GenDP: A Framework of Dynamic Programming Acceleration for Genome Sequencing Analysis

Yufeng Gu, Arun Subramaniyan, Tim Dunn, Alireza Khadem, Kuan-Yu Chen, Somnath Paul, Md Vasimuddin, Sanchit Misra, David Blaauw, Satish Narayanasamy, Reetuparna Das

Open-source: https://github.com/Yufeng98/GenDP
Genetic data is produced cheaper and faster

The explosive growth of genetic data motivates hardware acceleration for genome sequencing analysis

https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data
Data Source: https://www.ncbi.nlm.nih.gov/Traces/sra/sra_stat.cgi
How to accelerate genome sequencing pipelines

Reference genome (R)

Read (Q)

Read Alignment

Variant Calling

* Not proportional to the runtime for each stage
How to accelerate genome sequencing pipelines

Software tools

<table>
<thead>
<tr>
<th>Short Reads</th>
<th>Long Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bustard</td>
<td>Bonito, Guppy</td>
</tr>
<tr>
<td>BWA-MEM2</td>
<td>Minimap2</td>
</tr>
<tr>
<td>GATK Haplotype Caller, Platypus</td>
<td>Clair</td>
</tr>
</tbody>
</table>

General-Purpose Hardware

- Intel Genomics Kernel Library (GKL)
- NVIDIA Clara Parabricks Toolkit

Domain specific accelerator bridges the gap between general-purpose processing and custom accelerator

Custom Accelerators

- SquiggleFilter
- GenASM Darwin
- Brune-PairHMM

* Not proportional to the runtime for each stage
Genomic Sequencing Pipelines

(a) Reference-Guided Assembly

- **Sequencing**
  - Human Genome ~6 Gigabases
  - Software tools: fast5, fastq
- **Basecalling**
  - Reads (~millions-billions)
  - Software tools: .fastq
- **Read Alignment**
  - Reference genome (R)
  - Software tools: BWA-MEM (~30-40 %)
- **Variant Calling**
  - Aligned Reads
  - Software tools: GATK Haplotype Caller, Platypus (~40%)

(b) De-Novo Assembly

- **Sequencing**
  - Genome sizes: ~Mega -- Gigabases
  - Software tools: fast5, fastq
- **Basecalling**
  - Reads (~millions-billions)
  - Software tools: .fastq
- **Overlap, Layout, Consensus**
  - Software tools: Bonito (~30 %)
- **Error Correction and Polishing**
  - Software tools: Flye, Minimap2

(c) Metagenomics Classification

- **Sequencing**
  - Genome sizes: ~Megabases
  - Software tools: fast5, fastq
- **Basecalling**
  - Reads (~millions)
  - Software tools: .fastq
- **Abundance Estimation**
  - Software tools: Centrifuge, Minimap2
- **Phylogenetic Tree**

Genomic Sequencing Pipelines

(a) Reference-Guided Assembly

- **Sequencing**: Squiggle
- **Basecalling**: CGTGAAG, GAAAGTTT
- **Read Alignment**: Reference genome (R)
- **Variant Calling**: A T CG T G A G T T T, C G T GA G

<table>
<thead>
<tr>
<th>Software tools:</th>
<th>Short Reads</th>
<th>Long Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast5</td>
<td>Bustard</td>
<td>Bonito (~30%)</td>
</tr>
<tr>
<td>fastq</td>
<td>BWA-MEM2 (~30-40%)</td>
<td>Fие, Minimap2</td>
</tr>
<tr>
<td>.bam</td>
<td>GATK Haplotype Caller, Platypus (~40%)</td>
<td>Racon, Nanopolish</td>
</tr>
</tbody>
</table>

(b) De-Novo Assembly

- **Sequencing**: Squiggle
- **Basecalling**: GAGTCTTG, GTCTGTC, GAACTGC
- **Overlap, Layout, Consensus**: GAGTCTTG
- **Error Correction and Polishing**: GAGTCTTG

<table>
<thead>
<tr>
<th>Software tools:</th>
<th>Long Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast5</td>
<td>Bonito (~30%)</td>
</tr>
<tr>
<td>fastq</td>
<td>GATK Haplotype Caller, Platypus (~40%)</td>
</tr>
</tbody>
</table>

