Massively Parallel Biosequence Analysis

Richard Hughey
Computer Engineering Board of Studies
University of California, Santa Cruz
rph@ce.ucsc.edu
(408) 459-2939

Technical Report UCSC-CRL-93-14
April 2, 1993

Abstract

Massive parallelism is required for the analysis of the rapidly growing biosequence databases. First, this paper compares and benchmarks methods for dynamic programming sequence analysis on several parallel platforms. Next, a new hidden Markov model method and its implementation on several parallel machines is discussed. Finally, the results of a series of experiments using this massively parallel implementation are described.

Keywords: Computational biology, dynamic programming, parallel algorithms, systolic co-processors, hidden Markov models.

1 Introduction

The goal of the Human Genome Project is to decode and understand human genetic information [4, 10, 29]. Biosequence databases currently contain on the order of 10^8 characters in 10^5 full and fragmentary biosequences; massive parallelism is required to fully analyze this vast, rapidly growing store of information.

Biologically, the DNA (deoxyribonucleic acid) located in a cell encodes the structure of an organism in thousands or millions of nucleotides — three billion in the case of Homo sapiens. RNA (ribonucleic acid) molecules are generally shorter, containing up to a few thousand nucleotides. A nucleic acid sequence may encode one or more amino acid sequences, each sequence being a protein. Each set of three consecutive nucleotides specifies an amino acid to be added to the protein. The proteins are generated from the DNA with a complex chemical decryption algorithm involving the RNA. Proteins tend to be about 1000 amino acids long.

There are many tasks that the computational biologist would like to perform on biosequences. A few that are relevant to this discussion include:

Sequence comparison Determine how similar, by some biologically-significant comparison function, two sequences are. This can suggest sequences for further study or

analysis. A basic dynamic programming sequence comparison method, described below, requires $O(N^2)$ time, where N is the length of the sequences.

Sequence classification Given a sequence determined in the lab, decide its type, for example whether or not it is a globin such as hemoglobin.

Sequence alignment Find the best alignment between two sequences. That is, which parts of one sequence correspond to which parts of the other. If several important regions of the first have been experimentally determined, sequence alignment will ideally locate those regions in other sequences. Sequence alignment information can be extracted from most sequence comparison algorithms. Regions of the sequences that vary little between sequences or organisms are referred to as "conserved" regions.

Multiple sequence alignment Find the best alignment among a group of sequences [5]. This differs considerably from pairwise sequence alignment, and can require $O(N^M)$ time for M sequences, if straight-forward dynamic programming is used. Thus, heuristics using pairwise alignments are more commonly used. Multiple alignments can be instrumental for creating informed guesses about sequence function.

Subsequence variations The above problems can also be applied to finding and aligning the most similar length-ℓ subsequences of two or more sequences, or finding an optimal value for ℓ under some criterion.

An ultimate goal of biosequence analysis (with shades of El Dorado) is to determine a molecules' 3-dimensional structure from its 1-dimensional sequence of nucleotides or amino acids. A somewhat more practical task is to determine the secondary, or 2-dimensional, structure of a biosequence. While secondary structure is a simpler energy minimization problem than tertiary, the complexity of the energy functions and length of the molecules have steered research toward neural network pattern recognition of sequence segments that form particular structures. Current results, however, have only achieved around 60% accuracy in prediction [7]. Multiple sequence alignments can form a starting point for 2-D and 3-D structure prediction when crystallographic data is available for one or more of the aligned sequences.

1.1 Biosequence analysis methods

Perhaps the most commonly desired analysis is the determination of the similarity between two biosequences. Computational biologists have several metrics for comparison which involve different costs between nucleotides as well as the possible use of affine costs or gap penalties to indicate an added penalty for commencing a series of deletions or insertions in the matching [3, 5, 20, 26, 28]. Exact sequence comparison (with or without gaps) is an $O(n^2)$ dynamic programming algorithm: on a sequential machine, time proportional to the square of the sequence length is required to solve the problem.[1]

[1] Masek and Paterson have developed an $O(\frac{n^2}{\log n})$ algorithm for strings of equal length from a finite alphabet with a minor restriction on the cost function. It will be faster for values of n greater than 263 000, and is not amenable to parallelization [23].

