Massively Parallel Solutions for Molecular Sequence Analysis

Bertil Schmidt, Heiko Schröder
Nanyang Technological University
School of Computer Engineering
Singapore 639798
asbschmidt@ntu.edu.sg

Manfred Schimmler
Technische Universität Braunschweig
Institut für Datentechnik und Kommunikationsnetze
Hans-Sommer-Str. 66, 38106 Braunschweig, Germany
schimmler@ida.ing.tu-bs.de

Abstract

In this paper we present new approaches to high performance protein database scanning on two novel massively parallel architectures to gain supercomputer power at low cost. The first architecture is built around a Beowulf PC-cluster linked by a high-speed network and fine-grained parallel Systola 1024 processor boards connected to each node. The second architecture is the Fuzion 150, a new parallel computer with a linear SIMD array of 1536 processing elements on a single chip. We present the design of a database scanning application based on the Smith-Waterman algorithm in order to derive efficient mappings onto these architectures. The implementations lead to significant runtime savings for large-scale database scanning. This result shows that both architectures provide high-throughput sequence similarity analysis solutions at a good price/performance ratio.

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 [2]. The main drawback of this solution is that the more time efficient the heuristics, the worse is the quality of the results [13].

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 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 [5, 17], while more expensive supercomputers and networks of workstations are suitable architectures for the second [10]. This paper presents two solutions to high performance database scanning on two new architectures: a hybrid parallel computer and the Fuzion 150.

Hybrid computing denotes 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-clusters as in the Beowulf approach is currently one of the most efficient and simple ways to gain supercomputer power for a reasonable price. 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.

Our second solution is based on the Fuzion 150, a parallel computer consisting of a single-chip SIMD array of 1536 processing elements (PEs). Its architecture has been designed to accelerate large-scale visualisation and graphics. We will show that this approach is also beneficial for high performance computational biology.

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 parallel algorithm and its mapping onto this parallel architecture are explained in Section 4. Section 5 introduces the Fuzion 150 and Section 6 discusses the corresponding application mapping. The performance of both approaches is evaluated and compared to previous implementations in Section 7. Section 8 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 [18] 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 $H(i,j)$ for $1 \leq i \leq l1, 1 \leq j \leq l2$ is given by the following recurrences:

$$\begin{aligned} H(i,j) &= \max\{0, E(i,j), F(i,j), \\ &\quad H(i-1,j-1) + Sbt(S1_i, S2_j)\} \\ E(i,j) &= \max\{H(i,j-1) - \alpha, E(i,j-1) - \beta\} \\ F(i,j) &= \max\{H(i-1,j) - \alpha, F(i-1,j) - \beta\} \end{aligned}$$

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 $0 \leq i \leq l1, 0 \leq j \leq l2$. 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.

The dynamic programming calculation can be mapped efficiently 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.

A number of parallel architectures have been developed for sequence analysis. In addition to architectures specifically designed for sequence analysis, existing pro-

	–	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

Figure 1. Example of the SW algorithm to compute the local alignment between two DNA sequences ATCTCGTATGATG **and** GTC-TATCAC**. The matrix $H(i,j)$ is shown for the computation with gap costs $\alpha = 1$ and $\beta = 1$, 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**.**

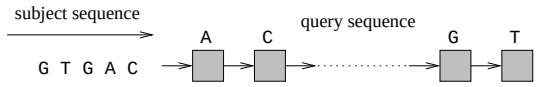

Figure 2. Sequence comparison on a linear processor array: the query sequence is loaded into the processor array (one character per PE) and a subject sequence flows from left to right through the array. During each step, one elementary matrix computation is performed in each PE.

grammable 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, e.g. P-NAC, SAMBA, Bioscan [11, 5, 17]. Reconfigurable systems are based on programmable logic such as field-programmable gate arrays (FPGAs), e.g. Spalsh-2, Biocellerator [6, 7], or custom-designed arrays, e.g. MGAP [3]. 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 first approach is based on instruction systolic arrays (ISAs). ISAs combine the speed and simplicity of systolic arrays with flexible programmability [8], 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, volume visualisation and cryptography [14, 15, 16]. The second approach is based on the SIMD concept. SIMD architectures achieve a high performance cost ratio and can at the same time used for a wide range of applications. Cost and ease of programming fall between the other two classes.

