Design and Implementation of a Parallel Architecture for Biological Sequence Comparison

Pascale Guerdoux–Jamet[2], Dominique Lavenier[2], Charles Wagner[2] and Patrice Quinton[2]

Irisa, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract. SAMBA is a full custom parallel hardware accelerator dedicated to the comparison of biological sequences. It implements a parameterized version of the Smith and Waterman algorithm allowing the computation of local or global alignments with or without gap penalty. The speed-up provided by SAMBA over standard workstations ranges from 50 to 500, depending on the application. SAMBA was designed with an effort of less than one person/year. This includes the design, fabrication and test of a full-custom VLSI chip which is used as a building block for the 128 processor systolic array implementing the string alignment algorithm. We describe the SAMBA architecture, its performance characteristics, and we detail its design steps, from the initial specification to its full implementation. We report a first application of SAMBA to the study of yeast orphan sequences.

1 Introduction

SAMBA – Systolic Accelerator for Molecular Biological Applications – is a hardware accelerator designed for speeding up biological sequence comparisons. Computations which require several hours on standard workstations are performed in a few tens of seconds on SAMBA.

SAMBA implements a parameterized version of the Smith and Waterman algorithm [15] [5]. By setting a few parameters, local or global comparisons can be performed, with or without gap penalty. Thus, a variety of softwares, such as BLAST [1], FASTA [12] or SSEARCH [13], may be implemented on SAMBA.

The complete SAMBA system comprises a workstation, a systolic array of 128 full custom hardwired 12-bit processors, and a FPGA-based interface (see Fig. 1). The FPGA interface is the PeRLe-1 board developed by Vuillemin et al. [2]; it acts as a hardware programmable driver for the systolic array.

SAMBA may be compared with two other hardware prototypes – BISP [3] and BIOSCAN [18] – also designed to speed up biological sequence comparisons with full-custom chips. Both systems include a linear systolic array made out of dedicated processors. BISP and BIOSCAN contain respectively 256 16-bit processors and 12 000 1-bit processors.

The architecture of BISP and SAMBA are similar. However, they differ on the way the systolic array is supplied with data: the BISP array is driven by a Motorola 68020 programmable processor while SAMBA uses FPGA technology.

This last approach is simpler and ensures that the high data throughput required by the systolic array is sustained.

BioScan does not support dynamic programming algorithms. Like blast, it has been designed to detect similar segments of identical length. Hence, the algorithm is simpler and this enables a very high density of processors (812 per chips) to fit onto silicon. The speed of Samba depends on the application. The

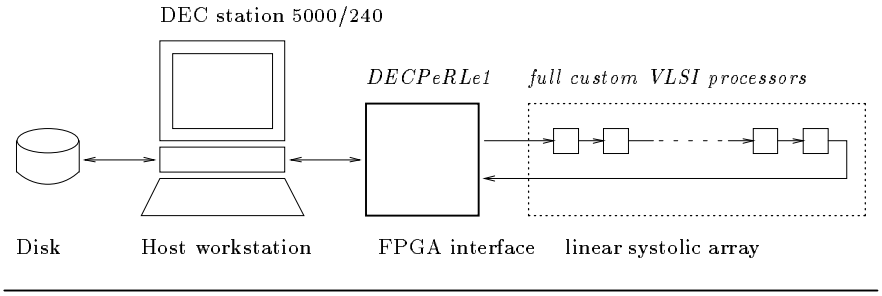

Fig. 1. SAMBA comprises a workstation (with its local disk), a systolic array made out of 128 VLSI full custom processors and a FPGA interface which fills the gap between a complete hardwired array of processors and a programmable Von Neuman machine

best speed is achieved for bank to bank comparison, that is to say, when two sequence data bases are matched against one another. In this case, the systolic array is continuously supplied with data, leading to a speed-up higher than two orders of magnitude over standard workstations.

The scan of biological banks – comparison of one query sequence against a bank, – provides also very interesting speed-ups which, however, are limited by disk access and depend on the length of the query sequence. Samba is thus better suited to intensive comparison tasks than to scanning biological banks, even if this latter task provides a noticeable speed-up.

