A Smith-Waterman Systolic Cell

C.W. Yu, K.H. Kwong, K.H. Lee and P.H.W. Leong

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, New Territories
Hong Kong SAR
{cwyu1,khkwong,khlee,phwl}@cse.cuhk.edu.hk

Abstract. With an aim to understand the information encoded by DNA sequences, databases containing large amount of DNA sequence information are frequently compared and searched for matching or near-matching patterns. This kind of similarity calculation is known as sequence alignment. To date, the most popular algorithms for this operation are heuristic approaches such as BLAST and FASTA which give high speed but low sensitivity, i.e. significant matches may be missed by the searches. Another algorithm, the Smith-Waterman algorithm, is a more computationally expensive algorithm but achieves higher sensitivity. In this paper, an improved systolic processing element cell for implementing the Smith-Waterman on a Xilinx Virtex FPGA is presented.

1 Introduction

Bioinformatics is becoming an increasingly important field of research. With the ability to rapidly sequence DNA information, biologists have the tools to, among other things, study the structure and function of DNA; study evolutionary trends; and correlate DNA information with disease. For example, two genes were identified to be involved in the origins of breast cancer in 1994 [1]. Such research is only possible through the help of high speed sequence comparison.

All the cells of an organism consist of some kind of genetic information. They are carried by a chemical known as the deoxyribonucleic acid (DNA) in the nucleus of the cell. DNA is a very large molecule and nucleotide is the basic unit of this type of molecule. There are 4 kinds of nucleotides and each have different bases, namely adenine, cytosine, guanine and thymine. Their abbreviated forms are "A", "C", "G" and "T" respectively. In this paper, the sequence is referred to as a string, and the bases form the alphabet for the string.

It is possible to deduce the original sequencing in DNA which codes for a particular amino acid. By finding the similarity between a number of "amino-acid producing" DNA sequences and a genuine DNA sequence of an individual, one can identify the protein encoded by the DNA sequence of the individual. In addition, if biologists succeed in finding the similarity between DNA sequences of two different species, they can understand the evolutionary trend between them. Another important usage is that the relation between disease and inheritance can

also be studied. This is done by aligning specific DNA sequences of individuals with disease to those of normal people. If correlations can be found which can be used to identify those susceptable to certain diseases, new drugs may be made or better techniques invented to treat the disease. There are many other applications of bioinformatics and this field is expanding at an extremely fast rate.

A human genome contains approximately 3 billion DNA base pairs. In order to discover which amino acids are produced by each part of a DNA sequence, it is necessary to find the similarity between two sequences. This is done by finding the minimum string edit distance between the two sequences and the process is known as sequence alignment.

There are many algorithms for doing sequence alignment. The most commonly used ones are FASTA [2] and BLAST [3]. BLAST and FASTA are fast algorithms which prune the search involved in a sequence alignment using heuristic methods. The Smith-Waterman algorithm [4] is an optimal method for homology searches and sequence alignment in genetic databases and makes all pairwise comparisons between the two strings. It achieves high sensitivity as all the matched and near-matched pairs are detected, however, the computation time required strongly limits its use.

Sencel Bioinformatics [5] compared the sensitivity and selectivity of various searching methods. The sensitivity was measured by the coverage, which is the fraction of correctly identified homologues (true positives). The coverage indicates what fraction of structurally similar proteins one may expect to identify based on sequence alone. Their experiments show that for a coverage around 0.18, the errors per query of BLAST and FASTA are about two times that of the Smith-Waterman Algorithm.

Many previous ASIC and FPGA implementations of the Smith-Waterman algorithm have been proposed and some are reviewed in Section 4. To date, the highest performance chip [6] and system level [7] performance figures have been achieved using a runtime reconfigurable implementation which directly writes one of the strings into the FPGA's bitstream.

