2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

CUDAlign 3.0: Parallel Biological Sequence Comparison in Large GPU Clusters

Edans F. de O. Sandes*, Guillermo Miranda†, Alba C. M. A. de Melo*, Xavier Martorell†‡, Eduard Ayguadé†‡
*University of Brasilia (UnB)
{edans, albamm}@cic.unb.br
†Barcelona Supercomputing Center (BSC)
{guillermo.miranda, xavier.martorell, eduard.ayguade}@bsc.es
‡Universitat Politècnica de Catalunya (UPC)
{xavim, eduard}@ac.upc.edu

***Abstract*—This paper proposes and evaluates a parallel strategy to execute the exact Smith-Waterman (SW) biological sequence comparison algorithm for huge DNA sequences in multi-GPU platforms. In our strategy, the computation of a single huge SW matrix is spread over multiple GPUs, which communicate border elements to the neighbour, using a circular buffer mechanism. We also provide a method to predict the execution time and speedup of a comparison, given the number of the GPUs and the sizes of the sequences. The results obtained with a large multi-GPU environment show that our solution is scalable when varying the sizes of the sequences and/or the number of GPUs and that our prediction method is accurate. With our proposal, we were able to compare the largest human chromosome with its homologous chimpanzee chromosome (249 Millions of Base Pairs (MBP) x 228 MBP) using 64 GPUs, achieving 1.7 TCUPS (Tera Cells Updated per Second). As far as we know, this is the largest comparison ever done using the Smith-Waterman algorithm.**

I. Introduction

In comparative genomics, biologists need to compare their sequences against other organisms in order to infer functional and structural properties. Sequence comparison is, therefore, one of the most basic operations in Bioinformatics [1], usually solved using heuristic methods due to the excessive execution times of their exact counterparts.

The exact algorithm to execute pairwise comparisons is the one proposed by Smith-Waterman (SW) [2], which is based on Dynamic Programming (DP), with quadratic time and space complexities. The SW algorithm is normally executed to compare (a) two DNA sequences or (b) a protein sequence (query sequence) to a genomic database, which is composed of several protein sequences. Both cases have been parallelized in the literature. In the first case, a single SW matrix is calculated and all the Processing Elements (PEs) participate in this calculation (fine-grained computation). Since there are data dependencies, neighbour PEs communicate in order to exchange border elements. For Megabase DNA sequences, the algorithm calculates a huge matrix with several Petabytes. In the second case, multiple small SW matrices are calculated usually without communication between the PEs (coarse-grained computation). With the current genomic databases, often hundreds of thousands SW matrices are calculated in a single *query* $\times$ *database* comparison.

GPUs (Graphics Processing Units) are highly parallel architectures which execute data parallel problems usually much faster than a general-purpose processor. For this reason, they have been considered to accelerate SW, with many versions already available, executing on a single GPU [3–7]. More recently, several approaches have been proposed to execute SW in multiple GPUs [8–12].

Very few GPU strategies [3, 12] allow the comparison of Megabase sequences longer than 10 Million Base Pairs (MBP). SW# [12] is able to use 2 GPUs in a single Megabase comparison to calculate the Myers-Miller [13] linear space variant of SW. CUDAlign [3] executes in a single GPU and obtains the alignment of Megabase sequences with a combined SW and Myers-Miller strategy. When compared to SW# (1 GPU), CUDAlign (1 GPU) presents better execution times for huge sequences [12].

In this paper, we propose and evaluate CUDAlign 3.0, an evolution of CUDAlign 2.1 [3] which executes the first stage of the SW algorithm in a fine-grained parallel way, comparing Megabase DNA sequences in multiple GPUs. In CUDAlign 3.0, we faced the challenge of distributing the computation of a huge DP matrix among several GPUs, with low impact on the performance. In the proposed strategy, GPUs are logically arranged in a linear way so that each GPU calculates a subset of columns of the SW matrix, sending the border column elements to the next GPU.

Due to the data dependencies of the SW recurrence relation, a slowdown in the communication between any 2 GPUs will slowdown the whole matrix computation [14]. To tackle this problem, we decided that computation must be overlapped with communication, so asynchronous CPU threads will send/receive data to/from neighbor GPUs while the GPU continues computing.

Sequence comparisons that deal with Megabase sequences can take hours or even days to complete. In this scenario, we developed a method to predict the execution time and speedup of a comparison, given the number of the GPUs and the sizes of the sequences.

CUDAlign 3.0 was implemented in CUDA, C++ and

978-1-4799-2784-5/14 $31.00 © 2014 IEEE
DOI 10.1109/CCGrid.2014.18

pthreads. Experimental results collected in a large GPU cluster using real DNA sequences show that the proposed solution is scalable and is able to obtain close to linear speedups for up to 16 GPUs. For example, we reduced the execution time of the human x chimpanzee chromosome 22 comparison (63 Millions of Base Pairs (MBP) x 62 MBP) from 33h20min in a single GPU to 2h13min when using 16 GPUs (14.8x speedup). We also evaluated our prediction method showing that, for the sequences compared, the maximum time prediction error was less than 0.45% considering 1 GPU and below 5% for the speedup prediction error.

The comparison of the largest human chromosome (chromosome 1) with its homologous chimpanzee chromosome (249 MBP x 228 MBP) was executed using 16, 32 and 64 GPUs. The 64-GPU execution calculated an SW matrix of 57 Petacells in 9 hours and 9 minutes with 32,768 CUDA cores.

The contributions of this paper are, thus, three-fold. First, we propose and evaluate CUDAlign 3.0, which is able to compare Megabase DNA sequences in multiple GPUs. Second, we show that exact methods such as Smith-Waterman can now be used to compare, in reasonable time, even two of the largest chromosomes in nature, such as the human and chimpanzee chromosomes 1. Third, we propose and evaluate a prediction method able to estimate the execution time and speedup of a comparison with very good accuracy.

The rest of this paper is organized as follows. In Section II we present the biological sequence comparison problem and the SW algorithm. In Section III we discuss different implementations of SW using multiple Processing Elements. Section IV briefly reviews the CUDAlign algorithm and we describe our multi-GPU strategy in Section V. In Section VI we present experimental results. Finally, in Section VII, we conclude the paper and outline future work.