The design of Samba took less than one person/year. It was started in September 1994, and the architecture was ready in December 1995. Since then, Samba has been used by biologists for sequence comparisons. The first application was to study so-called "orphan sequences" of the yeast genome, in order to compare the sensitivity of the Smith and Waterman algorithm with that of other less time consuming algorithms.

This paper is organized as follows: section 2 presents the class of algorithms supported by Samba. Section 3 explains how these algorithms are implemented on a linear systolic array. Section 4 gives more details on the hardware, particularly on the fpga-based interface and the Asic developed for the array. Finally, section 5 presents the results of a biological study that Samba made possible and analyses the speed of Samba.

2 The problem and its algorithmic solution

The algorithm executed by SAMBA is a parameterized version of the Smith and Waterman algorithm [15]. This algorithm was introduced in 1981 and extended by Gotoh [5] in 1982. It allows biological sequences to be aligned using two basic *edit operations*:

- *substitution* of characters;
- *insertion* or *omission* of characters; these last operations are called a *gap*.

By using a series of such edit operations, two sequences may be transformed into one another. The smallest number of edit operations required to change the first sequence into the second one is thus a measure of the distance between them, called the edit distance. The computation of the edit distance is achieved by dynamic programming, which consists of computing recursively an $N \times N$ matrix, where N is the length of the sequences, as we shall see in more details now. In subsection 2.1, we recall the principles of the Smith and Waterman algorithm, while subsection 2.2 explains how this algorithm was parameterized.

2.1 The Smith and Waterman algorithm

Consider two strings `S1` and `S2` of length respectively $l1$ and $l2$. For example,

```
S1 = AGTCCGAGGGCTACTCTACTGAAC
S2 = CCAATCTACTACTGCTTGCAGTAC
```

A comparison of these sequences will provide the alignments shown in figure 2.

```
S1   AGTCCGAGGG CTACTCTACT GAAC
             |:|:||||||
S2           CCAATCTACT ACTGCTTGCAGTAC

S1   AGTCCGAGGG CTACT.CTACT GAAC
             ||||| ||:||
S2        CCAAT CTACTACTGCT TGCAGTAC
```

Fig. 2. Examples of sequence alignments. Bar represent character matchs, and dotted lines represent gaps

To identify common subsequences, the Smith and Waterman algorithm computes the similarity $H(i,j)$ of two segments ending at position $\texttt{S1}_i$ and $\texttt{S2}_j$ of the two sequences `S1` and `S2`.

The computation of $H(i,j)$ is given by the following recurrences:

$$H(i,j) = \max \begin{cases} 0 & 0 < i \leq l1 \quad 0 < j \leq l2 \\ E(i,j) & \\ F(i,j) & \\ H(i-1,j-1) + \mathbf{Sbt}(\texttt{S1}_i, \texttt{S2}_j) & \end{cases} \tag{1}$$

where

$$E(i,j) = Max \begin{cases} H(i,j-1) - \alpha \\ E(i,j-1) - \beta \end{cases} \qquad 0 < i \leq l1 \quad 0 < j \leq l2$$

$$F(i,j) = Max \begin{cases} H(i-1,j) - \alpha \\ F(i-1,j) - \beta \end{cases} \qquad 0 < i \leq l1 \quad 0 < j \leq l2$$

Sbt is a character substitution cost table.

Initialization of these values are given by:

$$\forall i, 0 \leq i \leq l1 : H(i,0) = E(i,0) = 0$$
$$\forall j0 \leq j \leq l2 : H(0,j) = F(0,j) = 0$$

Multiple gap costs are taken into consideration as follows: α is the cost of the first gap; β is the cost of the following gaps. The total gap cost function $G(k)$ is given by: $G(k) = \alpha + \beta \times (k-1)$ with $\beta \leq \alpha$.

In the example of figure 2, with gap costs $\alpha = 20$ and $\beta = 10$ and **Sbt** function defined as:

$$\mathbf{Sbt}(x,y) = \begin{cases} 10 \ \textbf{if} \ (x = y) \\ -9 \ \textbf{otherwise} \end{cases}$$

the result of this algorithm is the matrix shown in figure 3. Each position $(\texttt{S1}_i, \texttt{S2}_j)$ of the matrix is a similarity value $H(i,j)$. The two segments of `S1` and `S2` producing this value can be determined by a backtracking procedure (see [17]). In that example, the best score is 62 and the two segments detected as similar are `CTACTCTACT` and `CCAATCTACT`.

