International Journal of Foundations of Computer Science
Vol. 23, No. 2 (2012) 407–429

DOI: 10.1142/S0129054112400205

EXACT PARALLEL ALIGNMENT OF MEGABASE GENOMIC SEQUENCES WITH TUNABLE WORK DISTRIBUTION

AZZEDINE BOUKERCHE

School of Information Technology and Engineering, University of Ottawa, Canada
boukerch@site.uottawa.ca

RODOLFO BEZERRA BATISTA*,
ALBA CRISTINA MAGALHAES ALVES DE MELO†,
FELIPE BRANDT SCAREL‡ and
LAVIR ANTONIO BAHIA CARVALHO DE SOUZA§

Department of Computer Science, University of Brasilia, Brazil
**rodolfobatista@gmail.com*
†*albamm@cic.unb.br*
‡*felipe@cic.unb.br*
§*lavir@cic.unb.br*

Received 24 August 2009
Accepted 22 August 2010
Communicated by El-Ghazali Talbi

Sequence Alignment is a basic operation in Bioinformatics that is performed thousands of times, on daily basis. The exact methods for pairwise alignment have quadratic time complexity. For this reason, heuristic methods such as BLAST are widely used. To obtain exact results faster, parallel strategies have been proposed but most of them fail to align huge biological sequences. This happens because not only the quadratic time must be considered but also the space should be reduced. In this paper, we evaluate the performance of Z-align, a parallel exact strategy that runs in user-restricted memory space. Also, we propose and evaluate a tunable work distribution mechanism. The results obtained in two clusters show that two sequences of size 24MBP (Mega Base Pairs) and 23MBP, respectively, were successfully aligned with Z-align. Also, in order to align two 3MBP sequences, a speedup of 34.35 was achieved for 64 processors. The evaluation of our work distribution mechanism shows that the execution times can be sensibly reduced when appropriate parameters are chosen. Finally, when comparing Z-align with BLAST, it is clear that, in many cases, Z-align is able to produce alignments with higher score.

Keywords: Parallel algorithms; dynamic programming.

1. Introduction

In the last decades, we have observed an unprecedented development in molecular biology. A very high number of organisms have been sequenced in genome projects, and many of those are waiting for further analysis.

Direct experimentation is the most reliable method to determine the function/structure of a newly sequenced organism. However, the experiments that must take place are very complex and time consuming. For this reason, it is far more productive to use computational methods to infer biological information from a sequence. This is usually done by comparing the new sequence with sequences that have already had their characteristics determined.

Biological sequence comparison is, therefore, one of the most basic operations in Bioinformatics. It is in fact a problem of finding an approximate pattern matching between two sequences, possibly introducing spaces (gaps) into them [13]. As a result of this operation, one or more alignments are produced.

The most common types of sequence alignment 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 algorithms must find the best match between parts of the sequences. One important issue to be considered is how gaps are treated. A simple solution assigns a constant penalty for gaps. However, it has been observed that keeping gaps together represents better the biological relations. Hence, the most widely used model among biologists is the affine gap model [7], where the penalty for opening a gap is higher than the penalty for extending it.

The algorithm proposed by Smith and Waterman (SW) [17] is one of the first exact methods in the literature to locally align two sequences with constant gap function. It is based on dynamic programming and it calculates a similarity matrix of size nxn, where n is the size of the sequences. SW has $O(n^2)$ time and space complexity . An algorithm based on SW that uses an affine gap function is proposed by Gotoh in [7].

Many methods were proposed to reduce space and/or time complexity of the basic algorithms (SW and NW [14]). An exact method was proposed in [12], based on [9], to compute optimal global alignments in linear space. If the sequences are known to be quite similar, faster exact methods such as [6] can be used.

In order to further reduce execution time, heuristic methods such as BLAST [1] were proposed. These methods combine exact pattern matching with dynamic programming in order to produce good solutions faster. BLAST can align sequences in a very short time, still producing good results. Nevertheless, its accuracy is expected to be worse than the accuracy of the exact methods, that produce optimal results.

Parallel processing can be used as an alternative to reduce the execution time of the exact methods. Most of the parallel strategies proposed in the literature [2, 3, 4, 5] use the wavefront method [15] to calculate the similarity matrix. In this method, the amount of parallelism is non-uniform, with neighbor communication.

In this paper, aligning huge genome sequences is defined to be the generation of pairwise alignments for sequences that are equal or longer than 1MBP (Mega Base Pairs). This is a challenging task since the algorithm to solve it must be optimized for both time and space. The most popular and publicly available software to generate exact alignments is SSEARCH (available at

www.biology.wustl.edu/gcg/ssearch.html), which compares a sequence with a genomic database, but the sequence size is limited to 500KBP (Kilo Base Pairs). For longer sequences, supermatcher, available at emboss.bioinformatics.nl, is presented as an alternative. However, the comparison of two 5MBP sequences with supermatcher fails due to insufficient memory. BLAST can be used to compare huge sequences, producing alignments in a short time.

Parallel SSEARCH was used in [11] to compare a sequence of 4KBP with a viral database of 105MB. Note that this problem is different from the problem of aligning two huge sequences. In this case, sequences that compose the database are small and distributed to the nodes. Each node compares a subset of small sequences in a sequential way.

In a previous paper, we proposed Z-align [2], a parallel exact strategy based on SW [17] that aligns biological sequences in restricted memory space, with the affine gap model. In its first version, Z-align used a uniform work distribution policy. Running on a 16-processor cluster, Z-align was able to compare 3MBP sequences. Nevertheless, it was observed that, for huge sequences, the idle time of the first processors was very high.

In the present paper, we evaluate the performance and the results produced by Z-align in two clusters (32 and 64 processors, respectively). With an adequate memory allocation strategy, Z-align was able to align two real DNA sequences of size 24MBP and 23MBP. As far as we know, this is the first time sequences of this size are aligned with an exact method. Also, we measured the speedups to compare real 3MBP sequences. In this case, a speedup of 34.35 was achieved for 64 processors. The evaluation of the tunable work distribution mechanism shows that execution times are highly dependent on the amount of work assigned to each processor. For work distribution, it must be taken into account that the reduction of processor idle time increases the amount of communication, for applications such as Z-align, that use the wavefront method. For this reason, tunable work division is highly appropriate.

The reminder of this paper is organized as follows. Section 2 presents the biological sequence alignment problem, the Smith-Waterman algorithm and its serial variations. Related work is discussed in Section 3. In Section 4, the Z-align strategy and the work division mechanism are described. Section 5 presents the experimental results. Finally, Section 6 concludes the paper and presents future work.

2. Biological Sequence Alignment

DNA and protein sequences are treated as strings represented, respectively, by elements of the alphabets $\Sigma^{DNA} = \{A, T, G, C\}$ and $\Sigma^{PROT} = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$. Aligning two biological sequences is a problem where the goal is to find a given pattern in a given text, allowing a limited number of errors [13].

2.1. *Similarity and alignment*

Two basic notions widely used in pairwise sequence comparison algorithms are similarity and alignment. The similarity is a measure of how similar two sequences are [13]. In general, very similar sequences present very few errors.

To measure the similarity between two sequences, a score is calculated. Given an alignment between sequences s and t, the following values are assigned for each column: ma, if both characters are identical (match); mi, if the characters are not identical (mismatch); and g, if one of the characters is a space (gap). The score is the sum of all these values. The similarity between two sequences is the highest score.