II. DNA Sequence Comparison

A DNA sequence is represented by an ordered list of nucleotide bases. DNA sequences are treated as strings composed of characters of the alphabet $\sum = \{A, T, G, C\}$. To compare two sequences, we place one sequence above the other, possibly introducing spaces (gaps), making clear the correspondence between similar characters [1]. The result of this placement is an alignment.

Given an alignment between sequences S_0 and S_1, a score is associated to it as follows. For each pair of characters, we associate (a) a punctuation *ma*, if both characters are identical (*match*); or (b) a penalty *mi*, if the characters are different (*mismatch*); or (c) a penalty g, if one of the characters is a space (*gap*). The score is the addition of all these values. The maximal score is called the similarity between the sequences. Figure 1 presents one possible alignment between two DNA sequences. In this figure, $ma = +1$, $mi = -1$ and $g = -2$.

$$\begin{array}{c} T\ A\ T\ G\ A\ G\ G\ T \\ T\ A\ T\ -\ A\ G\ G\ T \\ \hline +1{+}1{+}1{-}2{+}1{+}1{+}1{+}1 \end{array}$$
$$score = 5$$

Figure 1. Example of alignment and score

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

Figure 2. Similarity matrix for sequences S_0 and S_1

A. Smith-Waterman (SW) Algorithm

The algorithm SW [2] is based on Dynamic Programming (DP), obtaining the optimal pairwise local alignment in quadratic time and space. It is divided in two phases: create the similarity matrix and obtain the alignment.

The first phase receives as input sequences S_0 and S_1, with sizes $|S_0| = m$ and $|S_1| = n$. The DP matrix is denoted $H_{m+1,n+1}$, where $H_{i,j}$ contains the score between prefixes $S_0[1..i]$ and $S_1[1..j]$. At the beginning, the first row and column are filled with zeroes. The remaining elements of H are obtained from Equation 1.

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + (\text{if } S_0[i] = S_1[j] \text{ then } ma \text{ else } mi) \\ H_{i,j-1} + \text{g} \\ H_{i-1,j} + \text{g} \\ 0 \end{cases} \quad (1)$$

In addition, each cell $H_{i,j}$ contains information about the cell that was used to produce the value. The highest value in $H_{i,j}$ is the optimal local score.

The second phase obtains the optimal local alignment, using the outputs of the first phase. The computation starts from the cell that has the highest value in H, following the path that produced the optimal score until the value zero is reached.

Figure 2 presents a DP matrix with $score = 5$. In this case, two DNA sequences with sizes $m = 12$ and $n = 8$ are compared, resulting in a 9×13 matrix. In order to compare Megabase sequences of, for instance, 60 MBP a matrix of size 60,000,001 x 60,000,001 (3.6 Peta cells) is calculated. Considering that each cell has 4 bytes, 14.4 Peta bytes are used.

The original SW algorithm assigns a constant cost g to each gap. However, gaps tend to occur together rather than

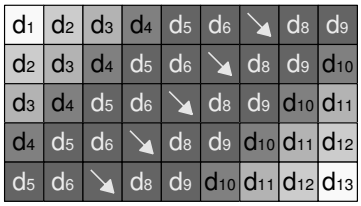

Figure 3. Wavefront Method [17]

individually. For this reason, a higher penalty is usually associated to the first gap and a lower penalty is given to the remaining ones (affine-gap model). Gotoh [15] proposed an algorithm based on SW that implements the affine-gap model by calculating 3 values for each cell in the DP matrix: H, E and F, where values E and F keep track of gaps in each sequence.

B. Parallel Smith-Waterman

In SW, most of the time is spent calculating the DP matrices and this is the part which is usually parallelized. In Equation 1, we can notice that cell $H_{i,j}$ depends on three other cells: $H_{i-1,j}$, $H_{i-1,j-1}$ and $H_{i,j-1}$. This kind of dependency is well suited to be parallelized using the wavefront method [16]. Using this method, the DP matrix is calculated by diagonals, and all cells in each diagonal can be computed in parallel.

Figure 3 illustrates the wavefront method. In step 1, only one cell is calculated in diagonal d_1. In step 2, diagonal d_2 has 2 cells, that can be calculated in parallel. In the further steps, the number of cells that can be calculated in parallel increases until it reaches the maximum parallelism in diagonals d_5 to d_9, where 5 cells are calculated in parallel. In diagonals d_{10} to d_{12}, the parallelism decreases until only one cell is calculated in diagonal d_{13}. The wavefront strategy suffers from reduced parallelism during the beginning of the calculation (filling the wavefront) and the end of the computation (emptying the wavefront).

III. Related Work

Table I lists 10 proposals of SW implementations for High Performance Computing (HPC) platforms with multiple Processing Elements (PE). The first one runs on a cluster of PlayStation 3 using the IBM Cell/B.E. processor. Three other proposals run on clusters of single/multi-cores and the last six on multiple GPUs. We must note that there are many other cluster implementations of SW in the literature. We chose these three approaches because they have distinct characteristics.

Column 2 identifies the type of computation: coarse- or fine-grained. The proposals that we considered fine-grained are the ones that use more than one PE to compute the same DP matrix; otherwise, the proposal is classified as coarse-grained. In Table I, it is clear that there are many cluster proposals that execute fine-grained comparisons. In GPUs and Clusters of Cell/B.E.s, the coarse-grained approach has been used in most of the papers.

Column 3 compares the work assignment policies used in the proposals. In the fixed approach, the work is statically assigned for each PE whereas in the self-scheduling approach the slave PEs dynamically retrieve work units from the master PE.

Column 4 provides the maximum number of PEs reported in the experimental sections of the papers. It can be seen that this number varies a lot. In the cluster approaches, the number of PEs varies from 18 to 6,144. In the GPU approaches, the number of GPUs varies from 2 to 10.

To measure the performance of SW, the metric GCUPS (*Billions of Cell Updated per Second*) is often used. This metric calculates the rate at which the cells of the DP matrix are updated. GCUPS marked with an "*" (column 5) were not provided in the papers but calculated with the sizes of the sequences and the reported execution times. GCUPS range from 0.25 (clusters) to 185.6 (GPUs). The GCUPS values provided in this table cannot be compared directly because they were obtained in different platforms, with different sequences. Nevertheless, they provide an indication of the potential of each platform. In a broad way, we can see that GPUs are able to produce much better GCUPS rates than the other platforms listed in the table.

