Searching Genetic Databases on Splash 2

Dzung T. Hoang
Department of Computer Science
Brown University
Providence, RI 02912-1910

Abstract

In this paper, we describe two systolic arrays for computing the edit distance between two genetic sequences using a well-known dynamic programming algorithm. The systolic arrays have been implemented for the Splash 2 programmable logic array, and are intended to be used for database searching. Simulations indicate that that the faster Splash 2 implementation can search a database at a rate of 12 million characters per second, several orders of magnitude faster than implementations of the dynamic programming algorithm on conventional computers.

1 Introduction

With the onset of the Human Genome Initiative [1] and constant advances in genetic sequencing technology, genetic sequence data are being generated at an ever increasing rate[1]. As a result, biologists are faced with an influx of new sequences that they would like to classify and study by comparing them to existing databases. The analysis of a newly generated sequence typically involves searching the database for similar sequences. With the enormous size of the database, fast methods are needed for comparing sequences [3].

In this report, we describe two systolic arrays for computing the edit distance and their implementations on the Splash 2 programmable logic array. One of the systolic arrays was previously implemented on P-NAC [4], a special-purpose VLSI chip, and later ported to the original Splash hardware [5]. The second systolic array is an improvement on the first for database search applications.

[1]Release 74.0 of GenBank, a database of DNA sequences, contains 97,084 entries with a total of 120,242,234 bases as of December 1992. It is estimated that by 1999, 1.6 billion base pairs will be sequenced each year [2].

1.1 Edit Distance

Biologists have developed several means to characterize the similarity between genetic sequences. One intuitively appealing measure is the *edit distance*. The edit distance between two sequences is defined as the minimum cost of transforming one sequence to the other with a sequence of the following operations: deleting a character, inserting a character, and substituting one character for another. (No character may take part in more than one operation.) Each operation has an associated cost, which is a function of the characters involved in the operation. The cost of the transformation is then the sum of the costs of the individual operations.

1.2 Dynamic Programming Algorithm

The calculation of the edit distance can be done with a well-known dynamic programming algorithm, which has an interesting history of independent discovery as detailed in [6]. We use the following formulation.

Let $S = [s_1 s_2 \cdots s_m]$ be the source sequence, $T = [t_1 t_2 \cdots t_n]$ the target sequence, and $d_{i,j}$ the distance between the subsequences $[s_1 s_2 \cdots s_i]$ and $[t_1 t_2 \cdots t_j]$. Then for $1 \le i \le m, 1 \le j \le n$,

$$\begin{aligned} d_{0,0} &= 0 \ , \\ d_{i,0} &= d_{i-1,0} + c(s_i, \emptyset) \ , \\ d_{0,j} &= d_{0,j-1} + c(\emptyset, t_j) \ , \end{aligned}$$

and

$$d_{i,j} = \min \begin{cases} d_{i-1,j} + c(s_i, \emptyset) \\ d_{i,j-1} + c(\emptyset, t_j) \\ d_{i-1,j-1} + c(s_i, t_j) \ , \end{cases} \quad (1)$$

where $c(s_i, \emptyset)$ is the cost of deleting s_i, $c(, \emptyset, t_j)$ is the cost of inserting t_j, and $c(s_i, t_j)$ is the cost of substituting t_j for s_i. The edit distance between S and T is simply $d_{m,n}$.

A straightforward sequential implementation of the dynamic programming algorithm requires $O(mn)$ time

0-8186-3890-7/93 $03.00 © 1993 IEEE

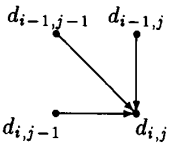


Figure 1: Locality of computation

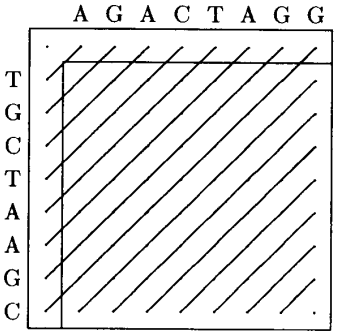


Figure 2: Parallel computation of DP distance matrix

and $O(\min(m,n))$ space to compute the edit distance. Masek and Patterson [7] give an algorithm with time performance of $O(n^2/\log n)$ for sequences of length n, provided that the sequence alphabet is finite and all costs are integral multiples of some real number r. However, they estimate that their algorithm will perform faster than the basic dynamic programming algorithm only for sequences of length 262,419 or longer. Better time performance can also be achieved by exploiting the inherent parallelism in Equation (1).

