FPGA Implementation of Systolic Sequence Alignment*

Dzung T. Hoang†

Department of Computer Science
Brown University, Providence, RI 02912, USA
email: dth@cs.brown.edu

Daniel P. Lopresti‡

Matsushita Information Technology Laboratory
182 Nassau Street, Princeton, NJ 08542-7072, USA
email: dpl@mitl.com

Abstract

This paper describes an implementation of a novel systolic array for sequence alignment on the SPLASH reconfigurable logic array. The systolic array operates in two phases. In the first phase, a sequence comparison array due to Lopresti [2] is used to compute a matrix of distances which is stored in local RAM. In the second phase, the stored distances are used by the alignment array to produce a binary encoding of the sequence alignment. Preliminary benchmarks show that the SPLASH implementation performs several orders of magnitude faster than implementation on supercomputers.

1 Introduction

The work presented in this paper was begun during one co-author's summer internship at the National Cancer Institute's Laboratory of Mathematical Biology in Fredrick, Maryland. The goal was to develop genetic sequence analysis algorithms for the SPLASH reconfigurable logic array [3]. A systolic sequence comparison algorithm that computes the edit distance between a pair of sequences had already been implemented on SPLASH [4]. Certain applications of interest to biologists at the laboratory, such as multiple alignment of genetic (DNA and RNA) sequences, however, require more than just the edit distance: a more informative analysis of the similarity, or homology, of the sequences in the form of an alignment is required. In this paper, we describe an implementation of a systolic algorithm for computing sequence alignments on SPLASH. Prior to our work, we know of no systolic array for computing sequence alignments.

1.1 Sequence Comparison and Alignment

Given a source sequence $S = s_1 s_2 \cdots s_m$ and a target sequence $T = t_1 t_2 \cdots t_n$, the *edit distance* between S and T is defined to be the minimum cost of transforming S to T through a series of the following *edit operations*: deleting a character, inserting a character, and substituting one character for another[1].

In some applications, such as approximate multiple sequence comparison [5] and protein folding [6], in addition to the edit distance, we need to know the series of edit operations that leads to a minimum cost transformation. A standard way to represent the transformation is with an *alignment*. In an alignment, the characters of the source and target sequences are arranged in a matrix with two rows. The source sequence, possibly with embedded null characters, '$-$', is placed in the first row. Similarly, the characters of the target sequence are placed in the second row. The matrix is analyzed column-wise. A column containing $\left[\begin{smallmatrix} x \\ - \end{smallmatrix}\right]$ indicates deletion of the character x; a column containing $\left[\begin{smallmatrix} - \\ y \end{smallmatrix}\right]$ indicates insertion of the character y; and a column $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ indicates substitution of y for x. A column consisting of two nulls is not allowed. Here is an example of an alignment: $\left[\begin{smallmatrix} A & G & A & C & T & A & - & G & G \\ T & G & - & C & T & A & A & G & C \end{smallmatrix}\right]$. For a given cost function, there may be more than one minimum-cost alignment. The alignment algorithm presented here

*A longer version of this paper can be found in [1].

† Supported during Summer 1991 by an NIH Summer Internship and afterwards by an NSF Graduate Fellowship.

‡ Supported by NSF grant MIP-9020570.

[1] A different set of edit operations may be defined to suit a particular application. For example, in text processing, a swap of two adjacent characters may be considered an edit operation. However, a different algorithm than presented here may be required to accomodate these additional edit operations.

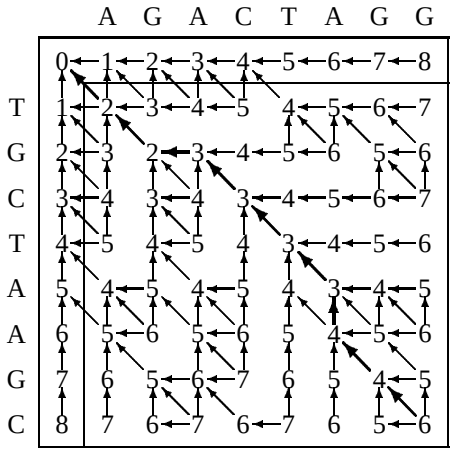

Figure 1: Dynamic programming table with minimization pointers

computes one such alignment.