To our knowledge, the best GCUPS to compare query sequences to a genomic database using platforms with one GPU was obtained by CUDASW++ 3.0 (119 GCUPS) [21]. Note that in this case, one GPU and 4 cores (executing SIMD operations) were jointly used. The best GCUPS to compare DNA sequences with one GPU was obtained by CUDAlign 2.1 [3]. When comparing sequences of 33 MBP $\times$ 46 MBP in a GTX 560Ti, it was able to attain 52.85 GCUPS.

Finally, the maximum size of the sequences compared is shown in column 6. When a *query* $\times$ *database* comparison is made, the size of the longest query sequence compared is provided. As can be seen from the table, the maximum size of the query sequence is often very small. Using two GPUs, SW# [12] was able to compare sequences of 33 MBP, the same maximum size compared by CUDAlign 2.1 with one GPU [3].

The last row of Table 1 presents the characteristics of this paper. With a fine-grained strategy that can be used in multi-GPU platforms, CUDAlign 3.0 was able to achieve 1726 GCUPS (or 1.72 TCUPS) and compared sequences of more than 200 MBP.

IV. Overview of CUDAlign

CUDAlign 1.0 [17] was proposed as a linear-space parallel algorithm able to compare Megabase DNA sequences in single-GPU systems. Since then, CUDAlign 2.0 [22] and 2.1 [3] were proposed, with several improvements for single GPUs. CUDAlign aligns two Megabase sequences

Table I
COMPARATIVE VIEW OF THE SW PROPOSALS FOR HPC PLATFORMS WITH MULTIPLE PROCESSING ELEMENTS

Ref.	Type Comp.	Work Assign	Number of PEs	GCUPS	Max. Size
Aji(2008)[18]	coarse/fine	Fixed	14 × Cell/B.E.s	0.42	3,584
Chen(2005)[14]	fine	Fixed	18 × CPU Cores	*0.37	132,290
Rajko(2004)[19]	fine	Fixed	60 × CPU Cores	*0.25	1,100,000
Hamidouche(2013)[20]	fine	Fixed	6,144 × CPU Cores	15.50	24,894,269
Manavsky(2008)[8]	coarse	Fixed	2×GPUs	3.61	567
Saeed(2010)[9]	coarse	Fixed	4×GPUs	1.30	1,024
Blazewicz(2011)[10]	coarse	Self-Sched	4×GPUs	11.68	1,143
Ino(2012)[11]	coarse	Self-Sched	8×GPUs	64.00	367
Liu(2013)[21]	coarse	Fixed	4 × CPU Cores + 2×GPUs	185.60	5,478
Kopar(2013)[12]	fine	Fixed	2×GPUs	*65.20	32,799,110
CUDAlign 3.0	**fine**	**Fixed**	**64×GPUs**	**1762.00**	**228,333,871**

in 6 stages. Stage 1 corresponds to the first phase of SW algorithm (Section II). It processes the whole DP matrix in order to obtain the optimal score and its position, using the affine-gap model (Section II). Stages 2 to 5 correspond to the second phase of SW, where the DP matrix is reprocessed in order to find the points that belong to the optimal alignment. Stage 6 is used to visualize the optimal alignment.

The rest of this section focuses on Stage 1, the most compute-intensive stage of CUDAlign [3] and the first candidate to be adapted to multi-GPU platforms.

The algorithm receives sequences S_0 and S_1 as input, with sizes m and n respectively. First, the DP matrix is divided in a grid with $\frac{n}{\alpha T} \times B$ blocks, where B is the number of CUDA blocks executed in parallel and T is the number of threads in each block. Each thread is responsible to process α rows of the matrix, so each block processes αT rows. Each diagonal of blocks is called external diagonal and each diagonal of threads, inside each block, is called internal diagonal.

To reduce the execution time of Stage 1, two optimizations were proposed [17]: Cells Delegation and Phase Division. Cells delegation avoids the wavefront (Section II-B) to be emptied and filled at each external diagonal calculation, providing full parallelism during almost all the execution. In order to do this, the GPU blocks have a parallelogram shape, instead of the typical rectangular one. The next block processes the pending cells of its left block. Additionally, blocks in the right-most column delegate cells to left-most blocks.

To avoid concurrency hazards, each external diagonal is processed in two phases: short phase and long phase. The short phase calculates the first T internal diagonals of the block and the long phase calculates the remaining internal diagonals of the block. Each phase corresponds to a kernel, and a more optimized kernel is used in the long phase, achieving faster execution time.

The memory accesses are designed to better fit in the CUDA architecture. Two main data structures are used. The *horizontal bus* is the area of the GPU global memory used to store the last row of each block. This area is used to transfer values to the block below, that will be processed in the next external diagonal. The *vertical bus* is the area of the GPU global memory used to store the last column processed by each thread. This area is used to transfer values to the block on the right, that will also be processed in the next external diagonal. Besides that, CUDAlign uses the read-only texture memory to read the input sequences more efficiently.

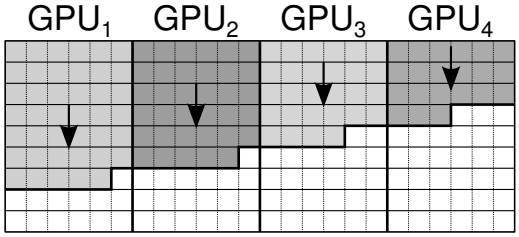

Figure 4. Columns distributions for 4 GPUs.

V. PROPOSED MULTI-GPU STRATEGY

CUDAlign 3.0 is an evolution of CUDAlign 2.1 (Section IV) that uses multiple GPUs to parallelize the computation of Stage 1. It parallelizes the computation of a single huge DP matrix among many GPUs and the data dependency between GPUs creates a path of communication between them. Since CUDAlign 3.0 is not an embarrassingly parallel application, the computation/communication ratio was carefully analyzed in order to produce good results.

The parallelization is done using a multi-GPU wavefront method, where the GPUs are logically arranged in a linear way, i.e, the first GPU is connected to the second, the second to the third and so on (Figure 4). Each GPU computes a range of columns of the DP matrix and the GPUs transfer the cells of their last column to the next GPU. These cells are computed by the short phase kernel (Section IV) and stored in global memory for further transfer to the next GPU. Although the blocks have a parallelogram shape (Section IV), the last column of each block is a straight line, so it is rectified by the short phase kernel. This rectification permits the communication between GPUs with equal or different block sizes.

