A Systolic Array for Rapid String Comparison

Richard J. Lipton and Daniel Lopresti
Department of Electrical Engineering and Computer Science
Princeton University, New Jersey

ABSTRACT

This paper presents a linear systolic array for quantifying the similarity between two strings over a given alphabet. The architecture is a parallel realization of a standard dynamic programming algorithm. Also introduced is a novel encoding scheme which minimizes the number of bits required to represent a state in the computation, significantly reducing the size of a processor. An nMOS prototype, to be used in searching genetic databases for DNA strands which closely match a target sequence, is being implemented. Preliminary results indicate that it will perform hundreds to thousands of times faster than a minicomputer.

1. Introduction

String comparison is an important operation in many disciplines; a particularly interesting application comes from the field of molecular biology. After isolating and "sequencing" a chain of DNA, which may consist of from tens to thousands of the four bases Adenine, Cytosine, Guanine, and Thymine (abbreviated A, C, G, and T), biologists often want to search a database of known DNA for close matches. In doing so they hope that previous results will help them draw conclusions about their new strand. One such database, maintained by the National Institutes of Health, contains millions of bases, so such searches can take hours of mainframe time. Hence, molecular biologists resort to heuristics; they only search against the portion of the database which they consider relevant and they use sub-optimal algorithms which trim the search. In doing so, they may miss an unexpected, but important, homology. Still, these searches require significant computational resources and time [1], [2], [3].

† This work was supported in part by DARPA Contract N00014-82-K-0549.

Presented in this paper is a linear (i.e., one-dimensional) systolic array for the string comparison problem in general, as illustrated by the DNA comparison problem in particular. It is a parallelization of an optimal dynamic programming algorithm, which promises to run thousands of times faster than the same algorithm run on a serial computer. As is characteristic of systolic arrays (an architecture first described by H.T. Kung and associates at Carnegie-Mellon University [4]) the machine is composed of a large number of simple processors which are highly regular and have only local communication requirements; hence it is ideal for implementation in VLSI.

Perhaps even more significantly, a technique is introduced which greatly simplifies the processing elements, allowing a relatively large number to be fit on a single chip. This observation, which minimizes the amount of data which must flow between adjacent processors, may be applicable to other systolic implementations.

2. Quantifying String Similarity

An intuitive metric for quantifying the similarity of two strings is their "edit" distance [5]. That is, a count of the number of basic editing operations:

i) insert
ii) delete
iii) substitute

needed to transform one string, the "source," into the other, the "target."
For example:

to transform ACG into TGG

ACG -*delete A*→ CG -*delete C*→ G -*insert G*→ GG -*insert T*→ TGG

which is two deletions and two insertions, so the difference between ACG and TGG is four. A substitution has cost two, as it is equivalent to a deletion and an insertion:

ACG -*substitute T for A*→ TCG -*substitute G for C*→ TGG

The edit distance between two strings can be calculated using a dynamic programming algorithm which would, in the above case, build the following table:

		T	G	G
	0	1	2	3
A	1	2	3	4
C	2	3	4	5
G	3	4	3	4

The difference, four, is the entry in the lower right corner of the matrix.

In general, the value d in the 2×2 table fragment:

		t_j
		...
	a	b
s_i	... c	d

is determined by the rule:

$$d = \min \begin{cases} b + 1 \\ c + 1 \\ \begin{cases} a & \text{if } s_i = t_j \\ a + 2 & \text{if } s_i \neq t_j \end{cases} \end{cases}$$

It is known that dynamic programming algorithms map well onto systolic arrays [6]. In particular, there is a tremendous potential for concurrency in the construction of an edit distance table; the entries on a given 45 degree diagonal can be calculated simultaneously because

they depend only on values up and to the left. This parallelism can be realized with a systolic array. All of the values on 45 degree diagonals will be calculated at the same time and all of the values on -45 degree diagonals will be calculated in the same processor. In this way the quadratic time serial algorithm becomes a linear time parallel algorithm requiring a linear number of processors.