1.2 Dynamic Programming

The edit distance can be computed sequentially with a well-known dynamic programming algorithm [7,8] in $O(mn)$ time. Let $S = s_1 s_2 s_3 \cdots s_m$ be the source sequence, $T = t_1 t_2 t_3 \cdots t_n$ be the target sequence, and $d_{i,j}$ be the edit distance between the subsequences $s_1 s_2 \cdots s_i$ and $t_1 t_2 \cdots t_j$. Then

$$\begin{aligned} d_{0,0} &= 0, \\ d_{i,0} &= d_{i-1,0} + c_{del(s_i)}, \quad 1 \le i \le m, \\ d_{0,j} &= d_{0,j-1} + c_{ins(t_j)}, \quad 1 \le j \le n, \end{aligned}$$

and

$$d_{i,j} = \min_{\substack{1 \le i \le m, \\ 1 \le j \le n}} \begin{cases} d_{i-1,j} + c_{del(s_i)}, \\ d_{i,j-1} + c_{ins(t_j)}, \\ d_{i-1,j-1} + c_{sub(s_i,t_j)}. \end{cases}$$

Here $c_{del(s_i)}$ is the cost of deleting s_i, $c_{ins(t_j)}$ is the cost of inserting t_j, and $c_{sub(s_i,t_j)}$ is the cost of substituting t_j for s_i.

An alignment can be constructed by creating pointers to indicate the minimization choices when evaluating the dynamic programming recurrence. An example dynamic programming table augmented with pointers is shown in Figure 1. By tracing a path from the lower-right corner to the upper-left corner, we can construct an alignment in reverse. The bold pointers in Figure 1 show the path that corresponds to the alignment given in a previous example.

2 SPLASH Reconfigurable Logic Array

SPLASH is a reconfigurable logic array developed at the Supercomputer Research Center (SRC) as a coprocessor card for the Sun VME bus. The SPLASH board contains 32 Xilinx XC3090 field-programmable gate arrays (FPGA) [9] with local connections to 32 1M-bit (128K by 8) static RAM chips. The FPGA's are connected linearly in a ring with input coming from a 32-bit FIFO queue connected to chip 0 and output going to a 32-bit FIFO queue connected to chip 31. A RAM chip is connected between each pair of adjacent FPGA chips and can be accessed by either FPGA. The data path connecting the FIFO's to the array consists of 36 unidirectional lines, 32 for data and 4 for control signals. Adjacent FPGA's, except for chips 0 and 31, are joined by a 68-bit programmable bidirectional bus, which shares connections to the local RAM. Chips 0 and 31 are connected with a 35-bit data path. This "wrap-around" connection allows data flow through the array in either direction.

At the heart of the SPLASH board are the Xilinx XC3090 FPGA's. Each FPGA contains 320 configurable logic blocks (CLB's) arranged in a 20×16 grid and surrounded by 144 input/output blocks (IOB's). The 144 IOB's surrounding each XC3090 FPGA provides connections to the control bus and programmable interconnections between adjacent chips and local RAM. Each IOB can be configured as either an input port, an output port, or a bidirectional input/output port, with optional latch or flip-flop operation. The programmability of the IOB's allows for flexibility in the interchip connections. For example, when the local RAM is not needed, it can be disabled and the IOB's connected to the RAM's address and data lines can be used for communication between adjacent FPGA's.

The reader is referred to [3] for a more complete description of SPLASH.

Using the FPGA technology in SPLASH, we were able to rapidly prototype the systolic array without having to construct any additional hardware. This approach also has advantages over software-only simulations in that it allowed us to detect and correct race conditions present in early prototypes.

3 Systolic Array for Sequence Alignment

Our systolic array for sequence alignment operates in two phases. In the first phase, the systolic array operates in sequence comparison mode to compute entries in the dynamic programming table and store them in local RAM. In the second phase, the stored table is used to construct an alignment with a *marker passing* systolic

