Darwin: A Genomics Co-processor Provides up to 15,000× acceleration on long read assembly

Yatish Turakhia
Stanford University
yatisht@stanford.edu

Gill Bejerano
Stanford University
bejerano@stanford.edu

William J. Dally
Stanford University
NVIDIA Research
dally@stanford.edu

Abstract

Genomics is transforming medicine and our understanding of life in fundamental ways. Genomics data, however, is far outpacing Moore's Law. Third-generation sequencing technologies produce 100× longer reads than second generation technologies and reveal a much broader mutation spectrum of disease and evolution. However, these technologies incur prohibitively high computational costs. Over 1,300 CPU hours are required for reference-guided assembly of the human genome (using [47]), and over 15,600 CPU hours are required for *de novo* assembly [57]. This paper describes "Darwin" — a co-processor for genomic sequence alignment that, without sacrificing sensitivity, provides up to 15,000× speedup over the state-of-the-art software for reference-guided assembly of third-generation reads. Darwin achieves this speedup through hardware/algorithm co-design, trading more easily accelerated alignment for less memory-intensive filtering, and by optimizing the memory system for filtering. Darwin combines a hardware-accelerated version of D-SOFT, a novel filtering algorithm, with a hardware-accelerated version of GACT, a novel alignment algorithm. GACT generates near-optimal alignments of arbitrarily long genomic sequences using constant memory for the compute-intensive step. Darwin is adaptable, with tunable speed and sensitivity to match emerging sequencing technologies and to meet the requirements of genomic applications beyond read assembly.

Keywords Hardware-acceleration, Co-processor, Genome assembly, Long reads.

ASPLOS'18, March 24–28, 2018, Williamsburg, VA, USA

ACM ISBN ISBN 978-1-4503-4911-6/18/03…$15.00
https://doi.org/http://dx.doi.org/10.1145/3173162.3173193

ACM Reference Format:
Yatish Turakhia, Gill Bejerano, and William J. Dally. 2018. Darwin: A Genomics Co-processor Provides up to 15,000× acceleration on long read assembly. In *Proceedings of 2018 Architectural Support for Programming Languages and Operating Systems (ASPLOS'18).* ACM, New York, NY, USA, 15 pages. https://doi.org/http://dx.doi.org/10.1145/3173162.3173193

1 Introduction

Since the completion of the first draft of the human genome in 2001 [46, 79], genomic data has been doubling every 7 months - far outpacing Moore's Law and is expected to reach exabyte scale and surpass Youtube and Twitter by 2025 [73]. This growth has been primarily led by the massive improvements in "genome sequencing" - technology that enables reading the sequence of nucleotide bases in a DNA molecule. Today, it is possible to sequence genomes on rack-size, high-throughput machines at nearly 50 human genomes per day [33], or on portable, USB-stick size sequencers that require several days per human genome [21]. This data explosion has been immensely valuable to the emerging field of personalized medicine [28] and in detecting when genomic mutations predispose humans to certain diseases, including cancer [64], autism [44] and aging [52]. It has also contributed to the understanding of the molecular factors leading to phenotypic diversity (such as eye and skin color, etc.) in humans [72], their ancestry [49] and the genomic changes that made humans different from related species [54, 80].

Third generation technologies produce much longer *reads* of contiguous DNA — tens of kilobases compared to only a few hundred bases in both first and second generation [67]. This drastically improves the quality of the *genome assembly*. For example, contiguity in the gorilla genome assembly was recently found to improve by over 800× using long reads [24]. For personalized medicine, long reads are superior in identifying structural variants i.e. large insertions, deletions and re-arrangements in the genome spanning kilobases of more which are sometimes associated with diseases; for haplotype phasing, to distinguish mutations on maternal versus paternal chromosomes; and for resolving highly repetitive regions in the genome [67].

Despite several advantages, the long read technology suffers from one major drawback - high error rate in sequencing. Mean error rates of 15% on Pacific Biosciences (PacBio)

technology and up to 40% on Oxford Nanopore Technology (ONT) have been reported [20, 23]. These errors are corrected using computational methods that can be orders of magnitude slower than those used for the second generation technologies. For instance, using BWA [47], aligning 54× coverage reads of the human genome to reference requires over 1,300 CPU hours on PacBio reads (15% error rate), compared to only 270 CPU hours for short Illumina reads (<2% error rate).

In this paper, we describe *Darwin*, a hardware-acceleration framework for genome sequence alignment, which accelerates the compute-intensive task of long-read assembly by orders of magnitude. Darwin can also perform other sequence-alignment tasks such as homology search [3] or whole genome alignments [30].

This paper makes the following contributions:

1. We introduce Diagonal-band Seed Overlapping based Filtration Technique (D-SOFT) which implements the filtering (or *seeding*) stage of the alignment in a manner that can be tuned to provide the required *sensitivity* while maintaining high *precision*. We also describe a hardware accelerator for D-SOFT that uses a memory system tailored for the algorithm.
2. We introduce Genome Alignment using Constant memory Traceback (GACT) - a novel algorithm using dynamic programming for aligning arbitrarily long sequences using constant memory for the compute intensive step. We also present a hardware accelerator for GACT. All previous hardware accelerators for genomic sequence alignment have assumed an upper-bound on sequence length (typically <10Kbp) [14, 59] or have left the trace-back step to software [22, 36], undermining the benefits of hardware-acceleration. We empirically show that GACT provides *optimal* alignments even for error rates up to 40%.
3. We describe an FPGA prototype of Darwin, the D-SOFT and GACT accelerators.
4. We describe the design of a Darwin ASIC in TSMC 40nm CMOS through place-and-route. The Darwin ASIC requires 412mm^2 area and 15W of power.
5. We evaluate Darwin on 3 read types, with error rates from 15% to 40%. In each case, we adjust Darwin's sensitivity to match or exceed that of state-of-the-art software. On reference-guided assembly, the Darwin ASIC achieves up to 15,000× speedup. For inferring overlaps between reads in *de novo* assembly, Darwin provides up to 710× speedup over state-of-the-art software. On FPGA, Darwin achieves up to 183.8× speedup for reference-guided assembly and up to 19.9× speedup for inferring overlaps in *de novo* assembly over state-of-the-art software.

The rest of the paper is organized as follows. Section 2 provides relevant background on genome sequence alignment, sequencing, and read assembly. The D-SOFT and GACT algorithms are described in Sections 3 and 4. Section 5 explains how Darwin combines D-SOFT and GACT to perform reference-guided and *de novo* assembly of long reads. Sections 6 and 7 describe the hardware design of the D-SOFT and GACT accelerators. Our experimental methodology is described in Section 8, and we present our results in Section 9. Section 10 discusses relevant related work and Section 11 concludes the paper.

2 Background

In this section, we provide some necessary background on genome sequence alignment, genome sequencing and genome assembly needed for the rest of this paper.

Genome sequence alignment: The sequence alignment problem can be formulated as follows: Given a query sequence $Q = q_1, q_2, .., q_n$ and a reference sequence $R = r_1, r_2, .., r_m$ ($m \geq n$), assign gaps (denoted by '-') to R and Q to maximize an alignment score. The alignment assigns letters in R and Q to a single letter in the opposite sequence or to a gap. For genomic sequences, the letters are in the alphabet $\Sigma = \{A, C, G, T\}$, corresponding to the four nucleotide bases.

Sequence alignment often uses the Smith-Waterman algorithm [69], since it provides optimal *local alignments* for biologically-meaningful scoring schemes. A typical scoring scheme rewards matching bases and penalizes mismatching bases, with these rewards and penalties specified by a substitution matrix W of size $\Sigma \times \Sigma$. The gaps in R and Q, known as *insertions* and *deletions*, are penalized by another parameter, *gap*. Smith-Waterman operates in two phases: *matrix-fill*, in which a score $H(i, j)$ and a traceback pointer is computed for every cell (i, j) of the dynamic programming (DP) matrix having dimensions $m \times n$ using the DP equations of Figure 1a, and *traceback*, which uses traceback pointers, starting from the highest-scoring cell, to reconstruct the optimal *local* alignment. Figure 1c shows an example DP-matrix computed using the parameters in Figure 1b for sequences R ='GCGACTTT' and Q ='GTCGTTT'. An arrow in each cell represents the traceback pointer, which points to the cell that led to its score, with no arrow for cells with score 0, where the traceback phase must terminate. Red arrows indicate that traceback path from the highest-scoring cell (highlighted using red) to reconstruct the optimal alignment shown in Figure 1d.

