from *SIAM News*, Volume 31, Number 8

Speeding Up Genome Computations With a Systolic Accelerator

By Dominique Lavenier

The comparison of DNA or protein sequences, a fundamental task in molecular biology, occurs in a variety of ways. The goal is to find similarities—areas with shared subsequences—between two or more sequences. This task is performed in applications ranging from the sequencing of DNA molecules to database scanning.

APPLICATIONS ON ADVANCED ARCHITECTURE COMPUTERS

Greg Astfalk, Editor

Similarities are detected by algorithms whose computational complexities are quadratic with respect to the length of the sequences. A comparison of sequences is thus time-consuming when a large amount of data (a large set of sequences, which is also called a "bank") must be processed. Several approaches can be taken to speed up the computation.

The simplest approach is to wait for improved technology in the form of increased processor speeds. This approach is not very fruitful since the sizes of biological databases are growing at a rapid rate, by a factor of 1.5–2 every year. This exceeds the growth rate of processor performance.

Another solution that has been widely adopted and that has proved very efficient consists of introducing heuristics into the comparison algorithms. Speedups of 10 to 100 can be achieved in this way. The use of heuristics has two major drawbacks: (1) They cannot be applied to all comparison algorithms, and (2) their application in some cases can seriously diminish the quality of the results. In practice, when a heuristic is efficient at reducing the execution time, the quality of the results is lower.

A last alternative for those seeking high-quality results in a short time is parallel computation, for which there are three possibilities: massively parallel machines, networks of workstations, and dedicated hardware. The first possibility works well. The sequences to be compared are dispatched to the nodes of the parallel machines, which independently perform their computations. The partial results are then merged to get the final results. Nevertheless, due to the high cost of parallel computers, this solution suits only a small number of laboratories.

Networks of workstations, a less expensive alternative, use computational resources already available in the laboratories. The parallelization is performed as in parallel machines; each workstation works independently on its own data. The heterogeneous collections of machines in most laboratories, however, make this approach quite difficult to implement. Machines from different manufacturers have different operating systems and can vary so widely in performance that efficient load-balancing of the computations becomes very difficult.

The solution we propose falls into the dedicated hardware category. The machine is based on a systolic array of fully custom processors connected to a host workstation. Current technology allows us to build a 128-processor machine in a few chips. A complete system can be housed on a single PCI board. The addition of low-cost, dedicated hardware to a PC or workstation for parallelizing the comparison algorithms can decrease the execution time by two orders of magnitude.

Basic Algorithm and Parallelization

Surprising relationships have been discovered between biological sequences that have little overall similarity but in which similar subsequences can be found. In that sense, the identification of similar segments (subsequences) is probably the most useful and practical method for comparing two sequences. Fifteen years ago Smith and Waterman [12] proposed a dynamic programming algorithm for detecting, between two sequences, highly similar pairs of segments.

The algorithm compares two sequences by computing a distance that represents the minimal cost of transforming one segment into another. Two elementary operations are considered: substitution and insertion/deletion (the latter being what is called a "gap operation"). Through a series of such elementary operations, any segment can be transformed into any other segment. The smallest number of operations required to change one segment into another can be taken as the measure of the distance between the segments.

More formally, let $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$ be two sequences that are to be compared. Let $d(x, y)$ be the substitution cost for changing x into y and g the cost of the insertion/deletion (gap) operation. $H(i, j)$ is defined as the maximum similarity of two segments ending at x_i and y_j. The Smith and Waterman algorithm is then given by the following recursion:

$$H(i,j) = Max\begin{cases} 0 \\ H(i-1,j-1) + d(x_i, y_j) \\ H(i-1,j) - g \\ H(i,j-1) - g \end{cases} \quad (1)$$

with $H(i, 0) = H(0, j) = 0$.

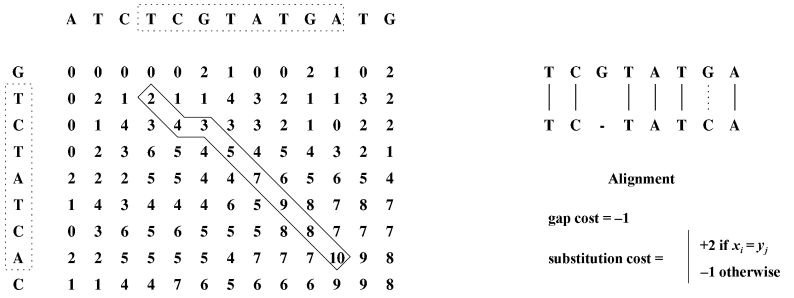

Figure 1. *Sample computation of the best alignment between the two DNA sequences* ATCTCGTATGATG *and* GTCTATCAC. *The matrix is first computed with a gap cost of –1, and a substitution cost of +2 if the characters are identical and –1 otherwise. From the highest score (+10 in the example), a traceback procedure delivers the alignment, the two subsequences* TCGTATGA *and* TCTATCA.

