A Parallel Implementation of Smith-Waterman Sequence Comparison Algorithm

Brian Hang Wai Yang
ID: 4469409

1.0 Abstract

Exploiting the Gotoh's enhancement, a novel VLSI implementation of Smith-Waterman algorithm is presented. We also present a unique way to place gates using bit-stacking technique. This enable the development of scalable specialized hardware solution that can readily migrate to more advanced VLSI process.

2.0 Introduction

Sequence database searching is among the most important and challenging tasks in bioinformatics. The ultimate choice of sequence search algorithm is that of Smith-Waternan. However, because of the computationally demanding nature of this method, heuristic programs have been developed. Increased speed has been obtained at the cost of reduced sensitivity.

The rapidly increasing amounts to genetic-sequence information available represent a constant challenge to developers of hardware and software database searching and handling. The size of GenBank/EMBL/DDBJ necleotide database has been doubling every 15 months (Benson et al 2000). The rapid expansion of the genetic sequence information is probably exceeding the growth in computer power, in spite of the fact that computing resource also have been increasing exponentially for many years. If this trend continues, increasingly longer time or increasing more expensive computers will be needed to search the entire database.

When looking for sequences in a database similar to a given query sequence, the search programs compute an alignment score for every sequence in the database. This score represents the degree of similarity between the query and database sequence. The score is calculated from the alignment of the two sequences, and is based on a substitution score matrix and a gap-penalty function. A dynamic programming algorithm for computing the optimal local-alignment score was first described by Smith and Waterman (1981) and later enhanced by Gotoh (1982) for linear gap-penalty functions.

Database searches using the optimal algorithm are unfortunately quite slow on ordinary computers, so many heuristic alternatives have been developed, such as FASTA and BLAST. These methods have reduced the running time by a factor of upto 40 compared with the best-known Smith-Waterman implementation, however, at the expense of sensitivity. As a result, a distantly related sequence may not be found in a search using these heuristic algorithms.

For high-speed implementation of the Smith-Waterman algorithm, some have exploited the single-instruction, multiple-data (SIMD) computers. A SIMD computer is able to perform the same operation on several independent data sources in parallel. With the introduction on Pentium MMX microprocessor in 1997, Intel made computing with SIMD technology available in a general-purpose microprocessor in the most widely used computer architecture - the industry-standard PC.

Several special-purpose hardware solutions have been also developed for Smith-Waterman algorithm, such as Paracel's GeneMatcher, Compugen's Bioaccelerator and TimeLogic's Decypher. These machines are able to process more than 2000 million matrix cells per second, and can be expanded to reach much higher speeds.

We present a novel VLSI implementation that exploits the locality of the algorithm. Further, an optimal placement-and-route paradigm that can generate *placed circuit* for any given size. This allows the circuit to be readily migrated to different fabrication process and device frequency.

3.0 Smith-Waterman Algorithm

To compute the optimal local-alignment score, the dynamic programming algorithm by Smith and Waterman (1981), as enhanced by Gotoh (1982), was used. Two sequences are $A = a_1 a_2 ... a_M$ and $B = b_1 b_2 ... b_N$. A weight $d(a_m, b_n)$ is given to an aligned pair of residues a_m and b_n. Usually $d(a_m, b_n) <= 0$ if $a_m = b_n$, and $d(a_m, b_n) > 0$ if $a_m != b_n$. These definitions come from Gotoh (1982), and they are equivalent to original definitions from Smith and Waterman(1981). Here is Gotoh's contribution: If the gap of length k has penalty $w_k = uk + v\ (u>=0, v>=0)$[1], Gotoh proves that the W-S algorithm can run in MN steps. The distance matrix $D_{m,n}$ has the following induction form

$$D_{m,n} = \mathrm{Min}[D_{m-1,n-1} + d(a_m + b_n), P_{m,n}, Q_{m,n}]$$

$$P_{m,n} = \mathrm{Min}[D_{m-1,n} + w_1, P_{m-1,n} + u]$$

$$Q_{m,n} = \mathrm{Min}[D_{m,n-1} + w_1, Q_{m,n-1} + u]$$

4.0 Implementation

With Gotoh's improvement, the value of $D_{m,n}$ only depends on value of 3 other cells, namely $D_{m-1,n-1}$, $D_{m,n-1}$ *and* $D_{m-1,n}$. In this section, we exploit this locality property to achieve a high speed implementation in VLSI technology.