Table 1 presents one possible global alignment between two DNA sequences and its associated score. In this case, $ma = +1, mi = -1$ and $g = -2$.

Table 1. Global alignment and score between sequences $s = ATAGCT$ and $t = GATATGCA$.

-	A	T	A	-	G	C	T	Score
G	A	T	A	T	G	C	A	
-2	+1	+1	+1	-2	+1	+1	-1	0

The most important types of alignments are global and local. If all the characters of sequences must belong to an alignment, then we are dealing with a global alignment problem. On the other hand, a local alignment is used if only parts of the sequences compose the alignment.

2.2. *Smith-Waterman algorithm (SW)*

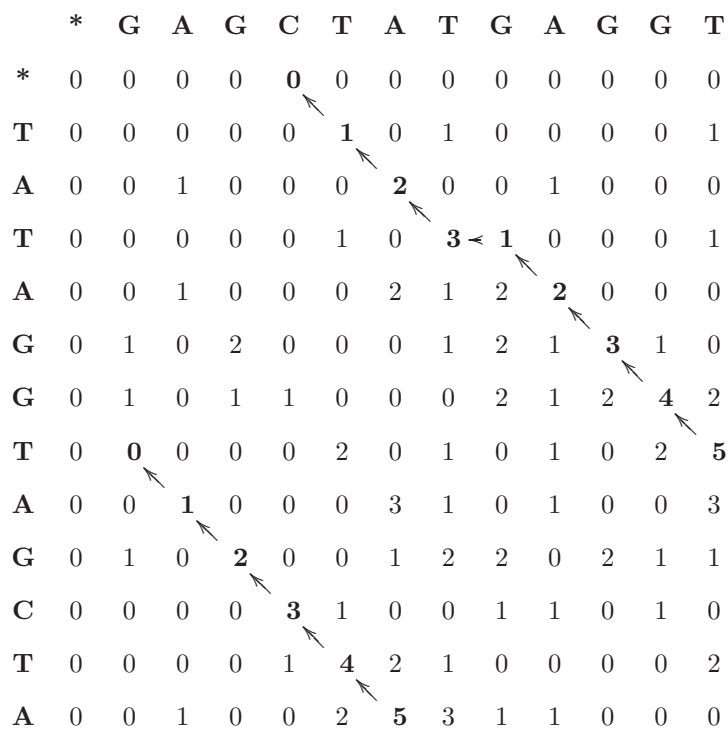
Needleman and Wunsh (NW) [14] proposed one of the first an exact algorithms to solve the global alignment problem. Smith and Waterman (SW) [17] modified NW to deal with local alignments. SW is based on dynamic programming and it is divided in two phases: create the similarity matrix and obtain the best alignment.

2.2.1. *Create the similarity matrix*

As input, SW receives sequences s and t, with $|s| = m$ and $|t| = n$, where $|s|$ represents the size of sequence s.

An array $D_{m+1,n+1}$ is built, where $D[i][j]$ contains the value of similarity between two prefixes of s and t ($sim(s[1..i], t[1..j])$), where $s[1..i]$ represents the prefix of sequence s with the first i characters.

At the beginning, the first row and column are filled with zeros. The remaining elements of D are obtained from Eq. 1. In Eq. 1, $p(i, j) = ma$ if $s(i) = t(j)$ and mi otherwise. The similarity score between s and t is the highest value.



	*	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. 1. Similarity matrix for local alignment between sequences $s = GAGCTATGAGGT$ and $t = TATAGGTAGCTA$.

$$D[i,j] = \max \left\{ \begin{array}{l} D[i-1][j] + g \\ D[i-1][j-1] + p(i,j) \\ D[i][j-1] + g \\ 0 \end{array} \right\} . \tag{1}$$

Figure 1 presents the similarity matrix between two sequences. The arrows indicate the cell from where the value was obtained.

2.2.2. *Obtain the best local alignment*

In order to obtain the best local alignment, the algorithm starts from the cell that has the highest score and follows the arrows until a zero-valued cell is reached. A left arrow in $D[i,j]$ (Fig. 1) 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 $t[j]$.

It must be noticed that multiple optimal local alignments can exist. In Fig. 1, there are two optimal alignments, that are shown in Table 2.

Table 2. Two optimal local alignments between sequences $s = GAGCTATGAGGT$ and $t = TATAGGTAGCTA$.

Alignment									Score
1	T	A	T	G	A	G	G	T	5
1	T	A	T	-	A	G	G	T	
2	A	G	C	T	A				5
2	A	G	C	T	A				

2.3. *Serial variations of the SW algorithm*

The algorithm SW (section 2.2) assigns a constant value to gaps. In [7], an algorithm is proposed by Gotoh where the opening of a gap has a greater penalty than its extension (affine gap model). In this algorithm, two matrixes are needed (P and Q), in addition to the similarity matrix. Fig. 2 illustrates the calculation of a similarity matrix cell in both SW and Gotoh.

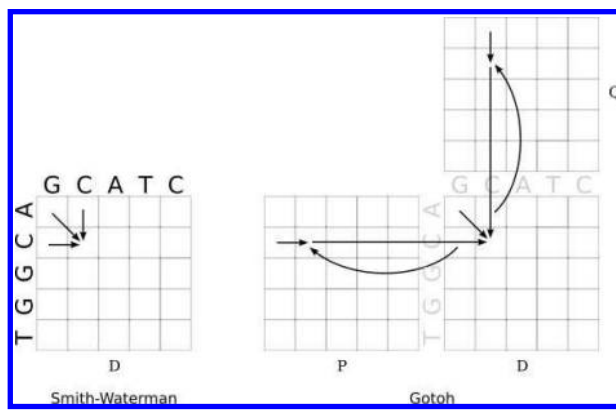


Fig. 2. A similarity matrix cell calculation in SW and Gotoh.

The use of Hirshberg's algorithm [9] is proposed by Myers and Miller [12] to compute global alignments in linear space. The algorithm uses a divide and conquer technique that finds a point where the optimal alignment occurs and recursively splits the similarity matrix to obtain the actual alignment. This approach can double the execution time [9], when compared with NW. In order to apply this algorithm to the local alignment problem, we need to find the coordinates of the rectangle that envelops the local alignment [8], transforming thus the local alignment problem into a global alignment one.

The method proposed by Fickett [6] tackles the global alignment problem and considers only similar sequences. It is based on the idea that, if the sequences are similar, the alignment between them is near the main diagonal. Thus, in order

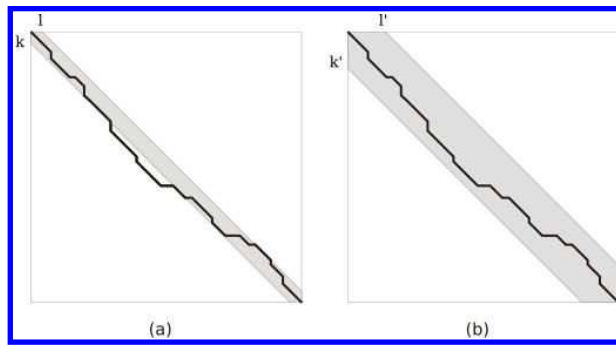


Fig. 3. Use of k-bands in Fickett's algorithm.

to compute the best alignment, it is sufficient to calculate and store only a small band (k-band) near the main diagonal. First, an initial band is estimated and the matrix is computed and stored only for this band (Fig. 3.a). The algorithm then finds the optimal score and does the traceback over the band. If the alignment is not contained in the band, it cannot be retrieved. In this situation, the band is augmented and the whole process is repeated, until the alignment can be retrieved (Fig. 3.b).