Given $H(i, j)$, a traceback procedure can be used to determine the alignment between the two segments. Figure 1 illustrates the detection of the best alignment between two small DNA sequences.

This algorithm, with slight modifications to the basic recursion, can be used in many applications. For the parallelization of equation (1), one processing element is associated with each value $H(i, j)$. Consider an array of $n \times m$ processors, denoted $P_{i,j}$, connected as indicated in Figure 2. Suppose that each $P_{i,j}$ is able to perform the computation expressed by equation (1). Figure 2 illustrates the way the data must be transmitted between processors. The data required by $P_{i,j}$ are represented by solid arrows. $H(i-1, j-1)$ is produced by $P_{i-1,j-1}$, $H(i, j-1)$ by $P_{i,j-1}$, and $H(i-1, j)$ by $P_{i-1,j}$. Having all this information, processor $P_{i,j}$ calculates $H(i, j)$ and provides the processors $P_{i+1,j+1}$, $P_{i+1,j}$, and $P_{i,j+1}$ with the data they need. These data are represented by dashed arrows in Figure 2.

Overall, this two-dimensional array operates on a diagonal basis. Assume that the computation starts at time 0. By time $t = i + j + 1$, all the processors $P_{i,j}$ are active. It can be verified that all the arguments needed for the computation of equation (1) have already been computed and have been routed correctly. Since only one diagonal of this array is active at a time, the implementation can be done on a linear array. Each processor will need to perform the computations for a column (m processors) of the array.

On a linear systolic array, the process of comparing two sequences consists of loading one sequence into the array (one character per cell) and sending the other sequence horizontally, character by character, one on each systolic cycle. If l_1 is the length of the first sequence and l_2 the length of the second, the comparison is performed in $l_1 + l_2 - 1$ systolic cycles, instead of the $l_1 \times l_2$ steps required on a sequential processor. Additionally, this architecture is very well suited for database scanning, where one sequence (the query sequence) must be compared, for example, to $O(10^5)$ sequences. The query sequence is loaded into the array first, and the bank is then sent through the array in a pipelined manner. The speedup is given as:

$$Sp = \frac{\left(l_q \times l_b\right) \times N}{l_q + \left(l_b \times N\right) - 1} \simeq l_q \qquad (2)$$

where l_q, l_b, and N are the length of the query sequence, and the average length and the number of sequences in the bank, respectively. This scheme supposes an array of l_q processors.

The SAMBA Accelerator

SAMBA (Systolic Accelerator for Molecular Biological Applications) is a hardware accelerator based on a fully custom systolic array designed for accelerating a class of algorithms involving biological sequence comparison. The complete SAMBA system comprises a workstation, a systolic array of 128 fully custom hardwired 12-bit processors, and a reconfigurable interface that acts as a hardware-programmable driver for the systolic array (see Figure 3).

SAMBA implements a parameterized version of the Smith and Waterman algorithm. By varying a few parameters, it is possible to perform local or global comparisons, with or without gap penalties. The accelerator can be used to speed up a variety of comparison approaches, represented by such software packages as BLAST [1], FASTA [10], and SSEARCH [11].

Performing sequence comparisons on a systolic array is not a new idea. Other systems based on these structures have been described in the literature. Related projects that have used dedicated systolic arrays are the BISP [4] and the BioSCAN [13] machines. Other machines, such as KESTREL [7] and RAPID-2 [2], are based on a programmable systolic array. FPGA (field-programmable gate array) systems like SPLASH-2 [8] and PeRLe-1 [6] use custom systolic arrays for sequence comparison. Thus, the general architecture of SAMBA cannot be considered revolutionary. It does, however, include many features not present in other systems.

