Cluster Comput (2007) 10: 187–202
DOI 10.1007/s10586-007-0020-0

An exact parallel algorithm to compare very long biological sequences in clusters of workstations

Azzedine Boukerche · Alba Cristina Magalhaes Alves de Melo · Edans Flavius de Oliveira Sandes · Mauricio Ayala-Rincon

Published online: 5 April 2007

Abstract Biological Sequence Comparison is one of the most important operations in Computational Biology since it is used to determine how similar two sequences are. Smith and Waterman proposed an exact algorithm (SW), based on dynamic programming, that is able to obtain the best local alignment between two sequences in quadratic time and space.

In order to compare long biological sequences, SW is rarely used since the computation time and the amount of memory required becomes prohibitive. For this reason, heuristic methods like BLAST are widely used. Although faster, these heuristic methods do not guarantee that the best result will be produced.

In this paper, we propose an exact parallel variant of the SW algorithm that obtains the best local alignments in quadratic time and reduced space. The results obtained in two clusters (8-machine and 16-machine) for DNA sequences longer than 32 KBP (kilo base-pairs) were very close to linear and, in some cases, superlinear. For very long DNA sequences (1.6 MBP), we were able to reduce execution time from 12.25 hours to 1.54 hours, in our 8-machine cluster. As far as we know, this is the first time 1.6 MBP sequences are compared with an exact SW variant. In this case, 30240 best local alignments were obtained.

Keywords Biological sequence comparison · Parallel algorithm

A. Boukerche (✉)
SITE, University of Ottawa, Ottawa, Canada
e-mail: boukerch@site.uottawa.ca

A.C.M.A. de Melo · E.F.O. Sandes · M. Ayala-Rincon
University of Brasilia, Brasilia, Brazil

1 Introduction

Biological sequence comparison, also called sequence alignment, is one of the most basic and important operations in Computational Biology, being widely used to determine how similar two organisms are. Usually, when a new organism is discovered, an automated sequencer is used to obtain the sequences of nucleotides or proteins that compose it. Having this new genetic sequence, the biologists must infer its functionalities. This is done by comparing the new genetic sequence with biological sequences for which the functionalities have already been established. A similarity between the sequences usually indicates common functionalities.

Given the huge size of sequence data produced by molecular genomics nowadays, sequence comparison is becoming a critical operation in genome projects [3]. This occurs because a great computational power is needed to perform thousands of comparisons daily against the genetic databases. Besides that, each day, more sequences are discovered and included in those genetic databases, that have presented an exponential growth rate in the last years. GenBank [20] from NCBI (National Center for Biotechnology Information) [19] is one of the most widely used publicly available genetic databases.

Biological sequence comparison is in fact a problem of approximate pattern matching between two biological sequences [18]. The most important types of biological sequence alignments are global and local. To solve a global alignment problem is to find the best match between the entire sequences. On the other hand, local alignment searches the best match between parts of the sequences.

In the literature, many algorithms were proposed to solve the sequence comparison problem. The algorithm NW [21] is an exact method based on dynamic programming that finds the best global alignment between two genomic sequences. The algorithm SW [27], derived from NW, obtains the best local alignment between two sequences. Given two sequences s and t, time and space complexity for both algorithms is $O(n^2)$, where n is the size of the sequences. For this reason, these exact methods are rarely used in genome projects. To obtain good alignments in reasonable time and/or space, heuristic methods such as BLAST [1] and FASTA [22] were proposed. Although there is no guarantee that the best alignment will be produced, these heuristic methods are widely used in genome projects since they are able to obtain good local alignments many times faster than SW.

An interesting alternative to accelerate the execution of the exact algorithms is parallel processing. In the literature, we can find exact parallel variants of NW [9, 24] as well as exact and heuristic parallel variants of SW [4, 7, 16, 30]. All of these algorithms return either one best alignment or a set of best (or good) alignments. The choice to obtain only a small set of alignments is based on the fact that the number of best alignments between two sequences can grow exponentially with the size of the alignment, making it a complex problem. Biologists, however, usually need to get many (or even all) best alignments in order to further analyze them manually.

In this paper, we propose and evaluate a new exact parallel variant of SW, which is able to obtain all the best local alignments between two very long DNA sequences in reduced space. The proposed parallel algorithm has two phases. In the first one, the similarity matrix used in the SW algorithm is fully calculated by all processors that compose the parallel system but only the maximum score(s) and its (their) coordinates are stored. The wavefront method [23] is used to distribute the work in this phase. In the second phase, part of the similarity matrix is re-calculated over the reverses of the biological sequences and the best local alignment(s) are retrieved. The master/slave model is used to distribute work in a *Self-scheduling* basis [28] in this phase. The first phase of the algorithm is executed in linear space on the length of the sequences and the second phase executes on quadratic space on the length of the best local alignments (on average, the size of the alignments is much smaller than the size of the sequences).

Our algorithm detects the coordinates of the best local alignments in quadratic time and linear space. After that, each of these alignments are generated in quadratic time and space on their size. This generation is done reducing the use of space towards linear since these best alignments will typically follow the diagonal.

The results obtained in a 8-machine cluster for long DNA sequences (greater than or equal to 32 kilo base-pairs (KBP)) presented speedups close to linear and superlinear speedups in some cases (128 KBP, 256 KBP and 512 KBP). To compare 1.6 MBP sequences, we were able to reduce execution time from 12.25 hours to 1.54 hours, when 8 processors were used. As far as we know, this is the first time 1.6 MBP sequences are compared with an exact variant of the SW algorithm. We also run our tests in a 16-machine cluster and obtained speedups close to linear for 256 KBP and 800 KBP sequences (14.98 and 15.63, respectively).

The remainder of this paper is organized as follows. Section 2 presents the biological sequence alignment problem and the basic algorithms to solve it. Section 3 discusses related work on serial and parallel variants of the basic algorithms to compare biological sequences. Section 4 presents our exact algorithm to obtain the best local alignments between two genomic sequences. In Sect. 5, we describe the parallelization strategy used in the algorithm proposed in Sect. 4. Experimental results are discussed in Sect. 6. Finally, Sect. 7 concludes the paper and presents future work.

2 Biological sequence comparison

Biological sequences are either DNA or protein sequences. In the first case, we consider ordered sequences of nucleotides (A, T, G, C) and, in the second case, we deal with ordered sequences of proteins. In this paper, only DNA sequence comparison is discussed.

DNA biological sequences are treated by the algorithms as strings composed by elements of the alphabet $\Sigma = \{A, T, G, C\}$. Since two DNA sequences are rarely identical, biological sequence comparison (or sequence alignment) is in fact a problem of approximate pattern matching [18].

2.1 Similarity score

To compare two sequences, we need to find the best alignment between them, which is to place one sequence above the other making clear the correspondence between similar characters [25]. In an alignment, spaces can be inserted in arbitrary locations along the sequences so that they end up with the same size.

In order to measure the similarity between two sequences, a similarity score is calculated as follows. Given an alignment between sequences s and t, the following values are assigned for each column: a) $+1$, if both characters are identical (*match*); b) -1, if the characters are not identical (*mismatch*); and c) -2, if one of the characters is a space (*gap*). The score is the sum of all these values. Note that the punctuation assigned to matches, mismatches and gaps is only one example of a possible punctuation. Figure 1 presents one possible alignment between two DNA sequences and its associated score.

$$\begin{array}{cccccccccccccc}
A & C & T & T & G & T & C & C & G & - & A & G & A \\
A & - & T & T & G & T & C & A & G & G & A & G & G \\
\hline
\end{array}$$
$$\underbrace{+1\ -2\ +1\ +1\ +1\ +1\ +1\ -1\ +1\ -2\ +1\ +1\ -1}_{score\,=\,3}$$

Fig. 1 Example of alignment and score

2.2 Algorithm NW for global alignment

The algorithm proposed by Needleman and Wunsh (NW) [21] is an exact method based on dynamic programming to obtain the best global alignment between two sequences in quadratic time and space.[1] It is divided in two phases: create the similarity matrix and obtain the best global alignment.

2.2.1 Create the similarity matrix

This phase of the algorithm receives as input sequences s and t, with $|s| = m$ and $|t| = n$, where $|s|$ represents the size of sequence s. For sequences s and t, there are $m + 1$ and $n + 1$ possible prefixes, respectively, including the empty sequence. The notation used to represent the *n-th* character of a sequence seq is $seq[n]$ and, to represent a prefix with n characters, from the beginning of the sequence, we use $seq[1..n]$.

The similarity matrix is denoted $A_{m+1,n+1}$, where $A_{i,j}$ contains the similarity score between prefixes $s[1..i]$ and $t[1..j]$.

At the beginning, the first row and column are filled with the values $-2x$, where x is the size of the non-empty sequence and -2 is the gap penalty. This represents the cost of aligning a non-empty subsequence with an empty one. In other words, the first row and column of the similarity matrix are initialized with values $A_{0,j} = -2j$ and $A_{i,0} = -2i$. Note that $A_{0,0} = 0$. The remaining elements of A are obtained from (1). The total score between sequences s and t is the value contained in cell $A_{m+1,n+1}$.

$$A_{i,j} = \max \begin{cases} A_{i-1,j-1} + (\text{if } s[i] = s[j] \text{ then } +1 \text{ else } -1) \\ A_{i,j-1} - 2 \\ A_{i-1,j} - 2. \end{cases} \tag{1}$$

Figure 2 presents the similarity matrix between sequences s = ATTGTCAGGAGG and t = ACTTGTCCGAGA. The arrows indicate the cell from where the value was obtained, according to (1).

