Reconfigurable Systems for Sequence Alignment and for General Dynamic Programming[§]

Ricardo P. Jacobi[1], Mauricio Ayala-Rincón[2], Luis G. A. Carvalho[1], Carlos H. Llanos[3] and Reiner W. Hartenstein[4]

Departamentos de [1]Ciência da Computação, [2]Matemática e de [3]Engenharia Mecânica, Universidade de Brasília 70910-900 Brasília D. F., Brazil
{[1]rjacobi@cic,[2]ayala@mat, [3]llanos@}.unb.br
[4] Fachbereich Informatik, Technische Universität Kaiserslautern, Germany
hartenst@rhrk.uni-kl.de

Abstract. Reconfigurable systolic arrays can be adapted for efficiently resolving a wide spectrum of computational problems: parallelism is naturally explored over systolic arrays and reconfigurability allows for redefinition of the interconnections and operations even at run time (dynamically). We present a reconfigurable systolic architecture which is applied to the efficient treatment of several dynamic programming methods for resolving well-known problems such as global and local sequence alignment, approximate string matching and longest common subsequence. Dynamicity of the reconfigurability is showed of relevance for practical applications to construction of sequence alignments. A VHDL description of the conceived architecture is implemented synthesized over an FPGA of the APEX family.

1 Introduction

A lot of effort have been done in recent years by the scientific community in order to better understand the Human Genoma. This effort was aided by the advances in computer algorithms and hardware, which allowed the identification of the almost 40 thousand human genes. The first results were published by Nature [11] and Science [20] in February, 2001. After the first results, a huge amount of biological data was generated and stored in databases. For instance, GenBank, one of the main public genome databases has been feed at an exponential rate in time last years.

Due to the huge size of DNA sequences, purely software based implementations of the Smith-Waterman algorithm [18], whose space and running time complexity belong to $O(mn)$ for sequences of size m and n, do not compete with high sensitive linear approximate solutions as the ones implemented in the well-known systems FASTA and BLAST. On the other side, using dedicated hardware, matching can be processed in parallel reducing the order to $O(m+n)$. But most of the dedicated hardware solutions are in first place expensive and in second place lacks of flexibility to be adapted to different problems. Solutions based on reconfigurable devices such as Field Programmable Gate Arrays (FPGAs) may provide for those needs. The former solutions can be classified as purely *soft*ware approaches oriented to the exploration of parallel *hard*ware architectures as in *Single Instruction Multiple-Data* (SIMD) and *MultiMedia eXtensions* (MMX) available in

[§] Work supported by the CAPES-DFG foundations. Partially supported by FINATEC.

Intel microprocessors [17]. The latter solutions can be classified as dedicated or *config-ware/morphware* approaches where the threshold between what is *hard* and what is *soft* is flexible allowing for sophisticated "algorithmic" solutions embedding reconfiguration and execution instructions [4].

This work presents a prototype of a reconfigurable systolic architecture adequate for treating problems which are solvable by general dynamic programming algorithms. The architecture was modeled by a rewriting-logic based methodology using ELAN in [3] following the original lines of design also applied for a space/time efficient implementation of the Fast Fourier Transform in [2]. Its hardware design and prototyping is presented here, which includes the architectural description, the specification with VHDL and simulation and synthesis using the Altera - Quartus II system [1]. While most hardware solutions limits to sequence comparison, which gives an estimation of the sequence similarity, our work target sequence alignment, producing effectively the sequence matchings.

The paper is organized as follows: section 2 introduce basic concepts on systolic arrays and reconfigurable systems; section 3 overviews previous work in the area; section 4 presents the systolic architecture and explains why reconfiguration is relevant; section 5 describes the VHDL specification, synthesis and results and section 6 presents conclusions and future work.

2 Systolic arrays and Reconfigurable Systems

The term *systolic array* has been coined probably by H. T. Kung around 1979 [10]. A systolic array is a mesh-connected pipe network of DPUs (datapath units), using only nearest neighbor (NN) interconnect. DPU functional units usually operate synchronously, processing streams of data that traverse the network (also asynchronous mode of operation is possible, where sometimes also the term *wavefront array* is used instead of *systolic array*). Systolic arrays provide a large amount of parallelism and are well adapted to a restrict set of computational problems: those which present strictly regular data dependencies. Some typical structures are shown in figure 1.