3. Related Work

In the SW algorithm and its variations, most of the time is spent calculating the similarity matrix D (Fig. 1) and this is the part which is usually parallelized. The access pattern presented by the matrix calculation is non-uniform and the parallelization strategy used is known as wavefront [15] (Fig. 4).

At the beginning of the computation, only processor $P1$ is computing (Fig. 4.a). When $P1$ finishes calculating the values of a border column, it sends them to processor $P2$, that can start calculating (Fig. 4.b). In Fig. 4.c, the maximum parallelism is attained. This parallelism decreases until the end of the computation, where only $P4$ computes.

In Fig. 4, the work distribution is uniform because, in a system with p processors, each processor calculates n/p columns, where n is the size of one of the sequences. It must be noticed that, although only the column-based assignment is shown in Fig. 4, row-based and antidiagonal-based assignments are also possible.

Table 3 presents a comparative view of the parallel and distributed strategies that execute parallel variants of the SW algorithm. Most of the parallel variants are exact, retrieving optimal alignments, with the exception of [3] and [19]. Also, most of the strategies [3, 4, 5] use the wavefront method (Fig. 4). All the wavefront-based solutions use uniform work assignment. The divide and conquer approaches distribute the work independently. The main difference between [19] and [16] is that the first one uses heuristics to obtain the alignment whereas the second one is an

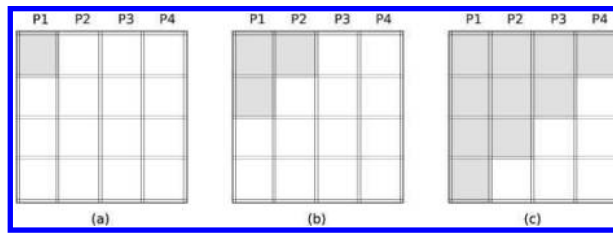


Fig. 4. The wavefront method with uniform work distribution.

exact method. The longest sequence size compared with the algorithms discussed in this section is 1.1 MBP. The problem treated by [10, 11, 20] is the comparison of a small query sequence with a genomic database. These databases are often composed by a great number of sequences and each processor compares the same sequence with a subset of the sequences that belong to the database. Therefore, even though a high amount of comparisons is made, the size of the resultant similarity matrices is not big. For instance, in [20], the maximum size of the query sequence is 2KBP.

Table 3. Characteristics of the parallel/distributed strategies that execute Smith-Waterman variants.

Ref.	Comparison type	Algo. type	Method	Parallel strategy	Size of the longest sequence
[3]	seq x seq	local	heuristic	wavefront	400KBP
[19]	seq x seq	local	heuristic	divide and conquer	16KBP
[16]	seq x seq	global	exact	divide and conquer	1.1MBP
[4]	seq x seq	local	exact	wavefront	800KBP
[5]	seq x seq	global	exact	wavefront	300KBP
[10]	seq x database	local	exact	distributed (coarse)	N/A
[11]	seq x database	local	exact	distributed (coarse)	N/A
[20]	seq x database	local	exact	distributed (coarse)	N/A

4. The Z-Align Parallel Strategy

The main goal of Z-align is to align huge sequences ($\geq$ 1MBP) with an exact parallel method. We claim that heuristic methods can achieve very good results when the sequences compared are small. As long as the size of the sequences increase, we observed that the heuristic results are not so good and, for closely related sequences, heuristic methods often produce a high amount of small alignments, since they are unable to recognize that, in fact, there is a big alignment, composed by sets of matches/mismatches with gapped regions between them.

Aligning huge sequences with exact methods is a challenging task due to huge memory requirements. For instance, in order to compare two sequences of 23MBP

with SW, we would need to store a similarity matrix of approximately 2.11PB, considering that each matrix cell stores an integer (4 bytes).

Of course, Hirschberg [9] could be used since it retrieves global alignments in linear space. However, in order to transform the local alignment problem into a global alignment one, we need to calculate the whole matrix once, in order to determine the coordinates (i_{end}, j_{end}) where the highest score ends. After that, we need to re-calculate the matrix, from the highest score position to the beginning, in order to determine the coordinates where the optimal alignment begins (i_{begin}, j_{begin}). In the worst case, the whole matrix is re-calculated. Having these two coordinates (i_{begin}, j_{begin}) and (i_{end}, j_{end}), Hirschberg [9] can be applied. In the worst case, the area comprised between (i_{begin}, j_{begin}) and (i_{end}, j_{end}) will be calculated twice [9]. Therefore, this approach uses linear memory at the expense of a great increase in the execution time.

4.1. *General view of Z-align*

In Z-align, we opted to use an alternative strategy, aiming to reduce the execution time. A general view of the Z-align strategy is shown in Fig. 5.

Z-align produces optimal local alignments with the affine gap model in four phases, briefly explained in this section. A detailed description of Z-align strategy can be found in [2].

In the first phase, the sequences to be compared are sent to all processors.

In the second phase (Fig. 5.a), the whole similarity matrices are calculated with the affine gap model in linear space. Computing starts from the end to the beginning. Each processor calculates a subset of the columns. By default, uniform work assignment is used where each processor calculates n/p columns, where n is the size of one sequence and p is the number of processors. Each set of columns is further divided into horizontal (h) and vertical (v) slices that define a block, where each block has size $(n/(p * h)) * (m/v)$. Fig. 6.b illustrates the default work distribution with $v = 8$ and $h = 2$.

Processing is done in a block-basis and only two rows are stored, the one being calculated and the previous one. Thus, this phase executes in linear space. The wavefront method (Fig. 4) is used and communication occurs between processors P_i and P_{i+1} when a border column is calculated.

Fickett's algorithm (Fig. 3) uses a band of width k along the main diagonal to reduce the processing time needed to retrieve the alignment. In the original algorithm, the value of k is estimated from the alignment score. In Z-align, phase 2 will calculate the whole similarity matrix in order to obtain the best score. For this reason, unlike Fickett, we are able to obtain the exact band which contains the alignment.

Therefore, besides the matrices D, P and Q (Fig. 2), we calculate, in phase 2, two additional dynamic programming matrices (DIV_{inf} and DIV_{sup}). The DIV matrices keep track of how the alignments diverge from the main diagonal and they

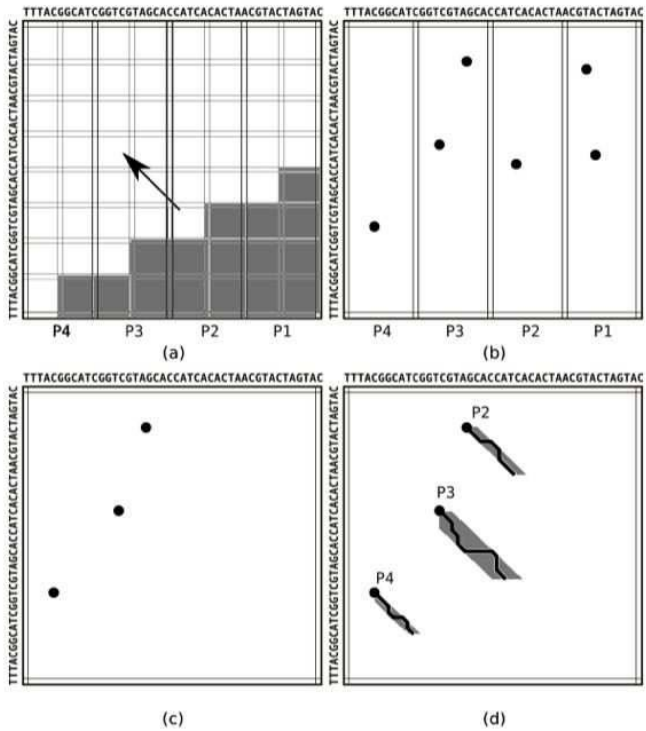


Fig. 5. General view of the Z-align strategy.

