Parallel Optimal Pairwise Biological Sequence Comparison: Algorithms, Platforms, and Classification

EDANS FLAVIUS DE OLIVEIRA SANDES, University of Brasilia, Brazil
AZZEDINE BOUKERCHE, University of Ottawa, Canada
ALBA CRISTINA MAGALHAES ALVES DE MELO, University of Brasilia, Brazil

Many bioinformatics applications, such as the optimal pairwise biological sequence comparison, demand a great quantity of computing resource, thus are excellent candidates to run in high-performance computing (HPC) platforms. In the last two decades, a large number of HPC-based solutions were proposed for this problem that run in different platforms, targeting different types of comparisons with slightly different algorithms and making the comparative analysis of these approaches very difficult. This article proposes a classification of parallel optimal pairwise sequence comparison solutions, in order to highlight their main characteristics in a unified way. We then discuss several HPC-based solutions, including clusters of multicores and accelerators such as Cell Broadband Engines (CellBEs), Field-Programmable Gate Arrays (FPGAs), Graphics Processing Units (GPUs) and Intel Xeon Phi, as well as hybrid solutions, which combine two or more platforms, providing the actual landscape of the main proposals in this area. Finally, we present open questions and perspectives in this research field.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Parallel Programming

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Parallel algorithms, biological sequence comparison, FPGA, GPU, multicores, CellBE, Intel Phi

ACM Reference Format:
Edans Flavius de Oliveira Sandes, Azzedine Boukerche, and Alba Cristina Magalhaes Alves de Melo. 2016. Parallel optimal pairwise biological sequence comparison: Algorithms, platforms, and classification. ACM Comput. Surv. 48, 4, Article 63 (March 2016), 36 pages.
DOI: http://dx.doi.org/10.1145/2893488

1. INTRODUCTION

In recent decades, we have observed many advances in genetics research. Currently, several genomic databases are publicly accessible through the Internet that contain the biological sequences of a myriad of organisms, including those of humans. The analysis of the contents of these sequences can lead to important discoveries concerning the identification of genes that can cause diseases, the identification of regions that can be used to silence genes, and the determination of evolutionary events, among others [Mount 2004]. The fantastic advances that we are witnessing today, and that we will surely continue to witness in the coming decades, would not have occurred without

This work is supported by Capes/Brazil, under grant no. PVE-02/2012.
Authors' addresses: E. F. de Oliveira Sandes and A. C. M. A. de Melo, University of Brasilia (UnB), Department of Computer Science, Campus UNB Asa Norte, Predio CIC/EST, 70910-900, Brasilia-DF, Brazil; A. Boukerche, University of Ottawa, School of Information Technology and Engineering (SITE), 800 King Edward Avenue, Ottawa, Ontario, Canada, K1N 6N5.

tools from bioinformatics. These tools use automated search procedures to unveil the structure and function of the recently discovered sequences.

Pairwise Biological Sequence Comparison aims to compare two sequences, searching for similarities, and is a crucial operation in many bioinformatics tools. Currently, pairwise comparisons are executed thousands of times every day, in research and industrial projects on all continents. The most widely used algorithms that compare two sequences of sizes n and m and provide the optimal result are, respectively, edit distance [Levenshtein 1966], NW [Needleman and Wunsch 1970], SW [Smith and Waterman 1981], Gotoh [1982], and MM [Myers and Miller 1988]. All of these algorithms use dynamic programming and execute in quadratic time ($O(nm)$). Since genomic databases are growing at an exponential rate and DNA sequences can have hundreds of millions of Base Pairs (BP), executing these algorithms can take days or even weeks. For this reason, the BLAST [Altschul et al. 1990] heuristic algorithm was proposed, among others. Even though results are obtained quickly with these algorithms, they do not aim to achieve the optimal result. In this article, we will discuss only optimal pairwise sequence comparison algorithms.

In the last two decades, there have been significant advances in the area of High-Performance Computing (HPC). Due to technological achievements and sophisticated component design, many computers are now able to attain Teraflops (Trillions of Floating-Point Operations per Second), and some even achieve Petaflops (Quadrillions of Floating-Point Operations per Second). In this scenario, HPC solutions make it possible to execute optimal pairwise sequence comparison algorithms in reasonable time; this is the focus of this article.

The research in HPC solutions for optimal pairwise comparison algorithms began in the 1980s, when the first hardware-based solution was proposed by Lipton and Lopresti [1985]. Since then, there are a myriad of papers describing HPC solutions for many different hardware and software platforms, using slightly different algorithms and targeting different types of biological sequences. This diversity presents a challenge when attempting to determine the overall global picture of the research field.

In the literature, some papers provide a review on HPC solutions for bioinformatics applications. A concise review on HPC for three important problems in bioinformatics—pairwise sequence comparison, multiple sequence comparison, and DNA Assembly—can be found in Aluru and Jammula [2014]. In Sarkar et al. [2010b], an overview of HPC solutions is provided for pairwise sequence comparison, multiple sequence comparison, phylogenetic tree reconstruction, and molecular dynamics. Even though these papers provide a good overall picture of the research area, they differ from this survey since they do not provide a uniform categorization of the solutions, nor do they discuss the problem, algorithms, and solutions in detail.

This article aims to organize the main characteristics of HPC solutions for optimal pairwise biological sequence comparison, allowing each solution to be described in a uniform way. We claim that each solution should be classified by considering three distinct aspects: problem, algorithm, and platform. Each aspect should itself be divided into subaspects that will describe the problem targeted, the algorithm used, and the HPC platform technique employed.

In this article, we do not propose a hierarchical categorization of the solutions. Since the problem, algorithm, and platform are highly connected, providing a hierarchical classification would favor one aspect to the detriment of the others. For instance, currently, solutions exist that use GPUs, FPGAs, multicores and/or Intel Xeon Phi, that compare DNA or protein sequences with sizes ranging from hundreds of characters (10^2) to hundreds of millions of characters (10^8), using local or global comparisons with linear- or affine-gap penalties, and a fine or coarse parallelization.

Furthermore, we do not provide an exhaustive list of the HPC solutions proposed so far that tackle the optimal pairwise sequence comparison problem. Since there are

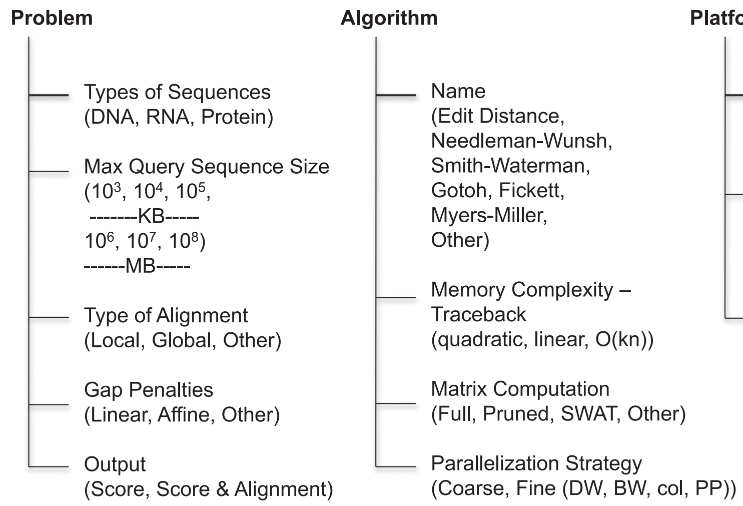
Problem
Types of Sequences (DNA, RNA, Protein)
Max Query Sequence Size (10^3, 10^4, 10^5, -------KB----- 10^6, 10^7, 10^8) ------MB-----
Type of Alignment (Local, Global, Other)
Gap Penalties (Linear, Affine, Other)
Output (Score, Score & Alignment)
Algorithm
Name (Edit Distance, Needleman-Wunsh, Smith-Waterman, Gotoh, Fickett, Myers-Miller, Other)
Memory Complexity – Traceback (quadratic, linear, O(kn))
Matrix Computation (Full, Pruned, SWAT, Other)
Parallelization Strategy (Coarse, Fine (DW, BW, col, PP))
Platform
Processing Unit (FPGA, CellBE, GPU, Intel Phi, CPU, Other)
Number of Processing Units (Single, Multiple Hybrid, Multiple Homogeneous, Multiple Heterogeneous)
Performance Measure (GCUPS)

Fig. 1. Proposed classification of parallel optimal pairwise sequence comparison solutions. Each solution is organized into 3 parts: problem, algorithm, and platform.

more than 100 papers in the literature that deal with this problem, it is impractical to discuss them all. Nevertheless, we do cover the major advances in more than 20 years in this field by discussing 34 papers, describing them in a uniform way. We believe that this selection of papers is sufficient to provide a very detailed landscape of the area and to draw comments about the future. Moreover, in order to reduce the scope of this survey, we did not include manycore System-on-Chip (SoC) solutions, such as those using Tilera64 [Esteban et al. 2013] and Supercomputing-on-Chip (SCoC) [Smith and Frenzel 2003]. Finally, in this article, we do not discuss HPC solutions for problems such as Longest Common Subsequence (LCS) [Chvatal and Sankoff 1975], sequence-profile comparisons [Eddy 1998], and DIALIGN alignments [Morgenstern et al. 1998], all of which are closely related, but not based on optimal pairwise sequence comparison.

The remainder of this article is organized as follows. In Section 2, we provide a general overview of our flat categorization. In Sections 3, 4, and 5, we explain in detail each aspect of our categorization (Problem, Algorithm, and Platform). Sections 6 to 10 discuss and compare in detail several solutions to the optimal pairwise comparison problem that use CPUs, FPGAs, CellBEs, GPUs, and Intel Xeon Phi, respectively. In Section 11, we discuss solutions for hybrid platforms. A global overview of all the proposals discussed in this survey is presented in Section 12. Section 13 concludes the article and outlines open research in this field.

2. MAIN CHARACTERISTICS OF BIOLOGICAL SEQUENCE COMPARISON APPROACHES IN HPC PLATFORMS

A strategy proposed in the literature to tackle the Optimal Pairwise Biological Sequence Comparison problem can be divided into three main aspects: the definition of the problem, the description of the algorithm used to solve the problem, and the HPC platform that implements the solution, as shown in Figure 1.

In the problem definition, researchers state the scope and some restrictions applied to the problem. This aspect is divided into 5 subaspects: types of sequences compared, maximum size of the query sequence, type of alignment, gap penalties, and output provided. A clear definition of the problem is very important for the biologists, since they will use it to correctly choose which approach is more appropriate to their particular application. For instance, if both the score and the alignment are required as output, the approaches that provide only the score should be discarded. In addition, if protein sequences of 10^4 amino acids are to be compared, solutions that compare query sequences with at most 10^3 amino acids should be rejected.

Having defined the problem to be solved, the next step is to choose the appropriate algorithm. This aspect is divided into 4 subparts: the algorithm itself, the memory complexity of the traceback phase, the overall matrix computation, and the parallelization strategy. Of course, the problem definition will limit the type of algorithm used. For example, if a local alignment is required, the Needleman-Wunsh algorithm, which calculates global alignments, must not be used. Also, if Megabase (10^6, 10^7, 10^8) DNA sequences are compared and the alignment is required, a linear memory complexity algorithm must be chosen.

The last aspect of our flat categorization is the targeted HPC platform and solution, which is divided into the following 3 subaspects: type of processing unit, number of processing units, and performance. The first two subaspects refer to the HPC infrastructure; the last subaspect provides a way to compare, in a broad sense, the performance of the proposed solutions presented in the referenced papers.

In Sections 3, 4, and 5, we discuss each subaspect of our classification in detail.

3. PROPOSED CATEGORIZATION: PROBLEM

In this section, we discuss the main characteristics of the Pairwise Biological Sequence Comparison problem (Figure 1): types of sequences and their sizes, types of alignments and score, gap penalties, and output provided. These characteristics have a considerable impact on the performance and/or quality of the results produced, and will be detailed in Sections 3.1 to 3.5.

3.1. Types of Sequences

Biological sequences are composed of an ordered set of residues, which can be either nucleotides (DNA/RNA sequences) or amino acids (protein sequences) [Mount 2004].

DNA sequences are composed of 4 nucleotides: A-Adenine, T-Thymine, G-Guanine, C-Cytosine. RNA sequences are also composed of 4 nucleotides, with U-Uracil instead of T-Thymine.

Protein sequences are composed of 20 amino acids: Y-Tyrosine, W-Tryptophan, V-Valine, T-Threonine, S-Serine, R-Arginine, Q-Glutamine, P-Proline, N-Asparagine, M-Methionine, L-Leucine, K-Lysine, I-Isoleucine, H-Histidine, G-Glycine, F-Phenylalanine, E-Glutamic acid, D-Aspartic acid, C-Cysteine and A-Alanine [Mount 2004].

3.2. Maximum Query Sequence Size

Proteins are sequences of amino acids with sizes ranging from very few to the tens of thousands. Publicly available proteins are maintained in the UniProtKB/Swiss-Prot public protein knowledge base [ExPASy 2014a], which is accessed daily thousands of times all over the world. TrEMBL is another important database that has about 50 million unannotated sequences.

In its release note 2014_09, October 1, 2014 [ExPASy 2014b], the UniProtKB/Swiss-Prot contains 546,439 sequence entries, comprising 194,445,396 amino acids. The smallest sequence in this release (GWA_SEOF—P83570) has 2 amino acids and the longest sequence (TITIN_MOUSE—A2ASS6) has 35,213 amino acids. The average sequence size is 355 amino acids.

There are many types of RNAs in nature, with different ranges of size. MicroRNAs (miRNA) have less than 100 nucleotides, whereas Ribosomal RNAs (rRNAs) can have up to a few thousand nucleotides.

DNA sequences can be much longer than protein and RNA sequences. Their size ranges from a few to hundreds of millions of BPs. Publicly available nucleotide sequences are kept in the NCBI Nucleotide genomic database [nuccore 2014], which is also frequently accessed. In its release note 206.0 (February 17, 2015), *nuccore* contains 181,336,445 DNA sequences with 187,893,826,750 BPs. In February 2015, the longest sequence in this database is the chromosome 3B of *Triticum aestivium*, accession number HG670306.1, which has 774,434,471 BPs[1]. *Triticum aestivium* is a type of bread wheat cultivated in China, which has 7 chromosomes [Mayer et al. 2014]. The longest human sequence in the same database is chromosome 1, accession number NC_000001, which has 248,956,422 BPs.

To summarize, the sizes of protein/RNA sequences range from 10^0 to 10^5 (tens of thousands of amino acids/nucleotides) and the sizes of DNA sequences range from 10^0 to 10^8 (hundreds of millions of nucleotides).

In this survey, we will use the name *query sequence* to denote the smallest sequence in a given pairwise comparison for DNA or RNA sequences. In the protein sequence comparison, one sequence is usually compared to a database, which is composed of many protein sequences. In this case, the *query sequence* is the sequence compared to the sequences that belong to the genomic database (*database sequences*).

