C-Based Design Methodology for FPGA Implementation of ClustalW MSA

Yan Lin Aung[1], Douglas L. Maskell[1], Timothy F. Oliver[1] Bertil Schmidt[2], and William Bong[3]

[1] School of Computer Engineering, NTU, Singapore
{yanl0003,asdouglas}@ntu.edu.sg, tim.oliver@computer.org
[2] Division of Engineering, Science and Technology, UNSW Asia, Singapore
Bertil.Schmidt@unswasia.edu.sg
[3] Network Storage Technology Division, A*STAR Data Storage Institute, Singapore
William_BONG@dsi.a-star.edu.sg

Abstract. Systolisation of the pairwise distance computation algorithm and mapping into field programmable gate arrays (FPGA) have proven to give superior performance at a lower cost, compared to the same algorithm running on a cluster of workstations. The primary design methodology for this approach is based on the hardware description languages such as VHDL and Verilog HDL. An alternative design methodology, however, is the use of a high level language such as C to describe the algorithms and generate equivalent hardware descriptions for implementation in FPGA so as to reduce time to market. This paper describes the design and implementation of the ClustalW first stage multiple sequence alignment based on the Smith-Waterman algorithm on a low cost FPGA development platform using a C language development tool suite. Performance evaluation results show that comparable performance could be achieved to that of Pentium 4 systems and other HDL-based solutions using even the smallest commercially available FPGA device with this design methodology.

Keywords: multiple sequence alignment, ClustalW, FPGA, sequence analysis.

1 Introduction

Multiple sequence alignment (MSA) is a generalized pairwise sequence alignment to include more than two sequences of protein or nucleic acid. The main purpose of MSA is to infer homology between sequences. But there are many other applications: finding diagnostic patterns, characterization of protein families, detection of similarity between new sequences and well-known families of sequences, and evolutionary analysis. As the time and space complexities for MSA are in the order of the product of the lengths of the sequences, many heuristic alignment methods have been developed. Among them, the progressive alignment method is a widely used heuristic. One popular MSA program, ClustalW, makes use of such a method and consists of three stages: distance matrix, guided tree

J.C. Rajapakse, B. Schmidt, and G. Volkert (Eds.): PRIB 2007, LNBI 4774, pp. 11–18, 2007.

and progressive alignment along the tree. The first stage computes and tabulates the distance value between every pair of sequences using pairwise sequence alignment. Then, a phylogenetic or guided tree is constructed using the distance matrix obtained in the first stage. Finally, sequences are progressively aligned according to the order specified by the guided tree to produce the final MSA. Unfortunately, the progressive alignment method still requires long periods of time to compute the MSA. Profiling of the ClustalW program on a single processor showed that almost 96% of the time is spent in the first stage [1].

1.1 Motivation

Many parallel processing approaches have been proposed and developed to speed-up the time consuming MSA first stage. Ebedes et al. have implemented parallel version of the MSA first stage on a small cluster of six workstations using the Message Passing Interface and reported that linear speed-up and almost 100% efficiency could be achieved [1].

Alternatively, it is also possible and very promising to map the MSA first stage into FPGA since the underlying architecture of the FPGA is well-suited for parallel processing. This approach is concerned with the systolisation of a pairwise distance computation algorithm and mapping it into FPGA, and it has proven to give superior performance at lower cost compared to the same algorithm running on a cluster of workstations [2], [3], [4]. The dominant design methodology for this approach has been based on hardware description languages (HDL), such as VHDL and Verilog HDL.

On the other hand, the capacity of FPGAs, like other semiconductor chips, increases in accordance with Moore's law as do the design complexities for these chips. The support provided by electronic design automation software tool vendors to cope with the increasing design complexity is lagging behind Moore's law, with conventional HDL-based designs often becoming bottlenecks in the design cycle. This has created a productivity gap between design complexity and design capacity [5]. A number of extensions to existing HDLs and software tools based on high level languages have emerged in an attempt to bridge this productivity gap. One such tool is CoDeveloper from Impulse Accelerated Technologies. CoDeveloper is a C language development tool suite for FPGAs which allows designers to use standard ANSI C to express highly parallel applications and algorithms.

With the above as the basis for our motivation, we have designed and implemented the ClustalW first stage multiple sequence alignment based on the Smith-Waterman algorithm on a low cost FPGA development platform using the C language development tool suite. We explain the pairwise sequence distance computation of the ClustalW MSA in detail and describe the recurrence equations, which were used to efficiently map the algorithm into FPGA hardware, in the next section.