Distance calculation is governed by a simple recurrence. The cost of transforming a string a into another string b, assuming for convenience that both are of length n, is the solution $d_{n,n}$ of the recurrence:

$$\begin{aligned}
d_{0,0} &= 0 \\
d_{i,0} &= d_{i-1,0} + \mathrm{dist}(a_i, \phi) \\
d_{0,j} &= d_{0,j-1} + \mathrm{dist}(\phi, b_j) \\
d_{i,j} &= \min \begin{cases} d_{i-1,j-1} + \mathrm{dist}(a_i, b_j) \\ d_{i-1,j} + \mathrm{dist}(a_i, \phi) \\ d_{i,j-1} + \mathrm{dist}(\phi, b_j), \end{cases}
\end{aligned}$$

where ϕ is the null character, and $\mathrm{dist}(a_i, \phi)$ is the cost of not matching a_i to any character in b. Edit distance, the number of insertions or deletions required to change one sequence to another, can be calculated by setting $\mathrm{dist}(a_i, \phi) = \mathrm{dist}(\phi, b_j) = 1$, and $\mathrm{dist}(a_i, b_j) = 0$ when the two characters match, and 2 otherwise.

The recurrence can be efficiently mapped to a parallel processor in several ways, perhaps the most obvious being to assign the i-th column of the dynamic programming matrix, as well as a_i, to processor PE_i. Comparison with gap penalties involves three recurrences of a similar form.

An efficient and widely used statistical heuristic for sequence analysis has been developed by Altschul and colleagues in implemented in the BLAST program [1]. The governing (simple) recurrence is:

$$\begin{aligned}
d_{0,i} &= d_{j,0} = 0 \\
d_{i,j} &= d_{i-1,j-1} + \mathrm{dist}(a_i, b_j).
\end{aligned}$$

The dist values are a table that reflects the evolutionary cost in mutating one amino acid to another (for example, variations on Dayhoff's matrix [9]). This method reduces the cell calculation from 3 minimizations and additions (9 with gaps) to a single addition. Statistical methods are then used to calculate, based on the $d_{i,n}$ diagonal point-mutation scores, a good alignment between sequences. The methods can include a window parameter, roughly corresponding to fixing a maximum gap length.

1.2 Hardware for biosequence analysis

Several single-purpose and specialized hardware systems have been built to address biosequence analysis problems. Four recent directions are illustrated by BioScan, BISP, B-SYS, and PAM and Splash.

BioScan is a massively parallel VLSI implementation of the BLAST algorithm [27]. BioScan chips are fabricated in 1.2 μm CMOS, contain 812 PEs (537 000 transistors), and run at 32 MHz. This chips can accept a character every 17 clocks, or 1.88 million characters per second (1.88 MCPS), though the actual system is slower. A working system of 16 chips (12 992 PEs) has been built, and an Internet server interface is currently under development. The system is preloaded with a weight table scaled according to sequence length before

performing the comparison. Internally, BioScan has 16-bit words and uses an infinity value to prevent overflow and minimum value (0) to prevent underflow.

BISP takes a different approach, implementing in hardware full dynamic programming with gaps [6]. Each 400 000-transistor, 1 μm CMOS chip contains 16 sequence comparison PEs. Each PE has about 35 16-bit registers, comparators, and multiplexors, in addition to local random access memory (RAM) for the storage of cost tables. About 20% of each chip is devoted to controlling the PEs. As with BioScan, sentinels prevent overflow and underflow. Although it operates with an 80 ns clock (one clock for each dynamic programming cell update), its limiting factor will be the 3 Mbyte/s data transfer rate to the host.

B-SYS is a general-purpose systolic array that, although optimized for a variety of sequence comparison algorithms, can perform several other functions such as text compression and decompression [17]. With its less aggressive implementation, each 85 000-transistor 2 μm CMOS chip features 47 PEs in a linear array. Each 8-bit ALU shares 16 registers with each of its two neighbors, enabling shared data and zero-overhead communication. The chips can execute instructions at about 4 MHz, and the basic sequence comparison operation requires 6–25 instructions (660–160 kCPS), depending on algorithm. Although sentinels can be programmed, typically modulo sequence comparison, in which dynamic programming values are compared modulo 256, is used [21, 16]. As with BISP, B-SYS can implement dynamic programming with gap penalties. A reimplimentation on the order of BISP or BioScan could place 256 PEs on each chip, and be clocked 3–6 times faster.

