HIGH PERFORMANCE BIOSEQUENCE DATABASE SCANNING USING FPGAS

K. Benkrid, Y. Liu, and A. Benkrid
The University of Edinburgh, School of Engineering and Electronics, the King's Buildings, Mayfield Road, Edinburgh, EH9 3JL, Scotland
k.benkrid@ieee.org

ABSTRACT

This paper presents the design and implementation of a generic and highly parameterised FPGA-based core for pairwise biological sequence alignment. The core is captured in the Handel-C language, which allows for high level software-like descriptions of hardware architectures. It implements the sequence alignment algorithm in hand using a pipeline of basic processing elements. This results in high performance FPGA implementations tailored to the algorithm in hand. For instance, actual hardware implementations of the Smith-Waterman algorithm for protein sequence alignment achieve speed-ups in excess of 100:1 compared to equivalent standard PC-based software implementations.

Index Terms— Sequence alignment, systolic arrays, FPGAs, Smith Waterman, Handel-C

1. INTRODUCTION

Biological sequence alignment aims to find out whether two or more sequences (e.g. DNA or protein sequences) are related, which is of great importance in early disease diagnosis, drug engineering, and the study of evolutionary development [1]. Because of the exponential growth of biosequence databases, however, there is indeed a need to speed-up biosequence analysis algorithms in order to keep up with this growth rate. One way to achieve this has been to develop heuristics to reduce the search space and hence the execution time of sequence alignment algorithms [2][3]. The main drawback of these implementations is that the quality of the results is inversely proportional to the speed of execution of the heuristics [4]. In this paper, we concentrate on the exhaustive search algorithms where the quality of results is guaranteed upfront. For these, many Single Instruction Multiple Data and systolic architectures using special purpose hardware have been built to speed up their execution [5][6][7]. More recently, reconfigurable hardware in the form of Field Programmable Gate Arrays (FPGAs) has been used as a high performance programmable platform for sequence alignment [8][9][10]. Although early results have proved very promising, FPGA-based bioinformatics and computational biology, in general, suffers from a big knowledge gap between bioinformaticians and molecular biologists on the one side and hardware engineers on the other side [9].

In an attempt to make a contribution towards bridging the aforementioned gap, this paper presents the design and implementation of a generic and highly parameterized FPGA core for pairwise biological sequence alignment. Compared to previously published FPGA implementations, our solution provides the most parameterized one, hence allowing users to tune in FPGA hardware to suit their particular needs.

The remainder of this paper is organized as follows. Section 2 presents important background on biological sequence alignment algorithms. In section3, we highlight previous work in the area of high performance biological sequence alignment. The design and FPGA implementation of our highly parameterized core is detailed in section 4, after which a comparative evaluation of our implementation results is given. Finally, conclusions and plans for future work are laid out.

2. BACKGROUND

Biological sequences evolve through a process of mutation, selection and random genetic drift [11]. Mutation, in particular, manifests itself through the introduction of gaps. The degree of alignment of sequences is measured by a score, which is obtained by a summation of terms of each aligned pair of residue in addition to possible gap terms. Score terms for each aligned residue terms are obtained from probabilistic models and stored in a *score* or *substitution* matrix e.g. BLOSUM50 [1].

Gap penalties depend on the length of the gap and are generally assumed independent of the gap residues. There are two main types of gap penalties: linear and affine. In the former, the cost of a gap of length g is $Penalty(g)=-g*d$, where d is a constant. In the latter case, a constant penalty is assigned for opening a new gap, while a linear and smaller penalty is given to subsequent gap extensions: $Penalty(g)=-d-(g-1)*e$, where both d and e are constants.

The following discusses two alignment algorithms with linear gap penalty, namely: the Needleman-Wunsch algorithm and the Smith-Waterman algorithm. The case of affine gap will be explained in section 2.2.

2.1 Alignment Algorithms with Linear Gap Penalties

The Needleman-Wunsch algorithm is a dynamic programming algorithm for finding the optimal global alignment between two sequences $X = x_1x_2...x_i...x_M$ (of length M), and $Y = y_1y_2...y_i...y_N$ (of length N) [12]. The matrix F of scores, which indicates the best alignment between sequences X and Y, is built recursively using the following equation:

$$F(i,j) = \max\begin{cases} F(i-1,j-1) + s(x_i,y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d, \end{cases} \quad (1)$$

Where $F(0,0)=0$, $F(i,0)=-i*d$ and $F(0,j)=-j*d$. The best alignment between X and Y is the largest score of three alternatives:

- An alignment between x_i and y_j, in which case the new score is $F(i-1,j-1)+s(x_i,y_j)$ where $s(x_i,y_j)$ is the substitution matrix score or entry for residues x_i and y_j.
- An alignment between x_i and a gap in Y, in which case the new score is $F(i-1,j)-d$, where d is the gap penalty.
- An alignment between y_j and a gap in X, in which case the new score is $F(i,j-1)-d$, where d is the gap penalty.

