High scoring segment selection for pairwise whole genome sequence alignment with the maximum scoring subsequence and GPUs

Abdulrhman Aljouie

Department of Computer Science,
New Jersey Institute of Technology,
Newark, NJ, USA
E-mail: aa547@njit.edu

Ling Zhong

Department of Computer Science,
New Jersey Institute of Technology,
Newark, NJ, USA
E-mail: lz25@njit.edu

Usman Roshan

Department of Computer Science,
New Jersey Institute of Technology,
Newark, NJ, USA
E-mail: usman@njit.edu

Abstract: Whole genome alignment programs use exact string matching with hash tables to quickly identify high scoring fragments between a query and target sequence around which a full alignment is then built. In a recent large-scale comparison of alignment programs called Alignathon it was discovered that while evolutionary similar genomes were easy to align, divergent genomes still posed a challenge to existing methods. As a first step to fill this gap we explore the use of more exact methods to identify high scoring fragments which we then pass on to a standard pipeline. We identify such segments between two whole genome sequences with the maximum scoring subsequence instead of hash tables. This is computationally extremely expensive and so we employ the parallelism of a Graphics Processing Unit to speed it up. We split the query genome into several fragments and determine its best match to the target with a previously published GPU algorithm for aligning short reads to a genome sequence. We then pass such high scoring fragments on to the LASTZ program which extends the fragment to obtain a more complete alignment. Upon evaluation on simulated data, where the true alignment is known, we see that this method gives an average of at least 20% higher accuracy than the alignment given by LASTZ at the expense of a few hours of additional runtime. We make our source code freely available at `web.njit.edu/\~usman/MSGA`.

Keywords: Genome Alignment; Anchor Selection; LASTZ; GPU

Biographical notes: Abdulrhman Aljouie is currently a PhD student in the Computer Science Department at the New Jersey Institute of Technology. His research interests are in genomics, in particular machine learning based disease risk prediction and high performance GPU-based genomics methods.

Ling Zhong is a former PhD student in the Computer Science Department at the New Jersey Institute of Technology. She defended her thesis titled "Algorithms for pre-microRNA classification and a GPU program for whole genome comparison" in 2017.

Usman Roshan is an Associate Professor in the Department of Computer Science at the New Jersey Institute of Technology. He finished his PhD in Computer Science from The University of Texas at Austin in 2004. His research interests are in cancer risk prediction from genomic data, high performance and machine learning based genomic methods, and machine learning methods for data representation and 0/1 loss optimization.

1 Introduction

Whole genome sequence alignment can identify evolutionary mechanisms at the genome level such as inversions and transpositions[1], conserved functional elements in non-coding regions [2], cis-regulatory regions [3], and non-coding RNAs [4]. An exact alignment approach for two large genome sequences is computationally unfeasible both in speed and space. Instead modern tools such as LASTZ [5] rely on substring matches with hash-tables to identify high scoring segments around which a larger alignment is then built [6, 7, 5, 8, 9]. In a recent large-scale study of whole genome alignment programs called Alignathon [10] it was discovered that evolutionary divergent sequences posed a challenge to existing programs.

Genome alignment programs search for best substring matches between two sequences Q (query) and T (target) using hash-tables for fast lookup. To allow for mismatches and gaps they use spaced seeds [11]. While these allow for a limited number of mismatches and gaps they can pose a challenge for higher evolutionary rates. We explore a method to overcome this limitation in this paper. We split the query genome into short fragments and align each to the target the maximum scoring subsequence. We perform this with the expectation that the maximum scoring subsequence, which is similar to Smith-Waterman local alignment, will capture high scoring fragments missed by the hash-tables. This, however, is extremely computationally expensive and so we use the parallelism of a Graphics Processing Unit (GPU) to speed this up. We then pass our high scoring segments to LASTZ for downstream processing to obtain a full alignment.

