Performance Improvement of the Smith-Waterman Algorithm

Laiq Hasan Zaid Al-Ars
Delft University of Technology
Computer Engineering Laboratory
Mekelweg 4, 2628 CD Delft, The Netherlands
Tel: +31 15 27 86172 Fax: +31 15 27 84898
Email: L.Hasan@ewi.tudelft.nl

Abstract: *Efficient sequence alignment is one of the most important and challenging activities in bioinformatics. Many algorithms have been proposed to perform and accelerate sequence alignment activities. Among them Smith-Waterman (S-W) is the most sensitive (accurate) algorithm, however, the fact that it is the most computationally intensive algorithm makes it necessary to implement hardware acceleration methods to make the algorithm viable for practical applications. In this paper, we present a novel approach to improve the performance of the S-W algorithm, using partially custom hardware. In this approach, customized hardware is used to accelerate the computationally intensive part of the algorithm, rather than implementing the entire algorithm in hardware. The paper shows the profiling results of a pure software implementation of the S-W algorithm. The profiling results identify that a specific small part of the algorithm consumes a disproportionately large amount of computational time, amounting to 72.33 % of the total runtime. The paper then shows that implementing this part in hardware, results in a 35.82 times speedup relative to its software equivalent.*

Keywords: *Bioinformatics, Sequence Alignment, Smith-Waterman Algorithm, Code Profiling, Custom Hardware.*

I. Introduction

In 1981, Smith and Waterman described a method based on *dynamic programming (DP)* [1], commonly known as the *Smith-Waterman (S-W)* algorithm [2], for local sequence alignment (i.e., identifying common regions in sequences that share local similarity characteristics). When obtaining the local alignment, a matrix $H_{i,j}$ is used to keep track of the degree of similarity between the two sequences to be aligned (A_i and B_j). Each element of the matrix $H_{i,j}$ is calculated according to the following equation:

$$H_{i,j} = \max \begin{cases} 0 \\ H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - d \\ H_{i,j-1} - d \end{cases} \quad (1)$$

where $S_{i,j}$ is the similarity score of comparing sequence A_i to sequence B_j and d is the penalty for a mismatch.

The whole algorithm is divided into three steps:

1. Initialization step
2. Matrix fill step
3. Trace back step

The matrix is first initialized with $H_{0,j} = 0$ and $H_{i,0} = 0$, for all i and j. This is referred to as the *initialization step*.

After the initialization, a *matrix fill step* is carried out using Equation 1, which fills out all entries in the matrix.

The final step is the *trace back step*, where the scores in the matrix are traced back to inspect for optimal local alignment. The trace back starts at the cell with the highest score in the matrix and continues up to the cell, where the score falls down to a predefined minimum threshold. In order to start the trace back, the algorithm requires to find the cell with the maximum value, which is done by traversing the entire matrix.

As an example, the S-W algorithm, is used to compute the optimal local alignment of two sequences (i.e., A = a g g t a c and B = c a g c g t t g). Assume that

$$S_{i,j} = \begin{cases} +2 \ \ if(A_i = B_j) \\ -1 \ \ else \end{cases}$$

and $d = 2$.

Table I illustrates the calculation of the DP matrix H and the trace back path (shown in bold digits). The best score found in the matrix is 6, and the corresponding optimal local alignment is

A:	a	g	-	g	t
B:	a	g	c	g	t

This paper aims at providing a novel approach to improve the performance of the S-W algorithm, using partially custom hardware. Some form of the S-W algorithm is also the core of FASTA [3] and BLAST [4], making its acceleration even more useful, as it can later be used for improving the performance of FASTA and BLAST as well.

TABLE I
THE DP MATRIX AND THE TRACE BACK PATH

		c	a	g	c	g	t	t	g
	0	**0**	0	0	0	0	0	0	0
a	0	0	**2**	0	0	0	0	0	0
g	0	0	0	**4**	**2**	2	0	0	2
g	0	0	0	2	3	**4**	2	0	2
t	0	0	0	0	1	2	**6**	4	2
a	0	0	2	0	0	0	4	5	3
c	0	2	0	1	2	0	2	3	4

The paper is organized as follows: Section II provides an overview of the related work and also introduces the partially custom hardware approach. Section III shows the profiling results of a software implementation of the S-W algorithm. Section IV shows the hardware description of the function (*fill_matrix_2*), which consumes 72.33 % of the total runtime and Section V gives a brief conclusion.