2 ClustalW MSA First Stage: Distance Matrix

The ClustalW program makes use of the following definition to compute the distance between two sequences.

Definition 1. *Given a set of n sequences $S = S_1, \ldots, S_n$. For two sequences $S_i, S_j \in S$, pairwise sequence distance $d(S_i, S_j)$ is defined as follows:*

$$d(S_i, S_j) = 1 - \frac{nid(S_i, S_j)}{min\{l_i, l_j\}} \tag{1}$$

where $nid(S_i, S_j)$ denotes the number of exact matches in the optimal local alignment of S_i and S_j (with respect to the given scoring system, i.e the substitution matrix sbt and gap penalty parameters α, β for affine gap penalties or just α for linear gap penalties) and l_i (l_j) denotes the length of S_i(S_j).

The Smith-Waterman algorithm can be used to compute the optimal local alignment of two sequences [6]. The algorithm compares two sequences by computing a distance that represents the minimal cost of transforming one segment into another using two elementary operations: match/mutation and insertion/deletion (also called a gap penalty). For two sequences, S_1 and S_2 with length l_1 and l_2, the Smith-Waterman algorithm computes the similarity $H_A(i, j)$ of two sequences ending at position i and j in order to identify common subsequences. The computation of $H_A(i, j)$, for $1 \leq i \leq l_1$, $1 \leq j \leq l_2$, is given by the following recurrences:

$$H_A(i,j) = \max\{0, E(i,j), F(i,j), H_A(i-1, j-1) + sbt(S_1[i], S_2[j])\} \tag{2}$$

$$E(i,j) = \max\{H_A(i, j-1) - \alpha, E(i, j-1) - \beta\} \tag{3}$$

$$F(i,j) = \max\{H_A(i-1, j) - \alpha, F(i-1, j) - \beta\} \tag{4}$$

where *sbt* refers to the character substitution table. Initial values are: $H_A(i, 0) = E(i, 0) = H_A(0, j) = F(0, j) = 0$ for $0 \leq i \leq l_1, 0 \leq j \leq l_2$. 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. This type of gap cost is known as affine gap penalty. Some applications also use a linear gap penalty, i.e. $\alpha = \beta$. For linear gap penalties the above recurrence relations can be simplified to:

$$H_L(i,j) = \max\{0, H_L(i, j-1) - \alpha, H_L(i-1, j) - \alpha, H_L(i-1, j-1) + sbt(S_1[i], S_2[j])\} \quad . \tag{5}$$

Each position of the matrix H_A (H_L) is the similarity value. The two segments of S_1 and S_2 producing this value can be determined by a traceback procedure. The value $nid(S_1, S_2)$ of the two sequences can then be computed by counting the number of exact character matches during the traceback procedure of the Smith-Waterman algorithm. Unfortunately, this procedure is not suitable for hardware implementation. Hence, we present a new recurrence relation for the *nid*-value computation [7] that is suitable for hardware implementation in this section. We first explain the idea for the linear gap penalty and then generalize it for affine gap penalties.

Definition 2. *Given two sequences S_1 and S_2 with length l_1 and l_2, linear gap penalty α and a substitution table sbt, the matrix $N_L(i, j)$ ($1 \leq i \leq l_1$, $1 \leq j \leq l_2$) is recursively defined as follows:*

$$N_L(i,j) = \begin{cases} 0 & \text{if } H_L(i,j) = 0 \\ N_L(i-1,j-1) + m(i,j) & \text{if } H_L(i,j) = H_L(i-1,j-1) \\ & \quad + sbt(S_1[i], S_2[j]) \\ N_L(i,j-1) & \text{if } H_L(i,j) = H(i,j-1) - \alpha \\ N_L(i-1,j) & \text{if } H_L(i,j) = H(i-1,j) - \alpha \end{cases} \quad (6)$$

where

$$m(i,j) = \begin{cases} 1 \text{ if } S_1[i] = S_2[j] \\ 0 \text{ otherwise } . \end{cases}$$

Theorem 1. *For the local alignment of two sequences S_1 and S_2 with linear gap penalty α and substitution matrix sbt,*

$$nid(S_1, S_2) = N_L(i_{max}, j_{max})$$

where (i_{max}, j_{max}) denotes the coordinates of the maximum value in the corresponding matrix H_L.

