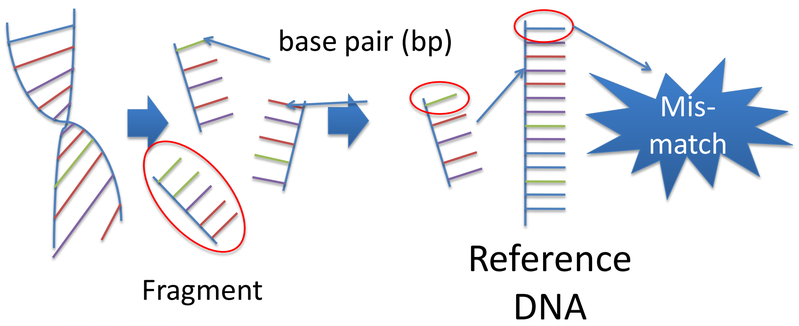
FastHASH: A New Algorithm for Fast and Comprehensive Next-generation Sequence Mapping

Hongyi Xin[1], Donghyuk Lee[1], Farhad Hormozdiari[2], Can Alkan[3], Onur Mutlu[1] [1]CMU, [2]UCLA, [3]University of Washington

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



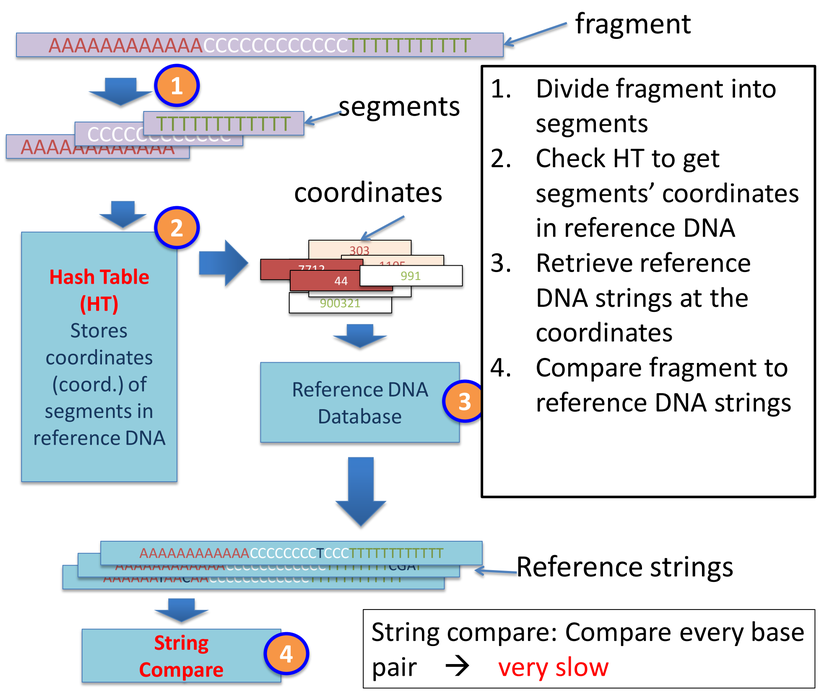
Next-generation DNA sequencing

- **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



fastHASH

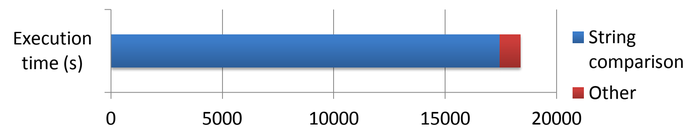
- **Problem with mrFAST:**
  - Slow: 5 hours to process 1M fragments (108 bp)
- **Our goal:**
  - Reduce the execution time while maintaining comprehensiveness