2.2 Parameterized version

SAMBA implements a modified version of the Smith and Waterman algorithm in order to cover a set of useful sequence alignment algorithms. The reader is referred to [9] for details on the parameterized equations.

Using a slight modification of equation (1), one can solve the following problems:

- find the alignment of `S1` and `S2` which maximizes a similarity measure with simple gap cost. This is the equation introduced by Needleman and Wunsch in [11]: the cost of k gaps is a simple function $G(k) = g \times k$ of the cost of one gap.
- locate similar subsequences between two sequences. This is probably the most useful algorithm for current research, since two biological sequences which present a small overall similarity may share surprising relationships on short segments.

In such a way, the most commonly used softwares, such as BLAST [1], FASTA [12] or SSEARCH [13], can be executed by SAMBA.

```
     C  C  A  A  T  C  T  A  C  T  A  C  T  G  C  T  T  G  C  A  G  T  A  C
A    0  0 10 10  0  0  0 10  0  0 10  0  0  0  0  0  0  0  0 10  0  0 10  0
G    0  0  0  1  1  0  0  0  1  0  0  1  0 10  0  0  0 10  0  0 20  0  0  1
T    0  0  0  0 11  0 10  0  0 11  0  0 11  0  1 10 10  0  1  0  0 30 10  0
C   10 10  0  0  0 21  1  1 10  0  2 10  0  2 10  0  1  1 10  0  0 10 21 20
C   10 20  1  0  0 10 12  0 11  1  0 12  1  0 12  1  0  0 11  1  0  0  1 31
G    0  1 11  0  0  0  1  3  0  2  0  0  3 11  0  3  0 10  0  2 11  0  0 11
A    0  0 11 21  1  0  0 11  0  0 12  0  0  0  2  0  0  0  1 10  0  2 10  1
G    0  0  0  2 12  0  0  0  2  0  0  3  0 10  0  0  0 10  0  0 20  0  0  1
G    0  0  0  0  0  3  0  0  0  0  0  0  0 10  1  0  0 10  1  0 10 11  0  0
G    0  0  0  0  0  0  0  0  0  0  0  0  0 10  1  0  0 10  1  0 10  1  2  0
C  .10.10  0  0  0 10  0  0 10  0  0 10  0  0 20  0  0  0 20  0  0  1  0 12
T    0 .1. 1  0 10  0 20  0  0 20  0  0 20  0  0 30 10  0  0 11  0 10  0  0
A    0  0.11.11  0  1  0 30 10  0 30 10  0 11  0 10 21  1  0 10  2  0 20  0
C   10 10  0 .2. 2 10  0 10 40 20 10 40 20 10 21  1  1 12 11  0  1  0  0 30
T    0  1  1  0.12. 0 20  0 20 50 30 20 50 30 20 31 11  1  3  2  0 11  0 10
C   10 10  0  0  0.22. 2 11 10 30 41 40 30 41 40 20 22  2 11  0  0  0  2 10
T    0  1  1  0 10  2.32.12  2 20 21 32 50 30 32 50 30 20 10  2  0 10  0  0
A    0  0 11 11  0  1 12.42.22 12 30 12 30 41 21 30 41 21 11 20  0  0 20  0
C   10 10  0  2  2 10  2 22.52.32 22 40 20 21 51 31 21 32 31 11 11  0  0 30
T    0  1  1  0 12  0 20 12 32.62.42 32 50 30 31 61 41 31 23 22  2 21  1 10
G    0  0  0  0  0  3  0 11 22 42 53 33 30 60 40 41 52 51 31 21 32 12 12  0
A    0  0 10 10  0  0  0 10 12 32 52 44 24 40 51 31 32 43 42 41 21 23 22  3
A    0  0 10 20  1  0  0 10  2 22 42 43 35 30 31 42 22 23 34 52 32 22 33 13
C   10 10  0  1 11 11  0  0 20 12 22 52 34 26 40 22 33 13 33 32 43 23 13 43
```

Fig. 3. An example of alignment matrix. Numbers followed by a dot belong to the best alignment

3 From the algorithm to a parallel architecture

In this section, we describe the parallel implementation of the Smith and Waterman algorithm, we explain how the results can be sent to the border processors, and we give the complexity of the algorithm.