II. RELATED WORK

Figure 1 gives classification of the work related to the hardware acceleration of the S-W algorithm [5].

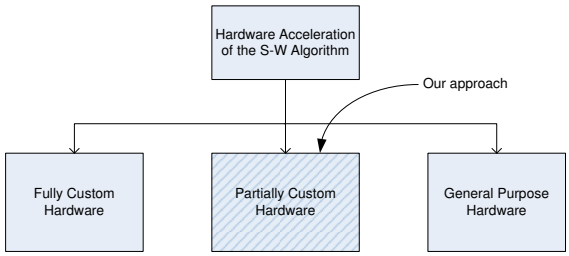

Fig. 1. Classification for hardware acceleration of the S-W algorithm

Fully Custom Hardware is the dedicated hardware designed to perform only some specialized tasks. In [6], the authors show the implementation of a fully custom processing unit to realize the execution of the S-W algorithm. The authors state that for conducting comparisons of multiple sequence pairs, using the same set of processing units, two approaches can be taken e.g. synchronous and asynchronous. The authors claim that the asynchronous parallel approach is (k-1)*(m-1) time steps faster than the synchronous parallel approach, where k represents the size of the existing sequences in the database, which grows exponentially. In [7], the design of a small fully custom processing element, called *Proclet*, is shown. This Proclet is used for a new VLSI implementation of the S-W algorithm. In [8], the authors show the runtime improvement of the S-W algorithm, using FPGA hardware. To quantitatively assess the runtime improvement, they first wrote the algorithm in software and then accelerated it, using FPGA custom instructions. The results showed that the hardware accelerated algorithm improved the processing runtime by an average of 287%.

General Purpose Hardware is more flexible and is designed to perform a wide variety of tasks. In [9], an implementation of the S-W algorithm is described on a general purpose fine-grained architecture, the MGAP, where the authors show that their implementation is about 5 times faster than the rapid implementation of a genetic sequence comparator using field programmable logic arrays [10]. In [11], it has been demonstrated that the streaming architecture of GPUs can be efficiently used for biological sequence database scanning. To derive an efficient mapping onto this type of architecture, the authors have reformulated the S-W algorithm in terms of computer graphics primitives and claim that the evaluation of their implementation on a high-end graphics card shows a speedup of almost sixteen, compared to a Pentium IV 3.0 GHz. The platforms in [9] and [11] are not customized particularly for hardware acceleration of the S-W algorithm and are therefore included in the general purpose hardware class.

Partially Custom Hardware is neither as flexible as general purpose hardware nor as fixed as fully custom hardware. We propose a novel approach to accelerate the S-W algorithm, using partially custom hardware. In this approach, we design a *Custom Computing Unit (CCU)* for the function of interest (identified in the code profile), rather than the entire application. Figure 2, shows the block diagram representation of our approach, where the block in gray represents the desired function, identified by the code profile. This is actually the function, for which, we need to design a CCU.

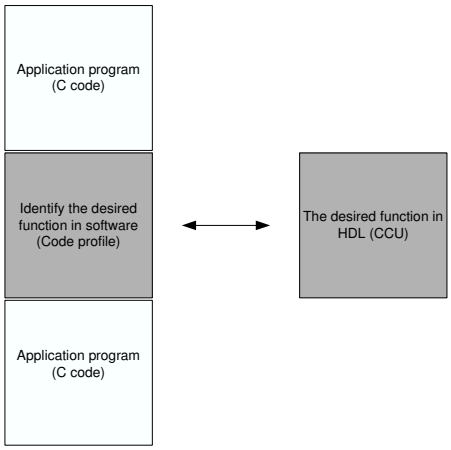

Fig. 2. Block diagram representation of the partially custom hardware approach

To our knowledge, no such approach has been presented in the literature as yet to accelerate the S-W algorithm.

III. Profiling Results

Figure 3 shows the functional description of a software implementation of the S-W algorithm. The *init_matrix* is a function used for initializing the scoring matrix. The *fill_matrix_1* performs two functions i.e. filling the matrix and at the same time keeping track of the maximum score in the matrix. The fill_matrix_2 function finds the corresponding maximum candidate for each cell in the matrix, according to Equation 1. The *trace_back_1* function performs the trace back, while the *trace_back_2* function keeps track of the direction of the trace back. We profiled a software implementation of the S-W algorithm, using the GNU profiler, gprof. Table II shows the code profiling results.

