Francesco Peverelli, Alberto Zeni, Enrico Cabri

CircFA: Circular RNA FPGA Aligner

Xilinx Open Hardware 2018

University: Politecnico Di Milano

Supervisor: Marco Domenico Santambrogio

Team number: xohw18-400

Project presentation video: `https://youtu.be/nHDAhXOTM4A`

June 30th, 2018

Contents

1

Inroduction

1.1 Abstract

Circular RNAs (circRNAs) are a widespread form of non-coding RNAs (ncRNAs). Although circular RNAs have been known for a long time, it is only recently that thier therapeutic implications are being investigated. Research has shown that they are suitable candidates to be genetic bio-markers for different types of cancer, since they are implied in Carcinogenesis. However, their identification involves a computationally intensive and very long process. This represents a serious obstacle to their ability to provide a fast and accurate life-saving diagnosis. Here we present circFA (Circular RNA FPGA Aligner), a hardware acceleration of the circular RNAs identification protocol, aimed at reducing diagnosis time and so producing a pharmacological therapy tailored on the specific patients genetic information. Our solution aims at accelerating parts of Bowtie2, one of the most used sequencing reads aligners, by implementing on FPGA a scalable version of the Smith-Waterman algorithm with affine gaps employed by the framework. This acceleration could be integrated in the currently available tool, thus accelerating the whole Circular RNA identification pipeline, and also providing a fast read aligner that can be useful in many other applications.

1.2 Context definition

Traditional cancer diagnoses establishments use a complex combination of clinical and histopathological data [1]. However, the retrieval of these data is long, time consuming and in some instances, impossible [1]. The latest technological advancements in RNA sequencing offer new paths to cancer screening such as biomarkers [2]. Biomarkers are biological molecules found in blood, other body fluids or tissues, they are a sign of a normal or abnormal process or of a condition or disease, they may be used to see how the body responds to a treatment for a disease or to identify a pathological condition and its

position in the body [3]. This approach to cancer diagnosis aims at detecting the pathology in its early stages and identify potential metastasis and their location before a critical spreading. Recent discoveries have shown that circular RNAs (circRNAs), biomolecules involved in central nervous system diseases, such as Alzheimers disease (AD) [4] and Epithelial to Mesenchymal Transition (EMT) [5] thus in Carcinogenesis of various forms of malignancies [6, 7], are suitable candidates to become genetic biomarkers [8, 9]. CircRNAs act as regulators of post-transcriptional products, behaving as sponges for microRNA (miRNA), playing an important role in fine-tuning gene expression, thus a possible disregulation of the pathway can lead to a pathological event [10]. These crucial discoveries led to a growing interest in the identification of circular RNAs aimed at the detection of their position in the genome, effects and expression levels in different body sections in physiological and pathological conditions. Such information positioned in dedicated databases integrated with a fast and efficient protocol would lead to a considerable reduction in diagnosis time, thus in the design of a therapy that would be tailored following the specific mutation and genetic heritage of the patient, leading to the use of drugs with less aggressive side effects and decreased resolution time. Latest circular RNAs identification pipelines count on a previous alignment step performed mostly by Bowtie2-based aligners. The alignment process requires a huge amount of time, often not available both to the patient and medic, even with very powerful machines. One of the most commonly used tools for in the circular RNAs identification pipeline is TopHat2, which provides very accurate and comprehensive results and uses the Bowtie2 software as its core to provide the actual read alignment functionalities. Bowtie2 is one of the most used read aligners and, among other algorithms, implements the Smith-Waterman (SW) algorithm for the scores computation. A seed-and-extend method is employed to align reads, together with a particular variation of the SW algorithm. This implementation of the algorithm considers the gaps penalties in reads to be linear, instead of the commonly used fixed penalty notion, thus providing more accurate results than its competitors.

1.3 Project aim

Our project accelerates the end-to-end sequence alignment functionality of Bowtie2. In particular our design implements the Smith-Watermann algorithm with affine gap penalties. The system is composed of one software and one hardware component. The host application can take as input two .fasta files containing the genome reads and the corresponding reference genome sequences previously identified by the seed-and-extend process. The hardware-accelerated component is a kernel which implements the Smith-Watermann algorithm on FPGA, for reads and reference sequences of any length. We placed particular emphasis on the ability to process reads of different lengths effectively, as this would be required in order to integrate the component into

a general sequence aligner. This hardware-accelerated component could be integrated in the Bowtie2 software accelerate the whole circular RNAs identification pipeline. This is however outside the scope of the project for the moment, thus we provide our tool as a stand-alone entity with its own input and output interface. The system has been deployed in an Amazon F1 instance and thus can be made available in the cloud.

