 2 as the size of the band in each side of the perfect match case (Eq. 6).

Algorithm 1 presents the pseudocode of Fickett-MM. It receives as input the subsequences S'_0 and S'_1 as well as the scores in the upper left and bottom right

Algorithm 1. Fickett-MM

Require: Subsequences S'_0 e S'_1, Scores $score_l$ and $score_r$
Ensure: *crosspoint*

```
1: /*Calculates the size of the band*/
2: score_d ← score_r − score_l
3: k ← calculate_band(S'_0, S'_1, score_d)
4: seq1_length ← size(S'_0)
5: j ← 0
6: loop
7:    /*Calculates the extremities of the columns inside the band*/
8:    upper_left ← Calculate_FickettMM_upper_left(j, k)
9:    lower_left ← Calculate_FickettMM_lower_left(j, k)
10:   upper_right ← Calculate_FickettMM_upper_right(seq1_length − j, k)
11:   lower_right ← Calculate_FickettMM_lower_right(seq1_length − j, k)
12:   /*Calculates the recurrence equation inside the band*/
13:   crosspoint1[j] ← Compute_FickettMM(upper_left, lower_left, S'_0, S'_1)
14:   crosspoint2[j] ← Compute_FickettMM(upper_right, lower_right, S'_0, S'_1)
15:   if j > seq1_length/2 then
16:      crosspoint ← crosspoint1[j] + crosspoint2[seq1_length − j]
17:      if check(crosspoint, score_d) = TRUE then
18:         return crosspoint
19:      end if
20:      crosspoint ← crosspoint2[j] + crosspoint1[seq1_length − j]
21:      if check(crosspoint, score_d) = TRUE then
22:         return crosspoint
23:      end if
24:   end if
25:   j++
26: end loop
```

cells ($score_l$ and $score_r$). The computation of the size of the band is done in lines 2 and 3 and its value is stored in k. Then, a loop is executed from lines 6 to 26 for every column j as follows. In lines 8 to 11, the algorithm calculates the extremeties of column j (forward direction) and column $seq1\_length - j$ (reverse direction) up to the middle row. Then, the recurrence equation is calculated for both columns j and $seq1\_length - j$, as illustrated in Fig. 5(d). The values of the cells in the middle row are stored in vectors *crosspoint*1 (line 13) and *crosspoint*2 (line 14). When the middle column is attained (line 15), crosspoints 1 and 2 are added accordingly (lines 16 and 20) and the algorithm checks if the results match $score_d$ (i.e. $score_r - score_l$). If one of these values match (lines 17 and 21), this crosspoint is returned (lines 20 and 24).

5 Experimental Results

Fickett-MM was implemented in C/C++/pthreads and integrated to the stage 4 of CUDAlign 4.0. In our tests, we used a desktop with a CPU Intel Core i7 3770 (4 hardware cores), 8 GB RAM, 1 TB disk and a GPU NVidia GTX 680 (1536 cores and 2 GB RAM).

The following parameters were used in the tests: *ma (match)* $= +1$, *mi (mismatch)* $= -3$, G_{first} *(First gap)* $= -5$, G_{ext} *(Gap extension)* $= -2$, *number of threads* $= 8$ and *final size of block* $= 24 \times 24$.

The experiments used real DNA sequences, retrieved from the NCBI (National Center for Biotechnology Information) at www.ncbi.nlm.nih.gov.

Table 1. Sequences used in the tests.

Comparison size	Sequence S_0		Sequence S_1	
	Accession	Name	Accession	Name
10K × 10K	AF133821.1	HIV-1 isolate MB2059	AY352275.1	HIV-1 isolate SF33
57K × 57K	AF494279.1	C. globosum	NC_001715.1	A. macrogynus
162K × 172K	NC_000898.1	H. herpesvirus 6B	NC_007605.1	H. herpesvirus 4
543K × 536K	NC_003064.2	A. fabrum C58	NC_000914.1	Rhizobium sp. NGR234
1M × 1M	CP000051.1	C. trachomatis	AE002160.2	C. muridarum
3M × 3M	BA000035.2	C. efficiens	BX927147.1	C. glutamicum
5M × 5M	AE016879.1	B. anthracis str. Ames	AE017225.1	B. anthracis str. Sterne
7M × 5M	NC_005027.1	R. baltica SH	AE016879.1	B. anthracis str. Ames
10M × 10M	NC_017186.1	A. mediterranei S699	NC_014318.1	A. mediterranei U32
23M × 25M	NT_033779.4	D. melanogaster chr. 2L	NT_037436.3	D. melanogaster chr. 3L
47M × 32M	NC_000021.7	H. sapiens chr. 21	BA000046.3	P. troglodytes chr. 22