In related work there have been several Smith Waterman implementations on GPUs for protein database search [12, 13, 14, 15]. For the problem of whole genome sequence alignment the Darwin platform [16] proposes new algorithms for whole genome alignment with constant memory on an FPGA and algorithms for short read mapping. The SW# [17] and CUDAlign [18] programs propose GPU implementations of Smith-Waterman for whole genome sequence alignment. However, their accuracy is unclear and their method is likely to miss inversions and transpositions since they perform Smith-Waterman on whole genomes instead of fragments.

Our approach is different from previous ones since we basically use a short read alignment method to compare fragments across two genomes. Since short read alignment programs are based on hash-tables we don't expect an improvement over existing whole genome alignment programs unless an exact method for short read alignment is used. Thus, our motivation for using MaxSSmap which is a previously published short read aligner based on a more exact alignment called the maximum scoring subsequence alignment.

We study our method on simulated multiple genome alignments from the Alignathon study [10]. We extract all pairwise alignments as well as the true ones from there. This serves as a benchmark to compare our method against the popular LASTZ program [5]. We show considerable improvements in the alignment accuracy given by our method at the expense of a few hours of additional compute time.

2 Methods

Before describing our method we define a maximum scoring subsequence alignment, which is basically a gapless local sequence alignment.

2.1 Maximum scoring subsequence alignment

The maximum scoring subsequence for a sequence of real numbers $\{x_1, x_2, ..., x_n\}$ is defined to be the contiguous subsequence $\{x_i, ..., x_j\}$ that maximizes the sum $x_i + ... + x_j$ $(0 \leq i, j \leq n)$. We can determine this with the following simple algorithm [19, 20]:

In order to apply this to sequence alignment consider two sequences Q and T where $|Q| = m$, $|T| = n$, and $m \leq n$. We begin at index $i = 0$ and find the maximum scoring subsequence between substrings $Q_{0...m-1}$ and $T_{i...i+m-1}$ for all values of $i = 0...n - m$. By treating the match and mismatch scores as a sequence of numbers we can apply Algorithm 1 to find the maximum scoring subsequence for all values of i while keeping track of the best score and subsequence. See Figure 1 for a toy illustration.

We summarize this in Algorithm 2. Now that we have described how to determine the maximum scoring subsequence alignment between two sequences we are ready to present our genome alignment method.

Algorithm 1 Maximum scoring subsequence

Input: Set of number $x_0, x_1, ..., x_{n-1}$
Output: Contiguous subsequence $M = \{x_i, ..., x_j\}$ that maximizes the sum $max = x_i + ... + x_j$
Procedure:

Let $M = \{\}, C = \{\}, max = 0, curr = 0$
for $i = 0$ to $n - 1$ **do**
  if $curr + x_i > max$ **then**
    $M = M \cup x_i$, $M = C$,$max = curr$
  else
    if $curr + x_i > 0$ **then**
      $C = C \cup x_i$,$curr = curr + x_i$
    else
      $C = \{\}$,$curr = 0$
    end if
  end if
end for

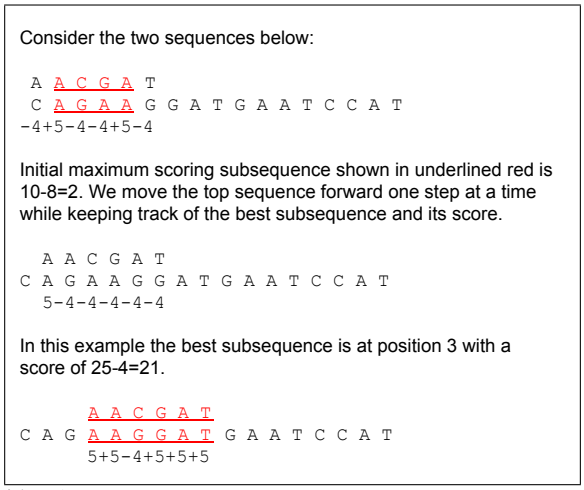


Figure 1 Toy illustration of Algorithm 1

Algorithm 2 Maximum scoring subsequence alignment

Input: DNA sequences Q and T where $|Q| = m$, $|T| = n$, and $m \leq n$, match cost m, mismatch cost mm
Output: Maximum scoring subsequence between Q and T
Procedure:

for $i = 0$ to $n - 1$ **do**
  1. Consider the sequence of match and mismatch scores $x_i, \forall i = 0...n - m$ by aligning $Q_{0...m-1}$ to $T_{i...i+m-1}$.
  2. Call Algorithm 1 with input $x_i, \forall i = 0...n - m$, let $curr$ be the maximum score obtained.
  if $curr > max$ **then**
    $max = curr$
  end if
end for

2.2 Maximum scoring subsequence genome alignment

In Figure 2 we describe the essential steps of our method. We start with the query genome (arbitrarily chosen) and create fragments all of a fixed length and separated by a specified step length. We then align each fragment in parallel

to the target genome with our previously published program MaxSSmap [21]. The MaxSSmap program divides the genome into fragments and returns the one with the best maximum scoring subsequence alignment against the read (using Algorithm 2). In order to speed this up the MaxSSmap program determines the maximum scoring subsequence alignment to all fragments in parallel on a Graphics Processing Unit (GPU).

We then pass the best maximum scoring subsequence alignments as high scoring fragments to LASTZ for downstream analysis. LASTZ extends each high scoring segment to determine a larger alignment. We describe the full method in Algorithm 3.

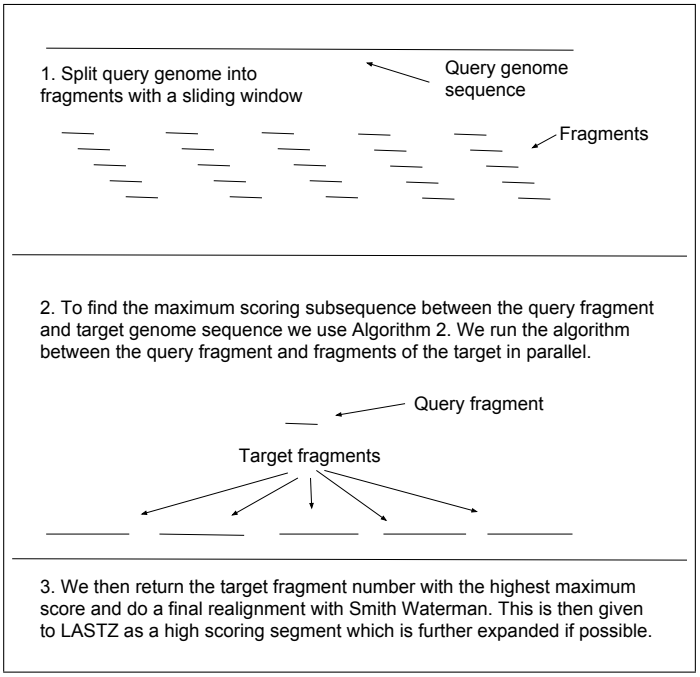


Figure 2 Overview of our genome alignment method

Algorithm 3 MaxSub Genome Alignment

Input: Input genomes Q and T of lengths m and n respectively
Output: Pairwise alignments of substrings of Q and T
Procedure:

1. Split the genome Q into fragments of length w with consecutive fragment separated by a step length of s nucleotides. We do this with a moving window across Q starting at position 0 and moving forward by s nucleotides each time. Let Q_f be the set of such fragments.
2. Treat the fragments in Q_f as *pseudo short reads* and run MaxSSmap [21] with input as Q_f as reads and T as the reference genome sequence. The MaxSSmap program performs short read alignment against the genome with the maximum scoring subsequence and runs on a Graphics Processing Unit for parallelism. This step produces aligned reads which we consider as high scoring segments between the query and target.
3. Pass our high scoring fragments on to LASTZ which expands the fragment to give a complete aligned block.