Proof. Consider the optimal alignment of all pairs of suffixes of the first i characters of $S_1(S_1[1 \ldots i])$ and the first j characters of $S_2(S_2[1 \ldots j])$. This alignment is called the optimal i, j suffix alignment (of S_1 and S_2). It can be found by computing a traceback in the matrix H_L starting from cell (i, j). We now show that for a given pair of indices i, j $(1 \leq i \leq l_1, 1 \leq j \leq l_2)$, $N_L(i, j)$ is equal to the number of exact matches in optimal i, j suffix alignment. The claim then follows from the fact that the optimal i_{max}, j_{max} suffix alignment is equal to the optimal local alignment.

Case 1: $H_L(i, j) = 0$. The corresponding alignment is empty and $N_L(i, j) = 0$.
Case 2: $H_L(i, j) = H_L(i-1, j-1) + sbt(S_1[i], S_2[j])$. The alignment ends with $S_1[i]$ aligned to $S_2[j]$, which contributes $m(S_1[i], S_2[j])$ to the number of exact matches. The remaining number is then equal to the number of exact matches found in the optimal $i-1$, $j-1$ suffix alignment. Hence, $N_L(i, j) = N_L(i-1, j-1) + m(S_1[i], S_2[j])$.
Case 3: $H_L(i, j) = H_L(i-1, j) - \alpha$. The alignment ends with $S_1[i]$ aligned to a gap, which contributes zero exact matches. The remaining number is equal to the number found in the optimal $i-1$, j suffix alignment. Hence, $N_L(i, j) = N_L(i-1, j)$.
Case 4: $H_L(i, j) = H_L(i, j-1) - \alpha$. Similar to Case 3 follows $N_L(i, j) = N_L(i, j-1)$.

Because $H_L(i, j)$ must be equal to one of these four cases, the Theorem is proven. □

For affine gap penalties our method is extended as follows.

Definition 3. *Given two sequences S_1 and S_2, affine gap penalties α, β, and substitution table sbt, matrix $N_A(i, j)(1 \leq i \leq l_1, 1 \leq j \leq l_2)$ is recursively defined as follows:*

$$N_A(i,j) = \begin{cases} 0 & \text{if } H_A(i,j) = 0 \\ N_A(i-1,j-1) + m(i,j) & \text{if } H_A(i,j) = H_A(i-1,j-1) \\ & \quad + sbt(S_1[i], S_2[j]) \\ N_E(i,j) & \text{if } H_A(i,j) = E(i,j) \\ N_F(i,j) & \text{if } H_A(i,j) = F(i,j) \end{cases} \quad (7)$$

where

$$m(i,j) = \begin{cases} 1 \text{ if } S_1[i] = S_2[j] \\ 0 \text{ otherwise} \end{cases}$$

$$N_E(i,j) = \begin{cases} 0 & \text{if } j = 1 \\ N_A(i,j-1) & \text{if } E(i,j) = H_A(i,j-1) - \alpha \\ N_E(i,j-1) & \text{if } E(i,j) = E(i,j-1) - \beta \end{cases}$$

$$N_F(i,j) = \begin{cases} 0 & \text{if } i = 1 \\ N_A(i-1,j) & \text{if } F(i,j) = H_A(i-1,j) - \alpha \\ N_F(i-1,j) & \text{if } F(i,j) = F(i-1,j) - \beta \ . \end{cases}$$

Theorem 2. *For the local alignment of the sequences S_1 and S_2, affine gap penalty α, β and substitution matrix sbt,*

$$nid(S_1, S_2) = N_A(i_{max}, j_{max})$$

where (i_{max}, j_{max}) denotes the coordinates of the maximum value in the corresponding matrix H_A.

Proof. Similar to the proof of Theorem 1 we show that for a given pair of indices i, j $(1 \le i \le l_1, 1 \le j \le l_2)$, $N_A(i,j)$ is equal to the number of exact matches in optimal i, j suffix alignment.