3.3. Type of Alignment and Score

3.3.1. Global and Local Alignments. The result of a pairwise sequence comparison operation is a score, which represents the similarity between the sequences, and an alignment, which allows the visualization of the correspondence between similar characters [Mount 2004].

In an alignment, the sequences/subsequences must have the same size. To accomplish this, spaces (gaps) can be added to the sequences.

The basic types of alignments are global and local. In a global alignment, all the characters of the sequences must be present. A local alignment contains only a subset of the characters of the sequences. Global alignments are useful when the biologists want to determine the total similarity of closely related sequences. Local alignments, on the other hand, are used to decide which parts of the two sequences are much alike. Both global and local alignments are used in the comparison of close homologs, and local alignments are also used to compare protein sequences to a database.

In order to measure the quality of an alignment between two sequences S_0 and S_1, a score must be calculated, considering three cases: (a) matches, (b) mismatches, and (c) gaps. If the characters of both sequences at the same column are identical (match), the punctuation *ma* is assigned. If the characters in the same column are different (mismatch), the penalty *mi* is used. Finally, if one of the characters in the same column is a gap, the penalty *g* is assigned. The score is obtained by adding all the values assigned to the columns. A high score indicates that the sequences have high similarity. Figures 2(a) and 2(b) present one example of global and local alignment between two DNA sequences (S_0 = ACTTGTCCG and S_1 = ATGTCAG).

3.3.2. Semi-Global Alignments. Another important type of alignment is the semi-global alignment. However, unlike global and local alignments, semi-global alignments are defined in many different ways in the literature.

In a broad sense, a semi-global alignment should ignore gap penalties in the head and/or tail of the sequences. In practice, different papers use different definitions based on this broad notion. For instance, in Durbin et al. [1998], the semi-global alignment

[1]This information was obtained with the following Entrez query to the nuccore database: *500000000:1500000000[Sequence Length] NOT "srcdb refseq"[Properties].*

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

(a) global alignment

$$\underbrace{\begin{array}{cccccc} T & G & T & C & C & G \\ T & G & T & C & A & G \\ \hline +1 & +1 & +1 & +1 & -1 & +1 \end{array}}_{score\,=\,4}$$

(b) local alignment

Fig. 2. Global and local alignments and scores. The punctuations for matches, mismatches, and gaps are $+1$, -1, and -2, respectively.

is defined to start in the first character of one of the sequences and to end at the last character of the other sequence, whereas in Liu and Schmidt [2014a], the semi-global alignment must begin and end, respectively, in the first and last character of the same sequence. In this survey, we will classify an alignment as *semi-global* if the paper in consideration uses the term *semi-global* to classify it. Semi-global alignments are often used when the sequences compared have very different lengths, as in DNA genome assembly applications.

3.3.3. Match/Mismatch Scoring Schemes. In order to calculate the similarity score between two sequences, we need to assign punctuations for matches and mismatches. These punctuations have a high influence on the biological significance of the output produced; for this reason, they should be carefully chosen.

The simplest way to assign punctuations for matches and mismatches is to provide a unique value, as shown in Figure 2. In this example, the value $+1$ is assigned for matches and the value -1 is assigned for mismatches. A particular case of this single-value assignment scheme is the edit distance (Section 4.5), which further simplifies the problem by assigning only nonnegative values for matches and mismatches.

More elaborate schemes assign one value for each possible match/mismatch pair, using substitution matrices of size 4x4 (DNA or RNA) or 20x20 (proteins). Even though some works on DNA sequence comparison use 4x4 substitution matrices, usually the single-value assignment is preferred. Nevertheless, for protein comparison, substitution matrices are frequently used. Percent Accepted Mutations (PAM) and Blocks Substitution Matrix (BLOSUM) are two very popular substitution matrices. These matrices assign a potentially different value for each match/mismatch pair. A high value indicates that the substitution of amino acid x by amino acid y is frequent, considering evolutionary events [Mount 2004]. For instance, in the BLOSUM62 substitution matrix, the punctuation for a mismatch between Q-Glutamine and R-Arginine is 1, whereas the mismatch between Q-Glutamine and I-Isoleucine is -3.

3.4. Gap Penalties

There are many types of gap penalties. The most frequently used are the linear-gap and the affine-gap models, which are discussed in this section. For a more detailed discussion on gap penalty models, see Gusfield [1997].

3.4.1. Linear Gaps. With the linear-gap model, all gaps receive the same penalty. The cost of a sequence of gaps of length x in this model is $\gamma(x) = x * g$, where g is the gap penalty. In Figure 2(a), a linear gap penalty is used, since all gaps receive the same penalty (-2) and the cost of a sequence of two gaps is -4.

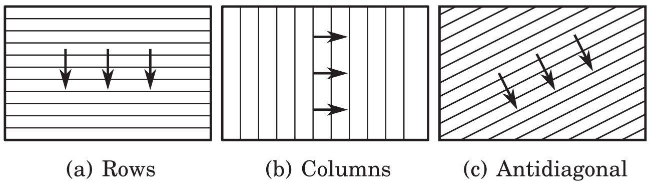
(a) Rows (b) Columns (c) Antidiagonal

Fig. 3. Different ways to calculate the DP matrices.

3.4.2. Affine Gap. In nature, gaps tend to occur together. This means that it is more common to have a sequence of gaps than isolated ones. The model that takes this observation into account is called the *affine-gap model* [Gotoh 1982]. In this case, a higher penalty is assigned for opening a gap (G_{open}) than for extending it (G_{extend}). The cost of a sequence of gaps of length x in the affine-gap model is $\gamma(x) = G_{open} + (x-1) * G_{extend}$. For instance, in Figure 2(a), the affine-gap model could assign $G_{open} = -2$ and $G_{extend} = -1$ and the cost of a sequence of 2 gaps would be -3. Alternatively, we can set the penalty for the first gap (G_{first}) as G_{open} added to one G_{extend} ($G_{first} = G_{open} + G_{extend}$).

3.5. Output of the Comparison

In many cases, biologists are interested only in the similarity between sequences, which is measured by the optimal score. Using this metric, biologists can discard sequences that have a score below a given threshold.

When a more detailed analysis is requested, both the optimal score and alignment should be provided. The optimal alignment is important when the biologists want to visualize the regions of similarity and the differences between sequences. These regions can indicate evolution-related events, and can also be used to infer properties.

4. PROPOSED CATEGORIZATION: ALGORITHM

4.1. Overview

Optimal Pairwise Biological Sequence Comparison algorithms calculate one or more Dynamic Programming (DP) matrices, using recurrence relations expressed as maximum or minimum operations on previously calculated DP values. Basically, the DP matrices can be calculated by row, column, or antidiagonal, as shown in Figure 3.

ALGORITHM 1: Skeleton to Compute DP Matrices

```
1: procedure ComputeDPMatrix($S_0$, $S_1$)
2:     InitializeFirstRow($H[0, 0..m]$)
3:     InitializeFirstColumn($H[0..n, 0]$)
       for $i \leftarrow 1, m$ do
         for $j \leftarrow 1, n$ do
4:           $H[i, j]$ := RecurrenceRelation($i$, $j$, $S_0[i]$, $S_1[j]$);
         end
       end
5:     best:=RetrieveOptimalScore();
6:     return best
7: end procedure
```

In Algorithm 1, we present a skeleton that can be used to compute the DP matrix row by row, obtaining the optimal score as a result. The procedure *ComputeDPMatrix* receives two sequences as input (S_0 and S_1), with sizes $|S_0| = m$ and $|S_1| = n$. Lines 1 and 2 execute procedures to initialize the first row and the first column, respectively. Then, every cell (i, j) in the DP matrix is calculated with the recurrence relation of the

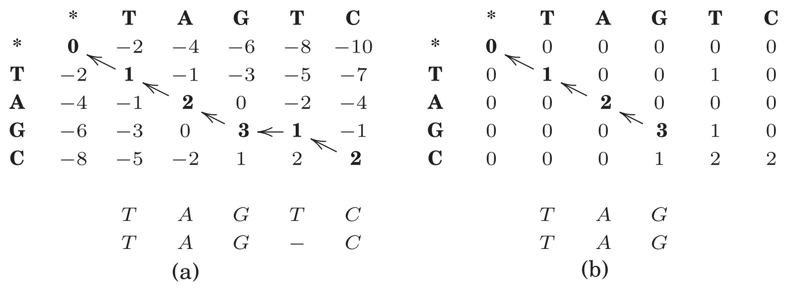

Fig. 4. DP matrices and alignments for S_0 and S_1 (mi = −1, ma = +1, g = −2). (a) NW matrix and optimal global alignment with score = 2; (b) SW matrix and optimal local alignment with score = 3.

sequence comparison algorithm (Line 4). The optimal score is retrieved in Line 5. If a global comparison is made, the optimal score is found in $H[m, n]$. In the case of a local comparison, the optimal score is the maximum value in H, and is updated while the recurrence relation is calculated (Line 4).

In the remainder of this section, we discuss the second aspect of our categorization (Figure 1), including the algorithms and techniques to reduce the memory complexity of the traceback stage, as well as techniques to accelerate the entire computation.

4.2. Global Alignment: Needleman-Wunsh (NW)

Needleman and Wunsch [1970] proposed an algorithm based on DP that retrieves the optimal global alignment in $O(mn)$ space and time. NW is executed in two phases: (a) calculate the DP matrix and (b) retrieve the alignment (traceback).

4.2.1. Phase 1: Calculate the DP Matrix. The DP matrix H is calculated as follows. The first row and column of the DP matrix are filled with $H_{i,0} = i * g$ and $H_{0,j} = j * g$, where g is the gap penalty and i and j represent the sizes of the nonempty sequences in the beginning of the alignment (Lines 1 and 2 in Algorithm 1). The remaining cells are calculated with the recurrence relation (Line 4 in Algorithm 1) expressed by Equation (1) [Needleman and Wunsch 1970].

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

In this equation, if DNA or RNA sequences are compared, $p(i, j)$ is the match punctuation (ma) if $S_0[i] = S_1[j]$ or the mismatch punctuation (mi), otherwise. If amino acid sequences are compared, $p(i, j)$ is given by a 20x20 substitution matrix, as discussed in Section 3.3.3. Each cell keeps an indication of which cell ($H_{i-1,j-1}$), ($H_{i,j-1}$), or ($H_{i-1,j}$) was used to produce the value of cell $H_{i,j}$ (arrows in Figure 4). The optimal score is the value in cell $H_{m,n}$.

4.2.2. Phase 2: Retrieve the Alignment (Traceback). To produce the alignment, the traceback procedure is executed from the bottom right cell in the DP matrix, following the indications until the top-left cell is attained.

Figure 4(a) illustrates the DP matrix calculated by NW. In this figure, the optimal score is 2 and the optimal global alignment, obtained in the traceback phase, is shown below the DP matrix.

4.3. Local Alignment: Smith-Waterman (SW)

When biologists are interested in calculating the similarity between the fragments of sequences, local alignment is usually applied. The SW algorithm [Smith and Waterman

1981] is a popular algorithm that calculates optimal local alignments. SW uses dynamic programming, has the same complexity as NW, and executes in two phases, as explained in Section 4.2. Nevertheless, NW and SW are distinct in three ways.

First, unlike NW, the elements of the first row and column of the SW matrix are set to zero (Lines 1 and 2 in Algorithm 1). This type of initialization prevents gaps from receiving a penalty if they are at the beginning of the alignment. In other words, a new local alignment can start at any point in the DP matrix. Second, the SW recurrence relation is slightly different from the NW recurrence relation, since no negative values are allowed in SW, as shown in Equation (2) [Smith and Waterman 1981], which is used in Line 4 of Algorithm 1. Finally, the cell that contains the optimal local score (*best* in Algorithm 1) is the cell $H_{i,j}$, which has the highest value in the DP matrix. This cell will be used to start the second phase of the algorithm (traceback). In the traceback phase, the SW algorithm starts from cell $H_{i,j}$, following the indications until a cell is found with a value of zero.

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + p(i,j) \\ H_{i,j-1} - g \\ H_{i-1,j} - g0 \end{cases} \tag{2}$$

Figure 4(b) illustrates the DP matrix calculated by SW. In this figure, the optimal score is 3 and the optimal local alignment is shown below the DP matrix.

4.4. Affine Gap: Gotoh

NW (Section 4.2) and SW (Section 4.3) use the linear-gap model. To produce more biologically relevant results, Gotoh [1982] proposed an algorithm that implements the affine-gap model (Section 3.4). The Gotoh algorithm calculates three DP matrices: H, E, and F, where H keeps track of matches and mismatches and E and F keep track of gaps in each sequence.

In Gotoh's algorithm, the recurrence relation (Line 4 in Algorithm 1) involves 3 DP matrices. Matrix H is calculated as shown in Equation (3); matrices E and F are calculated with Equations (4) and (5), respectively. Even within this algorithm, time and space complexities remain quadratic. To do the traceback, the arrows are followed in matrix H when a match/mismatch occurs, or in matrices E and F when there is a sequence of gaps.

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

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

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

4.5. Global Alignment: Edit Distance

The edit distance was proposed by Levenshtein [1966] and measures the minimum amount of operations (insertions, deletions, substitutions) required to convert Sequence A into Sequence B. In bioinformatics, the edit distance can be used as a simplified method of detecting homologous sequences.

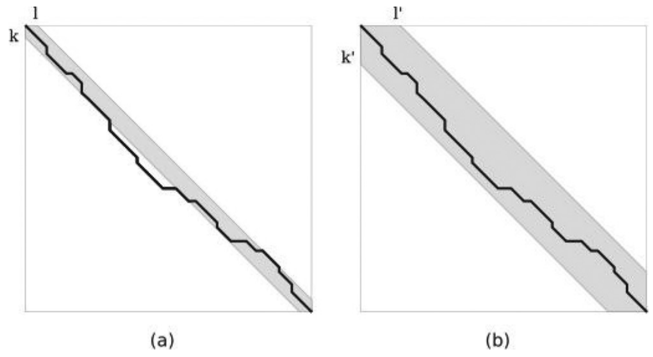

Fig. 5. Fickett's algorithm [Boukerche et al. 2012]. In (a), the estimated band is too narrow. In (b), the band is enlarged and the alignment is obtained.

Having Sequences S_0 and S_1 as inputs, the edit distance algorithm is executed in two phases. In Phase 1, the first row and column are initialized with $H[i, 0] = i$ and $H[0, j] = j$ (Lines 1 and 2 in Algorithm 1). The other cells of the DP matrix are calculated with the recurrence relation expressed by Equation (6) [Levenshtein 1966], which is used in Line 4 of Algorithm 1. In Equation (6), $p(i, j) = 0$ if $S_0[i] = S_1[j]$ or $p(i, j) = 1$, otherwise.

$$H_{i,j} = \min \begin{cases} H_{i-1,j-1} + p(i, j) \\ H_{i,j-1} + 1 \\ H_{i-1,j} + 1 \end{cases} \tag{6}$$

To obtain the alignment, the traceback procedure is executed in Phase 2 from the bottom right cell, following the arrows until the top-left cell is attained.