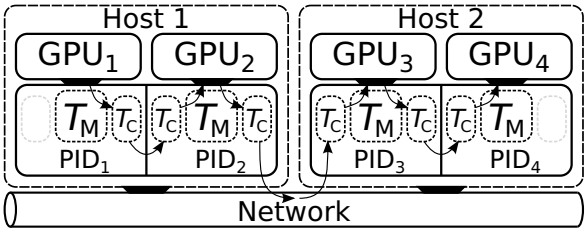

Figure 5. Multi-GPU threads chaining.

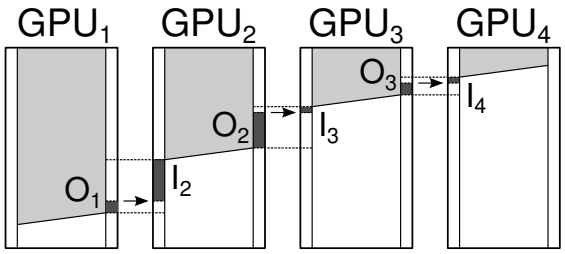

Figure 6. Multi-GPU communication buffers.

In this paper we use a static and even distribution of columns, since we are focusing on a large cluster environment with homogeneous dedicated GPUs. If we consider heterogeneous environment, the column distribution works as well using a weighted proportion that considers the computational power of each GPU.

A. Multi-Threaded Design

In CUDAlign 3.0, each GPU is bound to one process and each process has 3 CPU threads: one manager thread, that manages computation in GPU, and two communication threads, that handle inter-process communication.

Iteratively, the manager thread invokes the GPU kernels for the current external diagonal and transfers cells of the first/last column between GPU and CPU. The short phase kernel (Section IV) of CUDAlign 2.1 was modified to read/write the first/last columns from global memory, allowing the manager thread to transfer cells between GPU/CPU. Then, the manager thread interacts with the communication threads in order to receive its first column from the previous GPU and to send its last column to the next GPU. The transfers are made over chunks of cells with the same height of the block. Considering that each cell has two 32-bit words, each chunk has $8 \cdot \alpha \cdot B \cdot T$ bytes.

The inter-process communication is made by sockets that transfer cells from one process to the next. If one host contains more than one GPU, CUDAlign 3.0 forks one process per local GPU, and the connection is handled by loopback sockets. If CUDAlign 3.0 is executed in different hosts, then the connections are made using TCP sockets. Figure 5 illustrates the communication of 4 GPUs, where each host has two GPUs and each GPU has an associated process, with one manager thread (T_M) and two communication threads (T_C). The communication between GPUs 2 and 3 is made through the network and the communication between GPUs 1 and 2 and between GPUs 3 and 4 use the localhost loopback.

B. Input/Output Buffers

Each communication thread is associated to a circular buffer, that can be used as an input buffer (if it receives the column from the previous process) or as an output buffer (if it stores the column to be sent to the next process). If the input buffer is full or the output buffer is empty, its respective communication thread blocks until the status of the buffer changes. On the other hand, if the input buffer is empty or the output buffer is full, the manager thread blocks. It is important that the manager threads do not block often, in such a way that data can be continuously processed in all the GPUs and the inter-process communication be overlapped with the computation.

Figure 6 illustrates the communication buffers between 4 GPUs. Buffers I_2, I_3 and I_4 are the input buffers and buffers O_1, O_2 and O_3 are the output buffers. Each output-input pair of buffers $(O_{j-1} \rightarrow I_j)$ from successive GPUs is continually transferring data. These buffers are responsible to hide the inter-process communication in such a way that the overhead and small variations in the network may not be perceptible to the wavefront performance. Also, the amount of data waiting in each buffer may be an indication of the balancing quality. For instance, the pair of buffers $(O_1 \rightarrow I_2)$ in Figure 6 has more data waiting in the input buffer I_2 than in the output buffer O_1, giving an indication that GPU_1 is producing data faster than GPU_2 can process. In the pair of buffers $(O_3 \rightarrow I_4)$, the buffers are almost empty, indicating that GPU_3 and GPU_4 are processing in almost the same speed.

Note that when the multi-GPU wavefront is balanced, the buffer usage is almost constant. In this situation, the amount of data produced by one GPU in a given period of time is completely consumed by the next GPU, and the next GPU always has the necessary amount of data in the input buffers. When the multi-GPU wavefront is unbalanced, the output buffers may increase until they become full or the input buffers may decrease until they become empty, slowing down the computation due to blocking in the manager thread.

C. Prediction Method

In large GPU clusters, we need to specify the maximum execution time of an application. For executions that take hours, underpredicting the execution time would terminate the application prematurely and overpredicting may reduce the application priority in the queue. Therefore, a prediction method is important for efficient resource allocation. More-

over, speedup prediction helps to determine the appropriate number of GPUs in order to obtain a good parallel efficiency.

Considering that sequences S_0 and S_1 have sizes $n = |S_0|$ and $m = |S_1|$, the SW algorithm fits in the $O(mn)$ complexity class [2]. Even though most of the time is spent in the SW matrix calculation, there are many other activities, such as initialization of the data structures, read the input sequences, write the results, that must be taken into consideration. Therefore, the execution time of an SW comparison in a single GPU can be predicted with the time prediction formula shown in Equation 2, where c_1, c_2, c_3 and c_4 are constants related to each GPU environment. Constant c_1 corresponds to initialization procedures. Constants c_2 and c_3 correspond to the time spent for operations in $O(m)$ or $O(n)$ time complexity. Constant c_4 corresponds mainly to the matrix computation. The constants c_1, c_2, c_3 and c_4 can be statistically obtained by a regression analysis over the execution times of variable-length sequence comparisons.

$$t(m, n) = c_1 + c_2 m + c_3 n + c_4 mn. \quad (2)$$

With the time prediction formula for a single GPU (Equation 2), the CUPS rate can be easily calculated using Equation 3. Note that the limit of $CUPS(m, n)$ when m and n tend to infinity is $CUPS_{max} = 1/c_4$.

$$CUPS(m, n) = \frac{mn}{t(m, n)} = \frac{mn}{c_1 + c_2 m + c_3 n + c_4 mn}. \quad (3)$$

To obtain the time prediction formula for a multi-GPU comparison, we must consider the execution time of the last GPU and the time it waits until the wavefront becomes full in the previous GPUs. So, the multi-GPU time prediction formula is given in Equation 4:

$$T_p(m, n) = t(m, \frac{n}{p}) + (p - 1) \times t(\beta, \frac{n}{p})/2 \quad (4)$$

where p is the number of GPUs, $t(m, n/p)$ is the predicted computation time of the last GPU, β is the number of rows necessary to fulfill the wavefront of each GPU and $t(\beta, n/p)/2$ is the predicted time to fulfill the wavefront in each GPU. So, $(p - 1) \times t(\beta, n/p)/2$ is the predicted time until the last GPU starts its execution. It must be noted that this formula considers a dedicated multi-GPU environment.