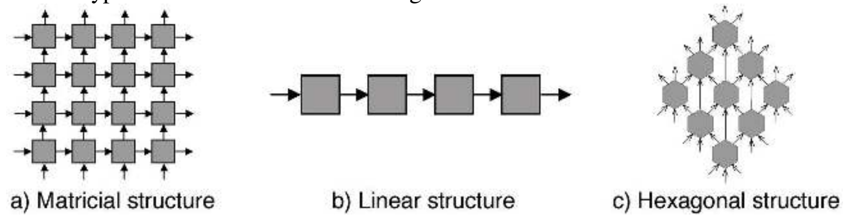

Figure 1. Some systolic structures.

Systolic array restrictions may be circumvented by using reconfigurable circuits: the same system may be reconfigured in order to deal with different tasks.

3 Related Work

In 1985, Lipton and Lopresti [13] had shown that the parallelism in Smith-Waterman algorithm can be mapped into a linear bidirectional systolic architecture. Each processing element (PE) in that structure computes one of the diagonals of the similarity matrix (figure 2).

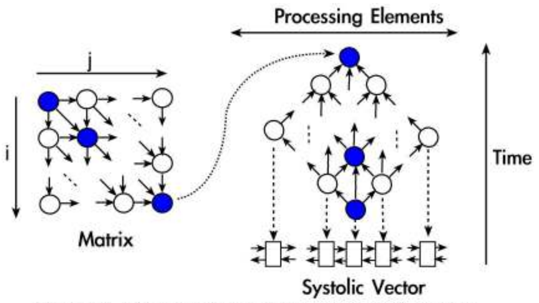

Figure 2. Mapping the matrix to a systolic vector.

Sequences to be compared should be input to the vector from opposite sides and were shifted at each clock cycle to cross the vector. If the two sequences had sizes m and n then the vector should have *n+m–1* elements. The result provided by this architecture is a value that indicates the degree of similarity between the sequences.

In 1992, Hoang [8] proposes a similar solution based on SPLASH [5] architecture, which is a matrix of programmable logic devices developed by the Supercomputer Research Center (SRC), using 32 XC30990 FPGAs from Xilinx [22]. They allowed the user to recover at least one alignment, using Lipton and Lopresti grading scheme. An improvement was later proposed based on a new version of SPLASH. Lavenier, in 1998, develop SAMBA [12] (Systolic Accelerator for Molecular Biological Applications) another systolic alternative to compute sequence comparison. Later, in 2002, a new version of Hoang approach was presented based on Virtex FPGAs from Xilinx [6]. Hoang solution also inspired HokieGene [16], a reconfigurable system implemented with Osiris [9] card, developed by the *Information Sciences Institute*. An architecture which is similar to this work was presented by Yamaguchi, Maruyama and Konagaya [24] in 2002. It uses a PCI card with a XCV2000E FPGA from Xilinx, which contains 43200 logic cells that can hold 144 processor nodes, but each processor takes four clock cycles to compare two bases. Another solution still based on Hoang was proposed in 2003 by Yu, Kwong, Lee and Leong [25]. It uses Xilinx XCV1000E FPGAs, with 27648 logic cells. Some ASICs were also developed, as BioScan [21] in 1991, KESTREL [7] in 1996 and the Proclets of Yang [23] in 2002. Commercial products in this area are DeCypher [19] from TimeLogic, based on FPGAs and GeneMatcher2 [15] from Paracel, which employs a dedicated ASIC.

There are some interesting remarks concerning these works. First, almost all of them only computes the comparison and do not produce the alignments between sequences and subsequences. Moreover, the cost function adopted was modified in order to simplify the hardware. This may produce inaccurate results according to researchers from biology field. In this work the cost function adopted follows the biological grading system.

