

Parallel Hardware for Sequence Comparison and Alignment

Richard Hughey

Computer Engineering, University of California, Santa Cruz, CA 95064

rph@cse.ucsc.edu

Abstract

Sequence comparison, a vital research tool in computational biology, is based on a simple $O(n^2)$ algorithm that easily maps to a linear array of processors. This paper reviews and compares high-performance sequence analysis on general-purpose supercomputers and single-purpose, reconfigurable, and programmable co-processors. The difficulty of comparing hardware from published performance figures is also noted.

Introduction

The vast databases produced by the Human Genome Project demand innovative tools for fast sequence and database analysis. Because sequence databases contain billions of characters it is important to locate areas of interest within a database or genome quickly.

Dynamic programming organizes sequence comparison by comparing shorter subsequences first, so their costs can be made available in a table (Figure 1) for the next longer subsequence comparisons. The final entry becomes the comparison rating. Exact sequence comparison is an $O(n^2)$ -time dynamic programming algorithm (there is an $O(n^2/\log n)$ -time version, but it has a large constant factor and is not amenable to parallelization (Masek & Paterson, 1983)). Distance calculation is governed by a simple recurrence. The cost of transforming a reference string b into another string a is the solution of a recurrence whose core is:

$$c_{i,j} = \min \begin{cases} c_{i-1,j-1} + \text{dist}(a_i, b_j) & \text{match} \\ c_{i-1,j} + \text{dist}(a_i, \phi) & \text{insert} \\ c_{i,j-1} + \text{dist}(\phi, b_j) & \text{delete,} \end{cases}$$

where $\text{dist}(a_i, b_j)$ is the cost of matching a_i to b_j , $\text{dist}(a_i, \phi)$ is the gap cost of not matching a_i to any character in b , and $\text{dist}(\phi, b_j)$ is the cost of not matching b_j to any character in a . Edit distance, the number of insertions or deletions required to change one sequence to another, can be calculated by setting $\text{dist}(a_i, \phi) = \text{dist}(\phi, b_j) = 1$, and $\text{dist}(a_i, b_j) = 0$ if $a_i = b_j$ or 2 other-

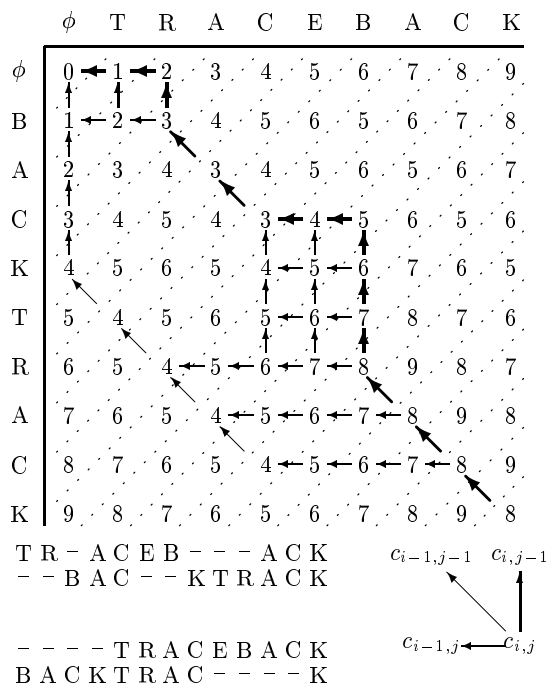


Figure 1: Dynamic programming example to find least cost edit of “BACKTRACK” into “TRACEBACK” using Sellers’ evolutionary distance metric (Sellers, 1974). Below the dynamic programming table are two possible alignments and an illustration of the data dependencies. Diagonals can be vectorized or computed in parallel.

wise. Though not often used in biology, edit distance is a common performance benchmark.