In this work, an FPGA-based implementation of the Smith-Waterman algorithm is presented. The main contribution of this work is a new 3 Xilinx Virtex slice Smith-Waterman cell which is able to achieve the same density and performance as an earlier reported cell [6], without the need to perform runtime reconfiguration. This has advantages in that the design is less FPGA device specific and thus can be used for non-Xilinx FPGA devices as well as ASICs. Whereas the runtime reconfigurable design requires JBits, a Xilinx specific API for runtime reconfiguration, the design presented in this paper was written in standard VHDL. Moreover, in the proposed design, both strings being compared can be changed rapidly as compared to a runtime reconfigurable system in which the bitstream must be generated and downloaded, which is typically a very slow process since a large bitstream must be manipulated and downloaded via a slow interface. This reconfiguration process may become a bottleneck, particularly for small databases. Furthermore, other applications may require both strings

		A	**C**	**G**
	0	1	2	3
A	1	?	?	?
T	2	?	?	?
C	3	?	?	?

(a) Initial table.

		S_i		
	a	b		
T_i	c	d		

(b) Equation 1 values.

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

(c) Final table.

Fig. 1. Figure showing the progress of the Smith-Waterman algorithm, when the string "ACG" is compared with "ATC".

to change quickly. The design was implemented and verified using Pilchard [8], a memory-slot based reconfigurable computing environment.

2 The Smith-Waterman Algorithm

The Smith-Waterman Algorithm is a dynamic programming technique which utilizes a 2D table. As an example of its application, suppose that one wishes to compare sequence S ("ACG") with sequence T ("ATC"). The intermediate values a, b and c (shown in Fig. 1(b)) are then used to compute d according to the following forumla:

$$d = min \begin{cases} \begin{cases} a & if \quad S_i = T_j \\ a + sub & if \quad S_i \neq T_j \end{cases} \\ b + ins \\ c + del \end{cases} \tag{1}$$

If the strings being compared are the same, the value a is used for d. Otherwise, the minimum of a plus some substitution penalty sub, b plus some insertion penalty ins and c plus some deletion penalty del is used for d. Data dependencies mean that entries d in the table can only be calculated if the corresponding a, b, c values are already known and so the computation of the table spreads out from the origin as illustrated in Fig. 2. As an example, the first entry that can be computed is that for "AA" in Fig. 1(a). Since $S_i = T - i = `A'$, according to Equation 1, $d = a$ and so the entry is set to 0. In order to complete the table, the template of Fig. 1(b) is moved around the table constrained by the dependencies indicated by Fig. 2.

The substitution, insertion and deletion penalties can be adjusted for different comparison requirements. If the presence of redundant characters is relatively less acceptable than just a difference in characters, the insertion and deletion penalties can be set to a higher value than the substitution penalty. In the

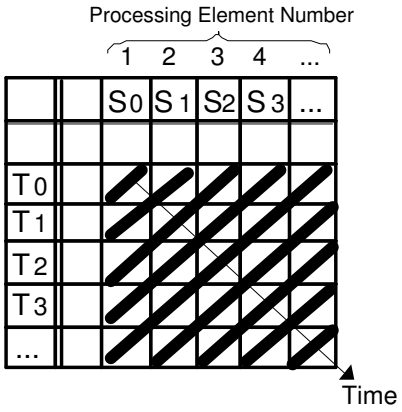

Fig. 2. Data dependencies in the alignment table. Thick lines show entries which can be computed in parallel and the time axis is arranged diagonally.

alignment system presented, the insertion and deletion penalties were fixed at 1 and the substitution penalty set to 2, as is typical in many applications.

If S and T are m and n in length respectively, then the time complexity of a serial implementation of the Smith-Waterman algorithm is $O(mn)$. After all the squares have been processed, the result of Fig. 1(c) is obtained. In a parallel implementation, the positive slope diagonal entries of Fig. 2 can be computed simultaneously. The final edit distance between the two strings appears in the bottom right table entry.