1. Gap opening penalty = $w_1 = u+v$. Gap extension penalty is $w_k - w_{k-1} = v$.

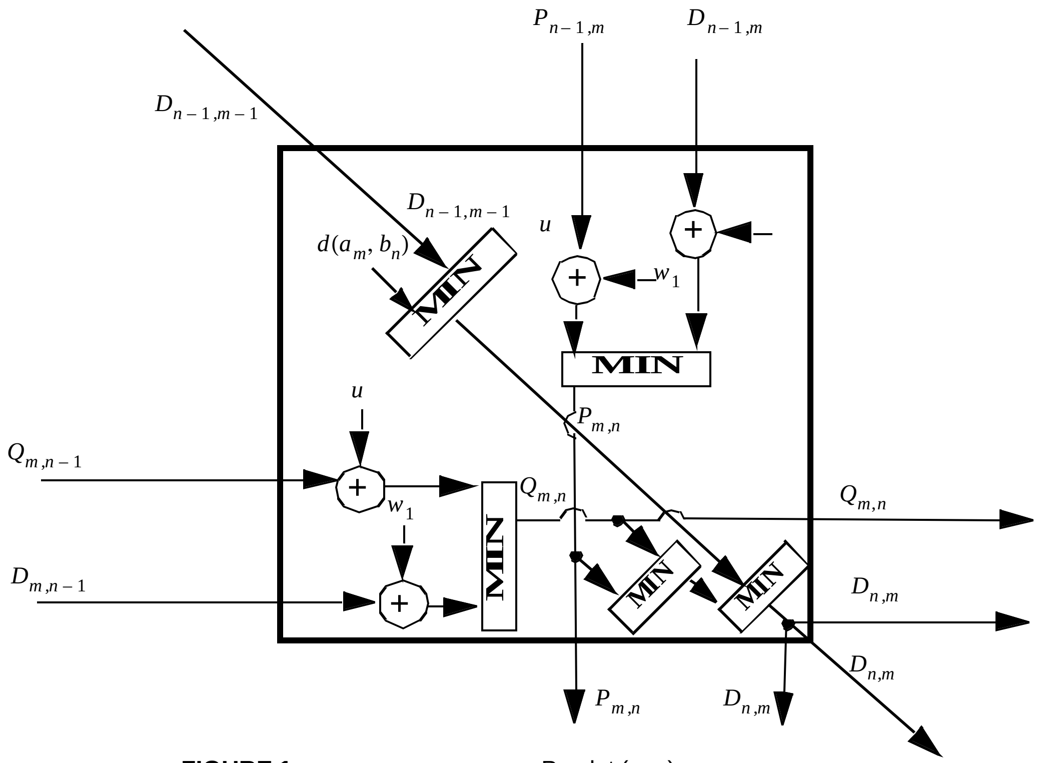

FIGURE 1. Proclet (m,n)

A small processing element called Proclet is designed as shown in Figure1 on page4. Note that each Proclet contains only simple combinatoric circuits such as adders and comparators (for finding minimum of two values). Proclet (m,n) is responsible for calculating the value of $D_{m,n}$. Proclet(m,n) has a number of input and output signals. All of these signals connects only to its nearest neighbors in an array.

TABLE 1. Input and Output Signals of Proclet (m,n)

	Input/Output	From/to Which Neighbor?
$D_{m,n-1}$	Input	From left neighbor - Proclet (m, n-1)
$Q_{m,n-1}$	Input	
$D_{m-1,n}$	Input	From upper neighbor - Proclet (m-1, n)
$P_{m-1,n}$	Input	
$D_{m-1,n-1}$	Input	From upper-left neighbor - Proclet (m-1, n-1)
$D_{m,n1}$	Output	To right neighbor - Proclet (m, n+1)
$Q_{m,n1}$	Output	

TABLE 1. Input and Output Signals of Proclet (m,n)

	Input/Output	From/to Which Neighbor?
$D_{m1,n}$	Output	To lower neighbor - Proclet (m+1, n)
$P_{m1,n}$	Output	
$D_{m1,n1}$	Output	To lower-right neighbor - Proclet (m+1, n+1)

A 2-dimension array of Proclets are placed on the silicon die. A 4x4 example is shown in Figure2 on page5. The dimension of the array has to be at least M and N, where M, N are the dimensions of the sequences to be aligned. In later section, we will discuss how to circumvent this limitation. For now, let us assume that the dimension of the Proclet array exceeds that of the 2 sequences to be aligned.