Because Smith-Waterman is computationally expensive, most alignment packages use a filtering step based on the *seed-and-extend* paradigm, made popular by BLAST [3], to reduce the work that must be done by Smith-Waterman. This approach uses *seeds*, substrings of fixed size k from Q, and finds their exact matches in R, called *seed hits*. Finding seed hits is $O(m)$ for each seed, but can be accelerated by preprocessing R to compute an index [3]. Once the seed hits are found, the extend step uses dynamic programming to

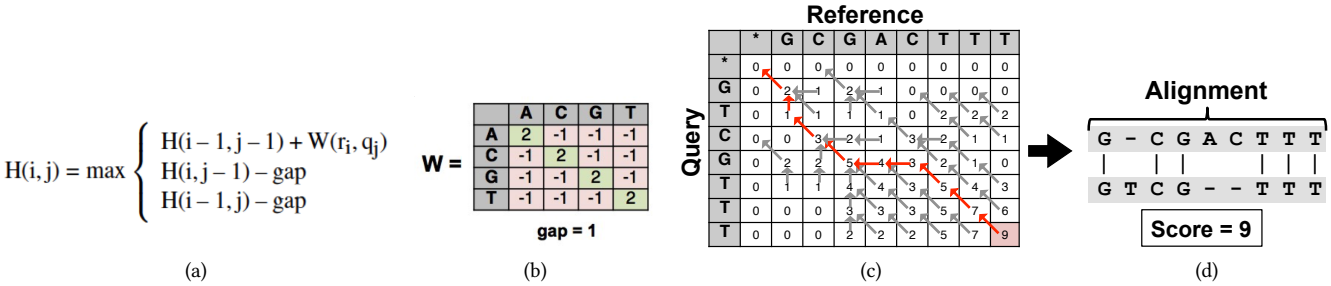


Figure 1: Overview of Smith-Waterman alignment algorithm. (a) Smith-Waterman equations. (b) Smith-Waterman scoring parameters (match=+2, mismatch=-1, gap=1). (c) DP-matrix with traceback pointers. (d) Optimal alignment.

compute DP-matrix scores for only the cells surrounding the seed hit, thereby avoiding the high cost of computing every cell of the DP-matrix. Alignments between R and Q having no exactly-matching substrings of size k or larger will not be discovered by this approach, resulting in lower *sensitivity*. The *seed-and-extend* paradigm trades sensitivity for speed but works well in practice and has been incorporated in a large number of alignment heuristics since BLAST [30, 39, 47, 48, 57, 71].

Genome sequencing and assembly: Genome/DNA sequencing is the process of shearing DNA molecules of the genome into a large number N of random fragments and reading the sequence of bases in each fragment. The size of the fragments depends on the sequencing technology. Long-read sequencing technologies, such as Pacific Biosciences and Oxford Nanopore Technologies (ONT), have a mean read length L of about 10Kbp. Typically, N is chosen such that each base-pair of the genome G is expected to be covered more than once, i.e. where *coverage* $C = (NL/G) > 1$. With high coverage, each base-pair is likely covered by several reads and small errors in each read can be corrected using a *consensus* of the reads. Despite an error-rate of 15% for each base pair, Pacific Biosciences reads have consensus accuracy of > 99.99% at a coverage of $C = 50$ [42].

Genome assembly is the computational process of reconstructing a genome G from a set of reads S. In *reference-guided* assembly a reference sequence R, which closely resembles the sequenced genome, is used to assist the assembly. Reference-guided assembly is $O(N)$ for N reads, produces fewer contigs compared to *de novo* assembly, and is good at finding small changes, or *variants*, in the sequenced genome. However, it can also introduce reference-bias and it does not capture well the large *structural variants* between reference and sequenced genome which are sometimes associated with disease [55].

In *de novo* assembly, G is constructed only from S, without the benefit (or bias) of R. The preferred approach to *de novo* assembly of long reads is using *overlap-layout-consensus*

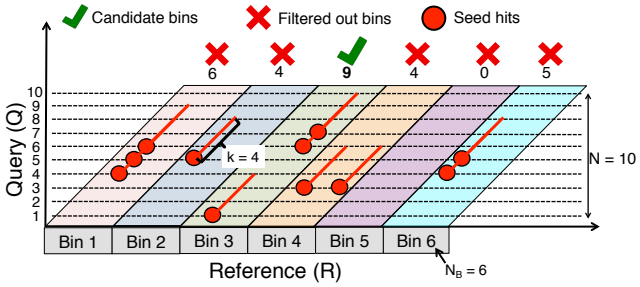


Figure 2: Illustration of D-SOFT algorithm for k=4, N=10, h=8, N_B=6.

(OLC) [50]. In the *overlap* phase, all read pairs having a significant overlap are found. During the *layout* phase, overlapping reads are merged to form larger contigs. Finally, in the *consensus* phase, the final DNA sequence is derived by taking a consensus of reads, which corrects the vast majority of read errors. By far, the most time-consuming step of OLC is the *overlap* step, which requires $O(N^2)$ time for N reads.

3 D-SOFT Filtering Algorithm

Motivation. For long reads with high error rates, simple *seeding* algorithms like BLAST [4] give a high false positive rate that results in excessive computation in the *extend* or alignment stage. To address this issue, [57] uses multiple seeds from Q, counting the number of matching bases in a band of diagonals and then filtering on a fixed threshold. We introduce D-SOFT, that uses this filtering strategy, but differs in its implementation. We highlight the advantages of D-SOFT versus [57] later in Section 10.

Description. Like [57], D-SOFT filters by counting the number of unique bases in Q covered by seed hits in diagonal bands. Bands in which the count exceeds h bases are selected for alignment. Figure 2 illustrates D-SOFT. The X- and Y-axes in the figure correspond to positions in R and Q, respectively. Similar to [57], R is divided into N_B *bins*, B bases wide ($N_B = \lceil m/B \rceil$). In Figure 2, N_B=6. Ten seeds with k=4 from Q are used from consecutive positions, starting at position 1. The seed start positions are marked by dotted horizontal lines.

For each seed, a red dot is marked on its horizontal line at the position of each hit in R. The "tail" of the red dot covers k=4 bases in R and Q, showing the extent of the match. The count over each diagonal band is equal to the number of bases in Q covered by seed hits in that band. For instance, seeds at positions 4, 5 and 6 in Q find a hit in bin 1 of R, but its count is 6 since the seed hits cover only 6 bases (4-9) in Q. Bin 3 also has 3 seed hits, but the hits cover 9 bases (1-4 and 5-9). With $h = 8$, only bin 3 is chosen as a candidate. However, if a seed hit count strategy is used, both bin 1 and bin 3 would have the same count (=3), even though bin 3, having more matching bases, has a better probability of finding a high-scoring alignment. Counting unique bases in a diagonal band allows D-SOFT to be more *precise* at high sensitivity than strategies that count seed hits [11, 71], because overlapping bases are not counted multiple times.

Algorithm 1: D-SOFT

```
1  candidate_pos ← [] ;
2  last_hit_pos ← [−k  for i in range(N_B)] ;
3  bp_count ← [0  for i in range(N_B)] ;
4  for j in start : stride : end do
5      seed ← Q[j : j + k];
6      hits ← SeedLookup(R, seed) ;
7      for i in hits do
8          bin ← ⌈(i − j)/B⌉ ;
9          overlap ← max(0, last_hit_pos[bin] + k − j);
10         last_hit_pos[bin] ← j;
11         bp_count[bin] ← bp_count[bin] + k − overlap ;
12         if (h + k − overlap > bp_count[bin] ≥ h) then
13             candidate_pos.append(< i, j >) ;
14         end
15     end
16 end
17 return candidate_pos;
```

As shown in Algorithm 1 D-SOFT operates using two arrays, *bp_count* and *last_hit_pos*, each of size N_B to store the total bases covered by seed hits and the last seed hit position for every bin, respectively. Seeds of size k are drawn from Q, with a specified *stride* between *start* and *end* (lines 4-5). For each seed, function *SeedLookup* finds all hits in R (line 6). For each *hit*, its *bin* and its *overlap* with the last hit in the bin is computed (lines 7-9). Non-overlapping bases are added to *bp_count* (line 11). When the *bp_count* for a bin first exceeds the threshold h, the last hit in the bin is added to a filtered list of candidate positions, *candidate_pos* (lines 12-13). This list is returned at the end of the algorithm (line 17) and is used as input to the GACT algorithm.

D-SOFT implements *SeedLookup* using a seed position table [30] as illustrated in Figure 3 for seed size k=3bp. For each of the 4^k possible seeds, a *seed pointer table* points to the beginning of a list of hits in a *position table*. In Figure 3,

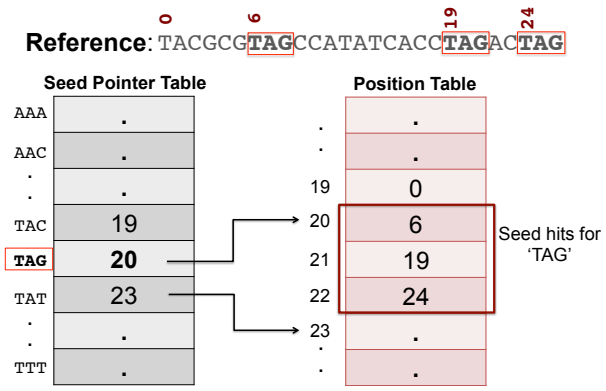


Figure 3: An example reference sequence and seed position table used in *SeedLookup*.

lookups to 'TAG' and 'TAT' in the pointer table give the start and end addresses in the position table for hits of 'TAG' in the reference. The seed position table has an advantage over suffix trees [27], or compressed tables based on Burrows Wheeler Transform [48], FM-index [45], etc in that the seed hits are stored sequentially, enabling faster and fewer memory accesses to DRAM.

4 GACT Alignment Algorithm

Motivation. Long-read technology requires the alignment of two long genomic sequences. To accelerate this alignment in hardware, the memory footprint must be kept small enough to be captured in on-chip SRAM. Smith-Waterman requires $O(mn)$ space to optimally align two sequences of lengths m and n. Hirschberg's algorithm [32] can improve the space complexity to linear $O(m+n)$, but is rarely used in practice because of its performance. As a result, heuristics, such as Banded Smith-Waterman [13], X-drop [84] and Myers bit-vector algorithm [56], that do not guarantee optimal alignments but have linear space and time complexity, have become popular. For long reads, these algorithms require prohibitive traceback pointer storage for hardware implementation.