The systolic array and the workstation are connected through a reconfigurable interface, which constitutes a link between a fully programmable von Neumann machine and the dedicated hardware. It is an important element of the system and

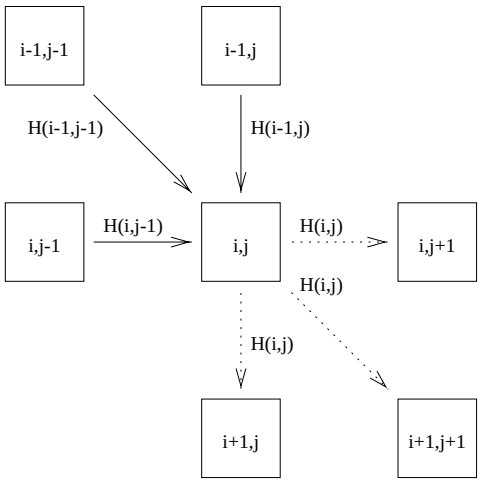

Figure 2. *Interprocessor connections: Equation (1) is parallelized through the association of one processing element* $P_{i,j}$ *with each value H(i, j). Processor* $P_{i,j}$ *receives data from processors* $P_{i-1,j-1}$, $P_{i,j-1}$, *and* $P_{i-1,j}$ *and sends data to processors* $P_{i+1,j+1}$, $P_{i+1,j}$, *and* $P_{i,j+1}$.

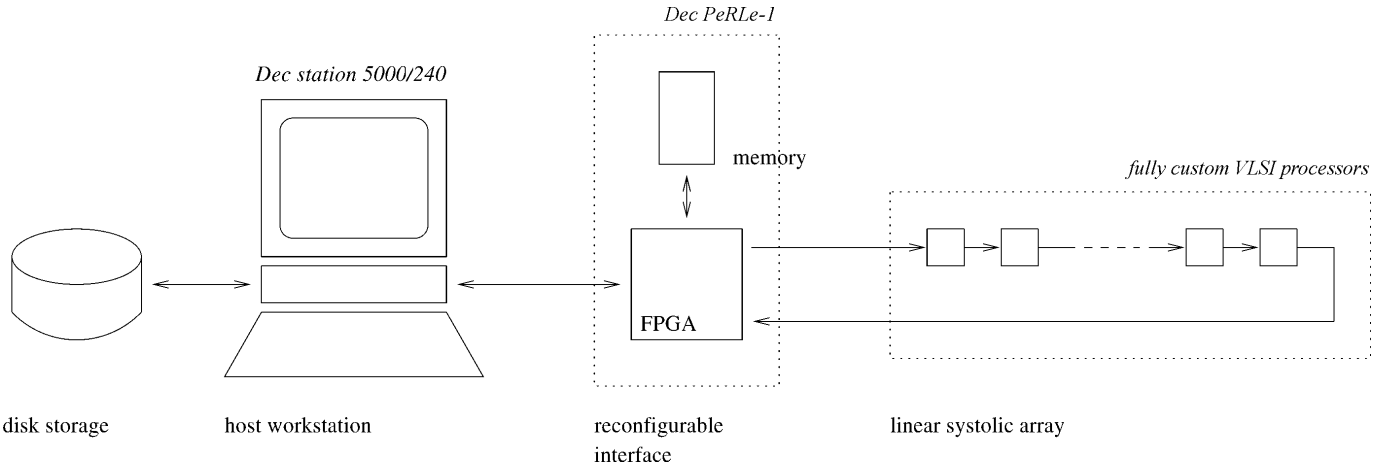

Figure 3. *SAMBA comprises a workstation, with local disk, a systolic array of 128 VLSI fully custom processors, and a reconfigurable interface that fills the gap between a complete hardwired array of processors and a programmable von Neumann machine.*

greatly influences the performance of SAMBA. The interface has the responsibility for partitioning the computations and filtering results on the fly, as explained later.

Recall that the process of comparing one sequence (a query) against a large set of sequences (a bank) consists of loading the query sequence into the array (one character per processor) and then pipelining the bank through the array. This scheme assumes an array equal in size to the length of the query sequence. In practice, this never happens. The query sequence is almost always too long (i.e., larger than 128 characters), requiring that the sequence comparison be split into several passes.

The partitioning operates as follows: The first 128 characters of the query sequence are loaded into the array. The entire bank then crosses the array; all the data output by the last processor are stored in memory. In the next step, the following 128 characters of the query sequence are loaded. The data stored previously are mixed with the bank and sent again to the array. The process is iterated until the end of the query sequence is reached. For reasonable efficiency, the partitioning must be performed at the clock rate of the array. Unfortunately, this rate cannot be sustained by standard microprocessors: Every 100 ns the array must be supplied with new data, prescribing a hardwired solution.