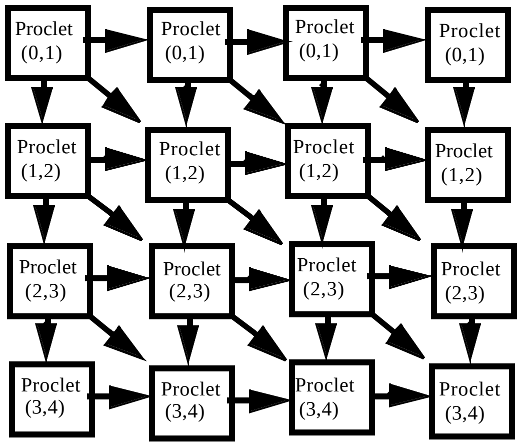

FIGURE 2. A 4x4 array of Proclets

To visualize operation of the Proclet array, one can think in terms of a wavefront. To begin, the wavefront constitutes Proclet(1, 1). Proclet(1, 1) calculates the value of $D_{1,1}$. Then the value of $D_{1,1}$, $P_{1,1}$, $Q_{1,1}$ are passed from Proclet (1,1) to Proclet(2,1) and Proclet (1,2), using the wires that connects neighboring Proclets. This enables Proclet (2,1) and Proclet(1,2) to calculate the value of $D_{2,1}$ and $D_{1,2}$ respectively. The wavefront has just propagated, and now constitutes of Proclet(2,1) and Proclet(1,2).

Next step, Proclet(2,1) and Proclet(1,2) provides $D_{2,1}$, $P_{2,1}$, $Q_{2,1}$, $D_{1,2}$, $P_{1,2}$, $Q_{1,2}$ for Proclet(3,1), Proclet (2,2) and Proclet (1,3) via the wires connecting neighboring Proclets. This allows Proclet(3,1), Proclet (2,2) and Proclet (1,3) to calculate $D_{3,1}$, $D_{2,2}$ and $D_{1,3}$.. The wavefront has just propagated again, and constitutes now of Proclet(3,1), Proclet(2,2) and Proclet(1,3).

In this manner, the wavefront propagates through the array. When it is done propagation through all Proclets, it has finished updating $D_{m,n}$ for every cell in the array.

