Scanning Biosequence Databases on a Hybrid Parallel Architecture

Bertil Schmidt [1], Heiko Schröder [1] and Manfred Schimmler [2]

[1] School of Computer Engineering, Nanyang Technological University, Singapore 639798
{asbschmidt,asheiko}@ntu.edu.sg
[2] Institut für Datenverarbeitungsanlagen, TU Braunschweig, Hans-Sommer-Str. 66, 38106 Braunschweig, Germany
masch@ida.ing.tu-bs.de

Abstract. Molecular biologists frequently scan sequence databases to detect functional similarities between proteins. Even though efficient dynamic programming algorithms exist for the problem, the required scanning time is still very high, and because of the exponential database growth finding fast solutions is of highest importance to research in this area. In this paper we present a new approach to high performance database scanning on a hybrid parallel architecture to gain supercomputer power at low cost. The architecture is built around a PC-cluster linked by a high-speed network and massively parallel processor boards connected to each node. We present the design of a parallel sequence comparison algorithm in order to derive an efficient mapping onto this architecture. This results in a database scanning implementation with significant runtime savings.

1 Introduction

Scanning protein sequence databases is a common and often repeated task in molecular biology. The need for speeding up this treatment comes from the exponential growth of the biosequence banks: every year their size scaled by a factor 1.5 to 2. The scan operation consists in finding similarities between a particular 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 functionality.

Comparison algorithms whose complexities are quadratic with respect to the length of the sequences detect similarities between the query sequence and a subject sequence. One frequently used approach to speed up this time consuming operation is to introduce heuristics in the search algorithms [1]. The main drawback of this solution is that the more time efficient the heuristics, the worse is the quality of the results [11].

Another approach to get high quality results in a short time is to use parallel processing. There are two basic methods of mapping the scanning of protein sequence

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 360-370, 2001.
© Springer-Verlag Berlin Heidelberg 2001

databases to a parallel processor: one is based on the systolisation of the sequence comparison algorithm, the other is based on the distribution of the computation of pairwise comparisons. Systolic arrays have been proven as a good candidate structure for the first approach [4,12], while supercomputers and networks of workstations are suitable architectures for the second [9]. This paper presents a new approach to high performance sequence database scanning that combines both strategies on a new hybrid parallel architecture, in order to achieve even higher speed.

Hybrid computing is the combination of the SIMD and MIMD paradigm within a parallel architecture, i.e. within the processors of a computer cluster (MIMD) massively parallel processor boards (SIMD) are installed in order to accelerate compute intensive regular tasks. The driving force and motivation behind hybrid computing is the price/performance ratio. Using PC-cluster as in the Beowulf approach is currently the most efficient way to gain supercomputer power. Installing in addition massively parallel processor cards within each PC can further improve the cost/performance ratio significantly. We designed a parallel sequence comparison algorithm in order to fit the characteristics of the hybrid architecture for a protein sequence database scanning application. Its implementation is described on our hybrid system consisting of Systola 1024 cards within the 16 PCs of a PC-cluster connected via a Myrinet switch.

This paper is organised as follows. In Section 2, we introduce the basic sequence comparison algorithm for database scanning and highlight previous work in parallel sequence comparison. Section 3 provides a description of our hybrid architecture. The new parallel algorithm and its mapping onto the hybrid architecture are explained in Section 4. The performance is evaluated and compared to previous implementations in Section 5. Section 6 concludes the paper with an outlook to further research topics.

2 Parallel Sequence Comparison

Surprising relationships have been discovered between protein sequences that have little overall similarity but in which similar subsequences can be found. In that sense, the identification of similar subsequences is probably the most useful and practical method for comparing two sequences. The Smith-Waterman (SW) algorithm [17] finds the most similar subsequences of two sequences (the local alignment) by dynamic programming.

