Architectural Optimizations for High Performance and Energy Efficient Smith-Waterman Implementation on FPGAs using OpenCL

Lorenzo Di Tucci[1], Kenneth O'Brien[2], Michaela Blott[2], Marco D. Santambrogio[1]
[1]Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Milano, Italy,
[2]Xilinx Research Labs, Dublin, Ireland,
lorenzo.ditucci@mail.polimi.it
marco.santambrogio@polimi.it
{*kenneth.o'brien,michaela.blott*}@xilinx.com

***Abstract*—Smith-Waterman is a dynamic programming algorithm that plays a key role in the modern genomics pipeline as it is guaranteed to find the optimal local alignment between two strings of data. The state of the art presents many hardware acceleration solutions that have been implemented in order to exploit the high degree of parallelism available in this algorithm. The majority of these implementations use heuristics to increase the performance of the system at the expense of the accuracy of the result. In this work, we present an implementation of the pure version of the algorithm. We include the key architectural optimizations to achieve highest possible performance for a given platform and leverage the Berkeley roofline model to track the performance and guide the optimizations. To achieve scalability, our custom design comprises of systolic arrays, data compression features and shift registers, while a custom port mapping strategy aims to maximize performance. Our designs are built leveraging an OpenCL-based design entry, namely Xilinx SDAccel, in conjunction with a Xilinx Virtex 7 and Kintex Ultrascale platform.**

Our final design achieves a performance of 42.47 GCUPS (giga cell updates per second) with an energy efficiency of 1.6988 GCUPS/W. This represents an improvement of 1.72x in performance and energy efficiency over previously published FPGA implementations and 8.49x better in energy efficiency over comparable GPU implementations.

I. Introduction

Smith-Waterman [1] is a well known sequence alignment algorithm for identifying relationships between strings of genetic data. Given two input strings $\mathcal{A}$, a database of length M, and $\mathcal{B}$, a query of length N, Smith-Waterman's goal is to find regions of similarity between subsequences of all possible lengths. An $M \times N$ matrix $\mathcal{H}$ is constructed to hold similarity scores for each subsequence comparison. The first row and columns of $\mathcal{H}$ are initialized to zero. Each element of the matrix ($\mathcal{H}_{i,j}$) represents the similarity of strings $\mathcal{A}$ from index 1 to i and $\mathcal{B}$ from 1 to j and is dependent on previous values to the left, diagonal and above as formally described in Equation (1).

$$\mathcal{H}_{i,j} = max \begin{Bmatrix} \mathcal{H}(i-1,j-1) + s(a_i,b_j) \\ max_{k\geq 1}\{\mathcal{H}(i-k,j) + W_k\} \\ max_{l\geq 1}\{\mathcal{H}(i,j-l) + W_l\} \\ 0 \end{Bmatrix} \qquad (1)$$

$$1 \leq i \leq M, 1 \leq j \leq N$$

The function $s(a_i, b_j)$ calculates similarity as shown in Equation (2). W_i denotes the gap-scoring scheme which are constants in our implementation.

$$s(a_i,b_j) = \begin{Bmatrix} 2 & if\ a_i\ and\ b_j\ match \\ -1 & Otherwise \end{Bmatrix} \qquad (2)$$

A second matrix $\mathcal{T}$ is formed to hold the direction path through matrix $\mathcal{H}$ to produce an alignment. The optimal alignment is produced by starting from the position of the maximum score in matrix $\mathcal{H}$ and following the directions found at the same index in $\mathcal{T}$, until a zero is found.

The importance of sequence alignment results, coupled with the explosion of available genetic data and the computational complexity of sequence alignment algorithms, ($\mathcal{O}(MN)$ in the case of Smith-Waterman) has driven a need for a high performance solution. The performance of a Smith-Waterman implementation is measured in billions of cell updates per second of this matrix (GCUPS). Additionally, due to the rising cost of powering and cooling data center infrastructure, energy efficiency, measured in performance per watt (GCUPS/W) has become a tier 1 metric for evaluating any data center solution. We therefore analyze our architecture for both performance and energy efficiency as key figures of merit.

The use of heuristics has been particularly popular due to the high computational requirements of this algorithm, in particular to reduce the execution time of the calculation of the similarity and traceback matrices. However, this speedup comes at a cost. The sensitivity of the algorithm is reduced and the algorithm might miss a similarity if the two sequences considered are distantly related [2]. Two commonly used tools for pairwise sequence alignment that leverage heuristic algorithms [3] are FASTA [4] and BLAST [5].

