Hardware Acceleration of Sequence Alignment Algorithms—An Overview

Laiq Hasan Zaid Al-Ars Stamatis Vassiliadis
Delft University of Technology
Computer Engineering Laboratory
Mekelweg 4, 2628 CD Delft, The Netherlands
L.Hasan@ewi.tudelft.nl

Abstract: *Sequence alignment is one of the most important activities in bioinformatics. With the ever increasing volume of data in bioinformatics databases, the time for comparing a query sequence with the available databases is always increasing. Many algorithms have been proposed to perform and accelerate sequence alignment activities. This paper introduces a taxonomy of the various sequence alignment algorithms found in the literature, with particular emphasis on the Smith-Waterman (S-W) algorithm. The paper also provides a classification of the available hardware acceleration methods used to speed up the S-W algorithm.*

Keywords: *Bioinformatics, Sequence Alignment, Dynamic Programming, Smith-Waterman Algorithm, Hardware Acceleration.*

1 Introduction

Biology is in the middle of a major paradigm shift driven by computing technology. Two decades before the formal inauguration of the *Human Genome Project (HGP)*, a new hybrid field (partly molecular biology and partly computer science) began to emerge. The new field was called *computational molecular biology* or *bioinformatics*, which may be defined as a discipline that generates computational tools, databases, and methods to support genomic and post genomic research [1].

The bioinformatics community is doing research in many subfields, such as gene structure prediction, phylogenetic trees, protein docking (2D, 3D) etc, but the most promising one is sequence similarity analysis or *sequence alignment*. In most common terms sequence alignment may be defined as an arrangement of two or more DNA or Protein sequences to highlight the regions of their similarity. This in turn indicates the genetic relatedness between the organisms.

This paper aims at providing an overview of various global and local sequence alignment methods in general, with particular emphasis on the *Smith-Waterman (S-W)* algorithm and it's hardware acceleration. S-W is the most sensitive, but computationally the most expensive sequence alignment algorithm. Thus it provides exact matches between sequences, at the cost of long processing run time and is thus considered as the top candidate for hardware acceleration.

The paper is organized as follows: Section 2 explains various methods and algorithms for sequence alignment, their complexities and comparisons. Section 3 reviews the work related to hardware acceleration of the S-W algorithm and Section 4 summarizes the paper in the form of a brief conclusion.

2 Sequence Alignment Methods

Traditionally, the methods of pairwise sequence alignment are classified as either global or local. Global methods attempt to match as many characters as possible, from end to end, whereas local methods aim to identify short stretches of similarity between two sequences [2]. Figure 1 gives various global and local sequence alignment methods. In the following subsections we are going to describe these methods briefly.

2.1 Global Methods

The most basic method of comparing two sequences is a visual approach known as a dotplot [3]. The sequences to be compared are arranged along the margins of a matrix. At every point in the matrix where the two sequences are identical a dot is placed (i.e. at the intersection of every row and column that have the same letter in both sequences). A diagonal stretch of dots will indicate regions where the two sequences are similar. Done in this fashion a dot plot as shown in Table 1 will be obtained (for clarity, dots are marked in the table as xs). The time and space complexity of the dotplot is $O(MN)$, where M and N are the lengths of the two sequences to be compared.

Table 1: The dotplot matrix.

	a	c	t	g	g
a	x				
c		x			
t			x		
g				x	x
g				x	x

In 1970, *Needleman and Wunsch (N-W)* proposed an alignment method [4] that can be obtained computationally by applying a straightforward *Dynamic Programming (DP)*

algorithm to find the optimal global alignment of two sequences A and B. The algorithm is based on finding the elements of a matrix H according to:

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

where $S_{i,j}$ is the similarity score of comparing A_i to B_j and d is the penalty for a mismatch. The matrix is initialized with $H_{0,0} = 0$.

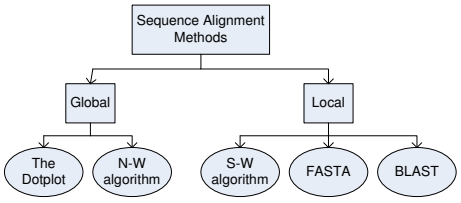

Figure 1: Various methods for sequence alignment.