[1]In fact, the algorithm originally proposed by the authors had cubic time and space complexity [10]. It was then modified to generate global alignments in quadratic time and space. In this article, we refer to this last version of the NW algorithm.

2.2.2 Obtain the best global alignment

In order to obtain the best global alignment, the algorithm starts from cell $A_{m+1,n+1}$ and follows the arrows until cell $A_{0,0}$ is reached. A left arrow in $A_{i,j}$ (Fig. 2) indicates the alignment of $s[i]$ with a gap in t. An up arrow represents the alignment of $t[j]$ with a gap in s. Finally, an arrow on the diagonal indicates that $s[i]$ is aligned with $t[j]$.

Note that many best global alignments can exist, since many arrows can exist in the same cell $A_{i,j}$, indicating that the score value was produced from more than one cell (Fig. 2). The space complexity of this phase is $O(\{min(m,n) + n'^2\} * k)$, where n' is the size of the longest alignment and k is the total number of alignments.

2.3 Algorithm SW for local alignment

To obtain the similarity between parts of the sequences, local alignment must be used. The exact algorithm that is usually used in this case is the one proposed by Smith and Waterman (SW) [27]. Like NW (Sect. 2.2), SW is also based in dynamic programming with quadratic time and space complexity. However, there are two basic differences between them, concerning the calculation of the similarity matrix (Sect. 2.2.1).

The first difference is on the initialization of the first row and column, which are filled with zeros in SW. In this way, gaps do not receive penalty if they are at the beginning of the alignment.

The second difference involves the equation used to calculate the remaining cells since, in SW, no negative values are allowed. In order to do that, the value zero is included in (1), generating (2).

$$A_{i,j} = \max \begin{cases} A_{i-1,j-1} + (\text{if } s[i] = s[j] \text{ then } +1 \text{ else } -1) \\ A_{i,j-1} - 2 \\ A_{i-1,j} - 2 \\ 0. \end{cases} \tag{2}$$

To obtain the best local alignment, the algorithm starts from the cell which has the highest value, following the arrows until the value zero is reached.

Figure 3 presents the similarity matrix to obtain local alignments between sequences s = TATAGGTAGCTA and t = GAGCTATGAGGT. Note that, in this example, two best alignments can be obtained, both of them with score 5.

3 Related work

3.1 Serial variations of the basic algorithms

Due to the rapid growth in the size of the genetic databases and the constant increase on the sizes of the sequences to

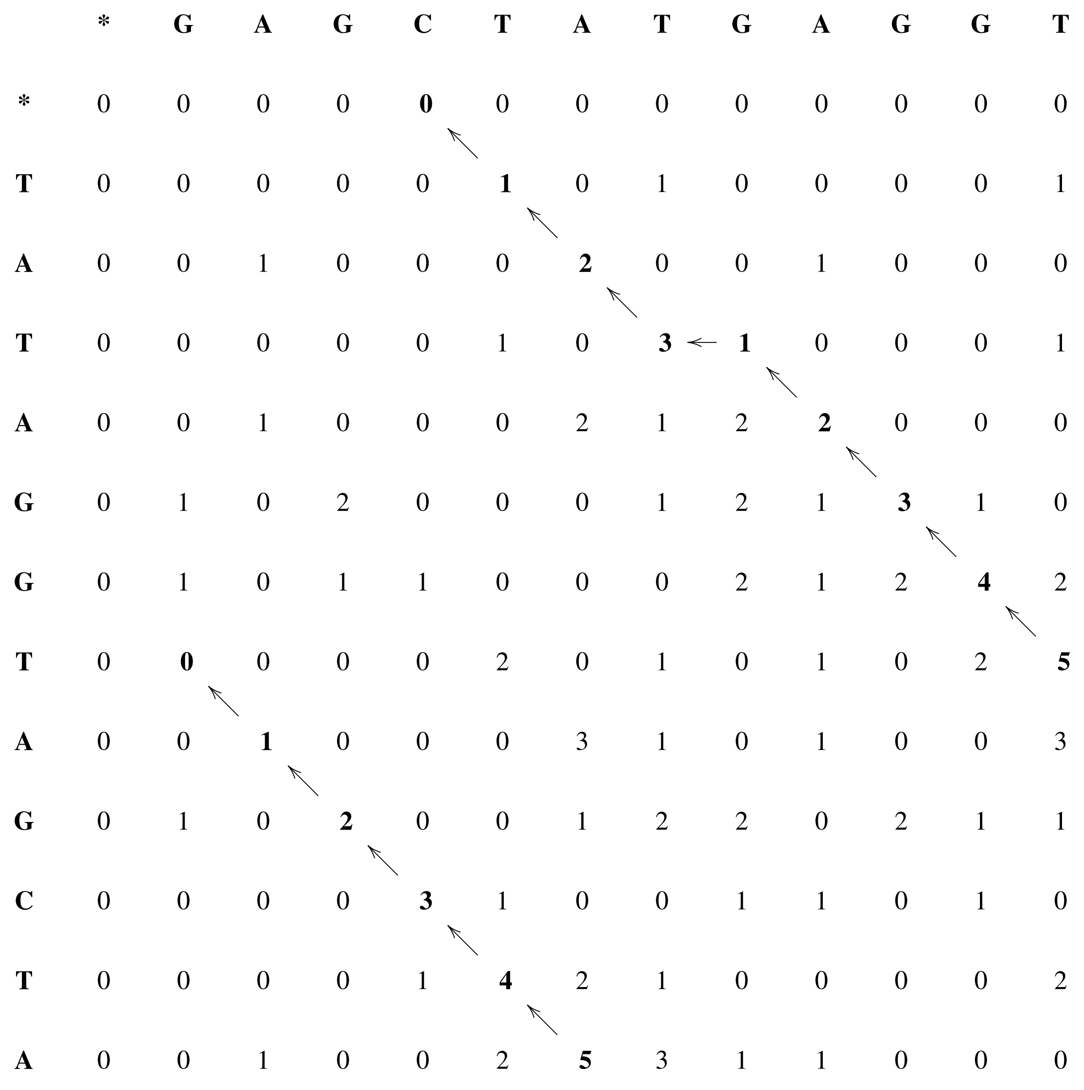
	*	A	C	T	T	G	T	C	C	G	A	G	A
*	**0**	−2	−4	−6	−8	−10	−12	−14	−16	−18	−20	−22	−24
A	−2	↖ **1**	← **-1**	−3	−5	−7	−9	−11	−13	−15	−17	−19	−21
T	−4	−1	0	↖ **0**	−2	−4	−6	−8	−10	−12	−14	−16	−18
T	−6	−3	−2	1	↖ **1**	−1	−3	−5	−7	−9	−11	−13	−15
G	−8	−5	−4	−1	0	↖ **2**	0	−2	−4	−6	−8	−10	−12
T	−10	−7	−6	−3	0	0	↖ **3**	1	−1	−3	−5	−7	−9
C	−12	−9	−6	−5	−2	−1	1	↖ **4**	2	0	−2	−4	−6
A	−14	−11	−8	−7	−4	−3	−1	↑ **2**	↖ **3**	1	1	−1	−3
G	−16	−13	−10	−9	−6	−3	−3	0	↖ ↑ **1**	↖ **4**	2	2	0
G	−18	−15	−12	−11	−8	−5	−4	−2	−1	↖ ↑ **2**	3	3	1
A	−20	−17	−14	−13	−10	−7	−6	−4	−3	0	↖ **3**	2	4
G	−22	−19	−16	−15	−12	−9	−8	−6	−5	−2	1	↖ **4**	2
G	−24	−21	−18	−17	−14	−11	−10	−8	−7	−4	−1	2	↖ **3**

Fig. 2 Similarity matrix for global alignment between sequences s and t

be compared, the use of algorithms such as NW and SW is becoming prohibitive. For this reason, many variations on these basic algorithms have been proposed in the literature that try to reduce computation time and/or the amount of memory needed. It is not known an exact serial algorithm with the same generality that executes in less asymptotic time than SW. Nevertheless, there are faster exact algorithms for particular cases.

Hirschberg [11] proposed an exact algorithm that calculates a global alignment between two sequences s and t in quadratic time but in linear space. The approach used splits sequence s in the middle, generating subsequences $s1$ and $s2$, and calculates the corresponding place to cut sequence t, generating subsequences $t1$ and $t2$, in such a way that the alignment problem can be solved in a divide and conquer recursive manner. This recursion roughly doubles the execution time, when compared with the original algorithm. Nevertheless, for long biological sequences, which would otherwise generate very huge similarity matrices, this approach can be appropriate.

