Hybrid Row-buffer Management

CS6600: Computer Architecture

Arjun Menon V, Akilesh Kannan
EE18B104, EE18B122

The code used and developed for this assignment can be found here - link

1 DRAM Organisation

DRAM, Dyanmic Random Access Memory, often referred to as the Main memory (or RAM in consumer electronics), is a collection of arrays of memory cells. Each memory cell consists of a transistor and a capacitor, and the voltage across it interpreted as 1s and 0s. Due to the high impact of the main memory access time on performance of the system, an efficient organisation is needed in order to exploit the highest level of concurrency and locality, improving performance.

From the point of view of the memory controller, the DRAM is organised into Channels, Ranks, Banks, Rows and Columns. Figures 1 and 2 illustrate the hierarchy in a visual manner.

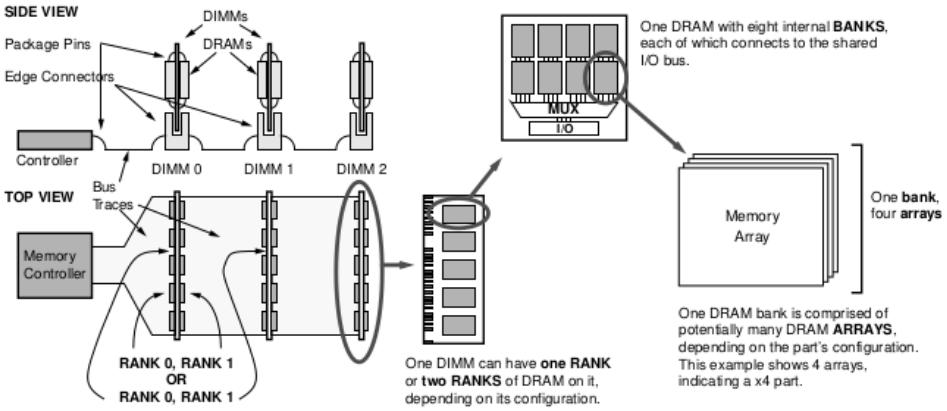

Fig. 1: DRAM Organisation [2]

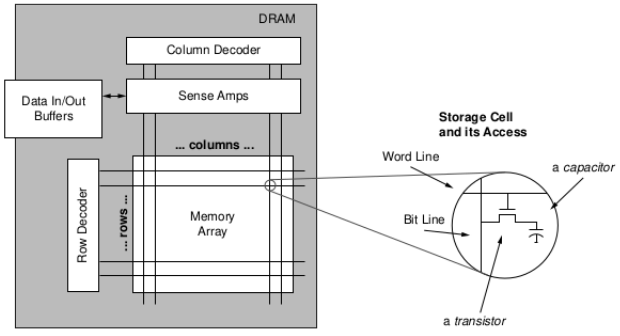

Fig. 2: DRAM Bank [2]

The sense amplifiers are called as *row buffers*. A single row from the memory array is placed into the row buffer using *PRE*charge (to make sure that the bank is in a voltage state that is suitable for activation), followed by an *ACT*ivate (to place the required row into the row buffer).

Due to the various non-idealities present in the transistor, memory cells always leak charge and must be *REF*reshed every now and then.

The job of the memory controller is to convert memory requests generated by the processor, into commands understandable by the DRAM, *while enforcing necessary timing constraints*. This makes the job of the memory controller very complex and it has many aspects to it. The memory controller performs Request Scheduling (to manage requests from the processor) and Command Scheduling (to manage commands to the DRAM module) for Read and Write operations; in addition, the memory controller also maps the memory address to a DRAM-understandable address, which corresponds to a row in a particular (channel, rank, bank).

2 Row Buffer Management Policies

The state of the row buffer is managed by a *Row-Buffer Management Policy* (RBMP). This is a crucial aspect of the memory controller, as it dictates the latency involoved in a memory access.

There are 2 main types of RBMPs - *Open page* and *Close page*.