VI. Experimental Results

CUDAlign 3.0 was implemented in CUDA C and pthreads, with sockets, and it was executed with $B = 480$ CUDA blocks, $T = 128$ threads in each block and $\alpha = 4$ rows per thread. The size of the communication buffers was set to 8MB, sufficient to hold up to 1 million cells.

CUDAlign 3.0 was tested in the Minotauro cluster, that has a theoretical peak performance of 185.78 TFLOPS and a power efficiency of 1,266 MFLOPS/W (*www.green500.org*). Minotauro belongs to the Barcelona Supercomputing Center (BSC), Spain, and contains 128 Bull B505 hosts. Each host has two 6-core Intel Xeon E5649 and two NVidia M2090 boards (Fermi architecture). Each M2090 GPU board contains 512 CUDA cores and 6GB of GDDR5 Memory. The hosts are connected by two Infiniband QDR network (40 Gbit/s each). Also, the hosts use 10 Gigabit Ethernet links to connect to a General Parallel File System (GPFS) providing a total of 2 PB of disk storage.

Table II
Sequences used in the tests.

Chr.	Human		Chimpanzee		Score
	Accession	Size	Accession	Size	
chr1	NC_000001.10	249M	NC_006468.3	228M	84608525
chr19	NC_000019.9	59M	NC_006486.3	64M	17297608
chr20	NC_000020.10	63M	NC_006487.3	62M	40050427
chr21	NC_000021.8	48M	NC_006488.2	46M	36006054
chr22	NC_000022.10	51M	NC_006489.3	50M	31510791
Cmp.	**Sequence 1**		**Sequence 2**		**Score**
	Accession	Size	Accession	Size	
23M	NT_033779.4	23M	NT_037436.3	25M	9063
10M	NC_017186.1	10M	NC_014318.1	10M	10235188
5M	AE016879.1	5M	AE017225.1	5M	5220960

A. Sequences used in the tests

In our tests, we used real DNA sequences retrieved from the National Center for Biotechnology Information (NCBI), available at *www.ncbi.nlm.nih.gov*. We chose to use in our experiments some of the human and chimpanzee chromosomes, where homologous chromosomes are represented with the same chromosome number. We also selected smaller Megabase sequences, for the scalability analysis. The accession numbers and sizes of the chromosomes and smaller sequences are presented in Table II. For validation purposes, the optimal local scores obtained during our tests are also presented in Table II. The SW score parameters used in the tests were: match: $+1$; mismatch -3; first gap: -5; extension gap: -2.

B. Execution at the Minotauro Cluster

In Minotauro, CUDAlign 3.0 was compiled with CUDA 4.1 64-bit and the socket communication between the processes in different hosts used the Infiniband network interface. Even though Minotauro has homogeneous GPUs, the communication is heterogeneous since GPUs in the same host communicate differently than GPUs in different hosts (Figure 5). All the sequences listed in Table II (except for the chromosomes 1) were compared with 1, 2, 4, 8 and 16 GPUs. When using 1 or 2 GPUs, the execution took place in a single host. When using more than 2 GPUs, multiple hosts were used.

Table III presents execution times and GCUPS obtained in Minotauro. The execution times of the chromosome comparisons varied from 1h18m (chr21 with 16 GPUs) up to 33h20m (chr20 with 1 GPU). The relative performance obtained in GCUPS was approximately 32, 64, 128, 250

Table III
EXECUTION TIMES AND GCUPS AT THE MINOTAURO CLUSTER.

Cmp.	Size	1×M2090		2×M2090		4×M2090		8×M2090		16×M2090	
		Time	GCUPS	Time	GCUPS	Time	GCUPS	Time	GCUPS	Time	GCUPS
chr20	63M×62M	119980s	32.43	60098s	64.74	30360s	128.15	15452s	251.78	7969s	488.21
chr19	59M×64M	115933s	32.46	58189s	64.67	29318s	128.36	14954s	251.66	7744s	485.97
chr22	51M×50M	78536s	32.49	39432s	64.71	19949s	127.91	10172s	250.85	5307s	480.83
chr21	48M×46M	68665s	32.59	34681s	64.52	17541s	127.56	8963s	249.63	4688s	477.27
23M	23M×25M	17419s	32.42	8805s	64.15	4504s	125.41	2355s	239.86	1277s	442.31
10M	10M×10M	3241s	32.33	1667s	62.85	877s	119.52	485s	216.11	288s	363.90
5M	5M×5M	851s	32.11	449s	60.94	248s	110.26	149s	182.85	100s	273.54

and 480 GCUPS using, respectively, 1, 2, 4, 8 and 16 Tesla M2090 GPUs. The difference between the minimum and maximum GCUPS using the same amount of GPUs was very low for the chromosome sequences: 0.49%, 0.34%, 0.63%, 0.86% and 2.29% for, respectively, 1, 2, 4, 8 and 16 GPUs.

C. Chromosome 1 Comparison

The human chromosome 1 is the largest human chromosome, with 249 MBP (Table II) and the 35th largest sequence in the NCBI nucleotide repository. As far as we know, the human chromosome 1 has never been compared to its homologous chimpanzee chromosome 1 using an exact algorithm. CUDAlign 3.0 was used to compare both chromosomes 1 in Minotauro.

By default, CUDAlign 3.0 places the input sequences in the GPU texture memory (Section IV), but CUDA limits the maximum 1D texture width to 2^{27} [23]. So, CUDA textures were disabled for sequences larger than 134 MBP. Accessing the sequences directly from the GPU global memory reduces the performance, when compared to accesses using texture. Comparing chromosomes 21 with and without textures, we observed a performance decrease of 7%.

Table IV
HUMAN AND CHIMPANZEE CHROMOSOMES 1 COMPARISON IN MINOTAURO.