3. The Systolic Architecture

A rhythmic pumping of data through simple processors distinguishes systolic arrays. Figure 1 demonstrates the data flow through one implementation of a linear comparison array using the strings of the earlier example. The source and target are shifted in simultaneously from the left and right respectively. Interleaved with the characters are the data values from the first row and column of the dynamic programming matrix. When two non-null characters enter a processor from opposite directions a comparison is performed (indicated by the darkened circles in the figure). On the next clock tick the characters shift out and the values following them shift in. This processor now determines a new state based on the result of the comparison, the two values just shifted in, and its previous state value, using the same rule as in the dynamic programming algorithm. When the strings are shifted out they carry with them the last row and column of the dynamic programming matrix, and hence the answer.

Strings too long to be compared in one pass through the array can be handled by making multiple passes. In this case, the initializing data values shifted in with the characters reflect a matrix row or column at an intermediate point in the calculation.

4. Minimizing Processor Size

Using the above scheme, comparing two strings of length n in one pass requires approximately $2n$ processing elements. Since DNA sequences of interest can be several thousand bases long it is necessary to fit a large number of processors onto a single chip. The greatest influence on the size of a single processor is the appearance that it must add and compare relatively large data values, on the order of $\log(n)$ bits. Even being optimistic, such adders and comparators require significant silicon area; it is unlikely that a naive implementation would fit more than a couple of processors onto one chip. There is, however, an important property of this algorithm which makes manipulating such large values unnecessary; a *constant* two bits will suffice for *any* length string comparison. The key observation is that