Case 1: $H_A(i,j) = 0$. The corresponding alignment is empty and $N_A(i,j) = 0$.
Case 2: $H_A(i,j) = H_A(i-1,j-1) + sbt(S_1[i], S_2[j])$. Similar to Case 2 in the previous proof follows $N_A(i,j) = N_A(i-1,j-1) + m(S_1[i], S_2[j])$.
Case 3: $H_A(i,j) = F(i,j)$. The alignment ends with $S_1[i]$ aligned to a gap, which contributes zero exact matches. The remaining number is equal to the number found in the optimal $i-1$, j suffix alignment. Depending on whether this alignment ends with a gap this number is either $N_A(i-1,j)$ or $N_F(i-1,j)$. Hence, $N_A(i,j) = N_F(i,j)$.
Case 4 $H_A(i,j) = E(i,j)$. Similar to Case 3 follows $N_A(i,j) = N_E(i,j)$.

Because $H_A(i,j)$ must be equal to one of these four cases, Theorem 2 is proven. □

3 Design and Implementation

We first implemented the recurrence equations in C, and ascertained that it produced equivalent results to the original ClustalW pairwise alignment stage. Hardware/software partitioning of the algorithm was carried out subsequently

using the CoDeveloper tool suite. Equivalent hardware descriptions were generated from the hardware portion of the C program for implementation in FPGA. We decided to create a compact microprocessor system built around the MicroBlaze soft processor from Xilinx in order to test and measure the performance of the generated design in contrast to conventional design verification techniques such as functional and timing simulation of the design. The high performance fast simplex link (FSL) of the processor system is used to connect the processor system and the MSA hardware processing element (PE).

In such a system, sequences, substitution matrix and gap penalties are stored in the processor system memory. They are then sent out to the hardware and pairwise alignment scores are read back from the hardware via the FSLs. We have targeted the system to the Xilinx ML403 development platform, which consists of a Virtex-4 FX12 FPGA. Our hardware PE can handle sequences with maximum length of 550 and supports both linear and affine gap penalties. Once the functionality of the design has been verified in hardware, design optimization techniques are employed to achieve high performance. In CoDeveloper, designers can implicitly or explicitly create the C code in such a way that the optimizer is able to detect the parallelism and generate highly parallel hardware. We mainly use the pipelining optimization technique in CoDeveloper to boost the design performance.

An initial optimized implementation of the system with a single MSA PE, which utilizes just 21% of the available hardware resources (slices), is able to achieve $\sim$16 MCUPS. As we can increase the device utilization by instantiating additional MSA PEs conveniently with CoDeveloper, we added two additional MSA PEs into the system. Figure 1 shows the system with three MSA PEs, which are connected to the processor via FSLs. The whole system now utilizes 99% of the available resources in which three MSA PEs contribute 62%. Performance evaluation of this system along with comparisons to a Pentium 4 system and other HDL-based solutions is provided in the next section.

4 Performance Evaluation

The above-mentioned system with three MSA PEs achieves a throughput of $\sim$36 MCUPS. Comparing this performance with that of a Pentium 4 2 GHz system, it is 2.4x faster. The FPGA on the ML403 platform is the smallest Virtex-4 FX family FPGA [8]. For larger FPGA devices such as Xilinx Virtex-4 LX80 and Virtex-II XC2V6000 FPGAs, 21 and 18 MSA PEs can be implemented into these devices respectively. Estimating 90% efficiency for MSA hardware, 302 and 259 MCUPS throughput can be expected. This performance is 20x and 17x faster than that of Pentium 4 2 GHz system.

Compared to the nearest FPGA-based Verilog HDL implementation, which makes use of XC2V6000 FPGA [9], our implementation is 3.9x slower since the implementation of [9] achieves a sustained performance of $\sim$1 GCUPS in the Smith-Waterman dynamic programming matrix. Although our implementation is also slower than other FPGA-based implementations using VHDL [10], [3],

these designs only implement edit distance, which is of theoretical interest but not used in practice [2] since it does not allow for different gap penalties and substitution tables. This is not the case for our implementation. Figure 2 shows the performance comparison of various systems discussed previously.