2

Design

The following sections will describe the hardware and software components designed as part of the project and explain their role in the overall system, as well as illustrate the development platforms used and test setup.

2.1 Software

The software application is a C++ host for the hardware kernel, which is implemented with the OpenCL paradigm through the interface provided by SDAccel. It consists of a simple interface where the user can provide as input arguments two FASTA files, containing the read sequences and the corresponding reference genome fragments. We suppose that these files have been produced by a tool performing, for example, a seed-and-extend step on a set of input sequence reads and an indexed reference genome. These are all steps that can be performed with most software aligners. We chose the FASTA format for the input sequences as it is a very common format supported by most genomics-related software. The software application parses the input with the help of a third party open source header that contains functionalities to store the sequences into appropriate data structures in memory. Then it processes each sequence, transforming the original ASCII encoding into a custom three-bit encoding. A straight-forward two-bit encoding was not possible since in addition to the four DNA bases, Bowtie2 uses and additional character N, which represents an ambiguity occurred in the sequencing phase and is treated as a separate case in the scoring process. Once the sequences have been encoded, the application sets up the necessary infrastructure to call the kernel. The kernel is then called for each pair of input read and reference, and it outputs the maximum score for the pair, its position in the dynamic-programming matrix and a matrix of directions which allows to perform the following backtracking phase. All these elements will be discussed in more detail in the hardware section, where we present the hardware kernel and its functionalities.

2.2 Algorithm overview

The Smith-Waterman algorithm performs local sequence alignment between two strings of nucleic acidic sequences. It is a dynamic programming algorithm which maximizes the similarity measure for the two sequences according to a specific scoring system, looking at all possible subsequences. The algorithm is optimal, therefore it is guaranteed to find the optimal solution with respect the selected scoring system. The variation of the algorithm implemented in this project uses a particular scoring scheme which accounts for different type of mismatches and affine gap penalties. This implies that there are more data dependencies to carry in order to calculate the correct solution to the sub-problems. The structure of the a solution is as follows:

1. Let $Q = q_1, q_2, ...q_m$ and $D = d_1, d_2, ...d_n$ be the query and reference sequences respectively
2. A scoring matrix $W(q_i, d_j)$ is defined for all residue pairs, and accounts the respective match and mismatch penalties
3. Two different gap penalties G_{init} and G_{ext} are defined for starting and continuing a gap in the alignment
4. We define the matrix to hold the alignment score for each pair of characters in the two sequences as $H(i, j)$ with $1 < i < m$ and $1 < j < n$
5. The alignment scores ending with a gap along D and Q respectively are held in the two matrices $E(i, j)$ and $F(i, j)$

Given these elements, we can define the equations which recursively describe the solutions to the dynamic programming probalem as

$$E(i,j) = max \begin{cases} E_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{init} \end{cases}$$

$$F(i,j) = max \begin{cases} F_{i-1,j} - G_{ext} \\ H_{i-1,j} - G_{init} \end{cases}$$

$$H(i,j) = max \begin{cases} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} - W(q_i, d_j) \end{cases}$$

The alignment score matrix is calculated for all (i, j) and the maximum score value is located. Moreover, in our implementation we define another matrix $D(i, j)$ which contains for each element information on whether the maximum value came from west, north, north-west or any combination of the three, in case of a tie. This allows to discard the three score matrices and output only this matrix of directions to perform the backtracking phase and identify the optimal alignment for the sequence.

2.3 Hardware

The hardware kernel has been designed to accelerate the execution of the Smith-Waterman algorithm on a pair of database and query sequences of variable length. This has been achieved using a scoring kernel which takes as input two sequences of fixed length and calculates a set of scoring values in parallel, and a surrounding infrastructure that provides the scoring kernel with the appropriate inputs and takes care of splitting the computation of the scoring matrices into smaller sections and managing the data dependencies among them.