Smith-Waterman implementations vary by the heuristics applied, the supported input data sizes, the number of queries processed in parallel in a batch format and differences in which stage of the algorithm is accelerated. These variations make evaluating existing implementations difficult. Our implementation performs all stages of the Smith-Waterman algorithm on the FPGA, with the exception of the traceback stage which is performed on the host CPU. Other implementations accelerate a subset of the algorithm e.g. [6] which accelerates

the calculation of a max score and directions contributing to its generation.

This paper presents our work on implementing a high performance, and highly energy efficient, *pure* Smith-Waterman implementation on Xilinx Virtex 7 and Kintex Ultrascale FPGAs using the Xilinx SDAccel toolflow. By *pure* we mean that our algorithm is not based on any heuristic method, hence it is as described by the original inventors [1]. We provide a performance and energy evaluation as well as a theoretical analysis to track optimizations and to demonstrate more general platform advantages FPGAs bring to Smith-Waterman using the Berkeley roofline model [7].

The rest of the document is organized as follows: Section II introduces the state of the art of Smith-Waterman implementation, summarizing best published performance results for single-FPGA implementation comparing a single pair query-database. Section III describes the theoretical analysis with roofline models. The hardware architecture with key optimization techniques are presented in Section IV. Section V explains the experimental environment and summarizes the measured results. Finally, in Section VI we conclude the work, briefly describe our experience with the OpenCL design environment and give some insights on future works.

II. State of the Art

Previous approaches to acceleration have included applying data level parallelism by way of SIMD instructions on both x86 and Cell architectures [8], [9], whereby with 3 and 33 (GCUPS) respectively, the performance on these implementations are significantly slower in comparison to accelerators. With the emergence of dedicated hardware accelerators, several GPU solutions have arrived [10]–[16]. Most notably Blazewicz [15] demonstrated a pure Smith-Waterman implementation on a single Nvidia GTX280 GPU, yielding a performance of $\sim$5 GCUPS for sequences of length 51, lessening to $\sim$4 GCUPS for the longest testing sequence length of 459. Liu [17] demonstrates a hybrid CPU-GPU implementation called CUDASW++ 3.0 attaining a performance of 83.3 GCUPS on an Nvidia GTX680 GPU aided by a CPU. It is to our knowledge the highest performing implementation of the Smith-Waterman algorithm to date. However, the power envelope of the specified GPU is 195W, significantly higher than our FPGA devices at 25W [18], thereby we outperform this implementation on efficiency significantly, as will be shown later on. Beyond CPU and GPU, other accelerator devices have been targeted. Intel's Xeon Phi has achieved 58.8GCUPS on a single Phi 5110P [19] using no more than 225W. This is more moderate in performance than the above quoted GPU numbers, while consuming similar amounts of power. FPGAs are also increasingly deployed as accelerators due to their low power consumption, high integer compute performance and improvements in programmability. The most advanced FPGA implementation to date has reached 24.7 GCUPS [20] for a single device in a 25W power envelope, significantly lower than the 225W envelope of HPC-grade GPUs. All above studies reported performance values for multiple queries against a single database, with performance improving with greater numbers of queries due to more efficient utilization of the hardware resources. Despite the overhead of the OpenCL-based design environment, we believe that our implementation achieves the highest performance of any FPGA accelerator with its 42GCUPS, while approaching GPU levels of performance. For CPUs, we provide a 14.15x speedup over best reported x86 numbers, and 1.28x over best reported Cell number. Furthermore, we significantly outperform all previous state of the art on energy efficiency.

TABLE I
Performance of pure Smith-Waterman algorithms in the state of the art

Platform	Performance [GCUPS]	Efficiency [GCUPS/W]
Altera Stratix V on Nallatech PCIe-385 [20]	24.7	0.988
Xtreme Data XD1000 [21]	25.6	0.43
Intel Xeon Phi 5110P [19]	58.8	0.2613
Nvidia Tesla K20 [16]	45	0.2
Xtreme Data XD2000i [22]	9	0.15
Nvidia GeForce GTX 295 [15]	30	0.104

Table I summarises published state of the art implementations. The first column shows the platform, the second column reports performance whereby the third column shows the ratio of performance over power consumption in terms of GCUPS/W. We assumed thermal design power (TDP) as worst case power consumption as no other numbers were available. We realize that actual consumption might be well below.