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 (Pentium II, 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 [9] into each PC. Systola 1024 contains an instruction systolic array *ISA* of size 32×32. The ISA [8] is a mesh-connected processor grid, where the processors are controlled by three streams of control information: instructions, row selectors, and column selectors (see Fig. 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. This guarantees that within each diagonal of the array the same instruction is active during each clock cycle. In clock cycle $k+1$ processor $(i+1, j)$ and $(i, j+1)$ execute the instruction that has been executed by processor (i, j) in clock cycle k. The selectors also move systolically through the array: the row selectors horizontally from left to right, column selectors

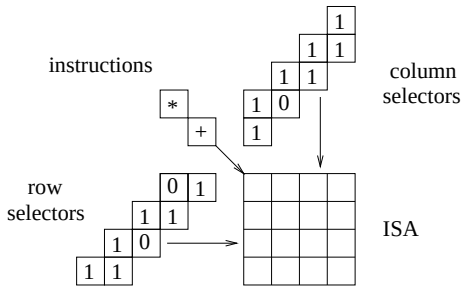

Figure 4. Control flow in an ISA.

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. 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. The ISA combines the advantages of fine-grained SIMD machines with the capability of efficiently performing special communication operations, so called *aggregate functions*. These are associative and communtative functions to which each processor provides an argument value. Examples for aggregate functions are broadcast and ring-shift operations along the rows or columns of the processor array. These are the key operations within the algorithm presented in the next Section.

In order to exploit the computation capabilities of the ISA, a cascaded memory concept is implemented on Systola 1024 (see Fig. 3 right). 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.

At a clock frequency of $f = 50$ MHz and using a word format of $m = 16$ bits each (bitserial) processor can execute $\frac{f}{m} = \frac{50}{16} \cdot 10^6 = 3.125 \cdot 10^6$ 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.

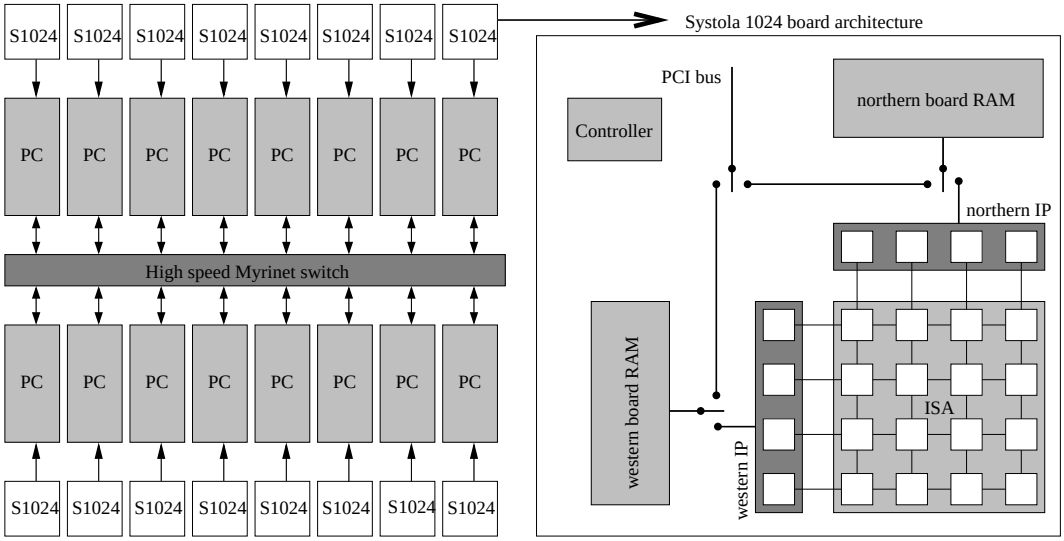

Figure 3. Architecture of our hybrid system: A cluster of 16 PCs with 16 Systola 1024 PCI boards (left). The data paths in Systola 1024 are depicted on the right.

