Sindh Univ. Res. Jour. (Sci. Ser.) Vol.44 (3) 449-452 (2012)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

An Optimized and Parallel Hardware Design for Dot Plot Algorithm

L. HASAN, H. ZAFAR, A. KHATTAK*, T.ALI**, I. DIN***

Department of Computer Systems Engineering, University of Engineering and Technology, Peshawar, Pakistan.

Received 18th January 2012 and Revised 11th June 2012

Abstract The dot plot algorithm is a useful computational tool for comparative genomics that offers an effective direct method for comparing genetic sequences. It creates a pair wise comparison between two DNA or protein sequences and renders the results as a dot matrix. Implementation in hardware reduces the *O(MN)* complexity of the dot plot to *O(M+N)*. In this paper, we develop an optimized and parallel hardware design approach that can be implemented on hardware platforms like FPGAs. The approach brings the complexity of the dot plot algorithm further down to *O(M)*.

Keywords—Bioinformatics; Comparative genomics; Dot plot; Sequence comparison; FPGAs; Computational complexity.

1. INTRODUCTION

An effective and direct method of comparing two sequences is a visual approach known as a dot plot (Attwood and Smith, 2001) (Gibbs and Mclntyre, 1970) (Mueller *et al.*, 2006). A dot matrix for two sequences, i.e. Sequence 1 and Sequence 2, is simply a grid with the presence of a point at position $p = (i, j)$, if the k-tuple beginning at the i^{th} position of Sequence 1 and the j^{th} position of Sequence 2 coincide (Mueller *et al.*, 2006).

The sequences to be compared are arranged along the margins of a matrix. At every point in the matrix where the two sequences are identical, a dot is placed (i.e. at the intersection of every row and column that has the same letter in both sequences). A diagonal stretch of dots indicates regions where the two sequences are similar. Done in this fashion, a dot plot as shown in **(Table 1)** is obtained (for clarity, dots are marked as ×s in the table).

Table 1. Dot plot matrix

	A	**C**	**T**	**G**	**G**
A	×				
C		×			
T			×		
G				×	×
G				×	×

Applications of dot plot in bioinformatics and comparative genomics include finding similarity between two genomes, searching a protein in an organism and finding genetic relatedness between different nucleotides etc. Heuristics based approaches like FASTA (Pearson and Lipman, 1985) and BLAST (Altschul *et al.*, 1990) provide improved performance for similar applications, but at the cost of reduced accuracy. Parallel versions of some common bioinformatics algorithms exist (Buyukkur and Najjar, 2008) (Hasan *et al.*, 2008) (Hasan and Al-Ars, 2009). The authors of (Oliver *et al.*, 2005) presented a parallel implementation of the dynamic programming (Giegerich, 2000) based Smith-Waterman algorithm (Smith and Waterman, 1981) for finding optimal local alignments. The core of the Smith-Waterman algorithm is similar to the dot plot algorithm, but narrows the comparison space by only following paths that lead to good local alignments.

In general, the dot plot algorithm considers two DNA or protein sequences from bioinformatics databases like SWISS-PROT (Boeckmann *et al.*, 2003). The two sequences are called the *query sequence* (N_q) and the *subject or database sequence* (N_s), whose lengths can be different, but in the ideal case is fairly similar. We proceed by creating a rectangular matrix in which the characters of N_q are mapped along the *x*-axis, and those of N_s along the *y*-axis. Initially, the matrix is filled with zeros. Each of its cells, $x_i y_j$ (where i varies between 1 and the length of sequence N_q, and j varies between 1 and the length of sequence N_s), is considered in turn and is assigned a value indicating the level of similarity between the two residue positions (N_q and N_s). In the simplest scheme, all cells remain zero, unless $N_q = N_s$, in which case the element is assigned a value 1.

++Corresponding Author: L. HASAN laiqhasan@nwfpuet.edu.pk +92-91-9216590

*Department of Electrical Engineering, University of Engineering and Technology, Peshawar, Pakistan.

**Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Netherlands

***WIT lab, Department of Electronics Engineering, University of Incheon, South Korea

Such a matrix can be visualized quite simply for short sequences, for example by printing out the matrix in a particular font, as shown in **(Table 1)** or for longer sequences, by using an appropriate graphics program. The plot is characterized by some apparently random dots (noise) and a central diagonal line, where a high density of adjacent dots indicates the regions of greatest similarity between the two sequences. For full length sequences, a plot must be reduced in size in order to be able to visualize the complete comparison and in doing so, the ×s in the magnified section shown in **(Table 1)** are reduced to dots (hence the dot plot), which, at sufficiently low magnification, will ultimately merge into lines. In contrast with identical sequences, two similar sequences will be characterized by a broken diagonal (Attwood and Smith, 2001). To fill the entire dot matrix using the dot plot algorithm, the time complexity is the number of entries, or $O(MN)$, where M and N are the lengths of sequences N_q and N_s, respectively. The space complexity of the dot plot is also $O(MN)$, as the entire matrix needs to be stored in memory.

This *O(MN)* complexity can be reduced to *O(M+N)* by implementation in hardware. In this paper, we develop an optimized hardware design approach that reduces the complexity of dot plot algorithm further down to *O(M)*.

The remainder of the paper is organized as follows: Section 2 presents our hardware design approach for the dot plot cell. Section 3 provides a dot plot array design based on our approach. Section 4 gives a brief conclusion.

2. MATERIALS AND METHODS

Dot Plot Cell Design

For years, the quadratic running time for the dot plot algorithm was acceptable because most available sequences were short, but to make it applicable in the era of bioinformatics databases that grow exponentially, there is a serious need of speeding it up. The $O(MN)$ complexity of the dot plot can be easily reduced to $O(M + N)$ by implementing it in hardware. Here, we look beyond this reduction and try to develop a hardware design approach that will bring the complexity further down to $O(M)$.

(Fig. 1) shows the dot plot cell design for reducing the complexity to *O(M)*, where the comparator compares the two input sequences and produces a result based on the match or mismatch. The result is a 1 if there is a match, otherwise 0.

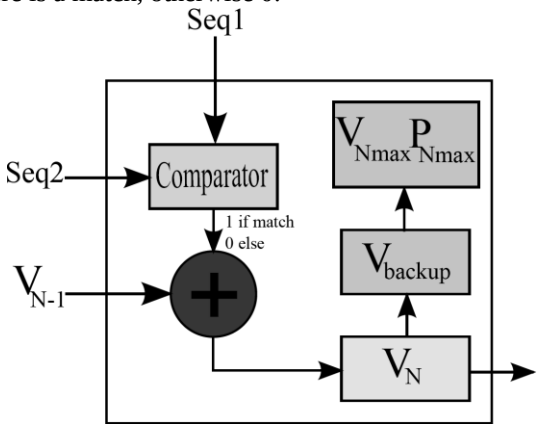

Fig. 1. Dot plot cell design

The adder adds the result of the comparator with the value from the previous cell. The result of the adder is stored in a register used for storing the current value of the cell. The current value is denoted by V_N, where V stands for value and N represents the number of the cell, such that $V_N = V_{N-1} + 1$.

There is another register called the backup register, which keeps a backup of the cell's maximum value. V_N is reset to 0 if there is a mismatch, but before resetting, its value is compared with the value of the backup register and if $V_N > V_{backup}$, then V_{backup} is updated with the new V_N value. The $V_{Nmax}P_{Nmax}$ block keeps track of the maximum value and its index.

3. DISCUSSION

Dot Plot Array Design

(Fig. 2) shows a 4-element dot plot array constructed using the cell design shown in **(Fig. 1)**. The operation of the array is explained with the help of an example given in **(Table 2)**.

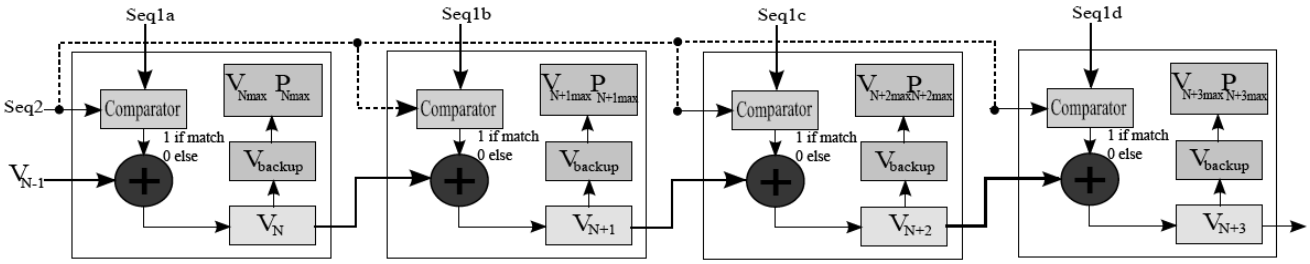

Fig. 2. 4-element dot plot array design

(Table 2(a)) gives an example, where two 4-character DNA sequences are compared using our dot plot implementation approach. Where,