4.6. *O*(*kn*) Space: Fickett

Fickett [1984] proposed an algorithm for the global alignment problem, which can be executed quickly if the sequences compared are very similar. In this case, the alignment between the sequences is confined in a small region near the main diagonal of the DP matrix. Thus, Fickett's algorithm calculates and stores only a small set of diagonals near the main diagonal (known as the k-band). Therefore, time and space complexity of this algorithm is $O(kn)$ [Fickett 1984].

Considering a heuristic measurement of the similarity of the sequences, a k-band is estimated. Within this k-band, the DP matrix is computed and stored. The optimal score is contained in cell $H_{n,m}$, and is used to do the traceback over the band. If the k-band was underestimated, the global alignment cannot be retrieved (Figure 5(a)). In this case, the algorithm enlarges the k-band and the DP matrix is calculated for the new k-band, until the entire alignment has been obtained (Figure 5(b)). In Algorithm 1, the second *for* is replaced by *for* $j \leftarrow i - k, i + k$.

4.7. Linear Space: Myers-Miller (MM)

When long biological sequences are compared, linear-space algorithms should be considered. One of the first linear-space algorithms for optimal pairwise global sequence comparison was proposed by Hirschberg [1975]. Hirschberg's algorithm calculates the DP matrix from the beginning to the middle row (i^*), storing only the last calculated row, thus in linear space. After that, it calculates the DP matrix from the end to the middle row (i^*), in linear space, considering the reverses of the sequences. At this point, there are two middle rows; one was calculated with the original sequences and

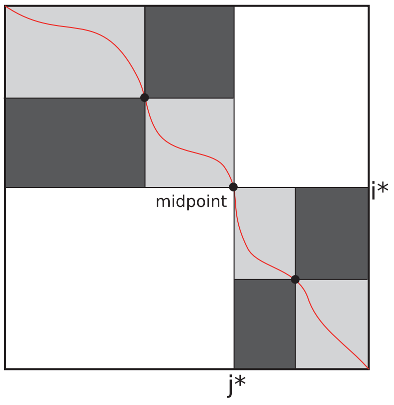

Fig. 6. Recursive splitting procedure in MM [Sandes and Melo 2013]. In the first recursion, the entire matrix is processed, and one midpoint is obtained. In the second recursion, the top-left and bottom-right parts of the matrix are processed and two additional midpoints are obtained.

the other was calculated with the reverses of the sequences. Hirshberg proved that the position where the addition of the corresponding values in these two middle rows is maximum belongs to the optimal alignment. This point is known as the midpoint, and divides the problem into two smaller subproblems, which are processed recursively, until trivial solutions are found.

Myers and Miller [1988] (MM) adapted Hirschberg's algorithm to the affine-gap model (Section 4.4), considering additional situations that handle a sequence of gaps. For that, two additional vectors are used.

In Algorithm 1, the first step of MM is executed by two sets of *for* statements: $(i \leftarrow 1, i^*;\ j \leftarrow 1, m)$ and $(i \leftarrow n, i^*;\ j \leftarrow m, 1)$. At the end of the first step, the midrows and vectors are added in order to find the midpoint. The midpoint occurs in cell (i^*, j^*), where j^* is the position in which the addition is maximum.

As in Hirschberg, when the midpoint is found, smaller subproblems are recursively processed (Figure 6), executing *for* loops accordingly.

4.7.1. GU Variation. In Guan and Uberbacher [1994], the MM algorithm was implemented using an additional matrix that stores the column where the optimal alignment occurs in each middle row. With this, the midpoint is accessed in constant time, accelerating the computation of the alignment. This algorithm was adapted to use dividing rows in addition to the columns. With this technique, the problem can be divided into independent subproblems, further accelerating the computation. The main drawback of this technique is that it requires additional memory. Another extension of this algorithm is FastLSA, which uses r dividing rows and maintains the CROSS matrix only for these rows [Driga et al. 2006].

4.8. Memory Complexity of the Traceback Stage

If the solution only outputs the score, it can run in linear memory, executing the first phase of the algorithms NW, SW, Gotoh, and ED, explained in Sections 4.2 to 4.5. In this case, it is sufficient to store only the last row, last column, or the last two diagonals computed so far, depending on the way that the DP matrices are calculated (Figure 3).

However, if alignment is also required, the algorithms mentioned earlier will run in quadratic space, imposing a great restriction in the sizes of the sequences. Therefore, if the output is the alignment, solutions that have linear memory complexity (Section 4.7) or $O(kn)$ memory (Section 4.6) are preferred.

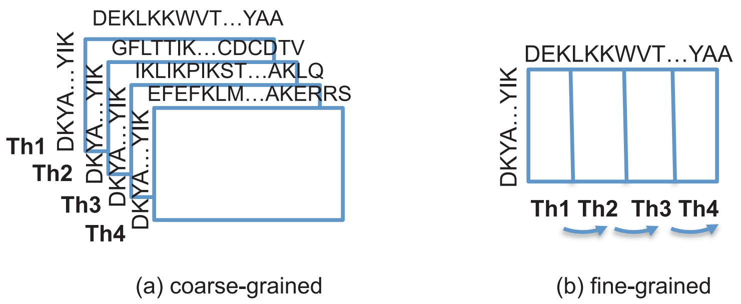

Fig. 7. Strategies to parallelize the SW Algorithm [Mendonca and Melo 2013]. In (a), each thread computes a different DP matrix and in (b) the same DP matrix is computed by several threads.

4.9. Types of Matrix Computation

Most solutions in the literature compute the entire DP matrix (Full Computation). However, some techniques have been proposed that avoid calculating DP cells that will not contribute to the optimal score/alignment. In this section, we present two optimizations: SWAT and block pruning.

4.9.1. SWAT. SWAT optimization was proposed by Green [1993] and is applied to the affine-gap model (Section 4.4). This technique is based on the observation that matrix F (Equation (5)) usually contains zero-valued cells, which do not contribute to the computation of matrix H (Equation (3)). With this, the main matrix (H) needs to access only matrix F when the corresponding value is higher than the penalty for opening and extending one gap (G_{first} in Section 3.4.2). Breaking this dependency, the computation of each column of H can be done in parallel and matrix F is calculated only when it is necessary, saving a great deal of processing time. This optimization is very effective if the value G_{first} is not too small.

4.9.2. Block Pruning. Block pruning was proposed by Sandes and Melo [2013]. Its goal is "to eliminate the calculation of blocks of cells that surely do not belong to the optimal alignment. These blocks have such a small score that it is not mathematically possible to lead to a score higher than a score that has already been produced" [Sandes and Melo 2013]. Therefore, before calculating a block of cells b, a pruning test is made. If the test returns true, block b is not calculated, accelerating the computation of the DP matrix. This optimization is effective if the sequences have high similarity.

4.10. Parallelization Strategy

Basically, the parallelization of the Biological Sequence Comparison algorithms that provide the optimal result can either be coarse-grained or fine-grained.

4.10.1. Coarse-Grained Parallelization. In coarse-grained parallelization, each computing thread receives the query sequence q_1 and some database sequences d. The threads then calculate the DP matrices for q_1 x d, with no communication (Figure 7(a)). In this case, since each DP matrix is calculated independently by a different computing thread, load-balancing strategies must be used. To achieve this, the sequences that belong to the database are usually sorted by size and distributed to the computing threads accordingly.

4.10.2. Fine-Grained Parallelization. In the fine-grained approach, the comparison of one query sequence and one database sequence (i.e., a single DP matrix) is made by several computing threads, as shown in Figure 7(b). In this article, we will consider four types of fine-grained parallelization, which are described here.

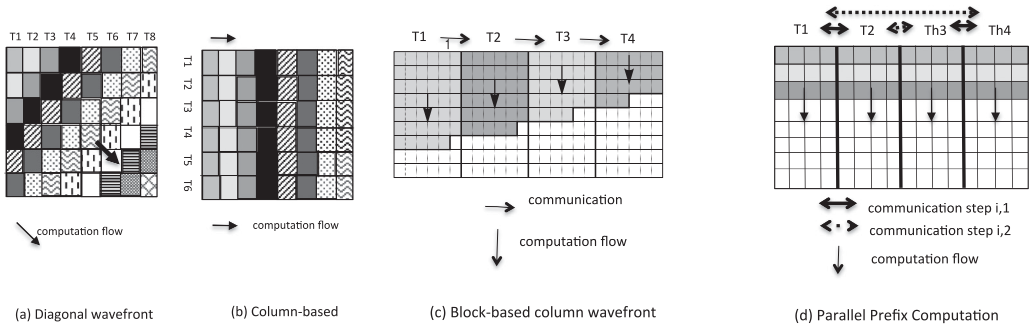

Fig. 8. Types of fine-grained computation. Adapted from Sandes and Melo [2013] and Sandes et al. [2014].

Diagonal Wavefront (DW). In the algorithms explained in Sections 4.2 to 4.7, each cell (i, j) depends on $(i-1, j)$, $(i-1, j-1)$, and $(i, j-1)$ (Equations (1) to (6)). Respecting this data dependency, we can easily see that each antidiagonal of the DP matrix can be calculated in parallel. This leads to a nonuniform parallelism, known as diagonal wavefront (DW). At the beginning, only the first cell (1, 1) is calculated by *Th1* (first antidiagonal). After that, cells (1, 2) and (2, 1) can be calculated in parallel by threads *Th1* and *Th2*. Then, cells (1, 3), (2, 2), and (3, 1) can be calculated in parallel by *Th1*, *Th2*, and *Th3*, and so on. The maximum parallelism (6 threads in this example) is attained in antidiagonals 6, 7, and 8. The parallelism then decreases until the last antidiagonal (cell (n, m)) is calculated by thread *Th8*. Figure 8(a) shows the computation of a 6x8 DP matrix, with the DW strategy.

Column-Based (Col). In this type of parallelization, the threads will calculate the same column of the DP matrix in parallel. Note that this will violate data dependencies of the algorithms explained in Sections 4.2 to 4.7. Nevertheless, if the SWAT optimization (Section 4.9.1) is used, the columns can be calculated in parallel. However, if cells of the F matrix need to be calculated (i.e., are above the threshold), a recomputation phase is executed. Column-Based parallelism is illustrated in Figure 8(b).

Block-Based Wavefront (BW). In BW, each thread will calculate a subset of columns of the DP matrix, as shown in Figure 8(c). At the beginning, the only thread that can compute is *Th1*. When thread *Th1* finishes calculating the values of a block of matrix cells, it sends its border column element to thread *Th2*, which can begin calculating. At this moment, *Th1* and *Th2* are calculating in parallel. In this technique, only one thread (*Th4*) is active at the end of the matrix computation.

Parallel Prefix (PP). Parallel Prefix computations can be used to break the dependency between neighboring threads, allowing all threads to start computing the same row in parallel. This technique works as follows [Rajko and Aluru 2004]. The DP matrix is partitioned into n/t pieces, where t is the number of threads. At each iteration i, t threads calculate their cells in parallel with a modified equation that computes local prefix maxima. At the end of each iteration, $O(\log t)$ communication steps are executed to obtain the global prefix maximum. In order to accelerate the traceback stage, each thread saves all the entries from its last column, with an adapted version of the GU optimization (Section 4.7.1). Figure 8(d) illustrates parallel prefix computation with 4 threads.

5. PROPOSED CATEGORIZATION: HPC SOLUTION

5.1. Processing Unit

The HPC solutions proposed in the literature for optimal pairwise biological sequence comparison use standard CPUs or accelerators such as FPGAs, GPUs, CellBEs, and

Intel Xeon Phis as the processing unit (PU). In the subsequent sections, we will give a brief overview of each of these PUs.

5.1.1. CPUs. Intel released its first 4-bit microprocessor 4004 in 1971. Since then, the design of microprocessors that compose general-purpose CPUs has undergone fantastic technological evolutions and challenges. Until 2005, most chips had only one processor, experimenting exponentially increasing clock rates. When the frequency wall was hit, researchers needed to continue providing increasing performance to the chips, spawning the multicore era. Currently, the most popular general-purpose processor microarchitectures—Intel Haswell and AMD Bulldozer—have 4 cores and 8 cores, respectively.

In the late 1990s, multimedia-oriented instructions were added to general-purpose CPUs. These Single-Instruction Multiple-Data (SIMD) instructions operate on a wide register, executing the same vector operation on multiple operands at once. Some examples of vector instructions are (a) Intel MMX (MultiMedia eXtensions), SSE (Streaming SIMD Extensions) and AVX (Advanced Vector eXtensions); (b) Sun VIS (Visual Instruction Set); and (c) Altivec (Motorola). More details about the evolution of microprocessors can be found in Borkar and Chien [2011].

5.1.2. FPGA. Field-Programmable Gate Arrays (FPGAs), proposed in the 1980s, are integrated circuits which can be programmed by the end user to realize different designs [Compton and Hauck 2002], including application-specific accelerators. Most of the FPGAs are organized as programmable blocks composed of registers, logic elements, and interconnections. The logic elements are usually Lookup Tables (LUTs), which are basically a memory that can be programmed to compute any function with up to n inputs (n-input LUT). The programmable blocks are usually organized as a grid, and the interconnections can also be configured. FPGAs are programmed with Hardware Description Languages (HDL) such as VHDL and Verilog. More details about the FPGAs can be found in Wolf [2004].

5.1.3. CellBE. The Cell Broadband Engine (CellBE) architecture was proposed in 2001 by a consortium formed by IBM, Sony, and Toshiba. The Cell processor was used in the Sony Playstation 3 and in supercomputers. The IBM Roadrunner was the fastest supercomputer in the world from June 2008 to November 2009, with 12,960 PowerXCell 8i accelerators. In the beginning of 2010, the CellBE was discontinued by IBM.

The CellBE is a heterogeneous multicore architecture composed of one PowerPC Processing Element (PPE) and eight Synergistic Processing Elements (SPEs), as well as memory and interrupt controllers, interconnected using the Element Interconnect Bus (EIB) [IBM 2007]. The SPEs are composed of the Synergistic Processing Units (SPUs), which have SIMD capability. Each SPE has 128b vector registers and 256KB of local memory. It must be noted that the SPEs do not have support for saturation arithmetic. Direct Memory Accesses (DMAs) are used to transfer data between two elements. In order to program CellBEs, the IBM CellBE SDK is used. More details about the CellBE architecture can be found in IBM [2007].