Splash and PAM are field-programmable gate array systems specifically designed for configuration as special-purpose co-processors [12, 2, 15]. Thus, they provide for the implementation of sequence comparison algorithms in hardware without the need to fabricate new chips for each algorithm. Programming specific algorithms is, unfortunately, time consuming. An edit-distance calculation with fixed costs and no gaps required nearly 3000 lines of code, and placed 30 PEs on each chip (the implemented algorithm required $2N - 1$ PEs to compare strings of length N over a 4-character alphabet). Implementation for a larger alphabet (the proteins), biologically-significant cost functions (at least a 6 bits), and gap penalties would severely decrease processor density and, as a result, performance. However, the effort in programming these systems is significantly lower than designing a VLSI chip, and they are general-purpose, able to perform a large number of functions.

Table 1 summarizes the performance of several of these machines. The number of PEs, maximum native sequence length, and the approximate number of chips are listed for each machine. Performance metrics include 100×100 sequence comparison, a metric first used with the nMOS P-NAC system [22]; the maximum attained or attainable number of dynamic programming cell updates per second (CUPS), generally not for the 100×100 problem; and the CUPS per chip, a rough measurement of efficiency or performance per dollar. B-SYS* figures are estimates of a 64-chip version (using the 47-PE chip) with a more sophisticated interface between host and co-processor (the prototype was built on an ISA card, and each instruction required 3 writes over the 8 MHz bus). The BISP times are derived from published system performance estimates. In the 100×100 column, BISP and B-SYS* are assumed to be able to perform several comparisons at once, breaking each comparison at a chip boundary, as the BioScan system enables (this handicaps BioScan, as 712 PEs are not used). All others are based on personal experimentation or extrapolation from published

Machine		Year	Chips	PEs	Max N	100×100 (s)	CUPS	CUPS/chip
Cray 2	[12]	85	—	1	—	6.5	153 k	
Sun 4/50		90	—	1	—	0.75	1333 k	
CM-2		87	3074	16384	—	0.17	10 M	2 k
MP-1		90	512	8192	—	0.031	55 M	107 k
P-NAC	[22]	87	9	270	134	0.91	1471 k	163 k
Splash	[12]	90	64	248	123	0.020	50 M	781 k
B-SYS	[17]	90	10	470	470	0.351	13 M	1300 k
B-SYS* (est)	[17]	90	64	3008	3008	0.002	1600 M	25 M
BISP (est)	[6]	91	64	1024	1024	0.0003	3072 M	192 M
BioScan	[27]	91	16	12992	12992	0.0004	25 G	1583 M

Table 1: Edit distance calculation (arranged by CUPS/chip).

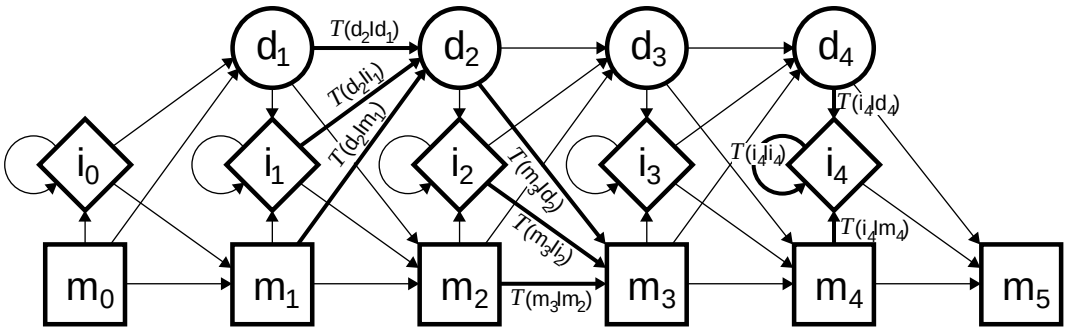

Figure 1: Hidden Markov model for protein comparison and alignment [13].