The method proposed by Fickett [8] considers only very similar sequences. In this case, the array A generated by the dynamic programming method is a square and its main diagonal starts at $A_{0,0}$ and ends at $A_{n,n}$, where n is the size of the sequences. To follow the main diagonal is to align both sequences without gaps. As long as gaps are introduced, the alignment leaves the main diagonal. If the sequences are very similar, the alignment between them is near the main diagonal. Thus, in order to compute the best global alignment(s), it is sufficient to fill only a small band (k-band) near the main diagonal. If k is the distance between the array element being computed and the element that composes the main diagonal on the same row, the algorithm has time and space complexity $O(kn)$.



	*	G	A	G	C	T	A	T	G	A	G	G	T
*	0	0	0	0	**0**	0	0	0	0	0	0	0	0
T	0	0	0	0	0	**1**	0	1	0	0	0	0	1
A	0	0	1	0	0	0	**2**	0	0	1	0	0	0
T	0	0	0	0	0	1	0	**3** ←	**1**	0	0	0	1
A	0	0	1	0	0	0	2	1	2	**2**	0	0	0
G	0	1	0	2	0	0	0	1	2	1	**3**	1	0
G	0	1	0	1	1	0	0	0	2	1	2	**4**	2
T	0	**0**	0	0	0	2	0	1	0	1	0	2	**5**
A	0	0	**1**	0	0	0	3	1	0	1	0	0	3
G	0	1	0	**2**	0	0	1	2	2	0	2	1	1
C	0	0	0	0	**3**	1	0	0	1	1	0	1	0
T	0	0	0	0	1	**4**	2	1	0	0	0	0	2
A	0	0	1	0	0	2	**5**	3	1	1	0	0	0

Fig. 3 Similarity matrix for local alignment between sequences *s* and *t*

Other algorithms that achieve improvements over the basic algorithms are the ones proposed by Ukkonen [29], Myers [17], Chang and Lawler [6] and Landau and Viskin [14].

3.2 Parallel variations of the basic algorithms

In the NW and SW algorithms, most of the time is spent calculating the similarity matrix between two sequences and this is the part which is usually parallelized. The access pattern presented by the matrix calculation is non-uniform and has been extensively studied in the parallel programming literature. The parallelization strategy that is traditionally used in this kind of problem is known as the wavefront method [23] since the calculations that can be done in parallel evolve as waves on diagonals.

Figure 4 illustrates the wavefront method. At the beginning of the computation, only one node can compute value

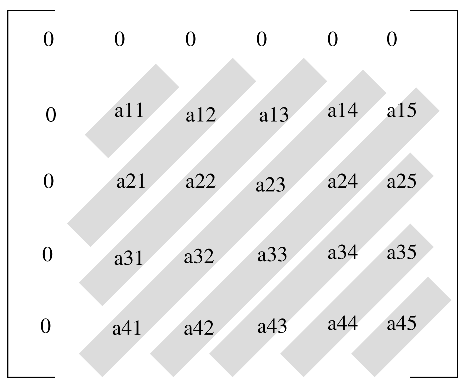


Fig. 4 The wavefront method used to exploit the parallelism presented by the SW algorithm

$A_{1,1}$. After that, values $A_{2,1}$ and $A_{1,2}$ can be computed in parallel, then, $A_{3,1}$, $A_{2,2}$ and $A_{1,3}$ can be computed independently, and so on. The maximum parallelism is attained at the matrix main anti-diagonal and then decreases again.

In the following paragraphs, most of the strategies discussed use the wavefront method. Also, assume sequences s and t with lengths m and n, respectively.

Galper and Brutlag [9] proposed a parallel SW algorithm that uses the wavefront method in a row-basis or in a diagonal-basis. Each processor communicates with two neighbors and computations evolve as waves on the similarity matrix diagonals. Synchronous and asynchronous modes are provided. Also, a mode called *windowed asynchronous diagonal* is provided for space saving on the similarity matrix calculation. The size of the window depends on the cost functions for insertion, deletion and substitution [18] and is the maximum cost of converting the query sequence s on sequence t. This algorithm was implemented in C with parallel macro extensions. Experimental results were collected in a 12-processor Encore Multimax shared memory machine. For sequences with lengths $m = 118$ and $n = 14908$, a speedup of 10.29 was achieved for 12 processors.

Boukerche et al. [5] proposed a parallel heuristic Smith-Waterman variant that reduces the size of the similarity matrix to $2m$ and, thus, only the row being calculated and the previous one are kept in memory. In order to do that, an additional data structure is used to keep the potential alignments, which are the ones whose score is above a given threshold. Work is assigned to the p processors in a row basis and the wavefront method is used. Each processor acts on a set of columns inside the row and communicates with 2 neighbors through a lazy synchronization protocol. This approach was implemented in C using the JIAJIA Distributed Shared Memory system [12]. The results were collected in a 8-processor cluster and real DNA sequences were used. The sequences tested were of approximately the same length and the lengths analyzed were 15 KBP, 50 KBP, 80 KBP, 150 KBP and 400 KBP. A speedup of 4.58 was achieved in the 400 KBP sequence comparison with 8 processors. A variant of this approach which uses blocking factors is proposed in [2]. This variant was implemented in JIAJIA and MPI (Message Passing Interface) and a speedup of 7.28 was achieved in the comparison of 50 KBP sequences with 8 processors.

Zhang, Quiao and Liu [30] proposed PSW-DC, a parallel heuristic Smith–Waterman variant that uses the divide and conquer method to reduce the size of the similarity matrix to $(mn)/p$, where p is the number of processors. In this approach, the similarity matrix is spread over the parallel system. The query sequence s is divided in such a way that each processor i receives only a subsequence s'_{pi} whose length is m/p and the whole sequence t, with length n. The SW algorithm is executed for these data, with no communication between the processors, producing p optimal local alignments $Alig_i$ between the subsequences s'_{pi} and t. In this case, the wavefront method is not used. An heuristic method called *Combine and Extend (C&E)* is proposed to generate the local alignments between s and t from the p similarity matrices between t and s'_i. The PSW-DC algorithm was implemented in C using MPI. The results were collected in a 16-processor cluster and indicate that, as long as processors are added, the sensitivity is reduced. The test sequences were artificially produced and had the following lengths: 1 KBP, 2 KBP, 4 KBP, 8 KBP and 16 KBP.

Martins et al. [15] proposed a multithreaded parallel variation of the NW algorithm that uses the wavefront method in its two-level parallelism (threads and processes). The algorithm retrieves the start and end coordinates of the alignments, but not the alignments themselves. To reduce the memory space needed, only two rows of the similarity matrix are actually stored in memory (the row that is being calculated and the previous one). The results presented in a 64-node cluster, each one with two processors, show good speedups for sequences longer than 100 KBP. Sequences with the following lenghts were compared: 30 KBP, 50 KBP, 100 KBP, 300 KBP and 900 KBP.

Rajko and Aluru [24] proposed an exact parallel algorithm that solves the global sequence alignment problem in $O(mn/p)$ time and requires $O((m+n)/p)$ space. The sequence alignment algorithm is extended to treat the affine gap penalty function [10]. Thus, the extended alignment problem is $(A, B, start\_type, end\_type)$, where A and B are sequences and *start_type* and *end_type* are in $-3, -2, -1, 1, 2, 3$, where types $1, 2$ and 3 correspond to a_i matched to b_j, a_i matched to a gap and b_j matched to a gap, respectively. Besides that, in the case of negative values, the original scores of the alignments can be modified to enforce a particular type of match. The scores for the extended alignment problem are calculated by adding the score obtained from the standard alignment problem with the modified score.

The key idea of this algorithm is to add an additional phase to find a partial balanced partition between subsequences of A and B. The partial balanced partition allows the subdivision of the original problem in independent subproblems, that can be solved in parallel [24]. To compute the partial balanced partition in parallel, three dynamic matrices are used to find the cells where a recursive decomposition of the problem can be performed. A partial balanced partition starts at the intersection of an optimal path with a diagonal. If the intersection is in a_i, b_j, then the problem can be decomposed in two independent subproblems: the alignment of $a_1, a_2, \dots, a_i$ with $b_1, b_2, \dots, b_j$ and the alignment of $a_{i+1}, a_{i+2}, \dots, a_m$ with $b_{j+1}, b_{j+2}, \dots, b_n$. This decomposition can be done recursively. An optimal alignment C between sequences A and B is proved to be the concatenation of optimal alignments to the subproblems corresponding to each region [24]. Note that this algorithm finds only one optimal global alignment. The authors state that this algorithm can be easily modified to solve the local alignment problem.

The proposed algorithm was implemented in C++ and MPI and executed on a IBM xSeries cluster with 60 machines. Only the complete match (where both sequences are equal) and complete mismatch (where both sequences are entirely different) cases were analyzed. Sequence sizes of 40 KBP, 80 KBP, 160 KBP and 320 KBP were considered. The results obtained show good speedups for both the complete match and the complete mismatch case for 60 processors when the sequence size is big enough (320 KBP). The authors also show a test for 1.1 MBP sequences in 60 processors. The complete match took around 2.58 hours in 60 processors, where 2.55 hours were spent in the first phase (computation of the partial balanced partition).

