Computer Organization Final Project

Part1

Contents

- Background
  - string matching
  - gem5
- Projects
  - Due dates
  - Part1
    - contents
    - Grading policies
  - Part2
    - contents
    - Grading policies
- Tutorial
  - docker
  - gem5

Main direction

Profile String Matching Algorithm with Gem5 to accelerate applications and answer questions .

Why string matching algorithm?

String matching algorithm is a very robust application these days.
For instance, spelling checker, spam filters are applications that we might have experience of.
Moreover, it can play a crucial role while doing genome analysis to find specific DNA patterns, helping doctors to identify symptoms, making the future of human beings better.

A C T C G C A A T A T G C T A G G C C A G C

A C T _ _ _ _ T T A T G C T A T G C _ _ G C

A C T T G T C T T A T G C

A C T _ G _ _ T T A _ _ C

Genome string length could be billions of characters, making it a terrifying workload to match any single string in it

What is gem5 simulator ?

A modular platform for computer-system architecture research.
Can help investigate impact of microarchitechture to applications.
For example, the cache size, cpu frequencies and memory architecture could all be configured.

What information can we get and for what?

Timing, memory bandwidth, and details on executed instructions.

```
---------- Begin Simulation Statistics ----------
final_tick                                 1070089506                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset)
host_inst_rate                                106165                       # Simulator instruction rate (inst/s)
host_mem_usage                             126015372                       # Number of bytes of host memory used
host_op_rate                                  158881                       # Simulator op (including micro ops) rate (op/s)
host_seconds                                    5.15                       # Real time elapsed on the host
host_tick_rate                              207583843                       # Simulator tick rate (ticks/s)
sim_freq                                 1000000000000                       # Frequency of simulated ticks
sim_insts                                     547269                       # Number of instructions simulated
sim_ops                                       819025                       # Number of ops (including micro ops) simulated
sim_seconds                                  0.001070                       # Number of seconds simulated
sim_ticks                                  1070089506                       # Number of ticks simulated
```

```
system.cpu.commit.op_class_0::No_OpClass          366      0.04%      0.04% # Class of committed instruction
system.cpu.commit.op_class_0::IntAlu           611895     74.71%     74.75% # Class of committed instruction
system.cpu.commit.op_class_0::IntMult          10021      1.22%     75.98% # Class of committed instruction
system.cpu.commit.op_class_0::IntDiv              21      0.00%     75.98% # Class of committed instruction
system.cpu.commit.op_class_0::FloatAdd            10      0.00%     75.98% # Class of committed instruction
system.cpu.commit.op_class_0::FloatCmp             0      0.00%     75.98% # Class of committed instruction
system.cpu.commit.op_class_0::FloatCvt             0      0.00%     75.98% # Class of committed instruction
system.cpu.commit.op_class_0::FloatMult            0      0.00%     75.98% # Class of committed instruction
system.cpu.commit.op_class_0::FloatMultAcc          0      0.00%     75.98% # Class of committed instruction
```

Release & Due Dates

Part1:

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

Part2:

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

Grading Policies

Part1 (8%)

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

Part2 (12%)

- Report (6%)
- Code Functionality(2%)
- Performance (4%)

Part 1

Files to Hand: CO2021_Student_id _Name.pdf , CO2021_Student_id_FP1.cpp

Report:

1. What is the bottleneck of string matching with suffix array, data access or compute complexity ? (Compare the memory access time and compute time) Please fill the chart in next page and use its contents to explain its bottleneck. (3%)
2. Explain your code, how did you speed up your code? And how is the cache miss rate improved? Please provide snapshots of your code and use at least 200 words to explain how. (2%)

Code:

Correctness (1%)

Improve the cache miss rate (the lower the better) and Modify the code to make it run faster (lower final_tick) . (1%)
The speed up only (final_tick) would be competited for performance score (1%)

Part1 HINTS and Chart

For question1, fill in the below chart to answer the answers:

	system.cpu.workload.profile_totaltime	profile_totMemAccLat_wo_overlap
Number of ticks		