4 The Reconfigurable Systolic Array

4.1 Conception of the Systolic Array

The conceived architecture is reconfigured for the treatment of problems such as: local and global sequence alignment (LSA and GSA) between two sequences: *s, t*; longest common subsequence (LCS) between two strings: *s, t*; *k*-approximate string matching (ASM) of a pattern in a string: *s, t.*

All these problems are similarly solved by dynamic programming algorithms that build a table *V* of size $m+1 \times n+1$, where *m* and *n* are the length of the two input sequences or strings ($|s| = m$, $|t| = n$). The computation of the *i, j*-th components of all these tables are based on the values of the previous components in the same row (*i,j-1*), column (*i-1,j*) and diagonal (*i-1,j-1*). Components of these tables are denoted by *V[i, j]* and symbols of the sequences by *s[i], t[j]*. These computations are respectively given for these problems by the following recurrence relations:

- *LSA: V[i,j] = max(V[i,j-1]-2, V[i-1,j]-2, V[i-1,j-1]+p, 0)*, where if *s[i]=t[j]* then *p=1* else *p = -1* and *V[i,0] = 0,* for *i=0..m* and *V[0,j]=0,* for *j=0..n.*
- *GSA: V[i,j] = max(V[i,j-1]-2, V[i-1,j]-2, V[i-1,j-1]+p)*, where if *s[i]=t[j]* then *p=1* else *p=-1* and *V[i,0]=-2×i,* for *i=0..m* and *V[0,j]=-2×j,* for *j=0..n.*
- *LCS: V[i,j] = max(V[i,j-1], V[i-1,j], V[i-1,j-1]+p)*, where if *s[i]=t[j]* then *p=1* else *p = 0* and *V[i,0] = 0,* for *i=0..m* and *V[0,j] = 0,* for *j=0..n.*
- *ASM: V[i,j] = min(V[i,j-1]+1, V[i-1,j]+1, V[i-1,j-1]+p)*, where if *s[i]=t[j]* then *p=0* else *p = 1* and *V[i,0] = i,* for *i=0..m* and *V[0,j] = j,* for *j=0..n.*

Components of these dynamic programming tables are sequentially computed from left to right and top to down, but parallelization is possible by computing all components in one (minor) diagonal in a sole step, starting from the first diagonal (*i+j=2*) and finishing in the last diagonal (*i+j=n+m*). Notice that for computing values in the diagonal *k* (*i+j=k*) it is necessary to maintain values of the previous two diagonals *k-1* (since, *i-1+j=i+j-1=k-1*) and *k-2* (since, *i-1+j-1=k-2*) as shown in figure 3.

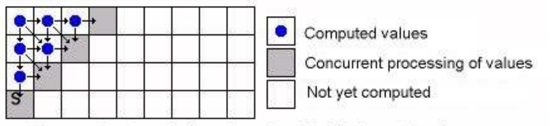

Figure 3. Paralelism in the Similarity Matrix

The basic processing element is depicted in figure 4 (a). The relative position of the neighbor values is indicated for the computation of value *w*. Note that *y* is the previous outcome of the same cell and is stored as the upper value. *z* was computed by the left neighbor in the previous step and is stored in an internal register for computing *w*. *x* was stored in the left neighbor as the upper value in the previous step, and is transferred at the same time than *z* to the cell computing *w*. Figure 4 (b) gives an idea of the processing steps. The three processing elements store "G C T" subsequence. At time *t* they are computing the values of the first dashed diagonal and at time *t+1* they are computing the values of the second dashed diagonal. For simplicity of representation, the left value was stored with the base being processed (as 5 in C5). This illustrates the data flow in the systolic array. Each cell produces, beyond the comparison value, a 3 bit relative pointer that indicates from where the alignment that produced that result came from. This information is used by the host where the FPGA is connected to recover the alignments for the similarity matrix.

4.2 Application of Dynamic Reconfiguration

Dynamic reconfiguration is useful for practical applications over molecular data. Once the systolic array proposed in the previous section is reconfigured for LSA, *detection* of the end positions of high scored alignments between two sequences is possible without writing out all the components of the dynamic matrix. For real molecular data this is necessary because the huge length of the usually treated sequences and the space complexity of the algorithm, which is in *O(mn)* for sequences of length *m* and *n.* For example, a practical solution for *constructing* the alignments of interest between two sequences *s* and *t,* consists in alternating the execution of the following two phases of reconfiguration and execution:

- Reconfigure the systolic array for executing LSA moving the sequence *t* from left to right, without constructing the dynamic matrix, and maintain only the current scores in each diagonal of the table and selecting the *good* ones.
- Once a *good* score is selected, say finishing at positions *i* and *j* of the sequences *s* and *t respectively,* the systolic array is reconfigured to execute the GSA operation in reverse order.