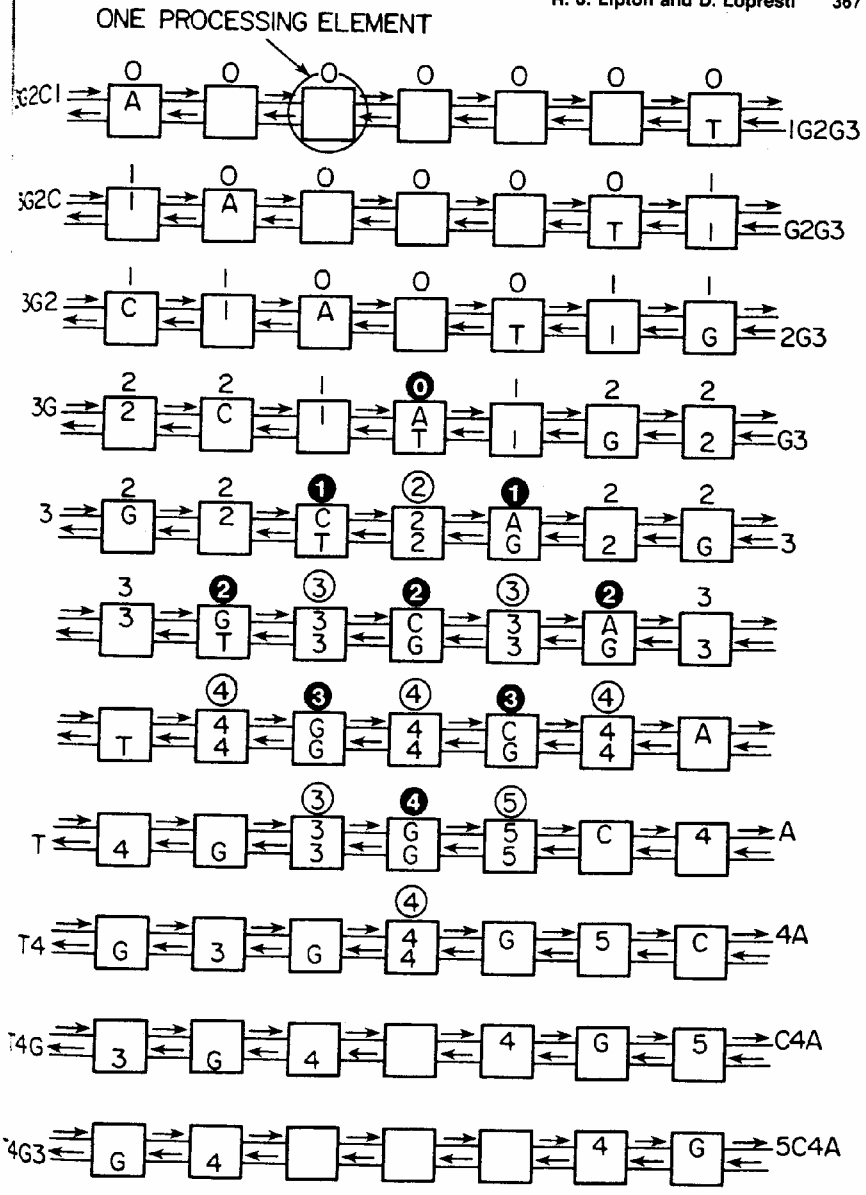

FIGURE 1. A SYSTOLIC ARRAY FOR STRING COMPARISON

adjacent processors' state values can not differ greatly in magnitude, hence it is only necessary to pass low order bits between processors.

More formally,

To Prove: it is possible to construct a string comparison table:

		t_1	t_2		t_{j-1}	t_j	
	0	1	2	...	$j-1$	j	...
s_1	1			...			...
					...		
s_{i-1}	$i-1$			...	a	b	...
s_i	i			...	c	d	...
					...		

using the rule:

(rule 1):

$$d = \min \begin{cases} b + 1 \\ c + 1 \\ \begin{cases} a & \text{if } s_i = t_j \\ a + 2 & \text{if } s_i \neq t_j \end{cases} \end{cases}$$

consistently by keeping only remainders modulo 4. In other words, determining the remainder of d modulo 4 only requires knowing the characters s_i and t_j and the remainders of each of a, b, and c modulo 4.

The essential point is that adjacent matrix elements must be close in value.

Lemma: the values of horizontal and vertical neighbors in the matrix differ by ± 1. That is, a 2×2 matrix fragment must look like:

a	$b = a \pm 1,\ d \pm 1$
$c = a \pm 1,\ d \pm 1$	d

Proof of Lemma: by induction on $i + j$, the sum of the row and column indicies.

Basis: for $i + j = 2$ the matrix fragment is the upper left corner:

		t_1		t_2
	0	1		2
s_1	1	0	or	2
s_2	2			

Only one value is calculated and by rule 1 it must be 0 or 2, hence horizontal and vertical neighbors differ by ± 1

Inductive hypothesis: assume the lemma is true for $i + j < n$.

Induction: say $i + j = n$:

			t_j
			...
		a	$b = a \pm 1$
s_i	...	$c = a \pm 1$	d

The values of b and c are restricted to $a \pm 1$ by the inductive hypothesis, hence the value of d can be restricted. Rewriting rule 1:

(rule 2):

$$d = \begin{cases} a & \text{if } b \text{ or } c \textit{ equals } a-1 \text{ or } s_i = t_j \\ a+2 & \text{if } b \text{ and } c \textit{ equal } a+1 \text{ and } s_i \neq t_j \end{cases}$$

In either of the above cases $b = d \pm 1$ and $c = d \pm 1$, thus horizontal and vertical neighbors always differ by ± 1. This completes the proof of the lemma.

Determining the remainder of d modulo 4 using rule 2 only requires deciding between the two cases $d = a$ and $d = a + 2$. Since b and c are guaranteed to be within ± 1 of a, it is only necessary to know the remainders of each of a, b, and c modulo 4 to make this differentiation. This completes the proof.

As an aside, a stronger statement is possible: only the next to least significant bit of both b and c are necessary to differentiate the cases, but the two low order bits are used in the current implementation to make initializing the array easier.

Of course, knowing only the remainder modulo four of the edit distance between the source and target strings would be of little use, but the fact that the entire last row and column of the comparison matrix are shifted out means that the full difference can be constructed outside of the array. All that is required is to pre-load a counter with the length of the target string and to increment or decrement it appropriately as the source values shift out of the right side of the array. Figure 2 demonstrates this.

