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

Yatish Turakhia  Gill Bejerano  William J. Dally

Stanford University

ASPLOS’18

João Sanches Ferreira
ETH Zürich
9 May 2019
Executive Summary

- **Motivation**: DNA sequencing technological improvements have resulted in longer reads, which results in higher quality genome assembly.
Problem
Cost per Genome

# Comparison of genomic sequencing generations

<table>
<thead>
<tr>
<th></th>
<th>First generation</th>
<th>Second generation</th>
<th>Third generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fundamental technology</strong></td>
<td>Size-separation of specifically end-labeled DNA fragments</td>
<td>Wash-and-scan SBS</td>
<td>Single molecule real time sequencing</td>
</tr>
<tr>
<td><strong>Resolution</strong></td>
<td>Averaged across many copies of the DNA molecule</td>
<td>Averaged across many copies of the DNA molecule</td>
<td>Single DNA molecule</td>
</tr>
<tr>
<td><strong>Current raw read accuracy</strong></td>
<td>High</td>
<td>High</td>
<td>Lower</td>
</tr>
<tr>
<td><strong>Current read length</strong></td>
<td>Moderate (800-1000 bp)</td>
<td>Short (generally much shorter than Sanger sequencing)</td>
<td>&gt; 1000 bp</td>
</tr>
<tr>
<td><strong>Current throughput</strong></td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td><strong>Current cost</strong></td>
<td>High cost per base, Low cost per run</td>
<td>Low cost per base, High cost per run</td>
<td>Low cost per base, High cost per run</td>
</tr>
<tr>
<td><strong>RNA-sequencing method</strong></td>
<td>cDNA sequencing</td>
<td>cDNA sequencing</td>
<td>Direct RNA sequencing</td>
</tr>
<tr>
<td><strong>Time to result</strong></td>
<td>Hours</td>
<td>Days</td>
<td>&lt; 1 day</td>
</tr>
<tr>
<td><strong>Sample preparation</strong></td>
<td>Moderately complex, PCR amplification is not required</td>
<td>Complex, PCR amplification is required</td>
<td>Various</td>
</tr>
<tr>
<td><strong>Data analysis</strong></td>
<td>Routine</td>
<td>Complex (due to large data volumes &amp; short reads)</td>
<td>Complex</td>
</tr>
<tr>
<td><strong>Primary results</strong></td>
<td>Base calls with quality values</td>
<td>Base calls with quality values</td>
<td>Base calls with quality values</td>
</tr>
</tbody>
</table>

Adapted from Schadt, et al. Hum Mol Genet 2010\textsuperscript{13}
Executive Summary

- **Motivation:** DNA sequencing technological improvements have resulted in longer reads, which results in higher quality genome assembly.

- **Problem:** Genomic sequencing technology is scaling, compute performance isn’t.
Executive Summary

- **Motivation**: DNA sequencing technological improvements have resulted in longer reads, which results in higher quality genome assembly.

- **Problem**: Genomic sequencing technology is scaling, compute performance isn’t.

- **Goal**: Introduce a co-processor to accelerate genomic sequence alignment – Darwin.
Executive Summary

- **Motivation:** DNA sequencing technological improvements have resulted in longer reads, which results in higher quality genome assembly.

- **Problem:** Genomic sequencing technology is scaling, compute performance isn’t.

- **Goal:** Introduce a co-processor to accelerate genomic sequence alignment – Darwin.

- **Solution:** Co-design algorithms and hardware targeted at long (3rd-gen) read assembly.
Executive Summary

- **Motivation**: DNA sequencing technological improvements have resulted in longer reads, which results in higher quality genome assembly.
- **Problem**: Genomic sequencing technology is scaling, compute performance isn’t.
- **Goal**: Introduce a co-processor to accelerate genomic sequence alignment – Darwin.
- **Solution**: Co-design algorithms and hardware targeted at long (3rd-gen) read assembly.
- **Evaluation**:
  - 3-4 orders of magnitude faster reference-guided assembly
  - 2 orders of magnitude faster de novo assembly
Background
DNA Sequencing

**Goal:** Find the complete sequence of A, C, G, T’s in DNA.

**Challenge:** There is no machine/technology/method that takes long DNA as an input, and gives the complete sequence as output.
DNA Sequencing

All sequencing machines chop DNA into pieces and identify relatively small pieces (but not how they fit together).
DNA Sequencing

- **short reads**
  - 50-300 bp
  - low error rate (~0.1%)

- **long reads**
  - 10K-100K bp
  - high error rate (~15%)

Size of human genome: **3.2 Billion bp**
DNA Sequence Alignment

- Compare a query sequence Q and a reference sequence R, to maximize an alignment score.
  - Identify insertions, deletions or mismatches.
Alignment Algorithms

