Computer Organization Final Project

Part2

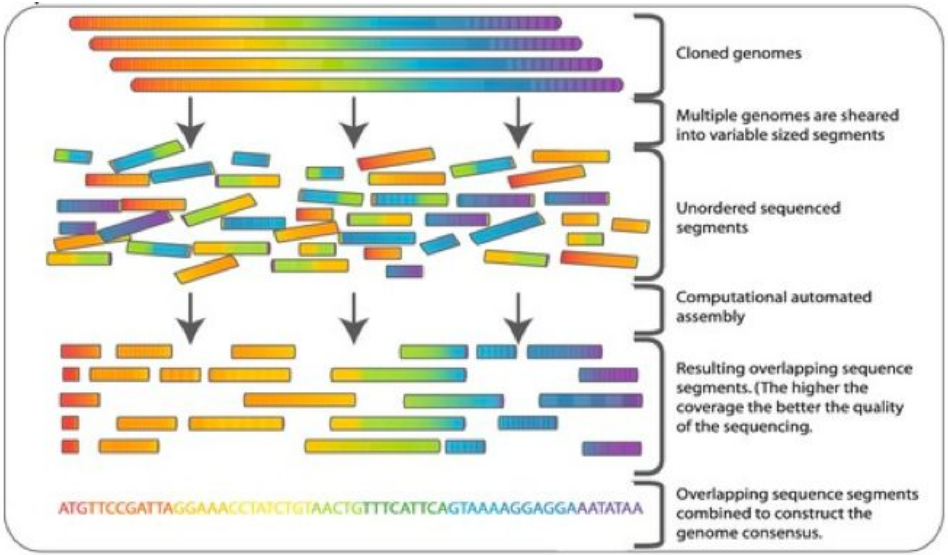
Contents

- Background
  - FM index
- Projects
  - Due dates
  - Part2
    - Contents
    - Grading policies
- Tutorial
  - FM index
  - Docker/gem5

FM index

BWA is a prominent software package for mapping sequences against a large reference genome, such as the human genome.

FM index is a critical operation taking a lot of time in the sequencing process, accelerating genome sequencing https://www.youtube.com/watch?v=kvVGj5V65io&ab_channel=BenLangmead



FM index

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

FM index

row	Rotation	Occ[A]	Occ[C]	Occ[G]	Occ[T]	SA
0	$TTAGC	0	1	0	0	5
1	AGC$TT	0	1	0	1	2
2	C$TTAG	0	1	1	1	4
3	GC$TTA	1	1	1	1	3
4	TAGC$T	1	1	1	2	1
5	TTAGC$	1	1	1	2	0

*But actually, we only need L[ ], Occ[] and SA to perform searching

Searching Algorithm - Find “TTA”

	$	A	C	G	T
Count	0	1	2	3	4

Define : Occ = occurence of character
last to first mapping L[i] = F[j]
LF[i] = C[char] + Occ[char,i] = j
ex: for “A”, L[3] = F[1] = A,
LF[3] = C[A] + Occ[A,3] - 1 = 1

Initial Range: find A
Min = C[A] = 1
Max = C[A+1] - 1 = C[C] - 1 = 1

row	L	Occ[A]	Occ[C]	Occ[G]	Occ[T]	SA
0	C	0	1	0	0	5
1	T	0	1	0	1	2
2	G	0	1	1	1	4
3	A	1	1	1	1	3
4	T	1	1	1	2	1
5	$	1	1	1	2	0

Searching Algorithm - Find “TTA”

	$	A	C	G	T
Count	0	1	2	3	4

Define : Occ = occurence of character

last to first mapping L[i] = F[j]

LF[i] = C[char] + Occ[char,i] = j

ex: for “A”, L[3] = F[1] = A,

LF[3] = C[A] + Occ[A,3] - 1 = 1

Initial Range: find A

Min = C[A] = 1

Max = C[A+1] - 1 = C[C] - 1 = 1

First round: find “T”

