Hardware Accelerator for Genomic Sequence Alignment

Jason Chiang, **Michael Studniberg, Jack Shaw, Stephen Seto**, **Kevin Truong***, Member, IEEE

Abstract— To infer homology and subsequently gene function, the Smith-Waterman algorithm is used to find the optimal local alignment between two sequences. When searching sequence databases that may contain billions of sequences, this algorithm becomes computationally expensive. Consequently, in this paper, we focused on accelerating the Smith-Waterman algorithm by modifying the computationally repeated portion of the algorithm by FPGA hardware custom instructions. These simple modifications accelerated the algorithm runtime by an average of 287% compared to the pure software implementation. Therefore, further design of FPGA accelerated hardware offers a promising direction to seeking runtime improvement of genomic database searching.

I. Introduction

THE Smith-Waterman (SW) algorithm is a well-known algorithm in computational molecular biology that finds the optimal alignment between two DNA sequences (one called target sequence and the other, search sequence) [1]. Determining how well two sequences align is useful for finding related genes or constructing phylogenetic trees [2]. However, the SW algorithm is not used in sequence database searching because it is too slow when executed against many sequences. Instead other faster algorithms such as FASTA[3] and BLAST[4] are used, but they are not guaranteed to find the optimal local alignment. Therefore, to achieve fast as well as optimal alignment, it is necessary to develop an approach to reduce the computational processing runtime of the SW algorithm. Briefly, this algorithm is done by first generating a two dimensional (2D) matrix with its size equal to the lengths of the DNA sequences. The score of each cell in the 2D matrix calculated from its neighboring cells. Finally, the optimal alignment between the two DNA sequences is determined by backtracking from the cell with the highest score to the first cell with a zero score.

Here, we reduced the runtime of the SW algorithm using a Field Programmable Gate Array (FPGA) board. This algorithm is a good candidate for hardware acceleration because the scoring calculation for each cell in the 2D matrix is repeated. Thus, a small acceleration of the scoring results in a huge overall acceleration. The FPGA is a reconfigurable device whose digital logic gates can be optimally configured to run specific functions through the implementation of custom microprocessor instructions. In particular, FPGA custom instruction allows the passing of multiple inputs and outputs in a single clock cycle whereas the pure-software implementation using a conventional microprocessor can only pass two. In this paper, we studied the improvement of computational processing time of the SW algorithm using custom instructions on an FPGA board. This was done by first writing the SW algorithm in pure software and then replacing the portion which was the most computational intensive with an FPGA custom instruction. Finally, we compared the processing runtime between the "pure software" and the "hardware acceleration" versions to calculate the percentage of runtime improvement.

This work was supported by the Banting Foundation and the Natural Sciences and Engineering Research Council of Canada (NSERC) under Research Grant RG-PIN 276250.

JC, MS, JS and SS are with the department of Electrical and Computer Engineering (ECE), University of Toronto, Toronto, ON, Canada.

JC is also with the Institute of Biomaterials and Biomedical Engineering (IBBME), University of Toronto, Toronto, ON, Canada.

*KT is also with ECE and IBBME, University of Toronto, Toronto, ON, Canada (kevin.truong@utoronto.ca)

II. Software Development of the SW Algorithm

We first described the SW algorithm and then its pure-software implementation in the C language. This pure-software implementation is necessary when comparing against the modification using hardware.

A.. SW Algorithm Description

The SW algorithm belongs to a class of algorithms known as dynamic programming. Dynamic programming is used when a large search space can be structured into a succession of stages such that the initial stage contains trivial solutions to sub-problems [5]. Typically, this involves structuring the problem to an iterative calculation of cells in a matrix. There exist different scoring schemes for the SW algorithm. Our scheme to compute the score in a cell of interest "X" is illustrated below:

$$\text{score\_X} = \max \text{of} \begin{cases} \text{i) score\_nw+1 if } S_i = T_j & \text{(match)} \\ \text{ii) score\_anw-1, if } S_i \neq T_j & \text{(mismatch)} \\ \text{iii) score\_n - 1} & \text{(gap penalty)} \\ \text{iv) score\_w - 1} & \text{(gap penalty)} \end{cases} \quad (1)$$

S_i is the i^{th} letter in the search sequence; T_j is the j^{th} letter in the target sequence; score_nw, score_n, score_w are the score of the cells in the upper diagonal, above and to the left of cell "X", respectively (Fig 1). Thus, the score of each cell in the 2D matrix (except for row 1 and column 1) is calculated by its neighbouring cells.