- **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

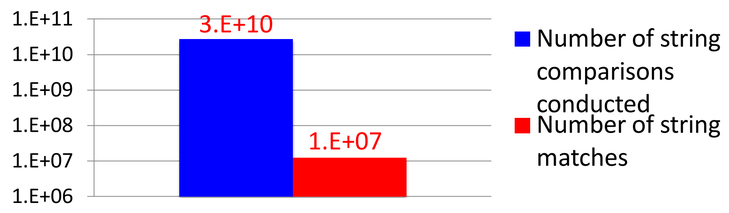
- **Current Result:**
  - 7x speedup for 1M fragments compared to mrFAST

Bottlenecks

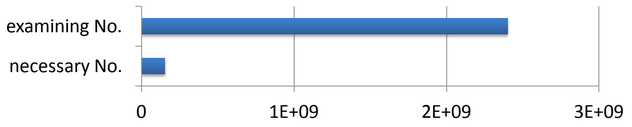
- **String comparisons take too long**
  - 95% of execution time



- **Most string comparisons are useless: result in no match**

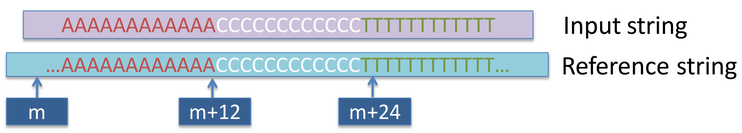


- **Examining too many unnecessary locations**

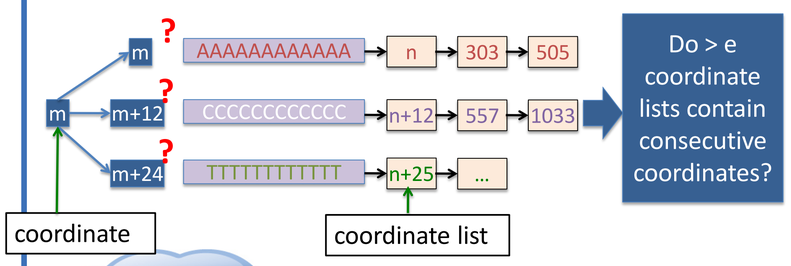


Adjacency Filtering (AF)

- **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

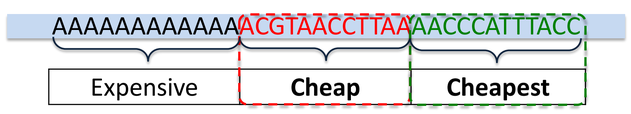


Cheap Segment Selection (CSS)

- **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)

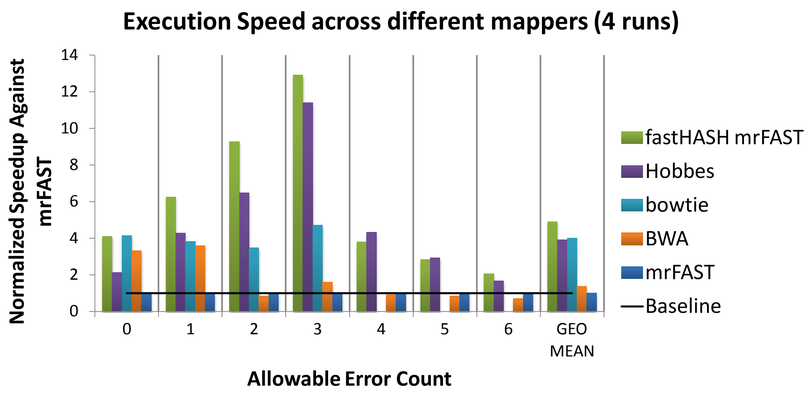
- **Example:** If e = 1, select the cheapest 2 segments

- **Effect of CSS:** The number of coordinates examined



Evaluation

- **Execution Time**
  - Fragment length: 108 base-pairs
  - Fragment size: 1 million



- **Future work**
  - Using GPU and FPGA to explore parallelism for further speedup

Intel Science & Technology Center for Cloud Computing

SAFARI

Carnegie Mellon University

Georgia Tech

intel

PRINCETON UNIVERSITY

UC Berkeley

UNIVERSITY of WASHINGTON