Reducing Storage Requirements in Accelerating Algorithm of Global BioSequence Alignment on FPGA

Fei Xia and Yong Dou

Department of Computer Science, National University of Defence Technology, Changsha, P. R. China, 410073
{xcyphoenix,yong_dou}@hotmail.com

Abstract. In the paper, we present storage optimization scheme for hardware accelerating Needleman-Wunsch algorithm. The scheme exploits the characteristics of back-tracking phase in which the back-trace path only travels in a constrained area. Our analysis shows that in addition to logic element resource and memory capacity, the number of RAM blocks is also one of the constrained factors for hardware accelerating bio-sequence alignment. The optimized algorithm only store part of the score matrix to reduce storage usages of FPGA RAM blocks, and implement more processing element in FPGA. We fit our design on FPGA chips EP2S130 and XC2VP70. The experimental results show that the peak performance can reach 77.7 GCUPS (Giga cell updates per second) and 46.82 GCUPS respectively. Our implementation is superior to related works in clock frequency, the maximal PE number and peak performance, respectively.

Keywords: Bioinformatics, FPGA, Global BioSequence Alignment, Needleman-Wunsch algorithm, Hardware Accelerator.

1 Introduction

With the technology development of the genome sequencing, the scale of Gene database expands steeply. In August 2005, the INSDC announced the DNA sequence database exceeded 100 gigabytes and the number of sequence reached over 52 million[1]. It is inefficient to scan the Gene-Bank using traditional software approach. In recent years, FPGA have emerged as performance accelerators capable of implementing fine-grained, potential massively parallelized algorithm for computation-intensive applications. The reconfigurable FPGA chips also enable algorithms to be implemented with different computing structures on the same hardware platform[2]. As a result, hardware accelerating bio-sequence matching attracts much more attention.

After Needleman-Wunsch algorithm was published in 1970, it soon became the standard technique in biological matching, which uses DP-based method (dynamic-programming) and is suitable for global alignment of pair-wise sequence with a certain similarity. It also spawned many variations, including the famous Smith-Waterman algorithm for local alignment.

M. Xu et al. (Eds.): APPT 2007, LNCS 4847, pp. 90–99, 2007.

Because sequence comparison algorithms based on DP have been proven to produce an optimal alignment, a number of hardware parallel architecture based on the Needleman-Wunsch or the Smith-Waterman algorithm have been proposed for sequence analysis[3],[4],[5],[6],[7],[8],[10] and most of them only concentrate on the scoring phase. Some implementations[5],[11] address on the structure and scale of PE array, but did not consider the storage problem of DP matrix. Works in[3],[6],[7] and[9] discussed how to accelerate the scoring process and enhance the performance scaling, but did not implement the backtracking process. Researches in[4],[8] mapped the backtracking process in FPGA, but did not take the storage optimization of scoring matrix into account.

Since the storage complexity of DP-based method is $O(M \times N)$ for two strings with size M and N. Furthermore, the required port number is linear scale with the size of PE array. With the growth of sequence length, it is difficult to implement both of the scoring and backtracking process of DP matrix in FPGA. To reduce the storage requirement in Needleman-Wunsch algorithm, we present a storage optimization scheme for global bio-sequence alignment applications with backtracking. Based on the analysis to the characteristic of Needleman-Wunsch algorithm, we find that given a fixed scoring method, the backtracking paths always fall into a limited area in DP matrix even in the worst cases, which means storing all elements of DP matrix is unnecessary. Our scheme uses two extra registers for checking address limits, but saves about 50% of storage space in general cases. The saved memory blocks can be used to implement more processing elements. Our experimental results show over 800 PEs can be fitted in an FPGA chip of Altera EP2S130, the maximal speedup reach 5.6 compared to closely related works, and achieve peak performance 77.7 GCUPS.