Sequence comparison using affine gap penalties involves three interconnected recurrences of a similar form (Gotoh, 1982). The extra cost for starting a sequence of insertions or deletions will, for example, make the second alignment of Figure 1 preferred over the alignment with four gaps. In the most general form of sequence comparison, a profile or linear hidden Markov model (Grib-skov *et al.*, 1990; Krogh *et al.*, 1994), all transition or gap costs (g) between the three states (match, insert, or

delete) and character costs are position-dependent:

$$c_{i,j}^M = \min(c_{i-1,j-1}^M + g^{M \rightarrow M}, c_{i-1,j-1}^I + g^{I \rightarrow M}, c_{i-1,j-1}^D + g^{D \rightarrow M}) + \text{dist}(a_i, b_j),$$

$$c_{i,j}^I = \min(c_{i-1,j}^M + g^{M \rightarrow I}, c_{i-1,j}^I + g^{I \rightarrow I}, c_{i-1,j}^D + g^{D \rightarrow I}) + \text{dist}(a_i, \phi),$$

$$c_{i,j}^D = \min(c_{i,j-1}^M + g^{M \rightarrow D}, c_{i,j-1}^I + g^{I \rightarrow D}, c_{i,j-1}^D + g^{D \rightarrow D}) + \text{dist}(\phi, b_j).$$

Local alignment (Smith & Waterman, 1981), which finds the most similar subsequences of two sequences, adds a fourth, constant term to each minimization, such as zero, representing the cost of starting the correspondence at an internal (i, j) pair, in which case highly similar regions must, for the equations above, have negative cost. To allow the correspondence to end anywhere within the dynamic programming matrix, the matrix is searched for the best final score. By changing signs, dynamic programming can be presented using similarity and maximization (more common in biology) rather than cost and minimization (more common in computer science).

The alignment of two sequences, or a mapping of which characters in one sequence correspond to which characters in the other, is also important. The easiest means of generating a sequence alignment is, for each of the $O(n^2)$ $c_{i,j}$ values, to store the choices made during minimization. The arrows of Figure 1 indicate these saved choices. There can be several equally valid alternative paths (or alignments) that produce the same score.

Another, somewhat simpler, recurrence is used by the BioSCAN co-processor (Singh *et al.*, 1993). In this $O(n^2)$ recurrence, the score along each diagonal (without insertions or deletions) is computed, and statistical analysis is used to generate a score similar to that of BLAST (Altschul *et al.*, 1990). Such searches are not as sensitive as the full affine cost computation but can be completed more quickly (Pearson, 1995b).

Algorithm

There are two basic methods of mapping sequence comparison to a parallel processor, one with coarse-grain parallelism, and the other with fine-grain parallelism.

The coarse-grain approach is appropriate for large database searches. In this case, the sequences are partitioned among the processing elements (PEs) so as to have similarly sized subsets of the database assigned to each PE. Then, multiple independent sequence analyses are performed. Each processing element must have

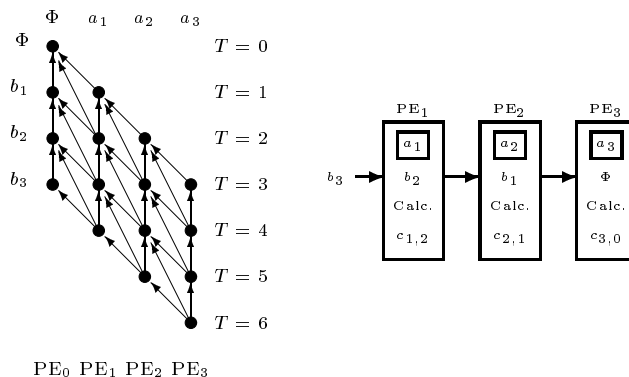


Figure 2: Mapping dynamic programming to a linear processor array. Arrows indicate dependencies between calculations; the b_3 value will travel from PE₀ at time 3 to PE₃ at time 6. On the right, the array is shown at time step 2 (PE₀ is not shown).

sufficient memory for its sequences (or significant segments of its sequences), as well as for two rows of the dynamic programming matrix: the previous row, and the row being currently formed. If only a small number of sequence comparisons are required, coarse-grain parallelism will offer little speedup because the database sequences will not be spread across all processing elements (for example, using only 100 of a CM-2's 65,536 PEs when comparing 100 sequences).

In the fine-grain approach, $O(n)$ processing elements are used to compare two sequences in $O(n + N)$ time, where N is the size of the database and n the length of the query sequence. The computation for each dynamic programming cell along a single diagonal of Figure 1 can be performed at once. A common mapping is to assign one processing element to each character of the query string, and then to shift the database through the linear chain of PEs (Figure 2). Here, n PEs are assigned to the characters in sequence a while the N characters of the database b shift through the array, one PE per time step. The final calculation of $c_{3,3}$ takes place in PE₃ at time step 6 using a_3 and b_3 , while $c_{2,3}$ and $c_{3,2}$ are computed during time step 5.

