LAB 4 - DRAM Memory Controllers Scheduling

1 General Instructions

- ◇ start by accepting the assignment link, which will create the repo for you on your github. Then clone this repo into the VM.
- ◇ Make sure you use the provided VM for developing and running the labs. We will not help debugging any other setup.
- ◇ In the lab room machines, you can login in to the VM with your group number. You will be asked to enter a password for the first time. Feel free to copy the VM into your own laptop to work on the labs at your time.
- ◇ clone the repository into a folder named macsim-4dm4-lab4; all instructions in this document assumes this folder as the main repo folder.
- ◇ In addition to the code modifications and output files, you are asked to prepare a pdf report of all the steps and answering the questions within the lab document. The first page of the report should be a declaration of contribution where each of the two members of the group lists clearly what they have done.
- ◇ All the files you are asked to produce or modify (including source, configuration, trace files, output files, and report) should be placed under labs/lab4 folder.
- ◇ The submission of the code, output files, and the lab report should be done by pushing to the repository before the deadline time.
- ◇ In addition to the declaration of contribution, we will be using github commit history to track the contributions of the group. Therefore, it is very critical to commit and push updates one at a time and do not wait until you finish the full lab to push once. This will be considered a red flag.
- ◇ Finally, please make sure you only push the required files, the report, the parameter files you modify for each step, plus any source files you modified. Log files and benchmark source files are not required.

2 Submission Deadline

The deadline for lab4 is **Tuesday Nov 21st, 11:59PM**. Please note that this is a programmed deadline in the environment, so make sure you submit in time since you will not be able to submit after that dictated deadline.

2.1 Importing the starter code of the lab

You should follow exactly the same process you have been following in past labs to import the starter code from the following invitation link and to create the project:

https://classroom.github.com/a/eNqIMwRy

3 MCSim – Getting Started

MCsim is a cycle-accurate DRAM memory controller simulation platform designed in C++ that takes benefit of the extensibility of classes. MCsim provides an interface to connect with external hardware simulators such as CPU and memory system simulators. With virtual functions, MCsim provides a simple interface for the designer to develop a scheduling policy effectively without re-implementing the other parts of the hardware.

- ◇ Start by reading the MCSim paper here: https://www.ece.mcmaster.ca/faculty/hassan/assets/publications/reza_MCsim_CAL.pdf
- ◇ Go through the README file in your repository

3.1 Run your first MCSim simualtion

- ◇ Run your first simulation by following these commands.
  - Understand what does each input argument mean (note that the run command should be all in one line..we are breaking it for readability in the document.
  - check the output file and make sense of what does it print

```
cd src
make
./MCsim -n 1 -s ../system/FRFCFS/FRFCFS.ini -G DDR3 -D 1600H -S 2Gb_x8 -R 1 -t Mem_Trace/1M_seq.trc
&> first_sim.out
```

4 Intra-Bank Scheduling

In this experiment, you are going to compare two common scheduling techniques to schedule memory requests: FCFS and FR-FCFS.

4.1 Single Core

- ◇ Look into the two systems' files and identify the key differences. Use code snippets to explain your arguments in the report.
- ◇ Run the following four experiments for the same configuration as in previsous section but change the systems and the trace input arguments accordingly and keep the output files named as instructed:
  1. FCFS with 1M_seq.trc, put the output in FCFS_1core_seq.out

2. FCFS with 1M_rand.trc, put the output in FCFS_1core_rand.out
3. FRFCFS with 1M_seq.trc, put the output in FRFCFS_1core_seq.out
4. FRFCFS with 1M_rand.trc, put the output in FRFCFS_1core_rand.out

◇ Draw a bar graph for the execution time of the four experiments, by following the template in Figure 4.1. Comment on the results.