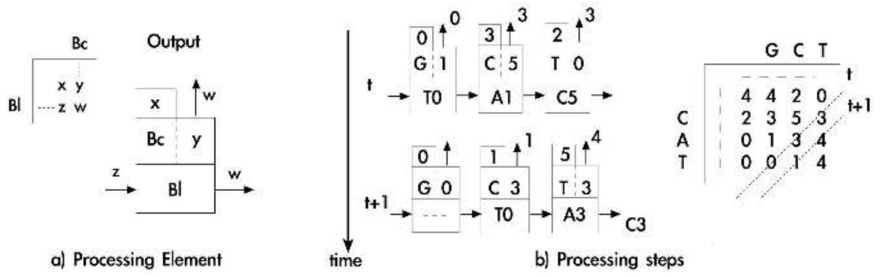

Figure 4. The processing element (a) and two steps of the flow of computation (b).

5 Implementation

The hardware implementation has limited size. Since biological sequences may have thousands of elements, often it will not match the size of the array. In this case, sequence partitioning is done by software. The architecture modeled in ELAN was refined to a structural VHDL description and synthesized and simulated in Altera Quartus II design environment.

5.1 Design in VHDL

The basic architecture of a systolic node to compute the dynamic algorithms includes registers to hold the neighbor values and a reference sequence character, adders to compute the cost and comparators to check for equality. It runs synchronously, performing one comparison for each clock cycle. The node `netlist` is presented in the Figure 5. It should be noted that the accumulated sequence matching cost grows from left to right, such that the number of bits needed to store the cost can be different for each node. The VHDL description of the nodes keep these values generic, such that they are defined when instantiating the cells.

The circuit for one node is simple. It compares the running base got from a data base with the base stored in the cell and produces this way the diagonal cost, which is the accumulated cost coming from column *j-1* and line *i-1* added to +1 or –1, as described in the previous section.

5.2 Simulation and Results

To verify the systolic array implementation simulations were performed in Quartus II environment. We show next an small example for illustration purposes. Figure 6 presents the similarity matrix obtained for the comparison of two pair of sequences, CATAG and ATAGC and CATAG and CATGA, using the LSA algorithm. This algorithm looks for the best matching between subsequences of the strings. The main difference with respect to the global matching is that it does not accumulate negative values, allowing local matches along the sequences. The arrows in the figure indicate several alternative matches and the encircled elements are those that need to be stored in a sparse matrix to recover the sequences, which is done by software.

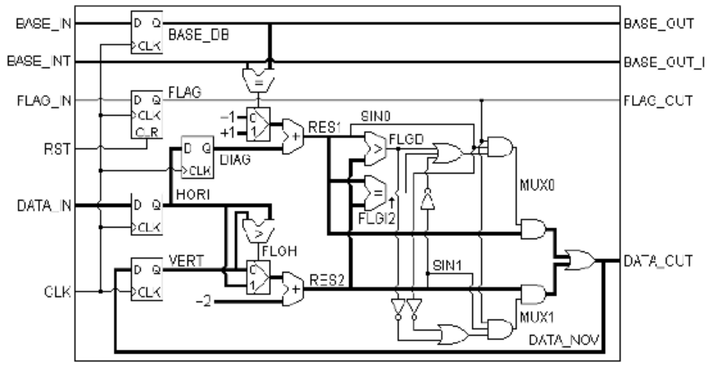

Figure 5. Systolic node structure.

The simulation of the comparison between CATAG and ATAGC sequences is shown in the Figure 7. The symbols are coded as follows: A = 00, T = 01, C = 10 and G = 11. Signal VAL shows the best score obtained in the comparison, which was 4 in this example. Signals MEM2, MEM3, MEM4 and MEM5 are the data outputs of the systolic array. CONTA is an auxiliary signal provided to help building the similarity matrix from the data produced by the array. To draw a profile of synthesis results, several arrays were generated and synthesized. The results are summarized in figure 8. The increasing curve (y-axis to the left) shows the number of logic elements required to implement arrays of varying sizes, indicated on the x-axis. For a systolic array of 50 nodes it was required around 4500 logic elements on an APEX device. Since the circuit size grows almost linearly, we can estimate the size of the vector for devices based on the same logic element. For instance, an APEX EP20K400, with 16.640 logic elements could hold 180 nodes. Last generation devices, like Stratix II or Virtex II could hold thousands of nodes. The decreasing curve (y-axis to the right) corresponds to the frequency attained by the array. It is interesting to note that initially the frequency decays strongly with the size of the vector and then stabilize for vectors with 30 or more cells, around 56 Mhz.