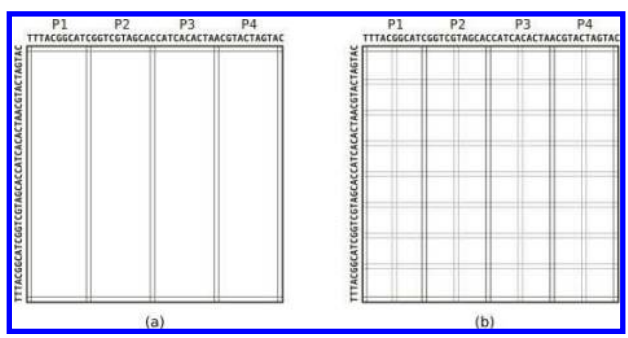


Fig. 6. Uniform work distribution in phase 2, with 2 horizontal and 8 vertical blocks per processor.

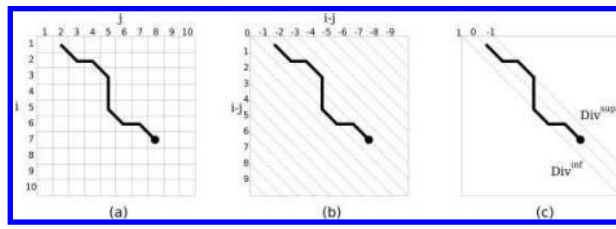


Fig. 7. Divergence matrices for an alignment. The final values ($DIV_{inf} = -1$ and $DIV_{sup} = +1$) refer to the beginning of the alignment.

contain, respectively, the inferior and superior difference between the coordinates of the best scores obtained so far (Fig. 7).

Phase 2 produces, for each processor, the coordinates of the beginning of the local alignment(s), the similarity score and the inferior and superior divergence (DIV_{inf} and DIV_{sup}) of each alignment (Fig. 5.b).

In the third phase, the results are collected and the global best score is known (Fig. 5.c).

In the fourth phase, the actual optimal local alignment(s) are produced, using data obtained in phase 2 (Fig. 5.d). In this phase, each processor retrieves one alignment, using a self-scheduling policy [18]. The area used to retrieve the alignments is defined by the DIV values, computed in phase 2. The processor starts calculating from the beginning to the end, in order to retrieve the coordinates were the optimal alignment ends. Having the beginning and end coordinates of the alignment, the processor retrieves the alignment, using the area in grey (Fig. 5.d). The memory used in this phase is defined by the user and, if the maximum amount of memory specified is attained, the last row computed (called cache line) is saved in memory and the rest of the memory is freed. Many rows can be saved this way [2], that will be used in the traceback process. The number of rows saved depends on the superior and inferior divergences, the size of the alignment and the maximum amount of memory specified by the user.

Since Z-align is intended for long-run executions, a checkpoint/restart mechanism was integrated with it. For Z-align, we implemented an application-specific strategy that tolerates multiple transient failures.

Our mechanism works as follows. After sending each message, the processor saves to its local disk the iteration number, the last row calculated in matrices D, P, Q, Div_{inf} and Div_{sup} and the data structure that keeps the highest score.

Since the checkpoint file is removed at the failure-free execution, its existence is sufficient to detect that a failure occurred. When this happens, the checkpoint file is used for recovery. In this case, all processors except $P1$ execute a blocking receive primitive. Processor $P1$ computes its row and sends its border to $P2$, that computes its row and sends its border to $P3$, and so on. In this way, the wavefront is reestablished.

4.2. *Tunable work distribution mechanism*

4.2.1. *Block cyclic attribution*

When Z-align uses uniform work distribution, the first processors finish processing much earlier than the last ones, remaining idle for a long period of time. In order to illustrate this situation, consider 4 processors with $h = 1$ and $v = 4$, as shown in Fig. 8. In the point of the wavefront calculation shown in Fig. 8.d, $P1$ has already finished processing but 37.5% of the matrices remain to be calculated (white blocks in Fig. 8.d). This problem gets worse when the number of processors is big and the size of the sequences is huge. For instance, when comparing 24MBP x 23MBP with 64 processors using the uniform work distribution, processor $P1$ remained idle for more than one day.

For this reason, we propose here the use of a tunable work distribution mechanism, called block-cyclic attribution, where a set of columns is assigned to each processor in a block cyclic way. The main idea is to divide the dynamic programming matrices in sections, called *splits*, and use the uniform distribution inside each *split*. The *split* value is a parameter of Z-align and must be provided by the user. If no value is provided, the default value ($split = 1$) is used, which corresponds to the original work distribution.

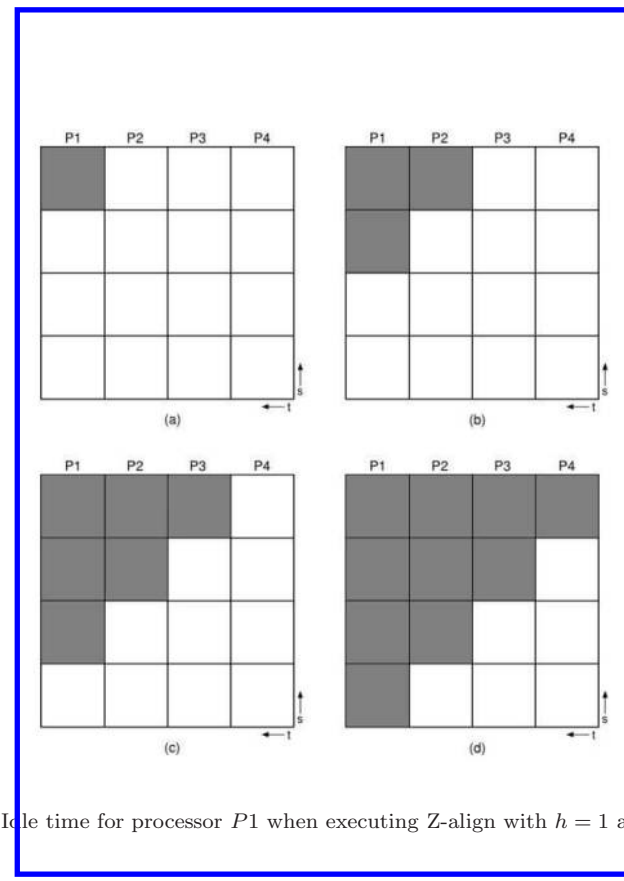


Fig. 8. Idle time for processor $P1$ when executing Z-align with $h = 1$ and $v = 4$.