III. Algorithm Analysis and Performance Prediction

In order to predict performance and track optimizations on different platforms, we leveraged the roofline model [7]. This graphical model exploits the so called "bound and bottleneck" analysis which aims to expose simple bottlenecks causing suboptimal performance of an application. The rooflines model the relationship between offchip memory bandwidth and the performance of the processor and represents the theoretical peak performance for a given platform. This is extremely useful for comparing different hardware platforms on a spectrum of applications. For performance prediction we need to correlate the *Operational Intensity* of an application with the rooflines. In the following subsection, we explain the basics behind the roofline. Then we conduct a static code analysis to determine the operational intensity of the algorithm. Finally in the last subsection, we correlate the operational intensity with the rooflines of our chosen FPGA platforms to predict the corresponding performance.

A. The Roofline Model

As previously mentioned, rooflines are a graphical tool to model the theoretical maximum performance of a platform taking available offchip memory bandwidth and theoretically possible peak performance into account. The performance is given as a function of the operational intensity of an application, which in essence represents the ratio between overall

instructions to be executed, and total memory footprint in Bytes. As such, the 2-dimensional chart of the roofline model, depicts the relation between *Operational Intensity*, memory performance and compute performance in units of floating point operations (flops) or any other user-defined operation (ops). Figure 1 shows an example of a roofline model. The graph is in log-log scale. The x-axis represents the operational intensity in terms of operation per each byte of data that is moved; the y-axis expresses the performance, by means of giga cell update per second. In the chart it is possible to identify two areas, the one below the slope line on the left is called the *memory bound* area, while the one on the right, under the horizontal line, is the *compute bound* area. These two lines represent respectively the peak memory bandwidth and the peak compute performance for a given platform. The latter is the absolute hardware limit of the platform, so no application can ever produce performance above that line. The roofline is computed using the following formula

$$GOPS = min\left(\binom{Peak\ Compute}{Performance}, \left(\binom{Peak\ Memory}{Bandwidth} \cdot \binom{Operational}{Intensity}\right)\right)$$

Examples of work exploiting the roofline model to predict system performance can be found here [23]. A semi-automated toolflow for implementing this approach on heterogeneous platforms has been presented in [24].

B. Algorithm Analysis

We performed a static code analysis in order to understand the complexity of the code, and to derive what kind of performance we could expect from our target device. For our kernel, we counted only the operations considered *relevant*. We identified three categories of operations, namely *indexing*, *arithmetic* and *comparison*.

TABLE II
STATIC CODE ANALYSIS - OPERATIONS

Work [Operations]	Theoretical	Example 256 x 65536
Indexing	$11N^2 + 11NM - 6N$	185M
Comparison	$6N^2 + 6NM - 5N$	101M
Arithmetic	$15N^2 + 15NM - 6N + 8M + 2$	253M
Total	$32N^2 + 32NM - 17N + 8M + 2$	539M

Table II, shows the amount of operations for our Smith-Waterman. The first column indicates the category of the operation, the second one, shows the theoretical formula for obtaining the amount of operations for the category and in the third column, we provide an example with a dataset where the query size (N) is equal to 256, while the database (M) is 65536 nucleotides. The memory traffic, is expressed in Table III, whereby the organization is as described above.

With this data, we can derive the operations per byte, or operation intensity with the formula as expressed in Equation (3).

$$O.I. = \frac{32N^2 + 32NM - 17N + 8M + 2}{65N + 65M - 64} \quad (3)$$

For the particular example given above, the operational intensity is 126 Operations per Byte, which we can leverage in the following subsections to predict platform performance.

TABLE III
STATIC CODE ANALYSIS - MEMORY

Work [Operations]	Theoretical	Example 256 x 65536
Data In	$N + M$	65K
Data Out	$64(N + M - 1)$	4.2M
Total	$65N + 65M - 64$	4.3M