2 Systolic Sequence Comparison

One notable property of the dynamic programming recurrence for computing edit distances is that each entry in the distance matrix depends on adjacent entries, as shown in Figure 1. This locality can be exploited to produce systolic algorithms in which communication is limited to adjacent processors.

One can map the edit distance computation onto a linear systolic array in several ways. Two mappings are described below. Both exploit the locality of reference by computing the entries along each antidiagonal in parallel, as shown in Figure 2. The two mappings differ primarily in the data movement.

2.1 Bidirection Array

The data flow shown in Figure 3 was used in the design of the Princeton Nucleic Acid Comparator (P-NAC) [4], a custom VLSI chip for DNA sequence comparison. Each processing element (PE) computes the distances along a particular diagonal of the distance matrix. A block diagram of the PE and a listing of the algorithm it executes are shown in Figure 4. The source and target sequences enter the array on opposite ends and flow in opposing directions at the same speed. Successive characters in the source and target sequences are separated by a null character for proper timing. In addition, there is one distance stream associated with each character stream[2]. At each step, the contents of the streams represent the characters to be compared and the distances along one of antidiagonals of the distance matrix. At the end of the computation, the resulting edit distance is transported out of the array on the distance streams.

In addition to the original P-NAC implementation, the bidirectional systolic array has been ported to the Splash programmable logic array [5] and now the Splash 2 programmable logic array. An extension of the bidirectional array to compute the alignment of two sequences in addition to the edit distance is described in [8,9].

Comparing sequences of lengths m and n requires at least $2\max(m+1, n+1)$ processors. The number of steps required to compute the edit distance is proportional to the length of the array.

In a typical database search, the same source sequence is compared against all sequences in the database. With the bidirectional array, the source sequence must be cycled through the array once for each target sequence in the database. At each step, at most half of the PE's are active. Also, the source and target sequences are both limited in length to half of the array's length.

2.2 Unidirectional Array

We now describe a *unidirectional* systolic array that remedies the shortcomings of the bidirectional array. The data flow of the unidirectional array is shown Figure 5. As the name suggests, data flows through the unidirectional array in one direction. The source sequence is loaded once and stored in the array starting from the leftmost PE. The target sequences are

[2] In an actual implementation, these two unidirectional distance streams can be combined into one bidirectional streams, using one storage register instead of two. Here we keep the distance streams distinct for clarity.

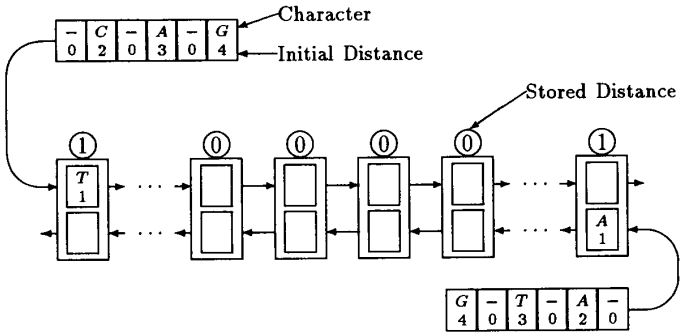
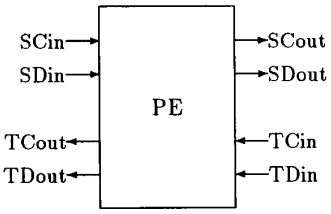


