Alberto Zeni, Marina Martí,

Politecnico di Milano, Milano, Italy

alberto.zeni@polimi.it, marina1.marti@mail.polimi.it

June 2021

Xilinx Open Hardware 2021

Team: xohw21-222

Advisor: Marco D. Santambrogio

Youtube video: https://www.youtube.com/watch?v=P2iyTYlKXXo

Github repository: https://github.com/marinamartiq/xohw21-Darwin-public

1. Introduction

Genomics data is transforming medicine and our understanding of life in fundamental ways; helping the diagnosing and decoding of diseases, the understanding of ancestry through genotype-phenotype matching, understanding molecular bases of evolution, and much more. However, from 2001 to 2018, the growth of genomics data has been of 125% per year, and it is far outpacing the Moore's Law. In contrast, CPU performance has slowed down significantly; from 20% per year in early 2000's to only 3% per year today, which results as a considerable gap between CPU performance and genomics data. This gap is expected to keep increasing in the future, as genomics data continues to double every year, as shown in figure 1.

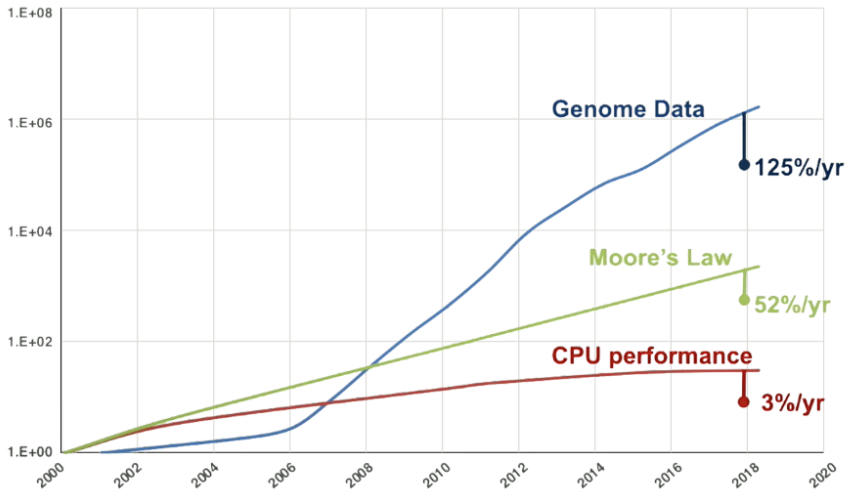

Figure 1. Genomics Data growth comparison with Moore's Law and CPU performance from 2000 to 2018.

As sequencing technologies incur prohibitively high computational costs, in order to enable the vast potential of exponentially growing genomics data, domain specific acceleration provides one of the few remaining approaches to continue to scale compute performance and efficiency, since general-purpose architectures are struggling to handle the huge amount of data needed for genome alignment.

The target application of this work is genomics assembly. The reason why we have such a massive growth in genomics data is due the technology of DNA sequencing, which enables us to read raw nucleotides from a DNA sample. However, the result sequences can be different from the original sequence from which the read was taken, and in particular, the read will have a certain number of substitutions, deletions or insertions with respect to the original sequence, and that is what we call alignment. Most of these errors tend to be stochastic in nature, and so we can naturally lower the errors or fix them, using a consensus of multiple alignment reads, and this process is what we call assembly.

Long reads tend to be very noisy, with a 15-40% of error rate, compared to 0-2% error rate in short reads. As a result, the assembly process, typically when using a reference sequence to guide the assembly, takes up to 5000 CPU hours, and when we assemble the genome *de novo* without the bias or help of a reference genome it can take up to 60000

CPU hours, which is orders of magnitude higher than in short read sequencing. This is the reason why the acceleration for long read assembly is needed.

Table 1. Short and long read assembly required hours.

	Reference-guided assembly (54x human)	**De novo assembly (54x human)**
Short reads (~100bp, 0-2% error rate)	Up to 200 CPU hours	Up to 2,000 CPU hours
Long reads (~10Kbp, 15-40% error rate)	Up to 5,000 CPU hours	Up to 60,000 CPU hours

Long reads are considered to be the holy grail of sequencing for many reasons. It is the only technology which enables us to read a wide spectrum of mutations, in particular structure variations, which is tandem duplications, inversions or pseudo gene insertions in haplotype phasing, which is to distinguish maternal a paternal mutations, and in resolving the repeats, since around 50% of the human genome is repetitive.