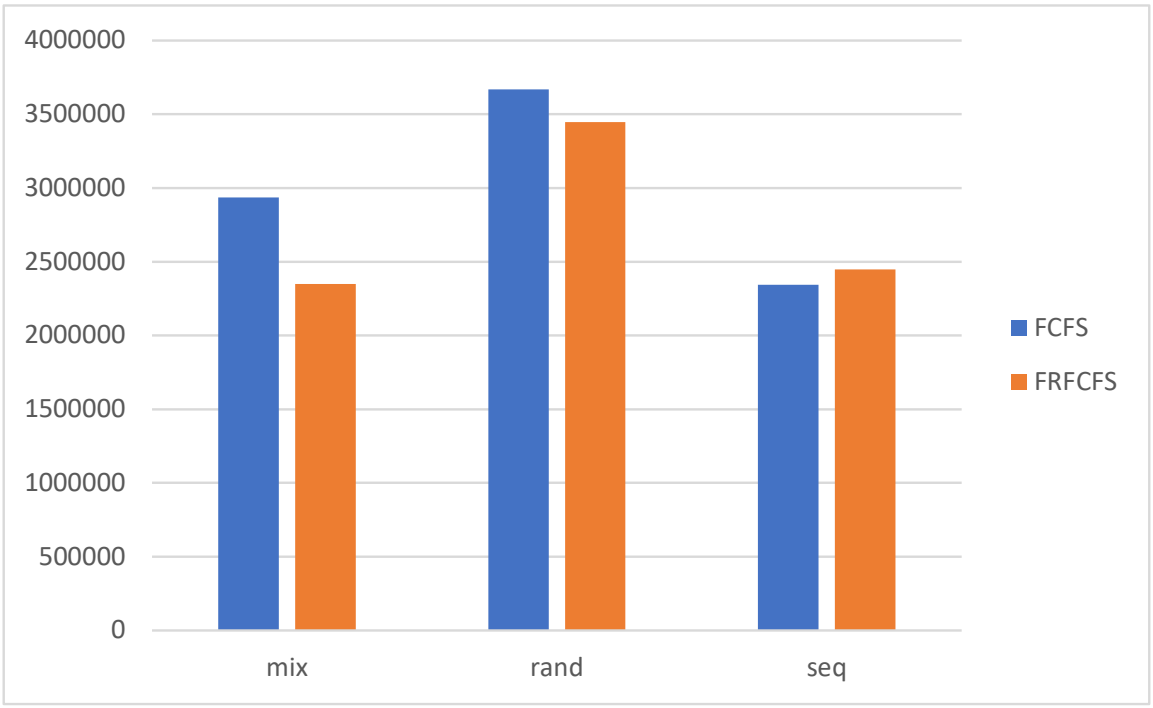

Figure 1: Example of the Figure you should draw for the FCFS vs FRFCFS single-core experiments.

4.2 Dual Core

Now, we are going to use the same two systems (FCFS and FRFCFS) but the memory is going to be shared by two cores (requestors).

◇ Look into the main.cpp file and identify how does the multi-requestor setup attach traces to requestors and understand how it interfaces the same memory to multi-requestors.

◇ Run the following three experiment for both FCFS and FRFCFS (i.e. 6 in total) and name the output files as given, where system will be either FCFS or FR-FCFS

1. one core runs the 1M_seq.trc trace and the other runs the 1M_rand.trc, name this [system]_2core_mix.out
2. Both cores run the 1M_seq.trc trace, name this [system]_2core_seq.out

3. Both cores run the 1M_rand.trc trace, name this [system]_2core_rand.out

◇ Draw a bar graph for the execution time of the six experiments by following the template in Figure 4.2. Comment on your results.

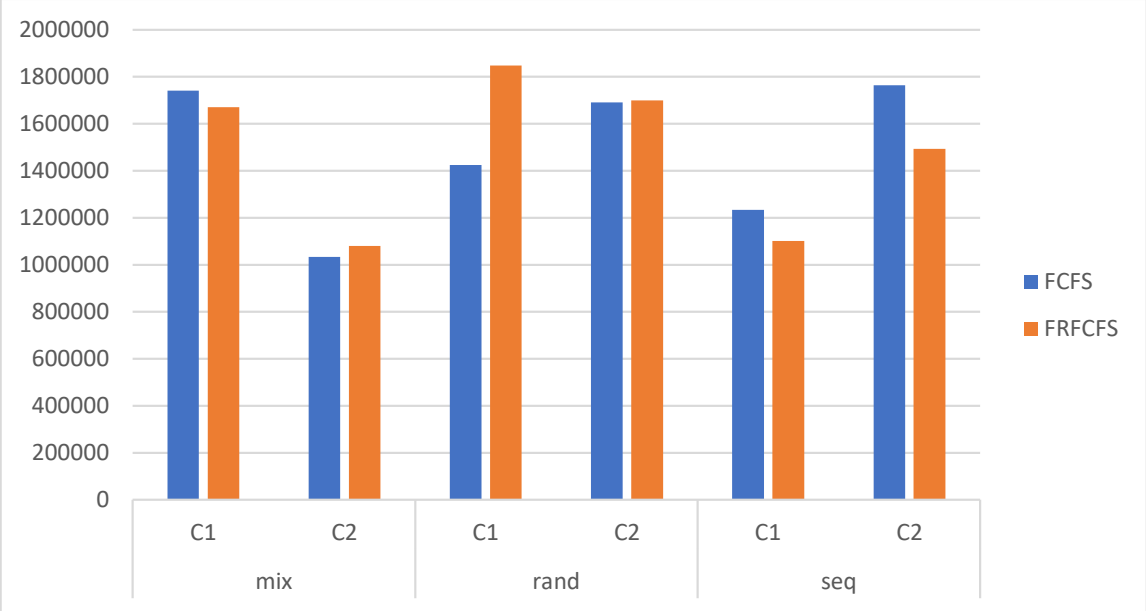

Figure 2: Example of the Figure you should draw for the FCFS vs FRFCFS dual-core experiments.