Description. GACT performs the alignment extension part of a *seed-and-extend* system. It finds an arbitrarily long alignment extension in linear time by following the optimal path within *overlapping tiles*. GACT uses two parameters: a tile size T and an overlap threshold O ($< T$). Empirically, we have found that GACT gives an optimal result (identical to Smith-Waterman) with reasonable values of T and O.

GACT for left extension, aligning R and Q to the left of position i^* in R and j^* in Q, is shown in Algorithm 2. The positions (i^*, j^*) are provided by D-SOFT. GACT maintains the start and end positions of R (Q) in the current tile in i_{start} (j_{start}) and i_{curr} (j_{curr}), respectively. The alignment traceback path is maintained in tb_left. The function *Align* (line 7) uses a modified Smith-Waterman to compute the optimal alignment between R^{tile} and Q^{tile} with traceback

Algorithm 2: GACT for Left Extension

```
1  tb_left ← [] ;
2  (i_curr, j_curr) ← (i*, j*) ;
3  t ← 1 ;
4  while ((i_curr > 0) and (j_curr > 0)) do
5      (i_start, j_start) ← (max(0, i_curr − T), max(0, j_curr − T)) ;
6      (R^tile, Q^tile) ← (R[i_start : i_curr], Q[i_start : i_curr]) ;
7      (TS, i_off, j_off, i_max, j_max, tb) ←
         Align(R^tile, Q^tile, t, T − O);
8      tb_left.prepend(tb) ;
9      if (t == 1) then
10         (i_curr, j_curr) ← (i_max, j_max);
11         t ← 0;
12     end
13     if ((i_off == 0) and (j_off == 0)) then
14         break;
15     end
16     else
17         (i_curr, j_curr) ← (i_curr − i_off, j_curr − j_off)
18     end
19 end
20 return (i_max, j_max, tb_left);
```

starting from the bottom-right cell of the matrix except for the first tile (where t is true) where traceback starts from the highest-scoring cell. *Align* returns the tile alignment score TS, the number of bases of R and Q consumed by this tile (i_{off}, j_{off}), traceback pointers tb, and (for the first tile) the position of the highest-scoring cell (i_{max}, j_{max}). *Align* restricts i_{off} and j_{off} in the traceback to at most $(T - O)$ bases to ensure successive tiles overlap by at least O bases in both R and Q. GACT left extension terminates when it reaches the start of R or Q (line 4), or when the traceback results cannot further extend the alignment (line 13). *Align* is the compute-intensive step of GACT and has a constant traceback memory requirement, $O(T^2)$, that depends only on the tile size T, and not on the length of sequences being aligned. This facilitates compact hardware design. The full traceback storage tb_left is linear in read length n but is not performance critical. For right extension, GACT works similar to Algorithm 2, but using reverse of R and Q.

Figure 4 shows GACT left extension on the example from Figure 1 with T=4 and O=1. GACT begins traceback from the highest-scoring cell (green) of the first tile (T1). Subsequent tiles begin traceback from the bottom-right cell (yellow) in the tile, where the traceback of previous tile terminates. This GACT alignment is the same as the optimal alignment of Figure 1d.

5 Darwin: System Overview

Darwin is a co-processor for genomic sequence alignment that combines hardware-accelerated GACT and D-SOFT algorithms. As shown in Figure 5a, Darwin accelerates D-SOFT

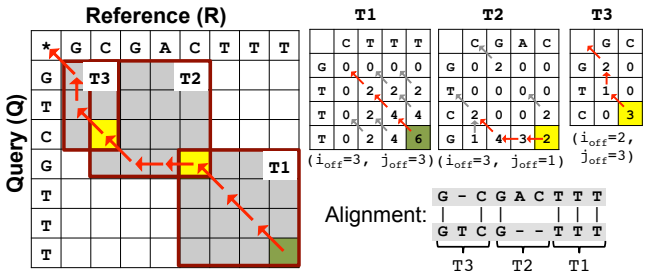


Figure 4: Illustration of left extension in GACT algorithm using an example dynamic programming (DP) matrix for parameters (T=4, O=1).

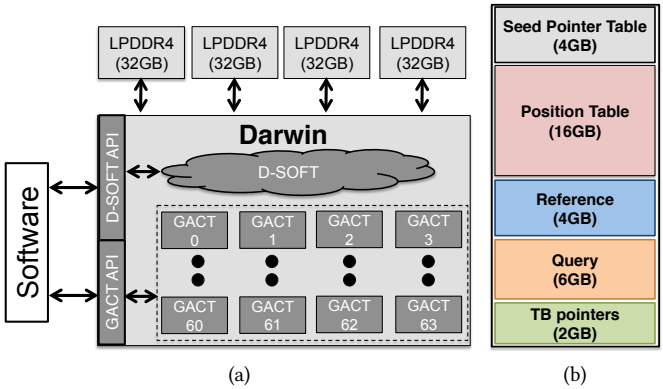


Figure 5: (a) Overview of Darwin's hardware. (b) DRAM memory space organization.

algorithm for filtering and uses 64 independent GACT arrays for alignments. Both D-SOFT and GACT parameters are programmable through software.

Darwin uses four 32GB LPDDR4 DRAM memory channels that store identical copies of the seed position table, the reference sequence, and the query sequences (Figure 5b). Replicating data structures helps keeping all DRAM channels load-balanced and fully-utilized at all times. A 4GB seed pointer table allows seed size $k \leq 15$ (4B pointer each for 4^k seeds), a 16GB position table stores positions for up to 4Gbp reference (larger reference can be divided across the banks or split into blocks). The 4GB reference and 8GB query partitions store up to a 4Gbp reference sequence and up to 8Gbp of query sequences (reads) in ASCII format. GACT traceback (TB) pointers (tb_left) are written in a 2GB space.

Figure 6 shows how Darwin is used for both reference-guided and *de novo* assembly. In both cases, the forward and reverse-complement of P reads {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. In *de novo* assembly the reads are concatenated to form the reference sequence with each read padded to take up a whole

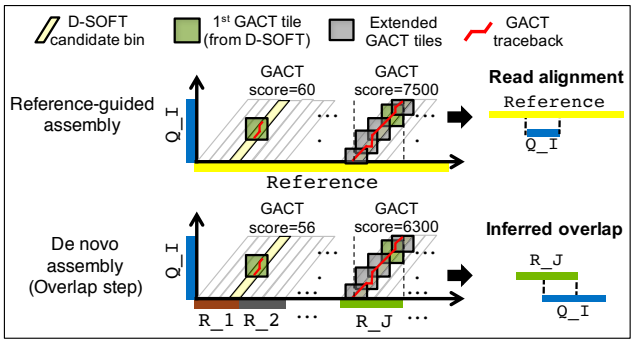


Figure 6: Reference-guided and *de novo* assembly (overlap step) using D-SOFT and GACT.

number of bins. 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. To filter out most D-SOFT false positives early, Darwin uses a minimum threshold h_{tile} for the alignment score of the first GACT tile (see Section 9). In D-SOFT, Darwin uses B = 128, $stride$ = 1 and discards high frequency seeds i.e. seeds occurring more than $32 \times |R|/4^k$, where $|R|$ is the size of the reference sequence.

The assembly algorithms are partitioned between hardware and software as follows. Software initializes D-SOFT and GACT parameters and receives candidate alignment positions from Algorithm 1 (D-SOFT), which is accelerated in hardware (Section 6). *Align* - the compute-intensive step in Algorithm 2 (GACT) - is hardware-accelerated (Section 7), the remainder of the alignment step is performed in software. Software also constructs the seed position table in *de novo* assembly.

6 D-SOFT Accelerator Design

The D-SOFT accelerator (Figure 7) is organized to optimize the memory references of Algorithm 1. D-SOFT software spends 70-90% of its runtime updating the bin arrays, *bp_count* and *last_hit_pos* (lines 10-11 in Algorithm 1) because these are largely random memory references to tables too big to fit in an LLC. In contrast *SeedLookup* is fast because it makes mostly sequential memory accesses. To optimize performance and energy, the D-SOFT accelerator maps the sequentially-accessed seed tables to off-chip DRAM and places the randomly-accessed bin-arrays in banks of on-chip SRAM. To provide flexibility threshold h and bin size B (power of 2) are programmable via software-accessible registers.

For each query sequence, software initiates D-SOFT by feeding N seed-offset pairs $(seed, j)$ to the accelerator. Incoming seeds are distributed over the DRAM channels. A seed-position lookup (SPL) block takes each seed, looks up its pointer in the seed-pointer table, reads the *hits* from the position table and calculates the *bin* of each seed hit. Bin-offset pairs for each hit (bin, j) are sent, via a NoC, to an on-chip SRAM bank (Bin-count SRAM) associated with the bin where *bp_count* and *last_hit_pos* are updated by the update-bin logic (UBL). When *bp_count* first exceeds the programmed threshold h, the *bin* is enqueued in the candidate FIFO. Whenever *bp_count* is first incremented to a non-zero value, the *bin* is enqueued in a NZ queue to aid in the clear operation between queries.

Darwin uses 2B (16b) for each bin âĂŞ– a 5b saturating counter for *bp_count* and 11b for *last_hit_pos*. The bin count SRAMs have a total capacity of 64MB divided into 16 banks to enable parallel update. This capacity allows up to N_B=32M bins to accommodate a 4Gbp reference sequence with B=128.

The NoC uses a butterfly topology [17] with 16 endpoints and requires 4 hops to route each (bin, j) pair to its destination bank. To preserve sequential ordering between *hits* corresponding to different offsets (j), the NoC waits for all *bins* corresponding to one seed to arrive at the destination before moving to the next seed. Because seeds are interleaved over the DRAM channels, the hits associated with each seed are located in separate per-channel FIFOs.