3.1 Basics

The parallelization of string comparison algorithms on linear systolic arrays has been abundantly described in the literature. Readers interested by details are refered to [8] [6] [3] [10] [18].

The architecture of SAMBA (see Figure 4) is composed of identical processors, linearly arranged, which perform one step of the matrix computation described in the previous section. Data move from the left to the right. Typically, a parallel comparison of two sequences is performed as follows:

1. the array is initialized with one sequence (usually called the query sequence) at the rate of one character per processor;
2. the other sequence (bank sequence) is flushed to the left of the array and progresses every systolic cycle;
3. the result is collected on the rightmost processor when the last character of the second sequence is output.

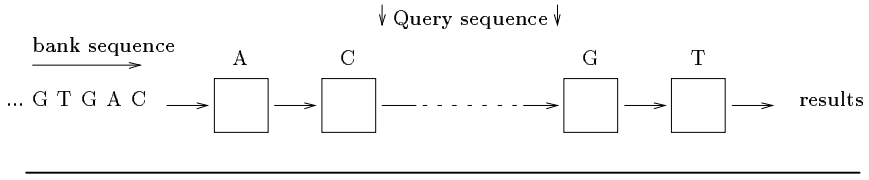

Fig. 4. Sequence comparison on a linear systolic array: one sequence (the query sequence) is stored into the array (one character per processor) and the other sequence flows from the left to right through the array. During each systolic step, one elementary matrix computation is performed on each processor. The result is available on the rightmost processor when the last character of the flowing sequence is output.

3.2 Getting the results

In a systolic array, only the border processors communicate with the "outside world". Hence, the results must necessarily be forwarded to the border to be output. In the present case, the computation consists of calculating a matrix in which the interesting values can appear anywhere: for example, the identification of similar segments requires the full matrix to be analyzed in order to detect high scores.

Forwarding values to the array extremities requires additional computations. The SAMBA algorithm is completed with the computation of two terms:

- $\texttt{MaxRow}(i,j)$, the maximum score $H(i,j)$ computed on the ith row of matrix H;
- $\texttt{IdxCol}(i,j)$, an index specifying the column where `MaxRow` is modified.

These two terms are computed using the following recurrence equations:

$$\texttt{MaxRow}(i,j) = \max \begin{cases} \texttt{MaxRow}(i,j-1) \\ H(i,j) \quad , \end{cases} \tag{2}$$

$$\texttt{IdxCol}(i,j-1) = \begin{cases} \texttt{IdxCol}(i,j-1) \textbf{ if } \texttt{MaxRow}(i,j-1) > H(i,j)) \\ \texttt{j} \textbf{ otherwise.} \end{cases} \tag{3}$$

Let N be the size of the array and i the systolic step number. During each systolic step, the rightmost processor of the array delivers `MaxRow`(i, N) and `IdxCol`(i, N). These informations enable the coordinates of local maxima to be found in matrix H.

3.3 Complexity

Let l_q be the length of the query sequence and l_b is the length of the bank sequence. The computation is performed in $l_q + l_b - 1$ systolic steps, providing a speed-up S_1 over sequential machines given by:

$$S_1 = \frac{l_q \times l_b}{l_q + l_b - 1} \quad .$$

When comparing one sequence against a bank of sequences, the speed-up may be increased by pipelining the bank sequences flowing through the array: when the last character of a sequence enters the array, the first character of the following sequence can be input during the next systolic cycle. For k sequences, $l_q + k \times l_b - 1$ systolic steps are required. The speed-up (S_k) is then given by:

$$S_k = \frac{k \times l_q \times l_b}{l_q + k \times l_b - 1} \quad .$$

When scanning a biological database, the number k of sequences is generally high and the speed-up S_k can be approximated by l_q (the length of the query sequence). This last case represents the application domain where SAMBA excels.