2.3.1 Scoring kernel

The most central element in our hardware design is the scoring kernel. This component is tasked with performing the actual computation of the scoring matrices and the direction matrix. Our implementation is based on a well-known architecture for hardware implementations of the Smith-Waterman algorithm, which uses a systolic array. This architecture leverages the fact that all the cells of the scoring matrix which lie on the same antidiagonal have no data dependencies among one another and can be computed in parallel, as shown in Figure 2.1. The kernel takes as inputs a query and a database sequence, as well as four buffers of the same dimension as the number of processing elements in the systolic array plus one. These buffers contain the values of E, F and H calculated for the previous antidiagonal, as well as the values of H calculated two cycles before. In this way we can retrieve the dependencies for each processing element, which correspond to the values $H(i-1, j-1)$, $H(i-1, j)$, $H(i, j-1)$, $F(i-1, j)$ and $E(i, j-1)$. The kernel also takes as input a range which identifies on which processing elements we need to perform the computation. The main loop, which is unrolled in hardware, calculates the values of the scoring matrices for the current iteration as well as the values of the directions matrix D which is placed in a temporary buffer. The maximum score value calculated for the current antidiagonal and its position are saved in a buffer. This is done in order to retrieve them and calculate the overall maximum score and its position once the scores for all the diagonals have been calculated. A second loop writes back the diagonal of the D matrix in a buffer in BRAM. This scoring function is placed within a pipelined loop that receives at each iteration the appropriate input query and database sequence. The policy for selecting the appropriate inputs will be further detailed in the following section. The systolic array has been dimensioned taking into account the variability of the length of the input queries in order to maximize the hardware utilization. This has led to the implementation of a systolic array of 42 processing elements.

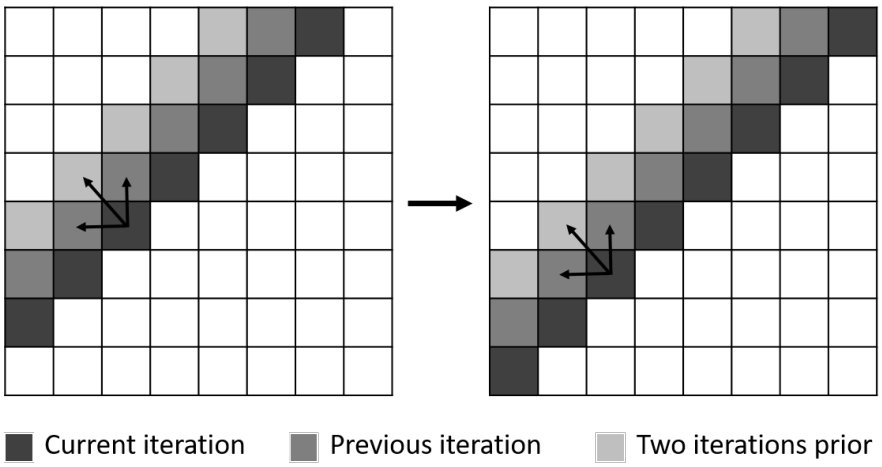

Fig. 2.1. Dependencies for a given cell of the alignment matrix at a given iteration

2.3.2 Scheduling

In order to calculate the complete scoring matrices and the corresponding directions matrix for a given input query and reference, we have split the computation into different tiles. We define a tile as a subsection of the scoring matrix. If we try to split the computation of the scoring matrix into smaller subdiagonals, we observe that there are data dependencies between the values calculated at a given iteration of a subdiagonal and the values of the subdiagonal right above it. Therefore we must calculate the complete antidiagonal starting from the bottom subsection, and proceeding to feed the appropriate data dependencies to the next subsection. Each tile stores the dependencies for a given subdiagonal, and at each iteration it computes the values for the next subdiagonal. In our design we have two types of tile: one which covers a horizontal subsection of the scoring matrix from the first row for a number of rows equal to the number of the processing elements in the scoring kernel, and several others which cover vertical subsections of the scoring matrix for the same number of columns. When the execution starts, a tile of the first type starts computing each antidiagonal. The characters of the reference sequence are given to the scoring kernel following a FIFO policy, whereas the characters of the query sequence are fixed, each assigned to one processing element. An example of this type of iteration is shown in Figure 2.2. When the number of elements in the antidiagonal exceeds the number of PEs in the systolic array, another tile of the second type starts to calculate the subdiagonal immediately below. At this point the first tile reads the data dependences written by this second tile and calculates the remaining subsection of the antidiagonal. The tile of second type will continue to compute the subdiagonals of a vertical section of the matrix, calculating at each execution the subdiagonal