(c) Metagenomics Classification

- **Sequencing**: Squiggle
- **Basecalling**: GAGTCTTG, GTCTGTC, GAACTGC

<table>
<thead>
<tr>
<th>Software tools:</th>
<th>Long Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast5</td>
<td>Bonito</td>
</tr>
</tbody>
</table>

Dynamic Programming Kernels in GenomicsBench

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Input Datatype</th>
<th>Applications</th>
<th>Chosen Tool</th>
<th>% Time Spent in Tool (single-thread)</th>
<th>Parallelism Motif</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmi</td>
<td>Short reads</td>
<td>Read Alignment Metagenomics Classification</td>
<td>BWA-MEM2</td>
<td>38%</td>
<td>Tree Traversal</td>
</tr>
<tr>
<td>bsw</td>
<td>Short reads</td>
<td>Read Alignment De-Novo Assembly</td>
<td>BWA-MEM2</td>
<td>31%</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>dbg</td>
<td>Short reads</td>
<td>Variant Calling De-Novo Assembly</td>
<td>Platypus</td>
<td>65%</td>
<td>Graph Construction Hash Table</td>
</tr>
<tr>
<td>phmm</td>
<td>Short reads</td>
<td>Variant Calling Error Correction</td>
<td>GATK Haplo caller</td>
<td>70%</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>chain</td>
<td>Long reads</td>
<td>De-Novo Assembly Read Alignment</td>
<td>Minimap2</td>
<td>47.4%</td>
<td>Dynamic Programming (1D)</td>
</tr>
<tr>
<td>spoa</td>
<td>Long reads</td>
<td>Error Correction</td>
<td>Racon</td>
<td>75%</td>
<td>Dynamic Programming Graph Construction</td>
</tr>
<tr>
<td>abea</td>
<td>Long reads</td>
<td>Basecalling Variant Calling</td>
<td>Nanopolish</td>
<td>71.4%</td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td>grm</td>
<td>NA</td>
<td>Population Genomics</td>
<td>PLINK2</td>
<td>92.8%</td>
<td>Dense Matrix Multiplication</td>
</tr>
<tr>
<td>nn-nv</td>
<td>Long reads</td>
<td>Basecalling</td>
<td>Brixton</td>
<td>95%</td>
<td>FP Matrix Multiplication</td>
</tr>
</tbody>
</table>

Dynamic programming is the fundamental algorithm in genome sequencing analysis and motivates a domain specific accelerator.
Sequence Alignment using Smith Waterman

Affine gap alignment (score calculation)
Weights used from BWA.
Data: HG002 (NA24385) paired-end protocol using Illumina Sequencers.

ISA Extension

(a) max
(b) add
(c) comp

Dynamic Programming (DP)

- Task: calculate the scores in the entire table.
- Initialization: the scores in first row and first column.
- Objective Function: calculate the score of a cell based on its upper, left and diagonal neighbors.

Compute a new score based on known scores
Genomics DP Kernels

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Application</th>
<th>Dimension and Size</th>
<th>Dependency</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bsw</td>
<td>Read Alignment</td>
<td>2D ~ 120×60</td>
<td>Last 2 Wave-fronts</td>
<td>Int 8/16</td>
</tr>
</tbody>
</table>

The divergence of dynamic programming kernels makes it challenging to design a domain specific accelerator.

### Similarity

- **Customization**
- **Programmability**

### Theorem

\[
score(i) = \max \{ \text{score}(i) + \text{weight}(i, j) \}
\]

\[
\text{for } i < j \leq i + N, \text{ do }
\]

\[
\text{anchor dependency}
\]

\[
\text{ reordered anchor dependency}
\]

\[
\text{chain}
\]
GenDP Framework

- **DPAx**: programmable dynamic programming (DP) accelerator.
- **DPMAP**: map the objective function of DP algorithm to DPAx accelerator.
DPAx Accelerator

Query sequence:

<table>
<thead>
<tr>
<th>Target sequence</th>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
<th>PE 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Target sequence:

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>T</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>G</td>
<td>T</td>
<td>T</td>
<td>C</td>
</tr>
</tbody>
</table>

Query sequence:

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
<th>PE 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Target sequence:

<table>
<thead>
<tr>
<th>PE 0</th>
<th>PE 1</th>
<th>PE 2</th>
<th>PE 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DRAM

Integer PE Array

PE 0 → PE 1 → PE 2 → PE 3

Integer PE Array

PE 4 → PE 5 → PE 6 → PE 7

Integer PE Array

PE 60 → PE 61 → PE 62 → PE 63

Floating Point PE Array

FP PE 0 → FP PE 1 → FP PE 2 → FP PE 3
PE Array Architecture

Query sequence

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>T</th>
<th>C</th>
</tr>
</thead>
</table>

Target sequence

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
</table>

PE 0

PE 1

PE 2

PE 3

Control Instruction Buffer

Decoder

Input Data Buffer

FIFO

Output Data Buffer

Data Buffer

Control Flow

Data Flow
### PE Architecture

#### Query sequence

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>T</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>O</td>
<td>G</td>
<td>T</td>
<td>T</td>
<td>C</td>
</tr>
</tbody>
</table>

#### Target sequence

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>G</td>
<td>T</td>
</tr>
</tbody>
</table>

#### Data movement

- **Computation**

#### PE

- **Control Instruction Buffer**
- **Control Decoder**
- **Register File**
- **Compute Instruction Buffer**
- **Compute Decoder**
- **Scratchpad Memory**

#### PE Array

- **Data Flow Control Flow**
- **PE Array [i-1]**
- **Control Instruction Buffer**
- **Decoder**
- **Input Data Buffer**
- **Output Data Buffer**
- **PE**
- **PE Array [i+1]**
- **Data Buffer**

#### Compute Unit Array

- **32-bit CU**
- **8-bit CU**

#### Diagram

- **DRAM Channels**
- **Input Data Buffer**
- **Output Data Buffer**
- **FIFO**
Design Choice Take Away

**Similarity**
- Local dependency
- Reduction tree data path
- Precision requirement
- Dependency patterns
- Long dependency
- Objective func. and datapath

**Difference**
- 1-Dimension systolic PE array with FIFO
- Compute unit – 2-level reduction tree
- 16 Integer PE array (SIMD compute unit) and 1 FP PE array
- PE arrays could execute separately or combined
- Software managed scratchpad memory
- Custom ISA for control and computation
**DPMap Algorithm**

- **DPMap**: map the objective function of DP algorithm to programmable compute units in DPAX.
  - **Partitioning**: Break the data-flow graph with 4-input ALU and Multiplier
  - **Seeding**: Look for vertices that could be mapped to the 2nd level
  - **Refinement**: Break the single-strand structure

More details in the paper
Methodology

• Evaluate 4 representative genomics DP kernels
  • Banded Smith-Waterman (BSW)
  • Pairwise Hidden Markov Model (PairHMM)
  • Partial Order Alignment (POA)
  • Chain

• CPU Baseline
  • Intel Xeon Platinum 8380
  • SIMD optimization with AVX512 and 80 threads

• GPU Baseline
  • NVIDIA A100

• GenDP
  • Obtain throughput with an in-house cycle-accurate simulator.
  • Synthesis with a TSMC 28nm process to estimate the area and power of DPAX.
GenDP Performance

- Metrics: Throughput/Area – Million Cell Updates per Second/mm² (MCUPS/mm²)
- GenDP achieves $157.8 \times$ throughput/mm² over GPU
- GenDP has 2.8x slowdown when compared to custom accelerators
- Generality on DP algorithms in other domains
  - Dynamic time warping – speech detection
  - Bellman-Ford – Robot motion planning
GenDP: A Framework of Dynamic Programming Acceleration for Genome Sequencing Analysis

Yufeng Gu, Arun Subramaniyan, Tim Dunn, Alireza Khadem, Kuan-Yu Chen, Somnath Paul, Md Vasimuddin, Sanchit Misra, David Blaauw, Satish Narayanasamy, Reetuparna Das

Open-source: https://github.com/Yufeng98/GenDP

Acknowledgement