This is the illustrated graphically in Figure 1 below.

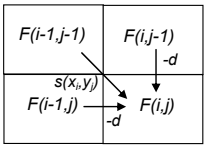

Figure1. Dynamic Programming and dependency illustration

It is, however, biologically more useful to look for the best alignment between subsequences of X and Y [1]. The Smith-Waterman algorithm finds exactly that, and is based on the following recursive equation [13]:

$$F(i,j) = \max\begin{cases} 0 \\ F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d \end{cases} \quad (2)$$

Compared to equation (1), the term 0 is added to the maximum. As a consequence of this, F(i,0) and F(0,j) should be set to 0's instead of $-i*d$ and $-j*d$ respectively.

2.2 Alignment with Affine Gap

A similar algorithm to the one presented for linear gap penalties can be used for affine gap. However, multiple values of each pair of residue *(i,j)* need to be computed instead of one. Figure 2 illustrates the case where three values need to be computed for each residue pair [1]. These equations assume that an insertion is not directly followed by a deletion (or vice versa). This can be guaranteed for the optimal path if the lowest possible mismatch score in the substitution matrix is greater than $-d-e$.

I G A x_i
L G V y_j

A I G A x_i
G V y_j -

G A x_i - -
S L G V y_j

$$F(i,j) = \max\begin{matrix} F(i-1,j-1) + s(x_i,y_j), \\ I_x(i-1,j-1) + s(x_i,y_j), \\ I_y(i-1,j-1) + s(x_i,y_j) \end{matrix} \qquad I_x(i,j) = \max\begin{cases} F(i-1,j) - d, \\ I_x(i-1,j) - e, \end{cases} \qquad I_y(i,j) = \max\begin{cases} F(i,j-1) - d, \\ I_y(i,j-1) - e, \end{cases}$$

Figure2. The case of affine gap penalties

Another algorithm is used in the case of affine gap penalties if the lowest possible mismatch score in the substitution matrix is greater than $-2e$. In it, there are two values for each residue pair instead of three as shown below.

$$F(i,j) = \max\begin{matrix} F(i-1,j-1) + s(x_i,y_j), \\ I(i-1,j-1) + s(x_i,y_j) \end{matrix} \quad (3)$$

$$I(i,j) = \max\begin{matrix} F(i,j-1) - d, \\ I(i,j-1) - e, \\ F(i-1,j) - d, \\ I(i-1,j) - e \end{matrix}$$

3. HIGH PERFORMANCE BIOLOGICAL SEQUENCE ANALYSIS: PREVIOUSE WORK

The computational complexity of the above dynamic programming algorithms for pairwise sequence alignment is proportional to the product of the lengths of the two sequences to be aligned i.e. O(M*N). Scanning a database with hundreds of thousands of sequences usually takes several hours on a PC [1][8]. Given the exponential growth rate of biological sequence databases, this problem is set to intensify.

In order to speed up sequence analysis algorithms, a number of parallel architectures have been developed. Single Instruction Multiple Data (SIMD) architectures based on a network of programmable processors are among these solutions, and include the MGAP [5], Kestrel [6] and Fuzion [7]. Although such architectures are capable of considerable speed-ups compared to a standard PC solution, they are often costly both in terms of design and programming [10].

