CO Final Project : Program-based Performance Analysis

NCTU 2020 Computer Organization

TA : Tzu-Chieh , Po-Shen

In this project you will learning…

- Basic knowledge
  - C/C++ Programming
  - Basic Linux Usage
- Impact of Architectural Optimizations and Tradeoffs
- Compute and Data intensive Applications
  - String Sorting
  - BWT Backward Searching

Schedule

Part	Contents	Scores	Due Date
I	Analysis	6	2020/5/7
II	Implementation	6	2020/5/28
III	Implementation	8	2020/6/11

Outline

- Introduction to Applications
  - Overlapping Strings – FM-Index
- Part I, II, III
- Grading Policy
- Other rules

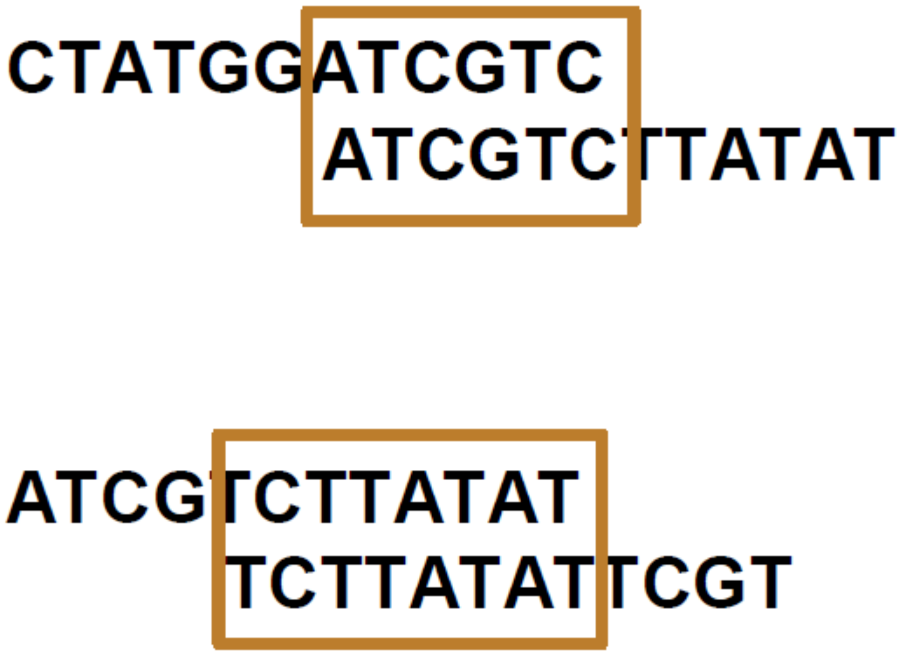
Overlapping strings

- Finding overlap between strings is a very common operation in various domains, particularly Computational Biology
- Given a set of Strings, find all the String pairs that overlap with each other from either of the ends
- **Performing String Overlaps efficiently** is critical to the applications that employee it, owing to **very large size** of data.

Overlapping Strings Illustration

String Set:

R0 : CTATGGATCGTC
R1 : ATCGTCTTATAT
R2 : ACGGATCATATA
R3 : TCTTATATTCGT



```
CTATGGATCGTC
      ATCGTCTTATAT

ATCGTCTTATAT
    TCTTATATTCGT
```

Overlappin tion

Requires $O(l^2)$ comparisons between each $O(n^2)$ number of pairs for string length l and string count n !

String Set:

R0 : CTATGGATCGTC
R1 : ATCGTCTTATAT
R2 : ACGGATCATATA
R3 : TCTTATATTCGT

$O(l^2n^2)$

```
    GTC
ATCGTCTTATAT
```

```
ATCGTCTTATAT
    TCTTATATTCGT
```

FM-Index for finding Overlaps

- Simple indexing structure
- Built using **suffixes** of each string
- Involves sorting of all suffixes (O(lnlog(ln))) lexicographically
- Low space complexity

Illustration

Position	0	1	2	3	4	5
Character	T	T	A	G	C	$

Suffix	Suffix Array	Rotation
TTAGC$	0	TTAGC$
TAGC$	1	TAGC$T
AGC$	2	AGC$TT
GC$	3	GC$TTA
C$	4	C$TTAG
$	5	$TTAGC