2. Related Works

The majority of hardware acceleration efforts for genomic alignment have focused on the Smith-Waterman (SW) and Needleman-Wunsch (NW) algorithms. These algorithms find exact pair wise alignments and have quadratic complexity in the length of the reads.

The SW algorithm performs local sequence alignment; that is, for determining similar regions between two strings of nucleic acid sequences. Instead of looking at the entire sequence, the SW algorithm compares segments of all possible lengths and optimizes the similarity measure. Like the NW algorithm, of which it is a variation, SW is a dynamic programming (DP) algorithm. As such, it has the desirable property that it is guaranteed to find the optimal local alignment with respect to the scoring system being used, which includes the substitution matrix and the gap-scoring scheme. The main difference to the NW algorithm is that negative scoring matrix cells are set to zero, which renders the local alignments visible. Traceback procedure starts at the highest scoring matrix cell and proceeds until a cell with score zero is encountered, yielding the highest scoring local alignment. Because of its quadratic complexity in time and space, it often cannot be practically applied to large-scale problems and is replaced in favour of less general but computationally more efficient alternatives.

Other heuristic approximations to the Smith-Waterman algorithm for pairwise alignment of sequences include Banded Smith-Waterman, X-drop and Myers bit-vector algorithm. All existing heuristics complete the matrix-fill step before starting traceback, and therefore, have a memory requirement that grows at least linearly with the length of sequences being aligned. Darwin uses an extension algorithm that is the first approximate Smith-Waterman algorithm with a constant memory requirement for the compute-intensive step, making it suitable for hardware acceleration of arbitrarily long sequence alignment. Moreover, its alignments, with sufficient overlap, are empirically optimal.

3. Background

The sequence alignment problem takes a query sequence *Q* of letters in {A, C, G, T}, corresponding to the four nucleotide bases, and tries to find the best match (alignment) with a reference sequence *R*, assigning each letter in *R* and *Q* to either a single letter in the opposite sequence, or to a gap. The scoring scheme rewards matching bases, and penalises mismatches and gaps. For example:

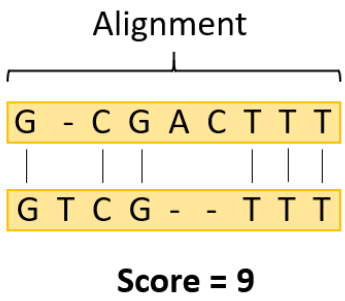

Figure 2. Sequence alignment scoring scheme example.

The *seed-and-extend* approach starts with substrings (the *seeds*) of fixed size *k* drawn from *Q*, and finds their exact matches in *R* (called *seed hits*). Once seed hits are found, the cells surrounding the hits are explored (using a dynamic programming approach).

This avoids the high cost of exploring the full space. A drawback of the seed-and-extend approach is that alignments with no exactly matching substrings of length greater than or equal to *k* will not be discovered, reducing the algorithm's *sensitivity*.

Sequencing is the process of shearing DNA molecules into a large number *N* of random fragments and reading the bases in each fragment. *N* is chosen such that each base-pair is expected to be covered multiple times, enabling errors to be corrected via a consensus mechanism.
Genome assembly is the process of reconstructing a genome *G* from a set of reads *S*. The reconstruction can be *reference guided*, using reference sequence *R* to guide re-assembly, or it can be *de novo*, reconstructing *G* from *S* without the benefit of *R*.

4. Darwin Coprocessor

Darwin is a coprocessor based on the hardware-software co-design for long read assembly. In particular, Darwin uses two novel algorithms, D-SOFT and GACT. The coprocessor is based on the classical seed-and-extend paradigm; D-SOFT performs the seeding step whereas GACT performs the extension step. The algorithms are designed to be accelerated in hardware.

The more compute-intensive step is the extension one, which is used by the GACT algorithm, while the goal of D-SOFT is to greatly reduce the search space for it, during the alignment of a reference *R* with a query *Q*.