2.1 Open Page RBMP

In the open page policy, once a row of data is brought to the row buffer different columns of the same row can be accessed again with the minimal latency of t_{CAS}. When another memory access is made to the same row, that memory access can occur with minimal latency since the row is already active in the row buffer and only a column access command is needed to move the data to the memory controller. However, when the access is to a different row of the same bank, the memory controller must first *PRE*charge the DRAM array, engage another row *ACT*ivation, and then perform the column access (*CAS*).

This policy exploits the spatial locality of the row buffer and thus improves performance significantly in applications where data access exhibits a high degree of spatial locality.

2.2 Close Page RBMP

In this policy, a row buffer is closed as soon as it is accessed, and a *PRE*charge is done on the the bank immediately after, making the bank ready for the next memory access. Thus, all memory accesses take the same amount of latency - $t_{CAS} + t_{RCD}$.

This policy doesn't exploit any locality and is favorable in workloads where the memory accesses do not show any spatial locality.

2.3 Hybrid RBMPs

Usually, due to the above policies being in either extremes of the spectrum, modern RBMPs are a mixture of open and close page policies, usually involving a scheme that looks at the "locality exploitation of recent accesses", and accordingly switches between open and close page policies.

The policy implemented as part of this assignment is as follows:

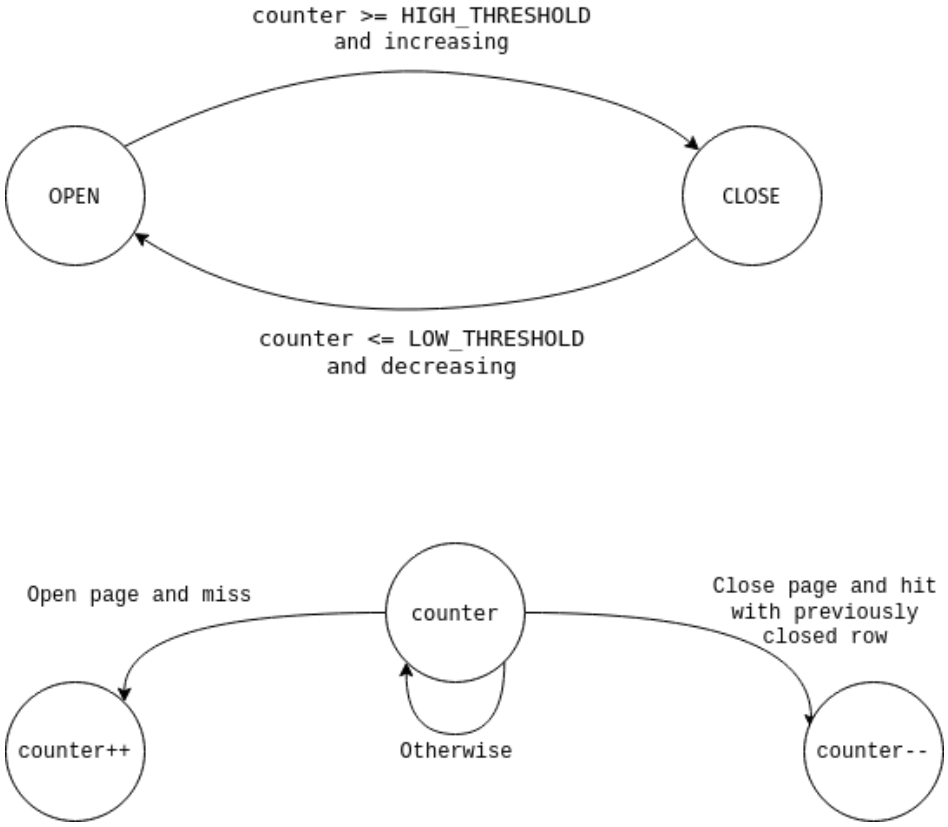

Fig. 3: Adaptive Page Management Policy

3 Code Organisation

We have compared the three Row Buffer Management Policies- Open-Page, Closed-Page and Adaptive- by evaluating the simulation cycles required to run benchmark traces for each RBM Policy. The simulations have been performed using the USIMM simulator [1, 3] which models the memory controller for a given configuration. All our simulations were run for a quad-core device with a 4-channel 4Gb x8 DRAM module. The code organisation is as follows:

- **main.c:** Top level function that calls the memory controller, scheduling algorithm (implemented in the function schedule()) and maintains record of timing and power metrics of the simulation.
- **memory_controller.c:** Implements memory request queues, memory command queues and address mapping, subject to DRAM timing constraints. The memory controller defines the issue_request_command() and issue_precharge_command() functions which are used by the schedule() function to implement the RBM policies.
- **scheduler-fcfs.c:** Implements the schedule() function with an open-page policy
- **scheduler-close.c:** Implements the schedule() function with a closed-page policy
- **scheduler_adaptive.c:** Defines the schedule() function and implements the adaptive page policy mentioned above

3.1 Address mapping schemes supported by USIMM:

- **Address Mapping 0:** *Physical Address* = {*row*:*column*:*rank*:*bank*:*channel*:*block_offset*}
  This address mapping scheme maximises memory access parallelism by placing adjacent physical addresses in different DRAM channels thereby enabling multiple addresses to be accessed in the same cycle.

- **Address Mapping 1:** *Physical Address* = {*row*:*rank*:*bank*:*channel*:*column*:*block_offset*}
  In address mapping one, blocks stored in nearby columns of a row have adjacent physical addresses. This mapping scheme when coupled with an Open Page Policy can exploit the spatial locality of memory accesses in applications; however, this comes at the cost of reduced parallelism since adjacent blocks are mapped to the same DRAM channel.

4 Results

Trace	Config	Simulation Cycles			Best Policy
		Open	Close	Adaptive	
black	4 channel	218317808	208520860	209908112	Close
comm1	4 channel	288462212	266310468	269285720	Close
comm2	4 channel	371416917	343988713	344187833	Close
face	4 channel	264310784	245820700	247326552	Close
ferret	4 channel	278517237	260116661	261466901	Close
fluid	4 channel	306021149	290215225	290959741	Close
freq	4 channel	194471670	186601801	186187537	Adaptive
stream	4 channel	217414784	205389388	206890212	Close
swapt	4 channel	322854277	306640281	308310365	Close