3 FPGA Implementation

In 1985, Lipton and Lopresti observed that the values of b and c in Equation 1 are restricted to $a \pm 1$ and the equation can be simplified to obtain [9]:

$$d = \begin{cases} a & if\ ((b\ or\ c)\ =\ a-1)\ or\ (S_i = T_j) \\ a+2 & if\ ((b\ and\ c)\ =\ a+1)\ and\ (S_i \neq T_j) \end{cases} \tag{2}$$

Using Equation 2, it can be seen that the data elements b, c and d only have two possible values. Therefore, the number of data bits used for the representation of b, c and d can be reduced to 1 bit. Furthermore, two bits can be used to represent the four possible values of the alphabet.

The processing element (PE) shown in Fig. 3 was used to implement Equation 2. A number of PEs are then connected in a linear systolic array to process diagonal elements in the table in parallel. As shown in Fig. 2, PEs are arranged horizontally and are responsible for its corresponding column. In the description that follows, the sequence that changes infrequently is S and the sequences from

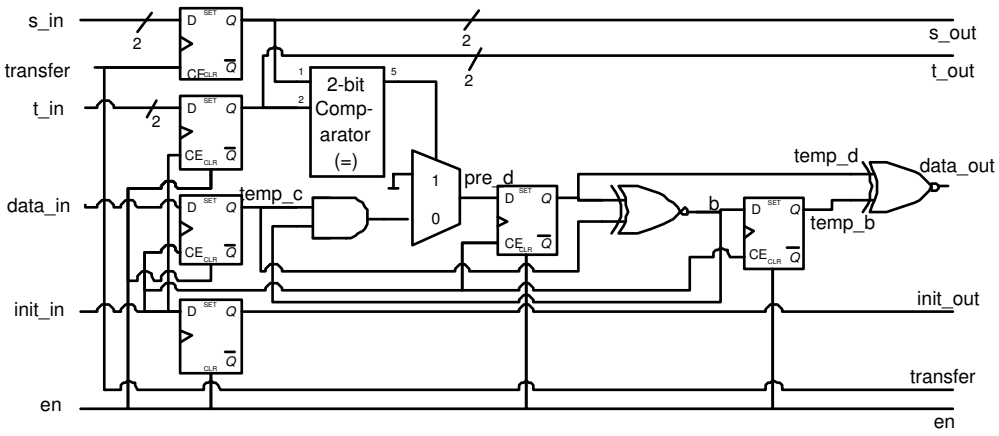

Fig. 3. The Smith-Waterman processing element (PE). Boxes represent D-type flip-flops.

the database are T. In each PE, two latches are used to store a character S_i. These characters are shifted and distributed to every processing element before the actual comparison process beings. The base pairs of T are passed through the array during the comparison process, during which the d of Equation 2 is also computed.

In order to calculate d, inputs a, b and c should be available. In the actual implementation, the new values b, c and d are calculated during the comparison of the characters as follows:

1. data_in is the new value of c and it is stored in a flip-flop. At the same time, this new c value and the previous d value (from a flip-flop) determines the new b value ($b = temp\_c$ XNOR $temp\_d$)
2. The new b value is stored in a flip-flop. At the same time, the output of a 2-to-1 MUX is then selected depending on whether $S_i = T_i$. The output of the MUX (a '0' value or (b AND $temp\_c$)) becomes the new d value. This new d value is stored in a flip-flop.
3. Values of b and d determine the data output of the PE ($data\_out = temp\_b$ XNOR $temp\_d$). The data output from this PE is connected to the next PE as its data input (its new c value)

When the $transfer$ signal is high, the sequence S is shifted through the PEs. When the en signal is high, all the flip-flops (except the two which store the string S) are reset to their initial values. The $init$ signal is high when new signals from the preceeding PE are input and the new value of d calculated. When the $init$ signal is low, the data in all the flip-flops are unchanged.