- **Smith-Waterman algorithm**
  - Identifies *similar regions* between two input sequences
  - Compares segments of all possible lengths
  - Ensures *optimal local alignment*

![Alignment Algorithms Diagram]

- **Initialze the scoring matrix**
- **Fill the scoring matrix**
- **Identify the highest score**
- **Traceback**

- **Substitution matrix:**
  - \( S(a_i, b_j) = \begin{cases} \text{+3,} & a_i = b_j \\ -3, & a_i \neq b_j \end{cases} \)

- **Gap penalty:**
  - \( W_k = kW_1 \)
  - \( W_1 = 2 \)
**Filtering Algorithms**

- **Problem**
  - Smith-Waterman (and similar algorithms) are **computationally expensive**.

- **Solution**
  - Use filtering step based on **seed-and-extend paradigm**.
  - This approach uses **seeds**, substrings of fixed size $k$ from $Q$, and finds their exact matches in $R$, called **seed hits**.
1 Sequencing

2 Read Mapping

3 Variant Calling

4 Scientific Discovery

reference: TTTATCGCTTCCATGACGCAG
read1: ATCGCATCC
read2: TATCGCATC
read3: CATCCATGA
read4: CGCTTCCAT
read5: CCATGACGC
read6: TTCCATGAC

Billions of Short Reads

Read Alignment
Novelty
Novelty

- **D-SOFT** – Filtering algorithm
  - Tunable sensitivity (tolerance to inexact matches).
  - High precision.

- **GACT** – Alignment algorithm
  - Arbitrarily long sequences, with optimal alignment for error rates of up to 40%.
  - Constant memory for the compute-intensive step.

- **Darwin implementation**
  - FPGA.
  - ASIC (simulated, by scaling up frequency).
Mechanisms
Mechanisms – D-SOFT
Mechanisms – SeedLookup

Figure 3: An example reference sequence and seed position table used in SeedLookup.
Mechanisms – D-SOFT
Mechanisms – GACT
Key Results:
Methodology and Evaluation
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.
### Key Results

<table>
<thead>
<tr>
<th>Size ($k$)</th>
<th>hits/seed (GRCh38)</th>
<th>Throughput (Kseeds/sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Software</td>
<td>Darwin</td>
</tr>
<tr>
<td>11</td>
<td>1866.1</td>
<td>16.6</td>
<td>1,426.9</td>
</tr>
<tr>
<td>12</td>
<td>491.6</td>
<td>66.2</td>
<td>5,422.6</td>
</tr>
<tr>
<td>13</td>
<td>127.3</td>
<td>259.3</td>
<td>19,081.7</td>
</tr>
<tr>
<td>14</td>
<td>33.4</td>
<td>869.5</td>
<td>55,189.2</td>
</tr>
<tr>
<td>15</td>
<td>8.7</td>
<td>2,257.1</td>
<td>91,138.7</td>
</tr>
</tbody>
</table>

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.
### Key Results

#### Reference-guided assembly (human)

<table>
<thead>
<tr>
<th>Read type</th>
<th>D-SOFT ((k, N, h))</th>
<th>Sensitivity</th>
<th>Precision</th>
<th>Reads/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Darwin</td>
<td>Baseline</td>
<td>Darwin</td>
</tr>
<tr>
<td>PacBio</td>
<td>(14, 750, 24)</td>
<td>95.95%</td>
<td>95.95%</td>
<td>3.71</td>
</tr>
<tr>
<td>ONT_2D</td>
<td>(12, 1000, 25)</td>
<td>98.11%</td>
<td>99.10%</td>
<td>0.18</td>
</tr>
<tr>
<td>ONT_1D</td>
<td>(11, 1300, 22)</td>
<td>97.10%</td>
<td>98.20%</td>
<td>0.18</td>
</tr>
</tbody>
</table>

#### De novo assembly \((C. elegans)\)

<table>
<thead>
<tr>
<th>Read type</th>
<th>D-SOFT ((k, N, h))</th>
<th>Sensitivity</th>
<th>Precision</th>
<th>Runtime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Darwin</td>
<td>Baseline</td>
<td>Darwin (Speedup)</td>
</tr>
<tr>
<td>PacBio</td>
<td>(14, 1300, 24)</td>
<td>99.80%</td>
<td>88.30%</td>
<td>47,524</td>
</tr>
</tbody>
</table>

#### De novo assembly (human)

<table>
<thead>
<tr>
<th>Read type</th>
<th>D-SOFT ((k, N, h))</th>
<th>Estimated Runtime (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Darwin (Speedup)</td>
</tr>
<tr>
<td>PacBio</td>
<td>(14, 1300, 24)</td>
<td>15,600</td>
</tr>
</tbody>
</table>

Table 4: Comparison of Darwin with baseline techniques on reference-guided and \textit{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.
Summary
Summary

- Volume of genomic data is rapidly increasing.
  - Need for efficient sequence alignment, unmet by present-day hardware.

