Parallel Processing for Scanning Genomic Data-Bases

D. Lavenier and J.-L. Pacherie [a] {*lavenier,pacherie*}*@irisa.fr*

[a]IRISA, Campus de Beaulieu,
35042 Rennes cedex, France

The scan of a genomic data-base aims to detect similarities between DNA or protein sequences. This is a time-consuming operation, especially when weak similarities are searched. Speeding up the scan can be managed using various strategies of parallelization. This paper presents two approaches carried on at IRISA: systolic and distributed parallelization.

1. Introduction

Scanning genomic data-bases is a common and often repeated task in molecular biology. The need for speeding up this treatment comes from the exponential growth of the banks (the genomic data-bases) of biological sequences: every year their size scaled by a factor ranging from 1.5 to 2. The scan operation consists in finding similarities between a particular sequence (called the *query* sequence) and all the sequences of a bank. This operation allows biologists to point out sequences sharing common subsequences. From a biological point of view, it leads to identify similar functionalities.

Similarities are detected by algorithms whose complexities are quadratic with respect to the length of the sequences. In practice, this time-consuming operation is reduced by introducing heuristics in the search algorithms. The main drawback is that the more efficient the heuristics (from the execution time point of view), the worse the quality of the results. Furthermore, some search algorithms cannot benefit from the heuristics.

Another approach to get high quality results in a short time goes through parallelization. In that case, one must determine the best approach among various possibilities: parallel computer, network of workstations, or dedicated hardware. Actually, the response is not universal; the three approaches provide equivalent numerical results; but from the user point of view, it may not bring the same level of satisfaction.

This paper discusses various parallel approaches that have been implemented and tested at IRISA for speeding up the scan of genomic data-bases. One is based on the systolization of the algorithms on a special-purpose VLSI coprocessor, while the other is based on the distribution of the computation. Both have their merits, but also their drawbacks.

In the next section, we present the data-base scanning problem and the two ways of parallelizing this computation. Sections 3 and 4 detail the two implementations and give some performance. Based on real experiments, and depending on various criteria, we conclude by discussing advantages and drawbacks for each approach.

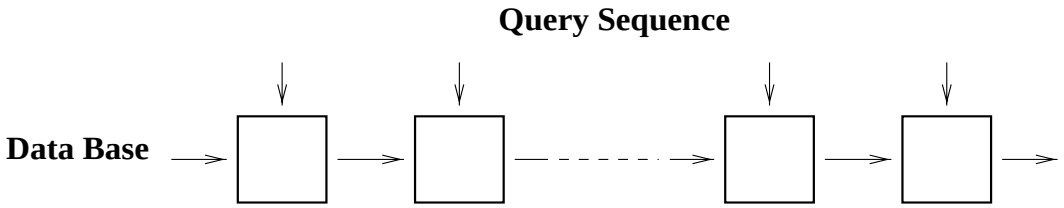

Figure 1. Principle of the systolic parallelization for scanning a genomic data-base: the query sequence is loaded in a linear systolic array (one character per processor) and the sequence of the bank (the data base) are pipelined through the array.

2. Parallelization of the scan of a genomic data-base

The scan of a genomic data-base involves three actors: a query sequence, a bank of sequences (the genomic data-base), and a method for finding the similarities between the query sequence and all the sequences in the bank. If NbS is the size of the bank (NbS = number of sequences), then the computation consists in NbS pairwise comparisons which give a score indicating the value of the best similarity found between the two sequences. The final result of the scan is a list of sequences having the best scores. Additional computations are usually performed to locate the similarity areas, but the computation time of this last operation is negligible with respect to the NbS pairwise comparisons.

Thus, speeding up the scan of a genomic data-base reduces essentially to speeding up the NbS pairwise comparisons. This can be achieved by parallelizing the computation using two approaches:

1. computing the pairwise comparison task on a dedicated systolic array: in that case, the search algorithm is parallelized, and the pairwise comparisons are performed sequentially.

2. splitting the data-bases into P sub-data-bases and performing the computation on a programmable parallel structure of P nodes: in that case, the search algorithm is executed sequentially, and P pairwise comparisons are performed in parallel.

The second approach can still be refined by considering two programmable parallel structures: networks of workstations and massively parallel computers. The two following sections respectively present the dedicated approach and the programmable parallel structures on which the scan of the genomic data-bases have been implemented and tested.