2 Needleman-Wunsch Algorithm Overview

The basic idea of Needleman-Wunsch algorithm is to use the best alignment of shorter subsequence to build the best alignment of two sequences gradually and recursively. In practice, a matrix F is used to store alignment scores of subsequence. When F is figured out, the scoring process is finished. Needleman-Wunsch algorithm is composed of two phases: firstly, calculate the DP matrix and store the computing trace; secondly, execute trace-back operation according to DP matrix. In this paper we use the convention that the query sequence S with length M is along the vertical dimension and the sequence L with length N in database along the horizontal dimension.

Scoring Phase: Suppose $F(i,j)$ represents the best alignment score between subsequence $S_{1...i}$ and subsequence $L_{1...j}$. The score $F(i,j)$ for grid cell (i,j) is computed following below equations($1 \leq \mathrm{i} \leq M, 1 \leq j \leq N$).

Initialization:

$$\begin{cases} F(0,0) = 0 \\ F(i,0) = F(i-1,0) + P(S_i, -) \\ F(0,j) = F(0,j-1) + P(-, L_j) \end{cases} \tag{1}$$

Recurrence relation:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + P(S_i, L_j) \\ F(i-1,j) + P(S_i, -) \\ F(i,j-1) + P(-, L_j) \end{cases} \tag{2}$$

Thus, we can translate the above recurrence relation (2) into a systolic parallel algorithm executing on linear processing elements, PE, as shown in Fig.1(A).

Each PE calculates a row of DP matrix and stores corresponding trace flag in its local memory. Multiple PEs compute different elements in a line perpendicular to the main diagonal concurrently. In the scoring stage, PE[n]($n \geq 1$) receives score and current character in sequence L from PE[*n-1*] (PE[0]'s input is supplied by array control module). PE[n] calculates the score of current grid, generates trace-back flag and transmits the scoring result and current character to PE[*n+1*]. Finally, in step S5, PE[n] stores the flags into *PE_LM[n]* (PE[n]'s local memory).

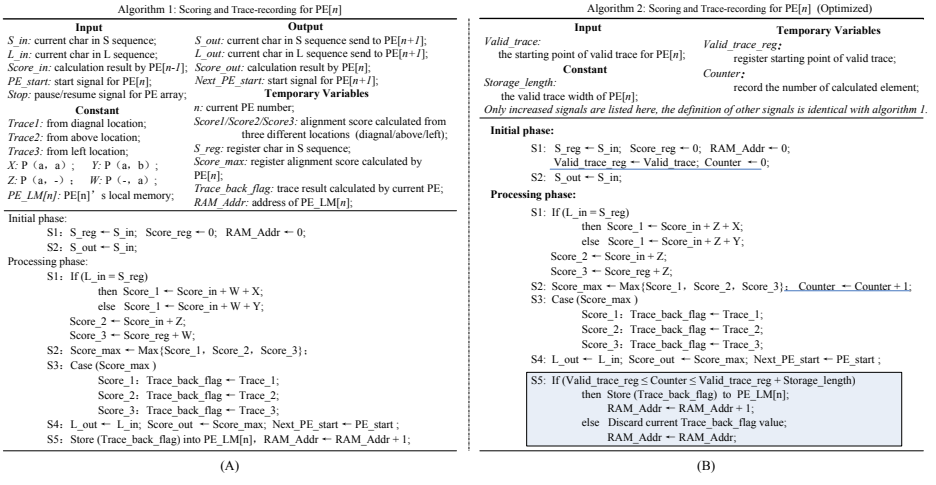

Fig. 1. (A)Scoring and trace-recording algorithm; (B)Optimized scoring and trace-recording algorithm for each PE

Trace-back Phase: After scoring phase, begin backtracking process as shown in Fig.2(A). The start point is set to the low-right element of DP matrix recorded in scoring phase. The trace-back processing can find out the location of the next trace-back point through the current flag. Then set the next point as the current trace-back point until reaches the top-left element of DP matrix.