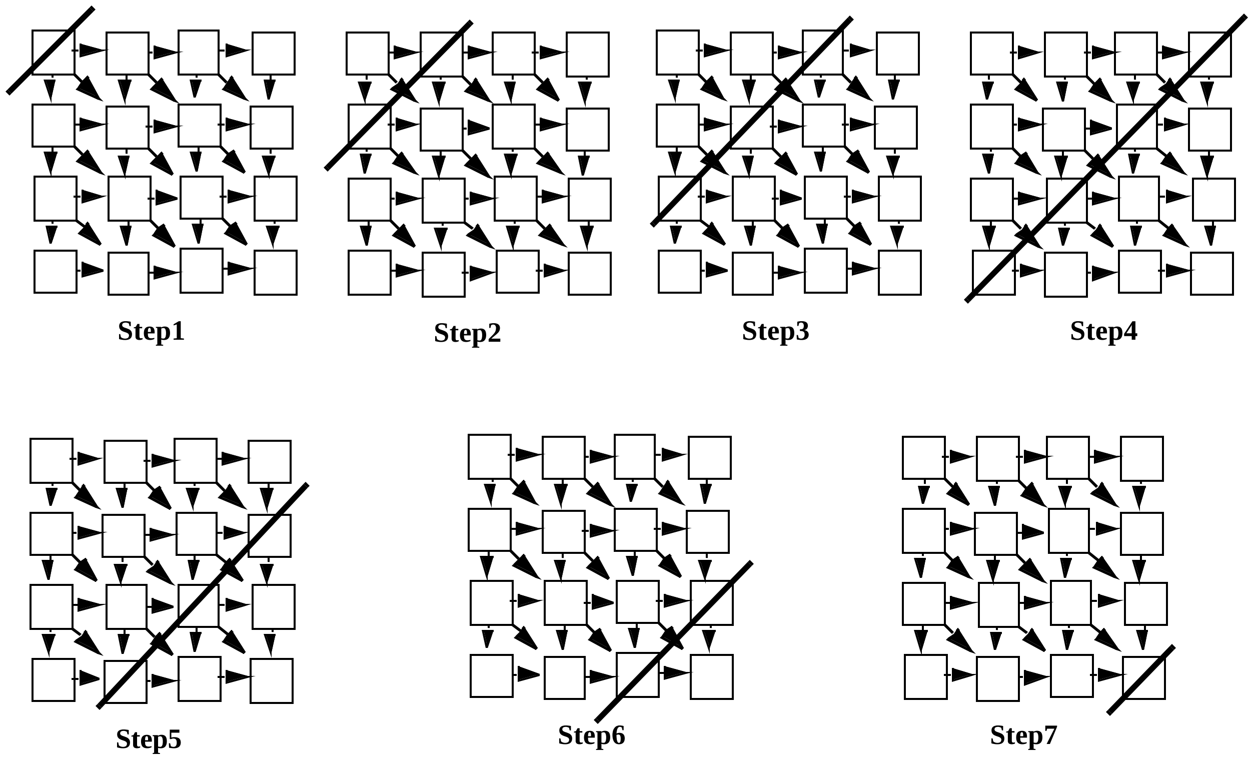

FIGURE 3. Wavefront Propagation in a 4x4 Proclet Array

5.0 Performance

We claim that this implementation provides a very fast way to update $D_{m,n}$ with current ASIC technology. The speed limit of general logic circuits is determined by propagation delay of the logic elements (i.e. adders and comparators) and also propagation delay of the wires connecting these logic elements. We use Figure4, “Propagation delays,” on page 7 as an illustration.

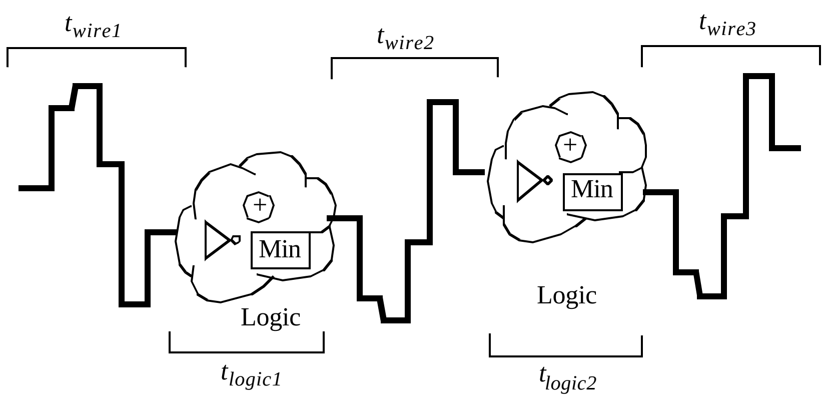

FIGURE 4. Propagation delays

We place the Proclets on the silicon, paying attention to locality of the terminals of the wires. Naturally we want to place neighboring Proclets close together on the silicon to minimize the length of the wires. In fact the placement scheme of Figure2, "A 4x4 array of Proclets," on page 5 is optimal. This way the wire between neighboring Proclet is made very short. The wires are made short so that its propagation delay is negligible compared with the propagation delay of the combinatoric logic.

The following table estimates the number of sequence alignment per second. .

TABLE 2. Proclet circuit propagation delay estimate

		Target value
(a)	Datapath width[a]	8bits [b]
(b)	Critical Timing Path[c]	ADD - MIN- MIN- MIN
(c)	Critical Timing Path Delay	500ps[d] * 4 = 2ns
(d)	Array Size	512 by 512
(e)	Each match takes	(c) * (d) = 512 *2ns = 1024ns
(f)	Performance	1/(e) = 976 K sequence alignments per second

a. Datapath width refers to the bit-width for values of $D_{m,n}$,, $P_{m,n}$ and $Q_{m,n}$

b. Rognes (2000) also uses a 8bit value.

c. In Figure5, "Critical Path of Proclet (m,n)," on page8, one of the critical timing path is highlighted in dark. Clearly, there are a few symmetric path, all of which are equally critical. Only one is highlighted.