3 Results

3.1 Experimental performance study

3.1.1 Datasets

We obtain simulated whole genome sequence alignments for primates and mammals from the Alignathon study [10] as a measure of ground truth. The genomes were simulated with the Evolver whole genome evolutionary model [22]. The Evolver evolutionary model is highly complex and incorporates all known genome evolutionary events.

The primate pairwise alignments are easy to determine as previously shown [10]. Thus we focus on the mammalian dataset that contains more evolutionary distant species relative to the primates. The mammallian dataset contains multiple whole genome alignments of several chromosomes from five mammals: dog, cow, rat, mouse, and human. We extract pairwise alignments of chromosomes between all pairs of genomes and use them as a ground truth. Specifically we study the accuracy of alignments of the following pairs of genomes:

Species	Chromosomes (length in parenthesis)
Human	F (41MB), J (88MB)
Dog	A (39MB), D (35MB) , F (64MB)
Rat	A (45MB), Q (54MB), R (88MB)
Cow	A (42MB), B (86MB)
Mouse	F (60MB), L (71MB)

Table 1 Genome sequences (and their lengths in million bases) used in our experimental study

3.1.2 Measure of accuracy

We measure the number of false positives, false negatives, true positives, and true negatives from which we calculate the precision, recall, and f-score. The false positives are number of aligned nucleotides that are present in the computed alignment but missing in the true one while the true positive measures number of correctly aligned nucleotides. The false negatives are aligned nucleotides in the true alignment that are missing from the computed one and the true negative is undefined. From these we calculate the precision, recall, and f-score as

$$precision = \frac{truepositive}{truepositive + falsepositive}$$

$$recall = \frac{truepositive}{truepositive + falsenegative}$$

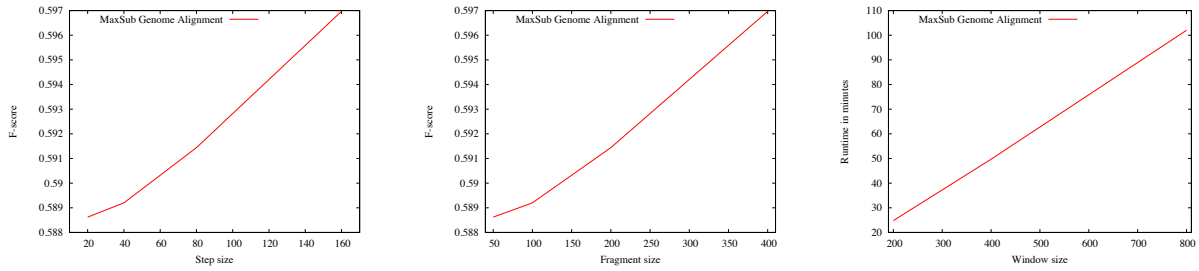
$$fscore = 2 \times \frac{precision \times recall}{precision + recall}$$

3.1.3 Experimental platform and source code

We conduct all experiments on computing nodes equipped with Intel Xeon E5-2630-v4 CPUs and NVIDIA Tesla P100 16GB Pascal GPUs. We implement our method with Python programs and make external system calls to MaxSSmap [21] for determining anchors.

3.1.4 Programs compared and parameters

We compare our genome alignment method to the popular LASTZ program [5]. LASTZ has been shown previously in the Alignathon study to perform competitively against other genome alignment programs [10]. We use optimal parameters for the human genome given in the UCSC genomewiki `http://genomewiki.ucsc.edu/index.php/Hg38_100-way_conservation_lastz_parameters`. We use these parameters for all genome sequences used in this study: `--inner=2000 --transition --ydrop=9400 --masking=0 --gap=400,30 --gappedthresh=3000 --format=maf`



(a) (b) (c)

Figure 3 Effect of step size and window size on the accuracy and runtime of our genome alignment method. Here we show f-scores for aligning the ratQ and dogA chromosomes. In (a) we fix the window size at 400 and in (b) we fix the step size to 40. In both cases we see that a higher window and step size gives better f-scores for this pair of chromosomes. In (c) we fix the step size to 160 and see that the runtime is almost linearly correlated with the window size.

