Speeding up genome computation with a systolic accelerator

Dominique Lavenier

The comparison of DNA or protein sequences is a fundamental task in molecular biology that occurs in a variety of ways. The goal is to find similarities, areas which share common subsequences, between sequences. The operation can range from sequencing of DNA molecules to database scanning.

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

The simplest approach is to wait for technology to improve processor speed. This approach is not very fruitful since biological databases are growing at a exponential rate. Every year the size of the banks are scaled by a factor ranging from 1.5 to 2. This exceeds the growth rate of processor performance.

Another solution which has been widely adopted consists of introducing heuristics into the comparison algorithms. This is a very efficient method. Speed-ups between 10 to 100 can be achieved. There are two major drawbacks to heuristics. They cannot be applied to all comparison algorithms, and if they are they may seriously diminish the quality of results. In practice, when a heuristic is efficient at reducing the execution time, its resultant quality is lower.

A last alternative to get high quality results in a short time is through parallel computation. Three possibilities exist for this approach. Massively parallel machines, networks of workstations, or dedicated hardware. The first possibility works well. The sequences to compare are dispatched onto the parallel machine nodes, which independently perform their own computations. The partial results are then merged to get the final results. Nevertheless, due to the parallel computer cost, this solution suits only a small number of laboratories.

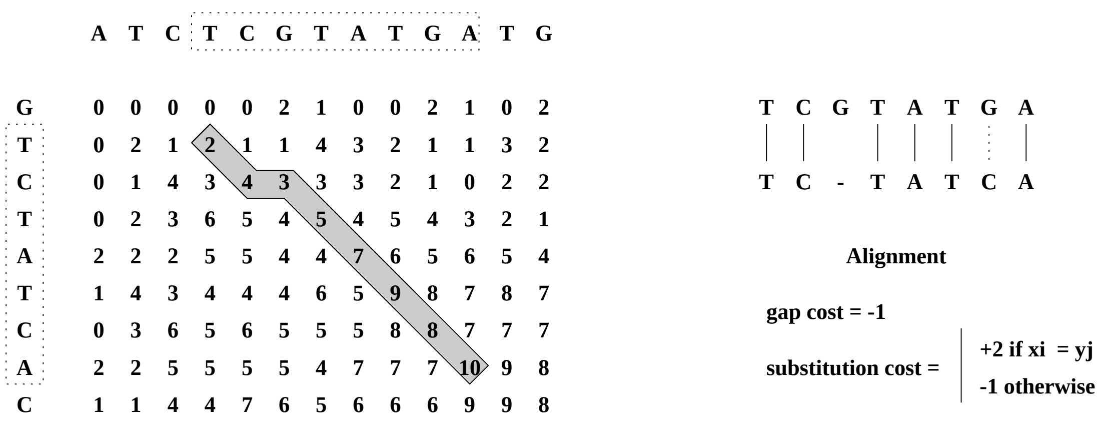
The networks of workstations is an alternative to expensive parallel machines. They use the computational resources already available in the laboratories. The parallelization is performed as in parallel machines; each workstation works independently on its own data. The heterogeneous collection of machines makes this approach quite difficult to implement. Machines come from various manufacturers with different operating systems and their performance may be so disparate that load-balancing the computation becomes inefficient.

The solution we propose falls into the dedicated hardware category. The machine is based on a systolic array of full-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 result in a reduction of two orders of magnitude in the execution time.

Basic Algorithm and Parallelization

Surprising relationships have been discovered between sequences that have little overall similarity. 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 the pair of segments, between two sequences, which present high similarity.

The algorithm compares two sequences by computing a distance which represents the minimal cost to transform one segment into another one. Two elementary operations are considered: substitution and insertion/deletion. The latter is called a gap operation. By using a series of such elementary operations any segment may be transform into any other segment. It is then possible to take the smallest number of operations required to change one segment to another as the measure of distance between them.



	A	T	C	T	C	G	T	A	T	G	A	T	G
G	0	0	0	0	0	2	1	0	0	2	1	0	2
T	0	2	1	2	1	1	4	3	2	1	1	3	2
C	0	1	4	3	4	3	3	3	2	1	0	2	2
T	0	2	3	6	5	4	5	4	5	4	3	2	1
A	2	2	2	5	5	4	4	7	6	5	6	5	4
T	1	4	3	4	4	4	6	5	9	8	7	8	7
C	0	3	6	5	6	5	5	5	8	8	7	7	7
A	2	2	5	5	5	5	4	7	7	7	10	9	8
C	1	1	4	4	7	6	5	6	6	6	9	9	8

Figure 1: Example: computation of the best alignment between the two DNA sequences ATCTCGTATGATG and GTCTATCAC. The matrix is first computed using 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.