Figure 9 illustrates this work assignment mechanism for 4 processors with $split = 2$. In this case, the dynamic programming matrices are divided in two sections. The first section contains the first half of the matrices and the second one contains the last half. As shown in Fig. 9.a,b,c, the matrices are processed as in the uniform work distribution case, until processor $P1$ finishes its first split. When this happens, $P1$ starts processing its second split, receiving the border columns from $P4$ (Fig. 9.d). In a similar way, when $P2$ finishes its first split, it receives the border column from P1 (Fig. 9.e) and so on.

In the case shown in Fig. 9, processor $P1$ finishes computing when there are 6 non-calculated blocks (Fig. 9.f), which corresponds to 18.75% of the matrices. Therefore, the area that comprises the non-processed cells is half the area left when the mechanism is not used (Fig. 8). Nevertheless, it must be noticed that the work division shown in Fig. 9 doubles the amount of communication between the processors.

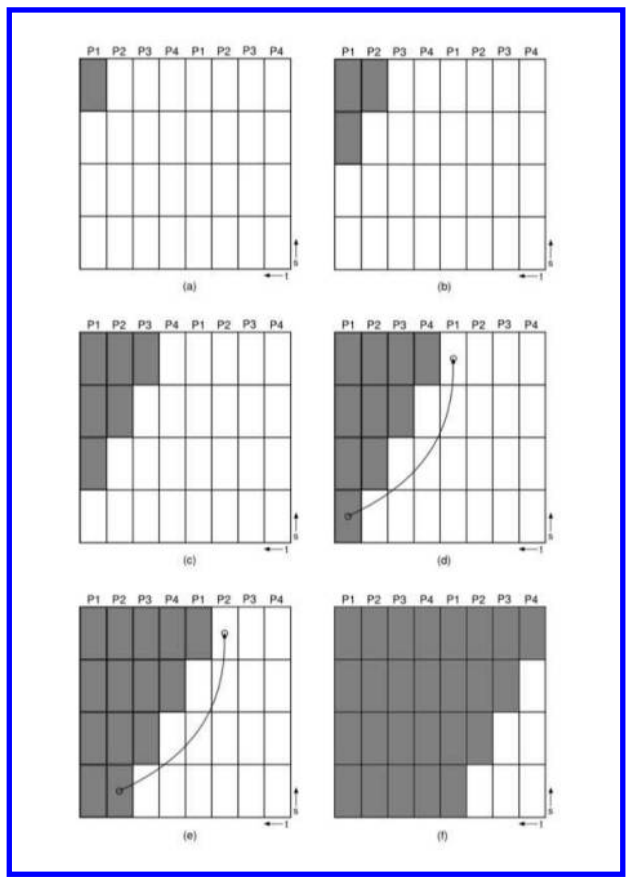


Fig. 9. Reduction of the idle time with the tunable work distribution mechanism ($split = 2$).

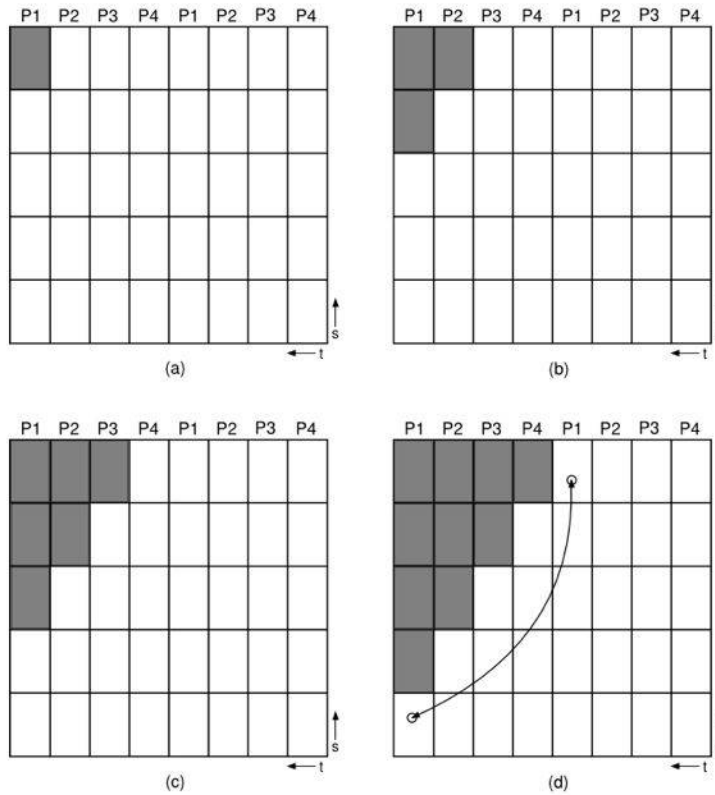


Fig. 10. Possible load imbalance when $split = 2$, $v = 1$ and $h = 5$.

Load imbalance can occur when the *split* parameter is used with some h and v configurations. Fig. 10 illustrates a situation that occurs when $split = 2$, $h = 1$ and $v = 5$. In this case, processor $P1$ is overloaded, since it has not finished to process its first split yet but processor $P4$ has finished calculating the border column and wants to send it to $P1$. To avoid such situations, if $split > 1$, we set the values h and v in such a way that a square division is obtained. In the case presented in Fig. 10, the value $h = np$ would be used.

4.2.2. *Impact of the v parameter*

In the original Z-align, the parameters h (horizontal) and v (vertical) are used to set the size of the block of the dynamic programming matrices (Fig. 6.b). The v parameter also defines the amount of data exchanged in each communication between neighbor processors. A high value of v means a high number of messages with less data in each message. In addition, a high value of v reduces the amount of idle time of the processors, at the expense of an increase in the number of messages.

Therefore, if the original Z-align parameter v is tuned in such a way that an appropriate value is used, the amount of idle time is expected to decrease, leading to a reduction on the total execution time.

5. Experimental Results

5.1. *Experimental environment*

The algorithm proposed in section 4 was implemented in C++ and MPI and it is publicly available at https://launchpad.net/zalign. Our tests were conducted on two clusters:

(1) *Cluster-SITE-Ottawa*: a dedicated 32-node cluster, where each node contains an Intel Xeon 3.4GHz Dual Core, 2 GB RAM and 120 GB disk. The nodes are interconnected by a Gigabit Ethernet switch. All the processors execute Linux Fedora Core 4, kernel 2.6.11-1.1369_FC4smp and LAM 7.1.1/MPI,
(2) *Cluster-IE-UnB*:a dedicated 32-node cluster, where each node contains a dual Core AMD 64 3.0GHz, 1 GB RAM and 120 GB disk. The nodes are interconnected by a Gigabit Ethernet switch. All the processors execute Solaris and mpich.

The following parameters were set in both Z-align and BLAST: 1 (match), -3 (mismatch), -5 (gap opening), -2 (gap extension). In our tests, memory in the fourth Z-align phase was restricted to 64MB.

Real DNA sequences obtained from the NCBI web page (www.ncbi.nlm.nih.gov) were used in the tests. The sequences compared, their size and the NCBI accession number are shown in Table 4.

