Multiple Sequence Alignment Using Reconfigurable Computing*

Carlos R. Erig Lima, Heitor S. Lopes, Maiko R. Moroz, and Ramon M. Menezes

Bioinformatics Laboratory, Federal University of Technology Paraná (UTFPR),
Av. 7 de setembro, 3165 80230-901, Curitiba (PR), Brazil
erig@utfpr.edu.br, hslopes@pesquisador.cnpq.br

Abstract. The alignment of multiple protein (or DNA) sequences is a current problem in Bioinformatics. ClustalW is the most popular heuristic algorithm for multiple sequence alignment. Pairwise alignment has exponential complexity and it is the most time-consuming part of ClustalW. This part of ClustalW was implemented using a reconfigurable logic hardware solution: Hardalign. The system was evaluated using data sets of different dimensionality, and compared with a pure software version running in a embedded processor, as well as running in a desktop computer. Results indicate that such implementation is capable of accelerating significantly part of the algorithm, and this is especially important for processing large protein data sets.

1 Introduction

Sequence alignment is a basic procedure frequently used in Bioinformatics. In the case of proteins, the alignment is done by a systematic comparison of the amino acids throughout the whole extension of the sequences. The main objective is to compute a score that indicates the similarity between proteins. Alignment can be done with a pair of sequences (pairwise alignment) or with several ones (multiple sequence alignment). In general, sequence alignment is the most important method for discovering and representing similarities between sequences.

From the computational point of view, sequence alignment, especially for multiple sequences, is a difficult task. In recent literature, many computational algorithms were proposed for this purpose. The main differences between them is the overall quality of the alignment and the computational effort required. Therefore, many heuristic methods have been proposed for multiple sequence alignment [3,6], since the exact algorithm is computationally unfeasible.

Recently, we have witnessed a pronounced growth of the hardware and software technologies for embedded systems, with many technological options arising every year. In particular, applications based on reconfigurable computing for sequence alignment can found in recent works [1,5].

The objective of this work is to implement a multiple sequence alignment algorithm in reconfigurable hardware, taking advantage of the possible parallelism of operations.

* This work was partially supported by the Brazilian National Research Council – CNPq, under research grants no. 305720/2004-0 and 506479/2004-8.

P.C. Diniz et al. (Eds.): ARC 2007, LNCS 4419, pp. 379–384, 2007.

2 The ClustalW Algorithm

This work is based on ClustalW [6], a progressive alignment algorithm for multiple sequencesThe algorithm is divided into three basic steps, as follows:

The first step is the pairwise alignment, where all pairs of sequences are aligned using a dynamic programming (DP) algorithm for global alignment. It builds a $m \times n$ matrix (m and n are the length of the two sequences) and computes a score walking backwards in the matrix, looking for the minimal cost associated with substitutions, insertions and deletions. This step is repeated iteratively for all $n(n-1)/2$ pairs of sequences to be aligned, thus obtaining a distance matrix. The computed score is meant as the similarity degree between two sequences and is computed as evolutionary distances using the Kimura [2] model.

The second step is the computation of an unrouted guide tree (a kind of phylogenetic tree) based on the constructed distance matrix. This tree shows the evolution of the sequences, grouped in pairs of minimum distances (scores) using a neighbor-joining clustering algorithm. This tree is a profile of the order in which sequences should be aligned for maximal efficiency of the next step.

Finally, the final step is a progressive alignment of the sequences, traversing the distance tree in order of decreasing similarity: sequence-sequence, sequence-profile, and profile-profile alignment. This step is based on an improved DP algorithm [6]. Although it is very time-consuming, the final alignment is not the optimal alignment for the sequences under study.