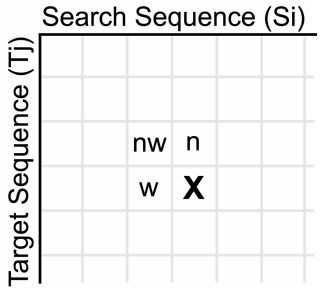

Figure 1. Basic structure of the SW matrix. nw, n, and w are the scores from the neighbouring cells and **X** is the single cell of interest.

Even though the SW algorithm is guaranteed to find the optimal pairwise local alignment, it is demanding of time and memory resources. If the two sequences being aligned are m and n in length respectively, then the computational complexity of this implementation is O(mn).

B. Software implementation

We developed a purely software-based version of the SW algorithm in C language to serve as the benchmark for comparison. A high-level description of the program is described as follows:

- Data acquisition phase: obtain the target and search sequences from two text files and determine their sequence lengths.
- Initialization phase: create an empty two 2D score matrix with the size equal to the sequence lengths. The first matrix, called the SW matrix, stores the scores from each comparison between the two query DNA sequences. The second, called the gap matrix, stores the direction of the gap for each cell.
- Evaluation phase: execute the SW algorithm. The score of each cell in the SW matrix is calculated based on the current letters being compared and the calculated scores and gaps from the neighbouring cells.
- Result phase: output the completed SW matrix, display the highest score in the matrix and determine the resultant alignment sequence as well as the total runtime for the algorithm.

Note that the hardware modification using a FPGA board only acted on the evaluation phase as this phase represents the most computationally intensive portion of the algorithm. Therefore, for programming convenience the evaluation phase was treated as an independent module with the parameters below:

Evaluation (score_nw, score_n, score_w, flag_nw, flag_n, flag_w, flag_gap, result_score)

flag_nw, flag_n, and flag_w are inputs that are equal to 1 when there is a match between the neighboring cells and the cell of interest and 0 otherwise; score_gap and the result_score are outputs that indicate the direction of the gap (0: no gap, 1: gap from the target sequence, 2: gap from the search sequence) and the final score for the cell of interest, respectively.

II. INTEGRATED ACCELERATION APPROACH AND COMPARISON

The final integrated system contains a microprocessor that executes the C program and hardware custom instructions. After execution, the percentage of runtime improvement is be calculated by comparing the runtime of the accelerated against the pure software implementation.

A. Custom Instruction for the Evaluation Module in Verilog

We accelerated the SW algorithm by replacing the evaluation module (section 2.2) with FPGA custom instructions written in Verilog. Custom instructions (CI) are assigned digital logic gates that perform user-defined operations. Particularly, we designed CIs on an Altera Nios II integrated development environment (IDE). The Nios II soft microprocessor was instantiated on an FPGA to allow rapidly prototyping of new designs. Since the format for the CI provided by Altera only permits two 32-bit inputs and our single cell CI designed requires 6 inputs (3 scores and 3 flags), the inputs must be partitioned and rearranged such that they can be all read in a single clock cycle. Recall that the inputs for evaluation module in the pure-software implementation are score_nw, score_n, score_w, flag_nw, flag_n, and flag_w. Using bit masking and shifting bit operations, all three input score from the neighboring cells along with their flags can be passed to the cell of interest in one clock cycle (Fig 2).

CIs were written in Verilog and instantiated on the FPGA with the Nios II microprocessor. This integrated the CIs into instruction set of the microprocessor. Therefore, the CIs could be called from a C program.

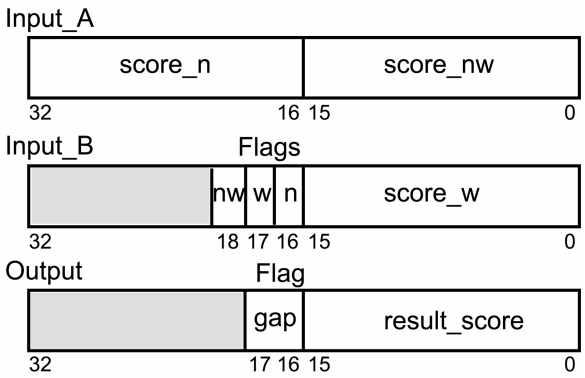

Figure 2. Bit partition of custom instruction for the evaluation module. The input_A and input_B are two 32-bit data containing the scores and flags from the three neighboring cells, and the output is one 32-bit data containing the final scores and the direction of alignment gap. The gray areas indicate the unused data bits.