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

Consider two strings $S1$ and $S2$ of length $l1$ and $l2$. To identify common subsequences, the SW algorithm computes the similarity $H(i,j)$ of two sequences

ending at position i and j of the two sequences $S1$ and $S2$. The computation of $H(i,j)$ is given by the following recurrences:

$$H(i,j) = \max\{0, E(i,j), F(i,j), H(i-1,j-1)+Sbt(S1_i,S2_j)\}, \quad 1\le i\le l1,\ 1\le j\le l2$$
$$E(i,j) = \max\{H(i,j-1)-\alpha, E(i,j-1)-\beta\}, \quad 0\le i\le l1,\ 1\le j\le l2$$
$$F(i,j) = \max\{H(i-1,j)-\alpha, E(i-1,j)-\beta\}, \quad 1\le i\le l1,\ 1\le j\le l2$$

where Sbt is a character substitution cost table. Initialisation of these values are given by $H(i,0)=E(i,0)=H(0,j)=F(0,j)=0$ for $0\le i\le l1$, $0\le j\le l2$. Multiple gap costs are taken into account as follows: α is the cost of the first gap; β is the cost of the following gaps. Each position of the matrix H is a similarity value. The two segments of $S1$ and $S2$ producing this value can be determined by a backtracking procedure. Fig. 1 illustrates an example

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

Fig. 1: Example of the SW algorithm to compute the local alignment between two DNA sequences ATCTCGTATGATG and GTCTATCAC. The matrix $H(i,j)$ is shown for the computation with gap costs α=1 and β=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 corresponding alignment (shaded cells), the two subsequences TCGTATGA and TCTATCA

The dynamic programming calculation can be efficiently mapped to a linear array of processing elements. A common mapping is to assign one processing element (PE) to each character of the query string, and then to shift a subject sequence systolically through the linear chain of PEs (see Fig. 2). If $l1$ is the length of the first sequence and $l2$ is the length of the second, the comparison is performed in $l1+l2-1$ steps on $l1$ PEs, instead of $l1\times l2$ steps required on a sequential processor. In each step the computation for each dynamic programming cell along a single diagonal in Fig. 1 is performed in parallel.

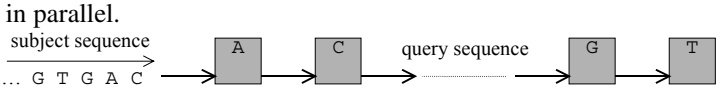

Fig. 2: Sequence comparison on a linear processor array: the query sequence is loaded into the processor array (one character per PE) and a subject sequence flows from left to right through the array. During each step, one elementary matrix computation is performed in each PE

A number of parallel architectures have been developed for sequence analysis. In addition to architectures specifically designed for sequence analysis, existing programmable sequential and parallel architectures have been used for solving sequence problems.

Special-purpose systolic arrays can provide the fastest means of running a particular algorithm with very high PE density. However, they are limited to one single algorithm, and thus cannot supply the flexibility necessary to run a variety of algorithms required analyzing DNA, RNA, and proteins. P-NAC was the first such machine and computed edit distance over a four-character alphabet [10]. More recent examples, better tuned to the needs of computational biology, include BioScan and SAMBA [4,12].

Reconfigurable systems are based on programmable logic such as field-programmable gate arrays (FPGAs), e.g. Splash-2, Biocellerator [5,6], or custom-designed arrays, e.g. MGAP [2]. They are generally slower and have far lower PE densities than special-purpose architectures. They are flexible, but the configuration must be changed for each algorithm, which is generally more complicated than writing new code for a programmable architecture.

Our approach is based on instruction systolic arrays (ISAs). ISAs combine the speed and simplicity of systolic arrays with flexible programmability [7], i.e. they achieve a high performance cost ratio and can at the same time be used for a wide range of applications, e.g. scientific computing, image processing, multimedia video compression, computer tomography, volume visualisation and cryptography [13-16]. The Kestrel design presented in [3] is close to our approach since it is also a programmable fine-grained parallel architecture. Unfortunately, its topology is purely a linear array. This has limited so far its widespread usage to biosequence searches and a computational chemistry application.