7 GACT Accelerator Design

The *GACT array* accelerates the compute-intensive *Align* routine in GACT (Section 4). The rest of Algorithm 2, is implemented in software. As shown in Figure 8a, the GACT array consists of a systolic array of processing elements (inspired by [51]), one traceback pointer SRAM per PE, and traceback logic. Our current implementation uses N_{pe} = 64 and T_{max} = 512, which requires 128KB SRAM for traceback memory per GACT array, with each of the 64 PEs connected to one 2KB SRAM bank.

Before starting operation, the GACT array is configured by loading 18 16-bit scoring parameters (16 parameters for matrix W and one parameter each for gap open o and gap extend e) and the tile size T. For each call to *Align*, the software provides coordinates i and j into R and Q respectively, and a bit t that indicates the first tile of a query. The GACT array accepts ASCII inputs for R and Q, but internally stores the sequences using 3-bits for an extended DNA alphabet $\Sigma_{ext} = \{A, C, G, T, N\}$, where N represents an unknown nucleotide and does not contribute to the alignment score.

Depending on whether t is set (unset), the GACT array outputs the score and coordinate (i^*, j^*) of the highest-scoring (bottom-right) cell of the tile a series of traceback pointers for the optimal alignment in the tile from the highest-scoring (bottom-right) cell. Each pointer is encoded with 2-bits to indicate an insert, a delete or a match.

A linear array of N_{pe} processing elements (PEs) exploits *wavefront parallelism* to compute up to N_{pe} cells of the DP-matrix for the Smith-Waterman algorithm with affine gap penalties [25]. Each clock cycle, each PE computes three

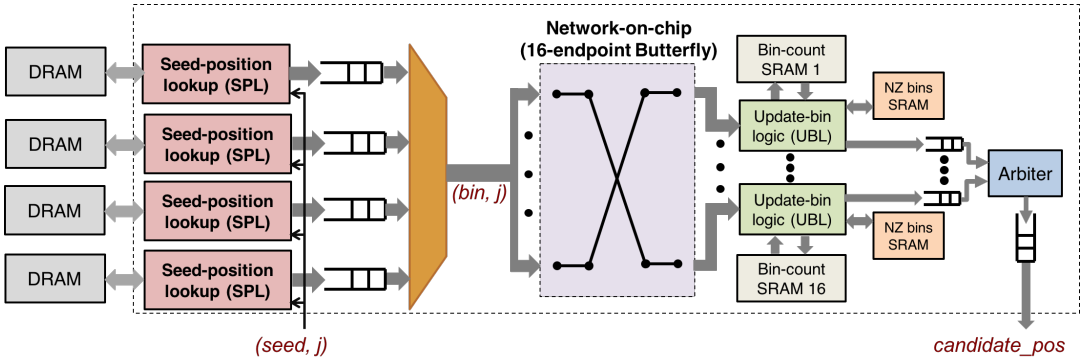


Figure 7: D-SOFT hardware design.

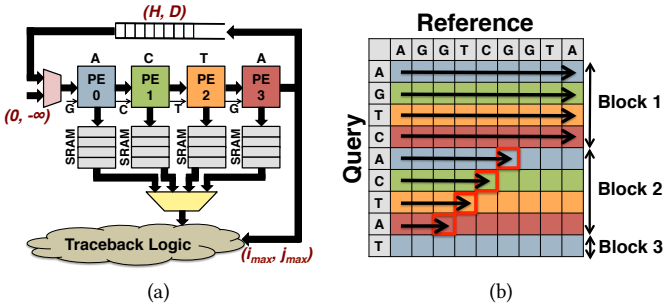


Figure 8: GACT hardware design. (a) Systolic architecture of GACT array with N_{pe}=4. (b) Query blocking for T=9.

scores and one traceback pointer for one cell of the DP-matrix. The three scores are:

$$I(i,j) = \max \begin{cases} H(i,j-1) - o \\ I(i,j-1) - e \end{cases} \quad (1)$$

$$D(i,j) = \max \begin{cases} H(i-1,j) - o \\ D(i-1,j) - e \end{cases} \quad (2)$$

$$H(i,j) = \max \begin{cases} 0 \\ I(i,j) \\ D(i,j) \\ H(i-1,j-1) + W(r_i, q_j) \end{cases} \quad (3)$$

Equations 1 and 2 compute the affine gap penalty for insertions I and deletions D respectively. Equation 3 computes the overall score for the cell. A four-bit traceback pointer is also computed for each cell: 1 bit each for equations 1 and 2 to indicate whether the insertion (deletion) score resulted from opening or extending a gap, and 2 bits for equation 3 indicating how the final score was arrived at — null (terminating), horizontal, vertical, or diagonal cell. When t is set, each PE maintains the maximum score and corresponding position for the cells it has computed. On completion, the global maximum score and position is computed in a systolic fashion.

Figure 8a shows an example GACT array with $N_{pe} = 4$ PEs computing the DP-matrix of Figure 8b, that corresponds to a single GACT tile with $T = 9$. For $T > N_{pe}$, the rows of the DP-matrix are divided into a number of query blocks, where a block consists of up to N_{pe} rows, one row for each PE. During the processing of one query block, each PE is initialized with the query base-pair corresponding to its row in the block. The reference sequence is then streamed through the array, shifting one base-pair every cycle. The systolic array architecture helps exploit the wave-front parallelism available in the computation of the query block. A FIFO is used to store the H and D scores of equations 3 and 2 of the last PE of the array, which gets consumed by the first PE during the computation of the next query block. The depth of this FIFO is T_{max}, corresponding to the maximum number of columns in the DP-matrix.

A memory of $4T_{max}^2$ bits stores the traceback pointers. This memory is divided into N_{pe} independent single-port SRAM banks, one bank for every PE as shown in Figure 8a. Each cycle, each PE pushes the 4-bit traceback pointer for the current cell, into this RAM.

The traceback (TB) logic starts at the bottom-right or highest-scoring cell in the DP array and traces back the optimal alignment by following the traceback pointers stored in the per PE SRAMs. It takes three cycles to traverse each step of the path: address computation, SRAM read, and pointer computation. Each cycle generates two bits that indicate the next step of the path: left, up, diagonal, or terminate. As it follows the traceback path, the TB logic also computes offsets i_{off} and j_{off} by keeping track of the number of bases of R and Q consumed along the optimal path. The traceback terminates when either: a terminating pointer is encountered, i_{off} or j_{off} reaches $(T - O)$ bases, or an edge of the DP-matrix (first row or column) is reached. As each 2-bit pointer is produced, it is pushed in a FIFO to be stored off-chip.

8 Experimental Methodology

Reference genome and read data. We used the latest human genome assembly GRCh38 using only the nuclear chromosomes (chromosomes 1-22, X, and Y) (removing the mitochondrial genome, unmapped and unlocalized contigs).

Read type	Substitution	Insertion	Deletion	Total
PacBio	1.50%	9.02%	4.49%	15.01%
ONT_2D	16.50%	5.10%	8.40%	30.0%
ONT_1D	20.39%	4.39%	15.20%	39.98%

Table 1: Error profile of the reads sets.

The resulting assembly size is 3.08Gbp. We generate three sets of reads using PBSIM [62] with length of 10Kbp each, coverage of 30×, and error rates of (15%, 30% and 40%): to match the error profiles of Pacific Biosciences (PacBio), Oxford Nanopore 2D reads (ONT_2D) and Oxford Nanopore 1D reads (ONT_1D), respectively. Default settings of continuous long read (CLR) model were used in PBSIM for PacBio reads at an error rate of 15.0%, while the PBSIM settings for ONT_2D and ONT_1D reads were taken from a previous study [71]. Table 1 shows the error profile for the three sets of reads. These correspond roughly to the error profiles observed using P6-C4 chemistry for PacBio reads, and R7.3 chemistry for Oxford Nanopore reads. Older Oxford Nanopore chemistry is intentionally used to demonstrate programmability for high-error reads. For *de novo* assembly, we used PBSIM to generate PacBio reads for 30× coverage of the WS220 assembly of the *C. elegans* genome, resulting in 3.0Gbp of raw reads. Using these raw reads, we tuned the settings in Darwin to exceed the sensitivity of baseline algorithm for finding overlaps and estimated Darwin's performance for 54× coverage of the human genome using 3Gbp read blocks.

Comparison baseline. All software comparison baselines were run with a single thread on a dual socket Intel Xeon E5-2658 (v4) CPU operating at 2.2GHz with 128GB DDR4 DRAM clocked at 2133MHz. A single thread on this processor has a power consumption of about 10W (measured using Intel's PCM power utility [35]), which we found to be the best iso-power comparison point to ASIC. Using all 32 threads on this processor gives 13× speedup for BWA-MEM [47] but at many times (7×) the power consumption. D-SOFT and GACT throughput on Darwin were compared against a hand-optimized software implementation of the algorithm, compiled with `-O4` optimization level using the `g++-5.4` compiler. GACT speedup was also compared to Edlib [70] (version 1.1.2) with `-p` option for enabling traceback path computation. Edlib uses a more restrictive Levenshtein distance based scoring for finding alignments, but is the fastest software aligner to our knowledge. Edlib throughput was evaluated on PacBio reads, and the algorithm is slower on more noisy ONT reads. GACT alignment scores for different (T, O) settings were compared to the optimal Smith-Waterman algorithm implemented using SeqAn library [18] (version 2.2.0) with 200,000 10Kbp reads for each of the three read types. A simple scoring scheme (match=+1, mismatch=-1, gap=1) was used in both GACT and software alignments.