The time complexity of initialization step is simply $O(M + N)$. The next step is filling in the matrix with all the scores, $H_{i,j}$. For each cell of the matrix, three neighboring cells (left, above, and diagonally upper-left) must be compared. Three separate scores are calculated based on all three neighbors, and the maximum score is assigned to the cell, which is a constant time operation [5]. Thus, to fill the entire matrix, the time complexity is the number of entries, or $O(MN)$. Finally during the traceback we can move a maximum of N rows and M columns, and thus the complexity of this is $O(M + N)$. Thus, the overall time complexity of this algorithm is $O(M + N) + O(MN) + O(M + N) = O(MN)$. Since this algorithm fills a single matrix of size MN, the total space complexity of this algorithm is $O(MN)$.

2.2 Local Methods

In 1981, Smith and Waterman described a method, commonly known as the Smith-Waterman (S-W) algorithm [6], for finding common regions of local similarity. When obtaining the local S-W alignment, $H_{i,j}$ for N-W algorithm is modified as follows:

$$H_{i,j} = \max \begin{cases} 0 \\ H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - d \\ H_{i,j-1} - d \end{cases} \quad (2)$$

The initialization and matrix fill steps for N-W and S-W algorithms have the same time complexity. The difference lies in the traceback step. With N-W, the traceback starts at the last cell in the matrix and traces the maximal score path back to the first cell. With S-W, traceback starts at the cell with the highest score in the matrix. For doing this, the algorithm requires to find the maximum cell, and this must be done by traversing the entire matrix, making the time complexity for the traceback $O(MN)$.Thus the total time complexity of the S-W algorithm is $O(M + N) + O(MN) + O(MN) = O(MN)$. The space complexity is also unchanged from the N-W algorithm. This is due to the fact that the same matrix is used and the same amount of space is needed for the traceback.

As an example, the S-W algorithm, is used to compute the optimal local alignment of two sequences (i.e., A = a g g t a c and B = c a g c g t t g). Assume that

$$S_{i,j} = \begin{cases} +2 \ \ if(A_i = B_j) \\ -1 \ \ else \end{cases}$$

and $d = 2$.

Table 2 illustrates the calculation of the dynamic programming matrix H and the path of tracing back (shown in bold digits). The best score found in the matrix is 6, and the corresponding optimal local alignment is

A: a g - g t
B: a g c g t

Table 2: The DP matrix and the tracing back path.

		c	a	g	c	g	t	t	g
	0	**0**	0	0	0	0	0	0	0
a	0	0	**2**	0	0	0	0	0	0
g	0	0	0	**4**	**2**	2	0	0	2
g	0	0	0	2	3	**4**	2	0	2
t	0	0	0	0	1	2	**6**	4	2
a	0	0	2	0	0	0	4	5	3
c	0	2	0	1	2	0	2	3	4

Apart from the S-W algorithm, there are other local search methods such as FASTA (Fast Alignment Search Tools - All) and BLAST (Basic Local Alignment Search Tool). Based on heuristics, they are faster, although much less sensitive than the the S-W algorithm.

FASTA was developed in 1985 by Lipman and Pearson [7]. Unlike the N-W and S-W algorithms, FASTA approximates the optimal alignment by searching and matching k-tuples (i.e. subsequences of length k). The algorithm assumes that related proteins will have regions of identity, and by searching with k-tuples, the FASTA algorithm allows small regions of local identity to be found quickly. For proteins, these k-tuples tend to be of length two.

BLAST [8] is similar to the FASTA algorithm, however, it uses words (w) instead of k-tuples. The computational complexity of both FASTA and BLAST comes out to be $O(MN)$. The space complexity for FASTA is $O(MN)$, whereas for BLAST, it is slightly higher than all other algorithms and it comes out to be $O(20^w + MN)$ (where w is the word size). Table 3 gives a comparison of various sequence alignment algorithms.

Table 3: Comparison of various sequence alignment algorithms.

Algorithm	Local/Global	Search Method	Time Complexity	Space Complexity
Dotplot	Global	Basic	$O(MN)$	$O(MN)$
Needleman-Wunsch	Global	Dynamic Programming	$O(MN)$	$O(MN)$
Smith-Waterman	Local	Dynamic Programming	$O(MN)$	$O(MN)$
FASTA	Local	Heuristic	$O(MN)$	$O(MN)$
BLAST	Local	Heuristic	$O(MN)$	$O(20^w + MN)$

Since S-W algorithm is considered as the top candidate for hardware acceleration, people are working on accelerating it by using various hardware implementation techniques. Section 3, gives a brief review of this work.