3 The Hybrid Architecture

We have built a hybrid MIMD-SIMD architecture from general available components (see Fig. 3). The MIMD part of the system is a cluster of 16 PCs (PentiumII, 450 MHz) running Linux. The machines are connected via a Gigabit-per-second LAN (using Myrinet M2F-PCI32 as network interface cards and Myrinet M2L-SW16 as a switch). For application development we use the MPI library MPICH v. 1.1.2.

For the SIMD part we plugged a Systola 1024 PCI board [8] into each PC. Systola 1024 contains an ISA of size 32×32. The ISA [7] is a mesh-connected processor grid, where the processors are controlled by three streams of control information: instructions, row selectors, and column selectors (see Figure 4). The instructions are input in the upper left corner of the processor array, and from there they move step by step in horizontal and vertical direction through the array. The selectors also move systolically through the array: the row selectors horizontally from left to right, column selectors vertically from top to bottom. Selectors mask the execution of the instructions within the processors, i.e. an instruction is executed if and only if both selector bits, currently in that processor, are equal to one. Otherwise, a no-operation is executed.

Every processor has read and write access to its own memory (32 registers). Besides that, it has a designated *communication register* (*C-register*) that can also be read by the four neighbour processors. Within each clock phase reading access is

always performed before writing access. Thus, two adjacent processors can exchange data within a single clock cycle in which both processors overwrite the contents of their own C-register with the contents of the C-register of its neighbour. This convention avoids read/write conflicts and also creates the possibility to broadcast information across a whole processor row or column with one single instruction. This property can be exploited for an efficient calculation of row broadcasts and row ringshifts, which are the key-operations in the algorithm in Section 4.

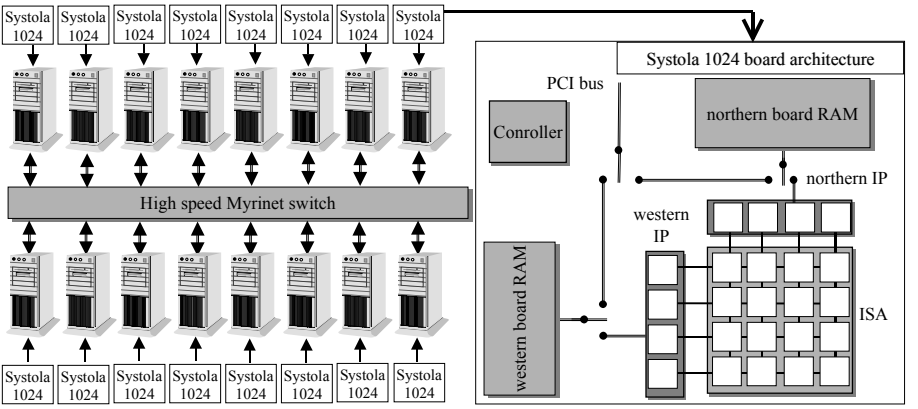

Fig. 3: Architecture of our hybrid system: A cluster of 16 PCs with 16 Systola 1024 PCI boards (left). The data paths in Systola 1024 are depicted on the right

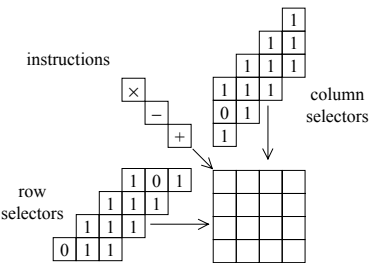

Fig. 4: Control flow in an ISA

For the fast data exchange with the ISA there are rows of intelligent memory units at the northern and western borders of the array called *interface processors* (IPs). Each IP is connected to its adjacent array processor for data transfer in each direction. The IPs have access to an on-board memory, those at the northern interface chips with the northern board RAM, and those of the western chips with the western board RAM. The northern and the western board RAM can communicate bidirectionally with the PC memory over the PCI bus.

At a clock frequency of f = 50 MHz and using a word format of m=16 bits each (bitserial) processor can execute $f/m = 50/16 \cdot 10^6 = 3.125 \cdot 10^6$ word operations per second. Thus, one board with its 1024 processors performs up to 3.2 GIPS. This adds up to a theoretical peak performance of 51.2 GIPS for 16 boards inside the cluster.