Because the BioSCAN algorithm has few dependencies (each $c_{i,j}$ value only depends on $c_{i-1,j-1}$), each row of the dynamic programming matrix can be calculated simultaneously, and the entire calculation can be completed in N time steps on n PEs. An alternative mapping for gapped sequence comparison, in which the sequences flow through the array in opposite directions, is also occasionally used because query sequence preloading is not

required.

If the array is not large enough for the complete query, the dynamic programming computation must be partitioned (Lipton & Lopresti, 1987). Fine-grain parallelism is the method of choice for the special-purpose processors, which generally have the highest performance.

For anything but a special-purpose processor, the coarse-grain approach, when a large database is being searched, will be faster, primarily because communication of intermediate results can be time consuming. Special-purpose linear processor arrays are designed to perform this communication at the same speed as the computation. Workstation clusters, general-purpose parallel machines, and even the fine-grain MasPar machines, can take a significant penalty from this communication step. The MasPar's MPsrch routine thus uses a coarse-grain approach, dividing the database among the processing elements, to gain a factor of 2 to 3 in performance over an earlier fine-grain implementation. If multiple special-purpose processors are available, both levels of parallelism can be applied.

Sequence alignment has different parallel processing needs than sequence comparison. First, there is the matter of scale: although sequence comparison against a database with millions of residues is common, sequence alignment against such a vast database is not. This is not to say that parallelization is not important for sequence alignment: the training of a profile or linear hidden Markov model, which can then be used for multiple sequence alignment or database search, has sequence alignment as part of its inner loop. Training a single model, and usually several models should be trained and the best selected, can require hundreds of alignments of each of the tens or hundreds of training sequences to the model. The SAM hidden Markov model package includes a fine-grain MasPar parallelization for training and scoring (Hughey & Krogh, 1996).

Second, the obvious version of sequence alignment requires $O(n^2)$ total space, or $O(n)$ space per PE. One solution to this is Hirschberg's divide-and-conquer approach to sequence alignment that uses only $O(n)$ total space (Hirschberg, 1975; Myers & Miller, 1988), and can be readily parallelized (Edmiston *et al.*, 1988; Huang, 1989; Ibarra *et al.*, 1992). The inner core of this algorithm is sequence comparison, and thus it can be adapted for use with most sequence comparison co-processors. An alternative family of algorithms, based on checkpoints, requires $O(n^{1.5})$ to $O(n \log n)$ space with two advantages. First, unlike the divide-and-conquer algorithm, the method can be used for serial or parallel forward-backward (Baum-Welch) linear hidden Markov model training. Second, it avoids the costly data-dependent

(host-level) repartitioning that can cause a 10- to 30-fold drop in speed between comparison and alignment (Grice *et al.*, 1996).

Architecture

Computer architectures for sequence analysis can be divided into five broad categories.

Workstations (WS)

Workstations can be slow, but are readily available and useful for many other tasks. Small networks of workstations can be used for coarse-grain parallelization of database searches.

This report includes data from a Sun UltraSparc 140/143 and a network of five DEC Alpha workstations (Pearson, 1995a).

Supercomputers (GS)

General-purpose supercomputers are, when available, the most flexible means of fast sequence analysis. Unfortunately, supercomputers have high cost.

This report includes data on the MasPar MP-2 and a 32-node Paragon. The MasPar is possibly the supercomputer best suited to sequence analysis (Nickolls, 1990). It is a single instruction stream, multiple data stream (SIMD) computer, meaning that all of its processing elements execute the same instruction at the same time. This enables MasPar to place 32 low-complexity PEs on each chip and build large, cost-effective machines. In contrast, the Paragon is a multiple instruction stream, multiple data stream (MIMD) machine optimized for general scientific computation which, for simple, integer-based sequence comparison, does not do much better than the network of Alpha workstations (Pearson, 1995a).

Single-purpose VLSI (SP)

Single-purpose VLSI can achieve the highest performance on one single algorithm for, typically, prices in the low tens of thousands of dollars.