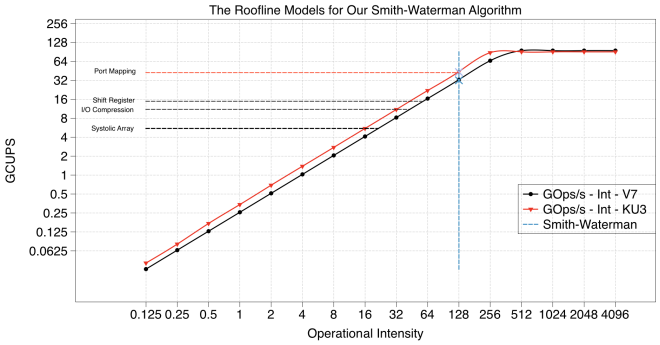

Fig. 1. The roofline model for our target boards showing the expected performance of our Smith-Waterman algorithm implementation and the achieved performance per each optimization applied

C. Performance Prediction

Figure 1 shows the roofline model for our target boards (ADM-PCIE-KU3 and ADM-PCIE-7V3) correlated with the vertical line that represents the operational intensity of our implementation. The 4 horizontal lines represent the performance achieved with each optimization, that will be explained later in this document. We find that, with an operational intensity of 126 Op/B, our kernel is still memory bound for both devices, although at the high end of the spectrum. More specifically, we would expect performance in the order of 1024 GOPS for the FPGA board based on the Virtex-7, while we would expect 1310.72 GOPS for the board based on the Kintex Ultrascale devices whereby available memory bandwidth is clearly the differentiating factor. With 32 operations per cell update, this translates to an expected performance of 32 GCUPS for Virtex-7 and 43.31 GCUPS for Kintex Ultrascale platforms respectively. The operational intensity described here has not been the first one we obtained. The kernel we started with, had an operational intensity of roughly 2 Operations per Byte. Here the roofline model played a key role in suggesting us to modify the kernel code in order to increase the amount of operations performed while at the same time trying to reduce the memory traffic. Directed by the model, we restructured our design obtaining the operational intensity described so far.

IV. Hardware Implementation

This section describes the hardware architecture and optimizations applied. As discussed, we have not exploited heuristics and implemented a *pure* version of the algorithm to maintain optimal results. We apply each optimization incrementally from the initial design until we reach our most optimized implementation.

A. Column-Based Systolic Array Architecture

We found that parallelism is best exploited in the calculation of the anti-diagonal of the similarity matrix $\mathcal{H}$.

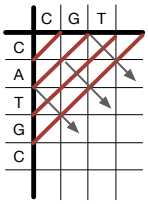

Fig. 2. Parallelism of the Similarity Matrix Calculation ($\mathcal{H}$)

The elements of the matrix are calculated in a wavefront pattern beginning at the upper left corner. An example of the wavefront pattern can be seen in Figure 2. Each element that lies on an anti-diagonal is dependent only on elements lying on the previous two anti-diagonals. For this reason, we decided to exploit a systolic array, that is column-based rather than diagonal-based. Furthermore, we decided to buffer out corners, in order to simplify corner cases during the Similarity ($\mathcal{H}$) and Traceback ($\mathcal{T}$) matrix calculation. This is essential because the additional logic to compute the corner cases would result in a slow down of the overall system, therefore we consider these corners separately. Padding of the corner cases has the advantage that we are always calculating the same fixed amount of elements. Specifically, our implementation is always computing N elements, whereby N is the size of the query string. Using a column-based systolic array and padding of corners have been key design decision in order to achieve the calculation of one anti-diagonal per clock cycle.

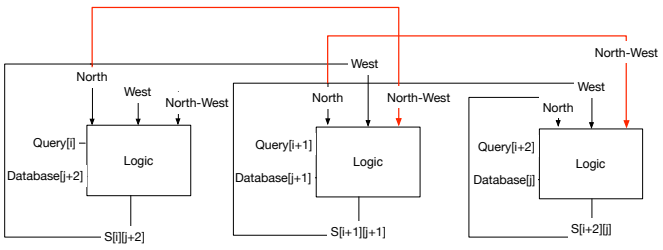

Fig. 3. The Systolic cell architecture implemented for our Smith-Waterman algorithm highlighting the inputs and outputs for each processing element

As seen in Figure 3, each systolic cell passes its output to the appropriate input of the subsequent systolic cell. In particular, the output of each processing element at time *t*, will be the North input of the same processing element and the West input of the next processing element at time *t+1*. Furthermore the North input at time *t+1* becomes the North-West input for the next processing element at time *t+2* (highlighted in red in Fig. 3). Thanks to this optimization, we have been able to get performance in the order of 5.5 GCUPS, however, thanks to the information given by our roofline model, we know there is room for more optimization.