4 Mapping of Sequence Comparison to the Hybrid Architecture

The mapping of the database scanning application on our hybrid computer consists of two forms of parallelism: a fine-grained parallelelisation on Systola 1024 and a coarse-grained on the PC-cluster. While the Systola implementation parallelises the cell computation in the SW algorithm, the cluster implementation splits the database into pieces and distributes them among the PCs using a suitable load balancing strategy. We will now describe both parts in more detail.

Systolic parallelisation of the SW algorithm on a linear array is well-known. In order to extend this algorithm to a mesh-architecture, we take advantage of ISAs capabilities to perform row broadcast and row ringshift efficiently. Since the length of the sequences may vary (several thousands in some cases, however commonly the length is only in hundreds), the computation must also be partitioned on the $N \times N$ ISA. For sake of clarity we firstly assume the processor array size N^2 to be equal to the query sequence length M, i.e. $M=N^2$.

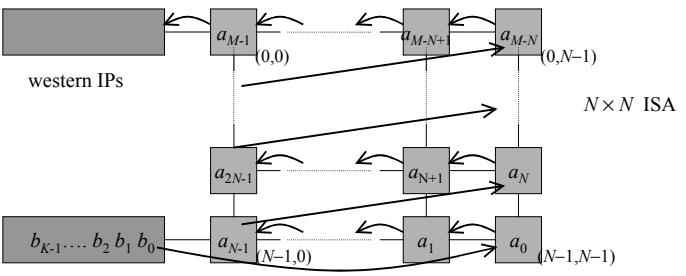

Fig. 5: Data flow for aligning two sequences A and B on an $M=N\times N$ ISA: A is loaded into the ISA one character per PE and B is completely shifted through the array in $M+K-1$ steps. Each character b_j is input from the lower western IP and results are written into the upper western IP

Fig. 5 shows the data flow in the ISA for aligning the sequences $A = a_0a_1\ldots a_{M-1}$ and $B = b_0b_1\ldots b_{K-1}$, where A is the query sequence and B is a subject sequence of the database. As a preprocessing step, symbol a_i, $i = 0,\ldots,M-1$, is loaded into PE (m,n) with $m = N-i$ div $N-1$ and $n = N-i$ mod $N-1$ and B is loaded into the lower western IP. After that the row of the substitution table corresponding to the respective character is loaded into each PE as well as the constants α and β. B is then completely shifted through the array in $M+K-1$ steps as displayed in Fig. 5.

In iteration step k, $1 \le k \le M+K-1$, the values $H(i,j)$, $E(i,j)$, and $F(i,j)$ for all i, j with $1 \le i \le M$, $1 \le j \le K$ and $k=i+j-1$ are computed in parallel in the PEs (m,n) with $m = N-i$ div $N-1$ and $n = N-i$ mod $N-1$. For this calculation PE (m,n) receives the values $H(i-1,j)$, $F(i-1,j)$, and b_j from its eastern neighbour $(m,n+1)$ if $n < N-1$, or from PE $(m+1,0)$ if $n = N-1$ and $m < N-1$, while the values $H(i-1,j-1)$, $H(i,j-1)$, $E(i,j-1)$, a_i, α, β, and $Sbt(a_i,b_j)$ are stored locally. The lower right PE $(N-1,N-1)$ receives b_j in steps j with $0 \le j \le K-1$ from the lower western IP and zeros otherwise.

These routing operations can be accomplished in constant time on the ISA. Thus, it takes $M+K-1$ steps to compute the alignment cost of the two sequences with the SW

algorithm. However, notice that after the last character of B enters the array, the first character of a new subject sequence can be input for the next iteration step. Thus, all subject sequences of the database can be pipelined with only one step delay between two different sequences. Assuming k sequences of length K and K=O(M), we compute K sequence alignments in time O($K \cdot M$) using O(M) processors. As the best sequential algorithm takes O($K \cdot M^2$) steps, our parallel implementation achieves maximal efficiency.