5.2. *Performance results*

The results presented in this section were obtained with the Cluster-SITE-Ottawa. Wallclock execution times, divergences and the size of the alignments obtained for the sequences analyzed are shown in Table 5. Z-align was executed exclusively in 64 processors for comparisons of 7Mx10M, 23Mx24M and 35Mx5M. Note that the times presented here are the sum of the time spent in all four phases and that the checkpoint/restart mechanism was disabled. Z-align took more than four days to align the Drosophila chromosomes (23Mx24M) with 64 processors. As far as we know, this is the first time optimal affine-gap alignments are generated for sequences longer than 2MBP.

As can be observed in Table 5, the scores of the alignments produced for the 1MBP and 5MBP comparisons were very high. In order to produce these alignments, Z-align used four and seven cache lines (Section 4), for the 1MBP and 5MBP comparisons, respectively. To produce the other alignments, no cache lines were used.

The speedups obtained are shown in Fig. 11. It can be observed that very good speedups were achieved for sequences with size equal or less than 500KBP. For sequences of 1MBP and 3MBP, speedups of 30.73 and 33.54 were obtained, respectively. These speedups are considered appropriate, since the Z-align application

Table 4. Organisms compared, ranging from 150KBP (Kilo base-pairs) to 35MBP (Mega base-pairs).

Comp.	Size S1	NCBI Number	Name	Size S2	NCBI Number	Name
150Kx 150K	162114	NC_000898	*Human herpesvirus 6B*	171823	NC_007605	*Human herpesvirus 4*
500Kx 500K	542869	NC_003064	*Agrobacterium tumef. str.C58* plasmid AT	536165	NC_000914	*Rhizobium sp.* plasmid pNGR234a
1Mx 1M	1044459	CP000051	*Chlamydia trachomatis* A/HAR-13	1072950	AE002160	*Chlamydia muridarum* Nigg
3Mx 3M	3147090	BA000035	*Corynebact. efficiens* Ys-314 DNA	3282708	BX927147	*Corynebact. glut* ATCC 13032
5Mx 5M	5302044	AE016879	*Bacillus anthr. str. Ames*	5303436	AE017225	*Bacillus anthr. str. Sterne*
7Mx 10M	7247732	NC_005027	*Rhodopirellula baltica SH 1*	10604090	NC_003997	*Bacillus anthr. Ames* complete genome
23Mx 24M	23340351	NT_033779	*Drosophila melanog.* chromos. 2L	10604090	NT_037436	*Drosophila melanog.* chromos. 3L
35Mx 5M	35765263	NC_000022	*Homo sapiens* chromos. 22 complete seq.	5312606	NC_005957	*Bacillus thur. ser. konkukian str. 97-27*

Table 5. Divergence values, scores and execution times in seconds (1, 2, 4, 8, 16, 32, 64 processors).

Comp.	Div.	Score	1	2	4	8	16	32	64
150Kx 150K	(0,0)	18	1117	573	256	146	74	39	22
500Kx 500K	(0,0)	96	9760	4989	2506	1256	638	328	176
1Mx 1M	(4027,-99)	471621	32094	23460	11757	6055	3118	1726	1,044
3Mx 3M	(40,-43)	14537	294000	146885	74737	55511	28474	15139	8764
5Mx 5M	(109,-137)	5220960	—	—	—	—	76143	41064	23234
7Mx 10M	(6,-1)	172	—	—	—	—	—	—	55656
35Mx 5M	(0,0)	38	—	—	—	—	—	—	132271
23Mx 24M	(5,0)	9063	—	—	—	—	—	—	400863

presents a medium communication rate between neighbor processors due to the wavefront method.

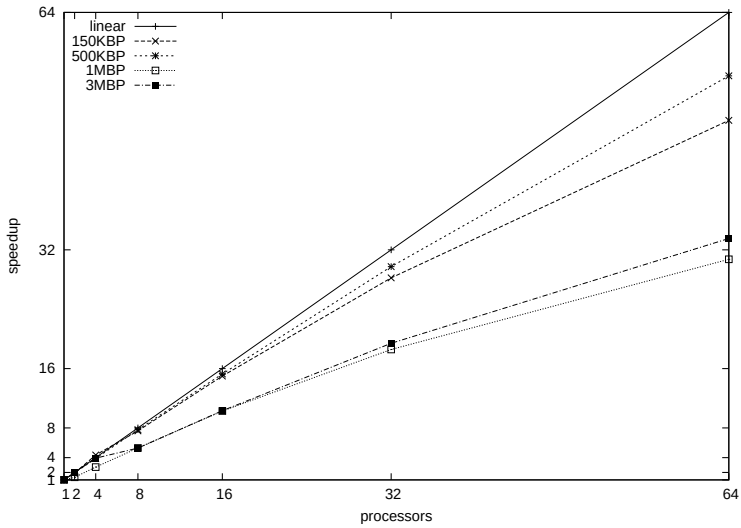


Fig. 11. Speedups for sequences ranging from 150KBP to 3MBP.

We also measured the average CPU load and RAM memory occupation for the pairwise comparisons in Table 5. The results are shown in Figs. 12 and 13. As shown in Fig. 12, the average amount of CPU used at the most compute-intensive phase of the Z-align application does not depend on the sizes of the sequences being compared. It ranges from 81.4% (3 MBP x 3 MBP) to 85.4% (24 MBP x 23MBP). It was observed that, in general, the time the processors were not computing, they were waiting at the MPI_Recv primitive. This happened because we used the Cluster-SITE-Ottawa, which is composed of Dual Core machines and, thus, the communication system is heterogeneous. Intra-node communication occurred faster and the processor waited for inter-node communication. However, this phenomenon did not lead the system to a severe load imbalance since more than 80% of the total execution time of the second phase was spent with computations, which is a very good result.

5.3. *Quality of the alignments*

Since Z-Align is an implementation of Smith-Waterman with affine-gap, it always returns the alignment(s) that has (have) the highest score. BLAST, on the other hand, is an heuristic method, so there is no guarantee that the alignment(s) with best score will be found. Nevertheless, BLAST is able to produce good results and, for this reason, it is widely used all over the world. In this section, we compare the alignments produced by Z-align with the ones produced by the heuristic method BLAST [1]. We used the publicly available *bl2seq* program with *blastn*, gaps enabled and searching only the top strand of the sequences.

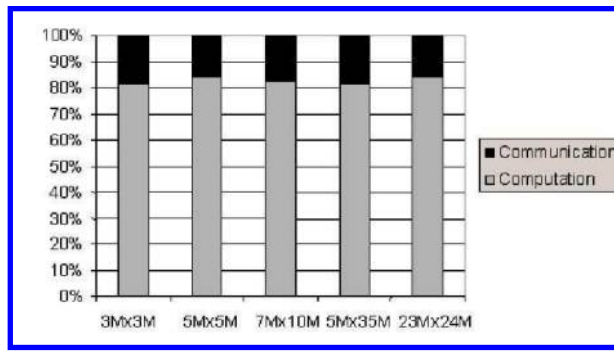


Fig. 12. Percentage of CPU and Communication in phase 2, for the 3Mx3M, 5Mx5M, 7Mx10M, 35Mx5M and 24Mx23M sequence comparisons.