More formally, let $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_m)$ be two sequences that are to be compared. Let $d(x, y)$ be the substitution cost to change x into y and g the cost of the insertion/deletion (gap) operation. $H(i, j)$ is defined to be 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:

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

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

Given $H(i, j)$ a traceback procedure may 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 can be taken as a basis, knowing that many applications

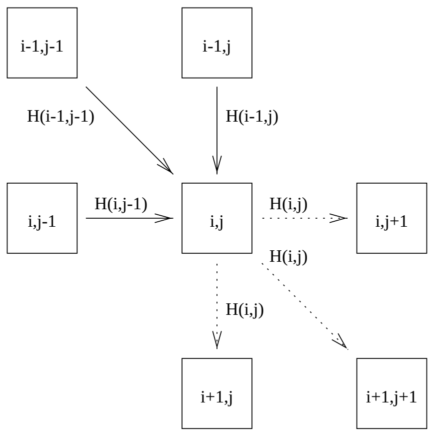


Figure 2: Inter processor connections: Equation (1) is parallelized by associating one processing element $P_{i,j}$ with each value $H(i,j)$. Processor $P_{i,j}$ receives data from processors $P_{i-1,j-1}$, $P_{i,j-1}$, $P_{i-1,j}$ and send data to processors $P_{i+1,j+1}$, $P_{i+1,j}$ and $P_{i,j+1}$.

can be treated by applying slight modifications to the basic recursion. Equation (1) can be parallelized by associating one processing element with each value $H(i,j)$. Consider an array of $n \times m$ processors denoted $P_{i,j}$ connected as indicated by Figure 2. Suppose that each $P_{i,j}$ is able to perform the computation expressed by equation (1). Figure 2 illustrates the way data must be transmitted between processors. The data required by $P_{i,j}$ is 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 the dashed arrows in Figure 2.

The overall operation of this two-dimensional array is made 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 may be done on a linear array. Each processor will have to perform the computation of a column (m processors).

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

$$Sp = \frac{(l_q \times l_b) \times N}{l_q + (l_b \times N) - 1} \quad \simeq \quad l_q \tag{2}$$

where l_q, l_b and N are respectively the length of the query sequence, the average length and the number of sequences of the bank. Note that 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 full 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 full custom hardwired 12-bit processors, and a reconfigurable interface which 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 local or global comparisons can be performed, with or without gap penalty. A variety of comparison approaches represented by software packages such as blast [1], fasta [10] or ssearch [11] may be sped-up via the accelerator.

Performing sequence comparison on a systolic array is not a new idea. Other systems based on these structures have been described in the literature. Related projects using a dedicated systolic array are the Bisp [4] and the BioScan [13] machines. Other machines, Kestrel [7] or Rapid-2 [2], are based on a programmable systolic array. fpga systems like Splash-2 [8] or PeRLe-1 [6] used custom systolic arrays for sequence comparison. Thus,

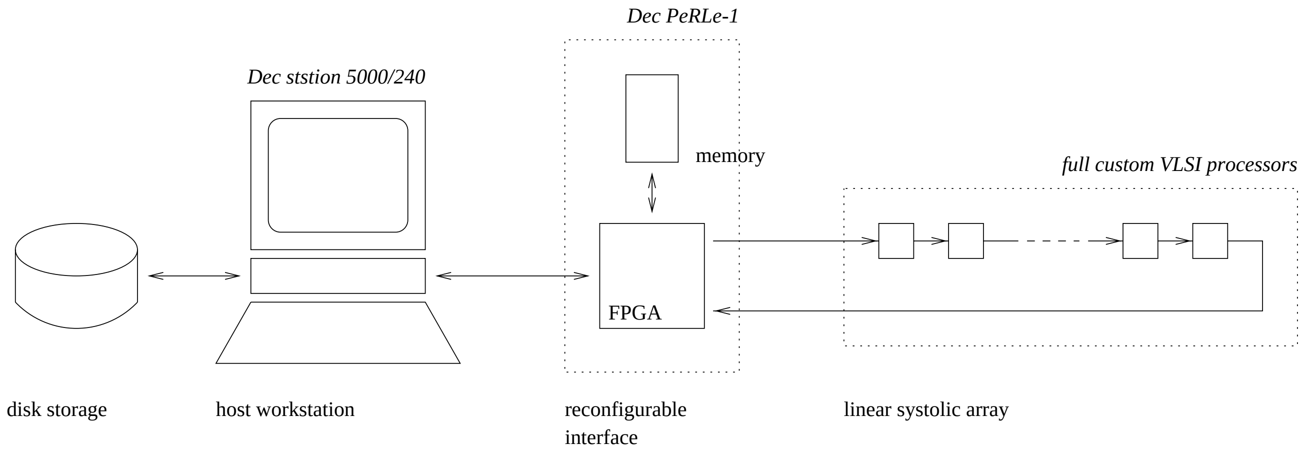


Figure 3: SAMBA comprises a workstation, with local disk, a systolic array of 128 VLSI full-custom processors and a reconfigurable interface which fills the gap between a complete hardwired array of processors and a programmable Von Neuman machine.

the general architecture of SAMBA cannot be considered as a revolutionary one. However it includes many features not present in other systems.