Because of the very limited memory of each PE, only the highest score of matrix H is computed on Systola 1024 for each pairwise comparison. Ranking the compared sequences and reconstructing the alignments are carried out by the front end PC. Because this last operation is only performed for very few subject sequences, its computation time is negligible. In our ISA algorithm the maximum computation of the matrix H can be easily incorporated with only a constant time penalty: After each iteration step all PEs compute a new value *max* by taking the maximum of the newly computed H-value and the old value of *max* from its neighbouring PE. After the last character of a subject sequence has been processed in PE (0,0), the maximum of matrix H is stored in PE (0,0), which is written into the adjacent western IP.

So far we have assumed a processor array equal in size of the query sequence length (M=N^2). In practice, this rarely happens. Assuming a query sequence length of $M = k \cdot N$ with k a multiple of N or N a multiple of k, the algorithm is modified as follows:

1. k ≤ N: In this case we can just replicate the algorithm for a $k \times N$ ISA on an $N \times N$ ISA, i.e. each $k \times N$ subarray computes the alignment of the same query sequence with different subject sequences.

2. k > N: A possible solution is to assign k/N characters of the sequences to each PE instead of one. However, in this case the memory size has to be sufficient to store k/N rows of the substitution table (20 values per row, since there are 20 different amino acids), i.e. on Systola 1024 it is only possible to assign maximally two characters per PE. Thus, for $k/N > 2$ it is required to split the sequence comparison into $k/(2N)$ passes: The first $2N^2$ characters of the query sequence are loaded into the ISA. The entire database then crosses the array; the H-value and F-value computed in PE (0,0) in each iteration step are output. In the next pass the following $2N^2$ characters of the query sequence are loaded. The data stored previously is loaded together with the corresponding subject sequences and sent again into the ISA. The process is iterated until the end of the query sequence is reached. Note that, no additional instructions are necessary for the I/O of the intermediate results with the processor array, because it is integrated in the dataflow (see Fig. 5).

For distributing of the computation among the PCs we have chosen a static split load balancing strategy: A similar sized subset of the database is assigned to each PC in a preprocessing step. The subsets remain stationary regardless of the query sequence. Thus, the distribution has only to be performed once for each database and does not influence the overall computing time. The input query sequence is broadcast to each PC and multiple independent subset scans are performed on each Systola 1024 board. Finally, the highest scores are accumulated in one PC. This strategy provides the best performance for our homogenous architecture, where each

processing unit has the same processing power. However, a dynamic split load balancing strategy as used in [9] is more suitable for heterogeneous environments.

5 Performance Evaluation

A performance measure commonly used in computational biology is *millions of dynamic cell updates per second* (MCUPS). A CUPS represents the time for a complete computation of one entry of the matrix H, including all comparisons, additions and maxima computations. To measure the MCUPS performance on Systola 1024, we have given the instruction count to update two H-cells per PE in Table 1.

Table 1: Instruction count to update two H-cells in one PE of Systola 1024 with the corresponding operations

Operation in each PE per iteration step	**Instruction Count**
Get $H(i-1,j)$, $F(i-1,j)$, b_j, max_{i-1} from neighbour	20
Compute $t = \max\{0, H(i-1,j-1) + Sbt(a_i,b_j)\}$	20
Compute $F(i,j) = \max\{H(i-1,j)-\alpha, F(i-1,j)-\beta\}$	8
Compute $E(i,j) = \max\{H(i,j-1)-\alpha, E(i,j-1)-\beta\}$	8
Compute $H(i,j) = \max\{t, F(i,j), E(i,j)\}$	8
Compute $max_i = \max\{H(i,j), max_{i-1}\}$	4
Sum	**68**

Because new H-values are computed for two characters within 68 instruction in each PE, the whole 32×32 processor array can perform 2048 cell updates in the same time. This leads to a performance of $(2048/68) \times f$ CUPS = $(2048/68) \times (50/16) \times 10^6$ CUPS = 94 MCUPS. Because MCUPS does not consider data transfer time and query length, it is often a weak measure that does not reflect the behaviour of the complete system. Therefore, we will use execution times of database scans for different query lengths in our evaluation.