Sort

Rotation	Suffix Array
$TTAGC	5
AGC$TT	2
C$TTAG	4
GC$TTA	3
TAGC$T	1
TTAGC$	0

Contd…

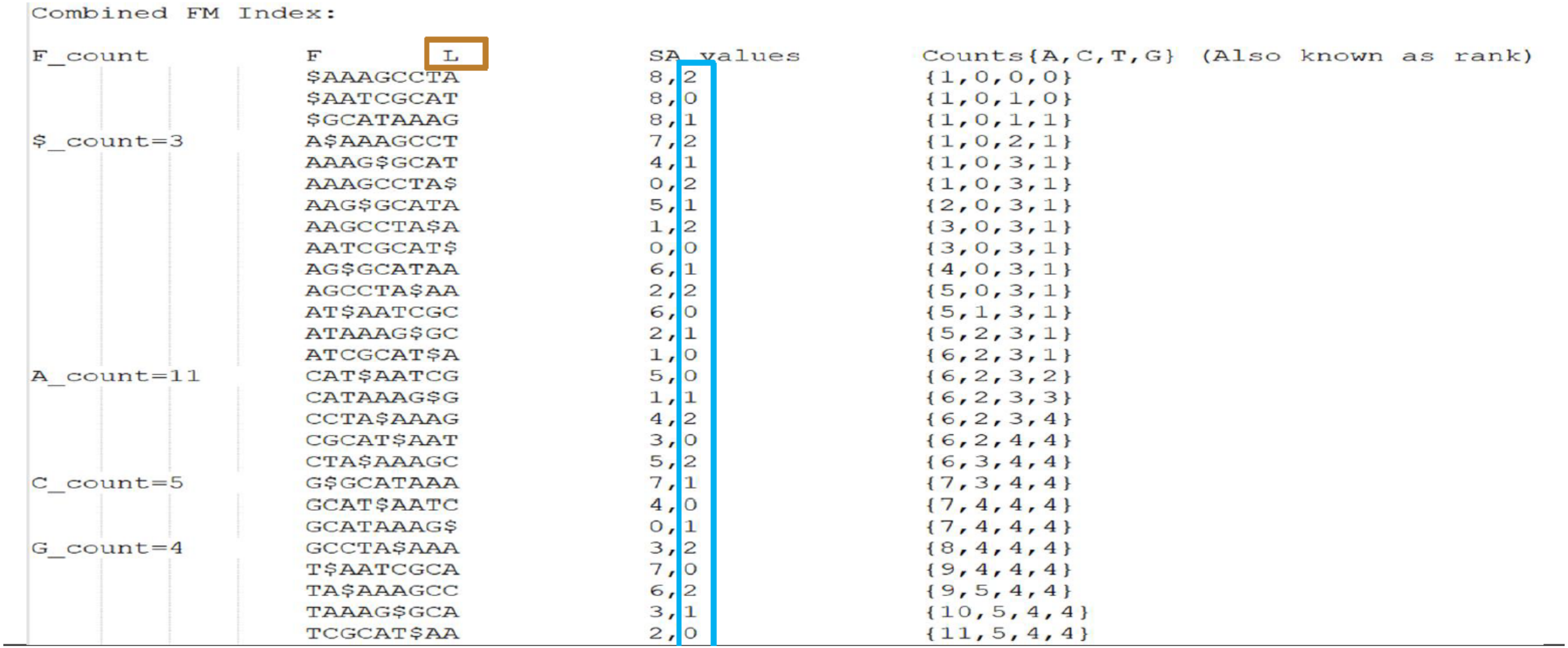
F_Count	Last Character	Suffix Array	Character Count			
			A	C	G	T
1	C	5	0	1	0	0
1	T	2	0	1	0	1
1	G	4	0	1	1	1
1	A	3	1	1	1	1
2	T	1	1	1	1	2
	$	0	1	1	1	2

Contd...

- To create FM-Index of multiple strings:
  - Create FM-Index for each string
  - Merge them together (**String Identification?**)
  - Do re-calculation of counts