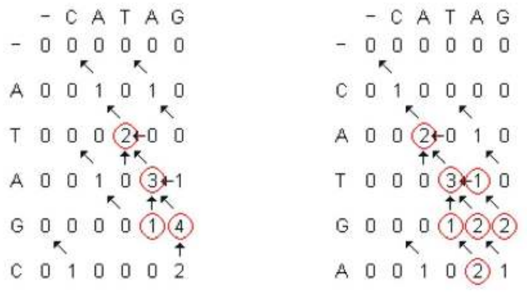

Figure 6. LSA for sequences CATAG , ATAGC and CATGA

The time needed to compute a sequence comparison is given by the time the running sequence takes to traverse the systolic array. Thus, if the reference sequence has n elements and the data base sequence has m elements, then we need $n + m$ clock cycles to compute the comparison. To get a rough estimation of the gain with respect to a software solution, suppose that the frequency keeps around 50 Mhz for larger devices. Considering that we can chain FPGA in order to implement larger sequences, an estimation of the speed up provided by the systolic vector compared to the time required by a cluster of workstations obtained from [14] is given in table 1. The real speed up should be less than this value because it does not include the communication among FPGAs. Even in this case, the gain in speed is of several orders of magnitude. The values do not take into account the time to recover the alignments.

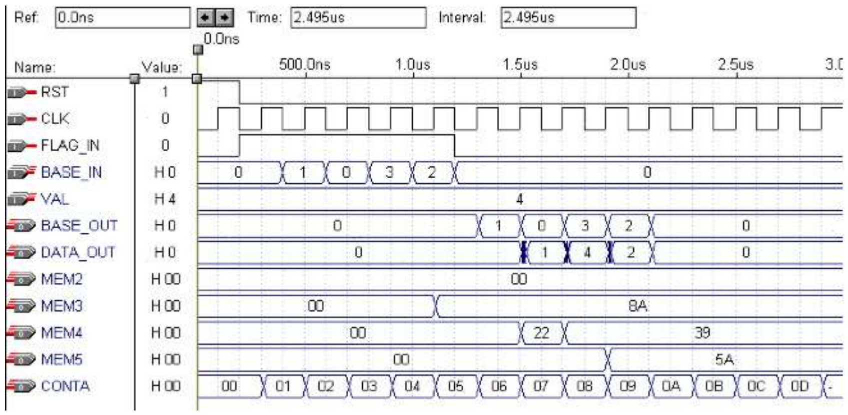

Figure 7. LSA simulation of sequences CATAG and ATAGC.

Table 1. Processing time for a cluster of workstations and the systolic array.

Seq. Size	1 proc.	2 Proc.	4 Proc.	8 Proc.	Systolic Array
15K x 15 K	296s	283,18s	202,18s	181,29s	0,000614s
50K x 50K	3461s	2884,15s	1669,53s	1107,02s	0,002048s
80K x 80K	7967s	6094,19s	3370,40s	2162,82s	0,003277s

6 Conclusions and Future Work

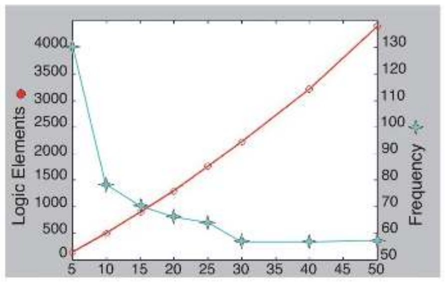

Figure 8. Size and Frequency (Mhz) x vector length

The systolic array derived in this work can speed up string comparison (string pattern matching and sequence alignment) algorithms by software in several orders of magnitude. Previous works in the literature on string comparison in hardware focused on sequence comparison for biological problems using different approaches but, if implemented with current technologies, should provide similar speed up. The main contribution of this work is the computation of the *sequence alignment* instead of *sequence comparison*. The systolic architecture generates alignments through relative one bit pointers that allow the host to recover the proper alignments by software in a post processing step. It should be noted that by using rewriting-logic in this work we could easily extend the range of problems covered by our systolic architecture through design exploration and simulation. The reconfigurability of the systolic architecture plays a fundamental role here, allowing the designer to switch from one algorithm to another. Current work address the integration of this architecture in cluster of reconfigurable workstations, where the FPGA's work in parallel and a distributed operating system controls the process.

