Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices

Stefan Dydel and Piotr Bała

Faculty of Mathematics and Computer Science,
N. Copernicus University, Chopina 12/8, 87-100 Torun, Poland,
{stef, bala}@mat.uni.torun.pl

Abstract. In this paper we show how to significantly accelerate Smith-Waterman protein sequence alignment algorithm using reprogrammable logic devices – FPGAs (Field Programmable Gate Array). Due to perfect sensitivity, the Smith-Waterman algorithm is important in a field of computational biology but computational complexity makes it impractical for large database searches when running on general purpose computers.
Current approach allows for aminoacid sequence alignment with full substitution matrix which leads to more complex formula than used in DNA alignment and is much more memory demanding. We propose different parellization scheme than commonly used systolic arrays, leading to full utilization of PUs (Processing Units), regardless of sequence length. FPGA based implementation of Smith-Waterman algorithm can accelerate sequence alignment on a Pentium desktop computer by two orders of magnitude comparing to standard OSEARCH program from FASTA package.

1 Introduction

Sequence comparison is one of the most important bioinformatic problems. Number of published protein sequences increases rapidly and current databases contain gigabytes of the data. Processing of such amount of data, especially sequence comparisons necessary for the scientific discovery, requires, in addition to the efficient data storage, significant computational resources. Moreover, users would like to achieve results fast, in most cases interactively, which cannot be achieved using single, general purpose computer.

For the sequence comparison, the most widely used is Smith-Waterman algorithm [17]. The computational complexity of the algorithm is quadratic which additionally makes it difficult to use for the processing of large data sets. This makes ground for investigation of new algorithms and new acceleration techniques which can make processing of actual and future data feasible.

We propose efficient implementation of the Smith-Waterman algorithm in reconfigurable hardware. Recent Field Programmable Gate Array (FPGA) chips provide relatively low cost and powerful way of implementing highly parallel algorithms. Utilization of FPGAs in the biological sequence alignment has a

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 23–32, 2004.
© Springer-Verlag Berlin Heidelberg 2004

long history [12], but most of the implementations cover special case of Smith-Waterman algorithm: DNA alignment with simple edit distance, which is equivalent to computing LLCS (length of longest common subsequence) [16]. Efficient bitvector implementations of LLCS dedicated to general purpose computers have been developed [16], potenitially leading to speeding up by a factor of 64 when implemented in modern processors. We investigate the possibility of implementing more general case of the Smith-Waterman algorithm allowing alignment of protein sequences (20 letters alphabet) with full substitution cost matrix (400 entries, 8 bits each) and 18 bits of alignment score using Xilinx Virtex 2 pro FPGAs. Currently Xilinx releases suitable FPGAs: XC2VP50 and XC2VP70.

Number of comparisons of protein residues per second using rigorous Smith-Waterman algorithm exceeds 10^{10} for a single XC2VP70 device and can scale up to 10^{12} in a case of a cluster of PCs each equipped with FPGA accelerator.

2 Problem Description

For the sequence comparison we assume widely used edit distance model. Distance between strings is measured in terms of edit operations such as deletions, insertions, and replacements of single characters within strings. Comparison of two strings is accomplished by using edit operations to convert one string into the other, minimizing the sum of the cost of operations used. This total cost is regarded as a degree of similarity between strings.
Let us define formally our goal, which is *local optimal alignment*.

Definition 1. *Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite set of characters - alphabet. Let ϵ denote the empty string. Let us define A^* as a set of all strings over alphabet A. Notation for a string $w \in A^*$ having $w_i \in A$ as the i-th character is $w = w_1 w_2 \ldots w_n,\ i = 1 \ldots n$.*

Example 1. We are particularly interested in the following alphabets important in computational biology:

1. $\{C, T, G, A\}$ - nucleotides: Cytosin, Thymin, Guanin, Adenin
2. $\{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ - 20 aminoacids.