5. The Implementation

Figure 3 is a plot of a single processing element. The layout was done using **allende** [7], a procedural language developed at Princeton, **caesar**, and the Berkeley PLA tools [8]. The four DNA bases are encoded as 4-bit nibbles. Wild cards, characters which match any subset of the bases, are also available. The source and target characters shift in from the left and right respectively at the bottom of the processor. The character comparison is performed in the lower PLA during the first clock phase. The result of the comparison, as well as the source and target data values, are input to the top PLA which

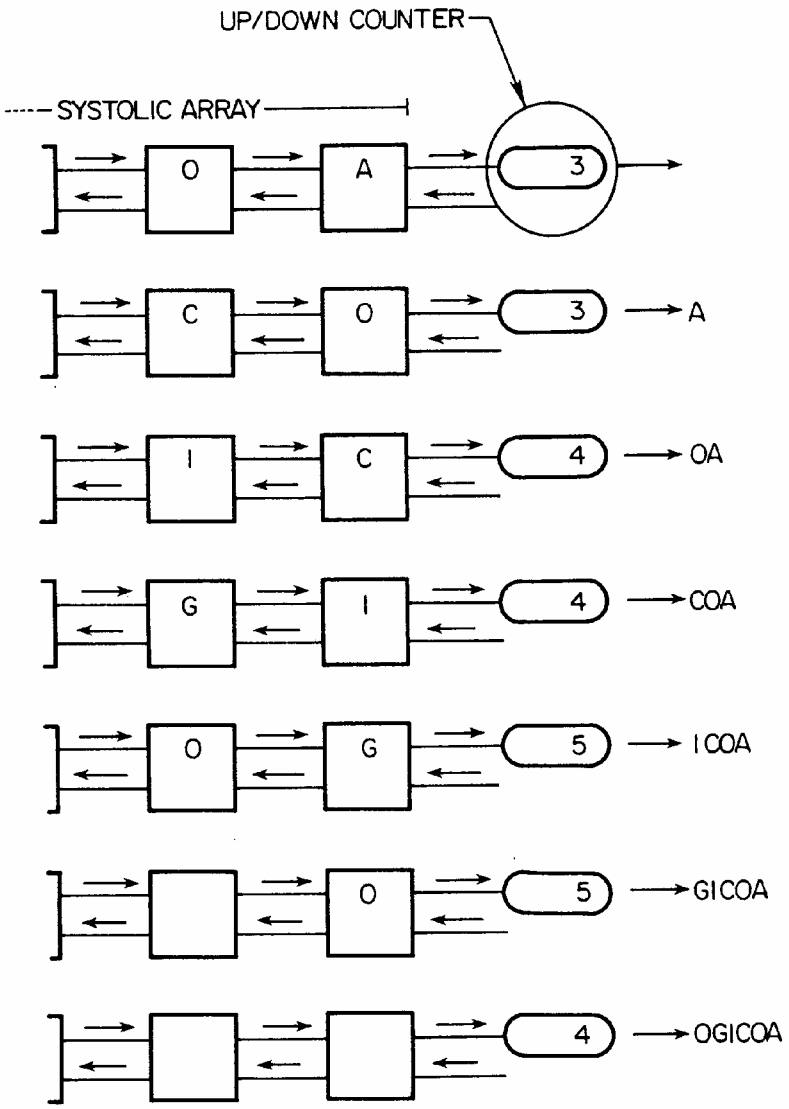

FIGURE 2. CONSTRUCTING THE EDIT DISTANCE

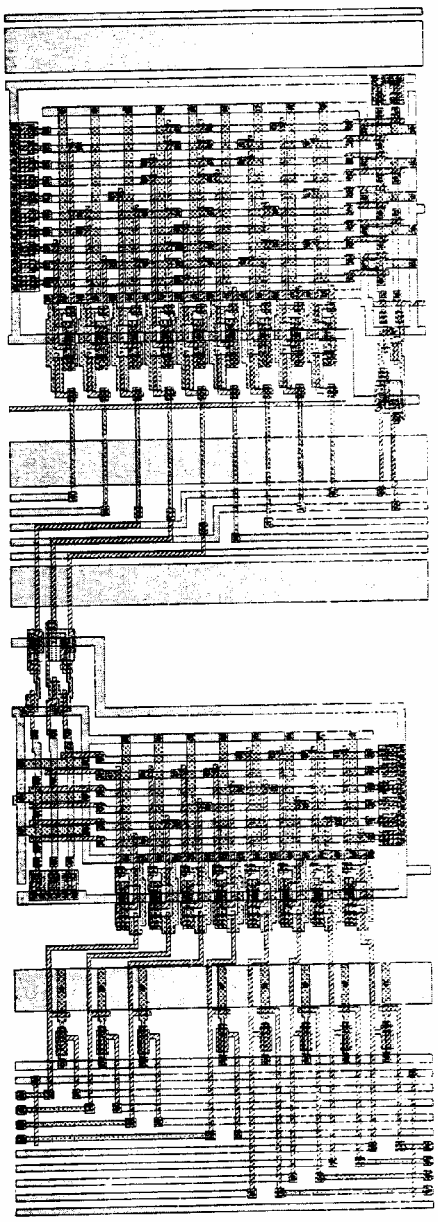

Figure 3 A Single Processing Element

determines a new state during the second clock phase.

The current *nMOS* design (lambda = 2.0 microns) fits 30 processors in a $4.6mm \times 6.8mm$ 40 pin MOSIS standard frame. A plot of the prototype is shown in figure 4. The processors are almost identical, they simply alternate as to whether they calculate their new state during the chip's phi *a* or phi *b*. The design is expansible; building a larger array involves simple connecting a number of chips side by side. Also implemented is a bypass option which shortens the path through a chip from 30 to 10 processors; this will keep short comparisons from needlessly passing through the entire array.