B. Input/Output Compression

The systolic array allowed us to increase the exploited parallelism on the board, but our roofline model suggested us that our implementation is memory bound. Therefore, as the memory bandwidth of our platform is limited, we had to find another way to increase the performance of the system. As the input data for Smith-Waterman is constructed from only four characters, $\{A, C, G, T\}$, compression of data in memory per character should help speeding up the algorithm. This domain knowledge and FPGA support for arbitrary width data types, facilitated a 2-bit representation of our data, greatly reducing our memory bandwidth demand. Within SDAccel, the maximum native word size is currently 512 bits. This data type matches the memory interface size to an SDAccel kernel, meaning we can read 256 nucleotides per cycle. The same data compression has been applied to the output of our algorithm: the direction matrix as there are only 4 possible directions: North, North-west, Center and West. Finally, we used this representation internally in the algorithm, it is directly handled as unaltered from the input interfaces considerably reducing the implemented hardware logic. Adding this optimization to our architecture allowed us to double system performance beyond 11 GCUPS. Compression also applies to other inputs: for example, proteins, composed of 20 characters, need at most 5 bits.

C. Shift Register

The previous optimizations allowed us to speed up our implementation considerably and a developer may think the maximum parallelism that the FPGA can offer has been reached, however thanks to the roofline model, we know that we can apply further optimizations to increase the performance of our kernel. The third optimization we applied, revealed itself necessary from the fact that in order to compute one anti-diagonal each clock cycle, we need to have all the data local to the kernel. However, the database string can be very large as a DNA genome can be composed of billions of characters [25] and the size of the local memory of the FPGA is limited. Furthermore, we noticed that by increasing the amount of data that the kernel was processing the performance of the overall system increased as we can exploit more parallelism from the device. For this reason we decided to increase the size of the database given to the FPGA. This presents a problem as the buffers containing the query and the database are partitioned. This means that they are implemented as logic in the circuit, rather than in BRAM. The compiler will connect each logic element implemented with all the possible paths that could request access. The reader may understand that this will drastically increase the complexity of the logic circuit through introduction of a large number of multiplexers. The solution we implemented for this is a *shift register*. At the beginning of the computation the kernel will copy the first N elements (with N the size of the query sequence) into the shift register and each clock cycle, it shifts in a new database character so that the new anti-diagonal can be computed. Thanks to this optimization, the size of the database-partition is fixed to N

and the FPGA is able to manage database of any dimension. Hence, we have been able to send more data to the FPGA, getting performance of approximately 15 GCUPS.

D. Port Mapping

Our previous implementation reaches approximately half of the performance estimated by our roofline model for that target board. The reason behind this, is that our model of the Virtex-7 based board does not consider that the memory bandwidth of the board is half due to reading and writing over the single memory channel. After this considerations, we decided to move the design to a platform with more memory bandwidth. The ADM-PCIE-KU3 board is equipped with two physical ports between the memory and the FPGA. This means that is possible to send in parallel both the inputs, or send and receive data from DRAM channels at the same time. Furthermore, SDAccel allows the user to define in which area of the global memory we store out input/output buffers and on which available port to send/receive the data. This level of customization allowed us to send the inputs in parallel to the FPGA and to receive the output in a faster way. We tested what was the best mapping of the ports in order to fully exploit the memory bandwidth and the results of this optimizations allowed us to get the performance estimated by our roofline model, in particular, we obtained 42.47 GCUPS.

V. Experimental Design

In this section we describe the experimental environment we created to test the functionality of our hardware architecture.

A. Hardware Setup

The PCIe-based boards used to test the algorithm are both made by AlphaData and are supported within the Xilinx SDAccel framework. They are the ADM-PCIE-7V3 based on the Xilinx Virtex-7 FPGA and the ADM-PCIE-KU3 based on the Xilinx Kintex Ultrascale FPGA. The host PC used is a x64 host with Red Hat Enterprise Linux 6 installed. The board communicates with the host over PCIe and the execution times of the algorithm have been measured exploiting the OpenCL *events*. As we implemented a *pure* Smith-Waterman, we decided to perform our test by pairing only one query to one database sequence.

B. The Xilinx SDAccel Framework

SDAccel is a new design tool by Xilinx for building accelerated applications on Xilinx FPGAs. The environment provided by this tool is compatible with existing environments supporting the OpenCL standard. The SDAccel design flow facilitates simulation of OpenCL kernels during development, followed by hardware synthesis of the kernel for execution on the target board. The advantage of developing for the FPGA is that the hardware architecture is not fixed and is specified by the developer through the OpenCL language. In addition to traditional OpenCL optimizations, a developer can make use of FPGA specific optimizations such as pipelining and custom memory architectures.