4 From the parallel architecture to the hardware

The design of SAMBA was decided in september 1994. The main design decision was to choose the technology of the systolic array. The choice was between a programmable or a fully dedicated architecture. It was decided to build a systolic array with fully dedicated processors, as the result would be about 10 times faster than a programmable version, and there would be no need to design a programming environment – a task which usually requires a lot of effort. Moreover, the algorithms to be implemented were very stables, and a parameterized architecture would make it possible to execute all the commonly used algorithms.

We now present the architecture of SAMBA (subsection 4.1), then we describe the design trajectory of its implementation in subsection 4.2.

4.1 Architecture of SAMBA

As mentioned previously, SAMBA is composed of a host workstation, a FPGA-interface and a full custom systolic array. This section describes in details each one of these elements.

Host workstation and I/O

The host of SAMBA is a DECstation DS5000/240 model, which uses a 40 MHz MIPS R3400 Risc processor. The I/O system is based on the TURBOchannel open interconnect developed by Digital and provides three slots for connecting optional peripherals. It runs on the 25 MHz memory system clock and has a peak bandwidth of 100 MB/s. It is connected to a custom I/O controller which interfaces the TURBOchannel on one side to several different devices on the other side, mostly controllers for particular types of peripherals or data communication interfaces, such as serial ports, SCSI controller, Ethernet, etc.

The SCSI controller can perform asynchronous or synchronous data transfers at up to 5 MB/s through DMA access. The hard disk drive available on the system has a capacity of 426 MB with a maximum bus bandwidth of 4 MB/s. This local disk is used to store the banks of biological sequences.

Host/array interface

The host/array interface uses the concept of PAM (Programmable Active Memories) introduced by Vuillemin et al. [2]. As mentioned by the authors, the purpose of a PAM is to implement a virtual machine which can be dynamically configured as a large number of specific hardware devices. The PAM is connected through two links to a host processor. A third connection allows a configuration bit stream to be downloaded into the PAM; after the configuration phase, the PAM behaves like an ASIC. The PAM may operate in different modes:

Stand-alone mode: the PAM is hooked to external systems through the external links;

Co-processor mode: the PAM is controlled by the host and specialized to speed-up some crucial computations;

Mixed mode: the PAM is both controlled by the host and connected to some specific hardware.

SAMBA uses the mixed mode. The external links are connected to the systolic array. The major role of the PAM is to provide the VLSI systolic array with data, and to filter the results on the fly.

The PAM used by SAMBA is the PeRLe-1 prototype board, a configurable co-processor organized around a central computational matrix of 16 Xilinx XC3090 FPGAs, surrounded by 4 memory banks for local storage, and 7 other FPGAs to implement switch and control functions. PeRLe-1 is connected to the workstation through one of the TURBOchannel extension slots, providing a very fast communication channel between the PAM and the CPU.

VLSI Systolic array

The array is a linear systolic array composed of 32 full custom chips – named API256 – distributed over two printed boards. One chip (designed in 1 micron CMOS technology, see figure 5) integrates 4 processors, leading to a systolic array

of 128 processors. Each chip contains about 100.000 transitors, and has a cycle of 100 ns. The datapath of the chip is 12 bits wide; this may be insufficient for some applications but is enough to validate the SAMBA prototype.