PNAC, a machine for calculating the edit distance over a four-character alphabet, was the first such design (Lopresti, 1987). BioSCAN is a fast and exceedingly high-density (812 PEs per chip) implementation of a statistical sequence analysis method similar to BLAST, and is available on the Web (<http://genome.cs.unc.edu/>) (Singh *et al.*, 1993). BISP is a hardware implementation of the full Smith & Waterman algorithm, though a complete system was not reported (Chow *et al.*,

1991). Mercury and SAMBA, both currently in working systems, follow the BISP approach by implementing gapped sequence analysis and profiles in hardware (Brutlag *et al.*, 1995; Lavenier, 1996). SAMBA uses PAM, mentioned below, as a host interface. The commercial Fast Data Finder (FDF) is primarily geared to text search, but has been adapted to sequence analysis, and a server is available for Fast Data Finder profile-search (http://ulrec3.unil.ch/software/FDFGP_form.html). The 5-board FDF machine has 3360 processing cells, about 40 of which are required for each amino acid position. Recirculation currently enables queries up to 800 amino acids (Roberts, 1989).

Reconfigurable Hardware (RH)

Reconfigurable hardware includes systems based on field-programmable gate arrays (FPGAs) and other architectures such as MGAP (Borah *et al.*, 1994). RH machines generally cost somewhat more than SP machines.

The trail blazing general-purpose RH machines are PAM and Splash, both based on Xilinx FPGA technology (Bertin *et al.*, 1989; Lemoine *et al.*, 1994; Gokhale *et al.*, 1991; Hoang, 1993). Edit-distance problems have been programmed on both of these systems, though the coding is laborious (the Splash 1 edit-distance code required about 1500 lines of code to place 24 PEs on each of 16 FPGAs and configuration required 0.47 s). The more recent Biocelerator (<http://sgbcd.weizmann.ac.il/>) and DeCypher II (<http://www.timelogic.com/>) are commercial machines based on Xilinx FPGAs specifically geared to sequence analysis (Compugen Ltd., 1994; Time Logic Inc., 1996).

The Micro-Grain Arithmetic Processor (MGAP) has its own reconfigurable architecture with a 64×64 mesh of sequence comparison cells on the original chips and a 128×128 mesh on the next-generation MGAP-2. The mapping works on 128 128-long comparisons at once — a linear pipeline would have somewhat lower total MCUPS but could handle an isolated query longer than 128 characters (Borah *et al.*, 1994).

Programmable co-processors (PCP)

Programmable co-processors strive for the algorithmic flexibility of reconfigurable systems and the speed and density of single-purpose systems. Cost and ease of programming falls between SP and RH.

Processors In Memory (PIM) is a programmable co-processor formed as a linear array of bit-serial processing elements and memory, with 64 1-bit PEs per chip, each with 2 Kilobits of memory (Gokhale *et al.*, 1995),

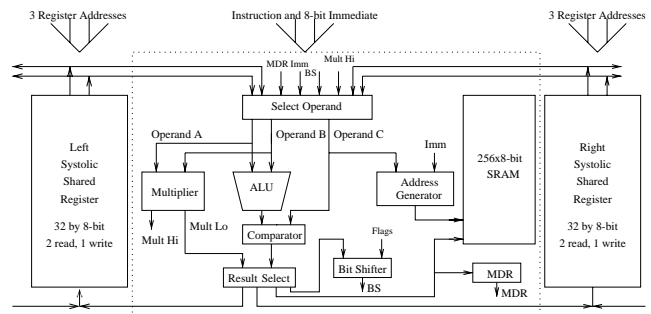


Figure 3: Kestrel programmable co-processor processing element.

Kestrel is a work in progress that draws from the B-SYS architecture (Hughey & Lopresti, 1991). The Kestrel processor is designed to greatly reduce the instructions required for sequence comparison while maintaining a high clock speed (Hirschberg *et al.*, 1996). A single-cycle instruction can add two numbers, compare the result to a third from memory, select the minimum and store the choice in the bit shifter, and communicate to the next PE by writing the minimum to a shared register (Figure 3). We are currently constructing prototype PE chips, simulation of which has provided us with justifiable timing estimates. The full system is expected to be complete in 1997.

Evaluation

Table 1 summarizes performance of several systems in millions of dynamic programming cell updates per second (MCUPS). For an $m \times n$ sequence comparison or alignment on an array of length l , the runtime in microseconds is approximated by

$$a + bm + cn + \left(1 + k \left\lfloor \frac{m}{l} \right\rfloor\right) \frac{ln}{MCUPS},$$

where a represents constant overhead (e.g., for program loading), bm represents query loading overhead, cn represents database loading overhead, ln is the number of dynamic programming cells calculated by the array for a database of length n in one pass, and $k > 1$ represents the co-processor's overhead in problem partitioning. At short query sequence lengths, the a and cn terms will dominate.