below the previous one. After this type of tile starts computing subdiagonals of maximum length, the input reference sequence remains fixed and the input query sequence shifts by one at each cycle. Whenever the tile of the first type computes $|PE|$ cycles, a tile of the second type is initialized and starts to compute a new vertical section. In this way we are able to cover the whole scoring matrix. An example of how the tiles map on the scoring matrix is shown in Figure 2.3. The main loop of the hardware kernel is a pipeline where the scoring kernel is called with the appropriate input arguments at each iteration, the resulting directions antidiagonal is written back in DRAM and the setNextValues() function takes care of computing which tile needs to execute at the next iteration and sets the inputs for the next execution of the scoring kernel. When all the antidiagonal have been computed, another loop computes the overall maximum score value and its position.

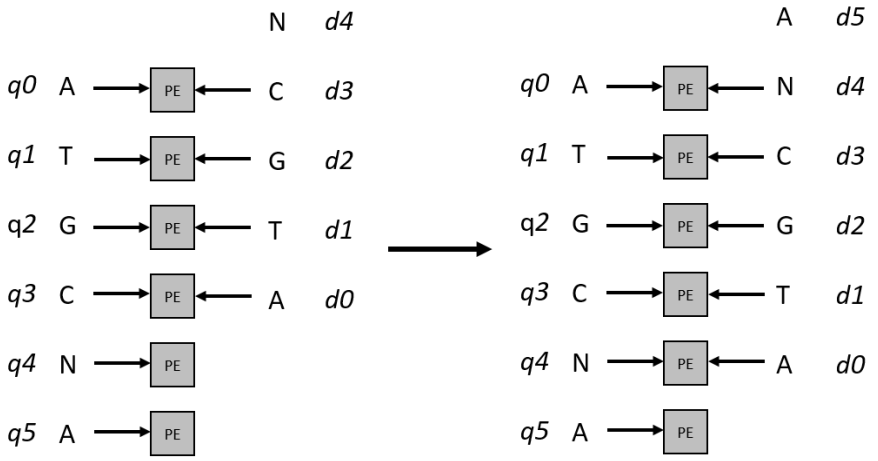

Fig. 2.2. The input read and reference characters for two subsequent iterations of the scoring kernel

2.3.3 Memory organization

In the generic case, we cannot know at compile time the maximum number of tiles that will be executing concurrently. For this reason we need a policy to instruct the kernel on where to write the results of a call to the scoring kernel at each iteration, and where to read the appropriate data dependencies for the next execution. In our design we use a buffer implemented directly in logic which holds at any given time all the inputs necessary for the next