3. Systolic parallelization

The pairwise sequence comparison problem is usually solved by dynamic programming methods. The great advantages of this approach are the high quality of the results and the efficiency of the parallelization on systolic arrays [3]. Schematically (see figure 1), the process of comparing two sequences consists of loading one sequence in a linear systolic array (one character per cell) and sending the other horizontally, character by character,

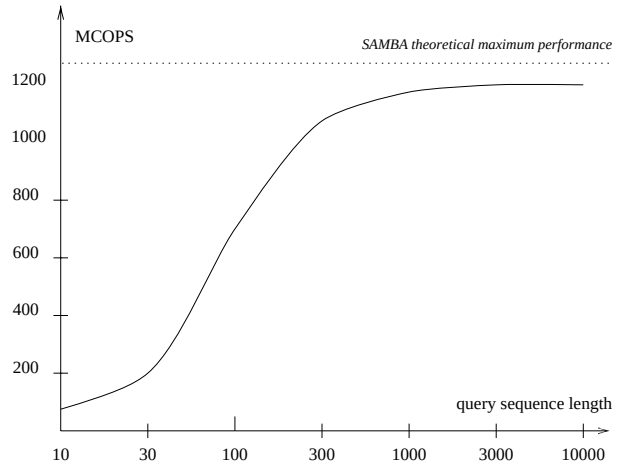

Figure 2. Comparison of a DNA query sequence against the virus section of the GenBank data-base: the curve reports the average MCOPS for different lengths of the query sequence. The longer the query sequence the better the SAMBA performance. This is due to the I/O disk system which prevents the accelerator to be fed at its maximum rate.

on each systolic cycle. If l_Q is the length of the query sequence and $l_{DB}i$ the length of the i-th sequence of the data-base, the pairwise comparison is performed in $l_Q + l_{DB}i - 1$ systolic cycles, instead of $l_Q \times l_{DB}i$ steps on a sequential processor.

The linear systolic structure has been implemented on a prototype called SAMBA (Systolic Accelerator for Molecular Biological Applications) [4,5]. The machine houses 128 dedicated full custom VLSI processors. The array is connected to a standard workstation through a FPGA interface which has the major role of managing the partitioning of the computation at the clock rate of the array.

As a matter of fact, and as explained above, comparing a query sequence against a data-base ideally assumes an array whose size is equal to the length of the query sequence. In practice, this never happens: the query sequence is 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 sequences are loaded in the array. Then, the entire data-base 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 merged with the data-base, and sent again to the array. The process is iterated until the end of the query sequence is reached.

The figure 2 shows the performance of SAMBA expressed in millions of computation cells per second (MCOPS) as a function of the length of the query sequence. This is the base unit which is traditionally taken; it represents the computation of one recursion of the dynamic programming algorithm. In that example, the scan of the virus section of the GenBank data-base has been made for different lengths of DNA query sequences. One may note that the longer the query sequence the better the performance. 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: 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.

Each processor performs 10 MCOPS, leading to a SAMBA peak performance of 1280 MCOPS. In other words, the scan of a genomic data-base can be done in a few dozens of seconds. For instance, the scan of a protein data-base (SWISS-PROT, release 34) with a query sequence of 1000 amino acids using an efficient algorithm [11] [10] is performed in approximately 30 seconds. By comparison, the fastest sequential implementation (using the same algorithm) requires more than 15 minutes on a 167 MHz UltraSparc workstation [2], where the comparison routine has been tuned to exploit efficiently the micro-parallelism provided by the VIS instruction set of the microprocessor.

4. Distributed Computations

Our strategy for distributing the computation has been driven by the following criteria: portability, load balancing, and adaptability.

Portability: The portability ensures the application to be able to run on a wide range of platforms, from local networks to parallel computers. Each platform must be exploited for its own characteristics without focusing on the code application and the user interface. For this purpose, we use a basic communication library (POM [8]) along with a parallel virtual machine model. This model exposes a full connected network with blocking FIFO communication channels, and is similar to the ones provided by PVM and MPI. The reason to choose POM is that it provides less but optimized functionalities.

