

A Network Congestion-Aware Memory Controller

Dongki Kim, Sungjoo Yoo and Sunggu Lee

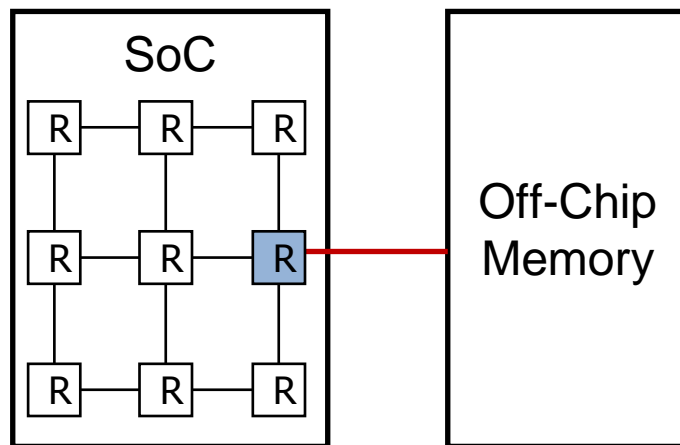
Embedded System Architecture Lab.
POSTECH

Agenda

- Background
- Motivational Example
- Our Idea
- Experimental Setup
- Results
- Summary

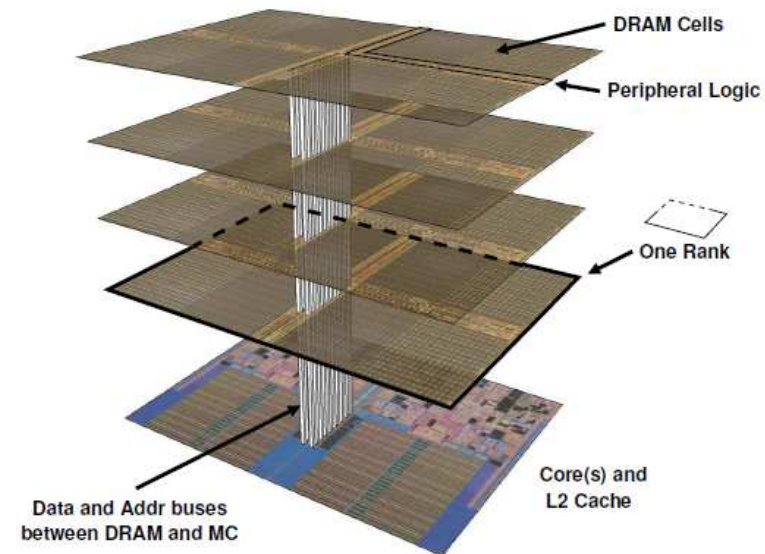
Background

- Conventional memory



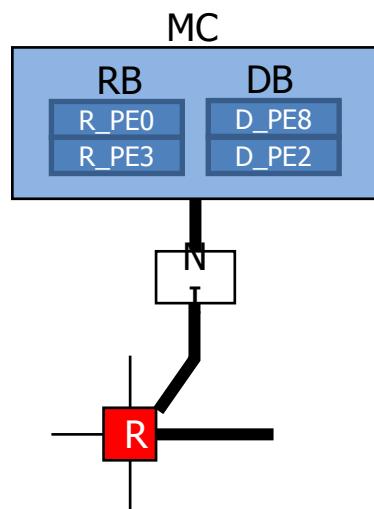
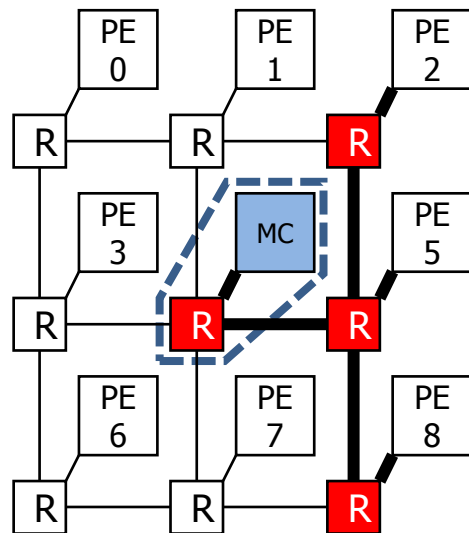
- Narrow memory I/O
- The router connected to memory is located at the side of SoC