Table 2. Execution time, speedup and characteristics of the alignment

Comparison	Fickett-MM (ms)	CUDAlign stage 4 (ms)	Speedup	Local score	Matches %	Mismatches %	Gaps %
10K × 10K	98.08	179.62	1.83×	5,091	89.12	9.64	1.24
57K × 57K	0.76	0.80	1.03×	80	92.50	5.00	2.50
162K × 172K	0.83	0.82	0.99×	18	100.00	0.00	0.00
543K × 536K	1.96	2.07	1.06×	48	88.04	11.96	0.00
1M × 1M	1,403.09	2,555.49	1.81×	88,535	79.76	17.12	3.12
3M × 3M	109.69	146.61	1.34×	4,226	83.05	10.46	6.49
5M × 5M	510.59	26,892.05	52.67×	5,220,960	99.95	0.00	0.05
7M × 5M	3.49	4.84	1.39×	172	84.07	12.74	3.19
10M × 10M	898.65	53,563.24	59.60×	10,235,188	99.99	0.01	0.00
23M × 25M	10.15	182.03	17.93×	9,063	99.88	0.05	0.07
47M × 32M	30,425.82	174,147.98	5.72×	27,206,434	94.38	1.54	4.08

Table 1 shows the accession number, the name and approximate size of each sequence.

Table 2 shows the execution times and speedups comparing Fickett-MM with CUDAlign stage 4 (optimized version). It can be seen in this table that, as expected, the best speedups are obtained when the sequences have high similarity, i.e., the local score is close to the size of the smallest sequence (Table 1). For the 5M × 5M and 10M × 10M comparisons, Fickett-MM executed more than 50 times faster than CUDAlign stage 4.

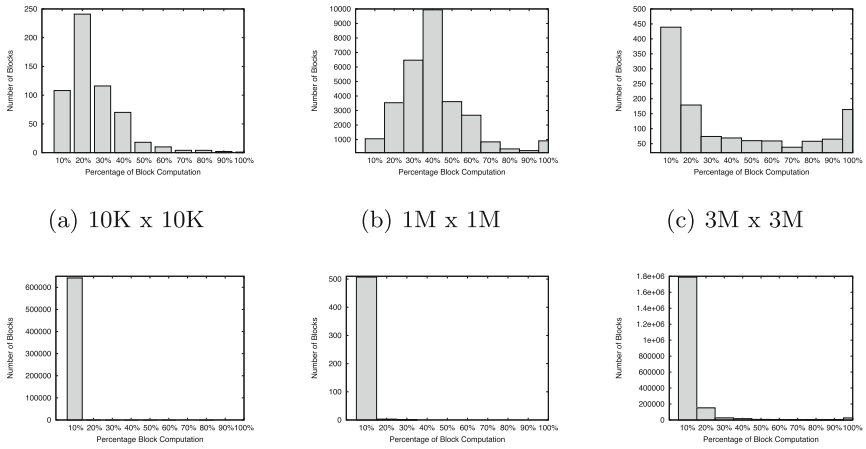
The comparison 23M × 25M obtained a high speedup (17.93×) even though the alignment is not so big. This suggests that, besides the similarity between the sequences, the shape of the alignment has a high influence over the speedups, as shown in the columns 5 to 8 in Table 2. It can be seen that alignments which have a high percentage of matches (>99%) have impressive speedups, with the exception of very small alignments (e.g. 162k × 172k comparison).

Table 3. Number of blocks vs. size of the band

Comparison	Band size (%)										Number of blocks
	0–10	10–20	20–30	30–40	40–50	50–60	60–70	70–80	80–90	90–100	
10k × 10k	108	**241**	116	70	18	10	4	4	2	1	574
57k × 57k	0	**2**	1	0	0	0	0	0	0	0	3
162k × 172k	0	0	0	0	0	0	0	0	0	0	0
543k × 536k	0	1	**2**	0	0	0	0	0	0	0	3
1M × 1M	1056	3538	6469	**9940**	3615	2677	841	357	241	913	29647
3M × 3M	**439**	179	74	69	60	59	38	58	65	164	1205
5M × 5M	**323193**	119	28	25	17	8	8	9	9	52	323468
7M × 5M	0	5	**14**	4	4	2	1	1	0	0	33
10M × 10M	**642225**	76	6	7	3	6	1	0	3	0	642327
23M × 25M	**507**	3	1	0	0	0	0	0	0	0	511
47M × 32M	**1788870**	151556	25926	16259	6943	5670	4212	2724	2734	25079	2029973

A detailed analysis of the alignment's shapes is presented in Table 3 and Fig. 8, showing the number of blocks that were processed with a given band size. The band sizes are given in percentage, calculated as the absolute size of the band divided by the size of the subsequence. For instance, if the size of the subsequence is 100 nucleotides and the size of the band is 12, the percentage is 12% and this block is counted in column "10–20%".