Load balancing strategy: The load balancing strategy has to split the data-base into pieces and to send them as quickly as possible towards the slave processes. Each message sent by the master process is then composed of several sequences whose number depends on the performance of the communication channel. The point is that the communication routines would stop the master if it attempts to feed a busy slave with too many sequences while other slaves need to be fed. On the other hand, small messages require a higher level of reactivity of the master to prevent starvation of some of the slaves. The problem is to find the appropriate size of the message to avoid overloading of the communication channel and the starvation of the slaves. Our strategy is to let the slaves inform the master that they will need sequences to process. This information is sent before the slaves have completely finished their current work. On receiving this information, the master starts the preparation of a new set of sequences and sends it. This mechanism allows the communication channel to be used as a temporary storage for the transmission of the sequences.

Adaptability: the calculation of the number of sequences to send is made each time the master has to feed a particular slave, and is performed independently for each slave. In other words, the number of sequences sent to each slave varies over the time. The performance evaluation of the slave communication channel is made using the time spent by the master to send a message. According to this evaluation, the next volume of sequences remains unchanged, is increased or decreased. To prevent this strategy from

causing eventual network thrashing behavior, a delay is applied before attempting to modify the size of the messages. This self regulation makes possible both variations of performance between the processing units and variations of the computational capability of each slave to be taken into account.

This strategy has been evaluated over various platforms. In order to establish comparison, we keep the following characteristics: the computer architecture, the number of processing units, and the communication network. The table below reports the average performance (in MCOPS) for scanning a protein data-base (SWISS-PROT, release 34) using query sequences of different lengths.

Architecture	Network	Nb Procs	MCOPS	*Threshold*	Speedup	Scalability Mode
Origin2000	-	1	8.9	250	-	FS
Origin2000	HIPPI	4	28	200	3.15	FS
Origin2000	HIPPI	8	62	110	6.97	FS
UltraSPARC	-	1	4.2	60	-	FS
UltraSPARC	Ethernet	4	9.4	200	2.20	PS
UltraSPARC	Myrinet	4	17	110	4.00	FS
UltraSPARC	Ethernet	8	39	500	9.30	PS
PentiumPro	-	1	3.6	60	-	FS
PentiumPro	Ethernet	4	13	90	3.60	PS
PentiumPro	Ethernet	8	28	170	7.78	PS

Table 1: Measures for various platform configurations are reported. The scan of the same data-base has been performed for various query sequence lengths. The threshold corresponds to the query sequence length from which the better speed-up is achieved.

First of all, the more number of nodes the better the speed-up. We can ever notice a *superlinear* speed-up (UltraSPARC, Ethernet, 8 processors)! It can be explained by the overlap between the effective computation (the sequence comparison) and the memory management (access to the data-base): in the distributed version, the master gets data from the genomic bank while slaves are concurrently performing sequence comparison. When only one node is involved these actions are performed sequentially.

Furthermore, the measures we have taken pointed out an interesting observation for the predictability of a specific platform: whatever the characteristics of the platforms, the executions always share the same basic behavior. The figure 3 (left side) highlights this behavior for both the MCOPS and the execution-time, regardless of the architecture. The curves are both composed of two phases delimited by a **threshold** point B.

On the left side of B, the efficiency (measured in MCOPS) continuously increases with the size of the query sequence while the time spent to process the whole bank remains constant. The phenomenon is due to an underloading of the slaves, that is some slaves are starving during the execution. Increasing the size of the query sequence leads to more calculations for each slave, and then more time is given to the master to send its messages. At the threshold point, the slaves are never starving and the application has reached the maximum of its potentiality. The combination of the size of the query sequence, the number of processors, the speed of the network and the power of processing units has