3 Pairwise Alignment with Hardalign

Hardalign [4] is a dedicated processing system, working as a peripheral of the NIOS II Altera embedded processor, interconnected via the Avalon bus Hardalign contains an arrangement of N Matrix Line Processor Units (MLPUs - see below) and all logic for driving the arrangement. The critical part of the process that requires computational power is the computation of the DP matrix and, for this reason, it is run in parallel by the MLPUs. The software running in the NIOS II processor performs the first two steps and writes data to internal registers of Hardalign. Then, the driving logic is started and NIOS II reads sequentially the results. A SDRAM memory is used for storing vectors generated by the MLPUs. Next, the progressive alignment routine is started by software running in NIOS II. This algorithm uses the vectors previously generated and finds a path from the last cell of the matrix towards the first one.

Each MLPU computes one cell of the DP matrix, in a single clock cycle, and it is responsible for computing a line of the matrix. Once completed, a new line is started until the matrix is fully done. Since N cells are computed by clock cycle, the whole matrix is concluded in $L.C/N$ clock cycles, where L is the number of lines and C the number of columns of the matrix. The substitution matrix is encoded as a combinatorial circuit: every possible combination of the 20 amino acids gives an evolutionary distance value as result.

The MLPU entity can be replicated to compute simultaneously several lines in parallel. All the data for an entity can be read from other entities.The line amino

acids are read in the beginning of the process, and the pipeline used for the first line of alignment should receive the column amino acids sequentially. Fig. 1(left) shows the MLPU entity in details, where an amino acid requested by a line is always requested by the pipeline that was above it in the previous clock cycle. The previous computed value is registered to be reused for the computation of the subsequent cells. The values computed in the two subsequent clock cycles are registered. They are necessary for the computation of the line below. The gap penalty is common to all of the entities and registered outside of the MLPU.

The pipeline entity is arranged to compute several lines in parallel. In Fig. 1(right) an arrangement with 3 MLPUs is shown, allowing the simultaneous computation to 3 lines of the DP matrix. The inputs of this entity are the gap penalty, the line amino acids and the column amino acids, in a sequential way.

Each MLPU entity has four outputs: two cell values, an amino acid value and a vector. Only the vector is necessary for the backtracking procedure later performed. The other outputs are important for the computation of the subsequent cells. In fact, only the vectors are defined as outputs, and the other signals are internal values of the pipeline. These vectors are defined like an output bus (vector bus) of $2 \times N$ bits, where N is the pipelines number. Initially, just the first MLPU contains valid values in their internal registers, and only some bits

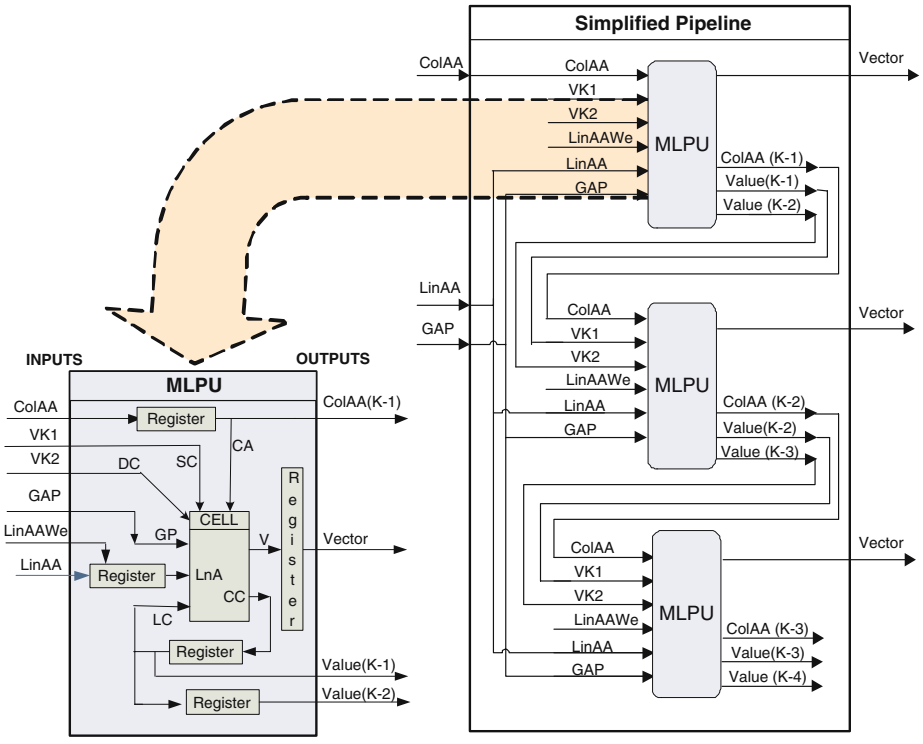

Fig. 1. Left: Block diagram of the MLPU. Right: Block diagram of the Pipeline.