D-SOFT starts with *N* substrings, called seeds, of size *k*, drawn from *Q*, and finds their exact matches, called seed hits, in *R*. *R* is pre-processed to generate a seed position table that enables fast look-up of seed hits for a given seed. Next, *R* is divided into N_B number

of unique bins, each associated with a diagonal band having a slope of 1. D-SOFT then counts the number of unique bases in Q covered by seed hits in each diagonal band, and if the count in a given band exceeds h bases, the band is selected for further extension. Figure 3 illustrates this algorithm for $k = 4$, $N = 10$, $h = 8$, $N_B = 6$). There are a total of 6 bins, where bins 1 and 3 both have three seed hits, but in bin 1 only six bases are covered in seed hits, whereas in bin 3, nine bases are covered, which implies a higher probability of finding high-scoring alignment. Since the threshold h is set to 8, only bin 3 is chosen as a candidate bin. Counting unique bases in a diagonal-band allows the algorithm to be more precise at high sensitivity compared to strategies that simply count the number of seed hits, since overlapping bases in seed hits are not counted multiple times. Parameters in D-SOFT can be adjusted to meet the requirements of different sequencing technologies.

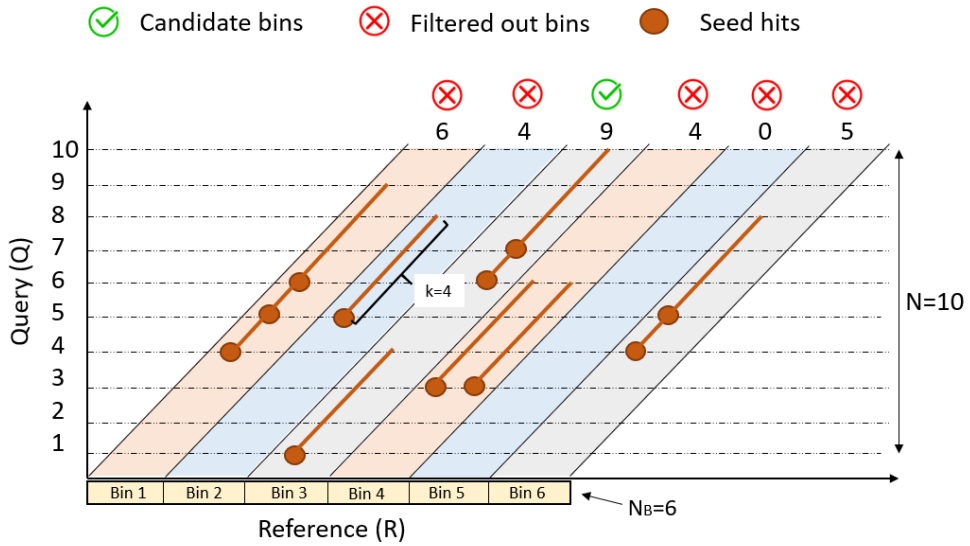

Figure 3. Illustration of D-SOFT algorithm for k=4, N=10, h=8, N_B=6.

GACT performs the extension stage in the seed-and-extend paradigm in which dynamic programming (DP) is used around the seed hit to assign gaps ("-") to R and Q to obtain an alignment that maximizes the score. Prior heuristics reduce the space and time complexity of the optimal Smith-Waterman algorithm from $O(mn)$ to linear $O(m+n)$, where m and n are the lengths of the two sequences being aligned. Long read sequencing technology requires aligning to two long sequences, where previous heuristics require prohibitive traceback pointer storage for hardware implementation. A key innovation of the GACT algorithm is that it finds an arbitrarily long alignment extensions with a constant amount of traceback memory (as opposed to linear or quadratic in previous algorithms) by following the optimal path within overlapping tiles of a fixed maximum size, as shown in Figure 4. GACT traceback is modified from the original Smith-Waterman algorithm to be able to switch alignments of multiple tiles into a single alignment. It uses two parameters, the tile size T and an overlap threshold O. This algorithm gives an optimal result (identical to Smith-Waterman) with reasonable values of T and O, for a tile of size T only T^2 pointers are required to be stored in memory. GACT hardware supports T_{max} = 512 base-pairs, which is enough to produce empirically optimal alignments for long reads. This limits the traceback pointer storage requirement for a tile to only 128 KB, allowing GACT to be massively parallelized in specialized hardware.