4. Mapping of Sequence Comparison onto the ISA

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 parallelisation 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 shows the data flow in the ISA for aligning the sequences $A = a_0 a_1 \ldots a_{M-1}$ and $B = b_0 b_1 \ldots b_{K-1}$, 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 $n = (N - i) \bmod N - 1$ and B is loaded into the lower western IP. After that the row of the substitution table corresponding to the respective

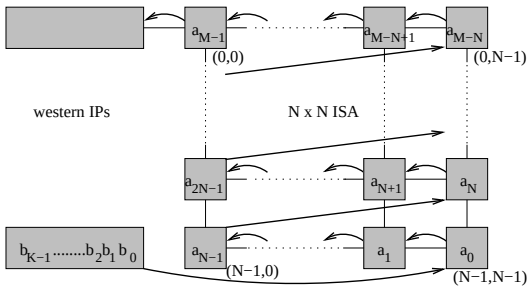

Figure 5. Data flow for aligning two sequences A and B on an $M = N \times N$ ISA: A is loaded into the ISA one character per PE and B is completely shifted through the array in $M + K - 1$ steps. Each character b_j is input from the lower western IP and results are written into the upper western IP.

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.

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.

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. $k \leq N$**:** 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 $\frac{k}{N}$ characters of the sequences to each PE instead of one. However, in this case the memory size has to be sufficient to store $\frac{k}{N}$ 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 $\frac{k}{N} > 2$ it is required to split the sequence comparison into $\frac{k}{2N}$ 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 [10] is more suitable for heterogeneous environments.

5. The Fuzion 150 SIMD architecture

Early SIMD architectures suffered to some extend due to the small amounts of area for each PE, e.g. [3, 4]. The increase in integration on ICs now allows a large SIMD array, with local PE memory and controllers on a single die. The Fuzion 150 system architecture shown in Fig. 6 provides a general-purpose processing solution for many application areas including network processing and large-scale visualisation and graphics [1, 12]. It combines control and data processing on the same silicon, whilst taking account of the very different processing requirements of the control and data planes. The control plane and housekeeping operations are performed on the embedded processing unit (EPU), which is a 32-bit ARC core. Data plane operations utilize the PEs in the Fuzion core.

The processor array is made up of six blocks of PEs. Every block consists of a linear array of 256 8-bit PEs. The PEs at the borders of each block are also connected with the borders of the next blocks. This provides a linear array of 1536 PEs. The PE itself is based around an 8-bit ALU, connected to a 32 Bytes register file. This register file is multi-ported to give neighbor communications, and concurrent access to the PE's own on-chip 2 KBytes DRAM. The PE also has direct access to a linear expression evaluator (LEE) that can perform an operation of the form $Ax+By+C$ on a per cycle basis. Data I/O for the processor array operates on a per block basis. A high performance I/O engine allows data transfer at up to 700 MBytes/s per block via the Fuzion bus. At a clock frequency of $f = 200$ MHz and using a word format of $m = 8$ bits each PE can execute $\frac{f}{m} = 200 \cdot 10^6$ word operations per second. Thus, the Fuzion 150 parallel computer performs up to 300 GOPS.