Communication with the array can be handled only through the left-most and the right-most processors. The results (especially when local alignment is performed) must therefore be routed to those processors. SAMBA processors are provided with a hardware mechanism for propagating information in this way. More precisely, the processors yield partial results, which need to be processed outside the array for recovery of the final result. Once again, to be efficient this treatment must be done on the fly, as results are output from the array, and cannot be performed by a standard processor.

The point is that for both operations, the treatments change according to the comparison algorithm being executed: The array is supplied with data in different ways for global and local comparisons, for example; the results delivered also have different meanings and must be handled differently. Hence, in neither case can a fixed hardware mechanism for supplying the array with data or for filtering results on the fly be definitively implemented. FPGA technology, with its combination of speed and flexibility, is the best alternative for meeting the requirements.

Another important feature of the SAMBA architecture is the availability of local storage near the array. As mentioned earlier, the partitioning process requires the data bank to cross the array several times. At first glance, storage of the entire bank in a fast local memory could be envisioned. A closer understanding of the partitioning mechanism yields a better solution: When a query sequence is compared against a bank, the SAMBA memory can hold only a subset of the bank; as the computation is performed on this subset, another subset is loaded from the disk. The overlap between the computation and the data acquisition is efficient owing to the partitioning operation, which takes time and permits the data from the bank to be accessed at a reasonable rate.

The main advantage of this solution is the small amount of memory required as compared with the size of the bank: The minimum amount of memory is determined by the sum of the lengths of the query sequence and the longest sequence in the bank. In practice, the problem is formulated differently: The memory size is a fixed resource that limits the length of the sequences to be processed.

SAMBA can be considered as a co-processor that is accessed when intensive biological sequence comparison is needed. Typically, the main operations that have to be accomplished with SAMBA are the initialization of the board and the loading of a query sequence or a bank subset.

The accelerator is controlled by means of a few procedure or function calls programmed inside a normal C program; the user needs no specific knowledge of the structure of the accelerator or how it works. A library of basic procedures that can be rapidly understood by programmers has been developed; these procedures stay close enough to the accelerator hardware to provide efficient speedups. Examples of procedures include initialization of the substitution costs between amino acids, selection of the comparison algorithm (local or global search, with or without gaps), comparison of two sequences, comparison of one sequence against a few sequences, and comparison of two subsets of sequences.

This approach has been chosen to cover a large range of applications requiring conventional sequence comparison treatments.

By carefully choosing the basic library procedures, programmers will not encounter too many limitations. This approach makes possible the choice of predefined programs, including classical programs already developed for bank scanning and user-defined programs tuned to specific applications.

Performance

We discuss performance in terms of speedups attained relative to contemporary workstations, since there are no universal benchmarks for such comparisons. No comparisons with massively parallel machines or with other dedicated systems are given, for two main reasons: (1) SAMBA is intended to boost personal computer or workstation performance via a plugged-in board, whose cost is trivial compared with that of programmable parallel computers. The Origin2000, with at least 30 processors, is an example of a machine that provides comparable performance for database scanning. (2) Comparisons with other hardware are never done in the same context: The technology, the algorithms, the applications, or the data are always different. Comparing the execution times of a standard machine with and without extra hardware is probably the best way to demonstrate the efficiency of such an approach. In our opinion, the peak performance results for systolic arrays reported in the literature are often weak measures in that they do not reflect the behavior of the complete systems.

	Query Sequence Length			
	100	300	1000	3000
SAMBA	0:40	0:50	1:50	3:20
150-Mhz DEC Alpha	8:15	24:30	83:00	280:00
Speedup	13	30	45	83

Table 1. *Execution times (in minutes:seconds) for scanning the SWISS-PROT protein bank.*

In any case, the reported times for the measurements we have made represent *total elapsed time*, as it directly affects the user; our times include time for reading the databases from the disk, as well as time for postprocessing.

Traditionally, the most typical use of hardware accelerators is in scanning databases, and SAMBA was first evaluated on that application. Table 1 reports the average times for scanning the continually growing SWISS-PROT protein bank (version 34, which contains 59,021 sequences and 21,210,389 amino acids) for protein query sequences of various lengths with the Smith and Waterman algorithm.

The first two rows of the table give the execution times for SAMBA and for a 150-MHz Dec Alpha workstation running SSEARCH [11]. As the times show, the longer the query sequence, the better the speedup. This is due mainly to the restricted bandwidth of the SAMBA I/O disk system, which prevents the array from being fed at its maximum rate: A short query sequence does not require the computation to be split into several passes, and the array is consequently fed at the disk rate, which is generally much slower than the array throughput.