5 Read/Write Scheduling

Some COTS MCs prioritize reads over writes because read accesses are more latency sensitive than write accesses. They queue write accesses into a dedicated write buffer and service them in batches. This mechanism mitigates the turn-around time required to switch the data bus between reads and writes. This is because the controller will be servicing batches of same type (either consecutive reads or writes) instead of alternating between both types. Accordingly, this mechanism can significantly improve memory throughput.

The system implements a commonly known write batching technique called dual water mark. Go through system/FRFCFS_Batch/RequestScheduler_FRFCFS_Batching.h to understand how does this technique works.

5.1 Traces

Now, to be able to test the impact of read/write scheduling, we need sequence of accesses with both reads and writes. To avoid the impact of hits/conflicts that we already explored, we are going to use the sequential trace, 1M. However, when you open it and count the READS/WRITES, you will realize that it has a 10% Writes. To study the impact of this

scheduling, we need to experiment with various percentages. Therefore, you are required to create the following traces:

1. All reads: this can be easily achieved by replacing all WRITE in the 1M with READ, name it 1M_RD
2. All writes: can be created using similar way as above, name it 1M_WR

5.2 Experiments

Now using these three traces, and using a dual-core setup, we are going to run the following four experiments twice: once for the FRFCFS and the second for FRFCFS_Batching:

1. Both cores have all reads, name output [system]_2core_RD.out
2. Both cores have all writes, name output [system]_2core_WR.out
3. One core runs the 1M_RD and the other runs the 1M_WR, name the output [system]_2core_RD_WR.out
4. both cores run the original 1M trace, name the output [system]_2core_MIX_MIX.out

◇ Draw a bar graph for the execution time of the eight experiments by following the template in Figure 5.2. Comment on the results.

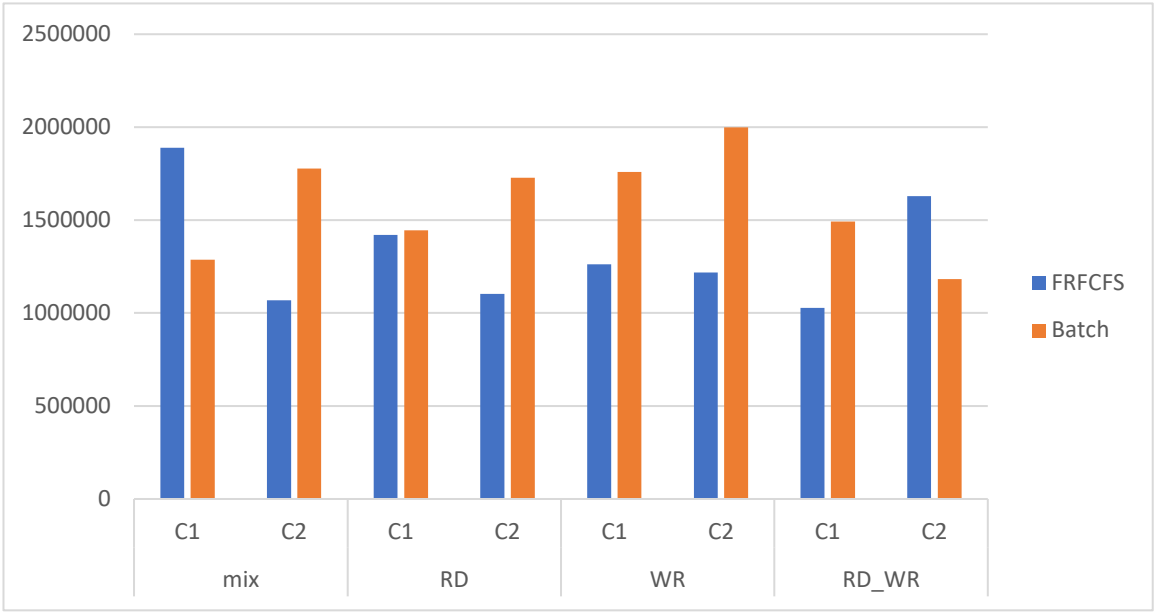

Figure 3: Example of the Figure you should draw for RD/WR scheduling experiment.

6 Add your Own Controller

We discussed in Section 4 both FR-FCFS and FCFS. FRFCFS favors Hit requests over Row Conflict ones. Despite being generally good for average performance, it can lead to starvation. To avoid starving the conflict requests, modern memory controllers typically deploy a threshold mechanism on top of FR-FCFS. This mechanism can be implemented with various flavors. We assume a simple implementation as follows. At most HIT_Threshold Hit requests can be reordered ahead of any other request targeting the same bank.

- ◇ Start from the FR-FCFS as an example of a system, use the same convention as in all existing systems, build your own system named FRFCFS_THRESHOLD that should be identical to FR-FCFS except that it adds this threshold on the reordering.
- ◇ You should have a new folder under systems and similar header and ini files.