Sequence 1: G C T A
and
Sequence 2: A C T G

Table 2(a). Example of the process

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

Table 2(b). Result

C	**T**
C	**T**

The bold digits in the table represent the values in the backup registers, whereas the other digits represent the V_N values. Sequence 1 is fixed along the array, such that each character is an input to a different *Processing Element (PE)*, whereas Sequence 2 is propagated through the array character by character.

During the first clock cycle letter **A** is passed through the array and compared with the corresponding Sequence 1 letters. The resultant V_N and V_{backup} values are recorded in the first row of the table. The same process is repeated for all characters in Sequence 2 and the resultant values are recorded in the table. Thus if the length of Sequence 2 is *M*, then the complexity of the entire process will be *O(M)*. The last row in the table gives the final V_N and V_{backup} values, whereas the position or index of the maximum backup value is given by the $V_{Nmax}P_{Nmax}$ block. The maximum backup value is traced back the number of steps equal to the maximum value itself, considering the current cell as the first step. This trace back is a constant time operation and does not affect the overall complexity. So the overall complexity stays the same as *O(M)*. The result of the process is given in **(Table 2(b))**.

4. CONCLUSION

This paper presented an optimized and parallel hardware design approach for dot plot algorithm used in bioinformatics and comparative genomics. The optimized design approach reduced the complexity of the dot plot algorithm from *O(MN)* to *O(M)* i.e. from quadratic to linear. The presented approach can be efficiently implemented on hardware platforms like FPGAs for achieving high performance genetic sequence comparison.

REFERENCES:

Altschul, S.F., Gish, W. Miller, W. Myers, and Lipman, D.J. (1990) A Basic Local Alignment Search Tool. Journal of Molecular Biology, **(215):** 403–410.

Attwood T.K. and D.J.P.Smith (2001) Introduction to Bioinformatics. Pearson Education, USA.

Boeckmann B., A. Bairoch R. Apweiler M.C. Blatter A. Estreicher E. Gasteiger M.J. Martin K. Michoud C. O'Donovan and I. Phan (2003) The SWISS-PROT Protein Knowledge Base and its Supplement TrEMBL. Nucleic Acids Research, **(31):** 365–370.

Buyukkur A.B. and W. Najjar (2008) Compiler Generated Systolic Arrays for Wavefront Algorithm Acceleration on FPGAs. International Conference on Field Programmable Logic and Applications (FPL08), Heidelberg, Germany.

Gibbs A.J. and G.A. Mclntyre (1970) The Diagram, a Method for Comparing Sequences, Its Use with Amino Acid and Nucleotide Sequences. European Journal of Biochemistry, **(16):** 1–11.

Giegerich R. (2000) A Systematic Approach to Dynamic Programming in Bioinformatics. Bioinformatics, **(16):** 665-677.

Hasan L. and Z. Al-Ars (2009) An Efficient and High Performance Linear Recursive Variable Expansion Implementation of the Smith-Waterman Algorithm. 31st Annual International Conference of the IEEE EMBS, Minneapolis, Minnesota, USA.

Hasan L., Y. Khawaja and A. Bais (2008) A Systolic Array Architecture for The Smith-Waterman Algorithm With High Performance Cell Design. Proceedings of IADIS European Conference on Data Mining, Amsterdam, The Netherlands.

Mueller C., M.M. Dalkilic and A. Lumsdaine (2006) High-performance Direct Pairwise Comparison of Large Genomic Sequences. IEEE Transactions on Parallel and Distributed Systems, 17 (**8):** 764–772.

Oliver T., B. Schmidt and D. Maskell (2005) Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs. FPGA'05, Monterey, California, USA.

Pearson W.R. and D.J. Lipman (1985) Rapid and Sensitive Protein Similarity Searches. Science, **(227):** 1435–1441.

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