For reference-guided assembly of PacBio reads, Darwin was compared to BWA-MEM [47] (version 0.7.14) using its `-x pacbio` flag. For Oxford Nanopore reads, we used GraphMap [71] (version 0.5.2), a highly-sensitive mapper optimized for Oxford Nanopore reads with default mode. We used DALIGNER [57] as a baseline technique for *de novo* assembly of long reads. We tuned D-SOFT's parameters to match or exceed the sensitivity of each baseline technique. We define sensitivity and precision as:

$$Sensitivity = \frac{TP}{TP + FN} \tag{4}$$

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

where TP, FP and FN are the number of true positives, false positives and false negatives, respectively. For reference-guided assembly, a true positive is when the read gets aligned to the reference within 50bp of the region of the ground-truth alignment reported by PBSIM. For *de novo* assembly, an overlap between two reads is considered a true positive if the two reads overlap by at least 1Kbp according to PBSIM, and if at least 80% of that overlap is detected by the algorithm under consideration. To evaluate the filtration ability of D-SOFT, we have also defined false hit rate as the average number of false hits (candidate alignment locations) for every true positive resulting after D-SOFT filtration, without using GACT.

ASIC Synthesis, Layout and Frequency. We performed ASIC synthesis using Synopsys Design Compiler (DC) [75] in a TSMC 40nm CMOS process. We used tcbn401pbwp standard cell library at the worst-case process-voltage-temperature (PVT) corner. Placement and routing were carried out using Synopsys IC compiler (ICC) [74] with 7 metal layers.

Frequency, power, and area of the ASIC are entirely determined by the SRAMs. For GACT, we synthesized one array with $N_{pe} = 64$. D-SOFT logic was separately synthesized, with the NoC generated using CONNECT [63]. SRAM memory was excluded during DC synthesis of GACT and D-SOFT. Cacti [68] was used to estimate the area and power of the SRAM memory. We generated a memory trace using a software run of D-SOFT and GACT and used Ramulator [40] and DRAMPower [12] to estimate DRAM timing and power respectively. Since DRAMPower [12] does not model LPDDR4, we estimated DRAM power conservatively using LPDDR3.

GACT throughput. We implemented a single GACT array on Xilinx Kintex 7 FPGA [1], clocked at 250MHz, to gather cycle counts for aligning sequences of different lengths and different (T, O) settings. Using the cycle counts, GACT array throughput on ASIC was estimated by scaling to ASIC frequency. Darwin uses 64 independent GACT arrays, and it's throughput scales linearly. We used Ramulator [40], with memory trace generated from a software run, to estimate memory cycles consumed at peak throughput.

	Component	Configuration	Area (mm^2)	Power (W)
GACT	Logic	64 × (64PE array)	17.6	1.04
	TB memory	64 × (64 × 2KB)	68.0	3.36
D-SOFT	Logic	2SPL + NoC+ 16UBL	6.2	0.41
	Bin-count SRAM	16 × 4MB	300.8	7.84
	NZ-bin SRAM	16 × 256KB	19.5	0.96
DRAM	LPDDR4-2400	4 × 32GB	-	1.64
Total			**412.1**	**15.25**

Table 2: Area and Power breakdown of individual components in Darwin. The critical path of Darwin is 1.18ns.

Each LPDDR4-2400 channel in Darwin is shared by 16 GACT arrays.

D-SOFT throughput. D-SOFT throughput is measured in seeds per second, which depends on seed size k. We collected memory traces for *SeedLookup* in D-SOFT using a software run for each k and used Ramulator [40] to estimate throughput accounting for memory cycles used by GACT arrays at peak throughput. The FPGA implementation of D-SOFT confirmed that the NoC and bin-count SRAM banks consume hits faster than the DRAM channels produce them, so that D-SOFT throughput is entirely memory-limited.

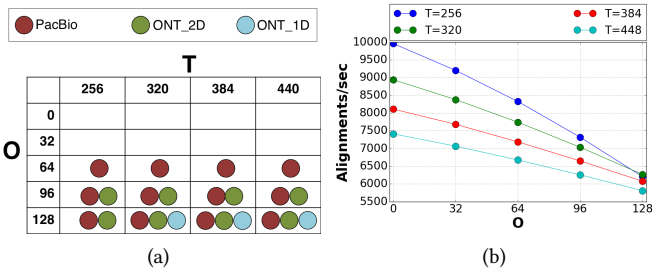
Darwin's performance on assembly. To estimate the ASIC performance of Darwin on read assembly, the assembly was first carried out on a software implementation of D-SOFT and GACT to determine D-SOFT parameters that match or exceed the sensitivity of the baseline algorithm and to compute statistics on the number of tiles to be processed by GACT and the number of seeds to be processed by D-SOFT. Combining these statistics with the previously derived ASIC throughput of GACT and D-SOFT, assembly time for Darwin was estimated using the slower of the two algorithms.

Darwin: FPGA performance. To estimate peak achievable FPGA performance, we synthesized GACT arrays on a single Intel Arria 10 [34] FPGA using the Microsoft Catapult [65] framework. Because of limited on-chip memory on FPGA, we used only 4 GACT arrays with traceback memory (remaining arrays were used only for single-tile GACT filtering) and implemented D-SOFT in software.

9 Results and Discussion

Darwin: Area, Frequency and Power. Table 2 shows the area and power breakdown of the different components of the Darwin ASIC. The critical path of Darwin has a delay of 1.18ns, enabling operation at 847MHz (3.4× the FPGA frequency). While Darwin is large in area, about 412mm^2, it's power consumption (including DRAM) is relatively small, about 15W. This is despite being implemented in a 40nm process. In a modern 14nm process, Darwin would be much smaller (about 50mm^2) and have much lower power (about 6.4W).

GACT: Performance and Throughput. Figure 9a shows the (T, O) settings in GACT for different reads types



(a) (b)

Figure 9: (a) (T, O) settings of GACT for different read types for which all 200,000 observed alignments were optimal. (b) Throughput of a single GACT array for pairwise alignment of 10Kbp sequences for different (T, O) settings.

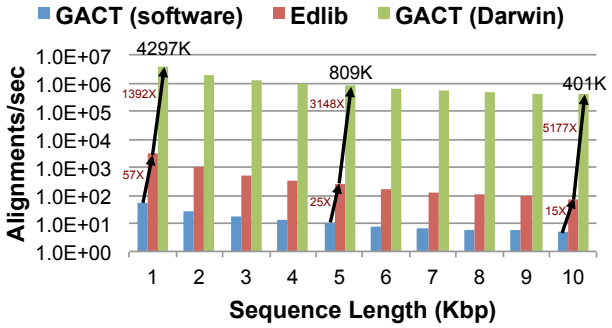
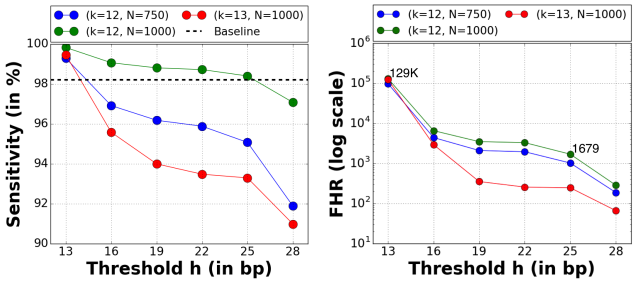


Figure 10: Throughput (alignments/second) comparison for different sequence lengths between a software implementation of GACT, Edlib library and the hardware-acceleration of GACT in Darwin.

for which all 200,000 observed alignments of 10Kbp reads to corresponding reference position are optimal. In general, we have observed that the deviations of GACT alignments from optimal arise near the edges of the tile, where the traceback path can be often affected by the DP-matrix cells exterior to the tile. Terminating traceback early in a tile by increasing O and recomputing the traceback pointers in the overlapping cells of the subsequent tile improves the GACT alignment scores. GACT provided optimal alignments for PacBio reads for $O \geq 64$. For $T \geq 320$ and $O = 128$, GACT produced optimal alignments for all three read types.

Figure 9b shows the throughput of a single GACT array in aligning 10Kbp sequences for different (T, O) settings. Since the number of GACT tiles for a fixed length alignment varies inversely to $(T-O)$, and each tile computation requires cycles proportional to T^2, the throughput varies proportional to $(T-O)/T^2$. We used (T=320, O=128) setting of GACT for the rest of the results in this paper since it provided empirically optimal alignments for all read types at highest throughput.

Figure 10 compares the throughput (in alignments per second) of the 64 GACT arrays on Darwin to Edlib [70] for



(a) (b)

Figure 11: Tuning (a) sensitivity and (b) false hit rate (FHR) of D-SOFT using threshold h for different settings of (k, N) for ONT_2D reads.