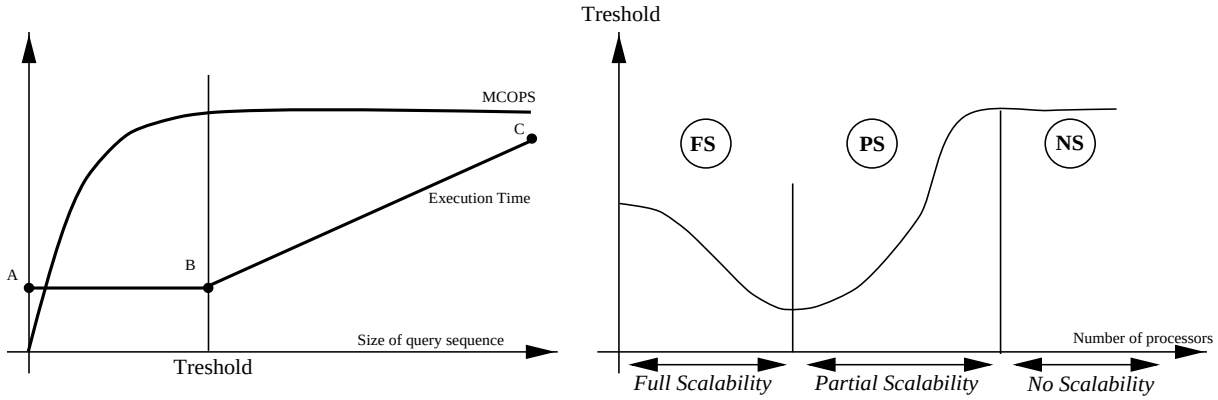

Figure 3. Performance behavior: whatever the platform, the MCOPS and the execution-time behave identically. On the first phase, the efficiency increases proportionally to the query sequence length. At point B the platform has reached the maximum of its possibilities.

reached the perfect balance. From this point, the execution time increases as a linear function of the length of the query sequence while the MCOPS remains stable.

We can also focus on the impact of the number of processing units on the performance for a given architecture. For each configuration it may be observed a global behaviour characterized by the evolution of the threshold point. This evolution is presented on the right side of the figure 3. There is a three-part curve. The first part (*Full Scalability*), outlines a situation where the network is sufficiently efficient to allow reduction of the threshold. Full scalability means that more processors increase the speedup and decrease the threshold point. During the next stage (*Partial Scalability*), increasing the number of processors allows the speed-up to be increased, but the threshold point is moved to a higher value. This means that the network is a bottleneck that limits the efficiency of the application. Thus, to get better performance, an improvement of the network is a better choice than increasing the number of processors. On the last stage (*No Scalability*), increasing the number of processors is definitively useless. Indeed, the master process is unable to manage the whole set of slaves.

To conclude this part, we can say that other strategies, like a static split of the data-base over a set of processing units [1], might provide better performance. However, we have found that those strategies usually rely on requirements on the underlying distributed architecture (number and power of processing units for example). Besides they suppose that the platform used is exclusively dedicated to the scanning of the data-base. On the other hand, our solution follows a much more adaptable approach. We are able to match new configurations (additional processing units, network and CPU improvements, data-base change, etc.) without any modification of the code and automatically make the most of the improvements. Besides, according to the needs of the biologists we offer an estimation of the more convenient platform characteristics (number of processing units, network efficiency). The threshold curve gives informations for platform improvements while the MCOPS measure and execution-time curves offer an estimation of performance

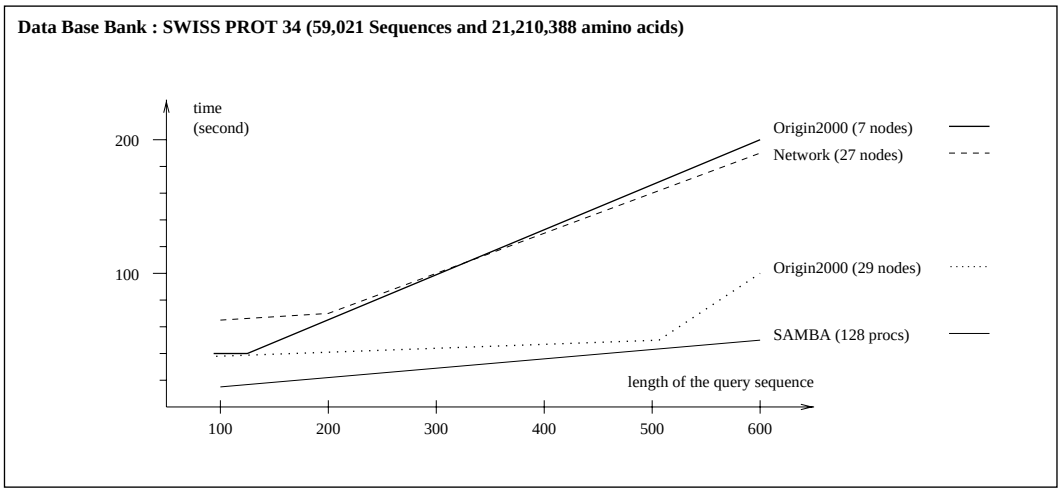

Figure 4. Times for scanning the SWISS-PROT protein data-base: the reported time is the total elapsed time, as it directly affects the user; it includes particularly the time for reading the data-base from the disk.

of a given configuration.