The GNU profiler gives information about the number of times, each function is called and the number of *Clock Ticks*, consumed by each function, whereas a

$$\text{Clock Ticks} = \frac{\text{Number of Clock Cycles}}{32}$$

So, $\text{Number of Clock Cycles} = \text{Clock Ticks} * 32$

The code is run on the Intel Pentium-IV (3.2 GHz) processor, for which the time period of the clock is

$$= \frac{1}{3.2\ \text{GHz}} = 0.312\ \ ns$$

So, the total time in Table II represents the product (total number of clock cycles * 0.312 *ns*).

Table II identifies that fill_matrix_2 is the function which is called the most number of times and consumes 72.33 % of the total runtime, making it the right candidate to be implemented in hardware. This will improve the computational processing time of the function. Table II shows the total time consumed by fill_matrix_2 function as 5.23 *ms*. This is actually the time when the code is repeated 100 times. This is due to the fact that in the software implementation, a code repetition counter is defined and for retrieving some meaningful profiling results, this counter is set to 100. So the actual time consumed by fill_matrix_2 function

$= \frac{5.23}{100}$ *ms*

= 0.05232 *ms*

= 52.32 μs

The comparison between the processing runtime of the pure software and the hardware versions will give us the percentage of runtime improvement, as given in Section IV.

IV. Hardware Description of the Function (fill_matrix_2)

We designed a CCU in VHDL for the function of interest (fill_matrix_2). Figure 4 shows the RTL schematic of the CCU, generated by the synthesizer (*Xilinx ISE 8.1i*). It is a synchronous comparator, which compares four 8 bits numbers and finds the maximum in two levels. For this purpose three similar 8 bit comparators are inferred, each having two 8 bits inputs *in1* and *in2*, a 1 bit reset input *rst_comp* and an 8 bits output *out_comp*. In Figure 4, *a*, *b*, *c* and *d* represent the four candidates for Equation 1. In the 1st level of the design, *a* is compared with *b* and *c* is compared with *d*, using two of the three comparators. Each comparator finds the maximum of the two numbers and provides the result to the third comparator in the second level. Thus the third comparator finds the maximum of all the four input candidates. To receive the final 8 bits output (*max_out*) at the rising edge of the clock, a D flip flop with a clear input is inferred. The post place and route simulation, shows the total delay equal to 0.0146 μs, where as the time consumed by it's software equivalent was 52.32 μs. The % runtime improvement is calculated using Equation 2.

% runtime improvement

$$= \left[\frac{\frac{1}{\text{CCU time}} - \frac{1}{\text{fill\_matrix\_2 time}}}{\frac{1}{\text{fill\_matrix\_2 time}}} \right] * 100\% \qquad (2)$$

$$= \left[\frac{\frac{1}{0.0146*10^{-6}} - \frac{1}{52.32*10^{-6}}}{\frac{1}{52.32*10^{-6}}} \right] * 100\%$$

= 3582 %

= 35.82 times.

The device used for the simulation was Xilinx Virtex2P (XC2VP30) with speed grade -6. The device utilization summary shows that 29 out of 13696 slices are used, so the design is very efficient in terms of resource utilization as well.

It is worth to mention that we expect the actual speedup achieved by our approach to be a little bit lower than the speedup we calculated (3582 %), since our profiling method does not account for the overhead in the computation time. This overhead is incurred by some inaccuracies in the gprof tool itself, in addition to the computational overhead in the operating system.