5.1.4. GPUs. Graphics Processing Units (GPUs) are highly parallel architectures that execute data-parallel applications, usually at a much faster rate than a CPU [Kirk and Hwu 2010]. They were proposed in the early 1980s to accelerate graphics-rendering applications and have experienced an astonishing development since then. The first GPUs had a fixed-function pipeline, which could not be programmed. In the mid-1980s, a more flexible pipeline was proposed, in which some functions, called shaders, could be programmed. The shaders allowed programmers to produce and run code in GPUs that did not execute computer graphics functions, extending the use of GPUs and creating the General-Purpose GPUs (GPGPUs). GPUs are often called stream processors, capable of executing vector instructions in a lock-step basis, following the

Single-Instruction Multiple-Thread (SIMT) model. In order to produce an optimized code for GPUs, programmers must (a) avoid branch divergence, reducing the number of "IF" statements; (b) carefully place the data over the memory hierarchy of the GPU; and (c) explore maximum parallelism, making sure that all threads are computing most of the time.

Currently, the market for GPU cards is concentrated in Intel, NVIDIA, and AMD. NVIDIA and AMD GPUs can attain Teraflops performance. For instance, the NVIDIA K40 has 2,880 cores and 12GB of RAM memory, and can achieve 4.29 Teraflops. NVIDIA cards are usually programmed with Compute Unified Device Architecture (CUDA) [NVIDIA 2015], and AMD cards are usually programmed with OpenCL [Munshi et al. 2011]. The current second fastest supercomputer in the world (*www.top500.org* in November 2014) is Titan, which has 18,688 NVIDIA Tesla K20X GPUs. More details about GPUs can be found in Kirk and Hwu [2010].

5.1.5. Intel Xeon Phi. Intel MIC (Many Integrated Core) is an architecture developed by Intel. The processor family that implements this architecture is the Intel Xeon Phi, which was made available in 2012. The Intel Xeon Phi is known to Intel as a coprocessor; in this article, however, it will be categorized with the accelerators. As GPUs and FPGAs, it is connected to a host computer by the I/O subsystem (PCIe). The Intel Xeon Phi is composed of Intel Pentium processing cores, cache and memory controllers connected through a high-speed bidirectional ring. Each processor has a vector processing unit with 512b registers, which are programmed with SIMD instructions. The Intel Xeon Phi 7120P (61 cores) can attain 2.14 Teraflops. The current fastest supercomputer in the world (November 2014) is the Tianhe-2, which has 52,000 Intel Phi accelerators. More details about the Intel Xeon Phi can be found in Jeffers and Reinders [2013].

5.2. Performance: CUPS

To measure the performance of Biological Sequence Comparision algorithms, the metric Cells Updated per Second (CUPS) is often used. This metric calculates how many of the DP's cells are calculated per second. Even though the performance metric is CUPS, most proposals in the literature achieve at least Millions of Cells Updated per Second (MCUPS), and some achieve Trillions of Cells Updated per Second (TCUPS). Currently, most of the proposals achieve Billions of Cell Updated per Second (GCUPS), which is calculated by $(m \times n)/(time \times 10^9)$, dividing the whole size of the DP matrix by the time needed to calculate it. When a query sequence is compared to a genomic database, the size of the query sequence (m) and the size of the genomic database (n) are used. When two DNA or RNA sequences (S_0 and S_1) are compared, m and n are the sizes of sequences S_0 and S_1.

In this survey, we decided to consider only the GCUPS obtained in experimental executions described in the papers as the performance metric. In some papers, experimental GCUPS are not explicitly provided. However, if the paper provides the execution times and the sequence sizes, we can easily calculate the GCUPS, as explained earlier. If it is not possible to calculate the GCUPS, we mark the paper as *NoP* (Not Provided).

The CUPS metrics should not be used to make a direct comparison among approaches since CUPS are obtained in different experimental setups, with different sequences, of different sizes. However, this metric provides a very good idea of the potential of each approach, being widely used in the literature to compare solutions for the Biological Sequence Comparison problem in a broad sense.

6. BIOLOGICAL SEQUENCE COMPARISON SOLUTIONS IN CPU

The use of general-purpose CPUs (Section 5.1.1) to accelerate sequence comparison computations has been popular since the 1990s. The first proposals were executed in clusters of computers in a fine-grained manner (Section 4.10.2), and often used the BW method to respect data dependencies. In the 2000s, many implementations of the SW algorithm that use SIMD instructions provided by general-purpose CPUs were also proposed. In this section, we first discuss solutions that do not use vector instructions (Sections 6.1 to 6.4), then present solutions with SIMD optimizations (Sections 6.5 to 6.9).

6.1. Galper and Brutlag (1990)

Galper and Brutlag [1990] proposed one of the first parallel strategies for retrieving alignments computed with the edit-distance algorithm (Section 4.5). It runs in quadratic memory and outputs x optimal alignments. The fine-grained wavefront method (Section 4.10.2) is implemented in two ways: using the block-based approach (BW) or the diagonal approach (DW). Also, an innovative approach called windowed asynchronous wavefront is provided, which relaxes the lock-step condition to compute antidiagonals. Results were obtained with an Encore Multimax shared memory machine with up to 12 processors. The maximum query size used in the tests was 118 nucleotides, and up to 10 optimal alignments were retrieved. The comparison of two DNA sequences of sizes 22 and 30,000 (22 x 30,000) achieved 0.0009 GCUPS.

6.2. Rajko and Aluru (2004)

In Rajko and Aluru [2004], an algorithm is proposed for global sequence alignment using the affine-gap function. This algorithm combines Hirshberg's algorithm (Section 4.7) with PP computations (Section 4.10.2) aiming to obtain a partition between subsequences of S_0 and S_1. With this partition, the problem can be divided into independent subproblems, which can be solved in parallel. In order to accelerate the alignment retrieval phase, the authors employ a variant of the GU variation (Section 4.7.1). The optimal global alignment is the concatenation of the optimal alignments found in each subproblem. This algorithm was tested in a cluster with 60 processors. A GCUPS rate of 0.24 was obtained when comparing two sequences of size 1,100,000.

6.3. Chen and Schmidt (2005)

In Chen and Schmidt [2005], a parallel adapted version of SW (Section 4.3) is proposed; it not only finds the optimal alignment, but also some near-best local alignments using MM (Section 4.7). A huge DP matrix is partitioned among different clusters of computers (intercluster) in such a way that a cluster is responsible for computing a set of columns (BW). Each cluster then assigns a subset of its columns to the processors that belong to it. In the computation, an additional field is calculated for each cell (i, j), containing the row where the optimal alignment occurs, in an adapted version of GU (Section 4.7.1). Thus, the alignment can be retrieved in linear time, at the expense of a considerable amount of additional memory. A dynamic load-balancing technique is proposed, in which the nodes exchange data with their direct neighbors to adjust the number of columns assigned. Experiments made in two connected clusters with a total of 18 processors (Intel PIII 733 MHz and Intel Itanium-1 733 MHz) presented a GCUPS of 0.036 when comparing two sequences of sizes 144,260 and 132,290, respectively.

6.4. Batista et al. (2008)

Z-align [Batista et al. 2008] is an MPI-based parallel strategy derived from the BW method to execute Gotoh in restricted memory space. Two additional DP matrices (called divergence matrices) are proposed, with the goal of restricting the space in the traceback procedure, in an adapted version of the Fickett algorithm (Section 4.6). This approach was able to compare sequences of up to 3 Million Base Pairs. The best

GCUPS (0.21) were obtained in a cluster of 8 dual-core nodes, when comparing real DNA sequences of sizes 3,147,090 and 3,282,708.

In Boukerche et al. [2012], this work was extended to compute the DP matrix in a cyclic way, by dividing it into xp subsets of columns, where p is the number of processing units and x is a constant. A GCUPS rate of 1.39 was obtained when comparing real DNA sequences of size 23 MBP x 24 MBP in a cluster of 64 cores.

6.5. Wosniak (1997)

One of the first algorithms to use SIMD instructions from general-purpose CPUs in Optimal Pairwise Comparisons was proposed by Wozniak [1997]. In this proposal, SIMD registers manipulated by the Visual Instruction Set (VIS) of the Sun Ultra Sparc processor are filled with Gotoh's DP cells belonging to the same antidiagonal (8 elements), which can be calculated in parallel using the DW method (Section 4.10.2). The proposed algorithm was tested with up to 12 processors, comparing sequences with up to 5,217 amino acids against the Swiss-Prot database, attaining up to 0.19 GCUPS.

6.6. Rognes and Seeberg (2000): SWMMX

Rognes and Seeberg [2000] proposed a Gotoh implementation that compares protein sequences with affine gap using the Intel vector instructions MMX/SSE. Cells are calculated in parallel, column by column, applying the SWAT optimization. In addition to using the SWAT optimization (Section 4.9.1), Rognes and Seeberg [2000] proposed the *query profile* optimization, which computes a small specific matrix P based on the query sequence and a given substitution matrix. This matrix is further used in the DP matrix computation. Each MMX/SSE instruction acts on 8 cells (i to $i + 8$) in a single column of the DP matrix, calculating them in a parallel SIMD way. Score values are restricted to 8b (0 to 255). Experiments were conducted that compared 11 sequences, with sizes ranging from 189 to 567 amino acids, using the genomic database Swiss-Prot. GCUPS of 0.15 were attained with one Pentium III 500MHz.

6.7. Farrar (2007)

The algorithm proposed in Farrar [2007] uses the concept of row-based parallelism and the query profile discussed in Section 6.6, but with a different pattern of data access, called striped. In the striped pattern, the cells calculated in an SIMD way belong to the same column and are, for instance, $\{1,\ 1 + t,\ 1 + 2t,\ 1 + 3t\}$, with a stripe factor of 4. This striped pattern reduces data dependencies, moving the F test to the outer loop, accelerating the computation. The algorithm was implemented in C, using SSE2 instructions, and the tests were conducted in an Intel Xeon Core 2 Duo 2.0GHz. Query sequences with sizes up to 567 amino acids were compared to the entire Swiss-Prot genomic database, and a peak performance of 3.0 GCUPS was attained.

6.8. Rognes (2011) - SWIPE

Rognes [2011] proposed a multithreaded, coarse-grained version using SSE code to compare protein sequences with the Gotoh algorithm. Basically, each SIMD operation computes elements from different comparisons. In this design, a task is defined as a chunk of x database sequences, that is, each task will compare a query sequence to a subset of database sequences. In the tests, 32 query sequences, with sizes ranging from 24 to 5,478 amino acids, were compared to the Uniprot Knowledge database (Swiss-Prot release 53.0 and TrEmbl release 36.0, with 4.6 Million protein sequences). The tests were conducted in two Intel Xeon six-core processors. A maximum of 106 GCUPS was achieved when the query sequence was longer than 100 amino acids.

Table I. Comparative View of CPU Papers

		Problem					Algorithm				HPC solution	
Section	Year	Type Seq	Max Size	Type Align	Gap	Out	Algo	Mem Comp.	Matrix Comp	Paral	#PU	Perf GCUPS
Without SIMD Extensions												
6.1	1990	DNA RNA	10^2	global	linear	score align	edit	quad	full	fine (DW BW)	12	0.0009
6.2	2004	DNA	10^6	global	affine	score align	NW Hirschberg	linear	full	fine (PP)	60	0.24
6.3	2005	DNA	10^5	local	affine	score align	Gotoh MM	linear	full	fine (BW)	18	0.036
6.4	2008	DNA	10^6	local	affine	score align	Gotoh Fick	$O(kn)$	kn	fine (BW)	16	0.21
	2012	DNA	10^7	local	affine	score align	Gotoh Fick	$O(kn)$	kn	fine (BW)	64	1.39
Using SIMD Extensions												
6.5	1997	Prot	10^3	local	affine	score	Gotoh	N/A	full	fine (DW)	12	0.19
6.6	2000	Prot	10^2	local	affine	score	Gotoh	N/A	SWAT	fine (col)	1	0.15
6.7	2007	Prot	10^2	local	affine	score	Gotoh	N/A	SWAT	fine (col)	2	3.00
6.8	2011	Prot	10^3	local	affine	score	Gotoh	N/A	SWAT	coarse	12	106.00
6.9	2014	DNA	10^2	local global	affine	align	Gotoh others	NoP	MVM SWAT	fine	128	900.00

6.9. Maleki et al. (2014)

Maleki et al. [2014] use linear algebra to transform DP problems, including local and global comparisons with affine gap, into independent sets of matrix-vector multiplications (MVM) and predecessor products, using the concept of rank convergence to obtain optimal alignments. The algorithm executes iteratively, stage by stage, where each stage is executed in parallel. The authors show that few steps are needed for convergence in the local alignment case. However, they also show that global alignment requires many more steps to converge, and that LCS does not converge, in some cases. For local alignments, Farrar's algorithm (Section 6.7) was used inside each stage. Results were collected in 8 nodes (2 x 8-core Intel Xeon 2.7GHz) of the Stampede cluster, and the communication among the cluster was done by MPI. Four Expressed Sequence Tags (ESTs) were used as queries and compared to human chromosomes 1 to 4 (up to 249 Millions of Base Pairs). Even though the authors did not provide the actual sizes of the ESTs used, ESTs are no longer than 800 nucleotides [Hoeppner et al. 2013]. This approach obtained about 900 GCUPS for the local alignment, with 128 cores.

6.10. Comparative View of the CPU Approaches

As shown in Table I and throughout this section, research on HPC solutions for CPU-based platforms took two different paths: exploring clusters of computers (Sections 6.1 to 6.4) or SIMD capability (Sections 6.5 to 6.8). Recently, a very different approach was proposed that combines SIMD capability with linear algebra (Section 6.9). The solutions for clusters compare DNA sequences, whereas the SIMD-based solutions compare protein sequences. They apply different optimizations targeted to the type of the sequence. The last proposal combines SIMD-based optimizations with linear algebra operations. Most of the approaches execute local comparisons with the affine-gap model. The cluster solutions compute both the score and the alignment, whereas the SIMD-based solutions (Sections 6.6 to 6.8) compute only the score.

When the alignment is provided, it is computed in quadratic memory (Section 6.1), linear memory (Sections 6.2 and 6.3), and $O(kn)$ memory (Section 6.4). The solutions in Sections 6.5 to 6.8 provide only the optimal score as output; thus, the memory complexity of the traceback stage (Mem. Comp.) is N/A. Even though the solution discussed in Section 6.9 retrieves the alignment, it did not provide the memory complexity (*NoP*). Most of the SIMD approaches use the SWAT optimization and query profiles to accelerate the computation. In Section 6.9, the problem is transformed into MVMs to break the dependencies.

With the exception of the proposal discussed in Section 6.8, all proposals employ fine-grained parallelism, either using wavefront-based methods, parallel prefix, column parallelism combined with SWAT, or matrix-vector multiplications. The performance of CPU-based approaches experienced a fantastic increase over the years, ranging from 90 MCUPS in 1990 to 900 GCUPS in 2014.

7. BIOLOGICAL SEQUENCE COMPARISON SOLUTIONS IN HARDWARE (FPGA/ASIC)

In this section, we discuss hardware-based approaches to execute Pairwise Biological Sequence Comparisons. Since the great majority of these approaches use FPGAs, we provide an overview of these solutions in Section 7.1 and discuss each solution separately. In Sections 7.2 to 7.5, we provide details about FPGA-based approaches. In Section 7.6, we discuss an Application Specific Integrated Circuit (ASIC)-based design.