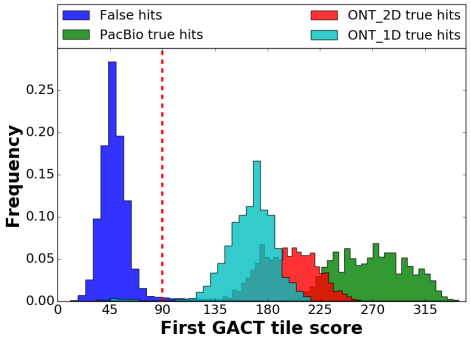


Figure 12: Histogram of the first GACT tile score (T = 384) for D-SOFT false hits and true hits of different read sets.

alignments of different sequence lengths. The figure shows that the throughput of GACT on Darwin (or in software) indeed scales inversely with the length of the sequences aligned — the throughput is reduced by roughly 10×, from 4,297,672 (54.3) to 401,040 (4.9) alignments per second when the sequence length increased by a factor of 10, from 1Kbp to 10Kbp. In comparison, Edlib runtime is quadratic in sequence length, and as a result, the speedup of Darwin with respect to Edlib increases with sequence length — from 1392× for 1Kbp sequences to 5177× for 10Kbp sequences. Moreover, unlike Edlib, GACT has a flexible scoring scheme based on [25].

At peak throughput, 64 GACT arrays in Darwin process 20.8M tiles per second each requiring two 320B sequential reads for R^{tile} and Q^{tile}, and one 64B write for storing the traceback path. This consumes 44.4% of the cycles of each of the 4 LPDDR4 memory channels. Darwin achieves roughly 80,000× speedup using hardware-acceleration of GACT versus the software implementation of GACT.

D-SOFT: Throughput and Sensitivity/Precision. Table 3 compares the throughput of D-SOFT on Darwin and its software implementation for different seed sizes (k) using the seed position table for human genome assembly (GRCh38). As k increases, the average number of hits per seed decreases, resulting in higher throughput. However,

Size (k)	hits/seed (GRCh38)	Throughput (Kseeds/sec)		
		Software	Darwin	Speedup
11	1866.1	16.6	1,426.9	85×
12	491.6	66.2	5,422.6	82×
13	127.3	259.3	19,081.7	73×
14	33.4	869.5	55,189.2	63×
15	8.7	2,257.1	91,138.7	40×

Table 3: Average number of seed hits for different seed sizes (k) and throughput comparison of D-SOFT on Darwin and its software implementation using human genome (GRCh38) for seed position table. 45% of available memory cycles in Darwin are reserved for GACT.

benefits of hardware-acceleration of D-SOFT are highest for short seeds with their longer sequential accesses to the seed position table.

Darwin's speedup on D-SOFT is due to three optimized memory access factors. First, performing bin updates to on-chip SRAM results in many times fewer *random* accesses to DRAM. Second, the hardware has longer sequential accesses to DRAM, resulting in higher effective bandwidth. Finally, the hardware accesses multiple DRAM channels in parallel. In our FPGA prototype, D-SOFT throughput is always limited by the DRAM bandwidth and not the on-chip bin updates, which occur at the rate of 5.1 updates/cycle (64% of theoretical maximum).

Figure 11 shows the sensitivity and precision (false hit rate (FHR)) of the D-SOFT filter for varying threshold (h), seed size (k), and number of seeds (N) in the reference-guided assembly of ONT_2D reads. Tuning D-SOFT parameters appropriately allows the filter to be orders of magnitude more precise (lower FHR), even at high sensitivity. Parameters k and N provide coarse-grain adjustment of both sensitivity and FHR while parameter h provides fine-grain control.

Figure 12 shows that it is easy to filter out a vast majority of D-SOFT's false hits using a simple threshold (h_{tile}) on the first GACT tile score itself — with h_{tile} = 90, 97.3% of D-SOFT's false hits can be filtered out after the first GACT tile itself, with less than 0.05% additional loss in sensitivity over D-SOFT. A larger tile size T = 384 is used for the first tile for easier distinction of true hits from false hits.

Darwin: Overall Performance. Table 4 compares Darwin's performance with baseline algorithms for reference-guided assembly of the human genome using 50,000 randomly sampled reads for each of the three read sets. We adjusted the D-SOFT parameters in Darwin to match or exceed the sensitivity of the baseline technique and still achieve over 1,000× speedup for each read type.

Figure 13 shows how Darwin achieves speedup by trading less work in the filtration stage — which is memory limited

Reference-guided assembly (human)							
Read type	D-SOFT	Sensitivity		Precision		Reads/sec	
	(k, N, h)	Baseline	Darwin	Baseline	Darwin	Baseline	Darwin
PacBio	(14, 750, 24)	95.95%	+3.76%	95.95%	+3.96%	3.71	9,916×
ONT_2D	(12, 1000, 25)	98.11%	+0.09%	99.10%	+0.22%	0.18	15,062×
ONT_1D	(11, 1300, 22)	97.10%	+0.30%	98.20%	+0.72%	0.18	1,244×
De novo assembly (*C. elegans*)							
Read type	D-SOFT	Sensitivity		Precision		Runtime (sec)	
	(k, N, h)	Baseline	Darwin	Baseline	Darwin	Baseline	Darwin (Speedup)
PacBio	(14, 1300, 24)	99.80%	+0.09%	88.30%	+1.80%	47,524	123×
De novo assembly (human)							
Read type	D-SOFT	Estimated Runtime (hours)					
	(k, N, h)	Baseline		Darwin (Speedup)			
PacBio	(14, 1300, 24)	15,600		710×			

Table 4: Comparison of Darwin with baseline techniques on reference-guided and *de novo* assembly. Darwin values are relative to the baseline technique.

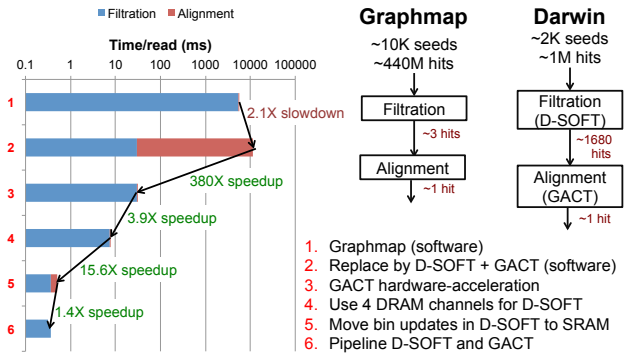


Figure 13: Timing breakdown between filtration and alignment stages for reference alignment of a single ONT_2D read in a series of steps going from Graphmap to Darwin.

and hence harder to accelerate – for more work in the alignment phase — which is easily hardware accelerated. The figure shows time on a log scale and shows the evolution from Graphmap (top) to Darwin (bottom). Graphmap runtime is dominated (99%) by the filtration stage. It spends enormous effort on filtration to avoid alignment — which is expensive in software. The next line shows Darwin implemented in software. Using fewer seeds and a smaller seed position table, D-SOFT achieves 182× speedup over Graphmap filter in software, but allows 560× more false hits. Even with linear-time GACT, doing alignment to filter the false hits takes 99.7% of the runtime and results in a 2.1× slowdown compared to Graphmap. Line 3 shows that accelerating GACT with 64 GACT arrays of 64 PEs each makes the alignment time negligible and gives a speedup of 380× compared to line 2. Using four DRAM channels (line 4) and performing bin updates in SRAM (line 5) together provide 61× acceleration. Darwin also pipelines D-SOFT and GACT for multiple reads, which adds another 1.4× speedup. Trading less filtering (D-SOFT) for more dynamic programming (GACT) is worse for software but greatly benefits hardware-acceleration, where the costs for dynamic programming, which requires few memory references, are significantly lower. This algorithm-architecture co-optimization is central to the speedup of Darwin.

Table 4 also shows the comparison of Darwin and DALIGNER [57] for finding pairwise overlaps between PacBio reads for *de novo* assembly of a 30× *C. elegans* genome. DALIGNER detected 99.8% overlaps in 47,524 seconds, and in comparison, Darwin required only 385 seconds to find 99.89% overlaps, nearly 123× speedup over DALIGNER. Of this, 370 seconds were taken by the software for construction of the seed position table and 15 seconds were required by Darwin to find overlaps. Since 54× coverage of human genome (168Gbp) would require the seed position table for read blocks of size 3Gbp to be constructed 56 times, and for each block, require 56× more reads to be processed, we estimate the overlap step in *de novo* assembly of 54× human genome to require roughly 22 hours on Darwin. In comparison, DALIGNER is estimated to require 15,600 CPU hours [57] for the same task. This implies a 710× speedup using Darwin over software. We expect that accelerating the construction of the seed position table in hardware can substantially improve these results but leave that optimization for future work.

Darwin: FPGA performance. We synthesized 40 GACT arrays of 32 PEs (4 arrays with traceback support), along with the Shell components of the Catapult framework [65] on the Arria 10 FPGA clocked at 150MHz. GACT arrays on FPGA provided a peak throughput of 1.3M tiles per second with T=320 (16× slower than ASIC) and D-SOFT was found to be the bottleneck in all cases. Speedup for reference-guided assembly ranged from 38.4× for ONT_1D reads to 183.8×

for ONT_2D reads. For *de novo* assembly on PacBio reads, speedup is estimated at around 19.9× to DALIGNER.

10 Related Work

Alignment heuristics and hardware-acceleration. Other heuristic approximations to the Smith-Waterman algorithm for pairwise alignment of sequences include Banded Smith-Waterman [13], X-drop [84] and Myers bit-vector algorithm [56]. Navarro et. al [58] have provided a detailed survey of algorithms used for approximate string matching, with applications ranging from genome sequence analysis to text retrieval. 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. GACT 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, GACT alignments, with sufficient overlap, are empirically optimal.