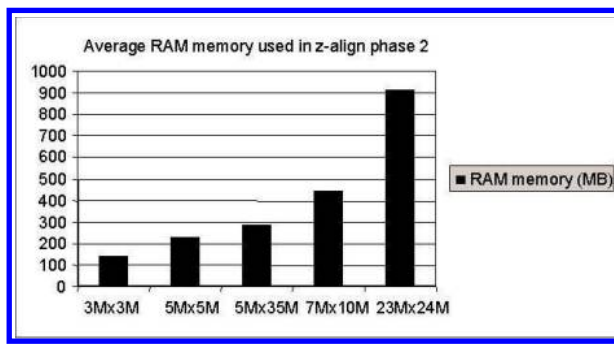


Fig. 13. Amount of memory (in MB) used in phase 2, for the 3Mx3M, 5Mx5M, 7Mx10M, 35Mx5M and 24Mx23M sequence comparisons.

Table 6. Alignments generated by Z-align and BLAST.

Comparison	Number of Z-align Alignments	Number of BLAST Alignments	Score of Z-align	Score of BLAST
150Kx150K	1	9	18	18
500Kx500K	2	19	96	92
1Mx1M	1	1,593	471,621	10,763
3Mx3M	1	5,057	14,537	5,329
5Mx5M	1	10,658	5,220,960	36,159
7Mx10M	2	329	172	157
24Mx23M	1	60,828	9,063	7,085
35Mx5M	1	40	38	24

Table 6 presents the number of alignments and the score obtained by both Z-align and BLAST. It can be noticed Z-align generates less alignments than BLAST. This was expected, since we opted to retrieve only the alignments with optimal score.

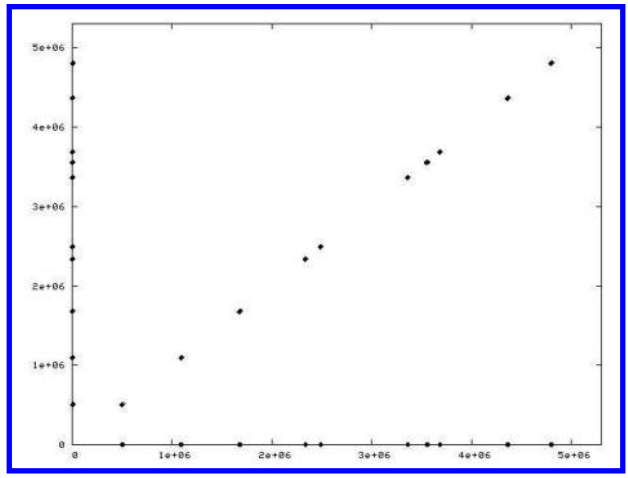


Fig. 14. Plot of the alignments retrieved by BLAST for the anthrax sequences (5MBP). Sequences AE016872 and AE0177225 (Table 4) are represented in the horizontal and vertical axes, respectively. Only alignments with score higher than 10,000 are plotted.

On the other hand, the alignments produced by Z-align have higher score than the alignments produced by BLAST, with the exception of the 150KBP comparison, where both methods retrieved exactly the same alignment. For the 5MBPx5MBP sequences, the score of the alignment found by Z-align is 144.38 times higher than the one produced by BLAST.

Figures 14 and 15 present a plot of the alignments found by Z-align and BLAST, respectively, when comparing two *Bacillus anthracis* (5MBP). For BLAST, only the alignments with score higher than 10,000 were plotted (Fig. 14). BLAST found 10,658 alignments (Table 6) that are close to each other whereas Z-align actually recognizes them as only one alignment, with score=5,220,960 (Fig. 15).

In Fig. 16, we can see the detail of the Z-align alignment, where a gapped region of size 408 was detected. In this case, the DNA region contained in (2,534,026-2,534,434) in AE017225 (*Bacillus anthracis Sterne*) does not occur in AE016879 (*Bacillus anthracis Ames*). This is a strong evidence of evolutionary phenomena. When using BLAST, the region that appears in Fig. 16 was considered as two separate alignments.

5.4. *Evaluation of the tunable work distribution mechanism*

In order to evaluate the gains of the tunable work distribution mechanism, we compared the Z-align execution times with original work assignment ($split = 1$) with the Z-align execution times obtained when the block-cyclic mechanism is active ($split > 1$). We also hand-tuned the value of the v parameter in the following way. We executed the 150Kx150K, 500Kx500K, 1Mx1M and 3Mx3M comparisons several times, increasing the value of v until the execution time got higher than the previous execution. When this happened, the value of v was set to the value that

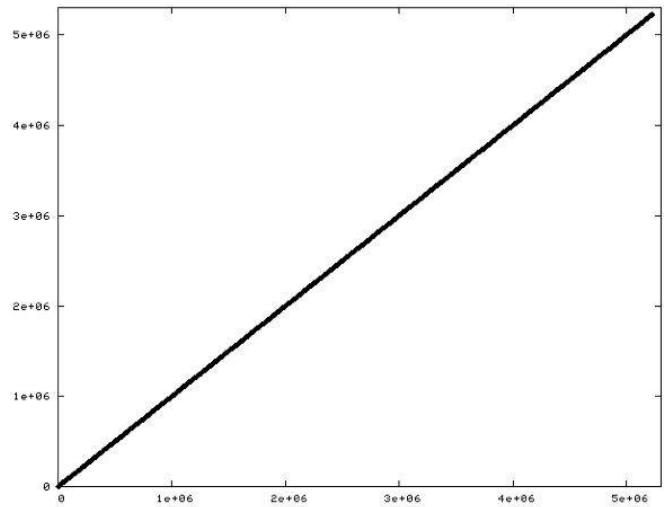


Fig. 15. Plot of the alignment retrieved by Z-align for the anthrax sequences (5MBP). Sequences AE016872 and AE0177225 (Table 4) are represented in the horizontal and vertical axes, respectively.

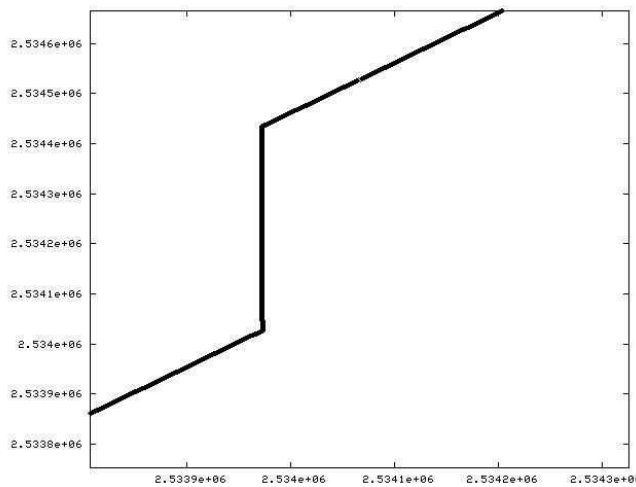


Fig. 16. Detail of the Z-align alignment for the anthrax sequences (5MBP). Sequences AE016872 and AE0177225 (Table 4) are represented in the horizontal and vertical axes, respectively. In both sequences, only the positions that range from 2,533,000 to 2,535,000 are shown.

led to the smallest execution time. Since this is a very time-consuming process, the comparisons of 5Mx5M, 7Mx10M, 24Mx23M and 35Mx5M were not executed in this section.

Table 7. Values of the parameters used for the original (split-orig, h-orig, v-orig), block cyclic (split-cyc, h-cyc, v-cyc) and hand-tuned (split-orig, h-orig, v-tuned) work distribution.