d. assume IBM CU-11 0.13um process

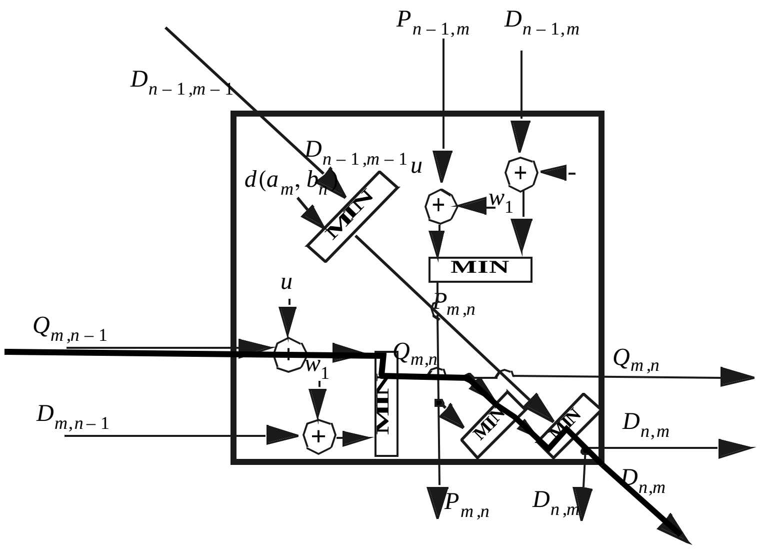

FIGURE 5. Critical Path of Proclet (m,n)

6.0 Future Work

We describes two concepts with potential to enhance performance.

6.1 Scale to larger M and N

In the baseline implementation described above, the length of the sequences to be aligned are restricted to the size of the Proclet array. Such restriction can be circumvented with a slight modification.

For example, we build a 4x4 Proclet array. To align two 16-element long sequences, we first apply the 4x4 Proclet array to the sub-sequences $\{a_1\ a_2\ a_3\ a_4\}$ and $\{b_1\ b_2\ b_3\ b_4\}$. Then we apply the Proclet array to the sub-sequences $\{a_1\ a_2\ a_3\ a_4\}$ and $\{b_5\ b_6\ b_7\ b_8\}$. After that, we apply the Proclet array to the pair of sub-sequences $\{a_5\ a_6\ a_7\ a_8\}$ and $\{b_1\ b_2\ b_3\ b_4\}$. This concept is illustrated in Figure6 on page9.

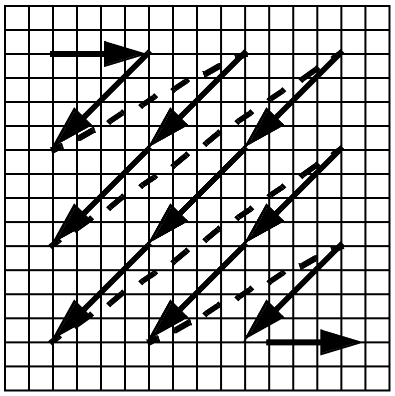

FIGURE 6. Using a 4x4 Proclet Array to compare two 16-long sequences

6.2 Pipeline Implementation

Note that in Figure3 on page6, only a small fraction of the Proclet in the array are used at any one time. With pipelining, we can fully utilize all Proclets to execute multiple comparisons simultaneously. This is especially useful when querying one sequence against a large set of sequences from a database.

Let's assume there is a large number of sequence pairs to be compared. We call the sequence pairs (A_1, B_1), (A_2, B_2), (A_3, B_3) , (A_4, B_4) ... In step 1, Proclet (1,1) works on (A_1, B_1). Then in step 2, Proclet (1,2) and Proclet (2,1) work on (A_1, B_1), exactly as described in Section 4.0. However, instead of allowing Proclet (1,1) to go idle, Proclet (1,1) work on a new sequence pair (A_2, B_2).

This concept is illustrated in Figure7 on page10.