Acceleration of sequence alignment based on existing algorithms has been well explored in hardware architecture and compiler communities. Hardware architectures that accelerate alignment of arbitrarily long sequences only handle the matrix-fill step of Smith-Waterman algorithm. Lipton et. al [51] were the first to propose a systolic array exploiting wavefront parallelism in the matrix-fill step. A number of papers [15, 22, 36, 76, 81, 82] have implemented variants of this architecture, particularly on FPGAs. They all leave the traceback step to software, which would require the software to recompute the score matrix around high-scoring cells, undermining the benefits of hardware acceleration. Architectures that also accelerate the traceback step [14, 59, 61, 78] restrict the length of sequences they align based on available on-chip memory for storing traceback state. For instance, Nawaz et. al [59] accelerate Smith-Waterman in hardware that can align reads up to 1Kbp to a reference of up to 10Kbp, but requires 2.5MB of traceback memory. In comparison, the proposed GACT architecture requires only 128KB, and aligns arbitrarily long sequences *optimally*. Compiler assisted high level synthesis of systolic arrays has been studied in [10, 26]. This work is orthogonal to GACT, but GACT hardware synthesis could potentially improve from the proposed techniques.

Filtration heuristics. D-SOFT belongs to a category of filtration heuristics based on the *seed-and-extend* paradigm, first made popular by BLAST [3], and subsequent filtration techniques based on counting the number of seed hits conserved in a band of diagonals. These include two-hit BLAST [4], GraphMap [71], and BLASR [11].

DALIGNER [57] was the first technique to directly count the bases in the seed hits of a diagonal band, so overlapping bases in the hits are not multiply counted. This allows DALIGNER filtration to be more precise even at high sensitivity. This algorithm inspired D-SOFT. D-SOFT and DALIGNER differ in their implementation. DALIGNER uses sort and merge operations on large pairs of read blocks to find overlaps, requiring a large number of random accesses to off-chip memory. In comparison, D-SOFT algorithm is better suited to hardware acceleration, as it performs highly sequential accesses using a seed position table to off-chip DRAM, and random accesses during filtration are handled by fast on-chip memory. The same speedup is not achievable using D-SOFT in software, since random accesses to update bins occur over a memory range of up to 64MB — too large to fit on-chip in standard processors.

There are filtration techniques, such as BWA-MEM [47], that are neither based on counting seeds in a band of diagonals nor the number of bases. Instead, BWA-MEM uses super maximal seeds and its performance has been evaluated in this paper. Other examples include Canu [43], M-HAP [6] and LSH-ALL-PAIRS [9], which are based on probabilistic, locality-sensitive hashing of seeds. While D-SOFT cannot generalize to these techniques, as our results on other filtration techniques indicate, it is still possible to tune D-SOFT parameters to match or exceed the sensitivity of these heuristics.

Finally, there has been work on improving the sensitivity of seed-based filtration and reducing the storage requirement of seeds. These are orthogonal to D-SOFT since seed position table and seeds are provided by software. Work on spaced seeds [38], transition-constrained seeds [60] and vector seeds [8], help improve seeding sensitivity. Minimizers [66] help reduce the storage requirement of seeds.

Sequence alignment frameworks. Like Darwin, some prior work has focused on implementation of a complete sequence alignment framework, implementing filtration as well as sequence alignment with relatively flexible parameters, and aided by hardware acceleration. Examples include TimeLogic [2], which provides FPGA-based framework for BLAST [3] and HMMER [37], and Edico Genome [78], which provides FPGA-based framework for acceleration of the BWA-GATK pipeline for reference-guided assembly on second generation sequencing, reportedly reducing the computational time of whole human genome to under 20 minutes [19]. Darwin leverages the tunable sensitivity of D-SOFT, and accelerates alignment of arbitrarily long sequences using GACT, overcoming two major programmability limitations of prior frameworks for long read assembly. Darwin handles and provides high speedup versus hand-optimized software for two distinct applications: reference-guided and *de novo* assembly of reads, and can work with reads with very different error rates. To our knowledge, Darwin is the first hardware-accelerated framework to demonstrate this level of programmability and speedup in more than one class of applications.

11 Conclusion And Future Work

With the slow demise of Moore's law, hardware accelerators are needed to meet the rapidly growing computational requirements of genomics. In this paper, we have described Darwin, a hardware accelerator that speeds up long-read, reference-guided assembly of human genome by up to 15,000×, and the overlap step in *de novo* assembly by over 710× compared to software. Darwin achieves this speedup through hardware/algorithm co-design, trading more alignment (which is easy to accelerate) for less filtering (which is memory limited), and by optimizing the memory system for filtering. Darwin uses GACT, a novel tiled alignment algorithm with overlap that requires linear time and constant memory (for the compute-intensive step) as a function of read length, making hardware acceleration of long read alignment with traceback feasible. Darwin performs filtering using D-SOFT which can be programmed (via parameters k, h, and N) to match the sensitivity of baseline algorithms while maintaining high precision. Darwin's filtering and alignment operations are general and can be applied to rapidly growing sequence alignment applications beyond read assembly, such as whole genome alignments [30], metagenomics [29] and multiple sequence alignments [77].

Whole genome alignments [30] provide the rocket fuel for comparative genomics — to study at the molecular level, how different life forms are related and different from one another [5, 16, 31, 53]. Thousands of new species will be sequenced over the next decade and newer, more polished assemblies would be generated for already available genomes using better sequencing methods [41, 83]. We plan to extend Darwin's framework for handling whole genome alignments. D-SOFT parameters can be tuned to mimic the seeding stage of LASTZ [30], single-tile GACT filter replaces the bottleneck stage of ungapped extension — improving the sensitivity while still providing orders of magnitude speedup, and GACT can be further improved to use Y-drop extension strategy of LASTZ to align arbitrarily large genomes with smaller on-chip memory while still providing near-optimal alignments for highly-divergent sequences. A similar approach can be adapted in extending Darwin to metagenomics [29], which involves taxonomic classification using a very large seed table to estimate the microbial population from sequencing of an environment sample (such as human gut or soil) and has far-reaching potential in medicine, industry and agriculture. Multiple sequence alignment has important applications in consensus step of *overlap-layout-consensus* (OLC) assemblers [50] as well as in comparative genomics, such as for ancestral genome reconstruction [7]. Progressive aligners [7, 77] use sequence-sequence alignment (like GACT) during initial stages and profile-profile alignments in subsequent stages. Profile-based alignments are challenging to accelerate using systolic arrays but would be handled in our future work. Future work will also accelerate construction of seed position table in hardware.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback. We thank Dr. Aaron Wenger, Prof. Derek Chiou, Prof. Christos Kozyrakis and Dr. Amir Marcovitz for providing valuable feedback on the manuscript. We thank Dr. Jason Chin, Dr. Ivan Sovic, Prof. Gene Myers, Prof. Serafim Batzoglou and Dr. Heng Li for their valuable insights into long read sequencing and assembly tools. We thank Albert Ng who generously shared the RTL code of his Smith-Waterman accelerator. We thank Prof. Boris Murmann and Kevin Zheng, who provided the access to ASIC CAD tools. We also thank Microsoft Research, NVIDIA Research, Xilinx and the Platform Lab (Stanford University) for supporting this work.

References

[1] Pico computing product brief: M-505-k325t. URL https://goo.gl/poeWUA.

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

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

[4] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[5] G. Bejerano, M. Pheasant, I. Makunin, S. Stephen, W. J. Kent, J. S. Mattick, and D. Haussler. Ultraconserved elements in the human genome. *Science*, 304(5675):1321–1325, 2004.

[6] 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.

[7] N. Bray and L. Pachter. Mavid: constrained ancestral alignment of multiple sequences. *Genome research*, 14(4):693–699, 2004.

[8] B. Brejová, D. G. Brown, and T. Vinař. Vector seeds: An extension to spaced seeds. *Journal of Computer and System Sciences*, 70(3):364–380, 2005.

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

[10] B. Buyukkurt and W. A. Najj. Compiler generated systolic arrays for wavefront algorithm acceleration on fpgas. In *2008 International Conference on Field Programmable Logic and Applications*, pages 655–658. IEEE, 2008.

[11] 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.

[12] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens. Drampower: Open-source dram power & energy estimation tool. *URL: http://www. drampower. info*.

[13] 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.

[14] P. Chen, C. Wang, X. Li, and X. Zhou. Accelerating the next generation long read mapping with the fpga-based system. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 11(5):840–852, 2014.

[15] Y.-T. Chen, J. Cong, J. Lei, and P. Wei. A novel high-throughput acceleration engine for read alignment. In *Field-Programmable Custom*

Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on, pages 199–202. IEEE, 2015.

[16] S. L. Clarke, J. E. VanderMeer, A. M. Wenger, B. T. Schaar, N. Ahituv, and G. Bejerano. Human developmental enhancers conserved between deuterostomes and protostomes. *PLoS genetics*, 8(8):e1002852, 2012.

[17] W. J. Dally and B. P. Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[18] A. Döring, D. Weese, T. Rausch, and K. Reinert. Seqan an efficient, generic c++ library for sequence analysis. *BMC bioinformatics*, 9(1):11, 2008.

[19] Edico Genome. Dragen bio-it platform. URL http://edicogenome.com/dragen-bioit-platform/.

[20] J. Eid, A. Fehr, J. Gray, K. Luong, J. Lyle, G. Otto, P. Peluso, D. Rank, P. Baybayan, B. Bettman, et al. Real-time dna sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009.

[21] M. Eisenstein. Oxford nanopore announcement sets sequencing sector abuzz. *Nature biotechnology*, 30(4):295–296, 2012.

[22] P. Faes, B. Minnaert, M. Christiaens, E. Bonnet, Y. Saeys, D. Stroobandt, and Y. Van de Peer. Scalable hardware accelerator for comparing dna and protein sequences. In *Proceedings of the 1st international conference on Scalable information systems*, page 33. ACM, 2006.