experimentation. Note that the table is for the simplest dynamic programming edit-distance calculation. BioScan implements a different algorithm on a larger alphabet, and BISP can do gap comparison in the same amount of time (P-NAC cannot, Splash would require a new configuration, and B-SYS would slow down by a factor of 4). Jones reports 75 MCUPS on a 64K CM-2 (corresponding to two times faster than the results in the table) by microcoding the inner loop of the dynamic programming algorithm [19]; Jones has also presented methods for database pattern searching with limited gap length on the CM-2 [18]. By making full use of modulo sequence comparison to reduce data communication, this author estimates that a factor of 2–4 performance increase is attainable over the CM-2 and MP-1 results of Table 1 for long sequences [16]. Core has compared dynamic programming and the BLAST algorithm on the CM-1 and Intel iPSC hypercube computer [8].

2 Hidden Markov model sequence analysis

An evolving and exciting means of biosequence analysis developed at UCSC uses a hidden Markov model (HMM) to generate position-dependent cost tables for each amino acid.

Given a set of biologically-similar sequences (such as the globins), the statistical model can be iteratively trained to become a close-as-possible match to *all* the sequences. Thus, the HMM is a probabilistic consensus sequence that contains a probability distribution over all amino acids at each node.

The protein HMM is displayed in Figure 1 (the reader is referred to the tutorial by Rabiner for an introduction to HMMs [25]). The squares are match states, corresponding to the matching of the model state to a character from the input sequence. The diamonds are insertion states, corresponding to characters in the sequence that are not matched. The circles are delete states, corresponding to skipping over a model node when aligning the sequence to the model. The connecting arrows are transition probabilities between nodes. The zeroth state is a special begin state, and the last is a special end state (real models are much longer than the one shown in the figure). In calculating the distance between the sequence and the model, one calculates the probability that the HMM could generate the given sequence. In aligning a sequence to a model, the most probable generation of that sequence is assumed.

For information on training and updating the HMM, the reader is referred to the UCSC technical report and the related, condensed conference paper [13, 14]. Suffice to say, the vast majority of training time is spent calculating the probability of entering each state with each character of a training sequence. This is solved two-step forward-backward dynamic programming algorithm implemented on a massively parallel array processor.

3 Massively Parallel Implementation

At their simplest level, the Maspar MP-1 and MP-2 computers are array processors with an 8-nearest-neighbor mesh connection and wraparound at the edges [24]. Each PE has 40 32-bit registers, 64 kbytes of locally addressable memory, and a bit-parallel ALU (the MP-1 has 32 4-bit PEs per chip, while the MP-2 has 32 32-bit PEs per chip). Each group of 16 PEs shares a connection to the global router which, as can be expected, is much slower than mesh communication. There is a single array control unit (ACU), a full-fledged processor responsible for broadcasting instructions to the array. Local memory access with a broadcast address is approximately 10 times slower than register access, while memory access with a locally generated address is 25 times slower.

The HMM evaluation algorithm is a 2-phase recursion on three variables, similar to affine sequence comparison. Throughout the algorithm, $-\log$ probabilities are used to reduce multiplication to addition, and a per-PE table of 7600 integers speeds the addition of probabilities.

In training a model, the model is first compared to every sequence in the training set. This is done by loading the model into the MP-1, one model position per PE, as many times as possible, using recursive doubling.[2] Next, a sequence is downloaded to the first processing element of each model via the DMA channel between the host and the PE array, and then the algorithm is initialized. For the forward step, the sequences are stepped through the