7.1. Overview of the FPGA/ASIC Solutions

In the literature, several proposals of FPGA-based designs (Section 5.1.2) accelerate sequence comparison applications by calculating the antidiagonals of the DP matrix in hardware with the DW method (Section 4.10.2). In the DW approach for FPGAs, an array processor with N Processing Elements (PEs) is designed, in which each PE computes one matrix cell per turn. Therefore, one processor array with N PEs can generate N scores at a time. Usually, systolic arrays [Kung 1982] are used, in which the PEs operate in a lock-step basis. Lipton and Lopresti [1985] proposed one of the first implementations of a Biological Sequence Comparison algorithm in a systolic array. Since then, variations of this design have been used in several different proposals.

The most common FPGA design for sequence comparison is the unidirectional systolic array. In this design, the characters that belong to the query sequence are stored in the PEs. The database sequence can be of any size, since it is fed into the FPGA and "passes through" it. At time $T0$ (Figure 9), the elements of the query sequence (A, C, G, A, and T) are already stored in the PEs. At time $T1$, the first PE compares A to C, using one of the recurrence relations shown in Equations (1) to (6). At $T2$, two comparisons are made simultaneously (C to C and A to T) by the first and second PEs, respectively. At $T3$, three cells are calculated in parallel by the first three PEs, and so on. Frequently, the size of the query sequence is greater than the number of PEs contained in the FPGA. To handle this, a partitioning technique is applied to split the query sequences. Two approaches can be used in this case. The first approach stores the scores of the last calculated column in the FPGA memory, and feeds back the systolic array with these values. In the second approach, several characters belonging to the query sequence are stored in the same PE, organized in a FIFO buffer.

A variation of the unidirectional systolic array is the bidirectional systolic array. In this design, the database and query sequences are fed at the same time into the PE array, in opposite sides and directions. Computation starts when the first characters of both sequences meet in the middle PE, and continues until every character has been compared.

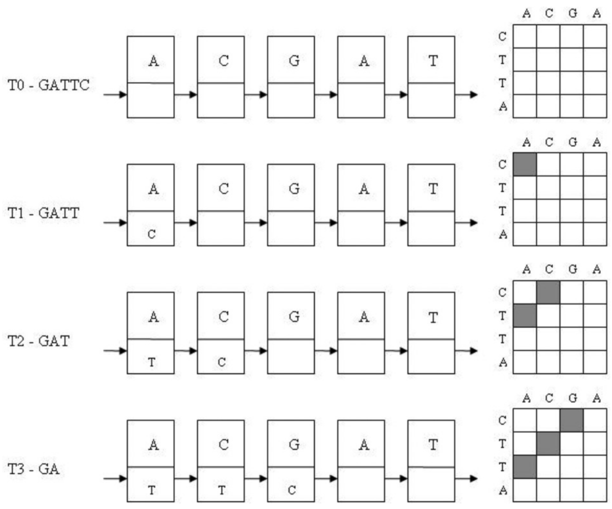

Fig. 9. A 5-PE systolic array to calculate the DP matrix [Boukerche et al. 2010].

7.2. FPGA: Hoang and Lopresti (1992)

A bidirectional systolic array was proposed by Hoang and Lopresti [1992] to calculate both the score and the alignment with the edit-distance algorithm (Section 4.5). The sequences are processed twice, and the alignment is output as a set of 2-bit representations that indicate (a) the alignment of both characters, (b) a gap in the first sequence, or (c) a gap in the second sequence.

Ten thousand alignments of DNA sequences of size 100 were performed in the SPLASH board, which is composed of 32 FPGAs Xilinx XC3090. The total of 248 PEs were obtained for this platform, with 0.27 GCUPS.

7.3. FPGA: Oliver et al. (2005)

Oliver et al. [2005] proposed the use of a unidirectional systolic array of PEs (Figure 9) to execute local comparisons with linear or affine gap (Section 3.4). The proposed architecture outputs the optimal score. Switching between gap functions is achieved by FPGA reconfiguration. Moreover, the PEs are placed in the FPGA according to a zigzag pattern, which provides good occupation of space. Several characters that compose the query sequence can be stored in the same PE. A FIFO is designed to accommodate the query-sequence characters, as well as to use the substitution matrix column and the database sequence itself.

The architecture was programmed in Verilog and synthesized for the Virtex II XC2V6000 FPGA board. Query sequences of up to 2,016 amino acids in size were compared with the Swiss-Prot database on a 252-PE array. This design achieved 10.6 GCUPS and 5.8 GCUPS for the linear- and affine-gap cases, respectively.

7.4. FPGA: Zhang et al. (2007)

Zhang et al. [2007] proposed the use of a unidirectional systolic array (Figure 9) to execute local comparisons for protein and DNA sequences, with both linear- and affine-gap functions. As a result, the optimal score is output. The authors used a partition technique that compares parts of the query sequence with the entire database sequence in each phase. The elements of the last column computed in each phase are stored in the FPGA internal memory, as explained in Section 7.1. As optimization, the authors propose a pipeline with uneven stage latencies. Also, intra-PE optimizations were proposed to reduce the number of *max* operations. The proposed architecture was synthesized for the XD1000 platform. In this platform, the main processor is an AMD64 Opteron, connected through HyperTransport to an Altera Stratix II FPGA. The main processor and the FPGA are located in the same board, and the FPGA acts as a co-processor. A PE array of 384 elements was implemented, all of which work at 66.7MHz. The maximum GCUPS attained was 25.6, when comparing two sequences of 65,536 amino acids.

7.5. FPGA: Caffarena et al. (2007)

Caffarena et al. [2007] proposed a unidirectional systolic array (Figure 9) that uses 1b arithmetic and differential coding to execute the edit-distance algorithm (Section 4.5), comparing DNA sequences with a linear-gap function. In the experimental results, sequences with up to 11,000 nucleotides were compared in an FPGA Xylinx XC2V6000-5 operating at 100MHz. This design was able to attain 156 GCUPS.

7.6. ASIC: Sarkar et al. (2010a)

Sarkar et al. [2010a] designed an ASIC composed of tiny PEs integrated through a Network-on-Chip (NoC) to retrieve global, local, and semi-global alignments of DNA or protein sequences, calculated with the affine-gap function. In order to rapidly retrieve the alignment, the PEs store all cells of the last column calculated (GU variation, Section 4.7.1). The authors designed a unidirectional systolic array (Figure 9) that can calculate alignments with either DW or PP (Section 4.10.2). In order to do that, the NoC is designed as a 2D mesh, which is used directly by the PP implementation. The DW implementation embeds a unidirectional ring in the 2D mesh. Each PE was synthesized using the 90nm Standard Cell Library. The switches that connect the PEs were also carefully designed, generating a custom hardware.

The authors aligned query sequences with up to 1,024 nucleotides/amino acids using PP and DW. They obtained the best execution times when comparing 1KBP x 1KBP sequences with 16 PEs, using PP. In this case, 243.8 GCUPS were attained.

7.7. FPGA: Wienbrandt (2013)

In Wienbrandt [2013], the Rivyera S3-5000 platform is used to accelerate the execution of SW with linear gap. Rivyera S3-5000 is composed of a multi-FPGA module and a host. The multi-FPGA module is equipped with 16 cards, each with 8 FPGAs XLINX Spartan3-5000 and 32MB of RAM memory, with a total of 128 FPGAs. The FPGAs are connected through a high-performance bus system, organized as a systolic chain. The host main board has an Intel i7 processor with 12GB of RAM memory. The host and the multi-FPGA modules are connected through PCIe.

The author proposed a unidirectional systolic array design (Figure 9) to compare DNA or protein sequences. With query sequences of exactly 100 characters, the DP matrix has 5b cells (nucleotides) or 6b cells (amino acids). For DNA comparisons the NUC44 substitution matrix is used, whereas the BLOSUM62 substitution matrix is used for protein comparisons. As output, the score is provided. In each FPGA, four different DNA comparisons take place simultaneously. Therefore, 512 simultaneous DNA comparisons are made in the entire FPGA system.

The experimental results were obtained for the comparison of 1 Million 100 BP short reads against the human genome (3.2 GBP). In this case, the whole application took 29 hours to complete, achieving a performance of 3.04 TCUPS.

7.8. Comparative Analysis of the FPGA/ASIC Approaches

In Table II, we provide a comparative view of the approaches discussed in Sections 7.2 to 7.7. In this table, we see that many approaches can be customized to compare either

Table II. Comparative View of FPGA/ASIC Papers

		Problem					Algorithm				HPC solution	
Section	Year	Type Seq	Max Size	Type Align	Gap	Out	Algo	Mem Comp.	Matrix Comp	Paral	#PU	Perf GCUPS
7.2	1992	DNA	10^3	global	linear	score align	edit	quad.	full	fine (DW)	32	0.27
7.3	2005	Prot	10^4	local	linear affine	score	SW Gotoh	N/A	full	fine (DW)	1	10.26
7.4	2007	DNA Prot	10^5	local	linear affine	score	SW Gotoh	N/A	full	fine (DW)	1	25.60
7.5	2007	DNA	10^4	global	linear	score	edit	N/A	full	fine (DW)	1	156.00
7.6	2010	DNA Prot	10^3	global local s-glo	affine	score align	Gotoh	O(m+n) /p	full	fine PP (DW)	1	243.80
7.7	2013	DNA Prot	10^3	local	linear	score	SW	N/A	full	coarse fine (DW)	128	3040.00

DNA or protein sequences. The maximum size of the query sequence was initially set to 10^3 [Hoang and Lopresti 1992], then increased to 10^4 [Oliver et al. 2005] and, finally, 10^5 [Zhang et al. 2007]. In the last proposals (Sarkar et al. [2010a] and Wienbrandt [2013]), the maximum size was reduced to 10^3. Most of the approaches execute local comparisons, and some provide global and/or semi-global matching.

Most of the approaches compute linear-gap penalties, and many also compute affine gaps. With the exception of Hoang and Lopresti [1992] and Sarkar et al. [2010a], all approaches provide only the score as output. In all approaches, all cells of the DP matrices are computed (full computation) with a fine-grained diagonal wavefront parallelization strategy. Sarkar et al. [2010a] also calculates the DP matrix in a fine-grained way, using PP.

The approaches use one FPGA (Oliver et al. [2005], Zhang et al. [2007], and Benkrid et al. [2009]), one ASIC [Sarkar et al. 2010a], 32 FPGAs [Hoang and Lopresti 1992] or 128 FPGAs [Wienbrandt 2013]. Over the years, we have observed an astonishing increase in the GCUPS rate, ranging from 0.27 GCUPS in Hoang and Lopresti [1992] to 3.04 TCUPS in Wienbrandt [2013]. This increase has occurred primarily due to the evolution of FPGA technology in the past 21 years and to the highly sophisticated current solutions, some supporting a great number of FPGAs.

8. BIOLOGICAL SEQUENCE COMPARISON SOLUTIONS IN CELLBE

The approaches that use the CellBE (Section 5.1.3) to implement Biological Sequence Comparison solutions take advantage of the SIMD capability of the SPEs, executing vector instructions to accelerate the computation of the DP matrix. All CellBE proposals must take special care to overlap communication and computation, since each SPE has a very limited amount of local memory (256KB). In addition, the SPEs do not provide support for saturated arithmetics; most approaches must implement it by themselves.

8.1. Szalkowski et al. (2008): SWPS3

In SWPS3 [Szalkowski et al. 2008], an implementation of Gotoh's algorithm is proposed for the CellBE and the Intel x86/SSE; both are used to compare a query sequence to a genomic database, giving the score as output. Query sequences can be segmented, and the maximum size allowed for both sequences is 10,000. Initially, the scores are stored in 8b integers that can be extended to 16b. Each SPE compares the same query sequence with different database sequences, thus using a coarse-grained approach, with the Farrar optimization (Section 6.7) for vector instructions inside each SPE (fine-grained). The results obtained on a PS3 with 6 SPEs show that, to compare query sequences of up to 4,000 amino acids to the Swiss-Prot database, a performance of 8 GCUPS is achieved.

8.2. Aji et al. (2008): Cell-SWat

Cell-SWat [Aji et al. 2008] is a tiled version of Gotoh designed for the CellBE. It can use all SPEs in a single comparison (fine-grained) or can use each SPE to process different comparisons (coarse-grained). In the fine-grained computation, the first phase of Gotoh is processed by the SPEs in such a way that the matrix is divided into block rows (Section 4.10.2). The block rows are themselves divided into tiles, which are processed in a vectorized way according to the DW method (Section 4.10.2). As an optimization, the matrix is stored as a one-dimensional array. At the end of the execution of this first phase, the score is obtained. The second phase of Gotoh (obtain the alignment) is executed in quadratic memory space, exclusively by the PPE.

Cell-SWat is extended in Aji and Feng [2008] for execution in a cluster of 14 PS3s. In this case, a genomic database is partitioned in x subsets, where x is the number of PS3 consoles. Then, each PS3 console compares the same sequence with a different subset. After the work distribution, there is no communication among the PS3 consoles. DNA query sequences of up to 3,584 residues are compared at 0.42 GCUPS.

8.3. Sarje and Aluru (2009)

A fine-grained parallel approach for retrieving optimal global and local alignments is proposed in Sarje and Aluru [2009]. The MM algorithm (Section 4.7) is implemented, combining PP with diagonal wavefront parallelizations (Section 4.10.2). This algorithm uses double-buffering as well as communication and computation overlap. The authors also implemented two different alignment algorithms (spliced and syntenic), also using wavefront computations, PP and the MM algorithm. This approach was tested in an IBM S20 Cell Blade with 16 SPEs, comparing query sequences of up to 2,048 nucleotides. It achieved 0.8 GCUPS when obtaining syntenic alignments for sequences of size 1,792 x 1,580.

8.4. Sanchez et al. (2011)

Sanchez et al. [2011] propose an approach for comparing long DNA sequences in the CellBE that uses double buffering to overlap computation and communication. Each SPE calculates a subset of rows of the DP matrix, which are assigned in a circular way. The subset of rows is divided into blocks (BW method). The SPEs directly access their left neighbor memory to retrieve data and write it into the right neighbor's memory, through DMA operations. The tests were conducted with an IBM Blade QS20, comparing sequences of 3.4 MBP and 1.8 MBP, respectively. A speedup of 15x was achieved, when compared to the execution with 1 SPE.

8.5. Comparative View of the CellBE Approaches

In Table III, we provide a comparative view of the CellBE approaches. Most papers compare either DNA or protein sequences; Sarje and Aluru [2009] also compare RNA sequences. In most of the approaches, the maximum size of the query sequence is below 10,000. The longest sequences are compared by Sanchez et al. [2011], with a query sequence size of 1.8 MBP. All the approaches compute the local alignment, and [Sarje and Aluru 2009] also execute syntenic and spliced alignment algorithms.