B. Runtime Comparison of the Accelerated against Pure Software Implementation

Fifteen test cases using sequence sizes ranging from 100 to 1,500,000 cells were used to analyze the performance of the accelerated implementation. First, the test cases were run on the pure software implementation and the runtimes were recorded. Next, the same procedure was performed on the accelerated implementation. The runtime improvement is calculated using the following equation:

$$\%\_runtime\_improvement = \left(\frac{\frac{1}{t_{ART}} - \frac{1}{t_{SRT}}}{\frac{1}{t_{SRT}}} \right) * 100\% = \left(\frac{t_{SRT}}{t_{ART}} - 1 \right) * 100\%$$

t_{ART} is the runtime for the accelerated implementation; t_{SRT}, runtime for the pure software implementation. Employing this equation, the average improvement from using the FPGA was 287% (fig 3).

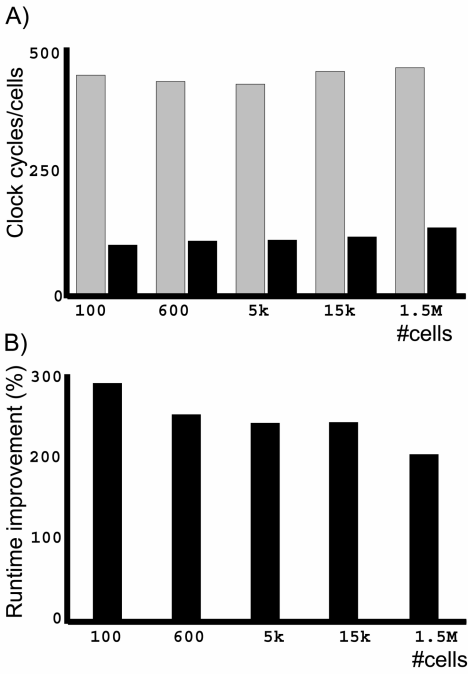

Figure 3. Runtime comparison and % improvement. A) The runtime (in clock cycles per cell) between the pure software version (in gray) and integrated system (in black) of five selected cell sizes. B) % runtime improvement for the accelerated against pure software implementation of the corresponding cell sizes.

C. Graphical User Interface

To provide a user-friendly environment for the final integrated system, a graphical user interface (GUI) was written in Visual Basic 6.0 (Fig 4). The GUI was designed to perform all operations on the board via a serial port. The main features of the GUI are described as follows:

- Inputing genomic sequences
- Saving output results to files
- Displaying of the computational progress
- Comparing the performances between the software and hardware-accelerated implementation
- Calculating runtime % improvement

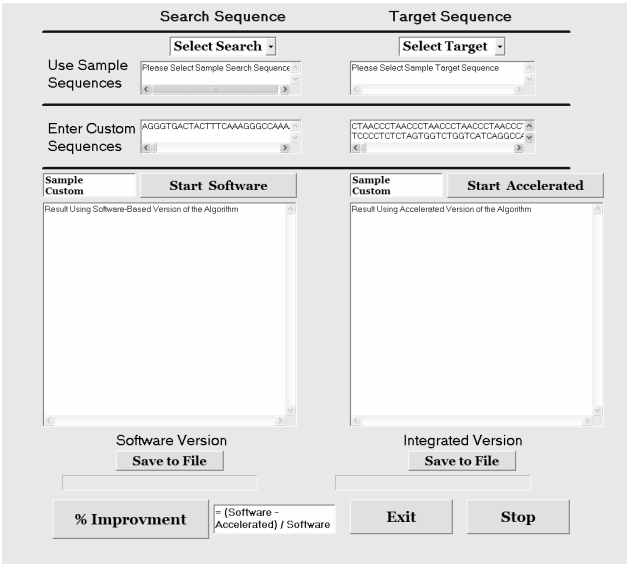

Figure 4. Screenshot of the GUI

III. CONCLUSION

Since the SW algorithm becomes computationally expensive for comparing sequences in a large database, we accelerated the runtime by using FPGA hardware. To quantitatively assess the runtime improvement, we first wrote the algorithm in software and then accelerated it using FPGA CIs. The results showed that the hardware accelerated algorithm improved the processing runtime by an average of 287%. Thus, using FPGAs is a promising direction for further research in improving genomic sequence searching.

ACKNOWLEDGEMENT

This work was supported by grants from the Canadian Foundation of Innovation (CFI) and the National Science and Engineering Research Council (NSERC).

REFERENCES

[1] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J Mol Biol*, vol. 147, pp. 195-7, 1981.

[2] R. D. Page, "GeneTree: comparing gene and species phylogenies using reconciled trees," *Bioinformatics*, vol. 14, pp. 819-20, 1998.

[3] D. J. Lipman and W. R. Pearson, "Rapid and sensitive protein similarity searches," *Science*, vol. 227, pp. 1435-41, 1985.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *J Mol Biol*, vol. 215, pp. 403-10, 1990.

[5] R. Giegerich, "A systematic approach to dynamic programming in bioinformatics," *Bioinformatics*, vol. 16, pp. 665-77, 2000.