Figure 3: Data flow through bidirectional systolic array



a) Block Diagram

```
loop
   SCout ← SCin
   TCout ← TCin
   if (SCin ≠ ∅) and (TCin ≠ ∅) then
      PEDist ← min { PEDist+c(SCin,TCin),
                     TDin+c(SCin,∅),
                     SDin+c(∅,TCin)
      SDout ← PEDist
      TDout ← PEDist
   elsif (SCin ≠ ∅) then
      SDout ← SCin
      TDout ← SCin
   elsif (TCin ≠ ∅) then
      SDout ← TDin
      TDout ← TDin
   else
      SDout ← PEDist
      TDout ← PEDist
   endif
endloop
```

b) Algorithm

Figure 4: Processing element for bidirectional array

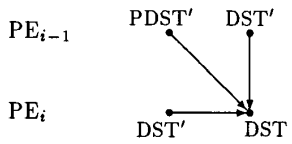


Figure 7: Computation graph for a unidirectional PE

streamed through the array one after another, separated by a control character. The tag stream identifies the sequences and sends control information to the PE's. With the source sequence loaded and the target sequences streaming through, the array can near 100% PE utilization. The length of the array determines the maximum length of the source sequence. The target sequences, however, can be of any length. Together, these properties make the unidirectional array more suitable and efficient than the bidirectional array for database searches.

The unidirectional PE is diagrammed in Figure 6a. In this configuration, each PE computes the distances in one row of the distance matrix. At each time step, the PE's compute the distances along an antidiagonal in the distance matrix, as depicted in Figure 2. Each PE stores two distances, DST and PDST. Denoting previously computed values of DST and PDST as DST′ and PDST′, respectively, the computation graph for a given PE is shown in Figure 7.

The algorithm executed by each PE in the unidirectional array is listed in Figure 6b. As shown, the algorithm compares one source sequence to a single target sequence. With some additional code, comparisons can be performed on multiple source and target sequences.

A unidirectional array of length n can compare a source sequence of length at most n and to a target sequence of length m in $O(n+m)$ steps.

3 Implementation

Both the bidirectional and unidirectional systolic arrays have been implemented on the Splash 2 programmable logic array, with versions for DNA and protein sequences.

Only a brief description of the Splash 2 architecture and programming environment is given here. The reader is referred to [10,11,12] for more details.

3.1 Splash 2

Splash 2 is a second-generation programmable logic array targeted at custom computing. A Splash 2 system consists of a interface board containing two Xilinx 4010 FPGA's and up to 16 programmable logic boards each containing 17 Xilinx 4010 FPGA's. Sixteen of the FPGA's are connected in a linear array with programmable crossbar connections. The 17th FPGA is used to control the crossbar. In addition, each FPGA is directly connected to a 256K × 16 static RAM.

Splash 2 is programmed using VHDL for design entry. Although VHDL was designed as a hardware description language, we are able to use its behavioral modeling capabilites to capture the essential algorithmic elements of the systolic arrays. We use structural modeling to specify the connection between array elements. At first, we perform simulations of the VHDL code for debugging and verification. Working VHDL code is then synthesized using a commercial VHDL compiler to generate a gate-level design. This is then fed to the Xilinx CAD tools to create a placement and routing for the 4010 FPGA. Timing results are then extracted from the placement and routing information. The design cycle is summarized in Figure 8.

3.2 Modulo Encoding

An important optimization used in the implementation of both systolic arrays involve a modulo encoding of the distances. With a fixed-length unsigned-integer data structure for the distances, there is possibility of overflow when comparing long sequences. In [4] a modulo encoding scheme is used for the distances. With a particular cost function for DNA sequences, only two bits are required for the encoding; for protein sequences, only four bits are needed. The modulo scheme reduces the design size, circumvents the overflow problem, and allows for easy scaling of the systolic array. To recover the integer distances, a counter, controlled by a simple state machine, is used at the output of the distance stream.

3.3 Configurable Parameters

Both systolic arrays were implemented with user-configuration in mind. The sequence alphabet and cost functions are defined in a VHDL configuration file, and can be customized for a particular sequence comparison application. A change in the parameters, however, would require a recompilation of the VHDL code. Versions for comparing DNA and protein sequences have been implemented.