At the end of backtracking, the path composed by trace-back points is the best alignment. In traditional Needleman-Wunsch algorithm, each PE is responsible for calculating and storing the all elements of corresponding row of DP matrix. The storage requirement is $M \times N$ (M and N are the length of input sequences). With the growth of sequence size, the storage requirements may exceed the

capacity of on-chip memory. In order to implement larger scale scoring and trace-back process on FPGA chips, we present a storage reduction strategy for Needleman-Wunsch algorithm.

3 Storage Optimization Strategy

Given two input sequences S and L, $|S| = M$, $|L| = N$. With universality, we suppose $N > M$. Moreover we using linear gap penalty model and the scoring scheme is shown as follows: $\mathrm{P}(\mathrm{a},\mathrm{a}) = \mathrm{x}, \mathrm{P}(\mathrm{a},\mathrm{b}) = -\mathrm{y}, \mathrm{P}(-,\mathrm{a}) = \mathrm{P}(\mathrm{a},-) = -\mathrm{z}$ $(\mathrm{x},\mathrm{y},\mathrm{z} > 0)$. Parameter G represents the number of gaps in sequence L. R is the number of replace operation in sequence S. The optimized algorithm is shown in Fig.1(B).

The basic calculating process of optimized algorithm is consistent with traditional Needleman-Wunsch algorithm. The main difference lies in S5, where two registers, *valid_trace_reg* and *counter* are used to check the address range so that only valid traces are stored in PE local memory. The former register is filled with starting location of valid trace range in the phase of initialization and the latter records the location of current point of DP matrix. Therefore, Each PE only records partial elements of corresponding row of DP matrix. As a result, the largest memory cost of PE is $N - M + 2G + 1$, the total memory cost of the optimized algorithm is:

$$(N - M + G + 1) \times M, \qquad G \leq \frac{(M - R) \times (x + y)}{2 \cdot z + x} \tag{3}$$

The correctness of formula(3) is proved as follows: (only consider the situation $N > M$; when $N \leq M$, the conclusion is the same). The selection of trace-back path is closely related with the location of gaps inserted in the alignment result. As for trace-back point $F(i,j)$, there are three possible choices in trace selection: vertical path (trace a in Fig.2(A), pointing to element $F(i-1,j)$, it means inserting a gap at the location of L_j in horizontal sequence); horizontal path (trace b pointing to $F(i,j-1)$, it means inserting a gap at the location of S_i in vertical sequence) and diagonal path (trace c pointing to $F(i-1,j-1)$, it means the two sequences generating a match or mismatch at the location of $F(i,j)$).

When $G = 0$, there is no gap in sequence L, that means trace-back path contains no vertical trace, then the gaps in sequence S is $N - M$. So there are only $N - M$ horizontal traces and M diagonal traces in the trace-back path. Whatever the alignment score is, all possible paths must be fall into the shadow parallelogram area in Fig.2(A). The trace 1, 2 and 3 are three possible paths.

Therefore, we can get all the information about backtracking phase recording the elements in the above parallelogram shadow area. Thus the length of PE local memory is $N - M + 1$, and the storage cost of algorithm is $(N - M + 1) \times M$, the proportion of saved storage cost is:

$$T = \frac{M - 1}{N} \tag{4}$$

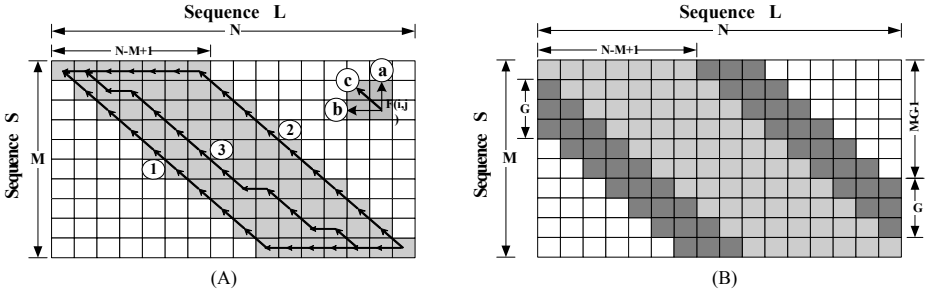

Fig. 2. (A)Valid trace area in DP matrix (G=0); (B)Valid trace area in DP Matrix (G≠0)