3.1.5 GPU parameters

In the GPU architecture there are several multiprocessors each of which can execute several threads (usually 32) at the same time. The number of jobs to run in this architectures can be arranged into blocks and threads [23]. In MaxSSmap the number of jobs to run is set to the target genome length divided by a specified target fragment length of 480 that we use in our experiments. We use a fixed number of threads of 256 in each run and obtain the number of blocks automatically by dividing the number of jobs by number of threads. For bacterial genomes we recommend a fragment of length of 48 and for human size genomes 4800. The GPU global required memory is the number of characters in the target genome. In our case the largest genome sequence is 88MB but for a human genome 3GB global memory would be required.

3.2 Effect of fragment and step size

We first study the effect of fragment and step size on our method with the ratQ chromosome as the query and dogA chromosome as the target. In Figure 3(a) we show the effect of increasing the step length with a fixed fragment size of 400. We see that the f-score increases as the step length increases. In Figure 3(b) we see that the f-score increases if we fix the step size at 40 and increase the fragment size. In both cases the improvement in accuracy is very little and similar but the effect on runtime is much greater. In Figure 3(c) we see that the runtime increases almost linearly as we increase the window size.

A larger window size means more coverage of the query and thus better hits and higher f-scores. However, this comes at a cost of runtime because each GPU thread is aligning a longer sequence than before. This may appear strange at first because a larger query window size means fewer sequences to be aligned, whereas a short length means more sequences but in both cases we have the same number of characters in the query reads. This can be explained by the fact that GPUs are optimal for running thousands of *short* functions quickly and simultaneously as opposed to functions that take longer to run [24]. A large window size means each GPU thread takes longer to find the best hit to the genome and thus increase the overall runtime.

3.3 Comparison to LASTZ on simulated data

We now compare the f-score of LASTZ to our method on several pairwise genome sequences. We set the fragment size to 800 and the step size to 160 in our method. In Figure 4 we see the f-score of our method and LASTZ on selected pairs of chromosomes. In every case we see that the high scoring segments given by our maximum scoring subsequence approach give alignments with higher f-scores than LASTZ. Thus the high scoring fragments given by our more exact approach indeed pays off in the downstream analysis.

In Figure 5 we compare the time taken (in hours) by our method and LASTZ. The runtime for our method includes the time for running MaxSSmap and for LASTZ to produce a final alignment with our high scoring segments (the latter is much lower than the former). We see that our method takes longer since it uses a more exact approach for finding

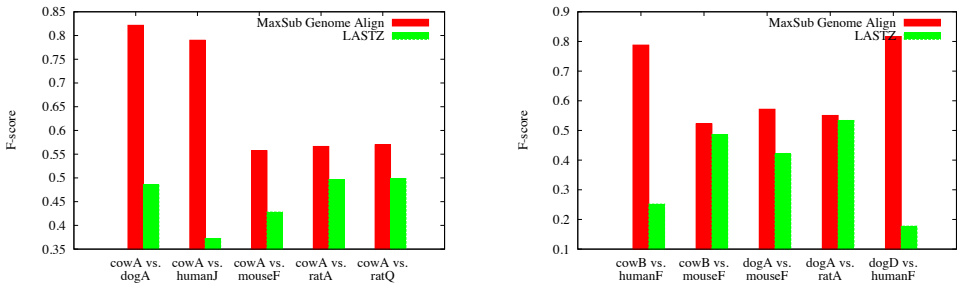


Figure 4 We see that our genome alignment method gives a higher f-score than LASTZ in every pair of genomes tested.

high scoring segments. Our runtimes are still tractable (comparable to LASTZ in some cases) and could potentially be lowered with a shorter window size than 800 that we use here (see Figure 3(c)).