R0:	**AATCGCAT**
R1:	**GCATAAAG**
R2:	**AAAGCCTA**

FM-Index for R0, R1 and R2



```
Combined FM Index:

F_count            F       L          SA values         Counts{A,C,T,G}  (Also known as rank)
                   $AAAGCCTA          8,2               {1,0,0,0}
                   $AATCGCAT          8,0               {1,0,1,0}
                   $GCATAAAG          8,1               {1,0,1,1}
$_count=3          A$AAAGCCT          7,2               {1,0,2,1}
                   AAAG$GCAT          4,1               {1,0,3,1}
                   AAAGCCTA$          0,2               {1,0,3,1}
                   AAG$GCATA          5,1               {2,0,3,1}
                   AAGCCTA$A          1,2               {3,0,3,1}
                   AATCGCAT$          0,0               {3,0,3,1}
                   AG$GCATAA          6,1               {4,0,3,1}
                   AGCCTA$AA          2,2               {5,0,3,1}
                   AT$AATCGC          6,0               {5,1,3,1}
                   ATAAAG$GC          2,1               {5,2,3,1}
                   ATCGCAT$A          1,0               {6,2,3,1}
A_count=11         CAT$AATCG          5,0               {6,2,3,2}
                   CATAAAG$G          1,1               {6,2,3,3}
                   CCTA$AAAG          4,2               {6,2,3,4}
                   CGCAT$AAT          3,0               {6,2,4,4}
                   CTA$AAAGC          5,2               {6,3,4,4}
C_count=5          G$GCATAAA          7,1               {7,3,4,4}
                   GCAT$AATC          4,0               {7,4,4,4}
                   GCATAAAG$          0,1               {7,4,4,4}
G_count=4          GCCTA$AAA          3,2               {8,4,4,4}
                   T$AATCGCA          7,0               {9,4,4,4}
                   TA$AAAGCC          6,2               {9,5,4,4}
                   TAAAG$GCA          3,1               {10,5,4,4}
                   TCGCAT$AA          2,0               {11,5,4,4}
```

Grading Policy

Goal : Accelerate String Sorting and BWT Search

- Part I :
  - (6 points) **Report :** Based on the algorithm and architecture of FM-Index, please propose a method to accelerate the establishment of FM-Index table.
- Part II:
  - (4 points) **Implementation :** Please speedup building FM-Index table .
  - (2 points) **Report :** Tell us what you did and what is the difference with ordinary backward search.
- Part III:
  - (6 points) **Implementation :** Please suggest ways to improve backward search on FM-Index and realized in your program.
  - (2 points) **Report :** Tell us how you did it and what is the difference with ordinary backward search.

Grading Policy

Goal : Accelerate String Sorting and BWT Search

- Scoring criteria :
  - **Implement :**
    - **Correctness : 70%**
    - **Speedup (Performance Rank) : 30%**
    - The fastest one will get 30% and the last one will get 1%
  - **Report :**
    - **Completion & Discussion : 80%**
    - **Detail explanation in your report : 20%**
    - **The report should include detail explanation and discussion of your design.**

Part I : Analysis (reference only)

- **Analyze the two processes of FM-Index :**
  - **String Sorting (with limited alphabet)**
  - **FM-Index Creation**
- **Analysis points:**
  - **Architectural:**
    - Cache-optimization, Loop-optimization, Strength Reduction, ILP, Memory optimizations and Tradeoffs, Shared Memory, SIMD Processing
  - **Algorithmic:**
    - Flow optimization, Parallelism, Data Partitioning

- **Online video for your reference:**
  - https://www.youtube.com/watch?v=kvVGj5V65io

Part I : Report (Goal : speedup)

- Proposed Algorithm (Detailed) – 30%
- Previous works here (Algorithmic only) – 30%
  - Another_reference
- Optimizations in your algorithm and architectural(Detailed) – 30%
- Is your design suitable for highly parallelizing? Why ? – 10%

Base Program For Part II and III

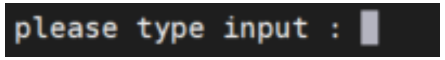
- Please refer to Github : https://github.com/Shalana/2020-CO-Final-Project

Part II
Goal : Please Speedup building FM-Index Table

- Completeness (4 points)
  - Your result must be correct (pass the check) – 70%
    - TA Test Pattern has been upload to Github
  - You get speedup compared to others– 30%
    - The fastest one will get 30% and the last one will get 1%
- Report (2 points)
  - Please describe your implementation algorithm and explain your results and all the optimizations you do– 100%

Part III
Goal : Please Improve Backward Search

- Completeness (6 points)
  - Your result must be correct .This means you must be able to give such a Output – 70%
    - Example:



```
please type input : █
```

ATCG

Your Output :

Number of reads match this substring: 5

  - Speedup (Performance Rank)– 30%
    - The fastest one will get 30% and the last one will get 1%

- Report (2 points)
  - How to run your program.
  - Describe your implementation algorithm and explain your results – 50%
  - Discussions on all the optimizations you do – 50%

Other rules

- It’s suggested that each team has 3-4 students.
  - Please list ID and name of your team members on part I report.
  - All members under one team will get the same grade.
- Compress your code and report into one zip file and upload to E3
  - Name your package as : LeaderID_FP1.zip
  - One team only need to upload one package to E3
  - Please name your report as : LeaderID_Report_FP1.pdf
  - Please make sure TA can compile your code and run.

Q&A

Could we use algorithm which had been published ?

For part I , You can use the algorithm of the published paper, and the comparison object (previous work) can be the code of the assistant on github

How to calculate the correctness scores of part II and part III

Thank you