When G≠0, there are G vertical traces in backtracking path. Then the path will transcend the shadow parallelogram area in Fig.2(A). Thus it's necessary to extend the recording area. Two gaps matching each other are impossible, so the number of blanks in sequence S is $N - M + G$. There are only four situations in sequence alignment: match, replace, delete and insert (in-del). According to the linear gap penalty model and the scoring scheme above, the matching score is $(M - G - R)\cdot x$, in-del (gap) penalty is $(N - M + G)\cdot z + G\cdot z$ and replace penalty is $R\cdot y$, so the alignment score of pair-wise sequence S and L is $(M - G - R)\cdot x - R\cdot y - (N - M + 2G)\cdot z$. The score in the worst condition is $(N - M)\cdot(-z) - M\cdot y$, (any pair of characters in sequence S and L can't match in this situation). The alignment score in common condition should be greater than the worst case, thus

$$(M - G - R)\cdot x - R\cdot y - (N - M + 2G)\cdot z \geq (N - M)\cdot(-z) - M\cdot y \quad (5)$$

As a result of (5), G has an upper limit:

$$G \leq \frac{(M - R)\times(x + y)}{2\cdot z + x} \quad (6)$$

Therefore, we can obtain all trace-back information by recording only the elements in the shadow area as depicted in Fig.2(B). Thus the largest length of PE local memory is $(N - M + 2G + 1)$, and the whole storage cost is $(N \times M) - (M - G)\times(M - G - 1)$.

According to the analysis above, if the scoring rule is fixed, the gaps inserted in sequence L are limited by the range of G given in formula (6). The proportion of saved storage of optimized algorithm is:

$$T = \frac{(M - G)\times(M - G - 1)}{M \times N} \quad (7)$$

In implementation we should consider not only the capacity of memory and LE (Logic Cell) but also the size of RAM block and the port constrains of FPGA.

Suppose: L_F is number of logic cells in FPGA, C_F is RAM capacity, P_F is the port number of RAM blocks in FPGA, M_C is the capacity of single RAM block,

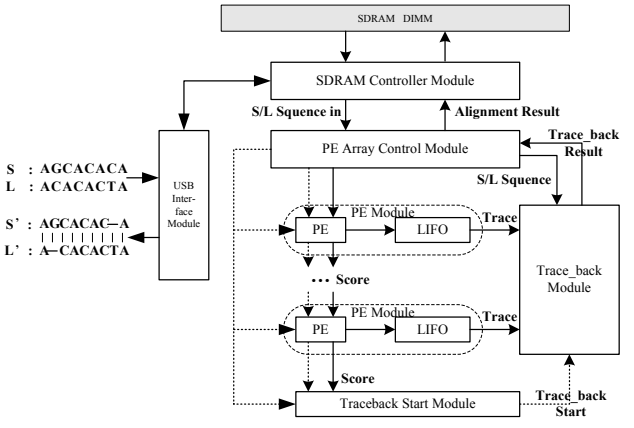

Fig. 3. The structure of N-W algorithm accelerator

P_{PE} is the port number PE uses and L_{PE} represents PE logic cost. The number of PE, N_{PE}, can be fit in a single chip must fulfill the following constrains:

(1) RAM capacity constrain: $N_{PE} \times (N - M + 2G + 1) \leq C_F$;
(2) Logic capacity constrain: $N_{PE} \times L_{PE} \leq L_F$;
(3) Memory port constrain: $N_{PE} \times P_{PE} \leq P_F$;

$$P_{PE} = \left[(N - M + 2G + 1) \cdot {}^{d}/_{M_C} \right] + 1 \tag{8}$$

Where d is storage cost of each element in DP matrix, $(N - M + 2G + 1) \cdot d$ is the memory cost of PE, square brackets means getting the floor of number.