Most approaches use affine gaps to compute the alignments with the Gotoh algorithm. As output, only Aji et al. [2008] and Sarje and Aluru [2009] provide the optimal

Table III. Comparative View of CellBE Papers

		Problem					Algorithm				HPC solution	
Section	Year	Type Seq	Max Size	Type Align	Gap	Out	Algo	Mem Comp.	Matrix Comp	Paral	#PU	Perf GCUPS
8.1	2008	Protein	10^3	local	affine	score	Gotoh	N/A	SWAT	coarse fine (col)	1	8.00
8.2	2008	DNA	10^3	local	affine	score align	Gotoh	quad	full	coarse fine (BW DW)	14	0.42
8.3	2009	RNA DNA	10^3	local spli synt	affine	score align	Gotoh MM others	linear	full	fine (DW PP)	1	0.80
8.4	2011	DNA	10^6	local	linear	score	SW	N/A	full	fine (BW)	1	NoP

alignment: the first retrieves alignments in quadratic memory space, whereas the second obtains alignments in linear space. The SWAT optimization is applied in Szalkowski et al. [2008] and the other approaches compute the entire DP matrices.

With the exception of Aji et al. [2008], the approaches use only one CellBE (either PS3 or IBM Cell Blade) to compare the sequences. Most approaches apply a fine-grained strategy, in which a single comparison is executed by multiple SPEs, which execute SIMD operations. In Szalkowski et al. [2008], the comparisons are distributed among the SPEs; each SPE executes different comparisons using the SIMD operations (coarse and fine-grained approach). The best GCUPS rate (8.00) was obtained by Szalkowski et al. [2008], comparing protein sequences smaller than 10,000 with the SWAT optimization.

9. BIOLOGICAL SEQUENCE COMPARISON SOLUTIONS IN GPU

Recently, the use of GPUs (Section 5.1.4) to implement biological sequence comparison algorithms has become very popular. This mainly occurs for two reasons. First, GPUs are able to attain very good GCUPS at a low cost. Second, programming GPUs is much simpler than programming hardware-based solutions such as FPGAs, for instance. In this section, we discuss seven popular GPU implementations.

9.1. Liu et al. (2006): DASW

Liu et al. [2006] proposed a local sequence comparison tool that computes a double affine-gap function in GPU, programmed with the OpenGL API. In the double affine-gap model, there is a different penalty for each extending gap. In this design, only small scores can be computed, since 16b variables are used. Two modes of operation are available: one that outputs only the score, and another that outputs the score and the alignment. The last mode of operation is executed in quadratic space. The DW method (Section 4.10.2) is implemented, and the diagonals currently being used are kept in circular buffers.

A total of 0.24 GCUPS and 0.18 GCUPS were achieved for the first and second modes of operation, respectively, when comparing a query sequence of 16,384 amino acids to a protein database with the NVIDIA GeForce 7800 GTX.

9.2. Manavski and Valle (2008)

Manavski and Valle [2008] proposed an implementation of Gotoh that runs in multiple GPUs. This is one of the first proposals to use CUDA. To accelerate access to the substitution matrix, the authors use the query profile technique (Section 6.5). Scores are restricted to 16b values. Each GPU thread compares the same query sequence to one database sequence (coarse-grained parallelism). Query sequences of up to 2,050 amino acids are supported, but tests show the comparison of query sequences of at most 567 amino acids. GCUPS rates of 1.89 and 3.61 were achieved for one and two GPUs NVIDIA GeForce 8800 GTX, respectively.

9.3. Liu et al.: CUDASW++

The first version of CUDASW++ was proposed in 2009, the second in 2010 and the most recent version (CUDASW++ 3.0) was proposed in 2013. In this section, we will discuss CUDASW++ 1.0 and 2.0, which target GPUs exclusively. CUDASW++ 3.0 uses both GPUs and CPUs, and will be addressed in Section 11.

CUDASW++ 1.0 [Liu et al. 2009] executes Gotoh comparisons in single GPU and multiple GPUs. The authors propose the use of two levels of parallelism: intertask and intratask parallelization. Intertask parallelization is coarse-grained, and assigns each query $\times$ database sequence comparison to a different thread. On the other hand, intertask parallelization uses multiple threads in a single query $\times$ database comparison (fine-grained mode). Small query sequences are compared with intertask parallelization, whereas long query sequences use the intratask mode. In order to achieve load balancing, query sequences are organized according to length. Also, great care is taken with the manipulation of the GPU memory hierarchy. Results of 16.087 GCUPS were obtained for the NVIDIA GeForce GTX 295, for a maximum query size of 5,478 amino acids.

CUDASW++ 2.0 [Liu et al. 2010] is a combination of CUDASW++ 1.0 and SIMD optimizations proposed for general-purpose CPUs. It uses query profile, the striped pattern with the SWAT optimization (Section 6.7), and a clever disposition of data over the GPU memory hierarchy. Each thread computes a partition, where a partition contains parts of the query sequence to be compared to a database sequence. Amino acid sequences of up to 5,478 characters were compared to the Swiss-Prot genomic database with the NVIDIA GeForce GTX 295, and a GCUPS rate of 29.7 was obtained.

9.4. Sandes et al.: CUDAlign

The main goal of CUDAlign is to compare Megabase DNA sequences with the Gotoh algorithm in a GPU. Its first version was proposed in 2010. In 2011 and 2013, CUDAlign 2.0 and 2.1 were proposed. Its latest version (CUDAlign 3.0) was proposed in 2014.

CUDAlign 1.0 [Sandes and Melo 2010] calculates the DP matrix with the wavefront method, assigning blocks to threads in a parallelogram shape, and provides the optimal score as the output. In order to increase performance, it carefully places data in the GPU memory hierarchy. Results were collected in the NVIDIA GTX 280 GPU, comparing sequences with sizes up to 32 MBP, achieving a maximum GCUPS of 20.37.

CUDAlign 2.0 [Sandes and Melo 2011] is a combination of the Gotoh and the MM algorithm (Section 4.7) that retrieves optimal local alignments in linear space. It is executed in 5 stages. Stage 1 obtains the optimal score, saving some rows to disk, as in the GU variation 4.7.1. Stages 2 to 5 execute a modified version of MM, retrieving the coordinates of the points that belong to the optimal alignment in a divide-and-conquer manner. The results collected in the NVIDIA GTX 285, comparing sequences with sizes up to 33 MBP, show a maximum GCUPS of 23.63.

CUDAlign 2.1 [Sandes and Melo 2013] calculates the alignment as in CUDAlign 2.0, with BP optimization (Section 4.9.2) being capable of pruning up to 53.7% of the DP cells. The results collected in the NVIDIA GTX 560 Ti comparing sequences with sizes up to 33 MBP show a maximum GCUPS of 58.21.

CUDAlign 3.0 [Sandes et al. 2014] executes the Gotoh algorithm in multiple GPUs with fine-grained parallelism and outputs the optimal score. It uses circular buffers to overlap computation and communication, connecting the GPU nodes with TCP sockets. The results collected in the Minotauro GPU cluster, with up to 64 GPUs NVIDIA Tesla M2090, present a maximum GCUPS of 1,726 when comparing DNA sequences of 228 MBP x 249 MBP. This huge comparison took 9 hours and 9 minutes to complete.

9.5. Ino et al. (2009, 2012)

In Ino et al. [2009], multiple nondedicated GPUs are used to execute the Gotoh algorithm for protein sequences. A master/slave design is proposed, in which the CPU is the master and the GPUs are the slaves. In this design, tasks are one comparison of a given query sequence with the entire genomic database, executed in a coarse-grained manner. When the master detects that the system is idle, that is, the screensaver is activated, one task is allocated to each GPU. When a GPU finishes executing its task, it requests another task, until there are no remaining tasks. In the tests, 64 query sequences of size 367 were compared to the Swiss-Prot database. The platform was composed of 8 heterogeneous GPUs (8800 and 7900 series). In this platform, a maximum of 3.09 GCUPS was obtained.

The work of Ino et al. [2009] was adapted by Ino et al. [2012] to exploit shorter periods of idle time; in this design, screensaver detection is not used. Local comparison tasks are distributed to the GPUs, with a scheduling algorithm that prioritizes those that have been idle for longer periods of time. Also, if the GPU that is computing a task becomes busy, tasks can be canceled and allocated to another idle GPU. The tests compared a set of query sequences ranging from 63 to 511 amino acids to a genomic database. The results obtained in a platform composed of 8 NVIDIA GPUs (5 GTX 285, 1 GTX 295, 1 FX 5800, and 1 8800 GTX), each connected to a host machine, show that 64 GCUPS can be achieved.

9.6. Kopar and Sikic (2013) - SW#

Korpar and Sikic [2013] implemented the MM algorithm (Section 4.7) with the parallelization strategy and BP optimization proposed in Section 9.4 for retrieving the local alignment between megabase DNA sequences. In the NVIDIA GTX 690 GPU (2 GPUs), the comparison of sequences with 33 MBP x 47 MBP was done in 23,614s, with a GCUPS rate of 65.20.

9.7. Liu and Schmidt (2014a): GSWABE

In Liu and Schmidt [2014a], a GPU strategy that computes global, semi-global, and local alignments with the Gotoh algorithm is proposed for short DNA reads. In this strategy, the DP matrix is divided into small tiles measuring 4x4, and each thread executes a different comparison. The alignments are retrieved in quadratic space, since the approach targets short DNA sequences. The results obtained in an NVIDIA Tesla K40, aligning sequences with size up to 500 BPs, present a GCUPS rate of 58.3.

9.8. Comparative View of the GPU Approaches

Table IV provides a comparative view of the GPU approaches. The papers compare either DNA or protein sequences. The maximum query sizes of the protein sequences range from 10^2 to 10^4. Two of the approaches that compare DNAs target Megabase sequences (10^7, 10^8) and one approach targets short DNAs (10^2). All approaches execute local comparisons, and Liu and Schmidt [2014a] (Section 9.7) also executes global and semi-global comparisons. All GPU approaches discussed in this section compute affine-gap penalties. Four approaches retrieve the alignment.

Table IV. Comparative View of GPU Papers

		Problem					Algorithm				HPC solution	
Section	Year	Type Seq	Max Size	Type Align	Gap	Out	Algo	Mem Comp.	Matrix Comp	Paral	#PU	Perf GCUPS
9.1	2006	Prot	10^4	local	affine	score align	Gotoh	quad	full	fine (DW)	1	0.24
9.2	2008	Prot	10^2	local	affine	score	Gotoh	N/A	full	coarse	2	3.61
9.3	2009	Prot	10^4	local	affine	score	Gotoh	N/A	full	coarse fine (DW)	1	16.09
	2010	Prot	10^4	local	affine	score	Gotoh	N/A	SWAT	coarse fine (col)	1	29.70
9.4	2010	DNA	10^7	local	affine	score	Gotoh	N/A	full	fine (BW DW)	1	20.37
	2011	DNA	10^7	local	affine	score align	Gotoh MM	linear	full	fine (BW DW)	1	23.63
	2013	DNA	10^7	local	affine	score align	Gotoh MM	linear	BP	fine (BW DW)	1	58.21
	2014	DNA	10^8	local	affine	score	Gotoh	N/A	full	fine (BW DW)	64	1726.47
9.5	2009	Prot	10^3	local	affine	score	Gotoh	N/A	full	coarse	8	3.09
	2012	Prot	10^3	local	affine	score	Gotoh	N/A	full	coarse	8	64.00
9.6	2013	DNA	10^7	local	affine	score align	MM	linear	BP	fine (BW DW)	2	65.20
9.7	2014	DNA	10^2	local global s-glob	affine	score align	Gotoh	quad	full	coarse	1	58.30

When small sequences are targeted (Liu et al. [2006] and Liu et al. [2014]), the alignment is retrieved in quadratic space. For long sequences, the proposals (Sandes and Melo [2013] and Korpar and Sikic [2013]) retrieve the alignment in linear space with block-pruning optimization. Most of the approaches use fine-grained parallelization strategies with tiling techniques, assigning part of a diagonal (DW), a block of diagonals (BW), or a part of a column (col) to each thread to enhance parallelism. The papers that employ coarse-grained parallelization sort the database sequences by size in order to achieve load balancing. As can be seen in the last column, the proposals for GPUs have also experienced a phenomenal increase in performance, ranging from 0.24 GCUPS (2006) to 1.7 TCUPS (2014).

10. BIOLOGICAL SEQUENCE COMPARISON SOLUTIONS IN INTEL PHI

Although the first Intel Phi was released in 2012 (Section 5.1.5), proposals have already been made to execute Optimal Pairwise Comparisons within it. We discuss three of these proposals in the following sections.

Table V. Comparative View of Intel Phi Papers

		Problem					Algorithm				HPC solution	
Section	Year	Type Seq	Max Size	Type Align	Gap	Out	Algo	Mem Comp.	Matrix Comp	Paral	#PU	Perf GCUPS
10.1	2014	protein	10^4	local	affine	score	Gotoh	N/A	full	coarse fine (BW)	4	228.4
10.2	2014	DNA	10^7	local	affine	score	Gotoh	N/A	full	fine (BW)	4	111.4
10.3	2014	DNA	10^3	local	affine	score	Gotoh	N/A	full SWAT	coarse fine (BW)	1	70.0

10.1. Liu et al. (2014): SWAPHI

A strategy for executing the Gotoh algorithm for query x database searches with one or more Intel Phis is proposed in Liu and Schmidt [2014b]. To the best of our knowledge, this is the first Intel Phi local comparison implementation. The database sequences are sorted according to length and are transferred from the host to the Intel Phi concurrently to the computation. Each thread receives a different query x database pair, calculating different DP matrices with SIMD instructions. To accelerate computations, the authors (a) use query profile; (b) create a sequence profile, composed of 16 consecutive database sequences; and (c) employ a careful design of memory allocations/deallocations. In the tests, a host with 4 Intel Phis was used to compare query sequences with up to 5,478 amino acids. A maximum performance of 228.4 GCUPS was attained.

10.2. Liu et al. (2014): SWAPHI-LS

Liu et al. [2014] propose a strategy for executing the Gotoh algorithm for long sequences with one or more Intel Phis. As output, the optimal score is provided. The DP matrix is divided according to the BW method, and each block is further divided into small tiles of fixed size, which are distributed to a team of threads. Each thread calculates its tile in a vectorized way. The connection between multiple Intel Phis is done with the MPI offload model. In the tests, a host with 4 Intel Phis was used to compare sequences with up to 50 MBP. A maximum performance of 111.4 GCUPS was attained.