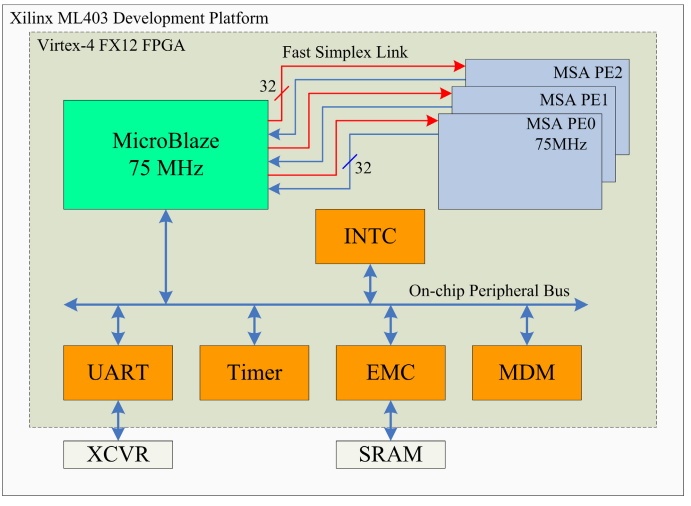

Fig. 1. MicroBlaze Processor System with Three MSA PEs

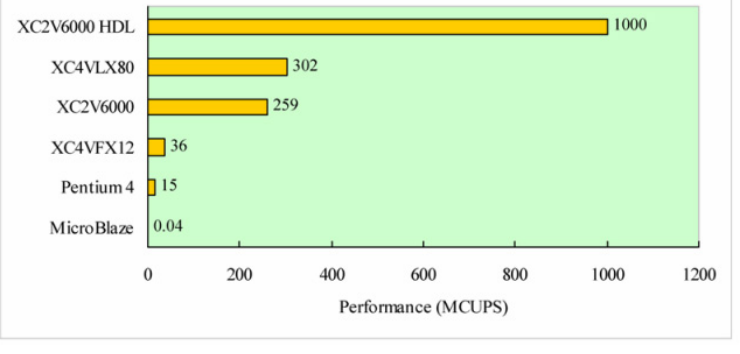

Fig. 2. ClustalW First Stage MSA Performance of Various Systems

5 Conclusion

Mapping of the first stage MSA algorithm into FPGA is solely based on the standard ANSI C language and application programming interfaces supported by the CoDeveloper tool. Neither HDL design nor simulation is done in this relatively new design methodology yet comparable high performance design can

be achieved with a significant reduction in the design time. Previous FPGA-based hardware accelerators [3], [9] have been based on large and power-hungry FPGA devices. We have demonstrated that a significant performance improvement is possible using even the smallest commercially available FPGA device, opening up possibilities for using these devices as on-demand accelerators for compute-intensive applications on mobile and power constrained devices.

Acknowledgments. We would like to thank Mr. David Pellerin from Impulse Accelerated Technologies for his effort in helping us to improve the MSA hardware design performance.

References

1. Ebedes, J., Datta, A.: Multiple Sequence Alignment in Parallel on a Workstation Cluster. Bioinformatics 20(7), 1193–1195 (2004)
2. Oliver, T.F., Schmidt, B., Maskell, D.L.: Reconfigurable Architectures for Bio-sequence Database Scanning on FPGAs. IEEE Trans. Circuits Syst. II 52, 851–855 (2005)
3. Hoang, D.T.: Searching Genetic Databases on Splash 2. In: IEEE Workshop on FPGAs for Custom Computing Machines, pp. 185–191. IEEE Computer Society Press, Los Alamitos (1993)
4. Yamaguchi, Y., Maruyama, T., Konagaya, A.: High Speed Homology Search with FPGAs. In: Pacific Symposium on Biocomputing, pp. 271–282 (2002)
5. Sullivan, C., Wilson, A., Chappell, S.: Using C based Logic Synthesis to Bridge the Productivity Gap. In: Proc. of the 2004 Conference on Asia South Pacific Design Automation, pp. 349–354 (2004)
6. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195–197 (1981)
7. Liu, W., Schmidt, B., Voss, G., Muller-Wittig, W.: Streaming Algorithms for Biological Sequence Alignment on GPUs. IEEE Trans. Parallel Distrib. Syst. (to be published)
8. Xilinx: Virtex-4 Family Overview. ds112.pdf (2007)
9. Oliver, T., Schmidt, B., Nathan, D., Clemens, R., Maskell, D.: Using Reconfigurable Hardware to Accelerate Multiple Sequence Alignment with ClustalW. Bioinformatics 21(16), 3431–3432 (2005)
10. Yu, C.W., Kwong, K.H., Lee, K.H., Leong, P.H.W.: A Smith-Waterman Systolic Cell. In: Proc. of 13th Int. Workshop Field Programmable Logic and Applications, pp. 375–384 (2003)