6. Mapping of Sequence Comparison on the Fuzion 150

Compared to Systola 1024 no fast ringshift operation is available on the Fuzion 150. Hence, we are using a different mapping scheme for SW cell computation. The communication pattern in each iteration step now consists of two steps: a read-left operation in even-numbered PE blocks and

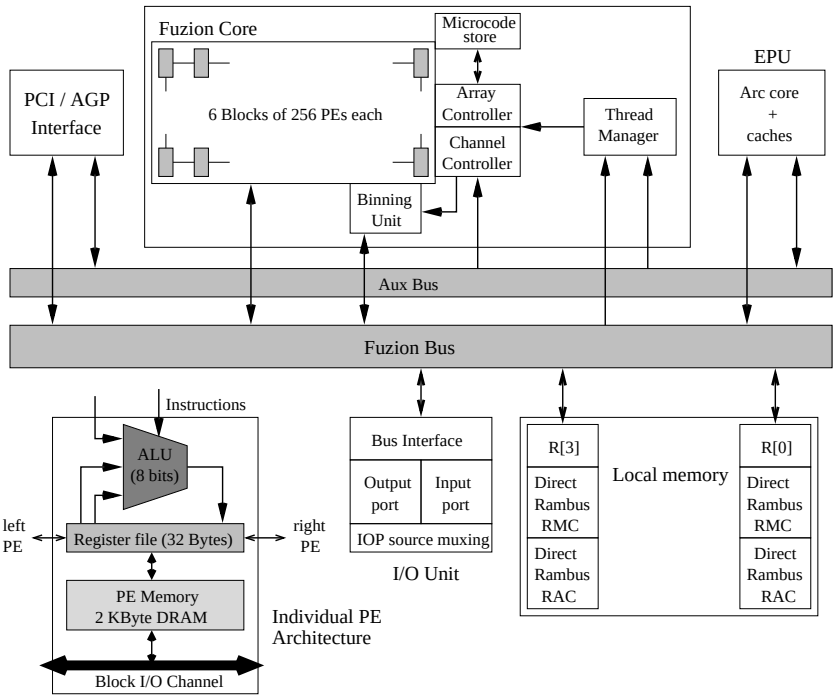

Figure 6. System block diagram of the Fuzion 150 architecture and the architecture of an individual PE.

a read-right operation in odd-numbered PE blocks (see Fig. 7).

For the detailed description we firstly assume the processor array size to be equal to the query sequence length M, i.e. $M = 1536$. Fig. 7 shows the data flow for aligning the sequences $A = a_0 a_1 \dots a_{M-1}$ and $B = b_0 b_1 \dots b_{K-1}$, where A is the query sequence and B is a subject sequence of the database. As a preprocessing step, symbol $a_i, i = 0, \dots, M-1$, is loaded into PE (m, n) (notation for PE m of block n, i.e. $m \in \{0, \dots, 255\}$, $n \in \{0, \dots 5\}$) with $n = i$ div 256 and $m = i \bmod 256$ for even n and $m = 255 - i \bmod 256$ for odd n. B is loaded into the local memory. 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 (Fig. 7). In iteration step $k, 1 \leq k \leq M + K - 1$, the values $H(i, j)$, $E(i, j)$, and $F(i, j)$ for all i, j with $1 \leq i \leq M, 1 \leq j \leq K$ and $k = i + j - 1$ are computed in parallel in the PEs (m, n) with $n = i$ div 256 and $m = i \bmod 256$ for even n and $m = 255 - i \bmod 256$ for odd n. For this calculation PE (m, n) receives the values $H(i-1, j)$, $F(i-1, j)$, and b_j from its neighbor, 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 in the local DRAM.

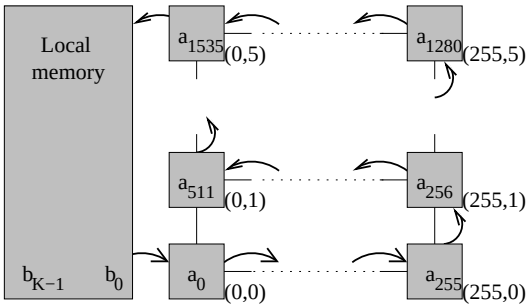

Figure 7. Data flow for aligning two sequences A and B on the Fuzion 150: A is loaded into the processor array one character per PE and B is completely shifted through the array in $M + K - 1$ steps.