In the table, a question mark by the year indicates that a completed system has not been reported. Reference's or the author's projections for systems without experimental times in some or all categories are similarly indicated — the price estimates are particularly tenuous, though the column is still felt to be useful as high per-

System	Reference	Type	Year	PEs/ Chip	PEs	Boards	System Performance, MCUPS			Est Price k\$
							4-char edit dist	protein affine	Global Align	
UltraSparc 140/143		WS	1996	1	1	1	13	2	1	13
5 Alpha AXP300s	(Pearson, 1995a)	WC	1994	1	5	5	30?	17	?	50
Maspar MP-2	(Nickolls, 1990)	GS	1992	32	1024	3?	92	35	10	100
Maspar MP-2	(Nickolls, 1990)	GS	1992	32	16684	18?	1260	500	60	1000
Paragon	(Pearson, 1995a)	GS	1994	1	32	32?	50?	25	?	500
PNAC	(Lopresti, 1987)	SP	1986	30	270	1	1	×	×	1?
BISP ^a	(Chow <i>et al.</i> , 1991)	SP	1991?	16	256	1	3200?	3200?	×	20?
FDF-3	(Paracel Inc, 1996)	SP	1992	20	3360	5	590	230	×	50
BioSCAN	(Singh <i>et al.</i> , 1993)	SP	1992	812	12992	1	×	25000*	×	20?
Mercury-1	(Brutlag <i>et al.</i> , 1995)	SP	1995	8	32	1	320	320	10	
Mercury-2	(Brutlag <i>et al.</i> , 1995)	SP	1995	8	64	2	640	640	10	
SAMBA	(Lavenier, 1996)	SP	1995	8	128	3	730	730	20?	60?
Splash I	(Hoang, 1992)	RH	1992	24	746	1	370	?	90*	40?
Splash II ^c	(Hoang, 1993)	RH	1993	24	64	1	3000?	?	?	50?
MGAP	(Borah <i>et al.</i> , 1994)	RH	1994	128	64×64	1	370?	?	×	30?
Biocelerator-1	(Compugen Ltd., 1994)	RH	1994	1	16	1	250	250	×	66 ^d
DeCypher II-1	(Time Logic Inc., 1996)	RH	1996	16	128	1	98	98	×	18
DeCypher II-15	(Time Logic Inc., 1996)	RH	1996	16	1920	15	1400	1400	×	173
MGAP-2 ^e	(Borah <i>et al.</i> , 1994)	RH	1996?	512	128×128	1	1480?	?	×	40?
B-SYS	(Hughey & Lopresti, 1991)	PC	1991	47	470	1	18	1?	×	8?
Kestrel	(Hirschberg <i>et al.</i> , 1996)	PC	1997?	64	1024	1	3000?	1600?	450?	30?

^a Requires 12 MB/s. The reference's proposed VME interface would reduce performance to 770 MCUPS.

^b Approximate alignment is available at MCUPS similar to protein comparison.

^c 3 MB/s disk would reduce estimate to 190 MCUPS.

^d Academic discount price of the 1994 1-board, 4-module system.

^e 3 MB/s disk would reduce estimate to 380 MCUPS.

Table 1: Sequence comparison performance on various machines. Question mark (?) indicates estimate or unknown. Asterisk (*) indicates values for other than affine costs on a 20-character alphabet.

formance at low cost is the hallmark of specialized coprocessors. For research systems, the estimate is based on the number of custom or FPGA chips in the system. FPGA-based systems tend to have a higher incremental cost because of the overhead of using another company's chip design. Custom chips tend to have a higher or considerably higher design cost to balance their lower incremental costs.

The PEs per processing chip (on-board memory and control logic are not counted) and total number of PEs provide some indication of the computational density of the machine. In general, higher computation per chip (or per unit area) is better. The board count is another indication of system complexity, though it can be misleading, as a "board" is not a standardized unit. For example, the Mercury and DeCypher II are implemented on personal computer boards, while BioSCAN, MGAP, and SAMBA are implemented on much larger VME boards, and the Biocelerator is implemented on a VME board that includes positions for up to four daughter-board processor

modules.