[23] S. Goodwin, J. Gurtowski, S. Ethe-Sayers, P. Deshpande, M. C. Schatz, and W. R. McCombie. Oxford nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. *Genome research*, 25(11):1750–1756, 2015.

[24] D. Gordon, J. Huddleston, M. J. Chaisson, C. M. Hill, Z. N. Kronenberg, K. M. Munson, M. Malig, A. Raja, I. Fiddes, L. W. Hillier, et al. Long-read sequence assembly of the gorilla genome. *Science*, 352(6281):aae0344, 2016.

[25] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of molecular biology*, 162(3):705–708, 1982.

[26] D. Greaves, S. Sanyal, and S. Singh. Synthesis of a parallel smith-waterman sequence alignment kernel into fpga hardware. URL http://www.cl.cam.ac.uk/~djg11/pubs/mrsc09a.pdf.

[27] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

[28] M. A. Hamburg and F. S. Collins. The path to personalized medicine. *New England Journal of Medicine*, 363(4):301–304, 2010.

[29] J. Handelsman. Metagenomics: application of genomics to uncultured microorganisms. *Microbiology and molecular biology reviews*, 68(4):669–685, 2004.

[30] R. S. Harris. *Improved pairwise alignment of genomic DNA*. ProQuest, 2007.

[31] M. Hiller, B. T. Schaar, V. B. Indjeian, D. M. Kingsley, L. R. Hagey, and G. Bejerano. A âĂIJforward genomicsâĂİ approach links genotype to phenotype using independent phenotypic losses among related species. *Cell reports*, 2(4):817–823, 2012.

[32] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[33] Illumina. Illumina hiseq x series of sequencing systems: Specification sheet. URL https://goo.gl/paq2X5.

[34] Intel. Intel arria 10 device overview. URL https://www.altera.com/en_US/pdfs/literature/hb/arria-10/a10_overview.pdf.

[35] Intel. Intel pcm power utility. URL https://goo.gl/4KumhA.

[36] X. Jiang, X. Liu, L. Xu, P. Zhang, and N. Sun. A reconfigurable accelerator for smith–waterman algorithm. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 54(12):1077–1081, 2007.

[37] L. S. Johnson, S. R. Eddy, and E. Portugaly. Hidden markov model speed heuristic and iterative hmm search procedure. *BMC bioinformatics*, 11 (1):431, 2010.

[38] U. Keich, M. Li, B. Ma, and J. Tromp. On spaced seeds for similarity search. *Discrete Applied Mathematics*, 138(3):253–263, 2004.

[39] W. J. Kent. BLAT - the BLAST-like alignment tool. *Genome research*, 2002.

[40] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer Architecture Letters*, 15(1):45–49, 2016.

[41] K.-P. Koepfli, B. Paten, G. K. C. of Scientists, and S. J. OâĂŹBrien. The genome 10k project: a way forward. *Annu. Rev. Anim. Biosci.*, 3(1):57–111, 2015.

[42] S. Koren and A. M. Phillippy. One chromosome, one contig: complete microbial genomes from long-read sequencing and assembly. *Current opinion in microbiology*, 23:110–120, 2015.

[43] 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.

[44] N. Krumm, T. N. Turner, C. Baker, L. Vives, K. Mohajeri, K. Witherspoon, A. Raja, B. P. Coe, H. A. Stessman, Z.-X. He, et al. Excess of rare, inherited truncating mutations in autism. *Nature genetics*, 47(6):582–588, 2015.

[45] T. W. Lam, W.-K. Sung, S.-L. Tam, C.-K. Wong, and S.-M. Yiu. Compressed indexing and local alignment of dna. *Bioinformatics*, 24(6):791–797, 2008.

[46] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

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

[48] H. Li and R. Durbin. Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[49] J. Z. Li, D. M. Absher, H. Tang, A. M. Southwick, A. M. Casto, S. Ramachandran, H. M. Cann, G. S. Barsh, M. Feldman, L. L. Cavalli-Sforza, et al. Worldwide human relationships inferred from genome-wide patterns of variation. *Science*, 319(5866):1100–1104, 2008.

[50] Z. Li, Y. Chen, D. Mu, J. Yuan, Y. Shi, H. Zhang, J. Gan, N. Li, X. Hu, B. Liu, et al. Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph. *Briefings in functional genomics*, 11(1):25–37, 2012.

[51] R. J. Lipton and D. P. Lopresti. *Comparing long strings on a short systolic array*. Princeton University, Department of Computer Science, 1986.

[52] C. López-Otín, M. A. Blasco, L. Partridge, M. Serrano, and G. Kroemer. The hallmarks of aging. *Cell*, 153(6):1194–1217, 2013.

[53] A. Marcovitz, Y. Turakhia, M. Gloudemans, B. A. Braun, H. I. Chen, and G. Bejerano. A novel unbiased test for molecular convergent evolution and discoveries in echolocating, aquatic and high-altitude mammals. *bioRxiv*, page 170985, 2017.

[54] C. Y. McLean, P. L. Reno, A. A. Pollen, A. I. Bassan, T. D. Capellini, C. Guenther, V. B. Indjeian, X. Lim, D. B. Menke, B. T. Schaar, et al. Human-specific loss of regulatory dna and the evolution of human-specific traits. *Nature*, 471(7337):216–219, 2011.

[55] J. D. Merker, A. M. Wenger, T. Sneddon, M. Grove, Z. Zappala, L. Fresard, D. Waggott, S. Utiramerur, Y. Hou, K. S. Smith, et al. Long-read genome sequencing identifies causal structural variation in a mendelian disease. *Genetics in medicine: official journal of the American College of Medical Genetics*, 2017.

[56] 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.

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

[58] G. Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[59] Z. Nawaz, M. Nadeem, H. van Someren, and K. Bertels. A parallel fpga design of the smith-waterman traceback. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 454–459. IEEE, 2010.

[60] L. Noé and G. Kucherov. Yass: enhancing the sensitivity of dna similarity search. *Nucleic acids research*, 33(suppl 2):W540–W543, 2005.

[61] C. B. Olson, M. Kim, C. Clauson, B. Kogon, C. Ebeling, S. Hauck, and W. L. Ruzzo. Hardware acceleration of short read mapping. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pages 161–168. IEEE, 2012.

[62] Y. Ono, K. Asai, and M. Hamada. Pbsim: Pacbio reads simulator-toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2013.

[63] M. K. Papamichael and J. C. Hoe. Connect: re-examining conventional wisdom for designing nocs in the context of fpgas. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, pages 37–46. ACM, 2012.

[64] E. D. Pleasance, R. K. Cheetham, P. J. Stephens, D. J. McBride, S. J. Humphray, C. D. Greenman, I. Varela, M.-L. Lin, G. R. Ordóñez, G. R. Bignell, et al. A comprehensive catalogue of somatic mutations from a human cancer genome. *Nature*, 463(7278):191–196, 2010.

[65] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 13–24. IEEE, 2014.

[66] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004.

[67] E. E. Schadt, S. Turner, and A. Kasarskis. A window into third-generation sequencing. *Human molecular genetics*, 19(R2):R227–R240, 2010.

[68] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical report, Technical Report 2001/2, Compaq Computer Corporation, 2001.

[69] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[70] M. Šošić and M. Šikić. Edlib: a c/c++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 2017.

[71] 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.

[72] O. Spichenok, Z. M. Budimlija, A. A. Mitchell, A. Jenny, L. Kovacevic, D. Marjanovic, T. Caragine, M. Prinz, and E. Wurmbach. Prediction of eye and skin color in diverse populations using seven SNPs. *Forensic Science International: Genetics*, 5(5):472–478, 2011.

[73] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson. Big data: astronomical or genomical? *PLoS biology*, 13(7):e1002195, 2015.

[74] Synopsys IC. Compiler user guide, 2013. URL http://www.synopsys.com.

[75] Synopsys Inc. Compiler, design and user, rtl and guide, modeling, 2001. URL http://www.synopsys.com.

[76] W. Tang, W. Wang, B. Duan, C. Zhang, G. Tan, P. Zhang, and N. Sun. Accelerating millions of short reads mapping on a heterogeneous architecture with fpga accelerator. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pages 184–187. IEEE, 2012.

[77] J. D. Thompson, D. G. Higgins, and T. J. Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic acids research*, 22(22):4673–4680, 1994.

[78] P. Van Rooyen, R. J. McMillen, and M. Ruehle. Bioinformatics systems, apparatuses, and methods executed on an integrated circuit processing platform, Jan. 5 2016. US Patent App. 14/988,666.

[79] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, et al. The sequence of the human genome. *science*, 291(5507):1304–1351, 2001.

[80] R. H. Waterson, E. S. Lander, R. K. Wilson, et al. Initial sequence of the chimpanzee genome and comparison with the human genome. *Nature*, 437(7055):69, 2005.

[81] Y. Yamaguchi, T. Maruyama, and A. Konagaya. High speed homology search with fpgas. In *Proceedings of the 7th Pacific Symposium on Biocomputing (PSB'02)*, pages 271–282, 2001.

[82] C. W. Yu, K. Kwong, K.-H. Lee, and P. H. W. Leong. A smith-waterman systolic cell. In *New Algorithms, Architectures and Applications for Reconfigurable Computing*, pages 291–300. Springer, 2005.

[83] G. Zhang. Genomics: Bird sequencing project takes off. *Nature*, 522 (7554):34–34, 2015.

[84] 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.