The factor presently limiting the number of processors which can be placed on one chip is power dissipation; estimates for the *nMOS* design indicate that it will consume approximately one watt. A CMOS version is planned which will run much cooler, so that with the same feature size from 50 to 80 processors can be placed on a chip.

The systolic array is data hungry; it requires two 4-bit characters, two 2-bit counts and output monitoring three million times a second to operate at full speed. Since the array is expected to function as a hardware accelerator for micro-and minicomputers a primary concern is the host to array interface. Work is just beginning in this area.

6. Expected Performance

The planned configuration of the linear systolic array has a minimum of 340 processors and a maximum of 1020 processors, selectable in multiples of 20 via the bypass option. This would require 34 chips for the array plus support logic, a reasonable number to fit on a standard printed circuit board. The canonical minicomputer for comparison is the 1 MIPS VAX 11/780†. Assume that the systolic array can be clocked at 3 *MHz* (**crystal** timings support this) and that the VAX takes 10 instructions to do the inner loop step in the dynamic programming algorithm. Then the time to compare a source and target string, both of the same length, is given by:

$$time_{systolic} = (length + no.\ of\ PEs)\ /3 \times 10^6$$

$$time_{VAX} = (length^2)\ /1 \times 10^5$$

† VAX is a trademark of Digital Equipment Corporation.