Comparison	split-orig	h-orig	v-orig	split-cyc	h-cyc	v-cyc	v-tuned
150Kx150K	1	20	p	2	1	64	128
500Kx500K	1	60	p	3	1	64	192
1Mx1M	1	150	p	4	2	64	256
3Mx3M	1	200	p	5	4	64	1000

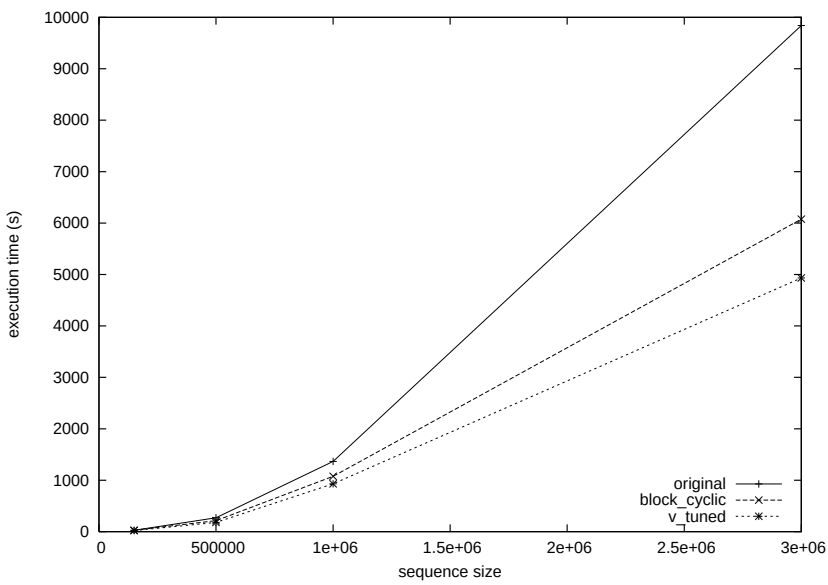


Fig. 17. Comparison between the uniform and block cyclic approaches.

Table 7 shows the values of the parameters used for each comparison. The results presented in this section were obtained with the Cluster-IE-UNB.

We compared the wallclock execution times obtained for the original, the block-cyclic and hand-tuned approaches, when using 64 processors to compare sequences whose size ranged to 150K to 3M, as shown in Table 8. A graphic of these results is shown in Fig. 17.

As long as we augmented the sequence size, better performance gains are obtained with the block-cyclic approach over the original one. The 150Kx150K comparison is the only case where the execution time of the uniform approach is lower than the block-cyclic (Table 8). In the other three comparisons, performance gains are obtained with the block-cyclic approach over the original Z-align parameter values. For the 3Mx3M comparison, the execution time is reduced from 9,841.25s to 6,077.01s, achieving a performance gain of 38.3%. In this case, a reduction of

Table 8. Execution times for the uniform and block cyclic approaches (64 processors).

Comparison	time original (s)	time block cyclic (s)	time hand-tuned (s)
150Kx150K	26.36	26.65	19.92
500Kx500K	269.50	213.66	179.38
1Mx1M	1,365.28	1,074.92	928.85
3Mx3M	9,841.25	6,077.00	4,931.62

more than one hour was obtained, when using the block cyclic tunable work division mechanism. Nevertheless, when we carefully hand-tuned parameter v, we were able to obtain gains over the block-cyclic approach, as can be seen in Table 8.

6. Conclusion and Future Work

In this paper, we evaluated the Z-align parallel exact strategy for the comparison of huge DNA sequences ($\geq$ 1MBP) in two clusters. Due to its memory allocation strategy, Z-align was able to align 23MBPx24MBP sequences. In this case, a speedup of 33.54 was obtained.

When comparing the Z-align results with the ones produced by BLAST, it is clear that Z-align, being an exact method, is able to obtain more significant alignments, with a higher score. This strongly suggests that the use of exact methods can drastically improve the quality of the alignments, being a fundamental tool to help biologist in the task of deciphering the genomic message encoded inside the sequences.

The evaluation of our tunable work distribution mechanism indicates that very good performance gains can be obtained if the appropriate value for each parameter is used, when huge sequences are compared.

As future work, we intend to use Z-align to align human and ape chromosomes (sizes higher than 35MBP) in a massively parallel system. Also, we intend to investigate a way to improve the use of CPU by Z-align in clusters of multi-cores.

References

[1] S. F. Altchul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, Basic local alignment search tool, *J. Comput. Mol. Bio.*. **215(3)**(1990) 403–410.

[2] R. B. Batista, A. Boukerche and A. C. M. A. Melo, A parallel strategy for biological sequence alignment in restricted memory space, *J. Parallel Dist. Comput.*. **68**(2008) 548–561.

[3] A. Boukerche, A. C. M. A. Melo, M. Ayala-Rincon and M. E. M. T. Walter, Parallel Strategies for Local DNA Comparison in a Cluster of Workstations, *4th Int. Workshop on Experimental Algorithms*, **LNCS 3503** (2005) 464–475.

[4] C. Chen and B. Schmidt, Computing large-scale alignments on a multi-cluster *IEEE Int. Conf. on Cluster Comput.* (2003).

[5] A. Driga et al. Fastlsa: a fast, linear-space, parallel and sequential algorithm for sequence alignment. *Int. Conf. Parallel Processing* (2003) 48–56.
[6] J. W. Fickett, Fast optimal alignments. *Nucleic Acids Research*, **12(1)** (1984) 175–179.
[7] O. Gotoh, An improved algorithm for matching biological sequences, *J. Mol. Biology*, **162** (1982) 705–708.
[8] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, **Cambridge University Press** (1997).
[9] D. S. Hirschberg, A linear space algorithm for computing longest common subsequences. *Commun. Assoc. Comput. Mach.* **18(6)** (1975) 341–343.
[10] A. Jacob et al., Whole Genome Comparison on a Network of Workstations, *Int. Symposium on Parallel and Distributed Computing* (2007).
[11] C. Janaki and R. R. Joshi, Accelerating Comparative Genomics using Parallel Computing, *In Silico Biology*, **3(4)** (2003) 429–440.
[12] E. W. Myers and W. Miller, Optimal alignments in linear space, *Computer Applications in the Biosciences*, **4** (1988) 11–17.
[13] G. Navarro, A guided tour to approximate string matching, *ACM Computing Surveys*, **33(1)** (2001) 31–88.
[14] S. B. Needleman and C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Mol. Biology*, **48** (1970) 443–453.
[15] G. Pfister, *In Search of Clusters: The Coming Battle for Lowly Parallel Computing*, **Prentice Hall Inc** (1995).
[16] S. Rajko and S. Aluru, Space and Time Optimal Parallel Sequence Alignments, *IEEE Trans. on Parallel and Dist. Syst.*, **15(2)** (2004) 1070–1081.
[17] T. F. Smith and M. S. Waterman. Identification of Common molecular subsequences, *Journal of Mol. Biol.*, **147(1)** (1981) 195–197.
[18] P. Tang and P. C. Yew. Processor Self-scheduling for multiple parallel nested loops, *Int. Conf. on Parallel Proc.* (1986) 528–535.
[19] F. Zhang, X. Qiao, and Z. Liu. A parallel smith-waterman algorithm based on divide and conquer. *ICA3PP* (2003).
[20] M. Noorian et.al., Performance Enhancement of Smith-Waterman Algorithm Using Hybrid Model: Comparing MPI and Hybrid Programming Paradigm on SMP Clusters, *IEEE Int. Conf. on Systems, Man, and Cybernetics* (2009).