ACCURATE PROFILING AND ACCELERATION EVALUATION OF THE SMITH-WATERMAN ALGORITHM USING THE MOLEN PLATFORM

Laiq Hasan
Computer Engineering Laboratory, Delft University of Technology (TU Delft)
Mekelweg 4, 2628 CD Delft, The Netherlands
L.HASAN@EWI.TUDELFT.NL

Zaid Al-Ars
Computer Engineering Laboratory, Delft University of Technology (TU Delft)
Mekelweg 4, 2628 CD Delft, The Netherlands

ABSTRACT

In this paper, we present an accurate method to evaluate the amount of acceleration gained by the hardware implementation of the Smith-Waterman algorithm. This is done using the MOLEN Processor Prototype (MOLEN platform), where algorithms can be executed both as software or as reconfigurable hardware. By profiling the algorithm, we identify a specific function that consumes 78% of the total runtime. Implementing this function in hardware results in a speedup of 2.16 in comparison with a software-only implementation. Since the hardware footprint needed for this implementation is rather small, this speedup is highly efficient in terms of resource utilization.

KEYWORDS

Bioinformatics, Sequence Alignment, Smith-Waterman Algorithm, Hardware Accelerators, FPGAs, MOLEN platform.

1. INTRODUCTION

Smith-Waterman (S-W) is the most accurate sequence alignment algorithm available, but it is also the most expensive computationally, particularly for long sequences (Smith T. F. & Waterman M. S. 1981). Faster algorithms like FASTA (Pearson W. R. & Lipman D. J. 1985) and BLAST (Altschul S. F. et al 1990) are available, but they achieve high speed at the cost of reduced accuracy. Thus it is highly desirable to accelerate the S-W algorithm in hardware.

Various approaches have been adopted to accelerate the S-W algorithm by implementing either the whole algorithm or some part of it in hardware and compare the performance with the software-only implementation (Chiang J. et al 2006), (Borah M. et al 1994), (Schroder A. et al 2006), (Blas A. Di. et al 2005), (Laiq Hasan & Zaid Al-Ars 2007), (Yamaguchi Y. et al 2002). The acceleration achieved by these methods is not measured in any standard way, which makes it difficult to compare the different approaches published in the literature (Hasan L. et al 2007). In the previous reported similar work the software version is usually implemented on a general purpose processor, whereas the hardware accelerated version is implemented on a different platform, such as FPGA. This leads to a rather less reliable acceleration evaluation.

In this paper, we propose an accurate speedup measurement method that is independent of the specific implementation. This is done by comparing both the software-only version and the hardware accelerated version of the S-W algorithm on the same platform in order to achieve an accurate profiling and acceleration evaluation. The paper is organized as follows: Section 2 provides a brief description of the S-W algorithm. Section 3 discusses conventional acceleration evaluation, done using two different platforms. Section 4

presents the way to perform an accurate speedup analysis of the software-only version of the S-W algorithm compared with its hardware accelerated version. Section 5 gives a brief conclusion.

2. DESCRIPTION OF THE S-W ALGORITHM

Based on *dynamic programming (DP)* (Giegerich R. 2000), the S-W algorithm (Smith T. F. & Waterman M. S. 1981) is a method used 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:

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

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 the following 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.

Figure 1 gives a block diagram representation of a pure software implementation of the S-W algorithm, where

- 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.
- The *trace_back_2* function keeps track of the direction of the trace back.

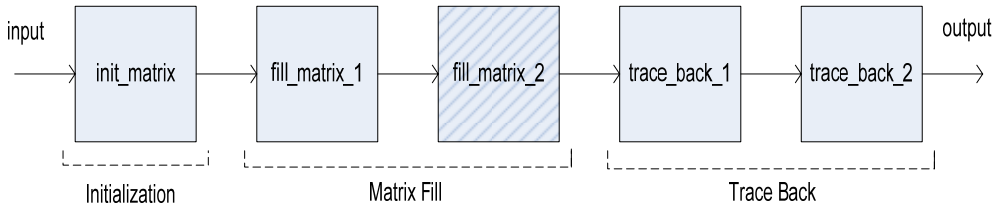

Figure 1. Functional description of a software implementation of the S-W algorithm

3. CONVENTIONAL ACCELERATION EVALUATION

Acceleration evaluation of the S-W algorithm is done by comparing the performance of a software-only version and a hardware-accelerated version of the algorithm. Conventionally, the software-only versions and

hardware-accelerated versions of the S-W algorithm are usually implemented on different platforms, as discussed later in this section. Table 1 gives examples of conventional speedup evaluation.