The systolic array and the workstation are connected through a reconfigurable interface. This constitutes a link between a fully programmable Von Neumann machine and the dedicated hardware. It is an important element of the system which greatly influences the performance of SAMBA. The interface has the responsibility for partitioning the computations and filtering results on-the-fly, as explained below.

Recall that the process of comparing one sequence (query) against a large set of sequences (bank) consists in first loading the query sequence in the array (one character per processor) and then pipelining the bank through the array. This scheme assumes an array whose the size is equal to the length of the query sequence. In practice this never happens. The query sequence is almost always too long (larger than 128 characters) and requires the sequence comparison to be split into several passes. The partitioning operates as follows: the 128 first characters of the query sequence are loaded in the array. Then, the entire bank crosses the array, while all the data output by the last processor are memorized. In the next step, the 128 following characters of the query sequence are loaded. The data previously stored are mixed with the bank, and sent again to the array. The process is iterated until the end of the query sequence is reached. To keep a reasonable efficiency, the partitioning must be performed at the clock rate of the array. Unfortunately, this rate cannot be sustained by standard micro-processors: 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. Then, the results (especially when local alignment is performed) must be routed to the extremity. Samba processors are provided with a hardware mechanism for propagating such information. More precisely, the processors yield partial results which need to be processed outside the array for recovering 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 on both operations, the treatments change according to the comparison algorithm being executed: global or local comparisons, for example, require that one supply the array with data differently; they also deliver results having different meanings and which must be handled differently. Hence, in both cases, a fixed hardware mechanism for supplying the array with data or filtering results on-the-fly cannot be definitively implemented. Fpga technology is the better alternative to meet the requirements, since it combines speed and flexibility.

Another important feature is the ability to have local storage near the array. As mentioned above, the partitioning process requires the data bank to cross the array several times. At first glance, one could envisage storing the entire bank in a fast local memory. A closer understanding of the partitioning mechanism yields a better solution. When comparing a query sequence against a bank, the Samba memory can hold only a subset of a bank: while computation is performed on this subset, another one is loaded from the disk. The overlap between the computation and the data acquisition is efficient thanks 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 size of memory it required compared to the size of the banks: the minimum size of the memory is determined by the sum of the query sequence length and that of the longest sequence of the bank. In practice, the problem is formulated differently: the memory size is a fixed resource which limits the length of the sequences to be processed.

SAMBA can be considered as a co-processor which is accessed when intensive biological sequence comparison is needed. Typically, the operations which have to be accomplished when using SAMBA concern mainly the initialization of the board and the load of a query sequence or the load of a bank subset.

From a programming point of view, the accelerator is controlled by a few procedure or function calls inside a normal C program, without forcing the user to have specific knowledge of the structure of the accelerator and how it works. A library of basic procedures has been developed to be rapidly understood by programmers; these procedures stay close enough to the accelerator hardware to provide efficient speed-up. Examples of procedures are: 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, comparison of two subsets of sequences, etc.

This approach has been chosen to cover a large range of applications which require conventional sequence comparison treatments. By carefully chosing the basic library procedures, programmers will not encounter too many limitations. This approach makes possible the choice of pre-defined programs, such as the classical programs already developed for bank-scanning, or user-defined programs tuned to a specific application.

Performance

We discuss performance in terms of speed-up relative to contemporary workstations. No comparison with massively parallel machines nor other dedicated systems are given. The two main reasons are: (1) SAMBA aims to boost personal computer or workstation performance by plugging in a board, whose price has nothing in common with programmable parallel computers. As an example, we can mention that the Origin2000 machine, with at least 30 processors, has comparable performance for scanning data-bases. Furthermore, comparing the execution time of a standard machine with or without extra hardware is probably the best way to demonstrate the efficiency of such approach. (2) Comparisons with other hardware are never done in the same context: the technology, the algorithms, the applications or the data are al-

	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
Speed-up	13	30	45	83

ways different. There exist no universal benchmarks in that domain. In our opinion, the peak performance of the systolic array, as it is often reported in the literature, is a weak measure since it does not reflect the complete behavior of a system.