3 Review of the hardware acceleration of the S-W algorithm

In computing, *hardware acceleration* is the use of specialized hardware to perform some function faster than is possible in software running on the general purpose CPU. The hardware that performs the acceleration, when in a separate unit from the CPU, is referred to as a hardware accelerator.

When run on a PC, S-W algorithm spends as much as 98.6 % of its time on calculating element of the $H_{i,j}$ matrix [11]. Theoretically, an accelerated system could calculate many elements of the $H_{i,j}$ matrix in parallel. Following are a few advantages of the S-W algorithm.

1. With so much time spent on repeating the same calculation, it is very well suited for parallelization.
2. It offers a high degree of instruction efficiency, i.e. the sequences of characters can be represented by as little as two to five bits.
3. All of the computations can be implemented using 24-bit integers.

A conventional 64-bit processor would devote all its processing power to these simple equations, so we have to look for alternative implementation options [11]. Work has been done for either accelerating the whole algorithm or the computationally expensive parts of it by implementing it on various available platforms. Figure 2 gives a classification of this work based on the methods of implementation. In the following subsections, we are going to discuss the hardware acceleration of S-W algorithm according to the classification given in Figure 2.

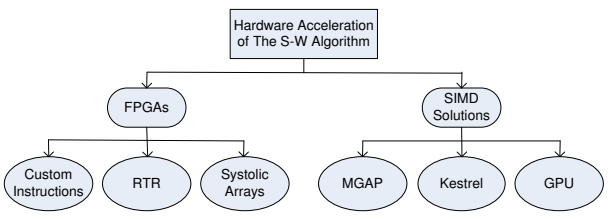

Figure 2: Hardware Acceleration of the S-W algorithm.

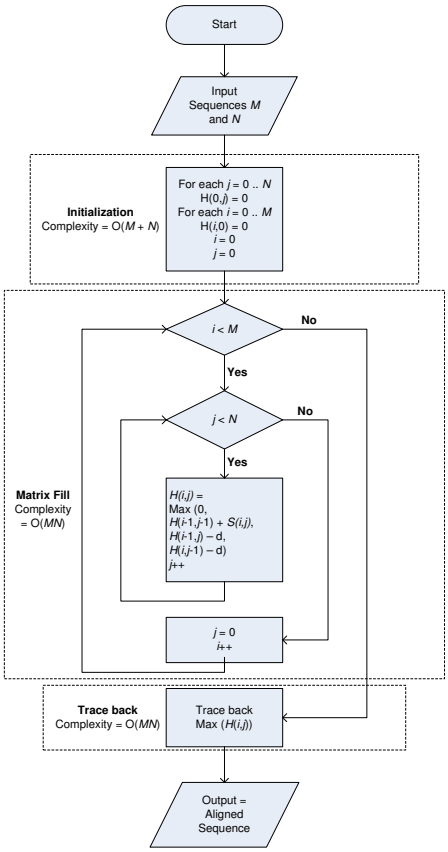

Figure 3: Smith-Waterman Algorithm's Flow Chart.

3.1 Theoretical basis

Figure 3 explains the three basic steps in the S-W algorithm (i.e. Initialization, Matrix Fill and Trace back) in the form of a flow chart.

Parallelization of the matrix fill stage in order to reduce the $O(MN)$ complexity is complicated by data dependencies whereby each cell $C_{j,i}$ depends on the values of three neighboring cells $C_{j,i-1}$, $C_{j-1,i}$, and $C_{j-1,i-1}$, with each of those cells in turn depending on the values of three neighboring cells, which effectively means that this dependency extends to every other cell in the region $C \leq j, \leq i$. This implies that it is possible to simultaneously compute $C_{1,3}$, $C_{2,2}$ and $C_{3,1}$, since these cells fall outside each others data dependency regions. In simple terms, assuming a square matrix (i.e. NxN, with both query sequences being of length N) and only considering the upper left half of the matrix, the cells that may be computed in parallel are given