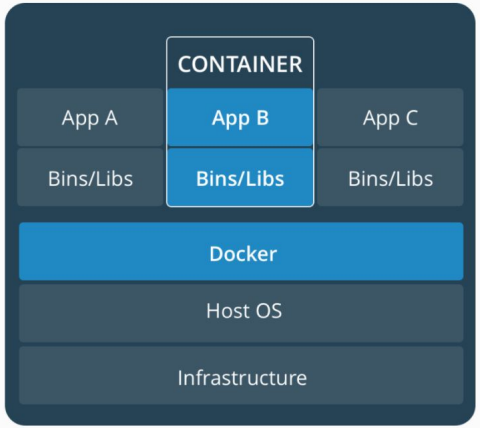
*tick = pico second

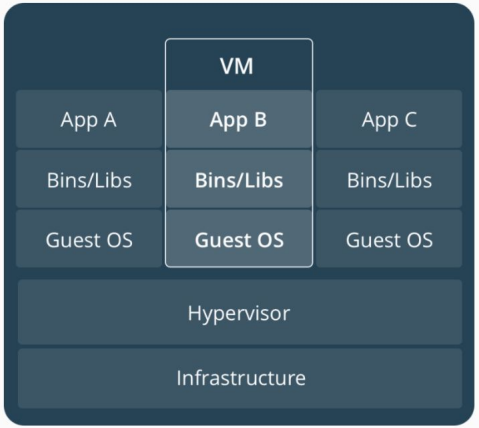
You can also refer to the following informations in ***m5out/stats.txt*** to write the report:

- 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

docker tutorial

What is a docker?





CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

docker tutorial

Install docker desktop for windows:

1. Download here https://hub.docker.com/editions/community/docker-ce-desktop-windows
2. Restart the machine and hold F1/ F2 / del /Esc while booting to enter BIOS
3. Enable Virtualization https://www.youtube.com/watch?v=ZDeje9wgDp4
4. Download and install the Linux kernel update package: https://docs.microsoft.com/zh-tw/windows/wsl/install-win10#step-4---download-the-linux-kernel-update-package
5. Start docker engine

docker tutorial

5. download image, create and start container (do this the first time):

   docker run -ti --name repo_gem5 kyens0612/gem5_profile:latest

   Would enter the environment for gem5 then

6. use Ctrl + d to exit container
7. To restart the container and enter its command line:

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

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

   docker cp <FILE_NAME> repo_gem5: <PATH_IN_CONTAINER>

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 String_matching/S.cpp then compile with: *$ make*
5. rum simulation for the code with:
   $ build/X86/gem5.opt configs/example/se.py -c String_Matching/main --cpu-type=DerivO3CPU --l1d_size=64B --l1d_assoc=1 --l1i_size=16kB --caches --mem-type=DDR4_2400_8x8 --mem-channels=1 --mem-size=16GB

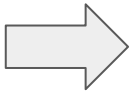
Suffix Array - Binary Search Tutorial

String matching algorithm - **Suffix array** Binary Search

Suffix array provides the starting positions of suffixes a string in **lexicographical** order
*adds “$” to hint end of string
* ‘ ’ < ‘a’ < ‘b’ < ‘c’ …. < ‘z’ < ‘$’

Can help accelerate the string matching process

Example of suffix array building of “banana”



Suffix	i
banana$	1
anana$	2
nana$	3
ana$	4
na$	5
a$	6
$	7

Suffix	i
$	7
a$	6
ana$	4
anana$	2
banana$	1
na$	5
nana$	3

String matching algorithm - Suffix array **Binary Search**

```
n = len(S)
def search(P: str) -> Tuple[int, int]:
    """
    Return indices (s, r) such that the interval A[s:r] (including the end
    index) represents all suffixes of S that start with the pattern P.
    """
    # Find starting position of interval
    l = 0  # in Python, arrays are indexed starting at 0
    r = n
    while l < r:
        mid = (l + r) // 2  # division rounding down to nearest integer
        # suffixAt(A[i]) is the ith smallest suffix
        if P > suffixAt(A[mid]):
            l = mid + 1
        else:
            r = mid
    s = l

    # Find ending position of interval
    r = n
    while l < r:
        mid = (l + r) // 2
        if suffixAt(A[mid]).startswith(P):
            l = mid
        else:
            r = mid - 1
    return (s, r)
```

First find leftmost position **L** of the matched string, then find the rightmost position **R** of the matched string

Suffix	i
$	7
a$	6
ana$	4
anana$	2
banana$	1
na$	5
nana$	3

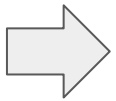
String matching algorithm - find “ana” -> L