GPUs	Time	GCUPS
16 GPUs	123742s	459.92
32 GPUs	62992s	903.48
64 GPUs	32965s	1726.47

We executed the chromosomes 1×1 comparison in Minotauro with 16, 32 and 64 GPUs. The results are shown in Table IV. The execution took 1 day and 10 hours (459.92 GCUPS) with 16 GPUs, 17 hours and 30 minutes (903.48 GCUPS) with 32 GPUs and 9 hours and 9 minutes (1726.47 GCUPS) with 64 GPUs, with a close to linear reduction in the execution time. Since each Tesla M2090 has 512 cores, the 64-GPU execution used 32,768 CUDA cores. With $B = 480$ blocks and $T = 128$ threads per block, we used up to 3,932,160 cooperative CUDA threads in this execution.

D. Multi-GPU Overhead

To measure the overhead of the input/output procedures, we compared the chromosomes 21 trimmed to the following sizes: 16M×4M, 16M×2M and 16M×1M. Comparing the execution time of these trimmed comparisons with and without the short phase modification explained in Section V-A, we noticed that the overhead introduced was no more than 0.3%.

Although the network activity is overlapped with the computation, it is important to know if the network is not a bottleneck, otherwise the output buffers may increase until the manager thread starts to block. During the chromosome 21×21 comparison, the average rate of transferred cells per GPU was 10370 cells/s in Minotauro (16GPUs). Considering that each cell comprises 8 bytes (64 bits), this execution consumed 664 kbit/s in Minotauro, a negligible value compared with the Minotauro Infiniband 40 Gbit/s network.

E. Time and Speedup Predictions

To predict the execution times and speedups of the CUDAlign 3.0 comparisons when varying the size of the sequences, we split the human and chimpanzee chromosomes 21 in 5 smaller sequences with the following sizes: 1M, 3M, 5M, 7M and 9M. Then, we generated 5×5 different combinations of these sequences, resulting in 25 comparisons with DP matrices varying from 10^{12} to 10^{14} cells. The execution times of these comparisons were obtained for a single GPUs (M2090) and the results were then analyzed using the prediction method proposed in Section V-C.

We made a regression analysis in order to obtain constants c_1, c_2, c_3 and c_4 (Table V), where the maximum error of the regression comparing the input data set was less than 0.7%. Figure 7 uses points to show the execution times of the 25 comparisons in M2090 and it uses dotted lines to present the predicted times.

Table V
PERFORMANCE PREDICTION CONSTANTS.

GPU	c_1	c_2	c_3	c_4
M2090	2.99	8.727e-07	9.227e-07	3.06697e-11

We then compared the predicted times (Equation 2) and the actual execution times obtained for each comparison in Minotauro with 1 GPU (M2090). The difference between the predicted and the actual time was less than 0.45%, showing that the prediction formula was very accurate for these comparisons.

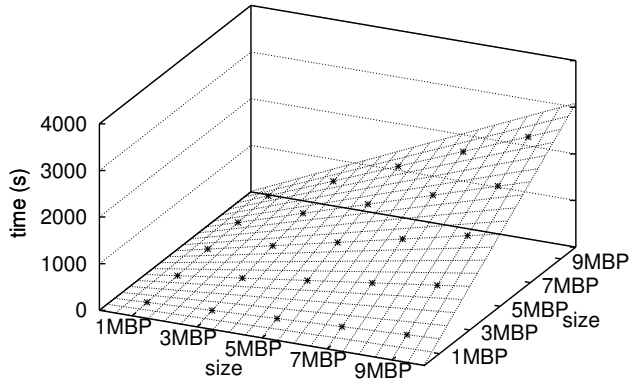

Figure 7. Predicted Execution Time.

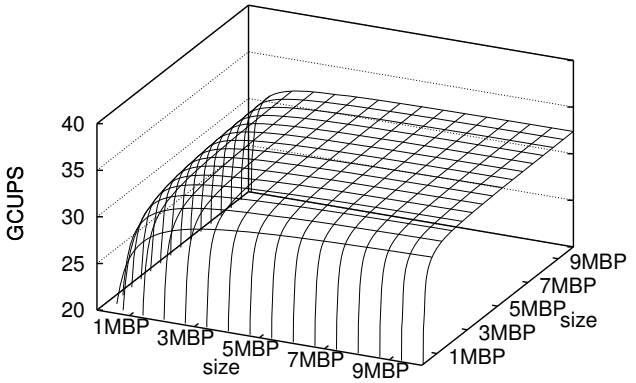

Figure 8. Predicted GCUPS.

Figure 8 presents the predicted CUPS for a range of sizes, calculated using Equation 3. The curve observed with sequences smaller than 1M indicates that the initialization and other algorithm overheads have more impact for smaller sequences.

The limit of $CUPS(m, n)$ when m and n tend to infinity is $CUPS_{max} = 1/c_4$ (Section V-C). Then, the maximum estimated GCUPS in the M2090 is 32.60. Note that the GCUPS for 1×M2090 presented in Table III are very close to the maximum predicted GCUPS.

Given the multi-GPU time prediction formula (Equation 4), Figure 9 shows the predicted speedup ($\frac{T_1}{T_p}$) in Minotauro for some comparisons using 1 to 2048 GPUs (2^9 to 2^{20} CUDA cores). Note that the speedup curves are close to linear for a larger number of GPUs when we have larger sequences. Table VI presents the predicted speedups for each comparison considering: a) 16 GPUs; b) the maximum number of GPUs with a parallel efficiency above 90%; and c) the maximum predicted speedup.

Table VI
PREDICTED SPEEDUP IN MINOTAURO.