Table 1. Examples of conventional speedup evaluation of the S-W algorithm

Reference	Platform used for hardware implementation	Compared with implementation on	Reported speedup
(Borah M. et al 1994)	MGAP	SPLASH	5×
(Schroder A. et al 2006)	GPU	Pentium-IV, 3.0 GHz	16×
(Blas A. Di. et al 2005)	Kestrel	Ultra SPARC-II, 500 MHz	17×
(Laiq Hasan & Zaid Al-Ars 2007)	FPGA	Pentium-IV, 3.2 GHz	36×
(Steve Margerm 2006)	FPGA	AMD Opteron processors	64×
(Oliver T. et al 2005)	FPGA	Pentium-IV, 1.6 GHz	170×
(Chiang J. et al 2006)	FPGA	Nios II soft microprocessor	287×
(Yamaguchi Y. et al 2002)	FPGA	Pentium-III, 1.0 GHz	330×
(Yang B. H. W. 2002)	VLSI	Not compared	---

In (Borah M. et al 1994) an implementation of the S-W algorithm is described on a general purpose fine-grained architecture, the *Micro Grained Array Processor (MGAP)*. The authors of (Borah M. et al 1994) show that their implementation is about 5 times faster than the rapid implementation of a genetic sequence comparator using field programmable logic arrays (Daniel P. Lopresti 1991). Showing thereby that massively parallel processor arrays, like the MGAP, possess the capability to solve computationally intensive problems in Molecular Biology efficiently and inexpensively.

In (Schroder A. et al 2006), it has been demonstrated that the streaming architecture of the *Graphics Processing Units (GPUs)* can be efficiently used for biological sequence database scanning. GPUs are single-chip processors, used primarily for computing 3D functions, but is also a good match for bioinformatics applications (e.g. S-W algorithm for sequence alignment). 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 processor.

The authors of (Blas A. Di. et al 2005) implemented the S-W algorithm on the Kestrel Parallel Processor for different query sizes. The Kestrel Parallel Processor is a single-board coprocessor with a 512-element linear array of 8-bit, SIMD processing elements (Blas A. Di. et al 2005). The performance was compared with the implementation on a 500 MHz, Ultra SPARC-II. The relative speedup for a query size of 100 is given in Table 1. The other query sizes considered were 250 and 500. The speedup achieved for the query size of 250 was 49×, whereas that for the query size of 500 was 99×.

In (Laiq Hasan & Zaid Al-Ars 2007), S-W algorithm has been divided into a number of functions, and then the time complexity of each function is measured (an activity commonly referred to as *code profiling*). A software-only implementation of the S-W algorithm is profiled on Pentium-IV, 3.2 GHz processor, using the GNU profiler. The profiling results identify the most time consuming function. This function is then designed in VHDL. The processing run time of a software-only implementation on Pentium-IV, 3.2 GHz processor and hardware implementation on a Virtex II Pro FPGA are compared to evaluate the % runtime improvement. The results show that the hardware implementation is 35.82 times faster than its equivalent software-only implementation.

In (Steve Margerm 2006), the performance of the S-W algorithm has been increased substantially by using run time reconfiguration. The percentage of time spent on calculating the elements of $H_{i,j}$ matrix was cut by nearly a third and the absolute time spent on the algorithm dropped from 6,461 seconds to a little over 100 seconds, approximately 64 times faster than the equivalent software-only implementation on AMD Opteron processors.

In (Oliver T. et al 2005), the authors present a new approach to bio-sequence database scanning using re-configurable FPGA-based hardware platforms to gain high performance at low cost. Their FPGA implementation achieves a speedup of approximately 170, as compared to a Pentium-IV, 1.6 GHz processor.