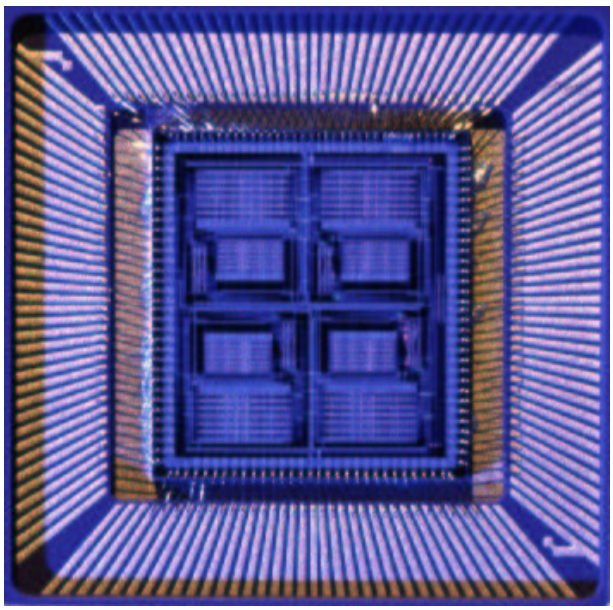

Fig. 5. Layout of the API256 chip

Each processor computes the recurrence equations presented in section 2 as well as the two auxiliary equations used for locating the results presented in section 3.2. A performance measure commonly used in computational biology is the number of *million of cell operation per second* (MCOPS), that is, the complete computation of one entry of matrix $H(i, j)$, including all comparisons, additions and maxima calculations. SAMBA performs such a computation in one systolic step. The peak speed of the 128 processor array is thus

$$P_p = Nb_{proc} \times freq = 128 \times 10^7 = 1280 \text{ MCOPS} \quad .$$

Furthermore, each processor contains a 32 word internal static memory for storing the substitution costs. This memory size is sufficient since a processor has to know a maximum of 20 substitution costs (they are 20 different animo acids) which refer to one character of the query sequence.

The systolic array is interfaced to the workstation through the direct custom links available on the PeRLe-1 board.

4.2 Design of SAMBA

SAMBA was designed as follows:

Functional specification: the specification of the sequence alignment application, including the parallel version of the algorithms, was written using the C-STOLIC language [14]. This language is a slight extension of C allowing parallel systolic algorithms to be simulated on Unix workstations. The C-stolic compiler provided three outputs: a C program emulating one processor of the systolic array, a C program emulating the interface between the host and the systolic array, and a C program for the host workstation.

Functional simulations: the functional specification was simulated on test data obtained from the application.

Architectural specification: the program run by the processors of the VLSI systolic array was translated "manually" into VHDL. Only the subset of VHDL accepted by the COMPASS VLSI synthesis tool was used, in order to be able to obtain a hardware specification.

Validation of the architecture: the architectural specification was run on the same data as the functional simulation, and validated by comparison.

Synthesis of the VLSI chip: the VLSI chip API256 was synthesized using the COMPASS hardware synthesis software, and sent to fabrication. The chips were fabricated by ES2 through the french multi-chip project. The 32 chips needed for the architecture were obtained in one run.

Board design: the systolic array was designed by assembling the 32 chips in two boards.

Interface design: thanks to the flexibility of the PAM, the interface was designed in two steps. First, the PAM was configured as a very simple link between the host and the systolic array, in order to test the full functionality of the architecture. In this configuration, the interface program produced by the C-STOLIC compiler was run by the host. Then, in order to reach the full speed of the systolic array, the PAM was configured to implement the interface program.

Application integration: finally, the whole application was integrated, by installing the application program on the workstation.

The design effort was 11 person/months: 1 month for functional specification and simulation, one month for architectural specification and simulation, 2 months for architectural synthesis, 2 month for chip testing, 0.5 month for board design, 4 month for interface design, and 0.5 month for final integration.

5 Application and speed of SAMBA

In this section we give some results obtained using SAMBA on a real intensive protein bank to bank comparison example. We first describe briefly the application, then we present and discuss the performance characteristics of SAMBA.

5.1 Application

SAMBA has been used continuously since December 1995. Several research projects are currently using this architecture. The most advanced one is described in [7], and can be summarized as follows.

When determining the sequence of some genome, some of the frames which are analyzed do not share significant similarities with protein sequences already known and stored in the databanks. These sequences are called "orphan sequences", and one problem for biologists is to find out why orphan sequences do not show similarity with databanks. Indeed, a sequence is said to be "orphan" if the computer program used for the comparison does not detect a significant similarity. Thus, this notion certainly depends on the program used.

The Smith and Waterman algorithm is known to be much more complex, and supposed to be more sensitive, than the programs commonly used on workstations, such as Fasta or Blast. Due to the complexity of this algorithm, no significant study has been performed to prove, or disprove, that it is worth using this technique. Using SAMBA, it was possible to make such a study.

The purpose was to compare a set of 814 proteins against release 31 of the databank SWISS-PROT using Smith and Waterman algorithm with different substitution matrices and different gap penalties. More precisely, the sequences to test belong to yeast protein sequences considered as orphan (i.e. with no similarities with other sequences) when compared to SWISS-PROT with software such as BLAST or FASTA. The idea was to evaluate the capability of the Smith and Waterman algorithm to find *parents* for some orphan sequences.