Min = C[T] + Occ[T, previous_min - 1] = 4

Max = C[T] + Occ[T, previous_max] - 1= 4

row	L	Occ[A]	Occ[C]	Occ[G]	Occ[T]	SA
0	C	0	1	0	0	5
1	T	0	1	0	1	2
2	G	0	1	1	1	4
3	A	1	1	1	1	3
4	T	1	1	1	2	1
5	$	1	1	1	2	0

Searching Algorithm - Find “TTA”

	$	A	C	G	T
Count	0	1	2	3	4

Define : Occ = occurence of character
last to first mapping L[i] = F[j]
LF[i] = C[char] + Occ[char,i] = j
ex: for “A”, L[3] = F[1] = A,
LF[3] = C[A] + Occ[A,3] - 1 = 1

Initial Range: find A
Min = C[A] = 1
Max = C[A+1] - 1 = C[C] - 1 = 1

First round: find “T”
Min = C[T] + Occ[T, previous_min - 1] = 4
Max = C[T] + Occ[T, previous_max] - 1= 4

Second round: find “T”
Min = C[T] + Occ[T,previous_min - 1] = 5
Max = C[T] + Occ[T,previous_max] - 1= 5

row	L	Occ[A]	Occ[C]	Occ[G]	Occ[T]	SA
0	C	0	1	0	0	5
1	T	0	1	0	1	2
2	G	0	1	1	1	3
3	A	1	1	1	1	3
4	T	1	1	1	2	1
5	$	1	1	1	2	0

Found at Row 5, according to SA, position in original string is SA [5]= 0

Sampling distance

- To further reduce memory space, sampling distance could be applied for Occ
  - Would require to reconstruct the Occ[] that are not sampled
  - Ex: sampling distance = 2

row	L	Occ[A]	Occ[C]	Occ[G]	Occ[T]	SA
0	C	0	1	0	0	5
1	T	-	-	-	-	2
2	G	0	1	1	1	3
3	A	-	-	-	-	3
4	T	1	1	1	2	1
5	$	-	-	-	-	0

Release & Due Dates

Part1:

4/27 Release ~ 5/11(midterm) Due

Part2:

5/18 Release ~ 6/22(A week after Final) Due

Grading Policies

Part1 (8%)

- Report (5%)
- Code Functionality(1%)
- Performance (2%)

Part2 (12%)

- Report (8%)
- Code Functionality(2%)
- Performance (2%)

Part2

Files to Hand: CO2021_Student_id _Name.pdf , CO2021_Student_id_FP2.cpp,
CO2021_Student_id_cmd.txt
*please follow naming rule, otherwise there would be a 1.5% penalty

Report:

1. Profile FM index with gem5, and identify its bottleneck as done in part 1.(2%)
2. Compare FM-index with suffix array, discuss the difference in the algorithm and the space required for the data structure. (2%)
3. Modify the Sampling distance and discuss its impact on searching part (Memory access,Compute time, Ratio...)(2%)
4. Accelerate the sample code and try to modify the cache to achieve better performance. Discuss why you did what modification ? (2%)

Coding:

Correctness (1%) * We will verify the STDOUT for correctness so please do not disable the cout<<
Improve the program provided (hints would be provided) (1%)
The speed up only (final_tick) would be competited for performance score (2%)

*Note: Please stick the algorithm to FM index, and please do not use any define or pragma in the code

How to start

1. Download FM_package.tar from e3
2. Copy file into docker:

   docker cp FM_package.tar repo_gem5: /usr/local/src/gem5_profile
3. Enter docker environment

   docker start repo_gem5

   docker exec -ti repo_gem5 bash

   cd gem5_profile
4. Extract the files from FM_package.tar with:

   tar -xvf FM_package.tar

```
root@9e2016e18801:/usr/local/src/gem5_profile# tar -xvf FM_package.tar
patterns.txt
reference_string.txt
FM_index/
FM_index/FM_index.cpp
FM_index/Makefile
FM_index/reference_string.txt
FM_index/main
FM_index/patterns.txt
root@9e2016e18801:/usr/local/src/gem5_profile#
```