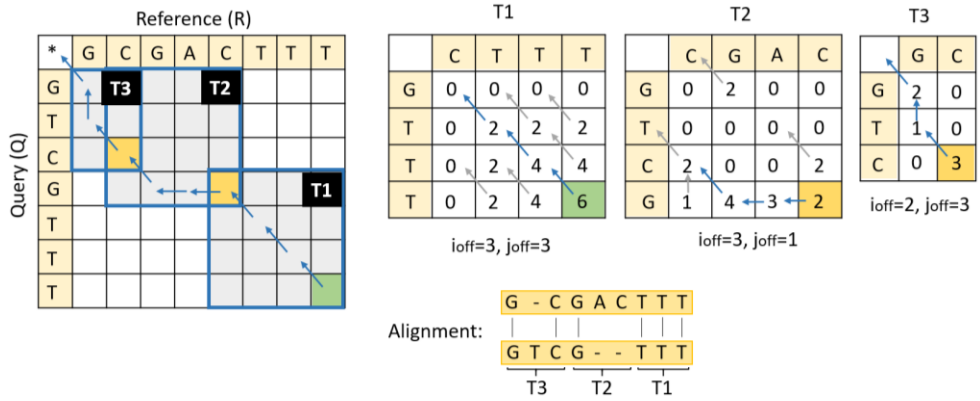

Figure 4. Illustration of extension stage in GACT algorithm using an example dynamic programming (DP) matrix for parameters (T=4, O=1).

Figure 5 shows how Darwin is used for both reference-guided and *de novo* assembly. In both cases, the forward and reverse-complement of *P* reads in $S = \{R\_1, R\_2, ..., R\_P\}$ are used as queries Q_I into a reference sequence R. For reference-guided assembly, an actual reference sequence is employed, while in *de novo* assembly, Darwin accelerates the most compute-intensive overlap phase, which requires finding pairs of overlapping reads using $O(P^2)$ algorithm. *N* seeds from each query Q_I are fed to D-SOFT and the last-hit position of each candidate bin is passed to a GACT array. GACT extends and scores the alignment.

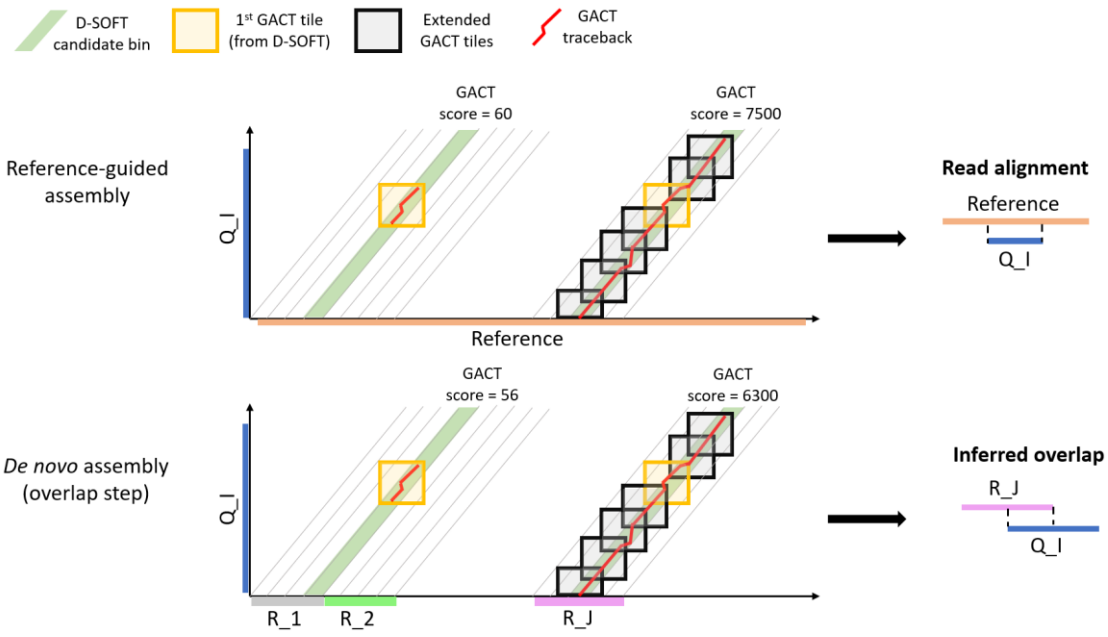

Figure 5. Reference-guided and *de novo* assembly (overlap step) using D-SOFT and GACT.