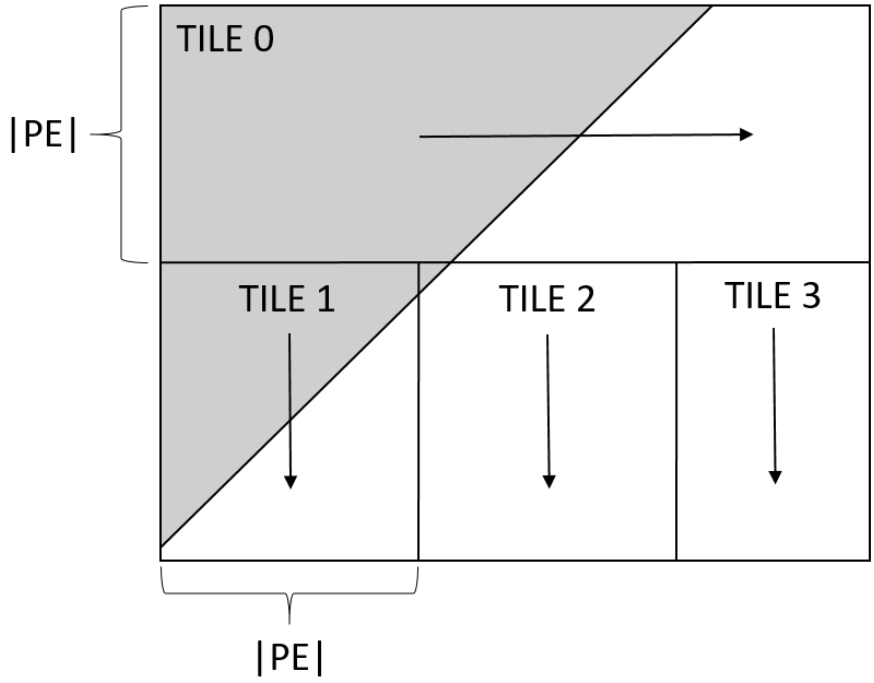

Fig. 2.3. Division of the scoring matrix into tiles and order of progression of the scoring computation

call to the scoring kernel as well as results of the previous iteration. If we have two active tiles, we can read the necessary dependencies as well as write the results of the computation at each cycle by alternating the positions from which we write and read from the buffer. If we have more than two tiles, when the results of an iteration are written in the buffer, the results of the iteration preceeding it are written to BRAM. In the same way, when a set of inputs is read from the buffer, the set of inputs for the iteration that will follow it are written from BRAM into the buffer. In this way we can store a variable number of results from different tiles in BRAM, and thus process queries of variable length, as long as the maximum concurrent number of tiles does not occupy all the BRAMs available in the BRAM buffer.

2.3.4 Tools and platforms

The hardware-accelerated components were developed using both Vivado HLS 2017.2 and SDAccel 2017.1.op. In this context a software version of the algorithm has been implemented for testing and benchmarking purposes.

2.4 Design reuse

In order to facilitate the reuse all or part of this design, all the source code is open source and publicly available at `https://github.com/albertozeni/circFAXOHW18public/` and has been appropriately commented and documented.

3

Results

In order to validate our design we have compared it to a single-threaded software implementation of the algorithm, available at `https://github.com/albertozeni/circFAXOHW18public/`. In order to account for the fact that typically the algorithm is implemented using vector instructions described in [11], we accounted for a 6x speedup in execution time. The software version was run on an Intel(R) Xeon E5-2680 v2 CPU with a frequency of 2.8 GHz, and compiled with gcc 4.8.5 with -O3 optimization. The hardware accelerated version was run on an Amazon F1 instance equipped with a Virtex Ultrascale+ VU9P FPGA. The design was synthesized with a clock frequency of 250 MHz. The resulting execution times are shown in Table 3.1. Our design showed a speedup over the vectorized software version of 1.46x, and over the simple software implementation of 8.8x. The resource utilization of the design is shown in Table 3.2. Although our design is not the state of the art FPGA-accelerated Smith-Waterman in terms of speedup, it is important to note that our design does not loose in performance when the length of the input reads grows. In our current implementation we decided to consider a maximum read length of 3024 bp, since it was a reasonable upper bound for the applicative domain of interest, but given that our design uses only 10.5% of the BRAM blocks available on the FPGA and that is the only critical resource which grows with the maximum read length, we could increase this upper bound even further. On the same token, since our kernel is relatively small with only 42 processing elements and it is not memory bound, we could use more instances of the kernel in parallel to increase the overall computation capabilities of our system.

Youtube presentation video: urlhttps://youtu.be/nHDAhXOTM4A

Software	Vectorized	FPGA
631.66 ms	105.27 ms	71.78 ms

Table 3.1. Execution times averaged over 20 executions of a query and reference pair of 3024 bp