Other solutions have used special purpose hardware for the implementation of parallel processing elements with the aim of increasing processing density and achieving even higher speed-ups. Such architectures also allow for systolic arrays to be implemented. Instances of this family of architectures include BISP [14], SAMBA [15] and BIOSCAN [16]. The advent of reconfigurable hardware in the form of FPGAs makes such architectures even more appealing. FPGAs after all are capable of providing considerable speed-ups compared to general purpose processors with the added convenience of reprogramability. An algorithm implementation could hence be tuned to different needs both at compile time and at run-time. Moreover, FPGAs are now riding the process technology curve [17] which makes them even more attractive a solution as a reliable high performance platform for biocomputing [18][19]. For instance, a number of FPGA implementations of the Smith-Waterman algorithm have been reported in the literature recently [8][9][10]. However, none of these implementations offers the same degree of parameterisability as our implementation. The latter was designed using a high level hardware language in the form of the ANSI-C based Handel-C language [20], and achieved performance figures comparable to the best results reported in the literature, if not better, as will be shown in Section 5. This in itself is important as it means that higher level hardware languages can be used to achieve high performance implementations of computational biology applications, and hence bridge the aforementioned gap between bioinformatics and computational biology applications and high performance hardware platforms.

4. OUR HARDWARE IMPLEMENTATION

Figure 3 presents a linear systolic array implementation for general purpose pairwise sequence alignment based on the dynamic programming algorithms presented in section 2 above. The linear systolic array consists of a pipeline of basic processing elements (PE) holding query sequence residues, whereas the subject sequence is shifted systolically through the array (from the database). A FIFO is used to store intermediate results for multi-pass processing, which is only needed when the FPGA chip in hand cannot fit all of the query sequence. Each PE holds one or more residues of the query sequence and performs one elementary calculation in one clock cycle thereby generating one alignment matrix element *F(i,j)*. However the calculation at PE_{i+1} depends on the result from PE_i, which means that each PE is one cycle behind its predecessor. The full alignment of two sequences of lengths *N* and *M* is hence achieved in *M+N-1* cycles. Figure 4 illustrates the execution of the recursive equation of the Smith Waterman algorithm in such architecture. The elements on each diagonal line can be computed in parallel in one clock cycle.

As stated above, multi-pass processing is needed when the query sequence cannot fit into the FPGA chip in hand. This is usually the case for real world databases. For instance, the number of PEs that could be implemented on a Xilinx XC2V6000 Virtex-II FPGA in the case of the Smith-Waterman Algorithm with affine gap penalties is ~250. This represents the maximum query length that can be fitted on the FPGA. Real world biological sequence lengths, however, are often in the hundreds if not in the thousands. In such cases, the algorithm in hand should be partitioned into small alignment steps which are then mapped onto a fix size linear

systolic array. This problem is well studied in the VLSI design arena and involves proper scheduling of input and intermediate data [21].

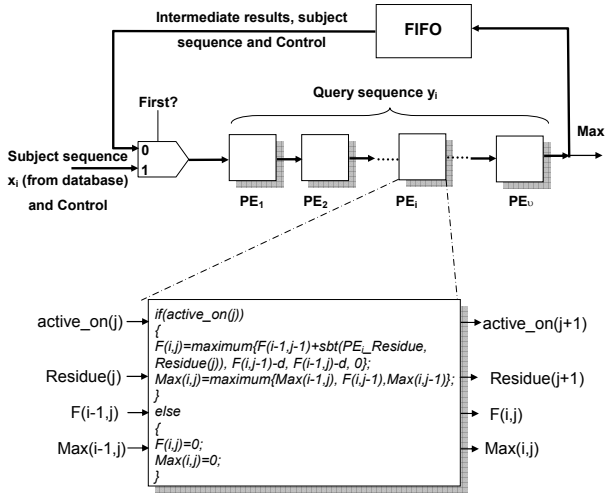

Figure3. Smith-Waterman sequence alignment algorithm on fixed size systolic array architecture

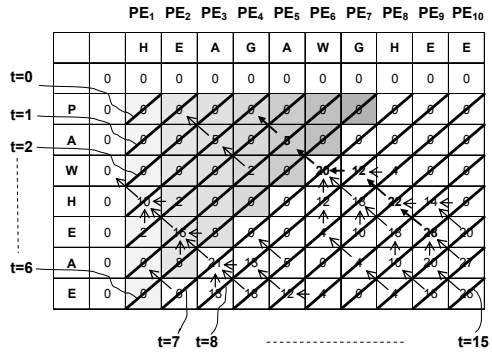

Figure4. Illustration of the execution of the Smith-Waterman example on the linear array processor of Figure 3

Based on the hardware architecture presented in Figure 3, we have captured all of the variations of a generic pairwise sequence alignment algorithm into a single FPGA core written in Handel-C [20]. The final core that we have developed is prameterisable in terms of the *sequence symbol type* e.g. DNA or Protein sequences, *query sequence*, *maximum subject sequence length*, the *match score* i.e. the score attributed to a symbol match depending on the substitution matrix used, the *gap penalty, linear or affine*, the *matching task* i.e. the alignment algorithm used to match sequences, which could be global or local, and the *match score threshold*, which is the match score threshold below which any subject sequence is rejected. The core automatically infers the necessary minimum processing wordlength based on the user-supplied parameters and harnesses constant propagation. It has been written without any FPGA-specific directives e.g. specific resource inference or placement constraints, which makes it directly retargetable across a variety of platforms including Xilinx and Altera FPGAs.