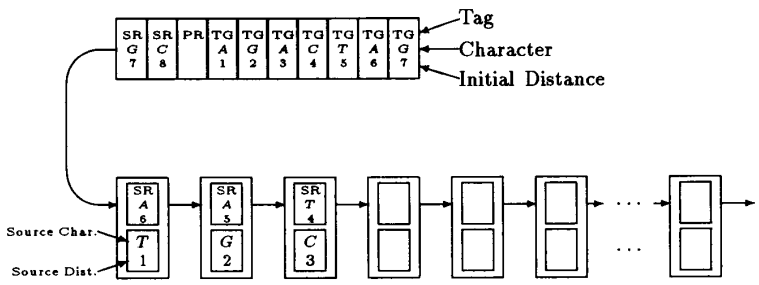
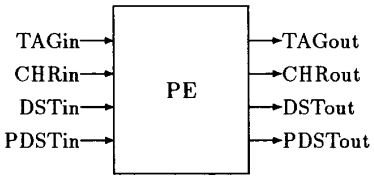


Figure 5: Data flow through unidirectional systolic array



a) Block Diagram

```
loop
  if (TAGin = SR) then
    CHRout ← ∅
    SRCch ← CHRin
    if (SRCch = ∅) then
      DSTout ← PDSTin
    endif
    PDSTout ← PDSTin
  else
    CHRout ← CHRin
  endif
  if (TAGin = PR) then
    if (CHRin = ∅) then
      DSTout ← PDSTin
    endif
    PDSTout ← PDSTin
  endif
  if (TAGin = TG) then
    if (SRCch ≠ ∅) and (CHRin ≠ ∅) then
      DSTout ← min { PDSTin+c(SCin,TCin),
                     DSTin+c(SCin,∅),
                     DSTout+c(∅,TCin) }
    elsif (SRCch = ∅) then
      DSTout ← DSTin
    endif
    PDSTout ← DSTin
  endif
endloop
```

b) Algorithm

Figure 6: Processing element for unidirectional array

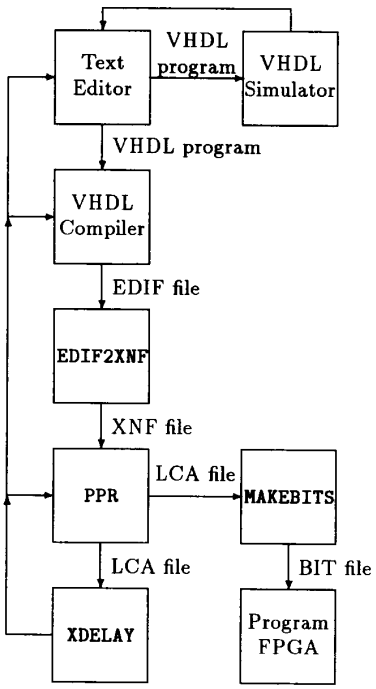


Figure 8: Programming cycle for Splash 2

3.4 Status of Implementation

Currently, DNA and protein versions of both the bidirectional and unidirectional arrays have been coded, synthesized, and timed. We are awaiting the completion of the Splash 2 hardware and system software in order to test the systolic arrays on actual hardware.

3.5 Bidirectional Array

The bidirectional array was coded in VHDL in 1482 lines for the DNA version and 1511 lines for the protein version. For the DNA version, each of the 16 array FPGA's (X1 to X16) contains 24 PE's, making a total of 384 PE's in a one-board Splash 2 system. The protein version packs 64 PE's into a one-board Splash system. The maximum throughput for database search is estimated at 5.5 million characters per second for the DNA version and 3.5 million characters per second for the protein version.

Table 1: Benchmark of DNA sequence comparison (Values are rounded to two decimal places.)