The study has shown that some of the orphan sequences were found to share probably significant similarities with sequences of SWISS-PROT, while these similarities were undetected by Blastp or Fasta. One can conclude that the Smith and Waterman algorithm is complementary to the other algorithms.

5.2 Performance characteristics of SAMBA

Bank to bank comparison

A sequential version of the Smith and Waterman algorithm is available in the the package FASTA as the program SSEARCH. To evaluate the speed of SAMBA, we compared the time needed by SSEARCH on the orphan sequence application.

SSEARCH was run on a DEC-Alpha workstation with a 21064 150MHz microprocessor. Measurements indicate that the computation of a one matrix cell is achieved in $0.25\,\mu$s – which corresponds to 4 MCOPS – when run as follows

```
ssearch -Q -b 20 -d 0 query_seq sprot31.fasta
```

Release 31 of SWISS-PROT contains exactly 43 470 sequences distributed on 15 335 248 amino acids. The 814 yeast sequences represent a total of 307 400 amino acids. The time t_{seq} for comparing this set of sequences against the bank is thus given by:

$$t_{seq} = 15\,335\,248 \times 307\,400 \times 0.25 \times 10^{-6} = 1\,178\,514\ s \quad .$$

This is equivalent to 327 hours (or 13 days and 15 hours) of non-stop computation. Knowing that this task should be repeated, at least, with three different substitution matrices and two different gap costs, such an experiment would have required nearly three months of intensive computation.

On SAMBA, the comparison of the 814 sequences against the bank is achieved in 1 hour and 45 minutes. The speed-up is thus 190 for this particular application. The time was measured using the UNIX command **`time`**. Thus, it is the total elapsed time, as it directly affects the user, including the time for reading the database from the disk and the time for re-computing on the workstation the exact score which could not be found by the array due to the 12-bit processor arithmetic (in that case the array indicates an overflow and the comparison is performed by the workstation). The bank to bank comparison, as illustrated by this example, fits completely the SAMBA functionalities, and the speed-up is therefore very high.

Scanning data bases

Other applications such as the scan of databases, that is to say, the comparison of one query sequence against one database, may also be performed very quickly, as we shall see.

Table 1 indicates the SSEARCH scan time (in second) on different workstations of the bank SWISS-PROT (release 31) with proteins of different length. It also indicates the speed-up when SAMBA is used instead. Obviously, the longer the

Query sequence length	10	30	100	300	1000	3000	10000
SAMBA	25	25	26	30	40	77	210
DEC-Alpha - 150 MHz CPU	57	120	350	1041	3468	11510	38450
speed-up	*2.3*	*4.8*	*13.5*	*34.7*	*86.7*	*150*	*183*
SUN-SPARC 5 - 110 MHz CPU	95	239	746	2215	7300	24269	80300
speed-up	*3.8*	*9.5*	*28.6*	*74*	*183*	*315*	*382*
DEC 5000/250 - 40 MHz CPU	182	548	1407	4054	12920	41169	131193
speed-up	*7.3*	*22*	*54*	*135*	*323*	*534*	*625*

Table 1. SSEARCH scan time (in seconds) of SWISS-PROT 31 for various length of the query sequence on SAMBA and on different workstations. The speed-up compared to SAMBA is also reported. These results show clearly that the speed-up increases with the length of the query sequence.

query sequence, the better the speed-up. This is mainly due to the restricted bandwidth of the I/O disk system which prevents the array from being fed at its maximum rate. Indeed, a short sequence does not require, on the array, the computation to be split into several passes. Consequently, the array is fed at the disk access rate, which is generally much slower than the array feeding rate. On the other hand, the comparison of a long query sequence – i.e., a sequence whose length is much higher than the number of processors of the systolic array – requires the computation to be partitioned into several passes which re-use the

data coming from the bank. The array average feeding rate is then substantially decreased and does not become a limitation factor.