So far we have assumed a processor array equal in size of the query sequence length ($M = 1536$). In practice, this rarely happens. Assuming a query sequence length less or larger the array size, our implementation is modified as follows:

1. $k \times M = 1536$**:** In this case we can just replicate the implementation for M PEs on each subarray of size N, i.e. k alignments of the same query sequence with different subject sequences are computed in parallel.

2. $k \times 1536 = M$**:** A possible solution is to split the sequence comparison into k passes. However, this solution requires I/O of intermediate results in each iteration step. This additional data transfer can be avoided by assigning $\frac{k}{M}$ characters of the sequences to each PE instead of one. On the Fuzion 150 it is possible to assign up to 16 characters per PE. Thus, query lengths of up 24576 characters can be processed within a single pass, which is sufficient for molecular applications.

7. 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 and on Fuzion 150, we have given the instruction count to update two H-cells per PE on Systola 1024 and 16 H-cells per PE on Fuzion 150 in Table 1.

Because new H-values are computed for two characters within 68 instructions in each PE, the whole 32×32 processor array can perform 2048 cell updates in the same time. This leads to a performance of $\frac{2048}{68} \times f$ CUPS $= \frac{2048}{68} \times \frac{50}{16} \times 10^6$ CUPS $= 94$ MCUPS. The corresponding calculation for the Fuzion 150 ($16 \times 1536 = 24576$ cells are updated within 1934 instructions) delivers a result of $\frac{24576}{1934} \times f$ CUPS $= \frac{24576}{1934} \times 200 \times 10^6$ CUPS $= 2541$ 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). During the computation of 16 new H-cells in each PE of the Fuzion 150, one new character is input via the Fuzion bus into each subarray that performs a sequence comparison. Thus, the required data transfer time is totally dominated by above computing times per iteration step of 68 instructions and 1934 instructions, respectively.

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 Fuzion 150 is $25 - 28$ times faster than a Systola board. It reaches the higher performance, because it has been built with 0.25μ CMOS technology, in comparison to 1.0μ for Systola 1024. Extrapolating to this technology both approaches should perform equally. However, the difference between both architectures is that Fuzion 150 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. [14, 15, 16].

For the comparison of different parallel machines, we have taken data from Dahle, Grate, Rice and Hughey [4] for a database search of the SW algorithm for different query lengths. The Fuzion 150 is three to four times faster than the much larger 16K-PE MasPar. The 1-board Kestrel [4] is six times slower than a Fuzion 150 chip. Kestrel's design is also a programmable fine-grained SIMD array. It reaches the lower performance, because it has been built with older CMOS technology (0.5μ). SAMBA [5] is a special-purpose systolic array for sequence comparison implemented on two add-on boards, which are around five times slower than the Fuzion.

8. Conclusions and Future Work

In this paper we have demonstrated that fine-grained parallel architectures are suitable solutions for high performance scanning of biosequence databases. We have presented efficient mappings of the SW algorithm on an ISA and a SIMD array that lead to high-speed implementations on Systola 1024 and Fuzion 150. By combining the fine-grained parallelism with a coarse-grained distribution within a Beowulf PC-cluster an even higher performance can be achieved at a good price/performance ratio.

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 consisting of a combination of fine-grained and coarse-grained parallelism, like scientific computing

Table 1. Instruction Count (IC) to update 2 (16) H-cells in one PE of Systola 1024 (Fuzion 150) with the corresponding operations.