Chen and Schmidt [7] propose an exact parallel variation of a linear space algorithm based on SW that finds a set of local alignments that are close to the best (near-best) [13]. The algorithm has three phases. In the first phase, the end coordinated of the best (and near best) alignments is found. In the second phase, the coordinates of the beginning of the alignments are found over the reversed sequences. In the second phase, having the beginning and the end coordinates of each non-overlapping alignment (best and near-best), the Hirschberg algorithm (Sect. 3.1) is used to retrieve the actual alignments. The results were obtained in two clusters: a 14-machine (Pentium III 733 MHz, Myrinet switch) and a 8-machine (Itanium-1 733 MHz, Myrinet switch) interconnected by a 30 Mb/s campus-area switch. The execution time obtained to compare 100 KBP sequences using both clusters (20 machines) was 275 seconds. A qualitative analysis is done in the comparison of two sequences of 600 KBP and 800 KBP, respectively. No results in execution times are shown for this last experiment.

3.3 Discussion

In Sects. 3.1 and 3.2, some serial and parallel variants of the NW and SW algorithms were presented. Most of the parallel variants are heuristic ([2, 5, 30]), retrieving the local alignments with scores above a given threshold. For these three papers, the maximum sequence size considered was 400 KBP.

Martins et al. [15] proposed an exact parallel variant of NW that uses the wavefront method to retrieve the start and end coordinates of the best global alignments. The longest pair of sequence analyzed had 900 KBP.

The algorithm proposed by [9] is exact and obtains one best local alignment between two sequences. The maximum sequence size analyzed was 14 KBP.

The algorithm proposed by [24] is an exact variation which is able to obtain one global alignment between two sequences in optimal space and time, which is a great result. However, the execution times obtained in the comparison of 1.1 MBP sequences with 60 machines for simple cases are very high (2.58 hours for complete match and 1.28 hours for the complete mismatch). Besides that, only one best global alignment is retrieved.

The parallelization strategy employed in the algorithm proposed by [7] is very similar to ours. However, there are two main differences: the use of Hirschberg in the third phase and the obtention of the near-best non-overlapping alignments. We decided not to use Hirschberg since the increase in computation time by using it is considerable. Also, we opted to possibly obtain all best alignments instead of only the non-overlapping ones.

None of the algorithms analyzed here is able to retrieve all best local alignments and they do not compare sequences longer than 1.1 MBP.

4 A new variant of the SW algorithm

The algorithm that we propose in this section executes in space $O(\min(m,n)+n'^2)$, where m and n are the sizes of sequences s and t, respectively, and n' is the maximum size of the biggest best local alignment between the sequences. Since, in general, $n' \ll n$ and $n' \ll m$, there is a considerable reduction on the amount of memory used, in the average case.

The main idea used in our algorithm is the fact that, having determined a cell $A_{x,y}$ where a best local alignment ends, there is a local alignment with the same score in cell $A^{rev}_{n-x+1,m-y+1}$ in the similarity matrix A^{rev}, which is generated by aligning sequences s^{rev} and t^{rev}, which are the reverse sequences of s and t, respectively. The actual best local alignments are then obtained by finding the global alignment between $s[1..x]^{rev}$ and $t[1..y]^{rev}$. The same idea was used in [7].

Algorithm 1 illustrates the proposed algorithm. Input sequences are seq_0 and seq_1. Note that phase 2 is divided in

Algorithm 1 $align(seq_0, seq_1)$

```
1: $(best\_score, coords) \leftarrow parallel\_smith\_waterman\_phase1(seq_0, seq_1)$
2: for all $coord \in coords$ do
3:     $(min_x, min_y) \leftarrow smith\_waterman\_phase2\_step1(seq_0, seq_1, best\_score, coord_x, coord_y)$
4:     $alignments \leftarrow smith\_waterman\_phase2\_step2(seq_0, seq_1, best\_score, coord_x, coord_y, min_x, min_y)$
5: end for
```

Fig. 5 Similarity matrix calculated at phase 1 of our algorithm

	*****	**A**	**T**	**C**	**C**	**G**	**A**	**A**	**G**	**T**	**T**	**G**	**A**	**C**	**C**	**T**	**T**	**C**	**A**	**G**	**T**	**T**	**G**	**A**	**G**
*****	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1
G	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1
T	0	0	1	0	0	0	0	0	0	2	1	0	0	0	0	1	1	0	0	0	2	1	0	0	0
A	0	1	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	1	0	0	1	0	1	0
G	0	0	0	0	0	1	0	0	2	0	0	2	0	0	0	0	0	0	0	2	0	0	2	0	2
G	0	0	0	0	0	1	0	0	1	1	0	1	1	0	0	0	0	0	0	1	1	0	1	1	1
G	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	2
G	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1
C	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0
T	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	2	1	0	0	0	1	1	0	0	0
C	0	0	0	2	1	0	0	0	0	0	0	0	0	1	1	0	1	2	0	0	0	0	0	0	0
C	0	0	0	1	3	1	0	0	0	0	0	0	0	1	2	0	0	2	1	0	0	0	0	0	0
G	0	0	0	0	1	4	2	0	1	0	0	1	0	0	0	1	0	0	1	2	0	0	1	0	1
A	0	1	0	0	0	2	5	3	1	0	0	0	2	0	0	0	0	0	1	0	1	0	0	2	0
G	0	0	0	0	0	1	3	4	4	2	0	1	0	1	0	0	0	0	0	2	0	0	1	0	3
T	0	0	1	0	0	0	1	2	3	5	3	1	0	0	0	1	1	0	0	0	3	1	0	0	1
G	0	0	0	0	0	1	0	0	3	3	4	4	2	0	0	0	0	0	0	1	1	2	2	0	1
A	0	1	0	0	0	0	2	1	1	2	2	3	5	3	1	0	0	0	1	0	0	0	1	3	1
C	0	0	0	1	1	0	0	1	0	0	1	1	3	**6**	4	2	0	1	0	0	0	0	0	1	2
G	0	0	0	0	0	2	0	0	2	0	0	2	1	4	5	3	1	0	0	1	0	0	1	0	2
G	0	0	0	0	0	1	1	0	1	1	0	1	1	2	3	4	2	0	0	1	0	0	1	0	1
G	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	2	3	1	0	1	0	0	1	0	1
C	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	1	4	2	0	0	0	0	0	0

steps 1 and 2, in order to reduce computation time. In step 1, the beginning of the alignments are retrieved. In step 2, having the beginning and the end coordinates of the alignments, the actual best local alignments are obtained. In step 2, unlike phase 3 in [7], we did not use Hirschberg (Sect. 3.1), but developed a new approach, based on the algorithm proposed by Fickett (Sect. 3.1), that is able to cut considerably the number of cells calculated.

The phases of the proposed algorithm are detailed in the following sections.

4.1 Phase 1—obtaining the coordinates of the best score

In this phase, the similarity matrix is fully calculated as shown in Sect. 2. However, only the coordinates and the value of the best score are stored. Since the similarity matrix does not need to be fully stored, the amount of memory used in this phase corresponds to two consecutive rows from A. This is due to the fact that the computation of a cell $A_{i,j}$ depends on the previous column and row (cf. (2)). Therefore, space complexity for this phase is $O(n)$.

While the similarity matrix is calculated, the algorithm maintains the highest score *best_score* obtained so far, as well as the coordinates (x, y) where it can be found.

Figure 5 presents the similarity matrix calculated in phase 1 to locally align sequences $s =$ GGTAGGGGCTCCGAGTGACGGGC and $t =$ ATCCGAAGTTGACCTTCAGTTGAG. As output, phase 1 produces the maximum score (in the example it is 6) and its coordinates ($x = 19$, $y = 13$, in the example).

4.2 Phase 2—obtaining the best local alignments

Having the value and the end coordinates of the best score, this phase executes a global alignment on sequences $s[1..x]^{rev}$ and $t[1..y]^{rev}$ until the scores *best_score* are reached.

As an example, suppose the sequences illustrated in Fig. 5. In this case, we need to align sequences $s[1..19]^{rev} =$ CAGTGAGCCTCGGGGATGG and $t[1..13]^{rev} =$ CAGTTGAAGCCTA. Note that, to obtain all best local alignments, we cannot stop after finding the first score that is equal to *best_score*. The search must continue, in the worst case, until the beginning of the reversed sequences.

Therefore, we propose two strategies to reduce the amount of memory used in this phase.

The first strategy is based on the fact that the maximum score value (*best_score*) is already known, since it was obtained in phase 1. Thus, computations for cells with low values that will not be able to produce *best_score* are discarded.

This can be easily seen by analyzing the number of cells that remain to be calculated. Let *score* be the score of a given cell, δx the number of columns to be processed and δy the number of remaining rows. The maximum score that

	*	**C**	**A**	**G**	**T**	**T**	**G**	**A**	**A**	**G**	**C**	**C**	**T**	**A**
*	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	1	0											
A	0	0	2	0										
G	0		0	3	1	0								
T	0			1	4	2	0							
G	0			1	2	3	3	1	0					
A	0			0	0	1	2	4	2	0				
G	0				0	0	2	2	3	3	1	0		
C	0						0	1	1	2	4	2	0	
C	0							0	0	0	3	5	3	1
T	0										1	3	**6**	
C	0													
G	0													
G	0													
G	0													
G	0													
A	0													
T	0													
G	0													
G	0													

Fig. 6 Similarity matrix calculated in phase 2 of the algorithm

can be generated from this cell is $score + min(\delta x, \delta y)$, since we are using the values for matches, mismatches and gaps presented in Sect. 2.1. If the maximum possible score from this cell is less than *best_score*, this cell can be discarded. Besides that, the zero values that were included in the similarity matrix to forbid negative values to be generated can also be discarded, since they cannot produce an alignment that starts at the beginning of the reversed sequences.