Definition 2. *An edit operation is an ordered pair (γ, ξ), where γ, ξ are strings having length one or equal to ϵ. Let E_A denote a set of all edit operations over alphabet A.*

Edit operations can be viewed as operations describing rewriting rules: (γ, ϵ) denotes deletion of character γ, (ϵ, ξ) denotes insertion of character ξ and (γ, ξ) denotes substitution of character γ by ξ.

Definition 3. *An alignment of strings $s, t \in A^*$ is a sequence of edit operations $((\gamma_1, \xi_1), \ldots, (\gamma_n, \xi_n))$ such that $s = \gamma_1 \ldots \gamma_n, t = \xi_1 \ldots \xi_n$. Let $Align(s, t)$ denote a set of all possible alignments of strings s, t.*

In order to work with optimal alignments we have to introduce optimization criterion.

Definition 4. *An edit score function* $\rho_{edit} : E_A \to \Re$ *assigns a real value to each edit operation.*

In computational biology function ρ_{edit} is known as a scoring matrix for proteins such as BLOSUM62 used by BLAST program [25].

Definition 5. *An alignment score function* $\rho_{align} : Align(s,t) \to \Re$ *is defined as* $\rho_{align}(((\gamma_1, \xi_1), \ldots, (\gamma_n, \xi_n))) = \sum_{i=1}^{n} \rho_{edit}((\gamma_i, \xi_i))$,
where $((\gamma_1, \xi_1), \ldots, (\gamma_n, \xi_n)) \in Align(s,t)$, *and* $s, t \in A^*$. *We assign a value equal to sum of all edit operations constituting particular alignment.*

Finally, one can define optimal local alignment, and local alignment score describing optimization goal.

Definition 6. *An Optimal alignment score opt is defined as follows:*
$opt(s,t) = \max\left(\rho_{align}(\alpha); \alpha \in Align(\hat{s}, \hat{t}),\ \hat{s} \subset s, \hat{t} \subset t\right)$. *The alignment* α *having maximum score in this sense is called an optimal local alignment.*

Obvious brute-force approach relays on generating all possible alignments of all substrings of s, t and evaluating its score. This leads to unreasonably high complexity. Using Smith-Waterman algorithm it is possible to reduce complexity to quadratic one.

3 Smith-Waterman Local Alignment Algorithm

This algorithm computes both optimal local alignment and its score. Let $s = s_1 \ldots s_m,\ t = t_1 \ldots t_n \in A^*$ be fixed input strings. Let us define matrix M:

$$M[i][j] = \begin{cases} 0, & i = 0 \text{ or } j = 0 \\ max \begin{cases} 0 \\ M[i-1][j] + \rho_{edit}(s_i, \epsilon) \\ M[i][j-1] + \rho_{edit}(\epsilon, t_j) \\ M[i][j] + \rho_{edit}(s_i, t_j) \end{cases} & i = 1 \ldots m,\ j = 1 \ldots n \end{cases} \tag{1}$$

Optimal local alignment score is equal to $\max\left(\{M[i][j];\ i = 1 \ldots m,\ j = 1 \ldots n\}\right)$. In serial approach we compute matrix M row after row. The optimal local alignment itself can be retrieved by tracing back from the maximum entry in M. This algorithm requires mn steps and at least mn space.

3.1 One Row Version

In order to achieve effective FPGA implementation we need to reduce memory requirement of the algorithm. Only one row c of a matrix M and few temporary variables is sufficient to implement serial version, which is also effective for PC

```
1.  int c[n+1]; for (j=0; j<=n; ++j) c[j]=0;
2.  max_score=-MAXSCORE;
3.  for (i=1; i<=m; ++i) {
4.      Diag=0;Left=0;
5.      for (j=1; j<=n; ++j) {
6.          Upper=c[j];
7.          DLU=max(Diag+sim(s[i],t[j]), max(Left,Upper) + GAP,0);
8.          c[j]=DLU;
9.          Diag=Upper; Left=DLU;
10.         max_score=max(DLU, max_score);
11.     }
12. }
```