The involved data transfer in each iteration step is: input of a new character b_j into the lower western IP of each $k \times N$ subarray for query lengths ≤ 2048 (case 1. in Section 4) and input of a new b_j and a previously computed cell of H and F and output of an H-cell and F-cell from the upper western IP for query lengths > 2048 (case 2. in Section 4). Thus, the data transfer time is totally dominated by above computing time of 68 instructions per iteration step.

Table 2: Scan times (in seconds) of TrEMBL 14 for various length of the query sequence on Systola 1024, a PC cluster with 16 Systola 1024, and a Pentium III 600. The speed up compared to the Pentium III is also reported

Query sequence length	256	512	1024	2048	4096
Systola 1024 (*speed up*)	294 (*5*)	577 (6)	1137 (*6*)	2241 (*6*)	4611 (*6*)
Cluster of Systolas (*speed up*)	20 (*81*)	38 (*86*)	73 (*91*)	142 (*94*)	290 (*92*)

Pentium III 600 MHz	1615	3286	6611	13343	26690

Table 2 reports times for scanning the TrEMBL protein databank (release 14, which contains 351'834 sequences and 100'069'442 amino acids) for query sequences of various lengths with the SW algorithm. The first two rows of the table give the execution times for Systola 1024 and the cluster with 16 boards compared to a sequential C-program on a Pentium III 600. As the times show, the parallel implementations scale almost linearly with the sequence length. Because of the used static split strategy the cluster times scale also almost linearly with the number of PCs. A single Systola 1024 board is 5-6 times faster than a Pentium III 600. However, a board redesign based on technology used for processors such as the Pentium III (Systola has been built in 1994 [8]) would make this factor significantly higher.

Fig. 6 shows time measurements of sequence comparison with the SW algorithms on different parallel machines. The data for the other machines is taken from [3]. Systola 1024 is around two times faster than the much larger 1K-PE MasPar and the cluster of 16 Systolas is around two times faster than a 16K-PE MasPar. The 1-board Kestrel is 4-5 times faster than a Systola board. Kestrel's design [3] is also a programmable fine-grained parallel architecture implemented as a PC add-on board. It reaches the higher performance, because it has been built with 0.5-µm CMOS technology, in comparison to 1.0-µm for Systola 1024. Extrapolating to this technology both approaches should perform equally. However, the difference between both architectures is that Kestrel is purely a linear array, while Systola is a mesh. This makes the Systola 1024 a more flexible design, suitable for a wider range of applications, see e.g. [13-16].

6 Conclusions and Future Work

In this paper we have demonstrated that hybrid computing is very suitable for scanning biosequence databases. By combining the fine-grained ISA parallelism with a coarse-grained distribution within a PC-cluster, our hybrid architecture achieves supercomputer performance at low cost. We have presented the design of an ISA algorithm that leads to a high-speed implementation on Systola 1024 exploiting the fine-grained parallelism.

The exponentially growth of genomic databases demands even more powerful parallel solutions in the future. Because comparison and alignment algorithms that are favoured by biologists are not fixed, programmable parallel solutions are required to speed up these tasks. As an alternative to special-purpose systems, hard-to-program reconfigurable systems, and expensive supercomputers, we advocate the use of specialised yet programmable hardware whose development is tuned to system speed.

Our future work in hybrid computing will include identifying more applications that profit from this type of processing power, like scientific computing and multimedia video processing. The results of this study will influence our design decision to build a next-generation Systola board consisting of one large 128×128 ISA or of a cluster of 16 32×32 ISAs.