References

1. Altera® Corporation. Quartus II User Guide. Available at http://www.altera.com. Accessed in 2004

2. M. Ayala-Rincón et al. *Modeling a Reconfigurable System for Computing the FFT in Place via Rewriting-Logic*. In IEEE CS Proc. SBCCI'03, pp 205-210, IEEE CS, 2003

3. M. Ayala-Rincón et al. *Modeling and Prototyping Dynamically Reconfigurable Systems for Efficient Comp. of Dynamic Programming Methods by Rewriting-Logic*. ACM Proc. SBCCI'04, pp 248-253, 2004

4. J. Becker and R. W. Hartenstein, *Configware and morphware going mainstream*, Journal of Systems Architecture 49:127-142, 2003

5. M. Gokhale. *Splash: A reconfigurable linear logic array*. Proceedings of 1990 International Conference on Parallel Processing (1990) 526–532

6. S.A. Guccione, E. Keller. *Gene matching using jbits*. Xilinx, Inc. (2002)

7. J. D. Hirschberg, R. Hughey, K. Karplus: *Krestel: A programmable array for sequence analysis*. In: Proc. Int. Conf. Application-Specific Systems, Architectures and Processors, IEEE CS (1996) 25 :34

8. D.T. Hoang. *A systolic array for the sequence alignment problem*. Technical Report CS-92-22, Brown University, Providence, RI (1992)

9. Information Sciences Institute - East http:// www.east.isi.edu/projects/SLAAC/:Slaac project. (2002)

10. H.T. Kung, C. E. Leiserson. *Systolic Arrays for VLSI Sparse Matrix Proc*. Society for Industrial and Applied Mathematics, pages 256-282, 1978-1979.

11. E. S. Lander et al.: *Initial sequencing and analysis of the human genome*. Nature 409 (2001) 860–921

12. D. Lavenier, *Dedicated Hardware for Biological Sequence Comparison*. The Journal of Universal Computer Science 2(2):77-86,1996.

13. R.J. Lipton, D. Lopresti: *A systolic array for rapid string comparison*. In: Chapel Hill Conference on VLSI. (1985) 363–376

14. R. C. F. Melo, et al. *Comparing Two Long Biological Sequences Using a DSM System*. Proc. Euro-Par 2003 - Parallel Processing, LNCS, Vol. 2790, pages 517-524, 2003.

15 Paracel Inc. http://www.paracel.com/products/pdfs/gm2_datasheet.pdf. *The Genematcher2 System Datasheet*. (2002) 24. Paracel, Inc http://www.paracel.com/. (2001)

16. K. Puttegowda, W. Worek, N. Pappas, A. Dandapani, P. Athanas. *A run-time reconfigurable system for gene-sequence searching*. Proceedings of the International VLSI Design Conference (2003)

17. T. Rognes and E. Seeberg, *Six-fold speed-up of Smith-Waterman Sequence Database Searches Using Parallel Processing on Common Microprocessors*. Bioinformatics 16(8):699-706, 2000. See also Sencel Bioinformatics `http://www.sencel.com`.

18. T. F. Smith and M. S. Waterman, *Identification of Common Molecular Subsequences*. Journal of Molecular Biology, 147:195-197, 1981.

19 TimeLogic Corp. http://www.timelogic.com. (2002)

20. J. C. Venter et al.: *The sequence of the human genome*. Science 291 (2001) 1304–1351

21. C. T. White et al. Bioscan: *A VLSI system based for biosequence analysis*, IEEE CS (1991) 504:509

22. Xilinx Inc. http://www.xilinx.com. Acessed in 2004.

23. B. H. W. Yang. *A parallel implementation of Smith-Waterman sequence comparison algorithm*. Technical Report ID: 4469409, Stanford (2002)

24. Y. Yamaguchi et al. *High Speed Homology Search Using Run-Time Reconfiguration*. 12th Int. Conference on Field-Programmable Logic and Applications, Springer-Verlag LNCS 2438:281-291.

25. C.W. Yu, K.H. Kwong, K.H. Lee and P.H.W. Leong, *A Smith-Waterman Systolic Cell,* 13th Int. Conference on Field-Programmable Logic and Applications, Springer-Verlag LNCS 2778:375-384, 2003.