C. Experimental Results

All results of the various optimization steps as described in the previous section, are summarized in Table IV.

TABLE IV
Performance of our Smith-Waterman with relation to the optimization applied

Board	Optimization	Performance [GCUPS]	Efficiency [GCUPS/W]
Kintex Ultrascale	Port Mapping	42.47	1.6988
Virtex-7	Shift Register	14.84	0.5936
Virtex-7	I/O Compression	11.04	0.4416
Virtex-7	Systolic Array	5.5	0.22

The first column shows the board targeted for the particular architecture, while the second one names the optimization applied as outlined in the previous section. The third and fourth columns state our performance and energy efficiency. The former expresses the performance by means of GCUPS, while the latter shows the ratio between performance and power consumption.
All the tests performed, have been done using a query size (N) equal to 256 characters, and a database size (M) of more than 1 million characters. We benchmarked the different architectures with different database sizes and the results reported are the average for each optimization. The optimizations have been applied incrementally, we started from implementing the systolic array, and ended mapping the memory ports. From Table IV, it is observable how the systolic array implementation brought significant benefits to us. With the guidance of the roofline models, we identified memory as a key bottleneck and led us to compression of inputs and outputs. This effectively doubled the performance. Our shift register optimization, enabled us to create an infrastructure that is capable of handling databases of any size.

TABLE V
Resource Consumption of our best Smith-Waterman Implementation on the Kintex Ultrascale

Name	Total	Available	Utilization(%)
BRAM	212	2160	9
DSP	0	2760	0
Flip Flop	60541	663360	9
LUT	125996	331680	37

Finally, as indicated in the roofline, the Kintex Ultrascale platform provided us with a significant improvement as it offers much higher memory access bandwidth while the algorithm is still within the memory bound spectrum of this platform. This allowed us to outperform existing implementations in regards to ratio of performance to power consumption. Furthermore, as it is observable in Table V, our implementation is occupying a very small area on the FPGA, hence it could be possible to extend this solution by instantiating more than one kernel on the same FPGA device.

VI. Conclusions and Future Works

We have described and implemented a hardware acceleration of a pure Smith-Waterman algorithm on FPGA using an OpenCL based design environment, namely Xilinx SDAccel. This framework brought many benefits to us. Despite the high abstraction available within the framework, we have been able to develop an implementation of a Smith-Waterman algorithm with state of the art performance. We demonstrate performance speedups of 14.15x and 1.28x over x86 and Cell processor implementations, a 1.72x speedup over the most advanced FPGA implementation.

For what concerns energy efficiency, our implementation has improvements of 8.49x over the best GPU implementation that translates in the architecture with the best ratio of performance to power consumption for single query on single database of the state of the art presented in Section II

Furthermore, there have been productivity benefits. As SDAccel completely automates the final integration of the kernel into the board and the creation of the hardware infrastructure that surrounds it, we had could concentrate more on our algorithm. Finally, thanks to the roofline model, we have been able to get some insight on how to optimize our kernel, as well as information regarding what kind of performance to expect from a specific board. Note that the model provided us with important information regarding the maximum performance obtainable from a target board but didn't tell us how to apply the different optimizations.

Our implementation currently performs an alignment of one query to one database. In future, we will adapt this code to align several queries at once as this would increase the parallelism and our implementation has a low area usage. Initial work on this front has proven promising. Other improvements can be obtained by increasing the size of the query (N) as more processing elements will work in parallel, which will be part of a broader design space study in the future.

Acknowledgements This paper has been funded by Fondazione Cariplo (www.fondazionecariplo.it/en) under the project "Progetto Cariplo MORPHONE 2016-1010: A Challenges Driven Design for Effective and efficient Autonomic Mobile Computing Architectures".

References

[1] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[2] H.-Y. Liao, M.-L. Yin, and Y. Cheng, "A parallel implementation of the smith-waterman algorithm for massive sequences searching," in *Engineering in Medicine and Biology Society, 2004. IEMBS'04. 26th Annual International Conference of the IEEE*, vol. 2. IEEE, 2004, pp. 2817–2820.