10.3. Wang et al. (2014): XSW

Wang et al. [2014] propose a strategy to execute the Gotoh algorithm to search for protein sequences in one Intel Phi, producing the optimal score as output. Thread parallelism (pthreads) is combined with SIMD parallelism in addition to an approach based on Rognes (Section 6.8). A subset of the database sequences is distributed to each thread, which calculates different DP matrices using SIMD instructions. The results were collected in an Intel Xeon Phi 7110 card (61 cores) with 244 threads. Sequences of up to 1,000 amino acids were compared to two genomic databases, including the Swiss-Prot, achieving a maximum GCUPS of 70.

10.4. Comparative View of the Intel Phi Approaches

Table V provides a comparative view of the Intel Phi approaches. These approaches compare long (Section 10.2) and short (Sections 10.1 and 10.3) DNA and protein sequences with affine gap, executing the Gotoh algorithm to retrieve the optimal score. With the exception of CUDAlign (Section 9.4), the GCUPS obtained with Intel Phis are already higher than those obtained with GPUs.

11. BIOLOGICAL SEQUENCE COMPARISON SOLUTIONS IN HYBRID PLATFORMS

In this section, we briefly discuss some recent approaches that execute Pairwise Sequence Comparisons using more than one type of processing unit.

11.1. Meng and Chaudhary (2010): FPGA + CPU

Meng and Chaudhary [2010] propose the use of a hybrid HPC system composed of FPGAs and CPUs to execute query x database comparisons. A master/slave architecture is designed; the master allocates local comparison coarse-grained tasks to the workers and merges the results. Tasks are assigned to the slaves in a manner that is proportional to the computing power of the device (CPU without SSE, CPU with SSE or FPGA). In addition, long sequence comparisons are executed in the CPU, due to FPGA space restrictions. The SSE implementation uses a column-based assignment similar to the one explained in Section 6.6. The FPGA implementation uses a unidirectional systolic array (Section 7.1). Query sequences of up to 1,420 amino acids were compared to three protein databases. The heterogeneous platform was composed of 10 CPUs-SSE (Dual Core AMD Opteron 2.2GHz), 1 FPGA (Xlinx Virtex II) and 1 Pentium IV 1.9GHz. In this platform, 11.13 GCUPS were attained.

11.2. Singh and Aruni (2011): GPU + CPU

Singh and Aruni [2011] proposed a strategy to locally compare protein sequences with the Gotoh algorithm in CPUs and GPUs in a coarse-grained way. The CPUs run the SWPS3 algorithm (Section 8.1), and the GPUs run an adapted version of CUDASW++ 2.0 (Section 9.3). In order to distribute work among CPUs and GPUs, it is empirically assumed that the performance of 4 SSE cores is equal to the performance of one GPU. In the tests, the Swiss-Prot database was compared to query sequences of up to 35,213 amino acids. The results show that 27 GCUPS can be obtained with one CPU (4 cores) and one GPU (Tesla C10600).

11.3. Mendonca and Melo (2013): GPU + CPU

In Mendonca and Melo [2013], a strategy is proposed to execute Gotoh comparisons in CPUs and GPUs that performs dynamic workload adjustment through replication. CUDASW++ 2.0 (Section 9.3) was executed in the GPUs and a modified version of Farrar (Section 6.7) was executed in the CPUs. Query sequences are compared to genomic databases, and a work unit is defined to be a query x database comparison. The workload is initially statically distributed, according to the relative computing power of each device (CPU or GPU). If a device becomes idle and all working units are already assigned, it selects a work unit that is being processed by another device and begins to compute it as well. Results were collected in two hosts, each with 2 GPUs NVIDIA GTX 580 and one CPU Intel i7 (4 cores). A set of query sequences (up to 5,000 amino acids) was compared to 5 genomic databases, including the Swiss-Prot. Using 4 GPUs and 8 cores, a maximum GCUPS of 172.82 was attained.

11.4. Liu et al. (2013): CUDASW++ 3.0, GPU + CPU

Liu et al. [2013] proposed the latest version of CUDASW++, which simultaneously uses CPUs and GPUs to compare protein sequences. Static load distribution based on clock frequencies, number of cores (CPUs) and number of SMs (GPUs) is used to assign work to the PUs. The CPU code is based on SWIPE (Section 6.8) and the GPU code uses CUDA PTX SIMD instructions, combined with the SWIPE approach to accelerate the computation. Results were collected in an Intel i7 3.5GHz (quad-core) combined with 3 GPUs: NVIDIA GTX 680 and NVIDIA GTX 690 (2 GPUs). Query sequences of up

Table VI. Comparative View of Hybrid Computing Environment Papers

		Problem					Algorithm				HPC solution	
Section	Year	Type Seq	Max Size	Type Align	Gap	Out	Algo	Mem Comp.	Matrix Comp	Paral	#PU	Perf GCUPS
FPGA + CPU Approaches												
11.1	2010	Prot	10^3	local	affine	score	Gotoh	N/A	full	coarse fine (DW)	1F 11C	11.30
GPU + CPU Approaches												
11.2	2011	Prot	10^4	local	affine	score	Gotoh	N/A	full SWAT	coarse fine (col)	1G 4C	27.00
11.2	2013	Prot	10^3	local	affine	score	Gotoh	N/A	full SWAT	coarse fine (col)	2G 8C	172.82
11.3	2013	Prot	10^3	local	affine	score	Gotoh	N/A	full SWAT	coarse fine (col)	2G 4C	185.60
Intel Phi + CPU Approaches												
11.4	2014	Prot	10^3	local	affine	score	Gotoh	N/A	full	coarse fine (DW)	1P 16C	62.60

to 5,478 amino acids were compared to the Swiss-Prot genomic database, achieving a maximum GCUPS of 185.6 (4 cores + GTX 690).

11.5. Rucci et al. (2014): Intel Phi + CPU

The main idea in Rucci et al. [2014] is to use the same basic implementation for both the Intel Phi and the CPU, programmed with OpenMP. In the basic implementation, query profiles (Section 6.6) are used, and the optimal score is output. Work is statically assigned to the CPU and to the Intel Phi based on their theoretical performance. Experimental results collected in a server with two 8-core Intel Xeon 2.6GHz and an Intel Xeon Phi (60 cores) comparing query sequences up to 5,478 amino acids achieved 62.6 GCUPS.

11.6. Comparative View of the Hybrid Approaches

Table VI provides a comparative view of the hybrid approaches. In all approaches, protein sequences are compared with affine gap, producing the score as output. All the CPU+GPU approaches use SWAT optimization. Two-level parallelization is used, in which a set of database sequences is first assigned to each device (coarse-grain). Then, each device compares a query sequence to its database sequences in a fine-grained manner, often using previously proposed algorithms/tools. Very good GCUPS are attained, ranging from 11.30 (2010) to 185.60 (2013). It is worthwhile to note that all hybrid approaches use CPUs combined with only one type of accelerator (F:FPGA, G:GPU, or P:Intel Phi).

12. PERFORMANCE EVOLUTION OF THE HPC APPROACHES FOR BIOLOGICAL SEQUENCE COMPARISON

In this section, we discuss the overall performance evolution of the Optimal Pairwise Sequence Comparison approaches over 25 years of research (1990 to 2014). Figure 10 provides, in logarithmic scale, the GCUPS evolution over the years. In this figure, we provide the maximum GCUPS and distinguish the solutions that use a single PU and

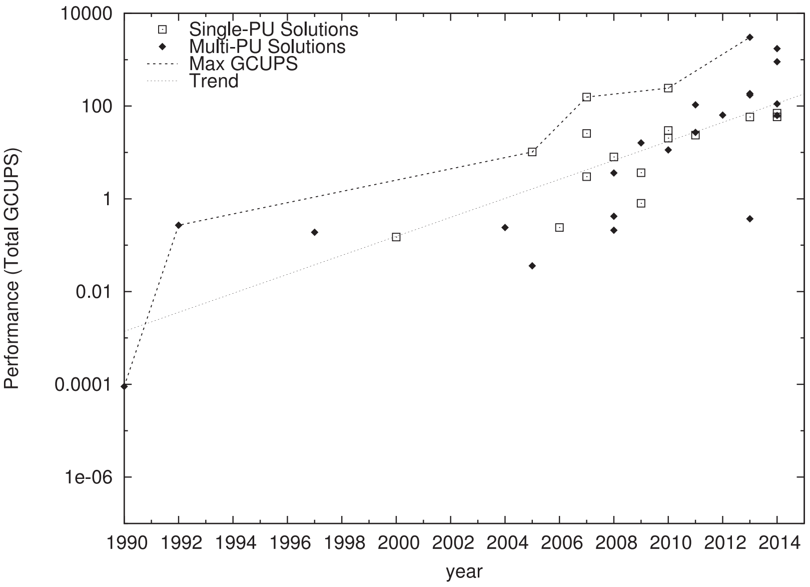

Fig. 10. Performance of the parallel approaches over the years (single and multiple PEs).

multiple PUs. A curve fitting gnuplot function that uses the nonlinear least-squares (NLLS) Marquardt-Levenberg algorithm was also utilized to generate the trend in logarithmic scale.

We may observe that the increase in GCUPS has undergone an exponential growth from 1990 to 2014, ranging from 0.0009 to 3,040, with an astonishing increase of 7 orders of magnitude. This increase was mainly due to the combination of two factors. First, there was a phenomenal technological evolution in CPUs and accelerators in the last 25 years, leading to a fantastic theoretical performance. Second, the research field of Pairwise Sequence Comparison has witnessed intense activity in the last decades, with great advancements in both (a) clever adaptation of the original sequence comparison algorithms and (b) sophisticated parallelization strategies. In our opinion, the combination of both factors made the achievement of TCUPS possible.

Analyzing the maximum GCUPS for single and multi-PU solutions, we note that, from 1990 to 2005, the best GCUPS were obtained by approaches that used multiple PUs. From 2005 to 2010, the focus seemed to be on improving single-PU solutions. Then, from 2010 to 2014, multiple PU solutions were again commonly used and multiple-PU solutions currently obtain the best GCUPs.

Figure 11 presents the performance in GCUPS per device for the parallel solutions discussed in this survey over the years. In this figure, it is very clear that CPU solutions that do not use SIMD instructions currently provide the worst performance. Even though CellBE solutions were able to provide performance comparable to GPUs in 2008, they were not able to obtain better performance after that year. FPGA/ASIC solutions were able to sustain the best maximum performance over the years. This can be explained by the combination of two factors: (a) the hardware is highly optimized, since it is designed specifically to do sequence comparison; and (b) the problem tackled is usually simple: linear-gap function, very small query sequences, and score as output. GPUs and CPUs with SIMD instructions solve more complex problem instances (affine gap, longer sequences) and have been able to provide very good improvement on the maximum performance achieved each year. Observing the maximum performance in

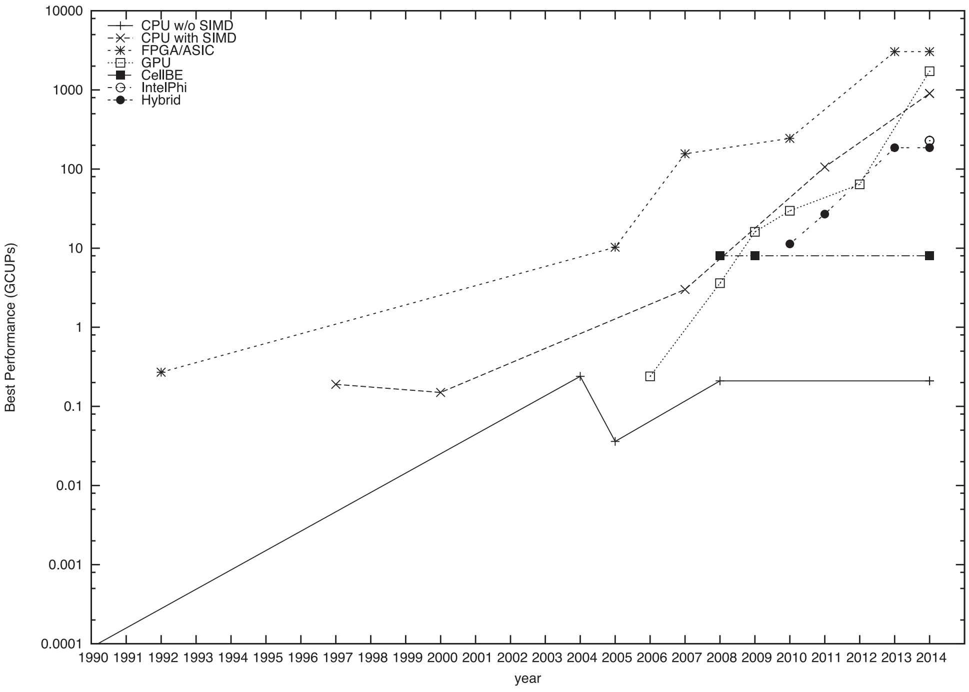

Fig. 11. Maximum performance of the parallel approaches over the years by device.

2014, it is probable that GPUs and/or CPUs with SIMD will be able to outperform FPGAs in the near future. The Intel Phi platform is also very promising: the first solutions were proposed in 2014 and the maximum performance for this device is already 228 GCUPs. Finally, hybrid solutions are also able to achieve good performance, but they require sophisticated load-balancing techniques that may negatively impact performance.

As can be seen in Figure 11, there are three approaches that obtained GCUPS higher than 500, using CPU (SSE), GPU, and FPGAs. Therefore, it is difficult to conclude which PU will lead to the best performance in the future. The number of PUs employed in these three approaches are 128 CPUs, 64 GPUs, and 128 FPGAs. This shows a strong trend towards using multiple PUs, which we think will continue in the near future. Since a lot of research has been done in CPU, GPU, and FPGA-based solutions, we think that these PUs will continue to be used.

Many researchers claim that hardware-based designs (FPGA/ASIC) require a lot of programming effort and that CPUs, GPUs, and Intel Phis should be preferred. It is true that it takes more time to obtain hardware-based designs. However, there is still active research in the area of hardware-based solutions for Biological Sequence Comparison, with impressive results; this research path should not be neglected.

Even though there has been active research in CellBE-based solutions, these solutions did not achieve the best GCUPS. We think that these results are due to the short life span of the CellBEs (2001–2010), the delay in proposing CellBE solutions, and the very small size of the SPE local memory, which increased the burden on communication. Surprisingly, hybrid solutions have not yet provided the best GCUPS. This type of solution is rather recent; thus, there is considerable room for improvement. In the near future, a lot of attention should also be paid to the Intel Phi solutions, which are already able to obtain more than 200 GCUPS.

13. CONCLUSION AND PERSPECTIVES

Pairwise Biological Sequence Comparison is a fundamental operation in bioinformatics, with the goal of identifying regions of similarity that can indicate functional and/or evolutionary relationships between the organisms. It is widely used as a building block in important problems such as genome assembly, phylogenetic/evolutionary analysis, and protein structure prediction, among others. In this context, the quality of the alignment is crucial, since it will greatly contribute to finding a good solution to these important problems. Optimal Pairwise Biological Sequence Comparison algorithms are able to obtain higher-quality alignments at the expense of additional execution time. Since the 1990s, it has become clear that the association between High-Performance Computing and Optimal Pairwise Comparisons is an excellent alternative.