by $C_{x-y+1,y}$ for $1 \leq y \leq x$ and $x \leq N$, which gives that x is the largest row value of the current set of computable cells. From this it may be concluded that the maximum number of cells that may be computed in parallel is N. This model is mirrored for the lower right half of the matrix. Expanding this model for non-square matrices introduces some non-linearities, but the parallelization implications remain the same. Given that it takes $x = N$ cycles to compute the upper left half of the matrix, the lower right half would consume $N - 1$ cycles, since it would be redundant to recompute the cells where $x = N$. Therefore, a parallelized S-W algorithm would require $2N - 1$ operations, or generalizing for non-equal query sequences of length M and N, parallel S-W algorithm requires $M + N - 1$ operations, i.e. an $O(N)$ implementation is possible. Table 4 gives a sample similarity matrix for two sequences A = gattaca and B = gactc. The cells in the anti diagonals are computed in parallel. Since there are 11 anti diagonals (ignoring the 1st row and 1st column, which are meant for initialization only), so a total of 11 cycles are required for this computation, and a maximum of 5 cells may be computed in parallel.

Table 4: A sample similarity matrix.

		g	a	t	t	a	c	a
	0	0	0	0	0	0	0	0
g	0	1	0	0	0	0	0	0
a	0	0	2	0	0	1	0	1
c	0	0	0	1	0	0	0	0
t	0	0	0	1	2	0	0	0
c	0	0	0	0	0	1	1	0

3.2 FPGAs

Field Programmable Gate Arrays (FPGAs) are reconfigurable data processing devices on which an algorithm is directly mapped to basic processing logic elements, e.g. NAND gates. To take advantage of using a FPGA, one has to implement massively-parallel algorithms on this reconfigurable device. Thus they are well suited for certain classes of bioinformatics algorithms, such as sequence alignment algorithms like the S-W.

3.2.1 FPGA Custom Instructions

In [9], the authors studied the improvement of computational processing time of the S-W algorithm using custom instructions on an FPGA board. This was done by first writing the S-W algorithm in pure software and then replacing the portion which was the most computationally intensive with an FPGA custom instruction. Finally, they compared the processing runtime between the pure software and the hardware acceleration versions to calculate the percentage of runtime improvement. The results showed that the hardware accelerated algorithm improved the processing runtime by an average of 287%. Thus using FPGA Custom Instructions is a promising direction for further research in improving genomic sequence searching.

3.2.2 Run-Time Reconfiguration

One way to further exploit the reconfigurable resources of FPGAs and increase their functional density is to reconfigure them during system operation. This process is referred to as Run-Time Reconfiguration (RTR). RTR is an approach to system implementation that divides an application or algorithm into time-exclusive operations that are implemented as separate configurations. In [10], an approach to realize high speed sequence alignment using run-time reconfiguration is proposed. With this approach, it is demonstrated that high performance can be achieved using off-the-shelf FPGA boards. The performance is almost comparable with dedicated hardware systems. The time for comparing a query sequence of 2048 elements with a database sequence of 64 million elements by the S-W algorithm is about 34 sec, which is about 330 times faster than a desktop computer with a 1GHz Pentium-III.

In [11], the performance of the S-W algorithm has been increased substantially by using run time reconfiguration. The percentage of time spent on calculating the elements of $H_{i,j}$ matrix was cut by nearly a third and the absolute time spent on the algorithm was dropped from 6,461 seconds to a little over 100 seconds, approximately 64 times faster than the equivalent software-only implementation.

3.2.3 Systolic Arrays

An arrangement of processors in an array (often rectangular) where data flows synchronously across the array between neighbors, usually with different data flowing in different directions. Each processor at each step takes in data from one or more neighbors (e.g. North and West), processes it and, in the next step, outputs results in the opposite direction (South and East). An example of a systolic array might be matrix multiplication. One matrix is fed in a row at a time from the top of the array and is passed down the array, the other matrix is fed in a column at a time from the left hand side of the array and passes from left to right. Dummy values are then passed in until each processor has seen one whole row and one whole column. At this point, the result of the multiplication is stored in the array and can now output a row or a column at a time, flowing down or across the array. Figure 4 is a simple example of a systolic array. In this configuration there are two vector array inputs, M and N. The processing cells have a value, U_{ij} , that is usually a result due to a defined algorithm within the cells.

In [12] the authors have proposed to feed the systolic array with multiple data at a time. That implies writing several nucleotides in a single bus write- cycle. The results from Table 5 show that their Virtex2 implementation is the fastest in terms of CUPS (Cell Updates Per Second).