It is important to note that the BioSCAN results are not for the affine cost model, but for its own BLAST-like algorithm. Also, several of the machines with no listing for sequence alignment could use the divide-and-conquer approach, as is done on Mercury. Slowdown from protein comparison for this more complex control would be similar to Mercury's. The checkpointing algorithm, which requires memory in each PE, is used for the Kestrel alignment estimate.

For large database searches, multiple machines or coprocessors and coarse-grain parallelism can be used to achieve higher performance. The commercial machines, and several of the research machines, are specifically designed to be expandable, so it is important to realize that faster systems are available, though at a higher price. For example, the Biocelerator features configurations from a single daughter-board (65 MCUPS) to four full boards (16 processor modules, 500 to 900 MCUPS for queries of lengths 500 to 5000, respectively), or even eight. The

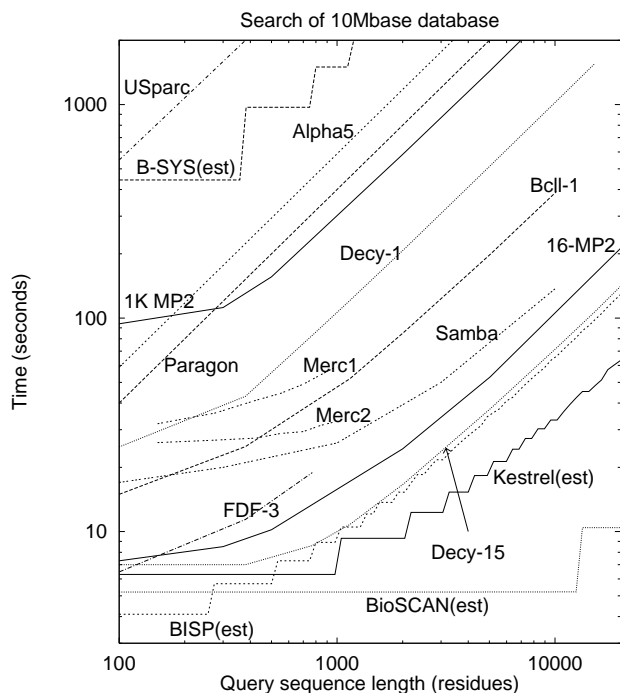


Figure 4: Estimated or experimental time for 10Mbase search (affine or BioSCAN).

largest FDF system shipped has over 85,000 FDF-PEs and six workstation hosts. The Paragon and, obviously, workstation networks, can also be expanded.

The table, however, makes the mistake of reducing performance to a single number. The numbers are typically for long queries, where setup time and database loading time are negligible, or just ignored altogether. Figure 4 shows actual or estimated performance for protein sequence comparison on several machines. At the low end of the graph, comparison times are dominated by constant factors, such as configuring the system, loading the query, and streaming the database through the array. At the high end, the machines reach their asymptotic performance as the dynamic programming calculation must be partitioned into several passes through the array. Although an individual protein sequence query of length 10000 is uncommon, similar performance can be expected on a group of sequences with total length 10000 for those systems that support such operations. Combining queries is only helpful, of course, if the queries are searching the same database. Queries can also be combined sequentially: if two successive queries require the same database, and the database has been loaded into memory, up to ~ 10 seconds can be saved by not reading the database from disk a second time.

The BISP, B-SYS, and Kestrel numbers are based on

the designer's estimated MCUPS and a constant offset for setup, the effects of which generally vanish at sequences of length ~ 500 . Achieving the Kestrel estimates will require an I/O rate of 1.5 MB/s and a 33 MHz board. The numbers for the Alpha workstations and Paragon are based on direct linear scaling from the MCUPS numbers. The Bioccelerator, DeCypher II, FDF-3, and MasPar results are experimental times, based on database searches of SwissProt-32 with 18,531,385 residues, divided by a normalization factor of 1.85. System configurations chosen for the plot are based entirely on availability for benchmarking. The Mercury and SAMBA curves are based on experimental data of a search of SwissProt-31, divided by a normalization factor of 1.53. The BioSCAN results are estimated from experimental throughput data.