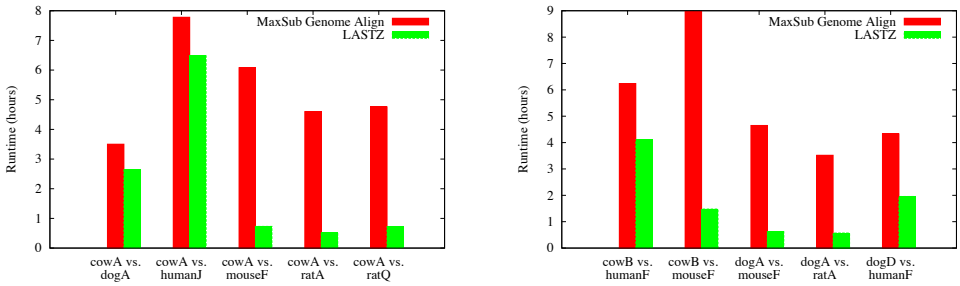


Figure 5 The increase in accuracy by our method comes at a cost of additional runtime. However, they are comparable in some cases and still tractable.

In total we examined 18 pairs of genomes and obtained an average f-score of 0.5 with our method vs 0.24 with LASTZ. Our method takes an average of 5.4 hours whereas LASTZ finishes in an average of 2.28 hours. Thus we obtain an improvement of doubled f-score at the cost of doubled runtime. However, the additional runtime is still in hours as opposed to days and weeks. We expect it to be half with a window size of 400 which makes it the same as LASTZ but it may lower the f-score as well.

4 Discussion and Future Work

Our work here is an initial study to evaluate the effect of using more exact high scoring segments for genome alignment. Instead of MaxSSmap other short read aligners could be used in our algorithm. For example CUDASW++ that uses Smith Waterman could give similar results as shown here, however it is also much slower than MaxSSmap [21]. We don't expect an advantage with mainstream short read aligners that are based on hash-tables. In results not shown here we obtained comparable results to the standalone LASTZ if we replace the MaxSSmap step in our algorithm with a state of the art short read aligner such as NextGenMap [25].

Our LASTZ alignment scores are lower than reported ones in the Alignathon study [10]. This is possibly due to post processing methods that we did not apply here. One such post-processing is to remove overlapping regions with the single_cov2 program that may affect final alignments given both by LASTZ and our method [26].

In future work we plan to compare this to other alignment programs such as Pecan [8] and study the effect of post-processing alignments (such as removing overlapping regions [26]). We will apply our method on real genome

sequences and evaluate the alignments there by counting number of correctly aligned genes as done previously [7]. We also plan to visualize our genome alignments to identify regions aligned differently with our method. Finally, our alignments may give more accurate evolutionary distance matrices which in turn would give better whole genome phylogenies.

5 Conclusion

We determine high scoring segments with the maximum scoring subsequence around which a whole genome alignment is then built. We show that our segments give more accurate alignments than segments identified with standard hash-table methods.

Acknowledgements

We thank the NJIT Academic and Research Computing Systems Group (ARCS) for their support in conducting computational experiments for this study.

References

[1] Aaron CE Darling, Bob Mau, Frederick R Blattner, and Nicole T Perna. Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome research*, 14(7):1394–1403, 2004.

[2] ENCODE Project Consortium et al. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57–74, 2012.

[3] Manolis Kellis, Nick Patterson, Matthew Endrizzi, Bruce Birren, and Eric S Lander. Sequencing and comparison of yeast species to identify genes and regulatory elements. *Nature*, 423(6937):241–254, 2003.

[4] Sebastian Will, Michael Yu, and Bonnie Berger. Structure-based whole-genome realignment reveals many novel noncoding rnas. *Genome research*, 23(6):1018–1027, 2013.