Fig. 1. Modification of the Simth-Waterman algorithm using only one row c

implementation, because all data fits into cache memory. The most interesting is optimal alignment score which is computed in the FPGA. For those scarce sequences from database which turn out to be enough similar to the query sequence, optimal alignments can be computed in desktop computer within short time. The algorithm written in C-like syntax is shown in the Fig. 1. Similarity matrix $sim(s,t)$ in program is an alias for $\rho_{edit}(s,t)$ and we assume constant penalty for inserting a gap: $GAP = \rho_{edit}(s_i, \epsilon) = \rho_{edit}(\epsilon, t_j)$. This algorithm is a basis for a parallelized and pipelined FPGA implementation.

4 Pipelined and Parallelized FPGA Implementation of FPGA Smith-Waterman Algorithm

Direct implementation of the Smith-Waterman algorithm (Fig. 1) led to 40 MHz frequency of the main clock. To make algorithm faster, pipelining was used. Pipelining is a technique that can speed-up a complex operations by breaking it into smaller concurrently executing stages. The maximum speed of pipelined operation is as good as the slowest single stage, at the cost of lengthening number of clocks required to obtain completion of full complex operation. Pipelining can be used only while processing independent data. We need at least k independent data chunks, where k is the number of stages. The key observation in Smith-Waterman algorithm is that each cell in a matrix can be calculated using its only three neighboring cells (Left, Upper and Diagonal), so data laying on "diagonal" does not depend on each other as presented in the Fig. 2.

In real implementation we have more complex pipelining schema (compared to the three stages shown in Fig. 2). There are 7 stages, each dependent on calculations of the previous one: Move generator, Sequence reading, Score matrix reading, Maximum counting (takes 3 cycles), Global Max finder. Additionally, temporal signals update is made. All pipelining stages are executed concurrently, synchronized by clock signal, but each stage computes different entry in a matrix. In this way, every clock cycle there is one new matrix entry which have passed all

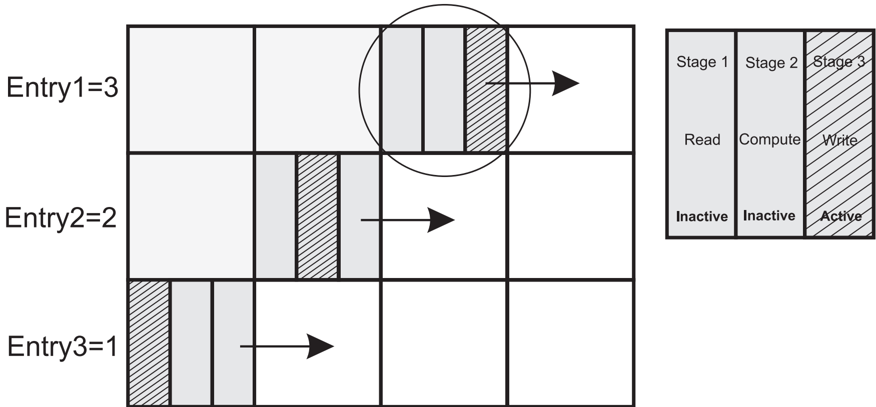

Fig. 2. Simplified view of the pipelining with three stages. For each matrix entry being calculated (shaded) there is stored its actual stage number executed: *(entry1)* is at stage 1, *(entry2)* at stage 2. Arrows indicate direction of movement. When an entry has passed all stages it moves to the next position in a row (if available) or advances to the next uncalculated row (shown on the Fig. 3)