The general observation is that PE density and I/O speed are critical. The networked workstations and the MIMD parallel processor have low PE density, while in general RH has medium density and SP and PH have high density, especially in the case of BioSCAN. Co-processors are only as good as their connection to data, though. For long sequence lengths, data repartitioning means that computation is the prime consideration, while for shorter lengths, I/O and startup time are the governing factors. For example, at high sequence lengths, SAMBA and the 16-K PE MasPar are much closer in performance than at short sequence lengths, where the MasPar's high-performance I/O system is a great aid.

Discussion

One of the most interesting parts of this study has been the difficulty in comparing architectures. A few of the problems that can come up in benchmark numbers include:

- Different problem sizes. While it is easy to normalize performance to, for example, a 10-million residue search, it is more difficult to adjust for different query lengths.
- Different algorithms. Edit distance performance is sometimes provided without any discussion of the more complex forms of sequence analysis.
- Peak versus real. Performance numbers are sometimes estimated peak chip speed rather than full system speeds. Configuration time and database loading are also often ignored.
- Performance projection. Performance numbers are sometimes projected to a larger system than the existing one. Or, as in the case of Kestrel, merely the result of simulation.

- Speedup versus performance. Speedup numbers can be useful, but only if the comparison implementation is the best possible. Speedup can always be calculated from the absolute timings, so one must be wary when only speedup is presented.
- Specific hardware. Sometimes references are unclear about the exact state of the hardware and what configuration is being used for experimental or estimated performance analysis.

All architectures have strengths and weaknesses for sequence comparison and alignment and other applications. Supercomputers, where affordable, provide good performance coupled with applicability to a wealth of problems. For less expensive and faster solutions, suitable for repetitious use as network servers or hidden Markov model trainers, specialized co-processors are an excellent solution. The reconfigurable, bit-oriented mesh designs work well for sequence analysis. Single-purpose hardware can be incredibly fast but is restricted to a single algorithm. Programmable co-processors attempt to mix the speed of single-purpose hardware and the programmability of more general approaches.

Because biologists' favored comparison and alignment algorithms are not fixed, programmable parallel solutions are required to speed these tasks. As an alternative to single-purpose systems, hard-to-program reconfigurable systems, and expensive general-purpose supercomputers, we advocate the use of specialized yet programmable hardware whose development is tuned to system speed.

Acknowledgments

This work was supported in part by NSF grant MIP-9423985. Further information can be obtained on the WWW

from <http://www.cse.ucsc.edu/research/kestrel/>, where the author plans to maintain links and information to parallel sequence analysis hardware and applications.

The author greatly thanks Douglas Brutlag (Mercury), Philipp Bucher and Arthur Thomas (FDF), John Burke (MasPar), Jim Lindelien (DeCypher II), Eli Mintz (Biocelerator), William Pearson (Alpha and Paragon), Raj Singh (BioSCAN), and Rachel Karchin (WWW benchmarking) for their help assembling this data. The author also thanks Jeffrey Hirschberg, Don Speck, the Kestrel team, and the anonymous reviewers for their helpful comments on this paper.

References

Altschul, S. F. *et al.* (1990). Basic local alignment search tool. *JMB*, **215**, 403–410.