[5] R.S. Harris. Improved pairwise alignment of genomic dna. *Ph.D. Thesis, The Pennsylvania State University.*, 2007.

[6] Stefan Kurtz, Adam Phillippy, Arthur Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):R12, 2004.

[7] Michael Brudno, Chuong B Do, Gregory M Cooper, Michael F Kim, Eugene Davydov, Eric D Green, Arend Sidow, Serafim Batzoglou, NISC Comparative Sequencing Program, et al. Lagan and multi-lagan: efficient tools for large-scale multiple alignment of genomic dna. *Genome research*, 13(4):721–731, 2003.

[8] Benedict Paten, Javier Herrero, Kathryn Beal, Stephen Fitzgerald, and Ewan Birney. Enredo and pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome research*, 18(11):1814–1828, 2008.

[9] Benedict Paten, Dent Earl, Ngan Nguyen, Mark Diekhans, Daniel Zerbino, and David Haussler. Cactus: Algorithms for genome multiple sequence alignment. *Genome research*, 21(9):1512–1528, 2011.

[10] Dent Earl, Ngan Nguyen, Glenn Hickey, Robert S Harris, Stephen Fitzgerald, Kathryn Beal, Igor Seledtsov, Vladimir Molodtsov, Brian J Raney, Hiram Clawson, et al. Alignathon: a competitive assessment of whole-genome alignment methods. *Genome research*, 24(12):2077–2089, 2014.

[11] Nils Homer, Barry Merriman, and Stanley F. Nelson. Bfast: An alignment tool for large scale genome resequencing. *PLoS ONE*, 4(11):e7767, 11 2009.

[12] Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions. *BMC bioinformatics*, 14(1):117, 2013.

[13] Sheng-Ta Lee, Chun-Yuan Lin, and Che Lun Hung. Gpu-based cloud service for smith-waterman algorithm using frequency distance filtration scheme. *BioMed research international*, 2013, 2013.

[14] Yang Liu, Wayne Huang, John Johnson, and Sheila Vaidya. Gpu accelerated smith-waterman. In *International Conference on Computational Science*, pages 188–195. Springer, 2006.

[15] Yu Liu, Yang Hong, Chun-Yuan Lin, and Che-Lun Hung. Accelerating smith-waterman alignment for protein database search using frequency distance filtration scheme based on cpu-gpu collaborative system. *International journal of genomics*, 2015, 2015.

[16] Yatish Turakhia, Kevin Jie Zheng, Gill Bejerano, and William J. Dally. Darwin: A hardware-acceleration framework for genomic sequence alignment. *bioRxiv*, 2017.

[17] Matija Korpar and Mile Šikić. Sw#–gpu-enabled exact alignments on genome scale. *Bioinformatics*, 29(19):2494–2495, 2013.

[18] Edans Flavius de O Sandes and Alba Cristina MA de Melo. Retrieving smith-waterman alignments with optimizations for megabase biological sequences using gpu. *IEEE Transactions on Parallel and Distributed Systems*, 24(5):1009–1021, 2013.

[19] J. Bentley. *Programming Pearls*. Addison-Wesley, 1986.

[20] Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Trans. Program. Lang. Syst.*, 7:113–136, January 1985.

[21] Turki Turki and Usman Roshan. Maxssmap: a gpu program for mapping divergent short reads to genomes with the maximum scoring subsequence. *BMC genomics*, 15(1):969, 2014.

[22] S. Batzoglou R. Edgar, G. Asimenos and A. Sidow. Evolver, 2009.

[23] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition, 2010.

[24] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[25] Fritz J Sedlazeck, Philipp Rescheneder, and Arndt von Haeseler. Nextgenmap: Fast and accurate read mapping in highly polymorphic genomes. *Bioinformatics*, 2013.

[26] Margulies, Elliott H and Hou, Minmei, and Miller, Webb. A Practical Guide to Using TBA. `https://www.bx.psu.edu/miller_lab/dist/tba_howto.pdf`, 2005.