ECEC-623

Project One: Introduction to Bank-level Parallelism and Memory Controller Design

Instructor: Dr. Anup Das
TA: Shihao Song (ss3695@drexel.edu)

1 Introduction

This project is intended to be a comprehensive introduction to memory bank-level parallelism and memory controller design. There are three parts to this project:

- In part one, you will evaluate the performance of an FCFS memory controller with workloads from SPEC CPU 2017.
- In part two, you will perform the same experiment as in part one but with Phase-change Memories (PCM) timings.
- In part three, you will design an FR-FCFS memory controller and evaluate its performance against parts one and two.

2 Development Environment

- Operating System: Linux.
- Simulator: `https://github.com/Shihao-Song/Computer-Architecture-Teaching`. Please *git clone* the repository to your Linux machine:

```
$ git clone https://github.com/Shihao-Song/Computer-Architecture-Teaching
```

3 Simulator Overview

1. You will be working under directory *Computer-Architecture-Teaching/C623/Memory_Controller*. To navigate to the directory:

```
$ cd Computer-Architecture-Teaching/C623/Memory_Controller/
```

2. To compile and run the simulator:

```
$ make
$ ./Main sample.workload
```

3. You should be able to see the following output:

```
End Execution Time: 150055192
```

4 Introduction to Memory Architecture

Every last-level cache (LLC) miss or eviction needs to be served by the DRAM. Figure 1 shows a sample architecture of a DRAM. In the beginning, all the DRAM requests either reads or writes are inserted in the transaction queue of a memory controller; the memory controller then schedules a request and sends it to DRAM via a memory channel that contains one rank and eight banks per rank. Each bank within a rank can operate individually, which referred to as *bank-level parallelism*. For example, suppose there are two read requests, R1, and R2 in the transaction queue, and it takes 50 cycles to serve a read request. If R1 and R2 both target to Bank 0, R2 has to wait for R1 to complete to get scheduled, so in total, it takes 100 cycles to serve R1 and R2, we call this situation (two consecutive requests targeting to the same bank) as a *bank-conflict*. However, if R1 aims Bank 0 while R2 targets Bank 1, both requests can be served simultaneously, which improves system performance significantly.

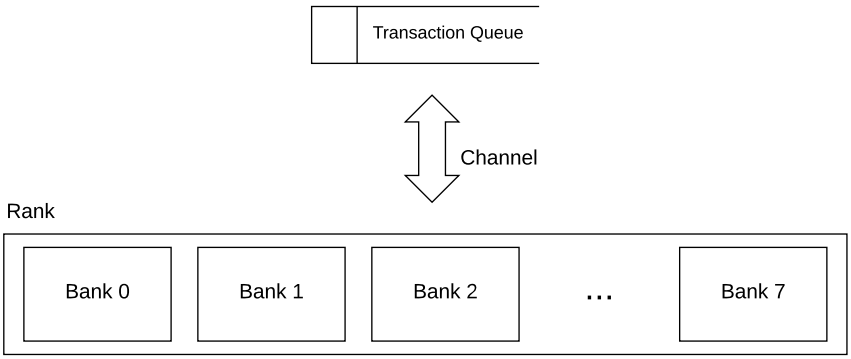

Figure 1: DRAM Architecture

5 Experiment

1. The simulator allows you to change the number of banks in a memory channel. To configure the number of banks:

   ```
   $ vim Controller.h
   ```

   The following snapshot shows a memory channel with 2 banks in total.

   ```
   static unsigned NUM_OF_BANKS = 2; // number of banks
   ```

2. Compare the end of executions with NUM_OF_BANKS set to 2, 4, 8, and 16.

3. One critical factor for performance evaluations is the access latency. Access latency is the number of (memory) clock cycles between the time a request is inserted in the transaction queue and the time it's scheduled. Implement a function that calculates the average access latency with NUM_OF_BANKS set to 2, 4, 8, and 16.

4. The existing memory controller equips with an FCFS scheduler, which serves the request in the same order of insertion. Implement a function that counts the number of bank-conflicts with NUM_OF_BANKS set to 2, 4, 8, and 16.

5. The simulator also allows you to change the timing parameters to Phase-change Memories (PCM). To configure the timings:

   ```
   $ vim Controller.h
   ```

The following snapshot shows PCM timings are in use.

```
// DRAM Timings
// static unsigned nclks_read = 53;
// static unsigned nclks_write = 53;
// PCM Timings
static unsigned nclks_read = 57;
static unsigned nclks_write = 162;
```

6. Repeat experiments 2, 3, and 4 with PCM timing parameters.

7. What if there is an out-of-order memory scheduler that can prioritize request that targets to a free bank? Suppose there are three requests, R1 (target to Bank 0), R2 (target to Bank 0), and R3 (target to Bank 1) in the transaction queue; when R1 is issued, Bank 0 stays busy till R1's completion. In the next clock cycle, the memory controller is not able to schedule R2 because Bank 0 is busy; however, all other banks stay free, which means R3 can be scheduled out-of-order. We call such a scheduler as an FR-FCFS scheduler. Implement an FR-FCFS scheduler and repeat experiments 2 and 3. Does an FR-FCFS scheduler always out-perform an FCFS scheduler?

6 Submission

1. Pick at least five workloads for your experiments.
`https://www.dropbox.com/sh/o9l12v0fksbdt9e/AACTBAv67FLIjSV9Cma_0_9Ua?dl=0`.

2. Report your experiments, zip it with the source codes, and submit through Bblearn.