It can be seen that the comparisons in which Fickett-MM achieved impressive speedups (5M × 5M, 10M × 10M and 23M × 25M) only processed less than 10% of their blocks. The comparison 47M × 32M achieved a very good speedup (5.72×) but not as impressive as the three comparisons previously cited because of some blocks in which the size of the band is big. The 543K × 536K comparison



(a) 10K x 10K (b) 1M x 1M (c) 3M x 3M

(d) 10M x 10M (e) 23M x 25M (f) 47M x 32M

Fig. 8. Percentage of block computation for 6 comparisons

is a very interesting case in which the alignment is so small (size = 18) that it does not fill one entire block.

6 Conclusion

In this paper, we proposed and evaluated Fickett-MM, a strategy that retrieves the optimal alignment between two biological sequences using multiple adjustable Fickett bands in linear space. In order to compute the size of each band, we proposed a formula that uses the best score computed so far in special rows/columns, guaranteeing that the optimal alignment will be encompassed by the band. The computation of the adjustable bands was integrated to CUDAlign stage 4, a modified and parallel version of Myers-Miller, which retrieves optimal alignments in linear space.

The results obtained with sequence comparisons whose sizes ranged from $10\text{K} \times 10\text{K}$ to $47\text{M} \times 32\text{M}$ show that Fickett-MM is able to attain impressive speedups when the alignment is huge and the sequences are very similar. In the $10\text{M} \times 10\text{M}$ comparison, the execution time was reduced from 53.5 s to 0.89 s.

As future work, we intend to port Fickett-MM to GPUs (CUDA and OpenCL). Also, we intend to adapt Fickett-MM to retrieve global and semi-global alignments, integrating it to the MASA tool [11].

References

1. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press, Cambridge (1999)
2. Fickett, J.W.: Fast optimal alignments. Nucleic Acids Res. **11**, 175–179 (1984)
3. Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. **162**(3), 705–708 (1982)
4. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Commun. ACM **18**(6), 341–343 (1975)
5. Liu, Y., Tam, T., Lauenroth, F., Schmidt, B.: SWAPHI-LS: Smith-Waterman algorithm on Xeon Phi coprocessors for long DNA sequences. In: IEEE International Conference on Cluster Computing, pp. 257–265 (2014)
6. Liu, Y., Wirawan, A., Schmidt, B.: CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. BMC Bioinformatics **14**, 117 (2013)
7. Maleki, S., Musuvathi, M., Mytcowicz, T.: Parallelizing dynamic programming through rank convergence. In: 19th ACM PPoPP, pp. 219–232 (2014)
8. Myers, E.W., Miller, W.: Optimal alignments in linear space. Comput. Appl. Biosci. **4**(1), 11–17 (1988)
9. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**(3), 443–453 (1970)
10. de Oliveira Sandes, E.F., Miranda, G., Martorell, X., Ayguade, E., Teodoro, G., de Melo, A.C.M.: CUDAlign 4.0: incremental speculative traceback for exact chromosome-wide alignment in GPU clusters. IEEE Tran. Parallel Dist. Syst. **27**(10), 2838–2850 (2016)

11. de Oliveira Sandes, E.F., Miranda, G., Martorell, X., Ayguade, E., Teodoro, G., de Melo, A.C.M.: MASA: a multiplatform architecture for sequence aligners with block pruning. ACM Trans. Parallel Comput. **2**(4), 28 (2016)
12. Rajko, S., Aluru, S.: Space and time optimal parallel sequence alignments. IEEE Trans. Parallel Distrib. Syst. **15**(12), 1070–1081 (2004)
13. Sarkar, S., Kulkarni, G.R., Pande, P.P., Kalyanaraman, A.: Network-on-chip hardware accelerators for biological sequence alignment. IEEE Trans. Comput. **59**(1), 29–41 (2010)
14. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. **147**(1), 195–197 (1981)
15. Wang, L., Chan, Y., Duan, X., Lan, H., Meng, X., Liu, W.: XSW: accelerating biological database search on Xeon Phi. In: IEEE AsHES, pp. 950–957 (2014)
16. Wienbrandt, L.: The FPGA-based high-performance computer RIVYERA for applications in bioinformatics. In: Beckmann, A., Csuhaj-Varjú, E., Meer, K. (eds.) CiE 2014. LNCS, vol. 8493, pp. 383–392. Springer, Cham (2014). doi:10.1007/978-3-319-08019-2_40