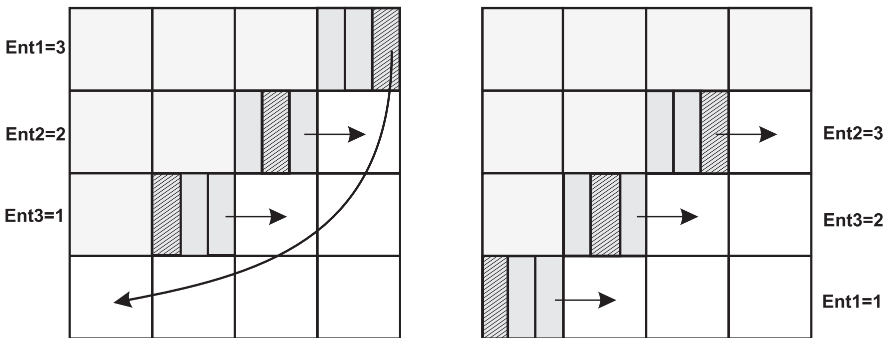

Fig. 3. Advancing to a new row: *(entry1)* has passed all stages and advances to the next uncalculated row, beginning with first stage

the stages of computation. Developed VHDL program uses a number of signals to keep track of calculations. More detailed view (showing only three entries being calculated) of temporal signals used in pipelining structure is presented in Fig. 4. In order to maintain boundary conditions properly, we use dummy buffer. It is needed when computations start from a new row, and during calculations of last rows. Each entry in a matrix needs information about its Upper, Left and Diagonal neighbor. For this purpose $H1, H2$ arrays are introduced. Depth of $H1$ and $H2$ is equal to number of stages and its width is 18 bits. These small arrays are implemented in Configurable Logic Blocks (CLBs). Each processing unit needs to know, which matrix entry is supposed to compute, so we need signals: $i_k, j_k, k = 1 \ldots 7$. The row c is red by $entry1$ and written to by $entry7$.

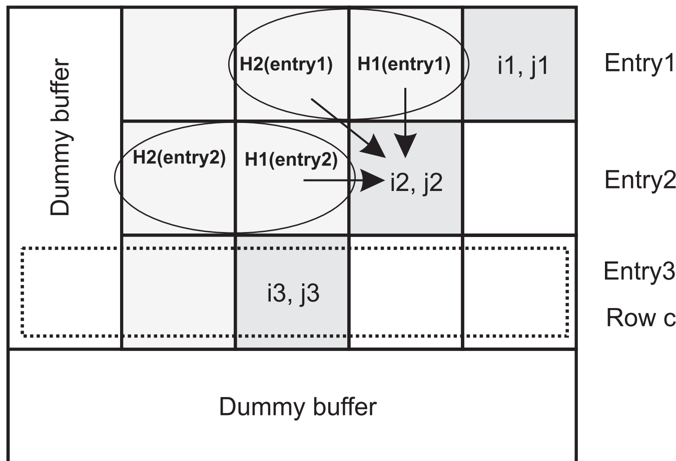

Fig. 4. How the history arrays $H1, H2$ and row c are kept and used: *(entry1)* uses the row c to retrieve its Upper, Diagonal neighbors when it advances to the next row. Other entries take this information from $H1, H2$ arrays. Row c is written by last entry. Dummy buffer filled with zeros is kept to maintain boundary conditions. In this way it is sufficient to store only one temporary row c, despite many stages

Information about neighbors is taken from arrays $H1, H2$ in a case of entries $2 \ldots 7$, and from the row c in a case of $entry1$. When $entry1$ is in first row, it reads c initially set to zero, but when it moves to next uncomputed row (Fig. 3) c is filled by the information provided by $entry7$.