[2]Typical experiments have model lengths of 100–500, and a maximum sequence length of up to a couple thousand.

```
for (si = 1, esi = 2-mod_len, lsi=1 ; lsi <= seq_len ; si++, esi++) {
  if (proc_num < si) {     /* shift characters through models; */
    xnetShiftE(pchar);     /* si is global loop index */
    if (esi > 0) {         /* saving in local memory */
      *ppseq++ = pchar;
    }
    if (proc_num == 0) {   /* First model node takes next char      */
        pchar = pseq[si];  /* and computes initialization value     */
        d3_i += pwentry.i_table[pchar] + (si == 1 ? pwi_d : pwi_i);
    } else {               /* compute prob of entering D, M, and I states */
      d3_x1 =
        p_sum3logX (d1_d + pwd_d,d1_m + pwd_m,d1_i + pwd_i,t1,t2,t3);
      d3_x2 = pwentry.m_table[pchar] +
        p_sum3logX (d2_d + pwm_d,d2_m + pwm_m, d2_i + pwm_i,t1,t2,t3);
      d3_i  = pwentry.i_table[pchar] +
        p_sum3logX (d3_d + pwi_d,d3_m + pwi_m,d3_i + pwi_i,t1,t2,t3);
    }                      /* Store dynamic programming values */
    *pcx++ = d3_d = d3_x1; /* in local memory                  */
    *pcx++ = d3_m = d3_x2;
    *pcx++ = d3_i;
    all {                  /* Shift costs to adjacent PEs      */
      d2_i = d1_i; d2_d = d1_d; d2_m = d1_m;
      d1_i = d3_i; d1_m = d3_m; d1_d = d3_d;
      xnetShiftE (d1_i); xnetShiftE (d1_m); xnetShiftE (d1_d);
    }
    lsi++;               /* lsi is parallel (sequence- and model-specific) */
  }                      /* that controls loop execution in the PE array   */
}
```

Figure 2: MPL code for forward computation.

model (using nearest-neighbor connections) to evaluate and store the forward values, and the sequence is collected again at the end PE. Next, the sequence is sent back through the array, and the probability of being in each model state given the entire sequence is calculated.

To update the model, the average probability distribution over all training sequences for each model node is required. These are generated in the array in parallel.

Throughout this operation, the transition probabilities and as much other data as possible are kept in the 40 local PE registers. The main loop of the forward calculation is shown in Figure 2.

A typical run of the training algorithm requires 20—60 reestimation cycles, up to a few billion cell update operations, and the process of finding a *good* model can require ten, one hundred, or more experiments. The training algorithm has a large number of real-valued parameters (heuristic parameters that encourage the model to use match states over insertions or deletions and govern the truncation and growing of the model, default amino acid distributions, default transition probabilities, confidence in the default distributions, and so on), and work is still underway to determine the best settings. Additionally, the annealing schedule is currently quite primitive, and results are sensitive to the random seed. It was

System	Rel. Porting Difficulty	Performance (kCUPS)	Performance (Sun 4/50s)
Sun 3/110	0	3.2	0.1
Sun 4/50	0	37.1	1.0
Decstation 5000/240	0	39.2	1.1
SGI 440VGX, 1 CPU	0	59.0	1.6
Dec Alpha 3000/500	0	107	2.9
C-Linda, 7 Decstation 5000s (240 and 125)	4	147	4.0
Cray Y-MP, 1 CPU, vectorized	8	167	4.5
8K MP-1, unoptimized	30	821	22
Cray Y-MP, 8 CPU, estimated	20	1300	35
8K MP-1, optimized	60	1530	41
4K MP-2, optimized	60	1580	43
16K MP-2, optimized	60	5100	140

Table 2: HMM training on various machines.

obviously impossible to experiment extensively with the parameters on workstations requiring one day of CPU time to perform a single run; the massively parallel implementation has enabled considerable new research.

4 Evaluation

In addition to serial machines, the HMM software has also been ported to a Cray Y/MP and C-Linda. Table 2 tabulates, for a variety of machines, two metrics of performance: HMM CUPS (note that a cell update involves more computation for HMM training than for edit-distance calculation) and speedup relative to a Sun 4/50 (Sparc-2), as well as a rough, dimensionless measure of the difficulty of adapting the code to each platform. Several interesting observations can be drawn from the table. First, there is a great difference in both performance and effort between a working array processor program (MP-1, unoptimized) and an efficient array processor program (MP-1, optimized).[3] Second, conversion to different architectures takes time. For the Maspar version, the entire dynamic programming routine was rewritten. The Cray case was simpler: the loop indices were modified to allow vectorization. The Linda version was the simplest, taking advantage of the course-grain parallelism available in training the model with several hundred sequences. The CM-2 results are estimated from a partial implementation of the dynamic programming operation; performance was severely handicapped by the lack of local addressing.