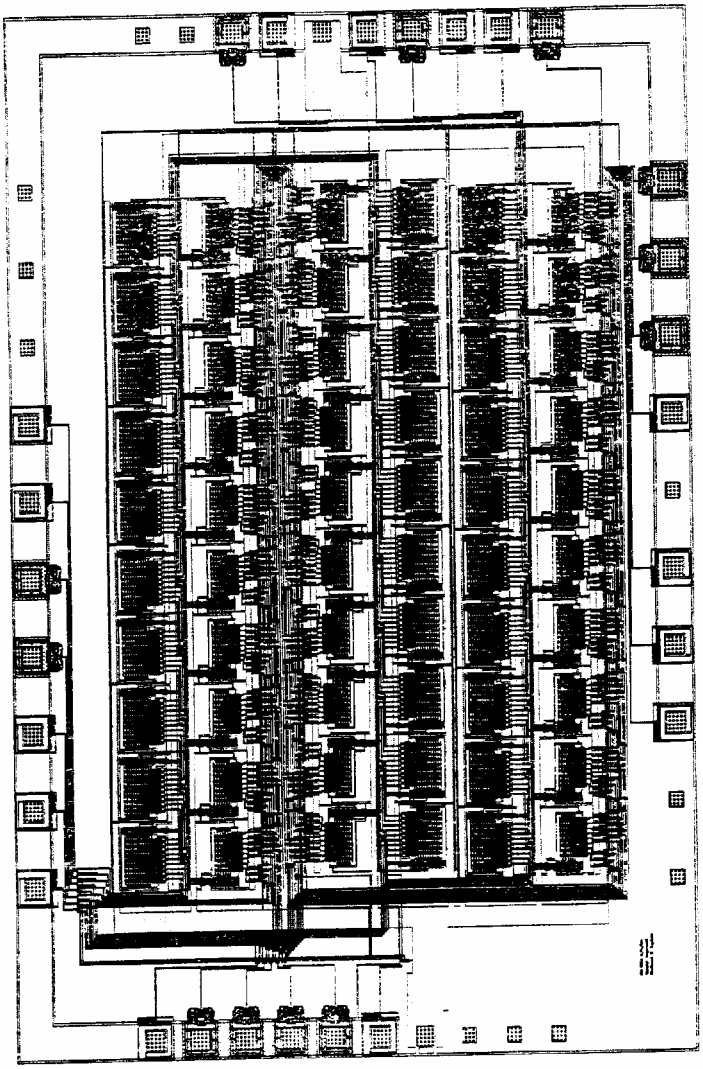

Figure 4 The Chip

length	PEs	systolic time	VAX time	speedup factor
15	340	1.18×10^{-4} secs.	2.25×10^{-3} secs.	19.1
50	340	1.30×10^{-4}	2.50×10^{-2}	192
100	340	1.47×10^{-4}	0.10	680
250	520	2.57×10^{-4}	0.62	2430
500	1020	5.07×10^{-4}	2.50	4930
1000	*	2.03×10^{-3}	10.0	4930

*four times the length 500 case through 1020 processors.

Figure 4 The Chip

7. Summary

This paper has presented a linear systolic array for rapid string comparison. Also introduced is an encoding scheme, significant in its own right, which minimizes the required state information internal to the array. Further research will determine whether this technique can be applied to other systolic algorithms.

An *nMOS* design for the specific problem of comparing DNA sequences is being implemented; it fits 30 processor on a single chip. Conservative estimates indicate that it will perform comparisons hundreds to thousands of times faster than a serial computer.

Acknowledgments

The authors would like to thank Doug Welsh of the Princeton Molecular Biology Department for his help. Daniel Lopresti gratefully acknowledges the support of a Garden State Graduate Fellowship.

References

[1] W. J. Wilbur and David J. Lipman, "Rapid similarity searches of nucleic acid protein data banks," *Proc. Natl. Acad. Sci. USA*, vol. 80, pp. 726-730, February 1983.

[2] James W. Fickett, "Fast optimal alignment," *Nucleic Acids Research,* vol. 12, number 1, pp. 175-179, 1984.

[3] Martin Bishop and Elizabeth Thompson, "Fast computer search for similar DNA sequences," *Nucleic Acids Research,* vol. 12, number 13, pp. 5471-5474, 1984.

[4] H. T. Kung and C. Leiserson, "Algorithms for VLSI processor arrays," *Introduction to VLSI Systems,* C. Mead and L. Conway, Eds. Reading, MA: Addison-Wesley, 1980.

[5] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. Ass. Comput. Mach.,* vol. 1, pp. 168-173, 1974.

[6] L. J. Guibas, H. T. Kung, and C. D. Thompson, "Direct VLSI Implementation of Combinatorial Algorithms," *Proc. Conf. VLSI: Architecture, Design, Fabrication,* California Institute of Technology, Pasadena, CA, 1979, pp. 509-525.

[7] Jose Monteiro da Mata, "The ALLENDE Layout System User's Manual," Dep. Elec. Eng. Comput. Sci., Princeton Univ., VLSI Memo No. 9, June 1984.

[8] *1983 VLSI Tools,* Robert N. Mayo, John K. Ousterhout, and Walter S. Scott, Eds. Comp. Sci. Div., University of California at Berkeley, Report No. UCB/CSD 83/115, March 1983.