Data Storage. Let us look at the details of the Xilinx Virtex 2 Pro implementation. Each aminoacid sequence is kept in 18 Kbit BlockRAM, which is true dual port memory with simultaneous read/write operations in each port. BlockRAMs are the key component of Xilinx devices, making it very suitable for the sequence alignment algorithms. Please note, that dual port BlockRAM memory is able to read from and write to a different address in a single clock. This enables to stages execute in parallel during reading and writing to a row c. Maximum BlockRAM density is found in Xilinx Virtex 2 Pro series (up to 444 in one chip), providing excellent maximum memory throughput. Large memory storage space is also needed for a row c, where temporal scores are kept. As it turn out, it is ineffective to keep aminoacid sequences of length 2048 and row c in distributed logic due to large number of CLBs utilized. This leads to low parallelism - we cannot implement many copies running in parallel within one chip. According to the algorithm properties we need five BlockRAMs for each algorithm copy: one for keeping edit score function ρ_{edit}, one for query sequence, one for database sequence, two for row c. Other signals are kept in CLBs. BlockRAMs can have different aspect ratio. In this implementation we store 2048 9 bit values, which is sufficient for keeping aminoacids. It is not enough for keeping temporary score in row c, so we use two BlockRAMs for that purpose, obtaining

18 bits. Assuming average protein sequence length $l = 300$ residues, one can see that I/O is not a bottleneck, because of quadratic complexity of the algorithm. If we implement 88 parallel PUs in FPGA we need to tranfer database sequences (query sequence is loaded once per database search) $88 * 5bit$ every 300 180 MHz cycles - giving about 30 MBytes/s transfer rate, and standard 32 bit/ 33 MHz PCI interface turns out to be sufficient. Data is transferred using second port of BlockRam used to store database sequence, which can be uploaded with a new database sequence independently of read performed during computations of current database sequence. For testing purposes we have used opencores [29] PCI interface and obtained 80 MB/s write transfer rate using Memec Design Spartan II PCI board.

Parallelism. Highly parallel FPGA implementation can be achieved at two levels. First is internal parallelism: by making many copies (up to 88) within one chip of the same processing unit executing the algorithm in order to process one fixed query sequence against many different sequences from a database. This is different parallelization method compared to often used systolic arrays, and leads to better PUs utilization. In systolic arrays average number of utilized PUs is dependent on sequence length [9].

Second level could involve hybrid solution: cluster of desktop computers, each equipped by FPGA accelerator in a form of PCI card. Not very intense communication between desktop computers can be performed by standard Ethernet. Parallelization level is limited by the FPGA size. Limiting factor in presented solution is the BlockRams count (5 BRams per PU). Control logic occupy about 44% of CLBs in Virtex 2 Pro when 97% of BlockRams are allocated (XC2VP70 chip). This explains difference in performance (Table. 1), between implementation in XC2V80 (contains 8 BRams) and XC2VP70 (contains 328 BRams).

5 Benchmarks

We used the following combination of software. Logical synthesis by Synplicity Amplify 3.5 [28] program. Place and Routing by ISE Xilinx tools [27]. Detailed timing analysis on post and route implementation revealed that the clock frequency is about 180 MHz on (-5) device speed version.

For the comparison purposes we have evaluated speed of widely used software from Pearson [19] FASTA 3.4 package - OSEARCH (dropnsw.c,v 3.4t23 March 5, 2003) running on Pentium IV 1.7 GHz CPU (gcc 3.2.2 , intel(c) 6.0 compiler) under Linux 9 and on SUN UltraSPARC-III 900 MHz with SUN Workshop 5.0 compilers. Best result are summarized in Table. 1. Please note that we have omitted Intel(c) icc compiler results, because it did not perform any better in this case (possibly due to lack of floating point operations). Benchmark was performed on aminoacid sequences of length 2048 residues.

Our program has been successfully verified by implementation on Memec Design Spartan II FPGA board, in limited version due to Spartan II architecture. Limitations apply to sequence length of 512 residues (smaller BlockRams).