In any case, the reported time of the measurements we have taken is the *total elapsed time*, as it directly affects the user; it includes time for reading the data-bases from the disk, as well as time for post-processing

Traditionally, the most typical use of the hardware accelerators is in scanning data-bases. Samba has been first evaluated on that application. Below is a table which reports the average time for scanning swiss-prot (version 34 - 59 021 sequences - 21 210 389 aa) using the Smith and Waterman algorithm for different lengths of a protein query sequence.

The first two lines give, respectively, the execution time (in minute:second) on Samba and on a 150 MHz Dec Alpha workstation running ssearch [11]. Note that the longer the query sequence, the better the speed-up. This is mainly due to the restricted bandwith of the 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. Consequently, the array is fed at the disk rate, which is generally much slower than the array throughput.

Better performance is provided for bank to bank comparison. In that case, the interactions between the host workstation and the array are limited: the reconfigurable interface is "programmed" to manage efficiently local comparison of blocks of sequences and allows the systolic array to be supply with data at its maximum rate. The following specific application, implemented on Samba by biologists, shows up the performance of the accelerator on that particular point.

Clustering sequences in family according to their homology is an important technique in investigation of genomes. For example, some functional categories of proteins, notably those involved in metabolite transport and transcription regulation, tend to form large clusters. The whole genome of *Escherichia Coli* was taken, and all the coding sequences compare behind itself using the Smith and Waterman algorithm. This represents 4285 sequences (1 355 128 aa). As we were only interested in clustering the sequences, SAMBA was programmed to report only the alignment with a score better than a threshold value. On SAMBA, the pairwise comparison took 41 minutes, while the same treatment performed on a recent workstation would have taken 127.5 hours, leading, for that particular application, to a speed-up of 186 !

Conclusions

The SAMBA prototype works since the end of 1995. It is used daily by biologists for comparison-intensive tasks or for scanning the usual data-bases through the SAMBA WEB server[1]. The chip we designed houses 4 processors, each performing 100 million 12-bit operations per second. It was designed in 1994 using a 1 μm CMOS technology. The 128 processor array is thus made of 32 chips. The reconfigurable interface is the PeRLe-1 board developed by Vuillemin et al. [3].

The SAMBA prototype could be largely improved by using up-to-date technology. As far as we can imagine it, the evolution of chip density would now allow between 16 and 20 processors to fit into a single chip, and running at a higher frequency. In the same way, the design of the reconfigurable interface could be fit into a unique FPGA component of the latest generation (the FPGA resources of the PeRLe-1 board are largely under-exploited !). As for the memory, only a few Mbytes is required (the prototype used 2 MBytes). Hence, the current three printed circuit boards could be easily reduced to a standard PCI board for plugging into any PC or workstation.

References

[1] S.F. ALTSCHUL, W. GISH, W. MILLER, E.W. MYERS, AND D.J.

[1]http://www.irisa.fr/SAMBA/

Lipman, *Basic local alignment search tool*, J. Mol. Biol., 215(1990), pp: 403–410.

[2] D. Archambaud, I. Saraiva, and J. Penne, *Systolic implementation of Smith and Waterman algorithm on a SIMD co-processor*, In Elsevier Science, editor, *Algorithms and Parallel VLSI Architectures III*, pp: 155–166, 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, editors, pp: 119–130, LNCS, Springer-Verlag, October 1992.

[4] E. Chow, T. Hunkapiller, and J. Peterson, *Biological Information Signal Processor*, In *ASAP*, pp: 144–160, September, 1991.

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

[6] P. Guerdoux-Jamet and D. Lavenier, *Systolic Filter for Fast DNA Similarity Search*, in ASAP'95, Strasbourg, July 1995.

[7] J.D. Hirschberg, R. Hughey, and K. Karplus, *KESTREL: A Programmable Array for Sequence Analysis.* in IEEE Computer Society Press, ASAP'96, Chicago, Illinois, August 1996, pp: 25–34.

[8] D.T. Hoang, *Searching Genetic DataBases on SPLASH-2.* in *FPGAs for custom computing machines*, D.A. Buell and K.L. Pocek, editors, pp: 185–191, IEEE Computer Society Press, 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*, 48:443–453, 1970.

[10] W.R. Pearson and D.J. Lipman, *Improved tools for biological sequence comparison*, Proc. Natl. Acad. Sci., 85(1988), pp: 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), pp: 635–650.

[12] T.F. Smith and M.S. Waterman, *Identification of Common Molecular Subsequences*, J. Mol. Biol, 147(1981), pp: 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*, in IEEE Int. Conf on Computer Design: VLSI in Computer and Processors, IEEE Computer Society Press, October 1991, pp: 504–509.