The second strategy adopted to reduce the amount of memory used is to split phase 2 in two steps: (*phase2-step1*), where all coordinates that have the maximum score in the reversed sequences are obtained in linear space. Having both the beginning and the end of each best local alignment, we can define a rectangle that includes all possible best alignments in this area (*phase2-step2*). This reduces the values of δx e δy, allowing more cells to be discarded. This phase is quadratic on the length of the best alignment, but we reduce the area calculated to approximately $1/3$ of this space (Fig. 6).

Using these strategies, Fig. 6 illustrates the cells that are actually calculated.

With the similarity matrix illustrated in Fig. 6, four local alignments are obtained from the reversed sequences s^{rev} e t^{rev} with score 6. After that, the original best local alignments are obtained by reversing the alignments over the reverse sequences. This can be easily done by printing the alignments from the end to the beginning.

To obtain all possible best local alignments, all coordinates that produced the score *best_score* must be stored. The reverse alignment must be executed for each one of these coordinates. Since the example from Fig. 6 has only one *best_score*, the four local alignments cited in the previous paragraphs are the only ones with score 6.

5 Proposed parallelization strategy

In this section, we propose a parallel version of the algorithm described in Sect. 4 where the calculation of the similarity matrix (phase 1) is done in parallel and the retrieval of the best local alignments (phase 2) is distributed among the nodes.

In our parallel strategy, there is a node called master node (M), which is responsible to receive the input sequences (seq_0 and seq_1) to be compared and distribute them among the slave nodes. In our algorithm, the master is typically the front-end of the cluster and it does not participate on the computation. We assume also that there are t slave processors $p_0, \ldots, p_{t-1}$ where i is the index of processor p_i.

Algorithms 2 and 3 are the algorithms executed by the master and the slave nodes, respectively.

5.1 Parallelization of phase 1

In this first phase, the master node reads input sequences seq_0 and seq_1 and broadcasts them to the t slave nodes.

The computation of the similarity matrix is divided in equal parts with size n^2/t, where n is the size of the sequences and t is the number of slave nodes. Each slave P_i

Algorithm 2 Master

```
1: copies input sequences to $seq[0]$ and $seq[1]$
2: broadcasts $seq[0]$ and $seq[1]$ to slaves
3: for all slave do
4:    receives $score'$ and $coords'$ from slave
5:    if $best\_score < score'$ then
6:       $best\_score \leftarrow score'$
7:       $coords \leftarrow coords'$
8:    else
9:       if $best\_score = score'$ then
10:         $coords \leftarrow coords \cup coords'$
11:      end if
12:   end if
13: end for
14: broadcasts $best\_score$ to all slaves
15: for all $coord \in coords$ do
16:    receives demand for work from $slave$
17:    $job \leftarrow coord$
18:    sends $job$ to $slave$
19: end for
20: sends $JOB_{end}$ to all slaves
```

Algorithm 3 Slave

```
1: $comm\_size \leftarrow$ number of slaves
2: $rank \leftarrow$ unique identifier in $[0..comm\_size - 1]$
3: receives $seq_0$ and $seq_1$ from master
4: $column_0 \leftarrow \frac{|seq_0|*my\_rank}{comm\_size} + 1$
5: $column_1 \leftarrow \frac{|seq_0|*(my\_rank+1)}{comm\_size}$
6: $(score', coords') \leftarrow smith\_waterman\_phase1(column_0, column_1)$
7: sends $(score', coords')$ to master
8: receives $best\_score$ from master
9: loop
10:   asks for $job$ from master
11:   if $job = JOB_{end}$ then
12:     break
13:   end if
14:   $(min_x, min_y) \leftarrow smith\_waterman\_phase2step1(job_x, job_y, best\_score)$
15:   {$in\ phase2\ step2,\ each\ alignment\ is\ stored\ in\ the\ shared\ NFS\ directory\ as\ soon\ as\ it\ is\ found$}
16:   $smith\_waterman\_phase2step2(min_x, min_y, job_x, job_y, best\_score)$
17: end loop
```

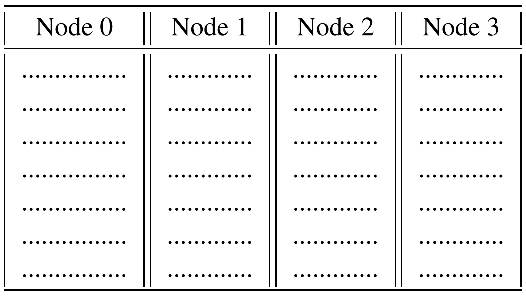


Fig. 7 Work division between 4 nodes in the first phase of the algorithm

will calculate its n^2/t cells using the wavefront method [23]. Computations evolve in a column basis, where each node shares a column of cells (*passage band*) with its neighbors. After calculating a block of values in the passage band, slave node P_i sends this block to slave node P_{i+1}. Figure 7 illustrates the work division among the nodes, where the double line areas correspond to the *passage band*. Note that, in the wave-front method, the amount of parallelism is non-uniform (Sect. 3.2) and, thus, assuming homogeneous nodes, node $t-1$ will be the last one to finish computing.

At the end of the computation, each slave P_i sends to the master M the best local score obtained (*score′*) and its coordinates (*coords′*) in the similarity matrix. Having received all best local scores, the master generates the best global score (*best_score*) and all coordinates where it can be found. Figure 8 illustrates this first parallel phase.

5.2 Parallelization of phase 2

In this phase, we use the master/slave model [26], where the master node distributes the coordinates of the best score among the slave nodes, that will actually obtain the local alignments.

Figure 9 illustrates the parallelization of phase 2. In this case, each pair of coordinates that belong to the set *coords′* is a work unit in a queue that is stored in the master. Work is distributed in a *Self-scheduling* basis [28], where the slaves process one working unit and, after finishing it, ask for one more working unit, until all work units are processed. We chose to use *Self-scheduling* because we do not know a priori how much time will be needed to generate the alignment(s), given a pair of coordinates. Note that many best local alignments can be found for a given pair of coordinates and, in such cases, *Self-scheduling* is a good choice [26].

When a best local alignment is found, the slave node places it in a public directory, which is shared through NFS (*Network File System*). When the master node detects that the work queue is empty, the algorithm ends.

6 Experimental results

The algorithm described in Sect. 5 was implemented in *C* and MPI, using *mpich* version 1.2.6. Our results were mainly collected in a 8-node cluster (*Cluster-UnB*), where each machine has two processors AMD Athlon XP 1.7 GHz, with 1 GB of RAM memory. The nodes are interconnected by a Gigabit Ethernet switch (1 Gb/s). The operating system used was Linux version 2.4.20-8smp. A 16-node cluster (*Cluster-Sun*) was also used, where each node is a AMD Opteron 2.2 GHz with 1 GB of RAM memory, interconnected also by a Gigabit Ethernet switch. The operating system used in *Cluster-Sun* was also Linux kernel 2.4. It should be noted that, in *Cluster-UnB*, only one processor was used in each node.

In our tests, we used random pairs of DNA sequences of sizes 1 KBP, 2 KBP, 4 KBP, 8 KBP, 16 KBP, 32 KBP, 64 KBP, 128 KBP, 256 KBP, 512 KBP, 800 KBP and 1600 KBP. Each pair of sequences was compared with 1, 2, 4 and 8 processors. In some cases, 16 processors were used. The block factor used for communication between neighbor slave processors (Sect. 5.1) in the first phase was 48 bytes. This value was empirically obtained with tests on 128 KBP sequences.

Table 1 presents the maximum score between each pair of sequences (*Max. Score*), the number of coordinates that contain this score (*# of Coordinates*), the number of distinct pairs of coordinates that generate best alignments (*# of Pairs*) and the total number of alignments found (*# of Alignments*).