String = “banana$”
pattern = “ana”
init: L=0 R=6

iter 0: mid = 3
“ana” < “anana$”
R = mid
=> L=0 R=3

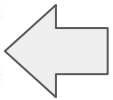
iter 1: mid = 1
“ana” > “a$”
L = mid+1
=> L=2 R=3

iter 2: mid = 2
“ana” < “ana$”
R = mid
=> L=2 R=2



FINAL L = 2

Suffix	i
$	7
a$	6
ana$	4
anana$	2
banana$	1
na$	5
nana$	3



String matching algorithm - find “ana” -> R

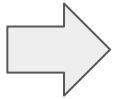
String = “banana$”
pattern = “ana”
init: L=0 R=6

iter 0: mid = 3
“anana$” starts with “ana” == true
L = mid
=> L=3 R=6

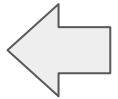
iter 1: mid = 5
“na$” starts with “ana” == false
R = mid -1
=> L=3 R=4

iter 2: mid = 4
“banana$” starts with “ana” == false
R = mid -1
=> L=3 R= 3

FINAL R = 3



Suffix	i
$	7
a$	6
ana$	4
anana$	2
banana$	1
na$	5
nana$	3



Time complexity

Finding a pattern with length m in string with length n,
the time complexity is O(m+logn)
given that a single suffix comparison needs to compare m character

Compared with naive string matching, the naive method would have time complexity of O(n^2)

*disadvantage: need space compexity O(n log n) to store the suffix array

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

Part2

Files to Hand: CO2021_Student_id _Name.pdf , CO2021_Student_id_FP2.cpp

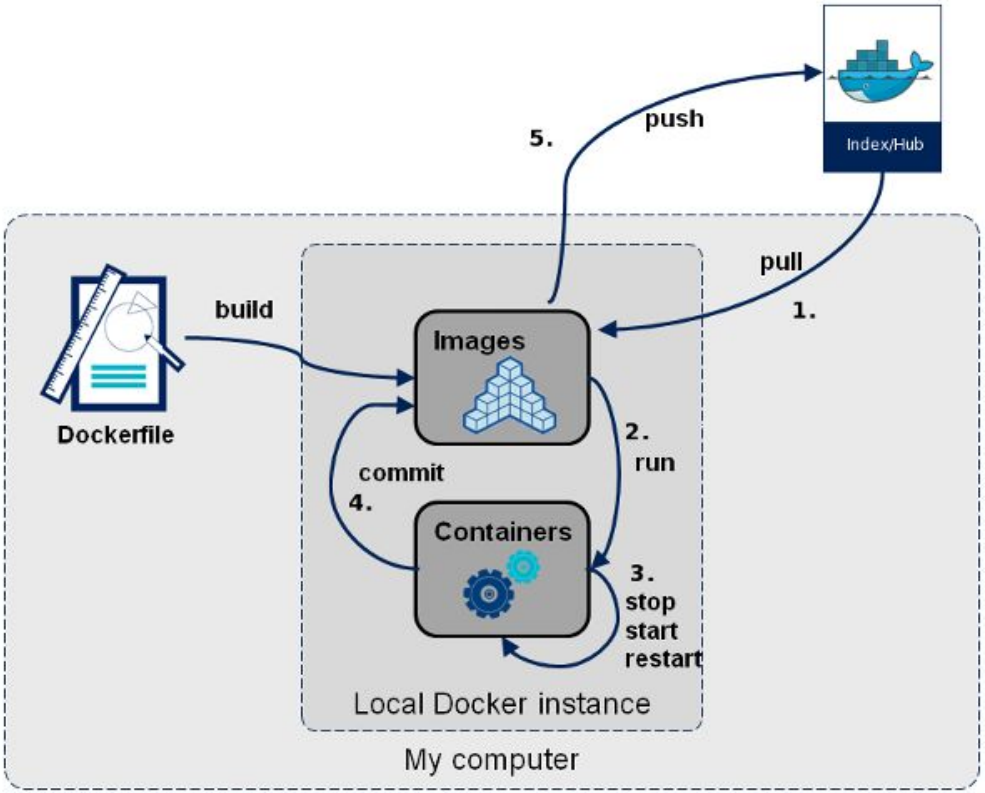
Report:

1. Profile FM index with gem5, compare its performance compared with suffix array.
2. Analyze the data placement impact with ramulator, what kind of placement policiy would result in better timing?
3. Accelerate the sample code and try modify the cache size to achieve better performance. Discuss how you did it ?

Coding:

Please accelerate the FM index code provided

*Tool Tutorial will be released video tutorial before release, please send email if you have any questions


Index/Hub
5.
push
pull
1.
build
Dockerfile
Images
2.
run
commit
4.
Containers
3.
stop
start
restart
Local Docker instance
My computer