Table 1. Comparison of various implementations of Smith-Waterman algorithm

Hardware	Compiler	Program	Residues /s
Pentium IV 1.7 GHz	gcc -O3 (3.3.2)	osearch34 -3	$49.30 \cdot 10^6$
UltraSPARC-III 900 MHz	cc -fast -xarch=v9a	osearch34 -3	$37.60 \cdot 10^6$
XC2V80-5	Amplify 3.5	this work	$180 \cdot 10^6$
XC2VP70-5	Amplify 3.5	this work	$11180 \cdot 10^6$

Implementation in currently available FPGA XC2VP70 turns out to be at least 200 times faster than Pentium IV 1.7 GHz.

6 Related Work

Acceleration of DNA sequence alignment using FPGAs and VLSI devices has a long history: VLSI implementation [12], SPLASH [11] and SPLASH2 [10] (computes simple edit distance for both 2 and 4 bit alphabet) system based on FPGA, however little attention has been paid to a more general case of protein seqence alignment with full 8 bit substitution matrix, where the computational problem does not reduce to simpler formulas used in LLCS computations. Moreover, it would be interesting to compare those DNA versions with accelerated by bitvector algorithms working on general purpose computers [16].

Direct comparison is quite difficult, because almost each implementation reported in literature is prepared for different parameter sets and is working on different hardware (different FPGA chips, VLSI or software). We list some of the implementations and compare its area of applications.

First group of implementations calculate DNA simple edit distance, mainly using systolic array parallelization. One of the latest achievements in DNA sequence alignment is presented in [1], where impressive number of 4,032 PEs were placed on an XCV1000E-6 device running 202 MHz, without the need for runtime reconfiguration. Similiar density is achieved using runtime configuration in [4]. Design working on 8 bit alphabet, but with constant value for a character mismatch is presented in [2]. Another solution based on FPGA is presented in [5], which is able to compute DNA simple edit distance score and the optimal alignment.

Second group involves existing implementations of more general case of the Smith-Waterman algorithm with full similarity matrix. [7,8,9,13] is a design using VLSI chips and is able to use full substitution matrix. Interesting FPGA design is presented in [3], but the sequence being compared is compiled into the design, which may be impractical for large FPGA chips. Commercial VLSI hardware is provided by Paracel [15]. Software implementation is given by commercial Celera [15]. The other known implementation is presented by Timelogic [26]. Their FPGA based machine Decypher is a commercial product, and details of their implementation are not published. Sencel company [6] sells interesting software solution, using MMX and similiar extended instruction set of general purpose

processors to achieve parallelism, but it is limited to 8 bit score in order to achieve 6 times faster computations.

7 Conclusions

DNA alignment with a simple edit score problem has a long history and there exist a number of efficient FPGA implementations.

We investigate the possibility of implementing more general case of the Smith-Waterman algorithm allowing alignment of protein sequences (20 letters alphabet) with full substitution cost matrix (400 entries, 8 bits each) and 18 bits of alignment score using Xilinx Virtex 2 pro FPGAs.

The algorithm has been adopted to FPGA architecture which resulted in good parallelization and efficient pipelining. Presented benchmarks show that our implementation can run on a modern FPGA chip over high speed which results in 200 times faster execution compared to the standard OSEARCH program compiled on a PC workstation.

Future work will develop new more complex and faster homology search algorithms suited for FPGAs. Number of such algorithms has been developed for general purpose processors [19,20,21,22,24], but most of them has significant memory requirements and cannot be directly implemented in FPGAs.

Other interesting topic is multidimensional sequence alignment [23], which is useful in determining functional relationships, but computationally very demanding.