How to start

- To modify code:
  - FM_index/FM_index.cpp → modify source code
  - *make* → compile
- To run gem5 simulation:

  build/X86/gem5.opt configs/example/se.py -c FM_index/main --cpu-type=DerivO3CPU --l1d_size=64B --l1d_assoc=1 --l1i_size=16kB --caches --mem-type=DDR4_2400_8x8 --mem-channels=1 --mem-size=4GB

```
root@9e2016e18801:/usr/local/src/gem5_profile# tar -xvf FM_package.tar
patterns.txt
reference_string.txt
FM_index/
FM_index/FM_index.cpp
FM_index/Makefile
FM_index/reference_string.txt
FM_index/main
FM_index/patterns.txt
root@9e2016e18801:/usr/local/src/gem5_profile#
```

Part2 HINTS and Chart

For question 1, fill in the below chart to answer the answers:

	system.cpu.workload.profile_totaltime	profile_totMemAccLat_wo_overlap
Number of ticks		

*tick = pico second

You can also refer to the following informations in ***m5out/stats.txt*** to write the report:

- final_tick // total simulated time
- system.cpu.dcache.overall_miss_rate::total // cache miss rate
- system.cpu.workload.profile_totaltime //total execution time in region of interest
- profile_totMemAccLat_wo_overlap // memory access time in region of interest

Part2 HINTS and Chart

For question 3, modify the parameters concerning cache in the command:

build/X86/gem5.opt configs/example/se.py -c FM_index/main --cpu-type=DerivO3CPU --l1d_size=64B --l1d_assoc=1 --l1i_size=16kB --caches --mem-type=DDR4_2400_8x8 --mem-channels=1 --mem-size=4GB

You can also refer to the following informations in ***m5out/stats.txt*** to write the report:

- final_tick // total simulated time
- system.cpu.dcache.overall_miss_rate::total // cache miss rate
- system.cpu.workload.profile_totaltime //total execution time in region of interest
- profile_totMemAccLat_wo_overlap // memory access time in region of interest

Part2 HINTS and Chart

For acceleration, here are some hints for acceleration:

1. Observe the provided strings and patterns
2. Data type / Encoding for L column / occ
3. Coding Style

You can modify all of the code provided, including building and searching part, but do not use data structure other than FM-index.

You can also refer to the following informations in ***m5out/stats.txt*** to write the report:

- final_tick // total simulated time
- system.cpu.dcache.overall_miss_rate::total // cache miss rate
- system.cpu.workload.profile_totaltime //total execution time in region of interest
- profile_totMemAccLat_wo_overlap // memory access time in region of interest

How to simulate a Cpp application?

1. start container : *$ docker start repo_gem5*
   $ docker exec -ti repo_gem5 bash
2. enter gem5/ directory:*$ cd gem5_profile*
3. build gem5 with *$ scons build/X86/gem5.opt -j 9*
4. modify code FM_index/FM_index.cpp then compile with: *$ make*
5. rum simulation for the code with:
   $ build/X86/gem5.opt configs/example/se.py -c FM_index/main --cpu-type=DerivO3CPU --l1d_size=64B --l1d_assoc=1 --l1i_size=16kB --caches --mem-type=DDR4_2400_8x8 --mem-channels=1 --mem-size=4GB

docker tutorial

1. To restart the container and enter its command line:

   docker start repo_gem5

   docker exec -ti repo_gem5 bash
2. copy file from container to local

   docker cp repo_gem5: <PATH_IN_CONTAINER> <FILE_NAME>
3. copy file from local to container

   docker cp <FILE_NAME> repo_gem5: <PATH_IN_CONTAINER>

Linux command line short tutorial

cd dir #change directory, enter the directory “dir”

cd .. #change directory, exit the current directory

ls #list the files in the current directory

mkdir dir #make directory “dir”

make # compile the cpp program