- Darwin – a co-processor for genomic sequence alignment that combines hardware-accelerated alignment (GACT) and filtering (D-SOFT) algorithms.
  - D-SOFT
    - Tunable sensitivity.
  - GACT
    - Can process arbitrarily long sequences
    - Requires constant memory for the compute-intensive step.

- 2-4 orders of magnitude improvement in sequencing performance, compared to baseline.
Strengths
Strengths

- **HW/SW co-design**
  - Memory system is optimized for filtering (which is more expensive than alignment).
    - 4 DRAM channels store identical copies of the seed position table.
    - Seed hits are stored sequentially.

- Filtering algorithm offers tunable sensitivity.

- Alignment algorithm is linear-time and constant-memory.

- Filtering and alignment can be used in other genomics applications:
  - Whole sequence alignments, metagenomics, multiple sequence alignments...
Weaknesses
Weaknesses

- Poor baseline
  - Baseline is single-threaded CPU
  - Hardware/Accelerator baseline is missing
    - Multi-threaded / PIM / GPU / FPGA...
    - Speedups are only given with reference to CPU, running a single-thread.

- Tiling is not novel – used typically in greed mapping.
  - Seems heuristic?
  - Guarantee of optimality?

- ASIC performance is only simulated by scaling up the frequency, with the FPGA version as a baseline.

- No direct comparison of D-SOFT / GACT with other filtering/alignment algorithms?
Thoughts and Ideas
Thoughts and Ideas

- What other algorithms can be modified to maximally exploit HW/SW codesign?
- What if we used different filters?
  - How would that affect sensitivity?
  - What’s the sensitivity of this filtering? How does it respond to alignments that are not true alignment?
- Can we use PIM? Could we do (some?) of the computation in-memory, to avoid having to move data from the DRAM memory to the accelerators?
Takeaways
Takeaways

- **Specialized hardware** is increasingly important
  - Specialization gives **efficiency**, parallelization gives **speedup**.
  - Specialization may require **changes to the algorithms**
    - Case in point: GACT, D-SOFT.

- Memory access time dominates
  - Optimizing access patterns is critical for performance.
  - Computation in memory pays off.

- Previous points show the importance of **HW/SW co-design**.
Open Discussion
Discussion Starters

Could Darwin be used for Whole Genome Alignment?

whole genome alignment (WGA) refers to the computational process of aligning entire genome sequences of two or more species in order to study their evolutionary relationship. In particular, WGA helps in identifying set of sequences that are orthologous (diverged after a speciation event) or paralogous (diverged after a duplication event) [28]. As
Discussion

- What must be changed in Darwin to achieve WGA?

![Diagram showing Darwin-WGA with binning](image)

Figure 4: Overview of Darwin-WGA (a) Target bins and query chunks constitute a diagonal band, at most 1 seed hit to be extended per diagonal band. (b) Tile for banded Smith-Waterman algorithm, blue represents the band calculated, yellow positions with the seed at its center. (c) Right and left extension from the anchor, tile overlapping and alignment reconstruction from the traceback pointers. The blue band on the right represents the calculated portion of the dynamic programming matrix.

The threshold parameter \( h \) concerns the number of seed hits per diagonal band. At most 1 seed hit is extended per diagonal band. This reduces redundant extensions for seed hits within the same diagonal band.
Can merge the filter + alignment operations be merged to gain efficiency?

For example, by incorporating them directly into the sequencer?
HiLive: real-time mapping of illumina reads while sequencing

Martin S. Lindner\textsuperscript{1,\*}, Benjamin Strauch\textsuperscript{1}, Jakob M. Schulze\textsuperscript{1}, Simon H. Tausch\textsuperscript{1,2}, Piotr W. Dabrowski\textsuperscript{1,2}, Andreas Nitsche\textsuperscript{2} and Bernhard Y. Renard\textsuperscript{1,\*}

Motivation: Next Generation Sequencing is increasingly used in time critical, clinical applications. While read mapping algorithms have always been optimized for speed, they follow a sequential paradigm and only start after finishing of the sequencing run and conversion of files. Since Illumina machines write intermediate output results, HiLive performs read mapping while still sequencing and thereby drastically reduces crucial overall sample analysis time, e.g. in precision medicine.
Discussion
Darwin: A Genomics Co-processor Provides up to 15,000 × acceleration on long read assembly