In [13] a concept to accelerate the S-W algorithm on the bases linear systolic array is demonstrated. The reason for choosing this architecture is outlined by demonstrating the

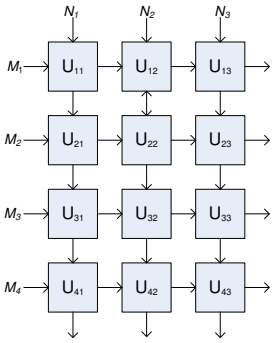

Figure 4: Systolic Array.

efficiency and simplicity in combination with the algorithm. Nevertheless, there are two key methodologies to speedup this massively parallel system. By turning the processing from bit-parallel to bit-serial the actual improvement is enabled. This change is performance neutral but in combination with the drafted early maximum detection a considerable speedup is possible. Another effect of this improvement is a data dependant execution time of the processing elements. Here the second acceleration prevents idle times to exploit the hardware and speeds up the computation. This can be accomplished by a globally asynchronous timing representing a self-timed linear systolic array. In [13] the authors have provided no performance estimation due to the initial stage of their work, that's why it can't be compared with other related work.

In [14] an improved systolic processing element cell for implementing the S-W algorithm on a Xilinx Virtex FPGA is presented. The implementation was successfully verified using on Pilchard (a reconfigurable computing platform), which provides a 133 MHz, 64-bit wide memory mapped bus to the FPGA.

Table 5 gives a performance comparison between [12] and [14], where the performance is measured in billion cell updates per second (BCUPS). It is obvious from Table 5, that the Virtex 2 implementation in [12] outperforms other implementations in terms of BCUPS. But it must be noted that maximum clock speed was assumed and that this is impractical for the majority of communication buses. The actual advantages of the proposal in [12] may arise when the systems compared work under the same conditions: same clock frequency and same sequence length.

3.3 SIMD Solutions

SIMD is an acronym for *Single-Instruction stream, Multiple-Data stream*. It is a type of multiprocessor architecture in which multiple sets of operands may be fetched to multiple processing units and may be operated upon simultaneously within a single instruction cycle. Several approaches based on the concept of SIMD will be discussed in the following subsections.

Table 5: Performance Comparison between [12] and [14].

System	Processing Elements per chip	Performance (BCUPS)	CLK (MHz)
[12]XC2V6000-5	4000	3200	200
[12]XCV1000E-6	1600	908	142
[14]XCV1000-6	4032	742	184
[14]XCV1000E-6	4032	814	202

3.3.1 Micro Grained Array Processor (MGAP)

In [15] an implementation of the S-W Algorithm is described on a general purpose fine-grained architecture, the MGAP. The authors of [15] show that their implementation is about 5 times faster than the rapid implementation of a genetic sequence comparator using field programmable logic arrays [16]. Showing thereby, that massively parallel processor arrays, like the MGAP, possess the capability to solve computationally intensive problems in Molecular Biology efficiently and inexpensively. The algorithm given in [15] takes $M + N$ steps to align two sequences. Therefore if there are K sequences to be aligned, the entire computation would require only $M + N + K$ steps. The sequential algorithm would have taken $O(MNK)$ steps to compute K alignments.

3.3.2 Kestrel Parallel Processor

The Kestrel parallel processor is a single-board coprocessor with a 512-element linear array of 8-bit, SIMD processing elements [17]. The system was designed and built at the University of California at Santa Cruz, where work on the Human Genome Project and other bioinformatics applications motivated development of a sequence analysis engine that could efficiently analyze databases containing billions of characters from DNA, RNA, or proteins. As a case study, the authors of [17], implemented the S-W algorithm on the kestral parallel processor for different query sizes. The results of their implementations are shown in Table 6

3.3.3 Graphics Processing Units

Graphics Processing Units (GPUs) are single-chip processors, used primarily for computing 3D functions. This includes things such as lighting effects, object transformations, and 3D motion. The GPU is a good match for bioinformatics applications (e.g. S-W algorithm for sequence alignment), as it is an inexpensive, high-performance SIMD architecture.