In (Chiang J. et al 2006), the authors studied the improvement of computational processing time of the S-W algorithm using *custom instructions (CIs)* on an FPGA board. This was done by first writing the S-W algorithm in pure software and then replacing the portion which was the most computationally intensive with an FPGA custom instruction. Particularly, they designed CIs on an Altera Nios II integrated development environment. The Nios II soft microprocessor was instantiated on an FPGA to allow rapid prototyping of new designs. Finally, they compared the processing runtime between the "pure software" and the "hardware acceleration" versions to calculate the percentage of runtime improvement. The results showed that the hardware accelerated algorithm improved the processing runtime by an average of 287%. Thus using FPGA CIs is a promising direction for further research in improving genomic sequence searching.

In (Yamaguchi Y. et al 2002), an approach to realize high speed sequence alignment using run-time reconfiguration is proposed. With this approach, it is demonstrated that high performance can be achieved using off-the-shelf FPGA boards. The performance is almost comparable with dedicated hardware systems. The time for comparing a query sequence of 2048 elements with a database sequence of 64 million elements by the S-W algorithm is about 34 sec, which is about 330 times faster than a desktop computer with a Pentium-III, 1.0 GHz processor.

In (Yang B. H. W. 2002), 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. The results show that the design achieves a performance of 976 Kilo Cell Updates Per Second (KCUPS), but is not compared with any reference design.

It is evident from Table 1, that no standard comparison approach is adapted. That is why, we can only look into each implementation on individual basis to see how much improvement is achieved in comparison with the reference provided for each implementation. In contrast, we propose to adapt a more accurate acceleration evaluation approach, as discussed below in Section 4. The results obtained from our implementation achieve a speedup of 2.16 in comparison with a software only implementation.

4. ACCURATE ACCELERATION EVALUATION

Implementing both software-only as well as hardware accelerated versions of the S-W algorithm on the same platform leads to an accurate acceleration evaluation. We have used the MOLEN platform for this purpose, since it contains both a general purpose processor in addition to a reconfigurable hardware module.

4.1 The MOLEN platform

Figure 2 shows the block diagram representation for the MOLEN platform (Vassiliadis S. et al 2004). The first block at the top indicates that either the software-only version or hardware accelerated version of the algorithm arrives as input to the arbiter. For every function in the algorithm, the arbiter decides whether to send it to the core processor or the reconfigurable processor. The arbiter does this by using specialized instructions for calling the hardware.

The core processor is the IBM Power PC, which is built in with the Virtex II Pro FPGA. There can be upto four such Power PCs in a Virtex II Pro FPGA. The core processor is used for implementing the software portion of the application.

The reconfigurable processor is the processor used for implementing the hardware portion of the application. It has two parts. The main part is the reconfigurable microcode unit, which is responsible for the entire operation of the reconfigurable processor and the *Custom Computing Unit (CCU)* design, which is application dependant. The CCU is embedded into the reconfigurable processor, using the interface given with the MOLEN platform. The reconfigurable processor utilizes the microcode unit and the CCU to improve performance of various applications. The details of the data interface between the core processor and the reconfigurable processor are given in (Vassiliadis S. et al 2004).