of the vector bus are valid. After some cycles the own MLPU apply valid values to the subsequent MLPU.

4 Computational Experiments and Results

The hardware was synthesized in an Altera Stratix II EP2S60F672C5ES device (*http://www.altera.com*) running at 40 MHz. We used the SOPC builder (System on a Programmable Chip Builder) software to integrate the several modules with the NIOS II core. The physical synthesis and simulations was done using Quartus II environment.

The first experiment is a performance analysis for different sizes of DP matrix, comparing an implementation in PC and the hardware (with 8 MLPUs). The C language version running in the PC version runs the same algorithm implemented in hardware. We used a PC with Athlon XP 1600+ processor, 512Mb de RAM DDR 266 and Windows XP operational system. Three experiments were done, as follows.

Table 1 shows the computation time for pairs of 20- to 2000-amino acids-long sequences. The resolution of the PC timer is 1 millisecond, precluding to measure the processing time for the 20 x 20 matrix. In this table, we observed a 1:10 ratio, approximately. A further improvement of Hardalign is the use of a DMA (Direct Memory Access) controller module to improve the transfer rate between the SRAM and the Hardalign Pipeline Data was obtained by simulation and the comparison is shown in the same Table. An improvement of about 28 times in performance can be observed comparing the approaches with and without DMA.

The second experiment analyzes the performances of multiple sequence alignment with and without the use of Hardalign. For this experiment, we fixed the number of proteins to 5 and analyzed the performance for sequences of several amino acids lengths, as shown in Table 2. For each comparison (Software approach versus Hardware approach), three columns are presented: the time necessary for the pairwise alignment of all pairs of sequences (Pairwise), not including data transfer time; the time necessary for the whole progressive alignment using the guide tree (Progressive); and the total time for the multiple alignment (Total).

Regarding the time for pairwise alignment only, an improvement of of about 110 times in performance can be observed comparing the hardware approach and the software approach. On the other hand, the total processing time did not presented a significant improvement. In this case, besides the speed-up observed in pairwise computation, the advantage obtained is surpassed by the larger time demanded in the progressive alignment computation step. Another feature observed was the advantage of the hardware approach for larger amino acids sequences. In this case, the efficiency of Hardalign can be better explored.

In the third experiment, performance was tested for several sets of proteins to be aligned using the same protein size. Table 3 shows the computation time for sets of 10- to 100 proteins using a 200-amino acids-long sequence. The meaning of the columns in both comparisons are the same as in Table 2.

The demand for LUTs (Look Up Table) and internal registers grows up linearly as function of the number of MLPUs. For 8 MLPUs, 2189 LUTs and 589 registers

Table 1. Performance comparison of pairwise alignment (Hardalign versus PC)

Matrix size (lines x columns)	Hardalign (ms)	Hardalign with DMA (ms)	Athlon XP (ms)
20x20	0.074	0.001	-
100x100	0.949	0.031	10
200x200	3.548	0.125	30
500x500	22.123	0.781	200
1000x1000	87.618	3.125	831
2000x2000	350.207	12.500	3465

Table 2. Performance comparison between hardware and software for pairwise alignment (different lengths, same set). Processing time is given in seconds.

