

FastHASH: A New Algorithm for Fast and Comprehensive Next-generation Sequence Mapping

Hongyi Xin¹, Donghyuk Lee¹, Farhad Hormozdiari², Can Alkan³, Onur Mutlu¹

SAFARI

¹ Departments of Computer Science and Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA

² Department of Computer Science, University of California Los Angeles, CA

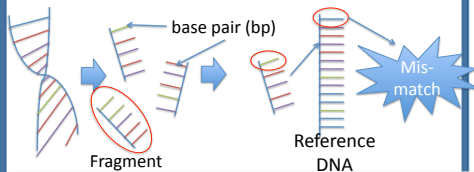
³ Department of Genome Sciences, University of Washington, Seattle, WA

Carnegie Mellon

Next-generation DNA Sequencing and the State-of-the-art Sequence Mapping Tools

Background: DNA Sequencing

- **Goal:** Acquire individual's entire DNA sequence
- **Mechanism:** Read DNA fragments and reconstruct it
 - Break DNA into pieces and store them as strings
 - Compare the strings to a known reference DNA string -- Search for matching coordinates in reference DNA
 - Stitch fragments together in corresponding order
- **Difficulties:** Individuals have mutations including
 - Mismatch, insertions and deletions; must tolerate



Challenge of Next-generation DNA Sequencing

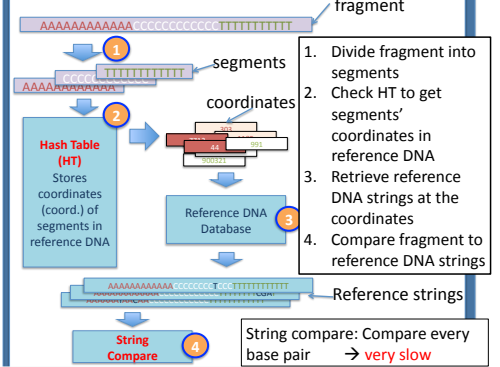
- **Next-generation DNA Sequencing:**
 - Instead of reading fewer long fragments, read many short fragments in parallel
 - This pushes the challenge to computation
- **Challenge:**
 - Shorter but many reads: billions of them
 - Mapping a fragment to entire reference genome is costly: cost does not reduce vs. a long fragment, and may increase for a shorter fragment
 - More potential mapping locations: harder to search for all possible matches in the reference DNA -- Even harder when mutations are allowed
- **Requirement:**
 - Algorithm that is fast and efficient which can process enormous amount of data

Existing Mapping Tools

- **Suffix tree or prefix tree based alignment tools:**
 - Newer tools use Burrows-Wheeler transformation -- Bowtie, BWA, SOAPv2
 - Advantage -- Fast in finding the exact match without mutations
 - Disadvantages
 - Very slow when mutations are allowed
 - Not comprehensive: does not search for all possible locations
- **Hash table based alignment tools:**
 - Use hash table for filtering non-matching coordinates -- mrFAST, mrsFAST
 - Advantage -- Comprehensive, and fast when comprehensive
 - Disadvantage -- Slower in searching for just the exact match

mrFAST

mrFAST Flow Chart



mrFAST: Two Key Components

- **Hash table (HT):**
 - Stores coordinates of segments in reference DNA
- | Segments | Coordinate list |
|--------------|-----------------|
| AAAAAAAAAAAA | 11 303 1105 |
| AAAAAAAAAAAC | 12 |
| AAAAAAAAAAAG | 229 400012 |
| AAAAAAAAAAAT | 304 798 4001451 |
| TTTTTTTTTTTT | 991 4991 800321 |
- Each segment looked up in HT to get coordinate list
 - For each coordinate in the list, look up reference string → **expensive**
- **String Compare:**
 - Compare input fragment against reference DNA
 - Check for mutations: mismatches, insertions and deletions (allow e mutations)
 - Need to compare every base pair → **very slow**

FastHASH

Our Goal and FastHASH

- **Problem with mrFAST:**
 - Slow: 5 hours to process 1M fragments (108 bp)
- **Our goal:**
 - Reduce the execution time while maintaining comprehensiveness
- **FastHASH Overview:** Two key components:
 - **Adjacency Filtering:** Reject obviously non-matching coordinates at early stage to avoid unnecessary expensive string comparisons
 - **Cheap segment selection:** Reduce the absolute number of coordinates that are subject to examination
- **Current Result:**
 - 38x speedup for 1M fragments compared to mrFAST

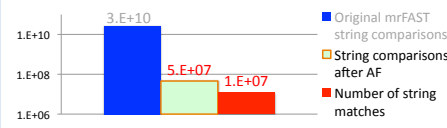
Our First Observation

- **String comparisons take too long**
 - 95% of execution time
-
- **Most string comparisons are useless: result in no match**
-

Adjacency Filtering (AF)

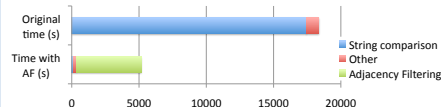
- **Goal:** Reduce the number of string comparisons
-
- **Observation:** If perfect match, consecutive segments should be at consecutive coordinates!
 - **Idea:** For a coordinate, check if consecutive coordinates are in the coordinate lists of consecutive segments
 - If yes → Do string comparison
 - If no → No need for string comparison
-

Effect of Adjacency Filtering



- String comparisons are drastically reduced: 3.7x speedup

Our Second Observation



- Adjacency Filtering becomes the bottleneck
- We can speed this up by avoiding the probing of long coordinate lists

Cheap Segment Selection (CSS)

- **Observation:** Hash table is imbalanced
 - **Cheap segments:** Segments that have few coordinates in hash table
 - **Expensive segments:** Segments that have many coordinates → lead to slow execution during AF
 - **Idea:** Select cheapest segments within a fragment
 - **Selecting the cheapest e+1 segments guarantees comprehensiveness** (at least one has no error)
 - **Example:** If $e = 1$, select the cheapest 2 segments
-
- **Effect of CSS:** The number of coordinates examined
-

Preliminary Results

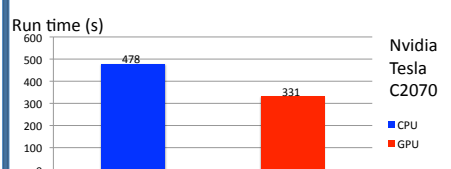
CPU Execution Time

- Input fragment set:
 - Fragment length: 108 base-pairs
 - Fragment size: 1 million
 - Number of errors: 3 mismatches, insertions or deletions
-
- Run time (s) Intel i7 2600 / 16 GB DRAM

Conclusions

- Adjacency Filtering provides 3.7x speedup
- Adjacency Filtering + Cheap Segment Selection provides 38x speedup

Preliminary GPU Execution Time of FastHASH



Conclusion

- GPU provides 1.44x speedup (early result)

Ongoing work

- Schedule work better on GPU for higher speedup