CS2610 - ASSIGNMENT 6

Internals of DRAM row-buffer management

Araj Khandelwal - CS20B008 | Lokesh Patil - CS20B047

PROBLEM STATEMENT

1. Implement a Hybrid-Row-Buffer Management Policy, known as Adaptive-Page Management Policy. The details of policy are explained below:

    a. Use a 4-bit saturation counter, and choose two threshold values, namely, High-Threshold and Low-Threshold, with a difference between them not more than 6 and not less than 4.

    b. Initially, set the counter to somewhere in the middle of the range between high and low thresholds and start with Open-Page policy.

    c. switch from Open-Page to Close-Page policy when the count is greater than High-Threshold and increasing, and from Close-Page to Open-Page policy when the count is less than Low-Threshold and decreasing.

    d. If currently in Open-Page policy, and a page-hit is observed, no action to be taken.
    e. If currently in Open-Page policy, and a page-miss occurs, increment the counter.

    f. If currently in Close-Page policy, and if a page-hit with the last closed page occurs, then decrement the counter.

    g. If currently in Close-Page policy, and a page-miss occurs, no action to be taken.

2. Compare the number of cycles (output) reported by USIMM for all the traces run with Open-Page, Close-Page, Adaptive-Page management policies. Report the best policy.

3. Compare the DRAM performance (in cycles) for the two address mapping schemes, provided in USIMM, for the best row buffer management policy determined in the previous part of the question.

MEMORY SYSTEM OVERVIEW

- **Memory Basics:** In a conventional memory system, a memory controller on the processor is connected to dual in-line memory modules (DIMMs) via an off-chip electrical memory channel.
  DiMM is a dual in-line memory module which consists of several ranks which is further divided into some no.of devices or banks. Each bank is subdivided into several sub-banks. Each sub-bank is basically a group of 2 dimensional arrays. The array consists of the DRAM cells in which data is stored in the form of voltage in capacitors.
- **Row Buffer Management:** The row-buffer is basically the sense-amplifier - the place where the data from the requested row is stored after the ACT command.
- **Address Mapping:** A cache line is placed entirely in one bank. The next cache line could be placed in the same row, or the next row in the same bank, or the next bank in the same rank, or in the next rank in the same channel, or in the next channel. The data mapping policy determines the extent of parallelism that can be leveraged within the memory system. The address bits are interpreted as follows, from left (MSB) to right (LSB): 1channel mapping policy:
  row : rank : bank : channel : column : blockoffset
  The second configuration (4channel, with ADDRESS MAPPING set to 0) tries to maximize memory access parallelism by scattering consecutive blocks across channels, ranks, and banks. The address bits are interpreted as follows: 4 channel mapping policy:
  row : column : rank : bank : channel : blockoffset
- **Memory Commands:** In every cycle, the memory controller can either issue a command that advances the execution of a pending read or write, or a command that manages the general DRAM state. The four commands corresponding to a pending read or write are:

• PRE: Precharge the bitlines of a bank so a new row can be read out.

• ACT: Activate a new row into the bank's row buffer.

• COL-RD: Bring a cache line from the row buffer back to the processor.

• COL-WR: Bring a cache line from the processor to the row buffer.

• Refresh: Forces a refresh to multiple rows in all banks on the rank. If a chip is in a power-down mode before the refresh interval, the rank is woken up by refresh.

- **Timing Parameters:** Only some of the above commands can be issued in a given cycle, depending on the current state of the ranks, banks, channel, and several timing parameters.
- **DRAM Refresh:** Every DRAM row must be refreshed within a 64 ms window while at a temperature under 85 degrees Celsius. The refresh process generally works as follows. The memory controller issues a Refresh command every 7.8μs (the tREF I parameter). This command initiates a refresh to multiple rows in all banks on the channel.

ROW-BUFFER MANAGEMENT POLICIES:

When access to a row is complete, the memory controller may issue a PRE to it, or it may leave it open for some amount of time. This behavior is determined by the page policy in use.