[3]The MP-2 features an ALU approximately eight times faster than the MP-1's. The bandwidth to each PEs local memory, which doubled between the two models, is the limiting factor.

Lengths	Number	Cum. Distrib.
12–79	2	0.02
206–297	6	0.07
393–499	55	0.67
614–761	21	0.91
830–858	6	0.98
1044–49	2	1.00

Table 3: Elongation factor training set statistics.

5 Biological Results

To evaluate the effectiveness of the massively parallel implementation (as well as gain familiarity with parts of the system outside the dynamic programming inner loop), the author has been studying elongation factors, a biologically interesting protein structure [11]. One particularly important aid has been the availability of a structural alignment of 3 members of the class from X-ray crystallographic data. This alignment provides a "sanity check" on the trained HMMs.

Another interesting feature of the elongation factors is the large variations in the length of the protein sequences, seen in Table 3. (The two shortest, 12 and 79 amino acids, were fragments and eliminated from the training set.) The conserved region of the sequences is several hundred amino acids long, and models of 400 to 500 positions worked best. Unlike earlier experiments at UCSC, these sequences were not clipped to the region of importance. Instead, free insertion modules (FIMs), allowing low-penalty insertions, were prepended and appended to the HMM before training to convert to a subsequence modeling program. The HMM rapidly converged to an alignment of the conserved region.

A large number of experiments, each with several random seeds, was performed. The most important parameter for generating a good HMM was the model length (the best alignment had a model length of 401), though other parameters were also varied. In all, about one hundred experiments were performed, each requiring around 5 minutes of 16K MP-1 CPU time (equivalent to about 5 hours on Sparc-2 workstation). Interesting models were first identified by distance statistics between the training set and the trained HMM (average, maximum, and sample deviation). The three structurally aligned sequences (SELB, Ef-Tu, and FIEC2 [11]) were then aligned to the model, and the results compared to the true alignment (automation of this processes is currently under development). This weeded out many models, eventually leaving a model that produces quite good multiple alignments between elongation factor regions and can also be used to identify elongation factors in the protein sequence databases. The multiple alignment of the three test sequences and one additional sequence is shown in Figure 3. Note that the third test sequence has 394 amino acids in an insertion state before its elongation factor region begins. The first four alignment rows correspond to and are virtually identical to the structural alignment reported by Forchhammer, Leinfelder, and Böck [11].

Figure 3: Alignment of four proteins with elongation factors.

6 Acknowledgments

This work has been supported in part by funds granted by the Division of Natural Sciences of the University of California, Santa Cruz. Jon Becher and Maspar Corporation generously provided time on several Maspar computers. Anders Krogh's help in understanding the protein code was invaluable. Charlie McDowell ported the code to C-Linda. Anne Urban implemented the inner loop on the CM-2; CM-2 time was generously provided by Thinking Machines Corporation and the Brown University Department of Computer Science. Saira Mian identified the elongation factors for study and provided biological insight into their analysis.

References

[1] S. F. Altschul *et al.*, "Basic local alignment search tool," *J. Mol. Biol.*, vol. 215, pp. 403–410, 1990.

[2] P. Bertin, D. Roncin, and J. Vuillemin, "Introduction to programmable active memories," Tech. Rep. 3, Digital Paris Research Laboratory, Rueil Malmaison, France, June 1989.

[3] M. J. Bishop and C. J. Rawlings, eds., *Nucleic Acid and Protein Sequence Analysis*. Oxford, England: IRL Press, 1987.

[4] C. R. Cantor, "Orchestrating the Humane Genome Project," *Science*, vol. 248, pp. 49–51, 6 Apr. 1990.

[5] S. C. Chan, A. K. C. Wong, and D. K. Y. Chiu, "A survey of multiple sequence comparison methods," *Bul. of Math. Biology*, vol. 54, no. 4, pp. 563–598, 1992.

[6] E. Chow, T. Hunkapiller, J. Peterson, and M. S. Waterman, "Biological information signal processor," in *Proc. Int. Conf. Application Specific Array Processors* (M. Valero *et al.*, eds.), pp. 144–160, IEEE Computer Society, Sept. 1991.