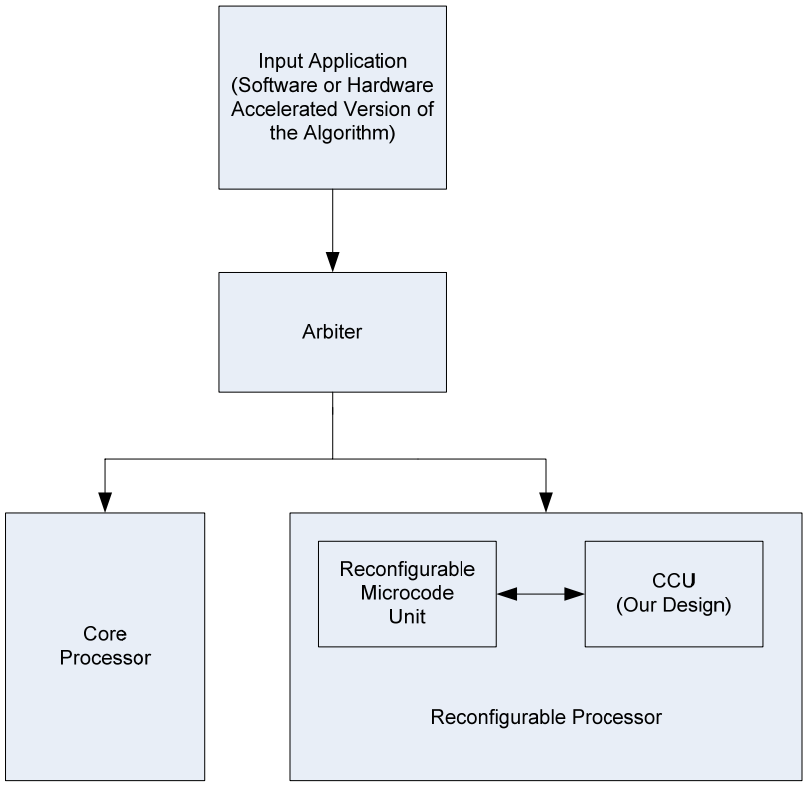

Figure 2. Block diagram representation of the MOLEN platform

4.2 Implementation of the S-W algorithm on MOLEN platform

The methodology for implementation is comprised of the following four steps.

1. Identifying the desired function in software (Code profile).
2. Designing a CCU for the identified function.
3. Replacing the identified function by the CCU design.
4. Comparing the cycles consumed by the software-only version and the hardware accelerated version of the identified function to measure the relative speedup.

The following discussion elaborates these steps.

Figure 1 gives a block diagram representation of a software-only implementation of the S-W algorithm. We compiled this software-only implementation using MOLEN Power PC Compiler (Elena Moscu Panainte 2007). The cycles consumed by each function in the code were evaluated, using the Power PC timer instructions. The Power PC has a clock frequency of 100 MHz, so the time period for one cycle is $\frac{1}{100}\ \mu s = 0.01\ \mu s$. Thus the time consumed by each function is equivalent to the number of cycles consumed multiplied by the time period for one cycle. The overall time consumed is the summation of time consumed by all functions, which is 172 μs and the % time consumed is the ratio of the time consumed by a function to the overall time consumed. Table 2 gives the function names, number of times that each function is called, the number of clock cycles consumed by each function, the amount of time consumed by each function in micro seconds and the % time consumed by each function. In Table 2, the fill_matrix_2 is highlighted as the most time consuming function and is the right candidate to be designed in hardware as a CCU. Table 2 shows that the cycles consumed by fill_matrix_2 function for 48 calls are 13392, so the cycles consumed for 1 call will be $\frac{13392}{48} = 279$. We designed a hardware module (CCU) in a hardware description language (VHDL) to support the identified function of interest (fill_matrix_2) using the interface described for the MOLEN platform. The device used for the implementation was Xilinx Virtex-II Pro (XC2VP30) with speed grade -7.

The device utilization summary in the synthesis report showed that 29 out of 13696 slices were used, so apart from achieving the speedup, calculated later in this section, the design is very efficient in terms of resource utilization as well.

Table 2: Profiling results of the software-only version of the S-W algorithm on the core processor of the MOLEN platform (Power PC 100 MHz processor)

Function name	Number of calls	Clock cycles consumed	Time consumed (μs)	% Time consumed
init_matrix	1	753	7.53	4.33
fill_matrix_1	1	688	6.88	4.00
fill_matrix_2	**48**	**13392**	**133.92**	**78.00**
trace_back_1	1	102	1.02	0.55
trace_back_2	5	2265	22.65	13.12

After designing the desired CCU, we annotated the definition of fill_matrix_2 function with (#pragma call fpga fill_matrix_2) in the C source code. We compiled the entire annotated C code, using the MOLEN Power PC compiler. An executable file (executable.elf) thus generated was downloaded locally. The CCU design was embedded into MOLEN using the Xilinx modular design flow. The generated bit stream was downloaded into the XUP V2P prototyping board by connecting a configuration cable to the prototyping board. To see the results on the terminal the serial communication port of the board was connected to the PC COM port and a session was set at 19200 bps, 8 data bits, no parity, 1 stop bit and no flow control.

Using the Power PC timer functions, we evaluated the cycles consumed by fill_matrix_2 function in the annotated C code, which came out to be 129. The comparison between the cycles consumed by the software-only version and the hardware accelerated version of fill_matrix_2 gives the relative speedup, where the

$$Speedup = \frac{Cycles\ consumed\ in\ software}{Cycles\ consumed\ with\ hardware\ acceleration} = \frac{279}{129} = 2.16$$

This speedup is more accurate, as the software-only and hardware accelerated versions are both implemented on the same platform. To ensure an accurate measurement of the speedup, all bottlenecks have been taken care of, such that only processing time is the limiting factor. Furthermore the approach is technology independent and can also be implemented on alternative available FPGAs, such as Virtex IV and Virtex V.