Each PE used 8 flip-flops as storage elements and 4 LUTs to implement the combinational logic. Thus the design occupied 4 Xilinx Virtex slices. Guccione

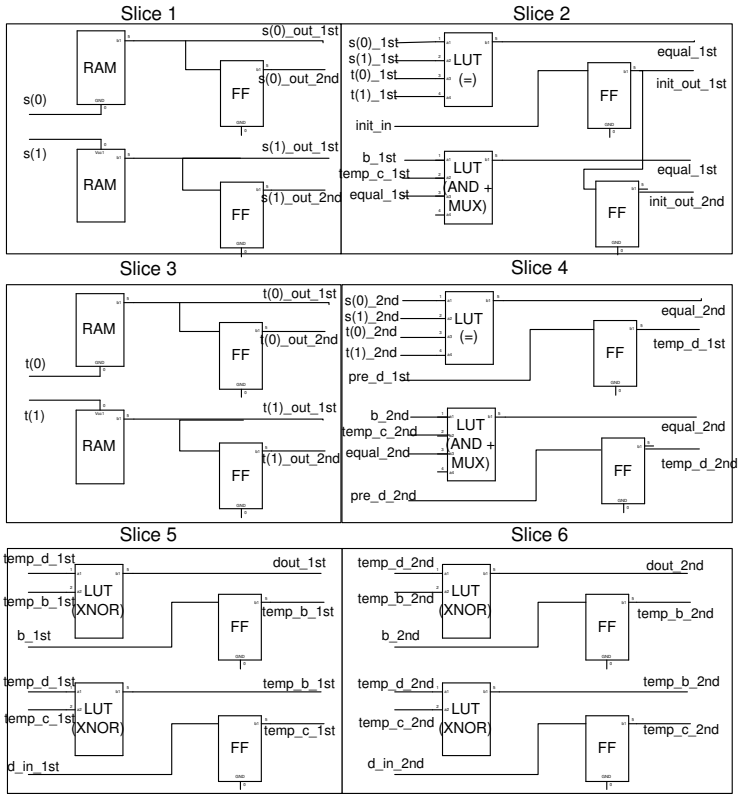

Fig. 4. Two processing elements mapped to 6 Virtex slices.

and Keller [6] used runtime reconfiguration to write one of the sequences into the bitstream, saving 2 flip-flops and implementing a PE in 3 slices. In the proposed approach, two otherwise unused LUTs were configured as shift register elements using the Xilinx distributed RAM feature [10]. Thus the design occupies 3 Xilinx Virtex slices per PE, without requiring runtime reconfiguration to change S. In the actual implemention, 2 PEs were implemented in 6 slices since sharing of resources between adjacent PEs was necessary in the actual implementation.

Fig. 4 shows the mapping of the PEs to slices. All the signals ending with "_1st" were used in PE Number 1, and signals ending with "_2nd" were used for PE2. The purpose of each signal can be understood by referring back to Fig. 3. It was necessary to connect the output of the RAM-based flip-flops directly to a flip-flop (FF in the diagram) in the same logic cell (LC) since internal LC connections do not permit them to be used independently (i.e. it was not possible to avoid connecting the output of the RAM and the input of the FF). Thus, Slice 1 was configured as a 2 stage shift register for consecutive values of S_i and Slice 3 was used for two consecutive values of T_i.