5. Implementation

The GACT array accelerators handle the compute-intensive *align* routine in GACT, with the rest of the algorithm implemented in software. The arrays are originally implemented on an Intel FPGA. In this work, they have been implemented using the Alveo Card U280, exploiting HBM to increase its performance.

The kernel has been written in RTL, and the communication between the host and the kernel occurs across an AXI bus. The main data processed by the kernel, which is in a large volume, is transferred through the global memory banks on the FPGA board, making use of 6 HBM banks. The kernel accesses the data from those global memory banks, in burst, and after it finishes the computation, the resulting data is transferred back to the host machine through the HBM memory banks.

The AXI4 master interfaces for global memory access are a total of six, which all have 64-bit addresses, which are the query sequence and the reference sequences with their respective addresses, and the direction read address. Each partition in the global memory becomes a kernel argument, and the memory offset for each partition is set by a control register programmed via the AXI4-Lite slave interface.

With respect to the scalar arguments, which are directly written to the kernel through the AXI4-Lite slave interface, there are a total of 11, which are the input register for the Darwin GACT logic. These inputs are control variables directly loaded from the host machine and they do not use global memory banks.

The RTL Darwin kernel consists of a top-level Verilog design which contains a control register and the Darwin sub-modules with a read and write module for each AXI4 master. The following figure illustrates the top-level design configured with six AXI4-master interfaces.

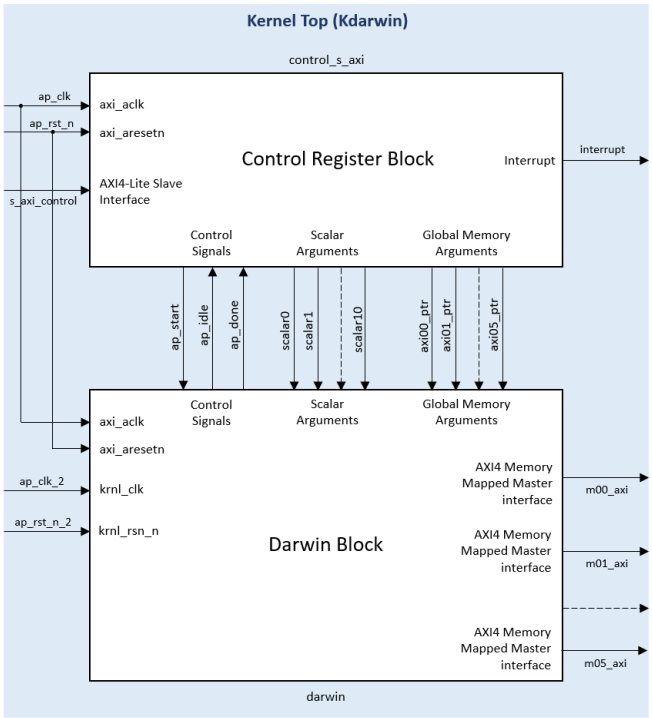

Figure 6. Top-level Darwin kernel structure.

The Darwin block, shown in the following figure, consists of the GACT Darwin logic, six AXI4 read masters, and six AXI4 write masters. Each master reads 16 KB of data, performs the necessary operations in the Darwin logic block, and then writes out the Darwin outputs back in place. Each master AXI uses a different HBM memory bank, from HBM[0] to HBM[5].