- Bertin, P., Roncin, D., & Vuillemin, J. (1989). Introduction to programmable active memories. Technical Report 3 Digital Paris Research Laboratory Rueil Malmaison, France.
- Borah, M., Bajwa, R. S., Hannehalli, S., & Irwin, M. J. (1994). A SIMD solution to the sequence comparison problem on the MGAP. In: *ASAP*, (Capello, P. *et al.*, eds) pp. 336–45, Los Alamitos, CA: IEEE CS.
- Brutlag, D., Deautricourt, J.-P., & Griffin, J. (1995). Personal Communication.
- Chow, E., Hunkapiller, T., Peterson, J., & Waterman, M. S. (1991). Biological information signal processor. In: *ASAP*, (Valero, M. *et al.*, eds) pp. 144–160, Los Alamitos, CA: IEEE CS.
- Compugen Ltd. (1994). Biocelerator information package. Obtained from compugen@datasrv.co.il.
- Edmiston, E. W., Core, N. G., Saltz, J. H., & Smith, R. M. (1988). Parallel processing of biological sequence comparison algorithms. *International Journal of Parallel Programming*, **17** (3), 259–275.
- Gokhale, M. *et al.* (1991). Building and using a highly parallel programmable logic array. *Computer*, **24** (1), 81–89.
- Gokhale, M. *et al.* (1995). Processing in memory: The Terasys massively parallel PIM array. *Computer*, **28** (4), 23–31.
- Gotoh, O. (1982). An improved algorithm for matching biological sequences. *JMB*, **162** (3), 705–708.
- Gribyskov, M., Lüthy, R., & Eisenberg, D. (1990). Profile analysis. *Methods in Enzymology*, **183**, 146–159.
- Grice, J. A., Hughey, R., & Speck, D. (1996). Reduced space sequence alignment. *CABIOS*. To appear.
- Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, **18** (6), 341–343.
- Hirschberg, J. D., Hughey, R., Karplus, K., & Speck, D. (1996). Kestrel: A programmable array for sequence analysis. In: *ASAP* pp. 25–34, Los Alamitos, CA: IEEE CS.
- Hoang, D. T. (1992). A systolic array for the sequence alignment problem. Technical Report CS-92-22 Dept. CS, Brown Univ. Providence, RI.
- Hoang, D. T. (1993). Searching genetic databases on Splash 2. In: *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, (Buell, D. A. & Pocek, K. L., eds) pp. 185–191, Los Alamitos, CA: IEEE CS.
- Huang, X. (1989). A space-efficient parallel sequence comparison algorithm for a message-passing multiprocessor. *International Journal of Parallel Programming*, **18** (3), 223–239.
- Hughey, R. & Krogh, A. (1996). Hidden Markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS*, **12** (2), 95–107.
- Hughey, R. & Lopresti, D. P. (1991). B-SYS: A 470-processor programmable systolic array. In: *ICPP*, (Wu, C., ed) volume 1 pp. 580–583, Boca Raton, FL: CRC Press.
- Ibarra, O. H., Jiang, T., & Wang, H. (1992). String editing on a one-way linear array of finite-state machines. *IEEE Transactions on Computers*, **41** (1), 112–118.
- Krogh, A., Brown, M., Mian, I. S., Sjölander, K., & Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *JMB*, **235**, 1501–1531.
- Lavenier, D. (1996). SAMBA: Systolic accelerators for molecular biological applications. Technical Report 988 IRISA 35042 Rennes Cedex, France.

- Lemoine, E., Quinqueton, J., & Sallantin, J. (1994). High speed pattern matching in genetic data base with reconfigurable hardware. In: *ISMB*, (Altman, R. *et al.*, eds) pp. 269–275, Menlo Park, CA: AAAI/MIT Press.
- Lipton, R. & Lopresti, D. (1987). Comparing long strings on a short systolic array. In: *Systolic Arrays*, (Moore, W., McCabe, A., & Urquhart, R., eds) pp. 363–376. Adam Hilger Boston, MA.
- Lopresti, D. P. (1987). P-NAC: A systolic array for comparing nucleic acid sequences. *Computer*, **20** (7), 98–99.
- Masek, W. J. & Paterson, M. S. (1983). How to compute string-edit distances quickly. In: *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison* pp. 337–349. Addison-Wesley Reading, MA.
- Myers, E. W. & Miller, W. (1988). Optimal alignments in linear space. *CABIOS*, **4** (1), 11–17.
- Nickolls, J. R. (1990). The design of the Maspar MP-1: A cost effective massively parallel computer. In: *COMPCON Spring 1990* pp. 25–28, Los Alamitos, CA: IEEE Computer Society Press.
- Paracel Inc (1996). FDF-3 product information. Pasadena, CA, <http://www.paracel.com>.
- Pearson, W. R. (1995a). Personal communication.
- Pearson, W. R. (1995b). Comparison of methods for searching protein sequence databases. *Protein Science*, **4** (6), 1145–1160.
- Roberts, L. (1989). New chip may speed genome analysis. *Science*, **244** (4905), 655–6.
- Sellers, P. H. (1974). On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.* **26**, 787–793.
- Singh, R. *et al.* (1993). A scalable systolic multiprocessor system for biosequence similarity analysis. In: *Symp. Integrated Systems*, (Snyder, L., ed) pp. 169–181, Cambridge, MA: MIT Press.
- Smith, T. F. & Waterman, M. S. (1981). Identification of common molecular subsequences. *JMB*, **147**, 195–197.
- Time Logic Inc. (1996). Decypher II product literature. Incline Village, NV, <http://www.timelogic.com>.