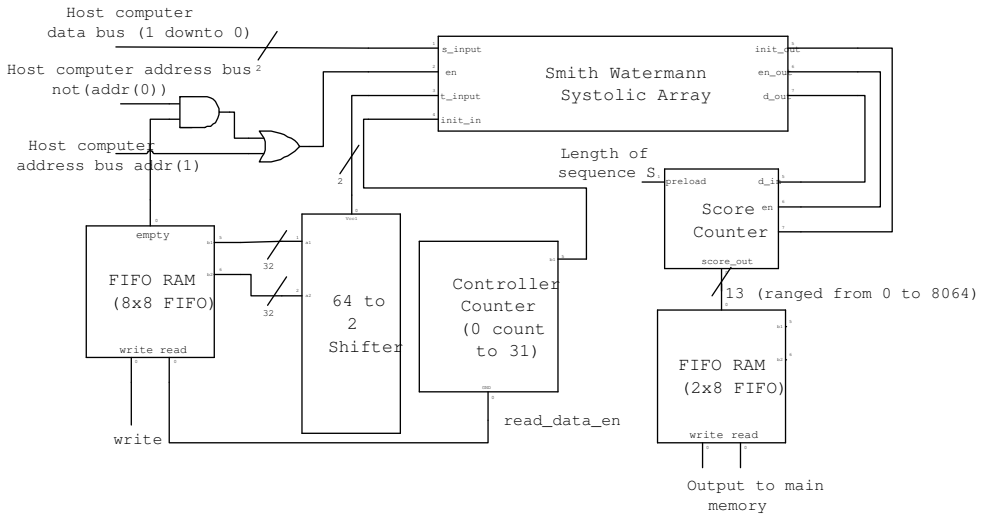

Fig. 5. System data path.

Fig. 5 shows the overall system data path. Since the T sequences are shifted continuously, the system used a FIFO constructed from Block RAM to buffer the sequence data supplied by a host computer. This improves throughput of the system since a large number of string comparisons can be completed before all of their scores are read from the controller, reducing the amount of idle time in the systolic array. The input and output data width of the FIFO RAM were both 64 bits. The wide input data width helped to improve IO bandwidth from the host computer to the FIFO RAM. A 64-to-2 shifter and a controller counter were used for reducing the output data width of the FIFO RAM from 64 bits to 2 bits, so as to allow data to be fed into the systolic array.

The Score Counter computes the edit distance by accumulating results calculated in the last PE of the systolic array. The output of the last PE is actually the d value in the squares of the rightmost column of the matrix, and differences in values of consecutive squares in the rightmost column must be 1. The $data_{out}$ of the last PE is '0' when $d = b$ - 1 , and the output '1' when $d = b +$ 1. Therefore, a Shift Counter was initialized to the length of the sequence S. It was decremented if the output value is '0', otherwise it was incremented. After the entire string T is passed through the systolic array, the counter contains the final string comparison score.

4 Results

The design was synthesized from VHDL using the Xilinx Foundation 5.1i software tools and implemented on Pilchard, a reconfigurable computing platform

Fig. 6. Photograph of the Pilchard board.

[8] (Fig. 6). The Pilchard platform uses a Xilinx Virtex XCV1000E-6 FPGA (which has 12288 slices) and uses a SDRAM memory bus interface instead of the conventional PCI bus to reduce latency.

A total of 4,032 PEs were places on an XCV1000E-6 device (this number was chosen for floorplanning reasons). As reported by the Xilinx timing analyzer, the maximum frequency was 202 MHz.

A number of commercial and research implementations of the Smith Waterman algorithm have been reported and their performance are summarized in Table 1. Examples are Splash [11], Splash 2 [12], SAMBA [13], Paracel [14], Celera [15], JBits from Xilinx [6], and the HokieGene Bioinformatics Project [7]. The performance measure of cell updates per second (CUPS) is widely used in the literature and hence adopted for our results.

Splash contains 746 PEs in a Xilinx XC3090 FPGA performing the Smith-Waterman Algorithm. Splash 2's hardware was different from Splash, which used XC4010 FPGAs with a total of 248 PEs. SAMBA [13] incorporated 16 Xilinx XC3090 FPGAs with 128 PEs altogether dedicated to the comparison of biological sequences.

ASIC and software implementations have also been reported. Paracel, Inc. used a custom ASIC approach to do the sequence alignment. Their system used 144 identical custom ASIC devices, each containing approximately 192 processing elements. Celera Genomics Inc. reported a software based system using an 800 node Compaq Alpha cluster.

Both the JBits and the HokieGene Bioinformatics Project were the latest reported sequence alignment systems using the Smith-Waterman Algorithm and use the same PE design. JBits reported performance for two different FPGA chips, the XCV1000-6 and the XC2V6000-5. The HokieGene Bioinformatics Project used an XCV6000-4. As can be seen from the table, the performance of the proposed design is similar to the JBits design on the same size FPGA (a XCV1000-6), and the JBits and HokieGene implementations on an XCV6000 gain performance by fitting more PEs on a chip, and our performance on the same chip would be similar.