The above analysis is suitable for other scoring rules. The optimized method not only can reduce memory cost without increasing design complexity but also increase the number of PEs. Furthermore, the experimental result shows that the port number of RAM blocks in FPGA is usually the main constraining factor.

4 Design and Implementation

We have implemented the optimized algorithm in the Altera StratixII EP2S130C5 FPGA. The test-bed of our algorithm accelerator includes a FPGA chip, two SDRAM modules and a USB Peripheral Controller. The algorithm core includes PE Array Control Module, PE Array, Trace-back Start Module and Trace-back Module. The structure of N-W algorithm accelerator is shown in Fig.3.

PE module contains score calculation unit (compute element in DP matrix), trace generation unit (generate trace mark) and trace storage LIFO (Last in First out queue, implemented by block RAM) shown in Fig.4(A). The structure of score calculation unit shown in Fig.4(B) consists of three adders and three comparators in terms of formula (2). Since the trace-back marks depend on scoring results, the score calculation becomes the critical path.

Trace-back module is in charge of finding out the valid trace-back flag and generating final alignment. It accesses PE local memory in trace-back phase.

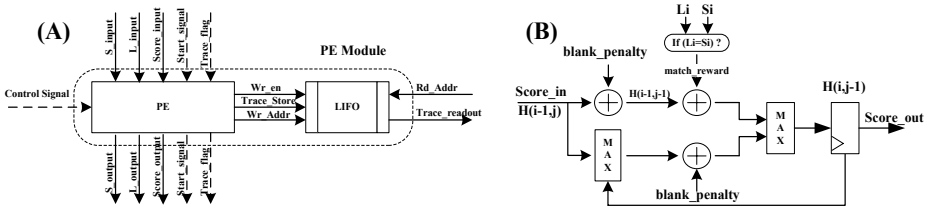

Fig. 4. (A) PE Module Structure and (B) Score Calculation Component

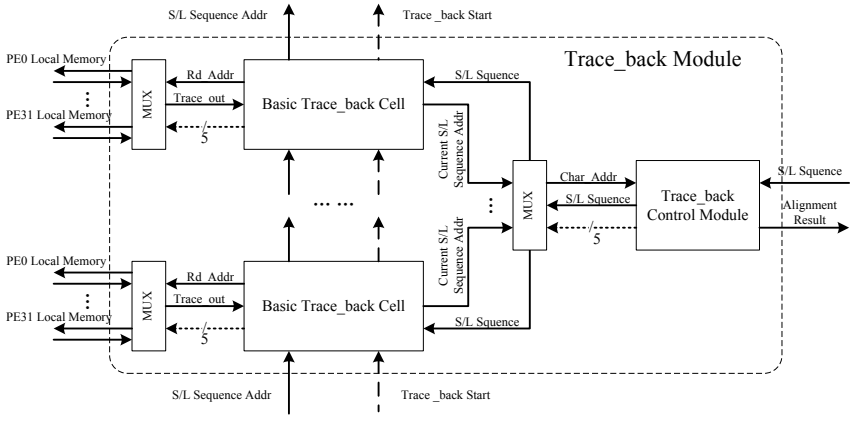

Fig. 5. The structure of trace-back module

When the scale of PE array is large enough, the huge multiplexer becomes the bottleneck in FPGA implementation. To solve the problem, we adopt the well-phased trace-back strategy. The structure of trace-back module shown in Fig.5 is composed of multiple basic trace-back cells and a control module. We divide the linear PE array into several groups so that each basic trace-back cell accesses the corresponding local memory group of PEs and controls the trace-back procedure of current stage (The experiments show that the 32 PEs per group is an optimal choice). The kernel of Basic Trace-back Cell is address generation component, which calculates the address of next trace-back point.