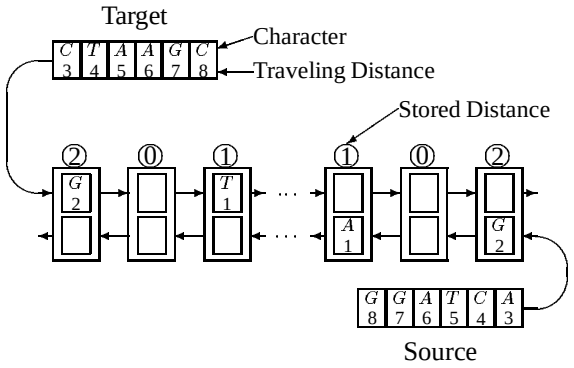

Figure 2: Systolic array for sequence comparison

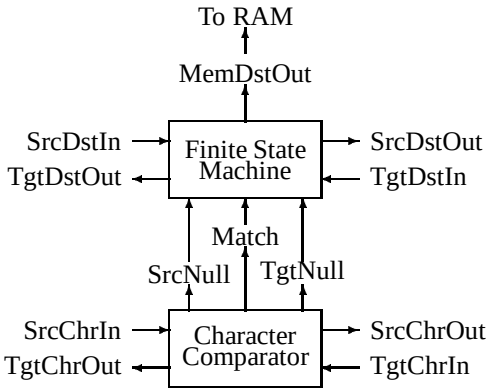

Figure 3: Block diagram of the sequence comparison PE

array.

Since the data path to local RAM is 8-bits wide, for convenience, eight processing elements (PE's) were placed on each FPGA chip, except for X0 and X31, where only four PE's were placed to leave room for I/O logic. This puts a total of 248 PE's on SPLASH, allowing for alignment of sequences up to 123 in length.

3.1 Phase One: Dynamic Programming

The dynamic programming recurrence can be mapped onto a linear systolic array that computes a single antidiagonal of the dynamic programming table at each step, with each PE in the array computing the distances along one diagonal. The resulting systolic array (Figure 2) and its implementation on SPLASH is described in [4]. The array is modified to save the dynamic programming table in local RAM. The first phase ends just after the edit distance, $d_{m,n}$, has been computed.

Figure 3 shows a block diagram of a sequence comparison PE. Each PE is implemented in 13 CLB's, eight for the character comparator and five for the finite state machine.

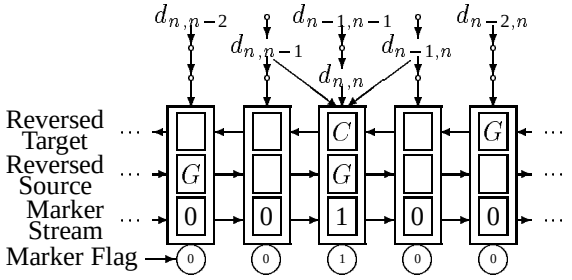

Figure 4: Systolic array for generating alignment

3.2 Phase Two: Marker Passing

The pointer traceback procedure for constructing an alignment, as described earlier, is performed systolically in the second phase. We can think of the traceback as a *marker passing* process in which the marker hops alongs a path created by the minimization pointers. Using the same antidiagonal mapping of the dynamic programming table to PE's as in phase one, we seek to move the marker from the lower-right corner of the table to the upper-left. Following a horizontal pointer would correspond to moving the marker left one PE. Similarly, following a vertical pointer corresponds to moving the marker right one PE. Finally, following a diagonal pointer corresponds to keeping the marker in the same PE. Where there are multiple pointers, one is arbitrarily chosen. In phase one, the minimization pointers were never actually computed. However, we can deduce the pointers originating from position (i, j) given $d_{i,j}$, $d_{i-1,j}$, $d_{i-1,j-1}$, s_i, and t_j. Therefore, by reading back the distances saved in local RAM and streaming the source and target sequences backwards through the array, the minimization pointers, and thus the movement of the marker, can be computed. The algorithm outlined above is realized by the systolic alignment array diagrammed in Figure 4.

The sequence alignment PE is diagrammed in Figure 5. The sequence alignment array uses the same character comparator in the sequence comparison array. The additional finite state machine is implemented in eight CLB's, bringing the total number of CLB's per PE for both phases to 21.