5. Discussion and Conclusion

For comparing performance between the systolic and the distributed approach, measurements have been conducted using the same input data and the same search algorithm (the Smith and Waterman algorithm [11]). This algorithm is known for its high quality, but is never used due to its expensive execution time. The figure 1 shows the average time for scanning a typical protein data-base (SWISS-PROT, release 34); different lengths of the query sequence are considered for the three following approaches: SAMBA, an Ethernet network of 27 heterogeneous SUN workstations (15 UltraSparc, 2 Sparc SS20, 10 Sparc SS5), and a parallel computer, Origin2000 from SiliconGraphics [6] with 7 and 29 nodes (processor R10 000, 190 MHz).

One must now consider the best solution for a biological laboratory which wishes to improve the time for scanning the genomic data-bases. Many criteria have to be considered: the size of the sequences submitted to the scan, the amount of sequences which must be treated daily, the computer resources available locally, the financial policy of the laboratory, the cost of managing the hardware and software, etc.

Focusing on the speed, the best score is achieved both by the Origin2000 parallel computer (using 29 nodes) or by the SAMBA dedicated machine. If we consider now the price of the two systems, the SAMBA solution, as a pluggable workstation device, is undoubtedly the best choice if this solution would be commercially available. The cost of a PCI SAMBA board is estimated to be less than $ 10,000 while a 32 processor Origin2000 parallel computer goes beyond $ 1 million !

Now, if the need is sporadic, even for processing long query sequences, it is probably

better to use the local computer resources and distribute the computation over a few machines. The computation time will be longer compared to a dedicated hardware or a parallel computer, but it will not constitute a bottleneck.

But we must go beyond the current needs. The point is that the size of the genomic data-bases are growing exponentially, and that this growth is superior to that of the micro-processors. Even if sequential implementations can be sporadically improved by using new microprocessor features (such the MMX instructions [2,7]), speed-up will mainly come from the increasing clock frequency (1.25 per year [9]); but this growth rate won't be enough to sustain the data-base explosion. In the future, more parallelism will be needed.

Designing larger linear systolic arrays or using larger parallel structures are no longer valid: the optimal size of the systolic arrays for scanning a data-base is the length of the query sequence; beyond that, no speed-up is achieved. The cost and the maintenance of a 100 processor parallel machine prohibits its use in the biological laboratory environment. For the next decades, the solution we advocate is a mixed approach between dedicated and distributed solution, that is plugging SAMBA-like boards into dedicated networked workstations. In that case, the efficiency of the SAMBA cards imposes the use of a high speed network such as ATM or Myrinet.

REFERENCES

1. E. Glemet and J.J. Codani. LASSAP, a LArge Scale Sequence compArison Package. *CABIOS*, 13(2):137–143, 1997.
2. A. Wozniak. Using video-oriented instructions to speed up sequence comparison. *CABIOS*, 13(2):145–150, 1997.
3. D. Lavenier. Dedicated Hardware for Biological Sequence Comparison. *Journal of Universal Computer Science*, 2(2):77–86, 1996.
4. D. Lavenier. SAMBA: Systolic Accelerator for Molecular Biological Applications. Technical Report RR 2845, INRIA, 1996.
5. P. Guerdoux-Jamet, D. Lavenier, C. Wagner and P. Quinton. Design and Implementation of a Parallel Architecture for Biological Sequence Comparison. In *LNCS 1123 (EURO-PAR'96)*, Lyon, France, 1996.
6. SiliconGraphics. The perfect system for evolving compute, memory, and I/O. http://www.sgi.com/Products/hardware/servers/products/Origin2000Desk.html, 1996.
7. B. Alpern, L. Carter and K.S. Gatlin. Microparallelism and High-Performance Protein Matching. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, San Diego, California, USA, 1995.
8. F. Guidec and Y. Mahéo. POM: a Parallel Observable Machine. In *Proceedings of PARCO'95*, Gent, Belgium, 1995.
9. J.E. Vuillemin. On computing power. *LNCS*, 782:69–86, 1993.
10. O. Gotoh. An Improved Algorithm for Matching Biological Sequences. *J. Mol. Biol.*, 162:705–708, 1982.
11. T.F. Smith and M.S. Waterman. Identification of Common Molecular Subsequences. *J. Mol. Biol*, 147:195–197, 1981.