References

1. C.W. Yu, K.H. Kwong, K.H. Lee and P.H.W. Leong. A Smith-Waterman Systolic Cell.*Proceedings of the Tenth International Workshop on Field Programmable Logic and Applications (FPL'03)*, Lisbon, pp. 375-384, 2003
2. B. West, R. D. Chamberlain, R. Indeck, and Q. Zhang. An FPGA-based Search Engine for Unstructured Database. *Proc. of 2nd Workshop on Application Specific Processors*, December 2003.
3. N. Weaver, Y. Markovskiy, Y. Patel, J. Wawrzynek. Post Placement C-slow Retiming for the Xilinx Virtex FPGA. *11th ACM Symposium of Field Programmable Gate Arrays (FPGA)* , 2003
4. S. A. Guccione and E. Keller. Gene matching using JBits. *Field-Programmable Logic and Applications, Reconfigurable Computing 12th International Conference*, p. 1168-1171, September 2-4, 2002.
5. Y. Yamaguchi, T. Maruyama and A. Konagaya. High Speed Homology Search with FPGAs. *Pacific Symposium on Biocomputing* 7:271–282, 2002.
6. T. Rognes and E. Seeberg. Six-fold speedup of Smith-Waterman sequence database searches using parallel processing on common microprocessors.*Bioinformatics*, Vol. 16 no. 8, 2000, p. 699-706
7. D. Lavenier. Speeding up genome computations with a systolic accelerator. *SIAM News*, vol. 31, no. 8, Oct. 1998.

8. J. D. Hirshber, R. Hughey, K. Karplus. Kestrel. A Programmable Array for Sequence Analysis. *Proc. Int. Conf. Application-Specific Systems, Architectures, and Processors, IEEE CS*, Aug. 19-21, 1996, pp. 25-35
9. D. Lavenier. SAMBA: Systolic Accelerators for Molecular Biological Applications, *IRISA Report, (PI-988)* , March 1996.
10. D. T. Hoang. Searching genetic databases on splash 2. *Proceedings 1993 IEEE Workshop on Field-Programmable Custom Computing Machines*, p. 185-192, 1993.
11. Hoang, Dzung T. FPGA Implementation of Systolic Sequence Alignment. *International Workshop on Field Programmable Logic and Applications*, Vienna, Austria, Aug. 31 - Sept. 2, 1992.
12. R.J. Lipton, and D. Lopresti. A systolic array for rapid string comparison. *Proceedings of the Chapel Hill Conference on VLSI*, p. 363-376, 1985.
13. Paracel, inc. http://www.paracel.com.
14. Sencel's search software. http://www.sencel.com.
15. Celera genomics, inc. http://www.celera.com.
16. M. Crochemore, C.Iliopoulos, Y. Pinzon, J. Reid. A Fast and Practical Bit-Vector Algorithm for the Longest Common Subsequence Problem. *Information Processing Letters*, Vol. 80, Issue 6, p. 279 - 285, Dec. 2001
17. T. F. Smith, M. S. Waterman. Identifcation of Common Molecular Subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.
18. M. S. Waterman. *Introduction to Computational Biology: Sequences, Maps and Genomes.* Chapman and Hall, London, 1995.
19. W. R. Pearson. Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics* 11(3):635-650, 1991
20. W. R. Pearson, D. J. Lipman. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA* 85(8), 2444–2448, 1988.
21. W. R. Pearson. Rapid and sensitive sequence comparison with fastp and fasta. *Methods in Enzymology*, 183:63–98, 1990.
22. B. Ma, J. Tromp, M. Li. PatternHunter: Faster and More Sensitive Homology Search. *Bioinformatics.* 18(3):440-5, 2002.
23. G. Z. Hertz, G. D. Stormo. Identifing DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics* 15(7/8):563–577, 1999.
24. A. Davidson. A Fast Pruning Algorithm for Optimal Sequence Alignment. *Proceedings 2nd Annual IEEE International Symposium on Bioinformatics and Bioengineering (BIBE 2001).* IEEE Comput. Soc. Los Alamitos CA, USA, pp.49–56, 2001.
25. S. Henikoff, J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. matl. Acad. Sci. USA.* 89, 10915-10919, 1992.
26. Timelogic home page. http://www.timelogic.com
27. Xilinx home page. http://www.xilinx.com
28. Synplicity home page. http://www.synplicity.com
29. Opencores home page. http://www.opencores.org