Tab. 1: Results for 4 channel, 4Gbx8 device configuration, Address Map 0

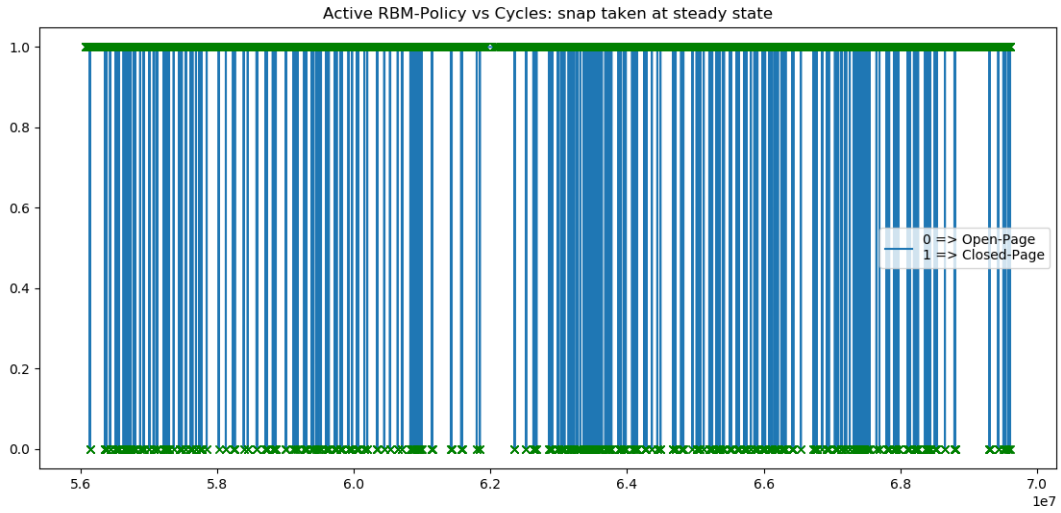

Fig. 4: Policy changing during run time

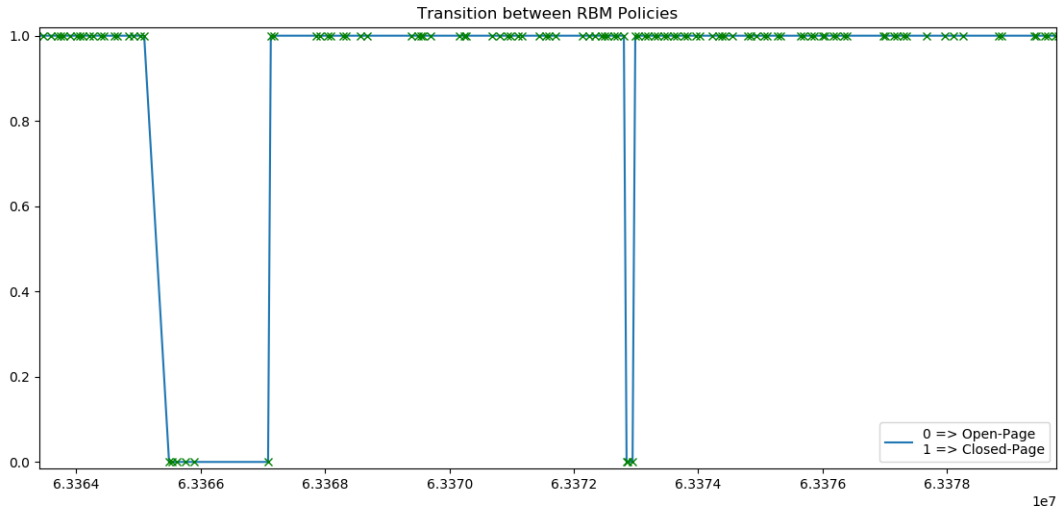

Fig. 5: Transition between policies: Hysteresis induced by the FSM counter in switching between policies can be seen

5 Inferences and Conclusions

From the above table and plots, it is evident that the Closed-Page Policy is the most efficient Row Buffer Management Policy when address mapping scheme 0 is used. This is further validated by the adaptive policy preferring the closed page policy in most cycles. Also, the difference in simulation cycles for closed- and adaptive-page policies is negligible, with the deviation amounting to $\leq 1.1\%$.

We attribute the superior performance of the Closed-Page Policy to the address mapping scheme. Scheme 0 places blocks with adjacent physical addresses across different channels enabling parallel access. At the same time, blocks placed in the same row in a bank showcase lower spatial locality (wrt physical address), reducing the perks of keeping the row buffer opened after an access. As mentioned above, in such a situation, the Open-Page Policy would incur an additional latency equal to t_{PRE}.

Trace	Config	Simulation Cycles (for best policy)	
		Address Mapping 0	Address Mapping 1
black	4 channel	208520860	211775680
comm1	4 channel	266310468	258677828
comm2	4 channel	343988713	339334952
face	4 channel	245820700	311752340
ferret	4 channel	260116661	294673705
fluid	4 channel	290215225	304425669
freq*	4 channel	186601801	257516784
stream	4 channel	205389388	215858996
swapt	4 channel	306640281	320290200

Tab. 2: Variation of simulation cycles for different address mapping schemes

Address Mapping Scheme	# Open-Page Accesses to Bank	Total Accesses to Bank	% Open
0	26329	440263	5.980
1	80306	290419	27.652

Tab. 3: Comparison of % of Open-Page Memory Accesses to (Channel, Rank, Bank) = (0, 0, 0) for the two Address Mapping Schemes when running `freq` trace on Adaptive Policy

References

[1] Chatterjee, N., Balasubramonian, R., Shevgoor, M., Pugsley, S. H., Udipi, A. N., Shafiee, A., Sudan, K., Awasthi, M., and Chishti, Z. USIMM: the Utah SImulated Memory Module: A Simulation Infrastructure for the JWAC Memory Scheduling Championship, 2012.

[2] Jacob, B., Ng, S., and Wang, D. T. *Memory Systems: Cache, DRAM, Disk*, 2 ed. Morgan Kaufmann Publishers, 2010.

[3] Utah Arch research group. USIMM Simulator.