5. PERFORMANCE AND EVALUATION

Celoxica's DK4 suite was used to compile our core into EDIF, whereas Xilinx ISE7.1 tool was used to generate the configuration bitstreams for Xilinx FPGAs. A single PE in the case of the Smith-Waterman algorithm for protein sequences, with linear gap penalty and 16 bit processing wordlength, consumes ~30 slices on average on Xilinx Virtex-II FPGAs, whereas an equivalent affine gap penalty using the equations given in Figure 2, consumes ~85 slices. An equivalent single PE with affine gap penalty using Equations (3), however, consumes ~70 slices.

Table 1 presents sample performance figures for instances of our core assuming protein sequences and single pass implementations. Affine2 and Affine3 refer to the affine gap models given in Equations (3) and Figure 2 respectively. The CUPS (or Cell Updates Per Second) performance is a common performance measure used in computational biology. Its inverse represents the equivalent time needed for a complete computation of one entry of the alignment matrix, including all the comparisons, additions and maximum computations. The peak CUPS of our implementation is measured by multiplying the number of PEs and the maximum clock frequency.

Number of PEs	Gap Penalty	Processing Word length	Max Speed (MHz)	Peak Performance (CUPS x10^9)
Needleman-Wunsch				
252	Linear	16	50.6	12.75
Smith-Waterman				
100	Linear	16	43.5	4.35
252	Linear	10	47.7	12.02
100	Affine2	16	66.7	6.67
168	Affine2	16	47.6	8.00
100	Affine3	10	58.8	5.88
168	Affine3	16	40.0	6.72

Table1. Core performance for different instances of our core on a Xilinx XC2V6000-4 FPGA

Table 2 presents sample results for instances of our core assuming protein sequences and multi-pass implementations with k (the number of passes, or folding factor) equal to 3 and 12 respectively.

Number of PEs	Gap Penalty	Processing Word length	Clock frequency (MHz)	Peak Performance (GCUPS)
k=3				
252	Linear	10	40.0	10.09
168	Affine2	10	62.5	10.50
168	Affine3	10	45.6	7.66
k=12				
168	Linear	10	40.3	6.77
119	Affine3	10	50.4	5.99

Table2. Core performance for different instances of a multi-pass implementation (i.e. k>1)

Equivalent software implementations written in C and running on a Pentium-4 1.6 GHz achieve ~50 MegaCUPS performance, which means that the above core outperforms equivalent software implementation by two order of magnitudes (100:1+). Given the relative cost of FPGAs compared to general purpose processors (often in the order of 10:1) the above performance largely offsets their cost, which shows that FPGAs could be an economical implementation platform for biological sequence analysis applications.

Compared to an implementation reported in [9] on a Xilinx XC2VP30 FPGA, our core achieves twice the speed. It also outperforms the implementation reported in [8] by 3:1. The

Verilog-based implementation reported in [10], however, is the closest to our core implementation as it is based on the same hardware architecture. Compared to it, our core performs almost as well despite the fact that we have not introduced any placement constraints. This is a testament to the Handel-C language as well as the corresponding synthesis tool.

Moreover, none of the above implementations offer the same degree of parameterisation as our core. Indeed, the implementation reported in [8] only supports the Smith-Waterman algorithm with linear gap penalty, whereas the implementation reported in [9] does not address the problem of partitioning/mapping. The implementation in [10] supports both partitioning/mapping and affine gap penalties. However, its affine gap model is based on the equations given in Figure 2 only, and hence does not take advantage of the hardware optimisations introduced by Equations (3).

6. CONCLUSION

In this paper, we have presented the design and implementation of a highly parameterised core for FPGA-based pairwise biological sequence alignment. The core is parameterised in terms of the sequence symbol type, the sequence lengths, the match score, the gap penalty and the matching task. It implements the algorithm in hand using a pipeline of basic processing elements, with a number of built-in hardware optimisations. These include automatic minimum wordlength inference and compile-time constant propagation. The core results in high performance FPGA implementations which outperform equivalent desktop-based software implementations by two order-of magnitudes. This shows that FPGAs could offer a relatively low cost high performance implementation platform for biological sequence alignment algorithms.