LUT	FF	BRAM	DSP
7.7%	2.2%	10.5%	0.0%

Table 3.2. The resource utilization percentages for the synthesized design

4

Conclusions

4.1 Challenges

During the development of this project our team has encountered many challenges, as well as opportunities to learn and experiment with different ideas. Since for all the members of the team this was the first hardware acceleration project using Vivado HLS and SDAccel, we had to spend time to familiarize with the Xilinx toolchain and face the challenges that come with designing for FPGA and hardware in general. Moreover, since the acceleration of the Smith-Waterman algorithm on FPGA is very much an explored topic, onto one hand we had an already developed body of literature to use as a starting point, but at the same time we had to struggle to find a way to add our own contribution and tailor our design specifically on the circular RNAs use case. The major setback for the project, which has considerably affected the resulting performance of the design, was the difficulty we encountered while designing the infrastructure surrounding the scoring kernel, responsible to manage the dependencies between subdiagonals. The complex design which resulted from this difficulty caused the pipeline of our system to have a very high initiation interval, which meant a considerable degradation in performance. Since there are no floating point operations involved in the computation, we believe that the initiation interval can be reduced with a simpler and more linear design. If we were to eliminate this bottleneck and match the ideal performance of the scoring kernel, we estimate a speedup with respect to our current design of about 22x. With regards to the team dynamic and decision making during the course of the project, we had to make many different technical and strategic decisions, and we managed to do this for the most part. In hindsight, we may have realized sooner that the design of the scheduling part had to be revised, through better team communication and coordination.

4.2 Future work

One of the future works for this project is to improve the design the scheduling part of the system, simplifying it in order to gain in performance. Once we are satisfied with the performance and resource utilization of a single kernel, we will explore the possibility to implement multiple instances of the kernel in the same FPGA in order

to leverage the intrinsic parallelism of the alignment process. This scenario opens up different possibility for optimization in terms of the sharing of computational and memory resources among the different kernels. Since with the instantiation of more computational kernels we will reach a point where the system performance is bound by the available memory bandwidth, we want to explore different possible encodings for the input reads and reference, as for example the Burrows-Wheeler Transform (BWT). Another future work will be the design of a hardware-accelerated backtracking functionality, which is the next step in the sequence alignment pipeline. Moreover, the final aim of this project is to be integrated with the state-of-the-art alignment software, in particular Bowtie2, to create a software/hardware pipeline for the identification of circular RNAs.

References

1. S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C.-H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, *et al.*, "Multiclass cancer diagnosis using tumor gene expression signatures," *Proceedings of the National Academy of Sciences*, vol. 98, no. 26, pp. 15149–15154, 2001.
2. D. E. Henson, S. Srivastava, and B. S. Kramer, "Molecular and genetic targets in early detection," *Current opinion in oncology*, vol. 11, no. 5, p. 419, 1999.
3. NIH National Cancer Institute., "Biomarker Definition."
4. W. J. Lukiw, "Circular rna (circrna) in alzheimers disease (ad)," *Front Genet*, vol. 4, no. 4, p. 307, 2013.
5. S. J. Conn, K. A. Pillman, J. Toubia, V. M. Conn, M. Salmanidis, C. A. Phillips, S. Roslan, A. W. Schreiber, P. A. Gregory, and G. J. Goodall, "The rna binding protein quaking regulates formation of circrnas," *Cell*, vol. 160, no. 6, pp. 1125–1134, 2015.
6. T. L. H. Okholm, M. M. Nielsen, M. P. Hamilton, L.-L. Christensen, S. Vang, J. Hedegaard, T. B. Hansen, J. Kjems, L. Dyrskjøt, and J. S. Pedersen, "Circular rna expression is abundant and correlated to aggressiveness in early-stage bladder cancer," *NPJ genomic medicine*, vol. 2, no. 1, p. 36, 2017.
7. Z. Kun-Peng, M. Xiao-Long, and Z. Chun-Lin, "Overexpressed circpvt1, a potential new circular rna biomarker, contributes to doxorubicin and cisplatin resistance of osteosarcoma cells by regulating abcb1," *International journal of biological sciences*, vol. 14, no. 3, p. 321, 2018.
8. S. Meng, H. Zhou, Z. Feng, Z. Xu, Y. Tang, P. Li, and M. Wu, "Circrna: functions and properties of a novel potential biomarker for cancer," *Molecular cancer*, vol. 16, no. 1, p. 94, 2017.
9. D. Rong, H. Sun, Z. Li, S. Liu, C. Dong, K. Fu, W. Tang, and H. Cao, "An emerging function of circrna-mirnas-mrna axis in human diseases," *Oncotarget*, vol. 8, no. 42, p. 73271, 2017.
10. T. B. Hansen, J. Kjems, and C. K. Damgaard, "Circular rna and mir-7 in cancer," *Cancer research*, vol. 73, no. 18, pp. 5609–5612, 2013.
11. M. Farrar, "Striped smith–waterman speeds database searches six times over other simd implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2006.