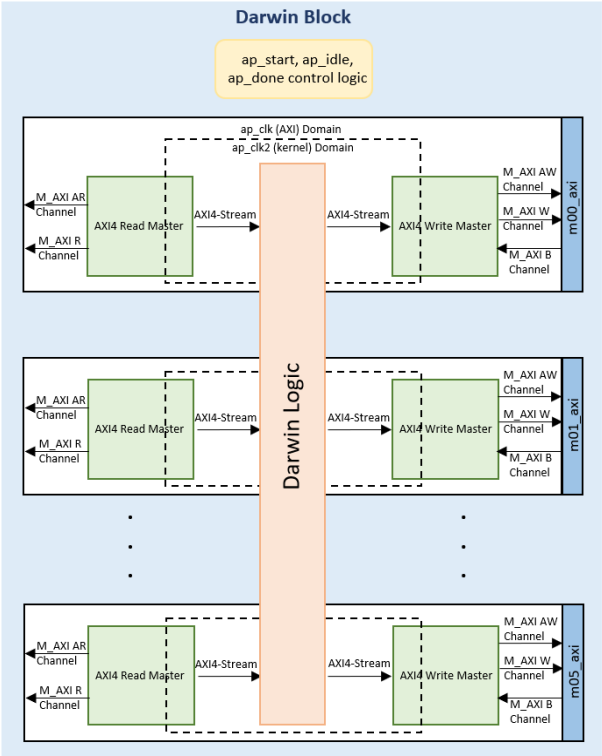

Figure 7. Darwin kernel RTL block structure.

6. Results

The final design has been implemented in the Alveo U280 Accelerator Card, exploiting HBM to increase the performance of the co-processor.

The implementation of the design resulted in 145.666.450 Cell Updates Per Second (CUPS) per tile, with a tile size set to the maximum which is 512 (T=512), this is 146 Mega CUPS per tile. Cell Update Per Second (CUPS) is a standard measurement unit widely used in these kinds of algorithms.

In the Darwin co-processor, Turakhia et al. [1] synthesized 40 GACT arrays of 32 PEs on the Arria 10 FPGA. GACT arrays on FPGA provided a peak throughput of 1.3 Milion tiles per second with the tile size set at 320 (T=320), thus having a sequence length of 320, as shown in Figure 8.

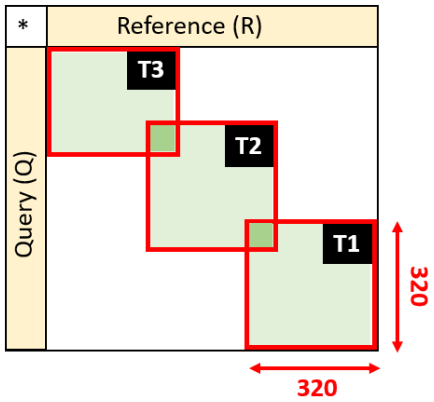

Figure 8. GACT array for tile size T = 320.

The actual design is only synthesized for 1 GACT array on the Alveo U280 FPGA. In order to compare the results with the original Darwin co-processor [1], the result of 1,3 M tiles per second for 40 GACT arrays has to be divided by 40, thus obtaining 32500 tiles per second for 1 GACT array.

Multiplying 32500 tiles times 320 x 320, the tile size, Cell Updates Per Second are obtained as a result, since each tile has 320x320 cells:

$$32500\ tiles\ x\ (320\ x\ 320)\ cells\ = 3{,}328\ GCUPs \tag{1}$$

Thus, the original Darwin co-processor [1], with 32 Processing Elements, and the tile size set at *T*=320, results in a performance of 3,328 Giga cells update per second.

The actual kernel design, implemented on the Alveo U280, of the current project has been tested for 4 PEs and *T*=320, obtaining a result of 140 µs to compute a single tile.

To obtain the actual cell updates per second, the following calculation has been applied:

$$\frac{(320\ x\ 320)\ cells}{140\ \mu s} = 0{,}73\ GCUPs \tag{2}$$

Obtaining a result of 0,73 GCUPs for 4 Processing Elements. Since the original Darwin implementation has been made with 32 PEs, in order to be able to compare both performances, the result has to be multiplied by 8, thus giving a final result of 5,851 GCUPs.

The speedup obtained results in **1,75X**, from the original Darwin giving 3,328 GCUPs to the accelerated one giving **5,851 GCUPs**.

7. Conclusion and future works

This work presents a genomic co-processor for long read alignment, the HPC FPGA implementation of the GACT alignment algorithm.