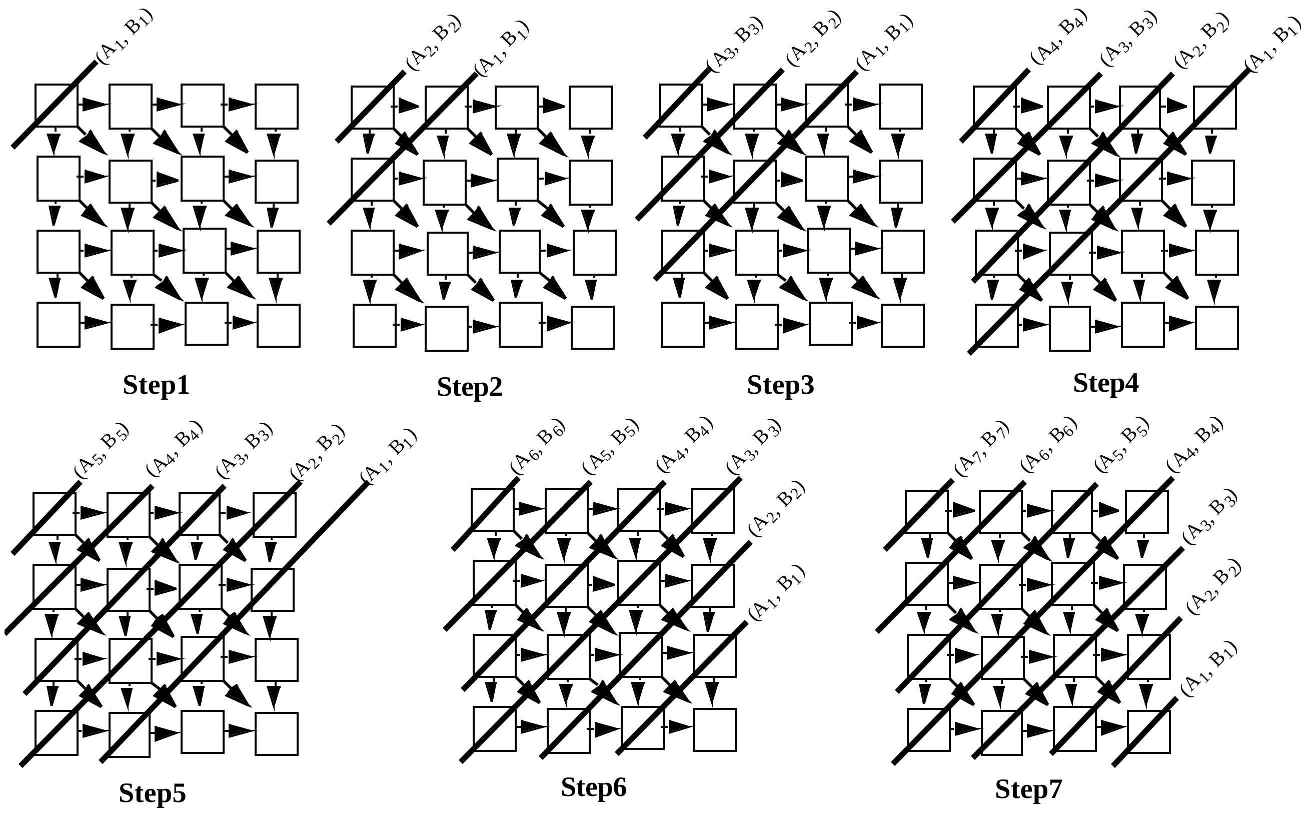

FIGURE 7. Pipeline Technique

7.0 Conclusion

We have presented a novel implementation of the Smith-Waterman algorithm. The approach is well-suited for customized VLSI with current fabrication technology. Performance of the new approach is estimated using actual data from the IBM ASIC process.

8.0 Reference

1. Benson, D.A., Karsch-Mizrachi, I,. Lipman, D.J., Ostell, J., Rapp, B.A. and Wheeler, D.L. (2000) Genbank. *Nucleic Acids Res.*, **28**, 15-18
2. Galper, A.R. and Brutlag, D.L. (1990) Parallel Similarity Search and Alignment with the Dynamic Programming Method, *provided by Professor Brutlag*
3. Gotoh, O. (1982) An improved algorithm for matching biological sequences *J. Mol. Biol.*, **162**, 705-708
4. Rognes, T. and SeeBerg E. (2000) Six-fold speedup of Smith-Waterman sequence database searches using parallel processing on common microprocessors, *Bioinformatics,* Vol.16, no. 8, 699-706

5. Smith, T.F. and Waterman, M.S. (1981) Identification of common molecular subsequences, *J. Mol. Biol*., **147**, 195-197

6. Advanced Cu-11 ASIC Databook (2000), *IBM Microelectronics*