Since at any step, the marker can move at most one PE from its current position, the marker can be registered on a systolic stream that moves across two PE's at each step. The output of the marker stream encodes the movement of the marker. Two consecutive 1's indicate that the marker moved right. Two 0's between successive 1's indicate that the marker moved left. A pattern of 10101 indicates that the marker did not move. The binary pattern exiting the marker stream can be decoded into a series of edit operations by a simple finite

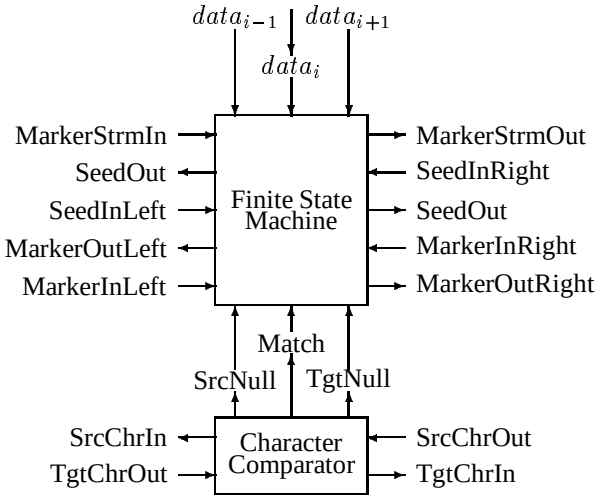

Figure 5: Block diagram of alignment PE

state automaton that counts the number of 0's between successive 1's.

4 Benchmarks

For timing, we performed 10,000 alignments of 100-long sequences on SPLASH. It took 0.50 seconds to initialize the SPLASH array and 3.2 seconds to run the alignments. Normalizing for 100 alignments gives 0.032 seconds. For comparison, the benchmarks for 100 comparisons of 100-long sequences found in [4] are summarized in Figure 6. We have not completed benchmarking sequence alignment on conventional computers and use these results for preliminary comparison. Computing an alignment would require additional processing and therefore take additional time in most implementations. Even including initialization time, the SPLASH implementation performs at least an order of magnitude better than implementations on commercial supercomputers, which, as tested, compute only the edit distance.

5 Conclusion

A systolic array for sequence alignment is presented and its implementation on SPLASH is described. Preliminary benchmarks show that the SPLASH implementation is several orders of magnitude faster than implementations on supercomputers costing many times more.

System	Time	Speed-Up
SPLASH	0.020 s	2,700
P-NAC	0.91 s	60
Multiflow Trace	3.7 s	14
Sun SPARCstation 1	5.8 s	9.3
Cray 2	6.5 s	8.3
Convex C1	8.9 s	6.0
DEC VAX 8600	31 s	1.7
Sun 3/140	48 s	1.1
DEC VAX 11/785	54 s	1.0

Figure 6: Benchmarks of 100 comparisons of 100-long sequences [4]

References

[1] D. T. Hoang, "A Systolic Array for the Sequence Alignment Problem," Brown University, Providence, RI, Technical Report CS-92-22, 1992.

[2] R. J. Lipton and D. P. Lopresti, "A Systolic Array for Rapid String Comparison," in *1985 Chapel Hill Conference on VLSI*, H. Fuchs, Ed. Rockville, MD: Computer Science Press, pp. 363–376, 1985.

[3] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely and D. Lopresti, "Building and Using a Highly Parallel Programmable Logic Array," *Computer*, 24, no. 1, pp. 81–89, January 1991.

[4] D. P. Lopresti, "Rapid Implementation of a Genetic Sequence Comparator Using Field-Programmable Logic Arrays," presented at Advanced Research in VLSI Conference, Santa Cruz, March 1991, Invited paper.

[5] B. A. Shapiro, "An Algorithm for Comparing Multiple RNA Secondary Structures," *Comput. Applic. Biosci.*, 4, no. 3, pp. 387–393, 1988.

[6] H. Margalit, B. A. Shapiro, A. B. Oppenheim and J. V. M. Jr., "Detection of Common Motifs in RNA Secondary Structures," *Nucleic Acids Research*, 17, no. 12, pp. 4829–4845, 1989.

[7] S. B. Needleman and C. D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino-Acid Sequence of Two Proteins," *Journal of Molecular Biology*, 48, pp. 443–453, 1970.

[8] R. A. Wagner and M. J. Fischer, "The String-to-String Correction Problem," *J.. Assn. Comput. Mach.*, 1, pp. 168–173, 1974.

[9] Xilinx, Inc., *The Programmable Gate Array Data Book*. San Jose, CA, 1991.