Operation in each PE per iteration step	IC Systola	IC Fuzion
Get $H(i-1,j), F(i-1,j), b_j, max_{i-1}$ from neighbour	20	22
Compute $t = \max\{0, H(i-1,j-1) + Sbt(a_i, b_j)\}$	20	576
Compute $F(i,j) = \max\{H(i-1,j) - \alpha, F(i-1,j) - \beta\}$	8	336
Compute $E(i,j) = \max\{H(i,j-1) - \alpha, E(i,j-1) - \beta\}$	8	448
Compute $H(i,j) = \max\{t, F(i,j), E(i,j)\}$	8	368
Compute $max_i = \max\{H(i,j), max_{i-1}\}$	4	184
Sum	68	1934

Table 2. Scan times (in seconds) of TrEMBL 14 for various query sequence lengths on Systola 1024, the PC cluster with 16 Systola 1024, the Fuzion 150, and a Pentium III 1 GHz. The speed up compared to the Pentium III is also reported.

Query sequence length	256	512	1024	2048	4096
Systola 1024 (speed up)	294 (4)	577 (4)	1137 (4)	2241 (4)	4611 (4)
Cluster of Systolas (speed up)	20 (53)	38 (56)	73 (58)	142 (60)	290 (59)
Fuzion 150 (speedup)	12 (88)	22 (97)	42 (102)	82 (105)	162 (106)
Pentium III 1 GHz	1053	2132	4252	8581	17164

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.

References

[1] www.clearspeed.com, 2001.

[2] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[3] M. Borah, R. Bajwa, S. Hannenhalli, and M. Irwin. A simd solution to the sequence comparison problem on the mgap. In *Proc. ASAP'94*, pages 144–160. IEEE CS, 1994.

[4] D. Dahle, L. Grate, E. Rice, and R. Hughey. The ucsc kestrel general purpose parallel processor. In *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, pages 1243–1249, 1999.

[5] P. Guerdoux-Jamet and D. Lavenier. Samba: hardware accelerator for biological sequence comparison. *CABIOS*, 12(7):609–615, 1997.

[6] D. Hoang. Searching genetic databases on splash 2. In *Proc. IEEE Workshop on FPGSs for Custom Computing Machines*, pages 185–191, 1993.

[7] R. Hughey. Parallel hardware for sequence comparison and alignment. *CABIOS*, 11(6):473–479, 1996.

[8] H. Lang. The instruction systolic array, a parallel architecture for vlsi. *Integration, the VLSI Journal*, 4:65–74, 1986.

[9] H. Lang, R. Maaß, and M. Schimmler. The instruction systolic array - implementation of a low-cost parallel architecture as add-on board for personal computers. In *Proc. HPCN'94*, pages 487–488. Springer, 1994.

[10] D. Lavenier and J. Pacherie. Parallel processing for scanning genomic data-bases. In *Proc. PARCO'97*, pages 81–88. Elseiver, 1998.

[11] D. Lopresti. P-nac: A systolic array for comparing nucleic acid sequences. *Computer*, 20(7):81–88, 1987.

[12] M. Meissner, S. Grimm, W. Strasser, J. Packer, and D. Latimer. Parallel volume rendering on a single-chip simd architecture. In *Proc. IEEE Symposium in Parallel and Large Data Visualization and Graphics*, pages 107–113, 2001.

[13] W. Pearson. Comparison of methods for searching protein sequence databases. *Protein Science*, 4(6):1145–1160, 1995.

[14] M. Schimmler and H. Lang. The instruction systolic array in image processing applications. In *Proc. Europto 96*, volume 2784 of *SPIE*, pages 136–144, 1996.

[15] B. Schmidt. Design of a parallel accelerator for volume rendering. In *Proc. Euro-Par 2000*, pages 1095–1104. Springer, 2000.

[16] B. Schmidt and M. Schimmler. A parallel accelerator architecture for multimedia video compression. In *Proc. Euro-Par 99*, pages 950–959. Springer, 1999.

[17] R. Singh. Bioscan: a network sharable computational resource for searching biosequence databases. *CABIOS*, 12(3):191–196, 1996.

[18] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.