V. Conclusion

In this paper, we provided a novel approach to improve the performance of the S-W algorithm, using partially cus-

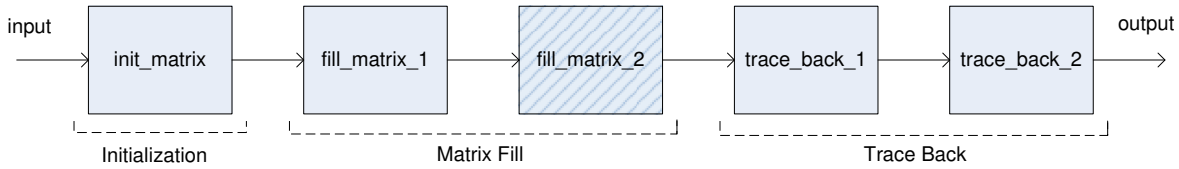

Fig. 3. Functional description of the software implementation of the S-W algorithm

TABLE II
PROFILING RESULTS FOR THE SOFTWARE IMPLEMENTATION OF THE S-W ALGORITHM

Function Name	No. of Calls	No. of Clock Ticks	No. of Clock Cycles	Total Time (*ms*)	% Time
init_matrix	100	71944	2302208	0.718	9.93
fill_matrix_1	100	32392	1036544	0.323	4.47
fill_matrix_2	**4800**	**524040**	**16769280**	**5.23**	**72.33**
trace_back_1	100	31232	999424	0.312	4.31
trace_back_2	500	64944	2078208	0.648	8.96

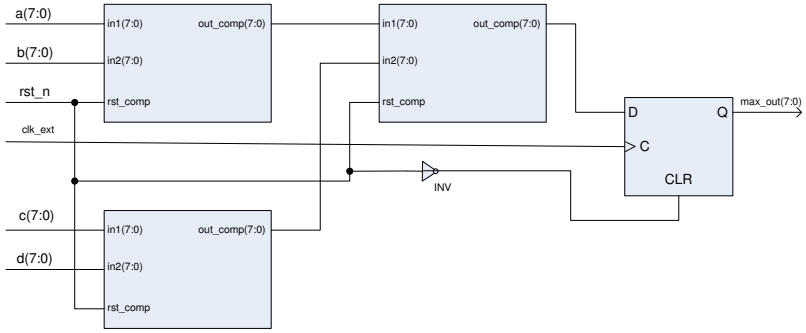

Fig. 4. RTL schematic of the custom computing unit for the function fill_matrix_2

tom hardware. We provided the functional description and the profiling results of a software implementation of the S-W algorithm. The profiling results given in Table II, show that the fill_matrix_2 is the most appropriate function to be implemented in hardware, as it consumes 72.33 % of the total execution time. We described the fill_matrix_2 function in VHDL and the post place and route simulation shows that the hardware implementation is 35.82 times faster than it's equivalent software implementation.

REFERENCES

[1] R. Giegerich. A systematic approach to dynamic programming in bioinformatics. *Bioinformatics*, vol. 16:665–677, 2000.

[2] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Jounal of Molecular Biology*, vol. 147:195–197, 1981.

[3] D.J. Lipman and W.R. Pearson. Rapid and sensitive protein simlarity searches. *Science*, 227:1435–1441, 1985.

[4] Gish W. Miller W. Myers E. W. Altschul, S. F. and D. J. Lipman. A basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[5] L. Hasan, Z. Al-Ars and S. Vassiliadis. Hardware Acceleration of Sequence Alignment Algorithms - An Overview, Proceedings of International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'07): 96–101, September 2 – 5, 2007, Rabat (Morocco).

[6] H. Y. Liao, M. L. Yin and Y. Cheng. A Parallel Implementation of the Smith-Waterman Algorithm for Massive Sequences Searching. In Proc. 26th Annual International Conference of the IEEE EMBS, San Francisco, CA, USA. September 1 – 5, 2004.

[7] Brian Hang Wai Yang. A parallel implementation of smith-waterman sequence comparison algorithm. Technical report, Biochemistry department, Stanford University, December 6, 2002.

[8] J. Chiang, M. Studniberg, J. Shaw, S. Seto and K. Truong. Hardware Accelerator for Genomic Sequence Alignment. Proceedings of the 28th IEEE EMBS Annual International Conference New York City, USA, Aug 30 – Sept 3, 2006.

[9] M. Borah R. S. Bajwa S. Hannenhalli M. J. Irwin. A SIMD Solution to the Sequence Comparison Problem on the MGAP. Proc. Int. Conf. on Application Specific Array Processors, 1994.

[10] Daniel P. Lopresti. Rapid implementation of a genetic sequence comparator using field programmable logic arrays. Conference on Advanced Research in VLSI: 138–152, 1991.

[11] A Schroder et. al. Bio-Sequence Database Scanning on a GPU. HICOMB, 2006.