In [18], it has been demonstrated that the streaming architecture of GPUs can be efficiently used for biological sequence database scanning. To derive an efficient mapping onto this type of architecture, the authors have reformulated the S-W algorithm in terms of computer graphics primitives and claim that the evaluation of their implementation on a high-end graphics card shows a speedup of almost sixteen compared to a Pentium IV 3.0GHz. They also claim that

Table 6: Comparison of the work reviewed in Section 3.

Reference	Cited in Section	Implemented on	Compared with	Speed up	Query size
[9]	3.2.1	FPGA	Software only	287x	—
[10]	3.2.2	FPGA	1 GHz P-III	330x	—
[11]	3.2.2	FPGA	Software only	64x	—
[15]	3.3.1	MGAP	SPLASH	5x	—
[17]	3.3.2	Kestrel	500MHz Ultra SPARC-II	17x	100
[17]	3.3.2	Kestrel	500MHz Ultra SPARC-II	49x	250
[17]	3.3.2	Kestrel	500MHz Ultra SPARC-II	99x	500
[18]	3.3.3	GPU	P-IV 3.0 GHz	16x	—

this is the first reported implementation of the S-W algorithm on graphics hardware.

Table 6 gives a comparison of the work reviewed in Section 3. It is evident from Table 6, that no standard comparison approach is adapted. That is why, we can only look into each implementation on individual basis to see how much improvement is achieved in comparison with the reference provided for each implementation.

4 Conclusion

In this paper, we provided a taxonomy of various sequence alignment algorithms found in the literature. The S-W algorithm proved to be the most accurate in performing sequence alignment, however it requires an exceptionally long time to complete (for long sequences), making it the most appropriate alternative for hardware acceleration. The paper also presented a classification of different hardware acceleration methods for the S-W algorithm, and gave a comparison between their respective speedups relative to some baseline performance. The paper shows that there is a need to identify a common measure to compare different acceleration methods in the literature.

References

[1] http://www.roseindia.net/bioinformatics/.

[2] http://www.dbmi.columbia.edu/bioinformatics/.

[3] T K Attwood and D J Parry-Smith. *Introduction to bioinformatics Cell and molecular biology in action series*.

[4] Saul Needleman and Christian Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–53.

[5] Gregory Egan Tirath Ramdas. A survey of fpga-based high performance computation in molecular biology and other domains. Technical report, MECSE, 2005.

[6] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, vol. 147:pp. 195–7, 1981.

[7] D.J. Lipman and W.R. Pearson. Rapid and sensitive protein simlarity searches. *Science*, 227:1435–1441, 1985.

[8] Gish W. Miller W. Myers E. W. Altschul, S. F. and D. J. Lipman. A basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[9] J Shaw S Seto J Chiang, M Studniberg and K Truong. Hardware accelerator for genomic sequence alignment. Proceedings of the 28th IEEE EMBS Annual International Conference New York City, USA, Aug 30-Sept 3, 2006.

[10] T. Maruyama Y. Yamaguchi, Y. Miyajima and A. Konagaya. High speed homology search using run-time reconfiguration. FPL 2002.

[11] Steve Margerm Cray Inc. Reconfigurable computing in real-world applications. *FPGA and Structured ASIC Journal (www.fpgajournal.com)*, February 7, 2006.

[12] C. Pedreira S.Bojanic, G.Caffarena and O. Nieto-Taladriz. High speed circuits for genetics applications. PROC. 24th International Conference on Microelectronics (MIEL 2004).

[13] H. Kreft G. Pfeiffer and M. Schimmler. Hardware enhanced biosequence alignment. International Conference on METMBS05.

[14] K.H. Lee C.W. Yu, K.H. Kwong and P.H.W. Leong. A smith-waterman systolic cell. FPL 2003.

[15] M. Borah R. S. Bajwa S. Hannenhalli M. J. Irwin. A SIMD Solution to the Sequence Comparison Problem on the MGAP. Proc. Int. Conf. on Application Specific Array Processors, 1994.

[16] Daniel P. Lopresti. Rapid implementation of a genetic sequence comparator using field programmable logic arrays. conference on Advanced research in VLSI pages 138-152, 1991.

[17] A. Di Blas et. al. The UCSC Kestrel Parallel Processor. *IEEE Transactions on Parallel and Distributed Systems*, 16(1):80–92, 2005.

[18] A Schroder et. al. Bio-Sequence Database Scanning on a GPU. HICOMB, 2006.