In this article, we propose a categorization of the Optimal Pairwise Sequence Comparison approaches that use HPC platforms, and we discuss 35 papers in a uniform way, providing a landscape of the advances in this area in the last 25 years. We show that there was an astonishing increase in the performance of the approaches, and that the TCUPS landmark has been crossed.

With this achievement, one could think that most of the strategies have already been proposed in this research field and that there remain very few discoveries to be made. This is certainly not true. Even with TCUPS performance, one optimal pairwise comparison of whole human chromosomes took more than 9 hours to complete (Section 9.4) and the optimal comparison of short reads with the entire human genome took 29 hours (Section 7.7). This shows that this field has advanced a lot but there is still a great distance to cover. Furthermore, biological sequence data is being produced at an exponential rate; this phenomenon has been called "DNA data deluge" [Schatz and Langmead 2013]. In order to cope, new highly parallel pairwise comparison approaches/tools must be developed.

In the near future, we expect that massively parallel multi-PU approaches to pairwise sequence comparison that achieve ExaCUPS will become a reality. In order to achieve the development of such approaches, we claim that the potential of hybrid multi-PU HPC platforms should be better scrutinized, leading to impressive gains in performance. Finally, we think that innovative strategies that avoid unnecessary calculations and/or employ advanced concepts of linear algebra, PP, and others should be explored in depth, being able to cope with the DNA Data Deluge era.

REFERENCES

A. Aji and W. Feng. 2008. Optimizing performance, cost and sensitivity in pairwise sequence search on a cluster of playstation. In *IEEE International Conference on BioInformatics and BioEngineering*.

A. M. Aji, W. Feng, F. Blagojevic, and D. S. Nikolopoulos. 2008. Cell-SWAT: Modeling and scheduling wavefront computations on the cell broadband engine. In *Proceedings of the 5th Conference on Computing Frontiers (CF'08)*. 13–22.

S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology* 215, 3, 403–410.

S. Aluru and N. Jammula. 2014. A review of hardware acceleration for computational genomics. *IEEE Design and Test* 31, 1, 19–30.

R. B. Batista, A. Boukerche, and A. C. M. A. Melo. 2008. A parallel strategy for biological sequence alignment in restricted memory space. *Journal of Parallel and Distributed Processing* 68, 4, 548–561.

K. Benkrid, Y. Liu, and A. Benkrid. 2009. A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment. *IEEE Transactions on VLSI Systems* 17, 4, 561–570.

S. Borkar and A. A. Chien. 2011. The future of microprocessors. *Communications of the ACM* 54, 5, 57–67.

A. Boukerche, R. B. Batista, A. C. M. A. Melo, F. B. Scarel, and L. A. B. C. Souza. 2012. Exact parallel alignment of megabase genomic sequences with tunable work distribution. *International Journal of Foundations of Computer Science* 23, 2, 407–429.

A. Boukerche, J. M. Correa, A. C. M. A. Melo, and R. P. Jacobi. 2010. A hardware accelerator for the fast retrieval of DIALIGN biological sequence alignments in linear space. *IEEE Transactions on Computers* 59, 6, 808–821.

G. Caffarena, C. Pedrera, C. Carreras, S. Bojanic, and O. Nieto-Taladriz. 2007. FPGA acceleration for DNA sequence alignment. *Journal of Systems and Software* 16, 2, 245–266.

C. Chen and B. Schmidt. 2005. An adaptive grid implementation of DNA sequence alignment. *Future Generation Computer Systems* 21, 7, 988–1003. http://dx.doi.org/10.1016/j.future.2005.03.001

V. Chvatal and D. Sankoff. 1975. Longest common subsequences of two random sequences. Technicial Report. STAN-CS-75-477, Stanford University, Stanford, CA.

K. Compton and S. Hauck. 2002. Reconfigurable computing: A survey of systems and software. *Computing Surveys* 34, 2, 171–210.

A. Driga, P. Lu, J. Schaeffer, D. Szafron, K. Charter, and I. Parsons. 2006. FastLSA: A fast, linear-space, parallel and sequential algorithm for sequence alignment. *Algorithmica* 45, 3, 337–375.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison 1998. *Biological Sequence Analysis*. Cambridge University Press, New York, NY.

S. R. Eddy. 1998. Profile hidden Markov models. *Bioinformatics* 14, 9, 755–763.

F. J. Esteban, D. David, P. Hernandez, J. A. Caballero, G. Dorado, and S. Galvez. 2013. Direct approaches to exploit many-core architecture in bioinformatics. *Future Generation Computing Systems* 29, 15–26.

ExPASy. 2014a. UniProtKBSwissProt protein knowledgebase guideline. http://web.expasy.org/docs/swiss-prot_guideline.html.

ExPASy. 2014b. UniProtKBSwissProt protein knowledgebase release 2015_04 statistics. ftp://ftp.uniprot.org/pub/databases/uniprot/previous_major_releases/release-2015_04/knowledgebase/UniProtKB_SwissProt-relstat.html.

M. Farrar. 2007. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics* 23, 2, 156–161.

J. W. Fickett. 1984. Fast optimal alignments. *Nucleic Acids Research* 1, 12, 175–179.

A. R. Galper and E. C. Brutlag. 1990. Parallel similarity search and alignment with the dynamic programming method. Technical Report. KSL-90-74, Stanford University, Stanford, CA.

O. Gotoh. 1982. An improved algorithm for matching biological sequences. *Journal of Molecular Biology* 162, 3, 705–708.

P. Green. 1993. SWAT Optimization. (1993). Retrieved February 26, 2016 from http://www.phrap.org/phredphrap/swat.html.

X. Guan and D. L. Uberbacher. 1994. A multiple divide-and-conquer (MDC) algorithm for optimal alignments in linear space. Technical Report. ORNL/TM-12764, Oak Ridge National Laboratory, Oak Ridge, TN.

D. Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY.

D. S. Hirschberg. 1975. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM* 18, 6, 341–343. DOI:http://dx.doi.org/10.1145/360825.360861

D. T. Hoang and D. P. Lopresti. 1992. FPL implementation of systolic sequence alignment. In *International Workshop on Field-Programmable Logic and Applications, Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*. 183–191.

M. Hoeppner, M. Latterner, and K. Siyan 2013. *The NCBI Handbook*. National Center for Biotechnology Information, Bethesda, MD.

IBM. 2007. Cell broadband engine architecture. (2007). http://www.ibm.com/developerworks/library/pa-cellperf/.

F. Ino, Y. Kotani, Y. Munekawa, and K. Hagihara. 2009. Harnessing the power of idle GPUs for acceleration of biological sequence alignment. *Parallel Processing Letters* 19, 4, 513.

F. Ino, Y. Munekawa, and K. Hagihara. 2012. Sequence homology search using fine grained cycle sharing of idle GPUs. *IEEE Transactions on Parallel Distributed Systems* 23, 4, 751–759. http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.239

J. Jeffers and J. Reinders. 2013. *Intel Xeon Phi Coprocessor High-Performance Programming*. Morgan Kaufmann, Burlington, MA.

D. Kirk and W. Hwu. 2010. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, Burlington, MA.

M. Korpar and M. Sikic. 2013. SW#-GPU-enabled exact alignments on genome scale. *Bioinformatics* 29, 19, 2494–2495.

H. T. Kung. 1982. Why systolic architectures? *IEEE Computer* 15, 1, 37–42.

V. I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10, 8, 707–710.

R. J. Lipton and D. Lopresti. 1985. A systolic array for rapid string comparison. In *Chapel Hill Conference on VLSI*, Vol. 3994. 363–376.

Y. Liu, W. Huang, J. Johnson, and S. Vaidya. 2006. GPU accelerated Smith-Waterman. In *ICCS 2006 (LNCS)*, Vol. 3994. Springer, 188–195.

Y. Liu, D. Maskell, and B. Schmidt. 2009. CUDASW++: Optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. *BMC Research Notes* 2, 1, 73.

Y. Liu and B. Schmidt. 2014a. GSWABE: Faster GPU-accelerated sequence alignment with optimal alignment retrieval for short DNA sequences. *Concurrency and Computation: Practice and Experience—early view* (2014).

Y. Liu and B. Schmidt. 2014b. SWAPHI: Smith-Waterman protein database search on Xeon Phi coprocessors. In *IEEE ASAP*. 184–185.

Y. Liu, B. Schmidt, and D. Maskell. 2010. CUDASW++2.0: Enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. *BMC Research Notes* 3, 1, 93. DOI:http://dx.doi.org/10.1186/1756-0500-3-93

Y. Liu, T. Tam, F. Lauenroth, and B. Schmidt. 2014. SWAPHI-LS: Smith-Waterman algorithm on Xeon Phi coprocessors for long DNA sequences. In *IEEE CLUSTER*. 257–265.

Y. Liu, A. Wirawan, and B. Schmidt. 2013. CUDASW++ 3.0: Accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. *BMC Bioinformatics* 14, 117.

S. Maleki, M. Musuvathi, and T. Mytcowicz. 2014. Parallelizing dynamic programming through rank convergence. In *ACM SIGPLAN PPoPP*. ACM, New York, NY, 219–232.

S. Manavski and G. Valle. 2008. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics* 9, Suppl 2.

K. X. Mayer et al. 2014. A chromosome-based draft sequence of the hexaploid bread wheat (triticum aestivum) genome. *Science* 345, 6194.

F. Mendonca and A. C. M. A. Melo. 2013. Biological sequence comparison on hybrid platforms with dynamic workload adjustment. In *IEEE IPDPSW - HiCOMB*. 501–509.

X. Meng and V. Chaudhary. 2010. A high-performance heterogeneous computing platform for biological sequence analysis. *IEEE Transactions on Parallel Distribted Systems* 21, 9, 1267–1280.

D. Morgenstern, K. Frech, A. Dress, and T. Werner. 1998. DIALIGN: Finding local similarities by multiple sequence alignment. *Bioinformatics* 14, 3, 290–294.

D. W. Mount. 2004. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, Cold Spring Harbor, NY.

A. Munshi, B. Gaster, T. G. Mattson, J. Fung, and D. Ginsburg. 2011. *OpenCL Programming Guide*. Addison-Wesley Professional, New York, NY.

E. W. Myers and W. Miller. 1988. Optimal alignments in linear space. *Computer Applications in the Biosciences* 4, 1, 11–17.

S. B. Needleman and C. D. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48, 3, 443–453.

nuccore. 2014. NCBI Nucleotide Database. (2014). Retrieved February 26, 2016 from http://www.ncbi.nlm.nih.gov/nuccore/.

NVIDIA. 2015. CUDA C Progamming Guide. https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

T. Oliver, B. Schmidt, and D. L. Maskell. 2005. Reconfigurable architectures for bio-sequence database scanning on FPGAs. *IEEE Transactions on Circuits and Systems* 52, 12, 851–855.

S. Rajko and S. Aluru. 2004. Space and time optimal parallel sequence alignments. *IEEE Transactions on Parallel Distribution Systems* 15.

T. Rognes. 2011. Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. *BMC Bioinformatics* 12, 221.

T. Rognes and E. Seeberg. 2000. Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics* 16, 8, 699–706.

E. Rucci, A. Giusti, M. Naioufi, G. Botella, and C. Garcia. 2014. Smith-Waterman algorithm on heterogeneous systems: A case study. In *IEEE CLUSTER*. 323–330.

F. Sanchez, F. Cabarcas, A. Ramirez, and M. Valero. 2011. Scalable multicore architectures for long DNA sequence comparison. *Concurrency and Computation: Practice and Experience* 23, 17, 2205–2219.

E. F. de O. Sandes and A. C. M. A. Melo. 2011. Smith-Waterman alignment of huge sequences with GPU in linear space. In *IEEE IPDPS*. 1199–1211.

E. F. de O. Sandes, G. Miranda, A. C. M. A. Melo, X. Martorell, and E. Ayguade. 2014. CUDAlign 3.0: Parallel biological sequence comparison in large GPU clusters. In *IEEE/ACM CCGrid*. 160–169.

E. F. O. Sandes and A. C. M. A. Melo. 2010. CUDAlign: Using GPU to accelerate the comparison of megabase genomic sequences. In *ACM SIGPLAN PPoPP*. ACM, New York, NY, 137–146.

E. F. O. Sandes and A. C. M. A. Melo. 2013. Retrieving Smith-Waterman alignments with optimizations for megabase biological sequences using GPU. *IEEE Transactions on Parallel Distribution Systems* 24, 5, 1009–1021.

A. Sarje and S. Aluru. 2009. Parallel genomic alignments on the cell broadband engine. *IEEE Transactions on Parallel Distribution Systems* 20, 11, 1600–1610.

S. Sarkar, G. R. Kulkarni, P. P. Pande, and A. Kalyanaraman. 2010a. Network-on-chip hardware accelerators for biological sequence alignment. *IEEE Transactions on Computers* 59, 1, 29–41.

S. Sarkar, T. Majumder, A. Kalyanaraman, and P. P. Pande. 2010b. Hardware accelerators for biocomputing: A survey. In *IEEE International Symposium on Circuits and Systems (ISCAS'10)*. IEEE, 3789–3792.

M. C. Schatz and B. Langmead. 2013. The DNA data deluge. *IEEE Spectrum* 50, 7, 28–33.

J. Singh and I. Aruni. 2011. Accelerating Smith-Waterman on heterogeneous CPU-GPU systems. In *5th International Conference on Bioinformatics and Biomedical Engineering (ICBBE'11)*.

S. F. Smith and J. F. Frenzel. 2003. Bioinformatics application of a scalable supercomputer-on-chip architecture. In *PDPTA*. 385–391.

T. F. Smith and M. S. Waterman. 1981. *Journal of Molecular Biology* 147, 1, 195–197.

A. Szalkowski, C. Ledergerber, P. Krahenbuhl, and C. Dessimoz. 2008. Swps3—fast multi-threaded vectorized Smith-Waterman for IBM CellBE. and x86 SSE2. *BMC Research Notes* 107–110.

L. Wang, Y. Chan, X. Duan, H. Lan, X. Meng, and W. Liu. 2014. XSW: Accelerating biological database search on Xeon Phi. In *IEEE IPDPSW—AsHES*. 950–957.

L. Wienbrandt. 2013. Bioinformatics applications on the FPGA-based high-performance computer RIVYERA. In *High-Performance Computing Using FPGAs*. Springer, 81–103.

W. Wolf. 2004. *FPGA-Based System Design*. Prentice-Hall, Upper Saddle River, NJ.

A. Wozniak. 1997. Using video-oriented instructions to speed up sequence comparison. *Computer Applications in the Biosciences* 13, 2, 145–150.

P. Zhang, G. Tan, and G. R. Gao. 2007. Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform. In *ACM HPRCTA*. ACM, New York, NY, 39–48.

Received April 2015; revised November 2015; accepted January 2016