The core has been captured in the Handel-C language which proved very convenient in describing scalable and parameterised hardware architectures, with a relatively lower learning curve compared to other hardware description languages.

The work presented in this paper is part of a bigger effort by the authors which aims to harness the computational performance and reprogramability features of FPGAs in the field of Bioinformatics and Computational Biology. Future work includes the implementation of sub-optimal sequence alignment algorithms including the BLAST algorithm on FPGAs, as well as the use of Hidden Markov Models for biological sequence analysis.

7. REFERENCES

[1] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G., '*Biological Sequence Analysis: Probabilistic Models for Proteins and Nucleic Acids'*, Cambridge University Press, Cambridge UK, 1998

[2] Altschul, S. F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. '*Basic Local Alignment Search Tool*', Journal of Molecular Biology, 215, pp. 403-410, 1990.

[3] Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. '*Gapped BLAST and PSI-BLAST: a new generation of protein database search programs*', Nucleic Acid Research, Oxford Journals, 25(17), pp. 3389-3402, 1997.

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

[5] Borah, M., Bajwa, R.S., Hannenhalli, S., and Irwin, M.J. '*A SIMD solution to the sequence comparison problem on the MGAP'*, ASAP'94 Proceedings, IEEE Computer Science, pp. 144-160, 1994

[6] Dahle, D., Grate L., Rice, E., and Hughey, R. '*The UCSC Kestrel general purpose parallel processor'*, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 1243-1249, 1999

[7] Schmidt, B., Schröder, H., and Schimmler, M '*Massively Parallel Solutions for Molecular Sequence Analysis*', Proceedings of the 1st IEEE International Workshop on High Performance Computational Biology, pp. 186-193, 2002.

[8] Yamaguchi, Y., Maruyama, T., and Konagaya, A. 'High Speed Homology Search with FPGAs', Proceedings of the Pacific Symposium on Biocomputing, pp.271-282, 2002.

[9] VanCourt, T. and Herbordt, M. C. 'Families of FPGA-Based Algorithms for Approximate String Matching', Proceedings of Application-Specific Systems, Architectures, and Processors, ASAP'04, pp. 354-364, 2004

[10] Oliver, T., Schmidt, B. and Maskell, D. 'Hyper customized processors for bio-sequence database scanning on FPGAs', Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays

[11] Harrison G. A., Tanner, J. M., Pilbeam D. R., and Baker, P. T. '*Human Biology: An introduction to human evolution, variation, growth, and adaptability*', Oxford Science Publications, 1988

[12] Needleman, S. and Wunsch, C. '*A general method applicable to the search for similarities in the amino acid sequence of two sequences'* Journal of Molecular Biology, 48(3), pp.443-453, 1970

[13] Smith, T.F. and Waterman, M.S. Identification of common molecular subsequences. J. Mol. Biol., 147, pp.195-197, 1981

[14] Chow, E., Hunkapiller, T., Peterson, J., Waterman, M.S. '*Biological Information Signal Processor*', Proceedings of Application-Specific Systems, Architectures, and Processors, ASAP'91, pp. 144-160, 1991.

[15] Guerdoux-Jamet, P., Lavenier, D. '*SAMBA: hardware accelerator for biological sequence comparison*', Computer Applications in Biosciences, CABIOS, 12 (6), pp. 609-615, 1997.

[16] Singh, R.K. et al. '*BIOSCAN: a network sharable computational resource for searching biosequence databases*', Computer Applications in Biosciences, CABIOS, 12(3), pp. 191-196, 1996.

[17] Butts, M. All chips will be reconfigurable, Tutorial, 13th International Conference on Field Programmable Logic and Applications, September 2003

[18] Hoang, D.T. '*Searching genetic databases on Splash 2*', in Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, pp. 185-191, 1993.

[19] TimeLogic Corporation, '*Decypher Scalable, High Performance Biocomputing Solutions*', http://www.timelogic.com/

[20] The Handel-C Language Reference Manual, Celoxica Plc, http://www.celoxica.com

[21] Kung, S. Y. '*VLSI Array Processors*', Prentice-Hall, 1988