[3] F. Zhang, X.-Z. Qiao, and Z.-Y. Liu, "A parallel smith-waterman algorithm based on divide and conquer," in *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*. IEEE, 2002, pp. 162–169.

[4] W. R. Pearson, "[5] rapid and sensitive sequence comparison with fastp and fasta," *Methods in enzymology*, vol. 183, pp. 63–98, 1990.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[6] A. Sirasao, E. Delaye, R. Sunkavalli, and S. Neuendorffer, "Fpga based opencl acceleration of genome sequencing software," *System*, vol. 128, no. 8.7, p. 11.

[7] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[8] M. Farrar, "Striped smith–waterman speeds database searches six times over other simd implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[9] M. S. Farrar, "Optimizing smith-waterman for the cell broadband engine," *Bioinformatics*, vol. 23, pp. 156–161, 2008.

[10] L. Ligowski and W. Rudnicki, "An efficient implementation of smith waterman algorithm on gpu using cuda, for massively parallel scanning of sequence databases," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–8.

[11] S. Manavski and G. Valle, "Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment," *BMC Bioinformatics*, vol. 9, no. Suppl 2, p. S10, 2008. [Online]. Available: http://www.biomedcentral.com/1471-2105/9/S2/S10

[12] Y. Munekawa, F. Ino, and K. Hagihara, "Design and implementation of the smith-waterman algorithm on the cuda-compatible gpu," in *BioInformatics and BioEngineering, 2008. BIBE 2008. 8th IEEE International Conference on*, Oct 2008, pp. 1–6.

[13] Y. Liu, D. Maskell, and B. Schmidt, "Cudasw++: optimizing smith-waterman sequence database searches for cuda-enabled graphics processing units," *BMC Research Notes*, vol. 2, no. 1, p. 73, 2009. [Online]. Available: http://www.biomedcentral.com/1756-0500/2/73

[14] L. Hasan, M. Kentie, and Z. Al-Ars, "Dopa: Gpu-based protein alignment using database and memory access optimizations," *BMC Research Notes*, vol. 4, no. 1, p. 261, 2011. [Online]. Available: http://www.biomedcentral.com/1756-0500/4/261

[15] J. Blazewicz, W. Frohmberg, M. Kierzynka, E. Pesch, and P. Wojciechowski, "Protein alignment algorithms with an efficient backtracking routine on multiple gpus," *BMC bioinformatics*, vol. 12, no. 1, p. 1, 2011.

[16] L. Huang, C. Wu, L. Lai, and Y. Li, "Improving the mapping of smith-waterman sequence database searches onto cuda-enabled gpus," *BioMed research international*, vol. 2015, 2015.

[17] Y. Liu, A. Wirawan, and B. Schmidt, "Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions," *BMC Bioinformatics*, vol. 14, no. 1, pp. 1–10, 2013. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-14-117

[18] A. Data. (2016) Adm-pcie-7v3 user manual. [Online]. Available: http://www.alpha-data.com/pdfs/adm-pcie-7v3%20user%20manual.pdf

[19] Y. Liu and B. Schmidt, "Swaphi: Smith-waterman protein database search on xeon phi coprocessors," in *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, June 2014, pp. 184–185.

[20] S. O. Settle, "High-performance dynamic programming on fpgas with opencl," in *Proc. IEEE High Perform. Extreme Comput. Conf.(HPEC)*, 2013, pp. 1–6.

[21] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the smith-waterman algorithm on a reconfigurable supercomputing platform," in *Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications: held in conjunction with SC07*. ACM, 2007, pp. 39–48.

[22] J. Allred, J. Coyne, W. Lynch, V. Natoli, J. Grecco, and J. Morrissette, "Smith-waterman implementation on a fsb-fpga module using the intel accelerator abstraction layer," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–4.

[23] G. Guidi, E. Reggiani, L. Di Tucci, G. Durelli, M. Blott, and M. D. Santambrogio, "On how to improve fpga-based systems design productivity via sdaccel," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 247–252.

[24] S. Muralidharan, K. O'Brien, and C. Lalanne, "A semi-automated tool flow for roofline analysis of opencl kernels on accelerators," in *First International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC'15), held in conjunction with Supercomputing 2015 http://h2rc.cse.sc.edu/2015/h2rc-p15.pdf*, 2015.

[25] Math and S. A. Center. (2016) What is the human genome and how big is it? [Online]. Available: http://www.edinformatics.com/math_science/human_genome.htm