Detailed results and analyses show significant performance acceleration exploiting High Bandwidth Memory. This project demonstrated runtime improvements of up to **1,75x** using the Xilinx Alveo U280, compared with the original Intel FPGA implementation.

The implementation has been done mapping six HBM memory banks of the target FPGA to the six AXI master interfaces of the kernel design, being just one Compute Unit (CU) in the design, so a single kernel. In order to increase even more the performance, future work can focus on implementing more Compute Units inside the FPGA, since the host file is already prepared for it. With more Compute Units the improvement can be even better than 1,75x with respect to the original GACT implementation, since more HBM memory banks can be exploited.

As future work, the seeding construction can be integrated on Hardware, which is something that Darwin does not perform yet.

The algorithms can be integrated within software used in the state of the art (Minimap, BWA-MEM, etc.) and within Darwin itself.

References

[1] Y. Turakhia, G.Bejerano, and W. J. Dally. "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly." *ACM SIGPLAN Notices,* vol. 53. ACM, 2018.

[2] Xilinx, 2020. *Alveo U280 Data Center Accelerator Card User Guide*. https://www.xilinx.com/support/documentation/boards_and_kits/acceleratorcards/
ug1314-u280-reconfig-accel.pdf.

[3] K.-M. Chao, W. R. Pearson, and W. Miller. "Aligning two sequences within a specified diagonal band". *Computer applications in the biosciences: CABIOS*, 8(5):481–487, 1992.

[4] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. "A greedy algorithm for aligning dna sequences". *Journal of Computational biology*, 7(1-2):203–214, 2000.

[5] G. Myers. "A fast bit-vector algorithm for approximate string matching based on dynamic programming". *Journal of the ACM (JACM),* 46(3):395–415, 1999.

[6] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. "Basic local alignment search tool". *Journal of molecular biology*, 1990.

[7] I. Sović, M. Šikić, A. Wilm, S. N. Fenlon, S. Chen, and N. Nagarajan. "Fast and sensitive mapping of nanopore sequencing reads with graphmap". *Nature communications*, 7, 2016.

[8] M. J. Chaisson and G. Tesler. "Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory". *BMC bioinformatics*, 13(1):238, 2012.

[9] G. Myers. "Efficient local alignment discovery amongst noisy long reads". *International Workshop on Algorithms in Bioinformatics*, pages 52–67. Springer, 2014.

[10] H. Li. "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem". *arXiv preprint arXiv*:1303.3997, 2013.

[11] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, and A. M. Phillippy. "Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation". *bioRxiv*, page 071282, 2016.

[12] K. Berlin, S. Koren, C.-S. Chin, J. P. Drake, J. M. Landolin, and A. M. Phillippy. "Assembling large genomes with single-molecule sequencing and locality-sensitive hashing". *Nature biotechnology*, 33(6):623–630, 2015.

[13] J. Buhler. "Efficient large-scale sequence comparison by locality sensitive hashing". *Bioinformatics*, 17(5):419–428, 2001.

[14] TimeLogic Corporation. URL http://www.timelogic.com.

[15] Y. Turakhia, G.Bejerano, and W. J. Dally. *Darwin Github*. https://github.com/yatisht/darwin

[16] Xilinx, 2021. *AXI High Bandwidth Memory Controller v1.0* https://www.xilinx.com/support/documentation/ip_documentation/hbm/v1_0/pg276-axi-hbm.pdf

[17] Y. Choi, Y. Chi, J. Want, L. Guo, and J. Cong, "When HLS Meets FPGA HBM: Benchmarking and Bandwidth Optimization". *arXiv:2010.06075v1 [cs.AR],* October 2020.

[18] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking High Bandwidth Memory on FPGAs". *IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020.

[19] Xilinx. 2020. *Vitis Unified Software Platform*. https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[20] ARM. 2011. *AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite*. www.arm.com

[21] H. Jun, J. Cho, K. Lee, H. Son, K. Kim, H. Jin, and K. Kim, "HBM (High Bandwidth Memory) DRAM technology and architecture". *Proc. IEEE Int, Memory Workshop*, 1-4, 2017.

[22] A. Zeni, G. W. Di Donato and F. Peverelli. *PALADIN Github*. https://github.com/albertozeni/XDropXOHW-Public