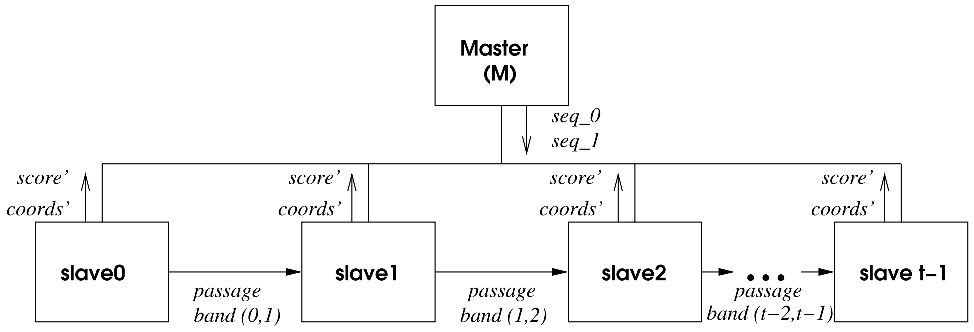


Fig. 8 Parallelization of phase 1

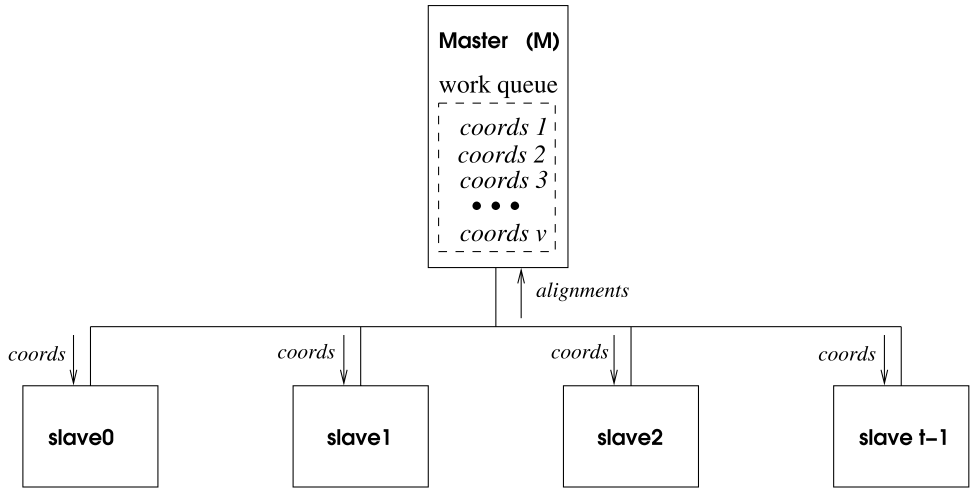


Fig. 9 Parallelization of phase 2

Table 1 Best scores and number of alignments found for the sequences

Size	Max. score	# of coordinates	# of pairs	# of alignments
1 KBP	17	2	2	12
2 KBP	18	2	2	384
4 KBP	20	1	2	72
8 KBP	20	1	4	128
16 KBP	20	6	10	2322
32 KBP	25	3	7	14688
64 KBP	25	1	1	168
128 KBP	30	2	6	1008
256 KBP	32	1	1	192
512 KBP	34	2	6	16800
800 KBP	36	2	2	288
1.6 MBP	39	1	2	30240

In this table, we can see that for some sequences (1 KBP, 2 KBP, 64 KBP, 256 KBP, 800 KBP), one best score coordinate generated only one pair of coordinates. In other sequences, such as the 16 KBP sequence, 6 different best scores of value 20 generated 10 pairs of coordinates.

More interesting, in all sequences studied, a pair of coordinates generated more than one best alignment. For instance, the comparison of 512 KBP sequences produced 16800 different best local alignments for only 6 different pairs of coordinates with the best score. Analyzing the alignments produced, we noticed that most of them were quite similar, with a difference in a gap position, for instance. However, for biologists, this difference can determine, in some cases, different proteins and thus, different funcionalities. For this reason, in our tests, we opted to generate all possible best alignments. The number of alignments produced can be easily limited, though.

Table 2 Total execution times (seconds), including phases 1, 2, initialization and termination

Size	1 proc	2 procs	4 procs	8 procs
1 KBP	0.27	0.34	0.53	0.94
2 KBP	0.36	0.42	0.67	1.14
4 KBP	0.48	0.45	0.68	1.21
8 KBP	1.19	0.82	0.85	1.18
16 KBP	4.28	2.42	1.73	1.75
32 KBP	17.87	10.19	6.48	5.05
64 KBP	66.69	31.18	15.77	8.52
128 KBP	282.34	133.48	62.46	31.60
256 KBP	1128.85	565.10	266.55	124.74
512 KBP	4516.57	2265.83	1139.22	536.48
800 KBP	11032.87	5529.29	2764.09	1391.37
1600 KBP	44131.39	22122.21	11050.79	5554.53

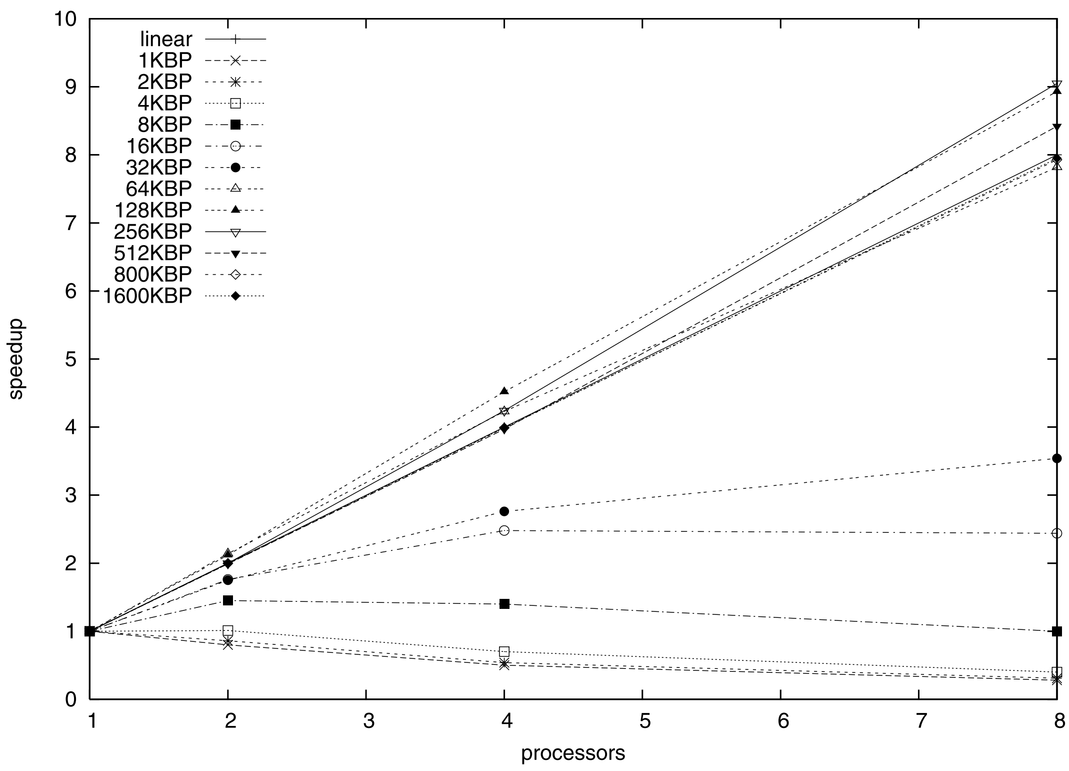


Fig. 10 Relative speedups from *cluster-UnB*

Table 2 presents the total execution times (seconds) of our parallel algorithm (phases 1 and 2) for different sequence sizes in *Cluster-UnB*. The relative speedups obtained are presented in Fig. 10.

As can be seen in Fig. 10, the speedups obtained for small sequences (less or equal than 32 KBP) are not good. In other parallel versions of SW [16, 24], the same phenomenon occurs. This is due to the fact that, since the sequences are small, the parallelism was not able to surpass the overhead introduced by the communication among the processors. However, as long as the sequence sizes are increased, very good speedups are obtained. In our tests, superlinear speedups were obtained for 128 KBP, 256 KBP and 512 KBP sequences. For 256 KBP sequences, the speedup obtained was 9.05, with 8 processors. This happens because, when we add more processors, each processor works on less data and that has an impact over the cache hit ratio.

Table 3 presents the execution time to execute phases 1 and 2 of our algorithm, with 8 processors. Table 4 shows the number of cells calculated in phase1, phase2-step1 and

Table 3 Execution times (seconds) from phase 1 and phase 2 with 8 processors

Size	Phase 1 (s)	Phase 2 (s)
1 KBP	0.120	0.003
2 KBP	0.260	0.020
4 KBP	0.109	0.002
8 KBP	0.299	0.006
16 KBP	0.559	0.020
32 KBP	1.949	1.540
64 KBP	7.536	0.004
128 KBP	30.471	0.035
256 KBP	123.808	0.010
512 KBP	532.834	1.919
800 KBP	1390.084	0.009
1600 KBP	5550.856	1.734

Table 4 Number of cells calculated in each phase

Size	Phase 1	Phase 2-step 1	Phase 2-step 2
1 KBP	1.0×10^6	3841	1219
2 KBP	4.0×10^6	5142	4478
4 KBP	1.6×10^7	4527	647
8 KBP	6.4×10^7	5568	2717
16 KBP	2.6×10^8	23000	5430
32 KBP	1.0×10^9	24901	5494
64 KBP	4.1×10^9	6131	710
128 KBP	1.6×10^{10}	14028	3398
256 KBP	6.6×10^{10}	8392	1309
512 KBP	2.6×10^{11}	30178	7150
800 KBP	6.4×10^{11}	24790	4694
1600 KBP	2.6×10^{12}	16982	3511

phase2-step2 (Sect. 4.2). This table confirms the theoretical results in [4]. The first phase is space linear on the size of the sequences, since we only use two rows. The second phase is quadratic on the size of the best alignments but, as can be seen in Table 4, a small number of cells (no more than 8000, in our tests) is stored in memory (phase2-step2).