Yatish Turakhia    Gill Bejerano    William J. Dally

Stanford University

ASPLOS’18

João Sanches Ferreira
ETH Zürich
9 May 2019
Backup Slides
Alignment Algorithms

- Smith-Waterman algorithm

\[ H_{ij} = \max \begin{cases} 
H_{i-1,j-1} + s(a_i, b_j), \\
H_{i-1,j} - W_1, \\
H_{i,j-1} - W_1, \\
0 
\end{cases} \]
Approximate String Matching

- **Edit distance** is defined as the minimum number of edits (i.e. insertions, deletions, or substitutions) needed to make the read exactly match the reference segment.

NETHERLANDS x SWITZERLAND

<table>
<thead>
<tr>
<th>N</th>
<th>E</th>
<th>T</th>
<th>H</th>
<th>E</th>
<th>R</th>
<th>L</th>
<th>A</th>
<th>N</th>
<th>D</th>
<th>S</th>
</tr>
</thead>
</table>
| S | W | I | T | Z | E | R | L | A | N | D | -

- match
- deletion
- insertion
- mismatch
Algorithm 1: D-SOFT

1. $\text{candidate_pos} \leftarrow []$;
2. $\text{last_hit_pos} \leftarrow [-k \text{ for } i \text{ in range}(N_B)]$;
3. $\text{bp_count} \leftarrow [0 \text{ for } i \text{ in range}(N_B)]$;
4. $\text{for } j \text{ in start : stride : end do}$
   5. $\text{seed} \leftarrow Q[j : j + k]$;
   6. $\text{hits} \leftarrow \text{SeedLookup}(R, \text{seed})$;
   7. $\text{for } i \text{ in hits do}$
      8. $\text{bin} \leftarrow \lceil (i - j)/B \rceil$;
      9. $\text{overlap} \leftarrow \max(0, \text{last_hit_pos}[\text{bin}] + k - j)$;
     10. $\text{last_hit_pos}[\text{bin}] \leftarrow j$;
     11. $\text{bp_count}[\text{bin}] \leftarrow \text{bp_count}[\text{bin}] + k - \text{overlap}$;
     12. $\text{if } (h + k - \text{overlap} > \text{bp_count}[\text{bin}] \geq h) \text{ then}$
        13. $\text{candidate_pos}.\text{append}(\langle i, j \rangle)$;
     14. $\text{end}$
   15. $\text{end}$
16. $\text{return } \text{candidate_pos}$;
Algorithm – GACT

Algorithm 2: GACT for Left Extension

1. $tb\_left \leftarrow []$;
2. $(i_{curr}, j_{curr}) \leftarrow (i^*, j^*)$;
3. $t \leftarrow 1$;
4. while (($i_{curr} > 0$) and ($j_{curr} > 0$)) do
   5. $(i_{start}, j_{start}) \leftarrow (\max(0, i_{curr} - T), \max(0, j_{curr} - T))$;
   6. $(R_{tile}, Q_{tile}) \leftarrow (R[i_{start} : i_{curr}], Q[i_{start} : i_{curr}])$;
   7. $(TS, i_{off}, j_{off}, i_{max}, j_{max}, t) \leftarrow$
      Align($R_{tile}, Q_{tile}, t, T - O$);
   8. $tb\_left$.prepend($t$);
   9. if ($t == 1$) then
      10. $(i_{curr}, j_{curr}) \leftarrow (i_{max}, j_{max})$;
      11. $t \leftarrow 0$;
   12. end
   13. if ($i_{off} == 0$) and ($j_{off} == 0$)) then
      14. break;
   15. end
   16. else
      17. $(i_{curr}, j_{curr}) \leftarrow (i_{curr} - i_{off}, j_{curr} - j_{off})$
   18. end
19. return $(i_{max}, j_{max}, tb\_left)$;
Mechanisms – GACT

\[
I(i, j) = \max \left\{ \begin{array}{l}
H(i, j - 1) - o \\
I(i, j - 1) - e
\end{array} \right.
\]

\[
D(i, j) = \max \left\{ \begin{array}{l}
H(i - 1, j) - o \\
D(i - 1, j) - e
\end{array} \right.
\]

\[
H(i, j) = \max \left\{ \begin{array}{l}
0 \\
I(i, j) \\
D(i, j) \\
H(i - 1, j - 1) + W(r_i, q_j)
\end{array} \right.
\]
Mechanisms – Darwin Overview

(a)

(b)
Mechanisms – D-SOFT HW Implementation
Mechanisms – GACT HW Implementation

(a) Mechanism Diagram

(b) Reference and Query Alignment