5 Experiments and Performance Comparison

We made experiments with different parameter G. To simplify experiment, we use the following scoring rules: $P(a,a) = 1$, $P(a,b) = -1$ and $P(-,a) = P(a,-) = -1$.

5.1 Reducing Local Memory Requirements

Our optimized storage scheme requires less memory size than traditional algorithm for storing DP matrix. This will save storage space for more PEs

implementation. Fig.6 indicates the experimental result. Given the PE number equals the length of sequence S, M=512 and G, the gaps in sequence L, equals 0, $M/8$, $M/4$ to $3M/8$ respectively. The saving storage percentage can be calculated as equation (7).

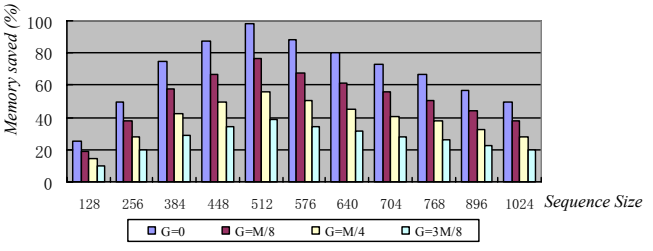

Fig. 6. Proportion of memory saved in different sequence size and parameter G

From the above figure, supposing the lengths of both sequence S and L equal to 512 and no gaps are inserted, our optimized scheme achieves the maximal storage reduction, nearly 99%, compared with traditional algorithm. Even in the worst cases, where the length difference between S and L rises and the inserted gaps also increases to $3M/8$, the reduction percentage still reach about 10%.

5.2 Increasing PE Number

Besides of the limitation in logical resource and memory capacity, the port number of FPGA RAM blocks also constrains the size of PE array. Since each PE occupies one LIFO, which is composed of at least one RAM block of FPGA. The saved storage will provide extra RAM blocks for more PE implementation. Table 1 shows the comparison of PE number implemented in FPGA EP2S130C5 with different length of sequence L.

Table 1. PE number for different sequence size

Sequence L(N)	128	256	512	1024	2048	4096
PEs(Traditional)	928	928	928	780	680	340
PEs (Optimized)	928	928	928	928	780	680

The maximum PE number fitted in FPGA is closely-related to the sequence size. In the condition of $N \leq 512$, the PEs can be fitted in FPGA is limited not by memory but logic resource. With the same FPGA logic resource, the maximal PE number is the same as 928.

But when $N > 512$, the storage factor takes more effects on the scale of PE array. The saved storage can implement more PEs in our optimized scheme than traditional algorithms. The difference between the maximal PE number increases greatly. When sequence length reaches 4096, our scheme can achieves double PE number, as shown in the last column of Table 1.

5.3 Experimental Result

We implemented our optimized algorithm on FPGA StratixII EP2S130F1020C5, supposing $N = 1024$, $M = 800$, $G = M/8 = 100$. The PE local memory capacity is $512 \times 2bit$ occupying two M512 RAM blocks or one M4K block. Fitting 800 PEs consumes 87% logical elements and the clock frequency reaches 97.13MHz, as shown in the first column of Table 2.

Table 2. Performance results and comparison ([*]: the usage of RAM Blocks)

	Ours		ASM[4]	HCP[3]	SRC[11]	PC[4]
FPGA	EP2S130C5	XC2VP70-5	XC2VP70-5	XC2V6000	XC2V6000	XeonPC
PEs Fitted	800	384	303	252	4 Engines	—
LEs (LUT)	87%	73%	—	—	/Chip	
M512 (%)[*]	394 (56%)	BRAMs				N-W
M4K (%)[*]	609(100%)	323 (98%)	—	—	4 Chips	Algo-
Mem Capacity	13%	11%				rithm
Clock (MHz)	97.13	121.93	77.5	55	100	3000
Speed (GCUPS)	77.7	46.82	23.48	13.9	42.7	0.046