As expected, the time needed in phase 1 is much higher than the time needed in phase 2. Note that the time spent in phase 2 depends on the number of best local alignments to be calculated (Table 1) while the time spent in phase 1 depends on the size of the sequences ($O(n^2)$). For 1.6 MBP sequences, where the highest number of alignments were computed (30240), 1.54 hours were spent in phase 1 and 1.73 seconds in phase 2, which corresponds to 0.03% of the total time spent. Still for the 1.6 MBP sequences, our algorithm dynamically allocated one row of 1.6 MB in phase 1, related with the beginning and end of two consecutive rows. If the original SW algorithm were used, at least 2.56 TB of memory would be necessary.

An interesting case is that of the 32 KBP sequence, where 44.13% of the execution time is spent in phase 2. This occurs because the sequences are relatively small and the number of best local alignments found is relatively high (14688). As explained in Sect. 4.2, the total number of best local alignments produced can grow exponentially with the size of the alignments. That's why such a high number of alignments could be produced. If the user wants, however, the total number of alignments produced can be limited.

In Fig. 11, we present one of the best local alignments produced for the 1.6 MBP sequences. This alignment has a score of 39 and occurs in fragments $s[918866..919068]$ and $t[209307..209513]$.

We also tested our algorithm in a cluster with 16 nodes (*Cluster-Sun*), which belongs to *Sun Microsystems Brazil*. In this cluster, we executed phase 1 for the 256 KBP and 800 KBP sequences. The total execution times obtained in this cluster where around 45% smaller than the ones obtained in *Cluster-UnB*. For instance, to compare the same

Fig. 11 One of the best local alignments obtained with the 1.6 MBP sequences

```
ACCCTTCATATTTGGGTACTAC-TGTCGCGAAACTCAAGAGAGCTTCCCCTGTACTTGCA
::::::.::::::::.::..:. :::. ::::::.::.::.:::::..::: : ::..::
ACCCTTTATATTTGGCTATAAGGTGTT-CGAAACCCAGGATAGCTTAACCT-T-CTCACA

TTCTAGAAAACGACCGATCCAG-A-GGTCTT-GGATAGTAAGACAACTCCTATACTCATA
:: ::.::.:. ::.::.::. : ::.:.. ::::::::::.::...:::.::. :::.
TT-AGGAACCC-CCTATTCACTATGGCCCGCGGATAGTAAGTCAGGACCTTTAG-CATG

TATCAGGCTTCGCCATTTAGGGACACGCTCTATATGGTCTATTTAGCTCTTCCTTTGTGC
.::.:.:.:::..::.: :::..::...:....:.:.:..:::.::. ::.:::::: ::
CATGATGGTTCTTCAAT-AGGTCCAATATTGCAAAGATGAATTGAGA-CTACCTTTG-GC

ACTGTATTGGAGTG-T-G-TGTACCCT-GAAGTAC
::.:.: :::..: : : ::.::::: :.:::::
ACAGGA-GGAACGCTCGCTGCACCCTAGCAGTAC
```

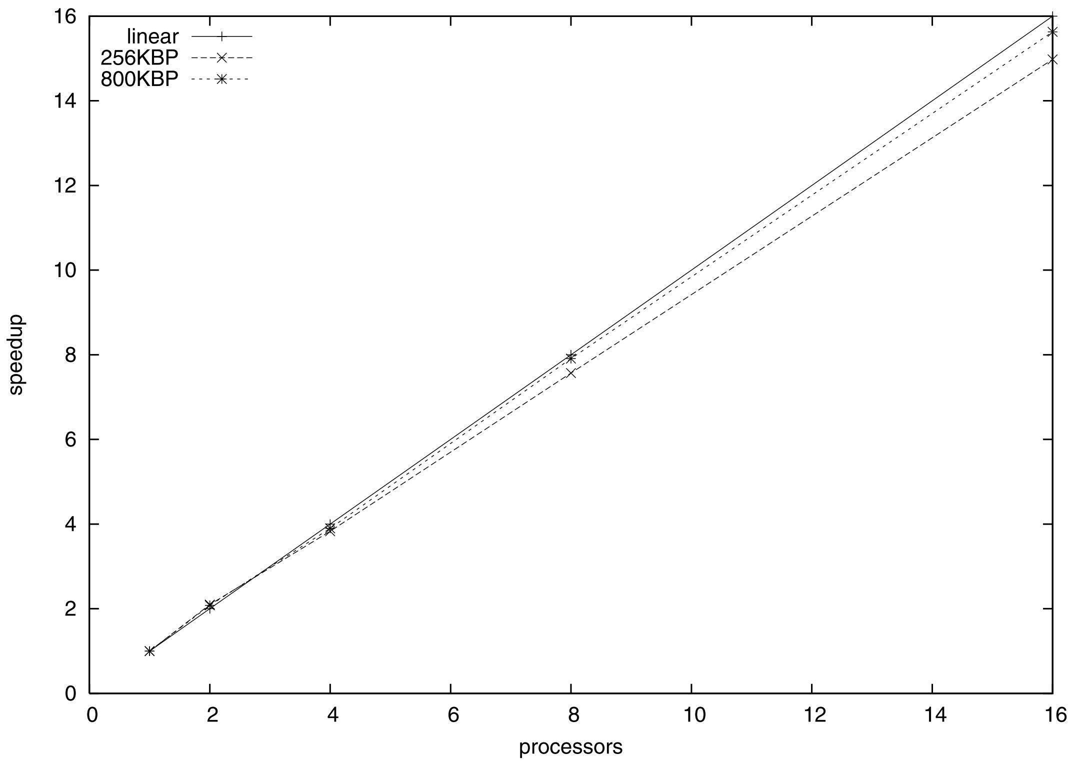


Fig. 12 Speedups obtained with the phase 1 of our algorithm in the *Cluster-Sun*

two 800 KBP sequences with 8 processors, 1391.37 seconds were spent in *Cluster-UnB* and 879 seconds in *Cluster-Sun*.

As can be seen in Fig. 12, the speedups obtained for the first phase of our parallel algorithm with 16 processors in *Cluster-Sun* were close to linear (14.98 for the 256 KBP sequences and 15.63 for the 800 KBP sequences).

7 Conclusion and future work

In this paper, we proposed, implemented and evaluated an exact parallel variant of the SW algorithm to perform local comparisons of long biological sequences. The exact variant of the SW algorithm proposed in this paper executes in linear space in phase 1 and also includes strategies to reduce the amount of cells calculated in the second phase. The parallel version of our algorithm executes in two phases. In the first phase, parallel computations evolve in a wavefront way, with communication between the neighbor processors. The second phase of our parallel algorithm uses the master/slave model to distribute work to slave nodes according to the *Self-scheduling* [28] task allocation policy.

The results obtained in a 8-node cluster (*Cluster-UnB*) and in a 16-node cluster (*Cluster-Sun*) presented speedups close to linear and, in some cases, superlinear, for long sequences (longer than 32 KBP). With the proposed algorithm, we were able to obtain all best local alignments between two 1.6 MBP sequences with the *Cluster-UnB* in approximately 1.54 hours, which is a very promising result. As far as we know, this is the first time 1.6 MBP sequences are compared with an exact parallel variant of the SW algorithm. The results obtained in the *Cluster-Sun* show that close to linear speedups are also obtained with 16 nodes.

As future work, we intend to replace the fixed gap penalty for a function (*affine gap*) as in [24] and compare the best alignments obtained in both approaches (fixed penalty and affine gap). Besides that, the proposed algorithm will be tested in a heterogeneous cluster and in a homogeneous cluster with more machines. Also, we intend to use our parallel algorithm to analyze real DNA sequences generated by genome projects.

Acknowledgements Dr. A. Boukerche's work is partially supported by NSERC, Canada Research Chair Program, and Ontario Distinguished Researcher Award. The authors would like to thank Antonio Gil B. de Barros (Caritas Informatica) and Carlos Thomaz (Sun Microsystems) for having conducted the tests with our parallel algorithm in the *Cluster-Sun*.