Cmp.	16 GPUs	90% eff.		Max. Speedup	
	Speedup	GPUs	Speedup	GPUs	Speedup
chr1	15.84	138	124.28	1044	385.70
chr20	15.34	36	32.42	273	100.31
chr19	15.31	34	30.69	268	97.83
chr22	15.18	29	26.15	221	81.32
chr21	15.12	27	24.37	207	76.17
23M	14.14	13	11.80	104	37.96
10M	11.64	6	5.45	45	16.48
5M	8.10	3	2.78	23	8.46

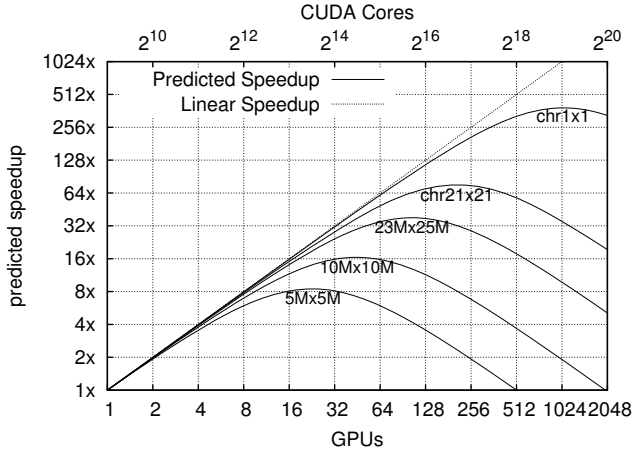

Figure 9. Predicted Speedup in Minotauro.

F. Actual Speedups

Considering the results obtained in the Minotauro cluster, Figure 10 shows the speedups of the comparisons up to 16 GPUs (8192 CUDA cores). The chromosome comparisons presented almost linear speedups. We can see that the speedup increases as we increase the sequence size. For example, with 16 GPUs, the speedups were 8.5x (5M), 11.3x (10M), 13.6x (23M), 14.7x (chr21), 14.8x (chr22), 15.0x (chr19) and 15.1x (chr20). The difference between the predicted and the actual speedups was less than 5%, as shown in Table VII.

Table VII
PREDICTED SPEEDUP ERROR IN MINOTAURO

Cmp.	Absolute Error %			
	2×GPU	4×GPU	8×GPU	16×GPU
chr20	0.09%	0.40%	1.06%	1.86%
chr19	0.10%	0.30%	1.13%	2.24%
chr22	0.08%	0.59%	1.17%	2.55%
chr21	0.66%	1.09%	1.79%	3.22%
23M	0.34%	1.10%	2.39%	3.65%
10M	1.04%	2.39%	4.36%	3.42%
5M	1.60%	3.41%	3.80%	4.78%

Figure 11 shows the relative performance (GCUPS) of all the chromosome comparisons with the exception of the chromosome 1 comparison. Below the x-axis, we can see

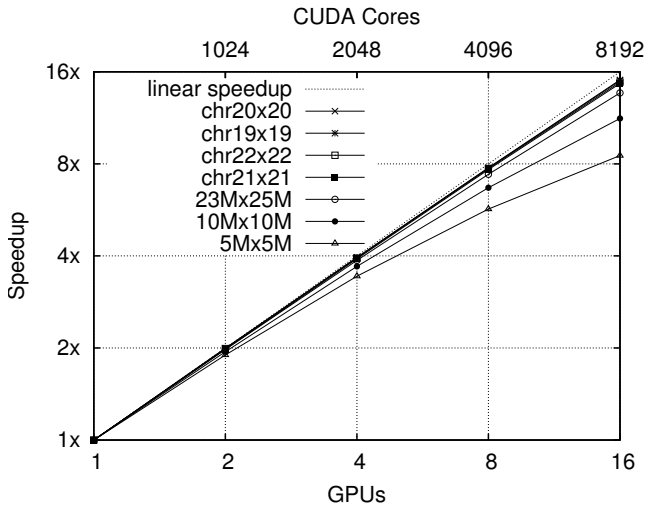

Figure 10. Actual Speedup in Minotauro.

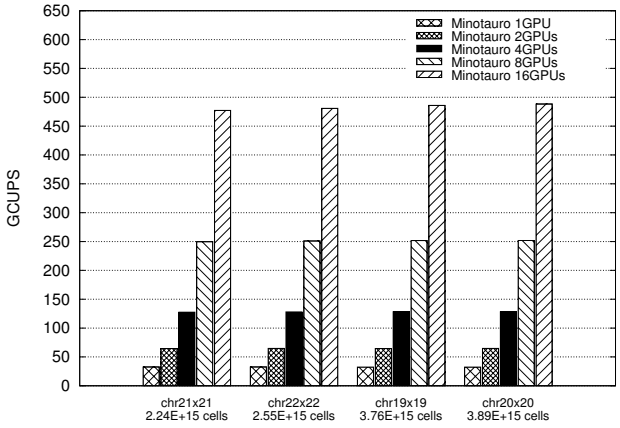

Figure 11. Throughput obtained with variable sequence sizes.

the number of cells calculated in each comparison. We can note that the GCUPS rate is almost identical considering the workload size from 2.24×10^{15} to 3.49×10^{15} cells.

Figure 12 presents, in logarithmic scale, the execution times of all the comparisons listed in Table II in Minotauro (up to 16 GPUs). The dashed diagonal lines represent the maximum GCUPS obtained for 1, 2, 4, 8 and 16 GPUs. From top to bottom, these lines correspond, respectively, to 32.59, 64.74, 128.36, 251.78 and 488.21 GCUPS. In Figure 12, we can notice that the executions maintain their high GCUPS from smaller sequences (5M) up to very huge sequences (249M). The chromosome 1 comparison had an additional overhead due to the absence of the textures (Section VI-C). Even with that, the GCUPS attained were much higher than the ones presented in the related work (Table I).

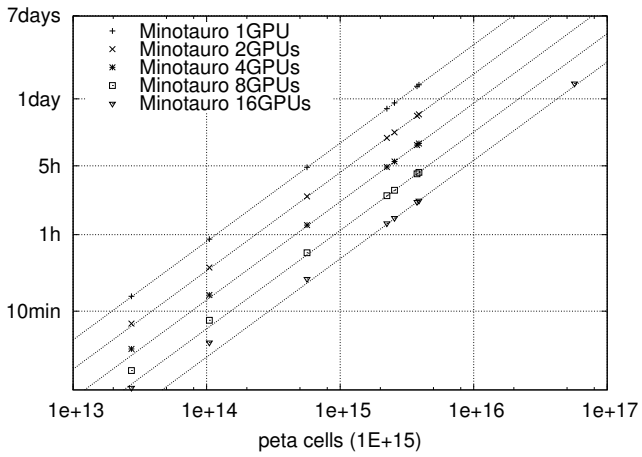

Figure 12. Execution times in log scale.

VII. Conclusions

This paper presented CUDAlign 3.0, a multi-GPU algorithm able to compare huge DNA sequences in feasible time. CUDAlign 3.0 was tested in Minotauro, a cluster with homogeneous dual-GPU hosts. The sizes of the sequences used in the tests varied from 5 MBP up to 249 MBP. In Minotauro, CUDAlign 3.0 presented results up to 1726.47 GCUPS when using 64 GPUs.