- **Open-Page row-buffer policy:** An open-page policy allows the memory controller to leave a page open after a read or write. When the next request comes, it is trying to access the same row, then it is a page hit, in which case only a CAS command is needed, otherwise we incur a page miss, in which case we need PRE, ACT and then a CAS command.
- **Closed-Page row-buffer policy:** A closed-page policy ensures that a DRAM page is closed immediately after every read or write. This effectively eliminates both page-misses and page-hits, making every access a page-empty

- **Adaptive-Page Management Policy:** In our adaptive-page policy, we shuffle between open page and closed page policy. We use a 4-bit counter, and choose two threshold values high_threshold and low_threshold, with a difference between them not more than 6 and not less than 4, set the counter to somewhere in the middle of the range between high and low thresholds and start with Open-Page policy. Now we change between open page policy and closed page policy according to the counter and its current derivative. If it's decreasing and is less than low_threshold then change to open-page policy and if it's increasing and is more than high_threshold then change to closed page-policy. Also, increment counter if we incur a page-miss in open-page policy and decrement the counter if we incur a page-hit in closed page policy.

USIMM Simulator:

The USIMM: the Utah SImulated Memory Module is a DRAM main memory system simulator that is being released for use in the Memory Scheduling Championship (MSC).

- ## Simulator Design:
    1. **Code files:** The code is organized into the following files:
        - **main.c** : Handles the main program loop that retires instructions, fetches new instructions from the input traces, and calls update memory( ). Also calls functions to print various statistics.
        - **memory controller.c** : Implements update memory( ), a function that checks DRAM timing parameters to determine which commands can issue in this cycle. Also has functions to calculate power.
        - **scheduler.c** : Function provided by the user to select a command for each channel in every memory cycle.
        - **configfile.h memory controller.h params.h processor.h scheduler.h utils.h utlist.h** : various header files.
    2. **Inputs:** The main( ) function in file main.c interprets the input arguments and initializes various data structures. Each subsequent argument represents an input trace file. Each such trace is assumed to run on its own processor core.

3. **Simulation Cycle:** The simulator then begins a long while loop that executes until all the input traces have been processed. Each iteration of the while loop represents a new processor cycle, possibly advancing the ROB.
4. **Scheduling:** Once a list of candidate memory commands for this cycle is determined by our scan, a schedule( ) function (in file schedule.c) is invoked. This is the heart of the simulator. In each memory cycle, each memory channel is capable of issuing one command. Out of the candidate memory commands, the schedule function must pick at most one command for each channel.
5. **Fetch Constraints and Write Drains:** We assume that non-memory (N) and memory write (W) instructions finish instantaneously, i.e., they are never bottlenecks in the commit process. Memory-writes will hold up the trace only when the write queue is full. To prevent this, it is the responsibility of the scheduler to periodically drain writes.
6. **Refresh Handling:** The simulator ensures that in every 8 × tREF I window, all DRAM chips on a channel are unavailable for time 8 × tRF C, corresponding to eight Refresh operations. If the user neglects to issue eight refreshes during the 8×tREF I time window, USIMM will forcibly issue any remaining refreshes at the end of the time window.

- ## Schedulers:

1. **FCFS, scheduler-fcfs.c:** Assuming that the read queue is ordered by request arrival time, our FCFS algorithm simply scans the read queue sequentially until it finds an instruction that can be issued in the current cycle. A separate write queue is maintained. When the write queue size exceeds a high water mark, writes are drained similarly until a low water mark is reached. The scheduler switches back to handling reads at that time. Writes are also drained if there are no pending reads.
2. **Close-Page, scheduler-close.c :** This policy is an approximation of a true close-page policy. In every idle cycle, the scheduler issues precharge operations to banks that last serviced a column read/write.
3. **scheduler-.c** : This is our code for the **Adaptive-page policy.**

PROCEDURE FOR RUNNING THE SIMULATOR:

Note: The main file runs in accordance with the scheduler.c file.

We run the following commands:

```
cd src/
make clean
make
```

This produces a usimm executable in the bin/ directory. To run the example simulation script,

```
cd ..
./runsim
```

The simulation should finish in tens of minutes. Use a truncated version of the trace files for shorter tests. To examine the simulation outputs,

```
view output/*
```

MEASUREMENTS:

1. **OPEN-PAGE POLICY:**

   We took the code from scheduler-fcfs.c and pasted it in scheduler.c and then ran the simulator according to the above commands.

   Below is the snippet of the whole output, which also has been included in the zip file(open-output.txt).

```
#--------------------------------------------------------------------------
Total Simulation Cycles                      281165956
--------------------------------------------------------------------------
```

2. **CLOSE-PAGE POLICY:**

We took the code from scheduler-close.c and pasted it in scheduler.c and then ran the simulator according to the above commands.