Since traditional algorithm needs $N \times M = 800 \times 1024$ memory cells, it is impossible to generate 800 RAM blocks with the capacity of $1024 \times 2bit$ in EP2S130. The optimized approach reduces local memory usage of each PE and saves 50% storage cost, which makes the implementation can be fitted in EP2S130. From Table 2 we also find that the memory usage in our work is only 13%. The reason is that large part of memory capacity in FPGA is implemented by M-RAM block with size of 1Mbits, which can only be used by at most two processing elements. Thus, the bottleneck lies in the number of RAM blocks, not memory capacity for sequence alignment application.

For comparison to related work, we also implement 384 PEs on FPGA chip of Xilinx XC2VP70-5. The result shows our design is superior to the implementation[4] in both PE number and clock frequency. The performance speedup is nearly 2.0. We also compared our performance to the closely related proposals, HCP[3] and SRC[11]. Our implementation achieves the peak performance of 77.7 GCUPS on EP2S130 and the speedup can reach 5.6 and 1.8 relatively.

In addition, we tested the execute time of global pair-wise sequence alignment with backtracking in our FPGA testbed. With sequence length 512, the scanning time of total 1000 sequences is 90.8ms. For the same application on a PentiumIV 2.6 GHz PC, the run time is 31770ms. Hence, our FPGA implementation achieves a speedup of approximately 350.

6 Conclusion

This paper presented the design and implementation of storage reduction strategy of global bio-sequence alignment with backtracking on FPGAs. The proposed

scheme can efficiently reduce the storage cost by shortening the length of PE local memory, and increase the scale of PE array fitted in FPGA. Experimental results showed our implementation is superior to related works in frequency, maximum PE number and peak performance.

References

1. GenBank Growth Statistics (March 7 2006), `http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html`
2. Regester, K., Byun, J., et al.: Implementing Bio-informatics Algorithms on Nallatech-Configurable Multi-FPGA Syatems.University of North Carolina (2005)
3. Oliver, T., Schmidt, B., Maskell, D.: Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs. In: Proc. ACM/SIGDA 13th International Symposium on Field Programmable Gate Arrays, pp. 229–237 (2005)
4. Court, T.V., et al.: Families of FPGA-Based Accelerators for Approximate String Matching. Journal of Microprocessors and Microsystems 31, 135–145 (2007)
5. Dydel, S., Bala, P.: Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices. In: Proc. IEEE Int. Conf. Field Programmable Logic and Application, pp. 23–32. IEEE Computer Society Press, Los Alamitos (2004)
6. Yu, C.W., Kwong, K.H., et al.: A Smith-Waterman Systolic Cell. Proc. IEEE Int. Conf. Field Programmable Logic and Application, 375–384 (2003)
7. Peiheng, Z., Xinchun, L., Xiangyang, J.: An Implemention of Reconfigurable Computing Accelerator Card Oriented Bioinformatics. Journal of Computer Research and development, 930–937 (2005)
8. West, B., et al.: An FPGA-based Search Engine for Unstructured Database. In Proc. of 2nd Workshop on Application Specific Processors, 25–32 (2003)
9. Herbordt, M.C., Model, J., et al.: Single Pass, BLAST-Like, Approximate String Matching on FPGAs. In: Proc. IEEE 14th IEEE Int. Symp. Field-Programmable Custom Computing Machines, pp. 217–226 (2006)
10. Michailidis, P.D., Konstantinos, G.: Margaritis:A Programmable Array Processor Architecture for Flexible Approximate String Matching Algorithms. Journal of Parallel and Distributed Computing 67, 131–141 (2007)
11. El-Ghazawi, T.: The High-Performance Reconfigurable Computing Era. GWU HPC Symposium (2006)