6 Conclusion

The SAMBA prototype demonstrates that a dedicated full custom systolic array is very well suited for biological sequence comparison. This is a low cost solution compared to the implementations on massively parallel machines such as MPSEARCH [16]) or commercial available systems such as BIOCCELERATOR [4]).

The speed of SAMBA comes from the association of full custom components and FPGA devices. The 128 processors of the systolic array deliver a high computational power, while the FPGA components of the interface board manage efficiently the host/array communications, computation partitioning, data supply and result filtering.

The design effort for building SAMBA was less than one person/year, and is therefore very moderate. This was possible thanks to the use of synthesis tools for the design of the custom VLSI systolic array, and to the use of reconfigurable logic for the interface.

SAMBA is well suited to sequence analyses that require intensive computations, such as bank to bank comparison or scanning banks with long query sequences.

The SAMBA prototype is currently split into three distinct printed boards: the FPGA-interface board and two 64-processor boards. Today's integration density makes possible to fit SAMBA onto a single printed board. The chip could easily contains twice as many processors running at double speed, while the interface could be reduced to a few state of the art FPGA components associated with a few Mbytes of local memory. We estimate that the cost of SAMBA would roughly the same as the cost of a middle range workstation. In other words, the computational power of SAMBA is accessible to every laboratory which has to face a daily need of power for sequence analysis.

Acknowledgements

This work was partially funded by the French Research Group GREG (Groupement de Recherches et d'Etudes sur les Génomes) and the French Coordinated Research Program ANM (Architectures Nouvelles de Machines)

References

1. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. – Basic local alignment search tool. – *J. Mol. Biol*, 215:403–410, 1990.
2. P. Bertin, D. Roncin, and J. Vuillemin. – Programmable active memories: a performance assessment. – In F. Meyer, B. Monien, and A.L. Rosenberg, editors, *Parallel Architectures and their efficient use*, pages 119–130. LNCS, Springer-Verlag, oct 1992.

3. E. Chow, T. Hunkapiller, and J. Peterson. – Biological Information Signal Processor. – In *ASAP*, pages 144–160, sep 1991.
4. Compugen. – The bioccelerator machine. – Israel, 1993.
5. O. Gotoh. – An Improved Algorithm for Matching Biological Sequences. – *J. Mol. Biol.*, 162:705–708, 1982.
6. P. Guerdoux-Jamet and D. Lavenier. – Systolic filter for fast DNA similarity search. – In *ASAP'95*, Strasbourg, July 1995.
7. P. Guerdoux-Jamet and J.L. Risler. – Searching for a family to orphan sequences with SAMBA, a parallel hardware dedicated to biological applications. – *Biochemistry*, 1996.
8. D. Lavenier. – An Integrated 2D Systolic Array for Spelling Correction. – *Integration : the VLSI journal*, 15:97–111, August 1993.
9. D. Lavenier. – Samba. Systolic Accelerators for Molecular Biological Applications. – Internal Publication 988, IRISA, Irisa, Campus de Beaulieu, 35042, Rennes-Cedex, France, April 1996.
10. D. Lopresty and al. – Building and using a highly parallel programmable logic array. – *Computers*, pages 81–89, jan 1991.
11. 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.
12. W. R. Pearson and D.J. Lipman. – Improved tools for biological sequence comparison. – *Proc. Natl. Acad. Sci.*, 85:3244–3248, 1988.
13. W.R. Pearson. – Searching protein sequence libraries: comparison of the sensitivity and selectivity of the smith and waterman and fasta algorithms. – *Genomics*, 11:635–650, 1991.
14. F. Raimbault and D. Lavenier. – ReLaCS for Systolic Programming. – In *ASAP'93*, October 1993.
15. T.F. Smith and M.S. Waterman. – Identification of common molecular subsequences. – *J. Mol. Biol*, 147:195–197, 1981.
16. S.S. Sturrock and J.F. Collins. – Mpsearch version 1.3. – Technical report, University of Edinburgh, Biocomputing Research Unit, 1993.
17. M.S. Waterman. – *Mathematical Methods for DNA Sequences.* – CRC Press, Inc, 1989.
18. 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: VSLI in Computer and Processors*, pages 504–509. IEEE Computer Society Press, oct 1991.

This article was processed using the LaTeX macro package with LLNCS style