Below is the snippet of the whole output, which also has been included in the zip file(close-output.txt).

```
#--------------------------------------------------------------------------
Total Simulation Cycles                      263273636
--------------------------------------------------------------------------
```

3. **ADAPTIVE-PAGE POLICY:**

We took the code from scheduler-close.c and pasted it in scheduler.c and then ran the simulator according to the above commands.

Below is the snippet of the whole output, which also has been included in the zip file(close-output.txt).

OVERALL OBSERVATIONS-POLICY COMPARISONS:

	OPEN-PAGE POLICY	CLOSED-PAGE POLICY	ADAPTIVE-PAGE POLICY
DRAM Cycles	421758733	392935533	392935533
Energy delay product	2.137696981	1.881123066	1.881969810

	Consecutive cache-lines to same row	Consecutive cache-lines striped across different banks
DRAM Cycles	281165956	264173256
Total memory system power	12.721520 W	12.721520 W

CODE-OVERVIEW:

ADAPTIVE-PAGE POLICY:

```
int counter; // counter value is 0-15
int high_threshold;
int low_threshold;
int start;
struct draddr prev_row;
```

- We declare the above variables to keep track of which policy to be used at present. If the counter value is greater than high_threshold and its value is increasing then we switch to close-page policy from open-page policy. And if the counter value is less than low_threshold and its value is decreasing then we switch to open-page policy from close-page policy.
- When we are in open-page policy and we incur a page miss, we increment the counter and set the prev_row to the current row. Below is the snippet for write drain mode.

```
if(start == 0 && prev_row.channel==wr_ptr->dram_addr.channel &&
prev_row.rank==wr_ptr->dram_addr.rank && prev_row.bank==wr_ptr->dram_addr.bank &&
prev_row.row==wr_ptr->dram_addr.row)
{
  // do nothing
}
else
{
  if(start==0) {counter++;} // increment counter if page miss
  start = 0;
  prev_row.channel = wr_ptr->dram_addr.channel; // now make prev_row = current_row
```

```
  prev_row.rank = wr_ptr->dram_addr.rank;
  prev_row.bank = wr_ptr->dram_addr.bank;
  prev_row.row = wr_ptr->dram_addr.row;
}
```

Below is the snippet for read drain mode.

```
if(start == 0 && prev_row.channel==rd_ptr->dram_addr.channel &&
prev_row.rank==rd_ptr->dram_addr.rank && prev_row.bank==rd_ptr->dram_addr.bank &&
prev_row.row==rd_ptr->dram_addr.row)
{
    // do nothing
}
else
{
 if(start==0) {counter++;} // increment counter when page miss
 start = 0;
 prev_row.channel = rd_ptr->dram_addr.channel; // prev_row = current_row
 prev_row.rank = rd_ptr->dram_addr.rank;
 prev_row.bank = rd_ptr->dram_addr.bank;
 prev_row.row = rd_ptr->dram_addr.row;
}
```

- When we are in close-page policy and we incur a page hit, we decrement the counter and when we incur a page miss, we just set the prev_row to the current row. Below is the snippet for write drain mode.

```
if( prev_row.channel==wr_ptr->dram_addr.channel && prev_row.rank==wr_ptr->dram_addr.rank &&
prev_row.bank==wr_ptr->dram_addr.bank && prev_row.row==wr_ptr->dram_addr.row)
{
  counter--; // decrement row if page hit
}
else
{
 prev_row.channel = wr_ptr->dram_addr.channel; // prev_row = current_row
 prev_row.rank = wr_ptr->dram_addr.rank;
 prev_row.bank = wr_ptr->dram_addr.bank;
 prev_row.row = wr_ptr->dram_addr.row;
}
```

- Full code is in the scheduler.c file which is in the src directory.

CONCLUSION:

We can infer from the table that adaptive-page policy and close-page value have almost the same value for DRAM cycles, and these values are more than the same for the open-page policy.

Therefore, we infer that the best policy is the hybrid-page policy for this set of input.

Comparing the DRAM performance(in cycles) for the two address mapping schemes, provided in USIMM, we infer that different address mapping schemes have different performance and hence selecting the proper address mapping scheme is essential for good performance.

We infer that Consecutive cache-lines striped across different banks type of address mapping is better than Consecutive cache-lines to the same row.