[7] B. I. Cohen and F. E. Cohen, "Predictions of protein secondary and teriary structure," in *Biocomputing: Genome Sequence Analysis* (D. Smith, ed.), San Diego, CA: Academic Press, 1992.

[8] N. G. Core, E. W. Edmiston, J. H. Saltz, and R. M. Smith, "Supercomputers and biological sequence comparison algorithms," *Computers and Biomedical Research*, vol. 22, pp. 497–515, 1989.

[9] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, "A model of evolutionary change in proteins," in *Atlas of Protein Sequence and Structure*, ch. 22, pp. 345–358, Washington, D. C.: National Biomedical Research Foundation, 1978.

[10] C. DeLisi, "Computers in molecular biology: Current applications and emerging trends," *Science*, vol. 246, pp. 47–51, 6 Apr. 1988.

[11] K. Forchhammer, W. Leinfelder, and A. Böck, "Identification of a novel translation factor necessary for the incorporation of selenocysteine into protein," *Nature*, vol. 342, pp. 453–456, 23 Nov. 1989.

[12] M. Gokhale *et al.*, "Building and using a highly parallel programmable logic array," *Computer*, vol. 24, pp. 81–89, Jan. 1991.

[13] D. Haussler, A. Krogh, S. Mian, and K. Sjölander, "Protein modelling using hidden Markov models: Analysis of globins," Tech. Rep. UCSC-CRL-92-23, University of California, Santa Cruz, CA, Sept. 1992.

[14] D. Haussler, A. Krogh, S. Mian, and K. Sjölander, "Protein modelling using hidden Markov models: Analysis of globins," in *Proc. Hawaii Int. Conf. System Sciences*, Jan. 1993.

[15] D. T. Hoang, "A systolic array for the sequence alignment problem," Tech. Rep. CS-92-22, Dept. Computer Science, Brown University, Providence, RI, Apr. 1992.

[16] R. Hughey, *Programmable Systolic Arrays*. PhD thesis, Dept. Computer Science, Brown University, Providence, RI, 1991. Tech. Rep. CS-91-34.

[17] R. Hughey and D. P. Lopresti, "A software approach to fault detection on programmable systolic arrays," in *Proc. Symp. Parallel and Distributed Processing* (B. Shirazi and H. Sudborough, eds.), pp. 523–526, IEEE Computer Society, Dec. 1990.

[18] R. Jones, "Sequence pattern matching on a massively parallel computer," *CABIOS*, vol. 8, no. 4, pp. 377–383, 1992.

[19] R. Jones *et al.*, "Protein sequence comparison on the Connection Machine CM-2," in *Computers and DNA, SFI Studies in the Sciences of Complexity, vol. VII* (G. Bell and T. Marr, eds.), pp. 1–9, Reading, MA: Addison-Wesley, 1989.

[20] A. M. Lesk, ed., *Computational Molecular Biology*. Oxford, England: Oxford University Press, 1988.

[21] R. J. Lipton and D. P. Lopresti, "Delta transformations to simplify VLSI processor arrays for serial dynamic programming," in *Proc. Int. Conf. Parallel Processing* (K. Hwang *et al.*, eds.), pp. 917–920, CRC Press, Aug. 1986.

[22] D. P. Lopresti, "P-NAC: A systolic array for comparing nucleic acid sequences," *Computer*, vol. 20, pp. 98–99, July 1987.

[23] 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*, pp. 337–349, Reading, MA: Addison-Wesley, 1983.

[24] J. R. Nickolls, "The design of the Maspar MP-1: A cost effective massively parallel computer," in *Proc. COMPCON Spring 1990*, pp. 25–28, IEEE Computer Society, Feb. 1990.

[25] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, pp. 257–286, Feb. 1989.

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

[27] R. Singh *et al.*, "A scalable systolic multiprocessor system for analysis of biological sequences," in *Proc. Symp. on Integrated Systems,* University of Washington, Apr. 1993.

[28] G. von Heijne, *Sequence Analaysis in Molecular Biology*. San Diego, CA: Academic Press, 1987.

[29] J. D. Watson, "The Human Genome Project: Past, present, and future," *Science*, vol. 248, pp. 44–48, 6 Apr. 1990.