Massively parallel read mapping on graphics cards

Johannes Köster

May 15, 2014
Outline

1. Next-Generation-Sequencing of DNA
2. Read Mapping
3. Algorithm
4. Results
Outline

1. Next-Generation-Sequencing of DNA
2. Read Mapping
3. Algorithm
4. Results
Next-Generation-Sequencing

1. Chop DNA/RNA into small fragments.
2. Ligate adapters to both ends.
3. Spread fragment solution across a flowcell with beads.
4. Amplify fragments into clusters (PCR).
5. Sequence fragments by adding fluorescent complementary bases ▶ reads.

Illumina, 2013
Outline

1. Next-Generation-Sequencing of DNA
2. Read Mapping
3. Algorithm
4. Results
Read Mapping

For each read...
find position in the known reference genome.

- A DNA sequence is a word over $\Sigma = \{A, C, G, T\}$.
- string matching, but with error tolerance
Read Mapping

For each read...
find position(s) with optimal alignment(s) to either strand of the reference:

ACTGTGGACTATCAATGGAC
GGTACTGTCTATCTATGGACCGTTAG

• Smith Waterman Algorithm

Too slow, therefore heuristics to find anchor positions:
• suffixarray/Burrows-Wheeler-Transformation (BWA, bowtie2)
• q-gram indices (RazerS3)
Read mapping on GPUs

Challenges:
- limited and slow memory
- branching interrupts parallelism

 تماماً q-gram index
 تماماً BWT

Idea:
- Use a special q-gram index with small memory footprint.
- Use parallelism to hide memory latency.
- Export branching into bitvector operations.
  - PEANUT – the Parallel Alignment UTility
Outline

1. Next-Generation-Sequencing of DNA
2. Read Mapping
3. Algorithm
4. Results
Algorithm

Main steps:

- Filtration
  find potential hits between reads and reference using a special q-gram index

- Validation
  validate hits using a bit-parallel alignment algorithm
Main steps:

- **Filtration**
  find potential hits between reads and reference using a special q-gram index

- **Validation**
  validate hits using a bit-parallel alignment algorithm
Q-Gram Index

For a given DNA sequence $T$:

- consider q-grams (substrings of length $q$)
  
  GGTACTGACGTTCTATGGACCGTTAG

- encode them as integers
  
  ACGT = 11 10 01 00 = 228

- array $P$ with concatenation of q-gram positions
- array $Q$ with address in $P$ for each q-gram
  
  size $4^q + |T|$

  $$P[Q[228]] \ldots P[Q[229]]$$
Q-Group Index

- assign each q-gram to a q-group \( \lfloor g/w \rfloor \)
- store occurrence of q-gram in a bit-vector
- two address arrays guide from q-group to positions of the q-gram in the text
  - size \( 2/w \cdot 4^q + \min\{4^q, |T|\} + |T| \)
Q-Group Index

less memory, because we consider only...

- q-groups at the top level
- occurring q-grams at the bottom

calculate address ranges in parallel by

- population counts
- prefix-sums
Algorithm

Main steps:

- **Filtration**
  find potential hits between reads and reference using a special q-gram index

- **Validation**
  validate hits using a bit-parallel alignment algorithm
Observations:
- calculating column $j$ needs only column $j - 1$
- each transition changes edit distance by at most 1

Myers bit-parallel algorithm\textsuperscript{1}:
- process graph column-wise
- maintain distance deltas in bitvectors

\textsuperscript{1}Myers, 1999. J. ACM 46.
Workflow

- load reads into buffer
- build q-group index of reads
- filtration of hits
- validation of hits
- postprocessing
- writing

IO
GPU
CPU
Outline

1. Next-Generation-Sequencing of DNA
2. Read Mapping
3. Algorithm
4. Results
Results

![Graph showing occupancy vs. block size for different filters: filter_reference, create_queries_index, validate_hits. The graph illustrates the relationship between block size and occupancy for each filter, highlighting trends and performance.](image-url)
Sensitivity

- assessed with Rabema\textsuperscript{2} benchmark on S. cerevisiae genome
- 100% for reads with error rate less than 7%
- 99.77% for error rates up to 10%
- 98.97% for error rates up to 20%

\textsuperscript{2}Holtgrewe et al. 2011. BMC Bioinformatics
Performance

Output types:

- **all** all alignments of a read
- **best** one of the best alignments
- **best-stratum** all best alignments

5 million simulated human reads:

<table>
<thead>
<tr>
<th>mapper</th>
<th>type</th>
<th>time [min:sec]</th>
<th>sens. [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEANUT</td>
<td>best-stratum</td>
<td>1:55</td>
<td>98.62</td>
</tr>
<tr>
<td>BWA-MEM</td>
<td>best</td>
<td>3:16</td>
<td>96.99</td>
</tr>
<tr>
<td>Bowtie 2</td>
<td>best</td>
<td>5:21</td>
<td>96.85</td>
</tr>
<tr>
<td>PEANUT</td>
<td>all</td>
<td>18:29</td>
<td>98.74</td>
</tr>
<tr>
<td>RazerS 3</td>
<td>all</td>
<td>199:55</td>
<td>98.83</td>
</tr>
</tbody>
</table>

Intel Core i7, 16GB RAM
NVIDIA Geforce 780, 3GB RAM
Performance

5 million real human exome reads:

<table>
<thead>
<tr>
<th>mapper</th>
<th>type</th>
<th>time [min:sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEANUT</td>
<td>best-stratum</td>
<td>1:33</td>
</tr>
<tr>
<td>BWA-MEM</td>
<td>best</td>
<td>1:58</td>
</tr>
<tr>
<td>Bowtie 2</td>
<td>best</td>
<td>3:12</td>
</tr>
<tr>
<td>PEANUT</td>
<td>all</td>
<td>10:52</td>
</tr>
<tr>
<td>RazerS 3</td>
<td>all</td>
<td>89:38</td>
</tr>
</tbody>
</table>

Intel Core i7, 16GB RAM
NVIDIA GeForce 780, 3GB RAM
### Performance

10 million human exome paired end reads:

<table>
<thead>
<tr>
<th>mapper</th>
<th>type</th>
<th>time [min:sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEANUT</td>
<td>best-stratum</td>
<td>3:08</td>
</tr>
<tr>
<td>BWA-MEM</td>
<td>best</td>
<td>4:44</td>
</tr>
<tr>
<td>Bowtie 2</td>
<td>best</td>
<td>8:18</td>
</tr>
<tr>
<td>PEANUT</td>
<td>all</td>
<td>21:54</td>
</tr>
<tr>
<td>RazerS 3</td>
<td>all</td>
<td>150:59</td>
</tr>
</tbody>
</table>

Intel Core i7, 16GB RAM
NVIDIA Geforce 780, 3GB RAM
Summary

PEANUT is a GPU based read mapper that outperforms other state-of-the-art mappers in terms of

- sensitivity
- speed

by introducing the q-group index with small memory footprint and exploiting

- bit-vector operations
- prefix sums
- population counts

http://peanut.readthedocs.org