The implementation was successfully verified using the Pilchard platform whcih provides a 133 MHz, 64-bit wide memory mapped bus to the FPGA. The processing elements and all other logic of the implementation operate from

System	Number of Chips	PEs per chip	System Performance (CUPS)	Device Performance (CUPS)	Run-time reconfiguration requirement
Splash(XC3090)	32	8	370 M	11 M	No
Splash 2(XC4010)	16	14	43 B	2,687 M	No
SAMBA(XC3090)	32	4	1,280 M	80 M	No
Paracel(ASIC)	144	192	276 B	1,900 M	N/A
Celera (software implementation)	800	1	250 B	312 M	N/A
JBits (XCV1000-6)	1	4,000	757 B	757 B	Yes
JBits (XC2V6000-5)	1	11,000	3,225 B	3,225 B	Yes
HokieGene (XC2V6000-4)	1	7000	1,260 B	1,260 B	Yes
This implementation (XCV1000-6)	1	4,032	742 B	742 B	No
This implementation (XCV1000E-6)	1	4,032	814 B	814 B	No

Table 1. Performance and hardware size comparison of previous implementations (processor core not including system overheads). Device performance is measured in cell updates per second (CUPS).

the same 133 MHz clock. The interface logic occupied 3% of the Virtex device. The working design was used mainly for verification performance and had a disappointing performance of approximately 136 B CUPS, limited by the simple polling based host interface used. A high speed interface which performs more buffering and is able to cause the memory system to perform block transfers between the host and Pilchard is under development.

5 Conclusion

A technique, commonly used in VLSI layout, in which two processing elements are merged into a compact cell was used to develop a Smith-Waterman systolic processing element design which computes the edit distance between two strings. This cell occupies 3 Xilinx Virtex slices and allows both strings to be loaded into the system without runtime reconfiguration. Using this cell, 4032 PEs can fit on a Xilinx XCV1000E-6, operate at 202 MHz and achieve a device performance of 814 B CUPS.

References

1. Y. Miki, et. al. A Strong Candidate for the Breast and Ovarian Cancer Susceptibility Gene, BRCA1. *Science*, 266:66–71, 1994.
2. European Bioinformatics Institute Home Page, FASTA searching program, 2003. http://www.ebi.ac.uk/fasta33/.
3. National Center for Biotechnology Information. NCBI BLAST home page, 2003. http://www.ncbi.nlm.nih.gov/blast.
4. T. F. Smith and M. S. Watermann. Identification of common molecular subsequence. *Journal of Molecular Biology*, 147:196–197, 1981.
5. Sencel's search software, 2003. http://www.sencel.com.
6. Steven A. Guccione and Eric Keller. Gene matching using JBits, 2002. http://www.ccm.ece.vt.edu/hokiegene/papers/GeneMatching.pdf.
7. K. Puttegowda, W. Worek, N. Pappas, A. Danapani and P. Athanas. A run-time reconfigurable system for gene-sequence searching. In *Proceedings of the International VLSI Design Conference*, page (to appear), Jan 2003.
8. P. Leong , M. Leong , O. Cheung , T. Tung , C. Kwok , M. Wong and K. H. Lee. Pilchard - a reconfigurable computing platform with memory slot interface. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, page (to appear), April 2001.
9. Richard J.Lipton and Daniel Lopresti. A systolic array for rapid string comparison. In *Proceedings of the Chapel Hill Conference on VLSI*, pages 363–376, 1985.
10. Xilinx. *The programmable logic data book*, 2003.
11. D. T. Hoang. A systolic array for the sequence alignment problem. *Brown University, Providence, RI, Technical Report*, pages CS–92–22, 1992.
12. D. T. Hoang. Searching genetic databases on splash 2. In *Proceedings 1993 IEEE Workshop on Field-Programmable Custom Computing Machines*, pages 185–192, 1993.
13. Dominique Lavenier. SAMBA: Systolic Accelerators for Molecular Biological Applications, March 1996.
14. Paracel, inc, 2003. http://www.paracel.com.
15. Celera genomics, inc, 2003. http://www.celera.com.