5. CONCLUSION

In this paper, we implemented both software-only and hardware-accelerated versions of the S-W algorithm, using the MOLEN platform and evaluated the cycles consumed by the most computationally intensive function within the algorithm in both cases. This approach gives us an accurate number of cycles consumed in both cases and helps in achieving a more accurate speedup evaluation than the already reported work in the literature, as the software-only and hardware-accelerated versions are both implemented using the same platform. The results demonstrate that our implementation approach achieves a speedup of 2.16, as compared to a software-only implementation.

ACKNOWLEDGEMENT

This work is jointly supported by Computer Engineering Laboratory, TU Delft, The Netherlands and Higher Education Commission of Pakistan.

REFERENCES

Altschul S. F. et al, 1990. A Basic Local Alignment Search Tool. *In Journal of Molecular Biology*, vol. 215, pp 403-410.

Blas A. Di. et al, 2005. The UCSC Kestrel Parallel Processor. *In IEEE Transactions on Parallel and Distributed Systems*, vol. 16(1), pp 80–92.

Borah M. et al, 1994. A SIMD Solution to the Sequence Comparison Problem on the MGAP. *Proceedings of the International Conference on Application Specific Array Processors*, San Francisco, California, USA.

Chiang J. et al, Aug 30-Sept 3, 2006. Hardware Accelerator for Genomic Sequence Alignment. *Proceedings of the 28th IEEE EMBS Annual International Conference*, New York City, USA.

Daniel P. Lopresti, 1991. Rapid Implementation of a Genetic Sequence Comparator Using Field Programmable Logic Arrays. *Conference on Advanced Research in VLSI*, pp 138-152, USA.

Elena Moscu Panainte, 2007. The Molen Compiler for Reconfigurable Architectures. *Ph.D. Thesis*, *Computer Engineering Laboratory*, TU Delft, The Netherlands.

Giegerich R., 2000. A systematic approach to dynamic programming in bioinformatics. *Bioinformatics*, vol. 16, pp 665-677.

Hasan L. et al, September 2–5, 2007. Hardware Acceleration of Sequence Alignment Algorithms - An Overview. *Proceedings of International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'07)*, pp 96-101, Rabat, Morocco.

Laiq Hasan and Zaid Al-Ars, November 29–30, 2007. Performance Improvement of the Smith-Waterman Algorithm. *Annual Workshop on Circuits, Systems and Signal Processing (ProRISC)*, Veldhoven, The Netherlands.

Mustafa Gok and Caglar Yilmaz, 2006. Efficient Cell Designs for Systolic Smith-Waterman Implementation. *Proceedings of the 16th International Conference on Field Programmable Logic and Applications (FPL)*, Meliá Madrid Princesa, Madrid, SPAIN.

Oliver T. et al, February 20-22, 2005. Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs. *FPGA'05*, Monterey, California, USA.

Pearson W. R. and Lipman D. J., 1985. Rapid and Sensitive Protein Similarity Searches. *In Science*, vol. 227, pp 1435-1441.

Schroder A. et al, 2006. Bio-Sequence Database Scanning on a GPU. *Proceedings of the Fifth IEEE International Workshop on High Performance Computational Biology (HICOMB)*, Rhodes Island, Greece.

Smith T. F. and Waterman M. S., 1981. Identification of common molecular subsequences. *In Journal of Molecular Biology*, vol. 147, pp 195-197.

Steve Margerm, Cray Inc, February 7, 2006. Reconfigurable Computing in Real-World Applications. *FPGA and Structured ASIC Journal (www.fpgajournal.com)*.

Vassiliadis S. et al, November 2004. The Molen Polymorphic Processor. *In IEEE Transactions on Computers*, vol. 53(11), pp 1363-1375.

Yamaguchi Y. et al, 2002. High Speed Homology Search Using Run-Time Reconfiguration. *Proceedings of the 12th International Conference on Field Programmable Logic and Application (FPL)*, Montpellier (La Grande-Motte) – France.

Yang B. H. W., December 6, 2002. A parallel Implementation of Smith-Waterman Sequence Comparison Algorithm. *Technical Report, Biochemistry department, Stanford University*, USA.