References

1. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Molec. Biol. **214**, 403–410 (1990)
2. Batista, R.B., Silva, D.N., Melo, A.C.M.A., Weigang, L.: Using a dsm application to locally align dna sequences. In: Proc. of the IEEE/ACM Int. Symp. on Cluster Computing and the Grid. IEEE Computer Society, Los Alamitos (2004)
3. Boukerche, A., Melo, A.C.M.A.: Computational Molecular Biology. In: Zomaya, A.Y. (ed.) Parallel Computing for Bioinformatics and Computational Biology, pp. 149–165. Wiley Interscience, Hoboken (2006)
4. Boukerche, A., Melo, A.C.M.A., Ayala-Rincon, M., Santana, T.M.: Parallel smith-waterman algorithm for local dna comparison in a cluster of workstations. In: 4th Int. Workshop on Experimental and Efficient Algorithms. Lecture Notes in Computer Science, vol. 3530, pp. 464–475. Springer, Heidelberg (2005)
5. Boukerche, A., Melo, A.C.M.A., Walter, M.E.M.T., Melo, R.C.F., Santana, M.N.P., Batista, R.B.: Performance evaluation of a local dna sequence alignment algorithm on a cluster of workstations. In: Proc. of the Int. Parallel and Distributed Processing Symposium (IPDPS2004). IEEE Society, Los Alamitos (2004)
6. Chang, W.I., Lawler, E.W.: Approximate string matching in sublinear expected time. In: IEEE Thirty-first Annual Symposium on Foundations of Computer Science, 1990, pp. 116–124
7. Chen, C., Schmidt, B.: Computing large-scale alignments on a multi-cluster. In: IEEE International Conference on Cluster Computing, 2003
8. Fickett, J.: Fast optimal alignments. Nucleic Acids Res. **12**(1), 175–179 (1984)
9. Galper, A.R., Brutlag, D.R.: Parallel similarity search and alignment with the dynamic programming method. Technical Report KSL 90-74, Stanford University, 1990, pp. 1–14
10. Gusfield D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Press Syndicate of the University of Cambridge, New York (1997)
11. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Commun. ACM **18**(6), 341–343 (1975)
12. Hu, S., Shi, W., Tang, Z.: Jiajia: An svm system based on a new cache coherence protocol. In: High Performance Computing and Networking (HPCN), pp. 463–472. Springer, Heidelberg (1999)
13. Huang, X., Miller, W.: A time efficient, linear-space local similarity algorithm. Adv. Appl. Math. **12**, 337–357 (1991)
14. Landau, G., Viskin, U.: Introducing efficient parallelism into approximate string matching and new serial algorithm. In: 18th ACM STOC, 1986, pp. 220–230
15. Martins, W.S., Cuvillo, J.B.D., Useche, F.J., Theobald, K.B., Gao, G.R.: A multithread parallel implementation of a dynamic programming algorithm for sequence comparison. In: Brazilian Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2001, pp. 1–8
16. Melo, R., Walter, M.E.T., Melo, A.C.M.A., Batista, R.B.: Comparing two long dna sequences using a dsm system. In: Euro-Par 2003: Parallel Processing. Lecture Notes in Computer Science, vol. 2790, pp. 517–524. Springer, Heidelberg (2003)
17. Myers, E.W.: An o(nd) difference algorithm and its variations. Algorithmica **1**, 251–266 (1986)
18. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (2001)
19. NCBI: Ncbi homepage. Website, http://www.ncbi.nlm.nih.gov/, Nov. 2004
20. NCBI: Submit to genbank. Website, http://www.ncbi.nlm.nih.gov/Genbank/index.html, Nov. 2004
21. Needleman, S., Wunsch, C.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**, 443–453 (1970)
22. Pearson, W.R., Lipman, D.J.: Improved tools for biological sequence analysis. Proc. Natl. Acad. Sci. USA **85**, 2444–2448 (1988)
23. Pfister, G.: In: Search of Clusters—The Coming Battle for Lowly Parallel Computing. Prentice-Hall, Upper Saddle River (1995)
24. Rajko, S., Aluru, S.: Space and time optimal parallel sequence alignments. IEEE Trans. Parallel Distributed Syst. **15**(12), 1070–1081 (2004)
25. Setubal J.C., Meidanis J.: Introduction to Computational Molecular Biology. Brooks/Cole, Boston (1997)
26. Shao, G.: Adaptive scheduling of master/worker applications on distributed computational resources. PhD thesis, University of California at San Diego (2001)
27. Smith, T., Waterman, M.: Identification of common molecular subsequences. J. Mol. Biol. **147**, 195–197 (1981)
28. Tang, P., Yew, P.C.: Processor self-scheduling for multiple nested parallel loops. In: Int. Conf. on Parallel Processing (ICPP), 1986, pp. 528–535
29. Ukkonen, E.: Algorithms for approximate string matching. Inf. Control **64**(1), 100–118 (1985)
30. Zhang, F., Qiao, X., Liu, Z.: A parallel smith waterman algorithm based on divide and conquer. In: Fifth Int. Conf. on Algorithm and Architectures for Parallel Processing (ICA3PP02), pp. 162–169. IEEE Society, Los Alamitos (2002)

Azzedine Boukerche is a Full Professor and holds a Canada Research Chair position at the University of Ottawa. He is the Founding Director of PARADISE Research Laboratory at Ottawa U. Prior to this, he held a Faculty position at the University of North Texas, USA, and he was working as a Senior Scientist at the Simulation Sciences Division, Metron Corporation located in San Diego. He was also employed as a Faculty at the School of Computer Science McGill University, and taught at Polytechnic of Montreal. He spent a year at the JPL/NASA-California Institute of Technology where he contributed to a project centered about the specification and verification of the software used to control interplanetary spacecraft operated by JPL/NASA Laboratory.

His current research interests include distributed computing, Bio-Inspired Computing, large-scale distributed simulation, wireless networks, mobile and pervasive computing, peformance evaluation and modeling of large-scale distributed systems, and parallel discrete event simulation. Dr. Boukerche has published several research papers in these areas. He was the recipient of the Best Research Paper Award at IEEE/ACM PADS'97, and the recipient of the 3rd National Award for Telecommunication Software 1999 for his work on a distributed security systems on mobile phone operations, and has been nominated for the best paper award at the IEEE/ACM PADS'99, ACM MSWiM 2001, and ACM MobiWac 2004.

Dr. A. Boukerche is a holder of an Early Career Research Excellence Award (previously known as Premier of Ontario Research Excellence Award), Ontario Distinguished Researcher Award, and Glinski Research Excellence Award. He is a Co-Founder of QShine Int'l Conference, on Quality of Service for Wireless/Wired Heterogeneous Networks (QShine 2004), served as a General Chair for several conferences, such as ACM/IEEE MASCOST 1998, IEEE DS-RT 1999-2000, ACM MSWiM 2000; Program Chair for ACM/IFIPS Europar 2002, IEEE/SCS Annual Simulation Symposium ANNS 2002, ACM WWW'02, IEEE MWCN 2002, IEEE/ACM MASCOTS 2002, IEEE Wireless Local Networks WLN 03-04; IEEE WMAN 04-05, ACM MSWiM 98-99, and TPC member of numerous IEEE and ACM conferences. He served as a Guest Editor for the Journal of Parallel and Distributed Computying (JPDC) (Special Issue for Routing for mobile Ad hoc, Specail Issue for wireless communication and mobile computing, Special Issue for mobile ad hoc networking and computing), and ACM/kluwer Wireless Networks and ACM/Kluwer Mobile Networks Applications, and the Journal of Wireless Communication and Mobile Computing.

Dr. A. Boukerche serves as a General Chair for the 8th ACM/IEEE Symposium on modeling, analysis and simulaltion of wireless and mobile systems, and the 9th ACM/IEEE Symposium on distributed simulation and realt time-application, and as Vice General Chair for the 3rd IEEE Distributed Computing for Sensor Networks (DCOSS) Conference 2007.

Dr. A. Boukerche serves as an Associate Editor for the Journal of Parallel and Distributed Computing, Wiley In't Journal of Wireless Communication and Mobile Computing, the ACM/Kluwer Wireless Networks, and the SCS Transactions on Simulation. He also serves as a Steering Committee Chair for the ACM Modeling, Analysis and Simulation fo Wireless and Mobile Systems Symposium, the ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks and the IEEE Distributed Simulation and Real-Time Applications Symposium (DS-RT).

Alba Cristina Magalhaes Alves de Melo received her PhD in Computer Science from the Institut National Polytechnique de Grenoble (INPG), France, in 1996, her M.S. in Computer Science from Federal University of Rio Grande do Sul, Brazil, in 1991 and her B.S. in Computer Science from University of Brasilia, Brazil, in 1986. She is currently an associate professor in the Computer Science Department at the University of Brasilia, Brazil. Her research interests include distributed shared memory, load balancing, cluster computing, grid computing and computational biology. She is a senior member of the IEEE Society.

Edans Flavius de Oliveira Sandes received his B.S. in Computer Science from University of Brasilia (UnB), Brazil, in 2006. He achieved the 2nd place on the Brazilian Olympiads in Informatics in 2001 and represented the brazilian team on International Olympiad in Informatics (IOI) in the same year. He has also won 2 bronze medals and 2 silver medals on ACM-ICPC South America Contest. His research interests include computational biology, cluster computing and anonymous networks.

Mauricio Ayala-Rincon received the B.S. degrees in Computer Engineering and Mathematics in the *Universidad de Los Andes*, Colombia in 1985 and 1987, respectively, and the Dr. rer. nat. degree in Computer science from the *Universität Kaiserslautern* in 1993. He is a professor in the Mathematics and Computer Science Departments at the *University of Brasilia (UnB)*, the vice-head of the former Department and the leader of the Group of Theory of Computation of his university. He received research awards from the Brazilian Mathematics Research Foundation FEMAT in 1998 and 2001 and receives a research award from the Brazilian Research Council CNPq since 2003. His current research focuses on algorithms, term rewriting systems, logic and semantics of programming languages and their applications in formal specification and verification of computer systems. He is a member of the European Association for Theory of Computation EATCS.