With CUDAlign 3.0, it was possible to compare the human and chimpanzee chromosomes 1. This comparison was executed in 9h09m (1726.47 GCUPS) using 64 GPUs. These results were only possible due the careful design of CUDAlign 3.0 in respect with the GPU computation and the inter-process communication. As far as we know, there is no other parallel exact algorithm that is able to compare human $\times$ chimpanzee chromosomes, except for the 21 chromosome in SW#[12] and in the single-GPU version of CUDAlign [3].

Additionally, the paper proposed a method which is able to predict the execution time and speedups with a very good accuracy. For the time prediction, the prediction error was below 0.45% (considering 1 GPU) and for the speedup prediction, it was below 5%.

This paper did not intend to give detailed biological analysis of the chromosomes, neither to align all of the human-chimpanzee homologous chromosomes. Nevertheless, it showed that CUDAlign 3.0 is able to compare even the largest human chromosomes, and that can be very useful for biologists in comparative genomics. With multiple GPUs and the highly scalable CUDAlign 3.0, the use of exact methods is now feasible for huge sequence comparison.

As future work, we intend to retrieve the alignments of all the homologous human $\times$ chimpanzee chromosomes in multiple GPUs. This will involve some modifications in CUDAlign 3.0 in order to store intermediate results in a parallel way. Although the results in this paper showed

optimal local scores, other types of comparisons are also desirable. So, future work will also include the retrieval of relevant alignments besides the local optimal one.

The static column distribution is suited for environments with dedicated computing power. In non-dedicated environments, where the nodes may vary their processing rate, dynamic load balancing is preferred over the static column distribution. Future works will develop methods to measure and adjust the column distribution to variable scenarios.

VIII. Acknowledgements

We thankfully acknowledge the support of the grant SEV-2011-00067 of Severo Ochoa Program, awarded by the Spanish Government, the Spanish Ministry of Science and Technology (TIN2007-60625, TIN2012-34557, CSD2007-00050) and the Generalitat de Catalunya (2009-SGR-980). This work is also partially supported by CNPq/Brazil (grants 242800/2012-2 and 211456/2013-6) and FAP/DF. Finally, the authors would like to thank the BSC Minotauro team for the help setting the environment and customizing the scripts for multi-GPU executions.

References

[1] D. W. Mount. *Bioinformatics: sequence and genome analysis*. Cold Spring Harbor Laboratory Press, 2004.

[2] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–197, March 1981.

[3] Edans Sandes and Alba. de Melo. Retrieving smith-waterman alignments with optimizations for megabase biological sequences using gpu. *IEEE Transactions on Parallel and Distributed Systems*, 24(5):1009–1021, 2013.

[4] Y. Liu, W. Huang, J. Johnson, and S. Vaidya. GPU Accelerated Smith-Waterman. In *ICCS 2006*, volume 3994 of *LNCS*, pages 188–195. Springer, 2006.

[5] W. Liu, B. Schmidt, G. Voss, A. Schroder, and W. Muller-Wittig. Bio-sequence database scanning on a GPU. *IPDPS*, 2006.

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

[7] Y. Liu, B. Schmidt, and D. Maskell. 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, 2010.

[8] Svetlin Manavski and Giorgio Valle. CUDA compatible GPU cards as efficient hardware accelerators for smith-waterman sequence alignment. *BMC Bioinformatics*, 9(S-2), 2008.

[9] Ali Khajeh-Saeed, Stephen Poole, and J. Blair Perot. Acceleration of the smith-waterman algorithm using single and multiple graphics processors. *J. Comput. Physics*, 229(11):4247–4258, 2010.

[10] Jacek Blazewicz, Wojciech Frohmberg, Michal Kierzynka, Erwin Pesch, and Pawel Wojciechowski. Protein alignment algorithms with an efficient backtracking routine on multiple gpus. *BMC Bioinformatics*, 12(1):181, 2011.

[11] Fumihiko Ino, Yuma Munekawa, and Kenichi Hagihara. Sequence homology search using fine grained cycle sharing of idle GPUs. *IEEE Trans. Parallel Distrib. Syst*, 23(4):751–759, 2012.

[12] M. Korpar and M. Sikic. Sw#-gpu-enabled exact alignments on genome scale. *Bioinformatics*, 2013.

[13] E. W. Myers and W. Miller. *Computer Applications in the Biosciences*, 4(1):11–17, 1988.

[14] Chunxi Chen and Bertil Schmidt. An adaptive grid implementation of DNA sequence alignment. *Future Generation Comp. Syst*, 21(7):988–1003, 2005.

[15] O. Gotoh. An improved algorithm for matching biological sequences. *J Mol Biol*, 162(3):705–708, December 1982.

[16] Gregory F. Pfister. *In search of clusters: the coming battle in lowly parallel computing*. Prentice-Hall, Inc., 1995.

[17] Edans Sandes and Alba. de Melo. CUDAlign: using GPU to accelerate the comparison of megabase genomic sequences. In *PPOPP*, pages 137–146. ACM, 2010.

[18] Ashwin M. Aji and Wu chun Feng. Optimizing performance, cost, and sensitivity in pairwise sequence search on a cluster of playstations. In *BIBE*, pages 1–6. IEEE, 2008.

[19] Stjepan Rajko and Srinivas Aluru. Space and time optimal parallel sequence alignments. *IEEETPDS: IEEE Transactions on Parallel and Distributed Systems*, 15, 2004.

[20] Khaled Hamidouche, Fernando Machado Mendonca, Joel Falcou, Alba Cristina Magalhaes Alves de Melo, and Daniel Etiemble. Parallel smith-waterman comparison on multicore and manycore computing platforms with BSP++. *International Journal of Parallel Programming*, 41(1):111–136, 2013.

[21] Y. Liu, A. Wirawan, and B. Schmidt. CUDASW++ 3.0: Accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions. *BMC Bioinformatics*, 14:117, 2013.

[22] Edans Sandes and Alba. de Melo. Smith-Waterman Alignment of Huge Sequences with GPU in Linear Space. In *IPDPS*, pages 1199–1211, 2011.

[23] NVIDIA. CUDA Programming Guide 5.0, 2012.