	NIOS II - Software			NIOS II - Hardalign		
#Amino Acids	Pairwise	Progressive	Total	Pairwise	Progressive	Total
30	0.03	0.56	0.63	0.01	0.56	0.61
90	0.21	3.46	3.73	0.02	3.41	3.50
200	1.04	14.67	15.85	0.05	14.89	15.08
300	2.98	30.80	34.26	0.09	31.24	31.80
901	35.20	263.94	301.31	0.46	269.74	272.29
1703	155.58	1060.01	1224.00	1.38	1009.75	1018.89

Table 3. Performance comparison between hardware and software for pairwise alignment (different sets, same lengths). Processing time is given in seconds.

	NIOS II - Software			NIOS II - Hardalign		
#Amino acids	Pairwise	Progressive	Total	Pairwise	Progressive	Total
10	13.467	35.710	50.006	0.308	35.297	36.307
20	30.121	84.643	116.650	2.097	74.270	78.368
50	215.595	228.133	455.059	27.530	194.432	235.377
100	776.435	494.265	1323.333	205.260	397.789	668.648

are used, taking into account only the pipeline logic, excluding the Avalon bus, NIOS II core, additional memories and pipeline drivers. The maximum frequency operation of pipeline is not affected by the number of MLPUs used.

5 Conclusions and Future Work

Multiple sequence alignment is an important problem in Bionformatics, but still open issue when dealing with large sequences. This work aims at exploring an alternative solution to this problem, using reconfigurable computing to substitute a computationally-intensive part of the ClustaW algorithm. The methodology for parallelizing a pairwise sequence alignment algorithm contributes to identify and circumvent the bottlenecks of a conventional software implementation.

The performance analysis reveals that the improvement by using the hardware approach achieves around 1:10 ratio of speed-up, compared with the conventional PC software processing. Using DMA, the ratio achieves around 1:280.

The performance of NIOS II software processing achieves a speed-up ratio of around 1:140 ratio, when compared with the NIOS II with Hardalign. This huge difference is due to features such as operational system, clock frequency and implementation languages used. Particularly, the comparison between different hardware platforms such as PC and FPGA-based kits is not completely fair, but necessary to emphasize significant performance differences.

Besides the significant minimization of computation time of the DP matrix, a relative low minimization in total computation time of ClustaW algorithm was observed: only 100% speed-up. There are some reasons for this shallow result: important time is spent to manage and transfer data, particularly in the progressive part of the algorithm and the fact that no parallel resources in the progressive part of the algorithm were used.

We observed that the growth in the number of proteins to be aligned decreases the speed-up of the system, and the growth of the length of the sequences increases speed-up. Therefore, the proposed system is more efficient for alignments with few sequences of large number of amino acids. Fortunately, this is the case of many real-world applications.

We believe that the proposed solution is an important contribution to both reconfigurable systems technology and Bioinformatics. Further research will focus on devising new parallelization levels for the multiple sequence alignment problem. ClustalW was developed to run using sequential processor. On the other hand, reconfigurable computing is a paradigm that strongly suggests the conception of new algorithms that explores multiple levels of parallelization.

References

1. Jacobi, R.P., Ayala-Rincon, M., Carvalho, L.G.A., et al.: Reconfigurable systems for sequence alignment and for general dynamic programming, *Genet Mol Res* **4** (2005) 543–552.
2. Kimura, M.: *The Neutral Theory of Molecular Evolution.* Cambridge University Press, New York (1983).
3. Lopes, H.S., Moritz, G.L.: A graph-based genetic algorithm for the multiple sequence alignment problem, *Lect Notes Artif Intel* **4029** (2006) 420–429.
4. Moritz, G.L., Jory, C., Lopes, H.S., Erig Lima, C.R.: Implementation of a parallel algorithm for pairwise alignment using reconfigurable computing. *Proc. IEEE Int. Conf. on Reconfigurable Computing and FPGAs*, (2006) pp. 99-105.
5. Oliver, T., Schmidt, B., Nathan, D., et al.: Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW. *Bioinformatics* **21** (2005) 3431–3432.
6. Thompson, J.D., Higgins, D.G., et al.: CLUSTALW: improving the sensitivity of progres-sive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice, *Nucleic Acids Res* **22** (1994) 4673–4680.