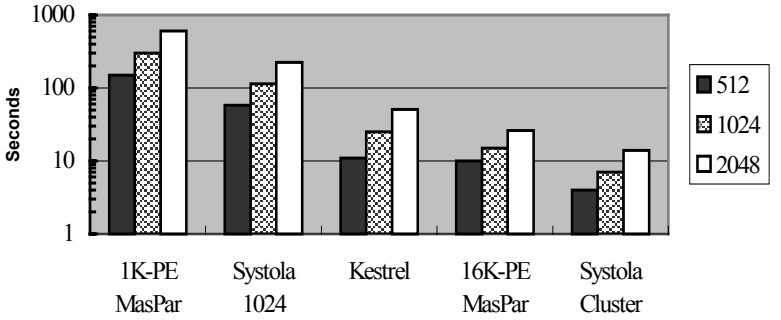

Fig. 6: Time comparison for a 10Mbase search with the SW algorithm on different parallel machines for different query lengths. The values for 1K-PE MasPar, Kestrel, and 16K-PE MasPar are taken from [3], while the values for Systola are based on the TrEMBL 14 scanning times (see Table 2) divided by a normalisation factor of 10

References

1. Altschul, S. F., Gish, W., Miller, W., Myers, E. W., Lipman, D. J.: Basic local alignment search tool, J. Mol. Biol., 215 (1990) 403-410.
2. Borah, M., Bajwa, R. S., Hannenhalli, S., Irwin, M. J.: A SIMD solution to the sequence comparison problem on the MGAP, in Proc. ASAP'94, IEEE CS (1994) 144-160.
3. Dahle, D., et al..: The UCSC Kestrel general purpose parallel processor, Proc. Int. Conf. Parallel and Distributed Processing Techniques and Applications, (1999) 1243-1249.
4. Guerdoux-Jamet, P., Lavenier, D.: SAMBA: hardware accelerator for biological sequence comparison, CABIOS 12 (6) (1997) 609-615.
5. Hoang, D. T.: Searching genetic databases on Splash 2, in Proc. IEEE Workshop on FPGAs for Custom Computing Machines, IEEE CS, (1993) 185-191.
6. Hughey, R.: Parallel Hardware for Sequence Comparison and Alignment, CABIOS 12 (6) (1996) 473-479.
7. Lang, H.-W.: The Instruction Systolic Array, a parallel architecture for VLSI, Integration, the VLSI Journal 4 (1986) 65-74.
8. Lang, H.-W., Maaß, R., Schimmler, M.: The Instruction Systolic Array - Implementation of a Low-Cost Parallel Architecture as Add-On Board for Personal Computers, Proc. HPCN'94, LNCS 797, Springer (1994) 487-488.
9. Lavenier, D., Pacherie, J.-L.: Parallel Processing for Scanning Genomic Data-Bases, Proc. PARCO'97, Elseiver (1998) 81-88.
10. Lopresti, D. P.: P-NAC: A systolic array for comparing nucleic acid sequences, Computer 20 (7) (1987) 98-99.
11. Pearson, W. R.: Comparison of methods for searching protein sequence databases, Protein Science 4 (6) (1995) 1145-1160.

12. Singh, R. K. et al.: BIOSCAN: a network sharable computational resource for searching biosequence databases, CABIOS, 12 (3) (1996) 191-196.
13. Schimmler, M., Lang, H.-W.: The Instruction Systolic Array in Image Processing Applications, Proc. Europto 96, SPIE 2784 (1996) 136-144.
14. Schmidt, B., Schimmler, M.: A Parallel Accelerator Architecture for Multimedia Video Compression, Proc. EuroPar'99, LNCS 1685, Springer (1999) 950-959.
15. Schmidt, B., Schimmler, M., Schröder, H.: Long Operand Arithmetic on Instruction Systolic Computer Architectures and Its Application to RSA cryptography, Proc. Euro-Par'98, LNCS 1470, Springer (1998) 916-922.
16. Schmidt, B.: Design of a Parallel Accelerator for Volume Rendering. In Proc. Euro-Par'2000, LNCS 1900, Springer (2000) 1095-1104.
17. Smith, T. F., Waterman, M.S.: Identification of common molecular subsequences, J. Mol. Biol. 147 (1981) 195-197.