Hardware	Specifics	CUPS
Splash 2	unidir; 16 boards	43,000M
Splash 2	bidir; 16 boards	34,000M
Splash 2	unidir; 1 board	3,000M
Splash 2	bidir; 1 board	2,100M
Splash 1	bidir; 746 PE's	370M
MP-1 [13]	8K PE's	32M
CM-2 [14]	16K PE's	5.9M
BSYS [14]	100 PE's	2.9M
SPARC 10/30GX	gcc -O2	1.2M
P-NAC [4]		1.1M
VAX 6620	VMS; CC	1.0M
SPARC 1	gcc -O2	0.87M
486DX-50 PC	DOS; gcc -O2	0.67M

3.6 Unidirectional Array

The unidirectional array was coded in VHDL in 1506 lines for the DNA version and 1528 lines for the protein version. In the DNA version, each of the 16 array FPGA's (X1 to X16) contains 14 PE's. In addition, the two interface FPGA's contain 12 PE's each, making a total of 248 PE's in a one-array-board Splash 2 system. The maximum throughput for database search is estimated at 12 million characters per second for the DNA version and 8 million characters per second for the protein version.

4 Benchmarks

Even though actual running times are currently unavailable for our systolic arrays, we present timing results generated by the CAD software and compare them to implementations on conventional computers and other hardware accelerators.

In order to make a fair comparison between splash and implementations of the dynamic programming algorithm on other architectures, we measure the performance of a solution in terms of the number of cells (entries in the DP distance table) updated per second (CUPS). In comparing two sequences of lengths n and m, a total of nm cells need to be calculated.

The benchmark results for DNA sequence comparison are listed in Table 1. The values given for Splash 1 and Splash 2 are peak values assuming that the length of the sequences are the maximum for the given configuration and that pipeline delays are ig-

nored. A straightforward implementation of the dynamic programming algorithm in the C language is used to benchmark the conventional computers listed. Typically, a run consisting of 1000 repetitions of a 1000 × 1000 comparison is used to calculate the CUPS.

5 Conclusion

Two systolic arrays for computing the edit distance between two genetic sequences are presented and their implementation on Splash 2 are described. Preliminary timing results show that the bidirectional and unidirectional arrays have a throughput of 5.5 and 12 million characters per second, respectively, for DNA database search. Compared to a direct implementation of the dynamic programming algorithm on several contemporary workstations and minicomputers, the Splash 2 implementations promise to deliver several orders of magnitude more performance.

Acknowledgements

The author would like to thank the Splash 2 team at the Supercomputing Research Center for their generous support of the work described here. The author is also indebted to Daniel Lopresti, Richard Hughey, Bruce Shapiro, and Morten Schultz for their contribution of ideas and constructive criticism.

References

[1] K. A. Frenkel, "The Human Genome Project and Informatics," *Comm. ACM*, 34, no. 11, pp. 41–51, November 1991.

[2] E. S. Lander, R. Langridge and D. M. Saccocio, "Computing in Molecular Biology: Mapping and Interpreting Biological Information," *Computer*, 24, no. 11, pp. 6–13, November 1991.

[3] ———, "Mapping and Interpreting Biological Information," *Communications of the ACM*, 34, no. 11, pp. 32–39, November 1991.

[4] 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.

[5] 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.

[6] D. Sankoff and J. Kruskal, Eds., *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*: Addison-Wesley, 1983.

[7] W. J. Masek and M. S. Paterson, "How to Compute String-Edit Distances Quickly," in *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, D. Sankoff and J. Kruskal, Eds. Addison-Wesley, 1983.

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

[9] D. T. Hoang and D. P. Lopresti, "FPGA Implementation of Systolic Sequence Alignment," presented at 1992 International Workshop on Field Programmable Logic, Vienna, Austria, August 1992.

[10] J. M. Arnold, D. A. Buell and E. G. Davis, "SPLASH 2," presented at ACM Symposium on Parallel Algorithms and Architectures, June 1992.

[11] D. A. Buell, "A Splash 2 Tutorial," Supercomputing Research Center, Bowie, Maryland, Technical Report SRC-TR-92-087, December 1992.

[12] J. M. Arnold, "The Splash 2 Software Environment," presented at IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 1993.

[13] R. P. Hughey, Personal Communications.

[14] ———, "Programmable Systolic Arrays," Brown University, Computer Science Tech. Report CS-TR-91-34, May 1991, Ph.D. Thesis.