Better performance is attained in bank-to-bank comparisons. In that case, the interactions between the host workstation and the array are limited: The reconfigurable interface is "programmed" to manage local comparisons of blocks of sequences efficiently, and the systolic array is supplied with data at its maximum rate. The following specific application, implemented on SAMBA by biologists, illustrates that aspect of the performance of the accelerator.

The clustering of sequences in families, according to their homology, is an important technique used by biologists to investigate genomes. Some functional categories of proteins, for example, notably those involved in metabolite transport and transcription regulation, tend to form large clusters. Using the Smith and Waterman algorithm, we examined the entire genome of *Escherichia coli*—4285 sequences and 1,355,128 amino acids)—for similar coding sequences. As our interest was limited to the clustering of sequences, SAMBA was programmed to report only alignments with scores above a threshold value. The pairwise comparison took 41 minutes on SAMBA; the same treatment performed on a currently available workstation would have taken 127.5 hours. The speedup achieved with SAMBA for that particular application was 186.

Conclusions

The SAMBA prototype, which has been available since the end of 1995, is used daily by biologists for comparison-intensive tasks or for scanning databases through the SAMBA Web server (http://www.irisa.fr/SAMBA/). The chip we designed (in 1994, using 1 – μm CMOS technology) houses four processors, each performing 100 million 12-bit operations per second. The 128-processor array is thus made up of 32 chips. The reconfigurable interface is the PeRLe-1 board developed by Vuillemin et al. [3].

The SAMBA prototype could be vastly improved with up-to-date technology. Given the increases in chip density, we imagine that we would now be able to fit between 16 and 20 processors (running at higher frequencies) into a single chip. In the same way, the design of the reconfigurable interface could now fit into a unique latest-generation FPGA component (the FPGA resources of the PeRLe-1 board are largely under-exploited!). As to memory, only a few Mbytes are required (the prototype used 2 MBytes). Hence, the current three printed circuit boards could easily be reduced to a standard PCI board, which could be plugged into any PC or workstation.

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, *Basic local alignment search tool*, J. Mol. Biol., 215 (1990), 403–410.

[2] D. Archambaud, I. Saraiva, and J. Penne, *Systolic implementation of Smith and Waterman algorithm on a SIMD co-processor*, in *Algorithms and Parallel VLSI Architectures III*, Elsevier Science, New York, 1995.

[3] P. Bertin, D. Roncin, and J. Vuillemin, *Programmable active memories: A performance assessment*, in *Parallel Architectures and Their Efficient Use*, F. Meyer, B. Monien, and A.L. Rosenberg, eds., Springer-Verlag, New York, October 1992.

[4] E. Chow, T. Hunkapiller, and J. Peterson, *Biological information signal processor*, ASAP, September 1991.

[5] O. Gotoh, *An improved algorithm for matching biological sequences*, J. Mol. Biol., 162 (1982), 705–708.

[6] P. Guerdoux-Jamet and D. Lavenier, *Systolic filter for fast DNA similarity search*, ASAP'95, Strasbourg, July 1995.

[7] J.D. Hirschberg, R. Hughey, and K. Karplus, *KESTREL: A programmable array for sequence analysis*, ASAP'96, Chicago, August 1996.

[8] D.T. Hoang, *Searching Genetic DataBases on SPLASH-2*, in *FPGAs for Custom Computing Machines*, D.A. Buell and K.L. Pocek, eds., IEEE Computer Society Press, Los Alamitos, CA, April 1993.

[9] S.B. Needleman and C.D. Wunsh, *A general method applicable to the search of similarities in the amino acid sequence of two proteins*, J. Mol. Biol, 48 (1970), 443–453.

[10] W.R. Pearson and D.J. Lipman, *Improved tools for biological sequence comparison*, Proc. Natl. Acad. Sci., 85 (1988), 3244–3248.

[11] W.R. Pearson, *Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith and Waterman and FASTA algorithms*, Genomics, 11(1991), 635–650.

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

[13] C.T. White, R.K. Singh, P.B. Reintjes, J. Lampe, B.W. Erickson, W.D. Dettloff, V.L. Chi, and S.F. Altschul, *BioSCAN: A VLSI-based system for biosequence analysis*, IEEE International Conference on Computer Design: VLSI in Computer and Processors, October 1991.

Dominique Lavenier, a permanent CNRS researcher, is currently working at the Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes. His research interests include VLSI and FPGA design, CAD tools, parallel architectures, and string processing (molecular biology).