- 3D stacked memory



- Wide memory I/O
- The router connected to memory can be located in the center of SoC

Motivational Example



- **Network congestion-induced memory blocking**
- **Scenario**

1. PE2 and PE8 are in the congested area → D_PE2 and D_PE8 in the data buffer cannot be sent to the router.
2. Data buffer is full → R_PE0 and R_PE3 in the request buffer cannot be processed.
3. Request buffer is full → MC cannot receive any more requests.

→ **Memory performance loss**

→ **System performance degradation**

Congested area: Shaded routers and thick links

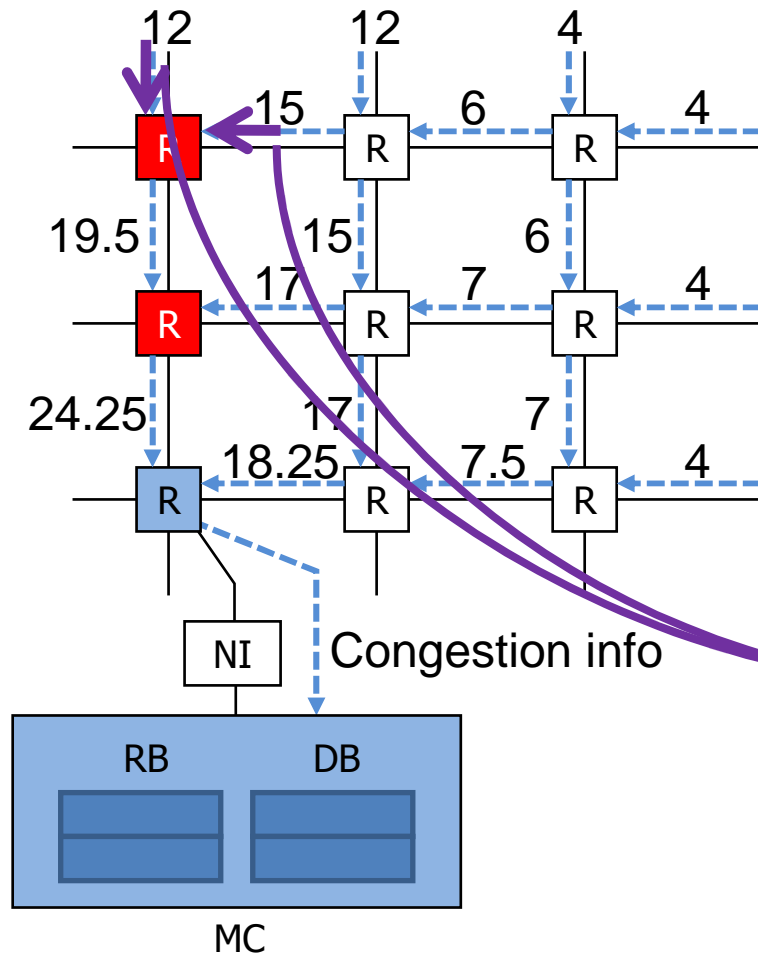
R_PEx: Read request from PEx

D_PEx: Read data to PEx

RB: Request buffer

DB: Data buffer

Preliminary: Non-Local Congestion Information



- **Congestion Information based on Regional Congestion Awareness (RCA) [Gratz, 2008]**

local congestion info =

of used vc + # of used xb

* vc : virtual channel

* xb : cross-bar

non-local congestion info_{out} =

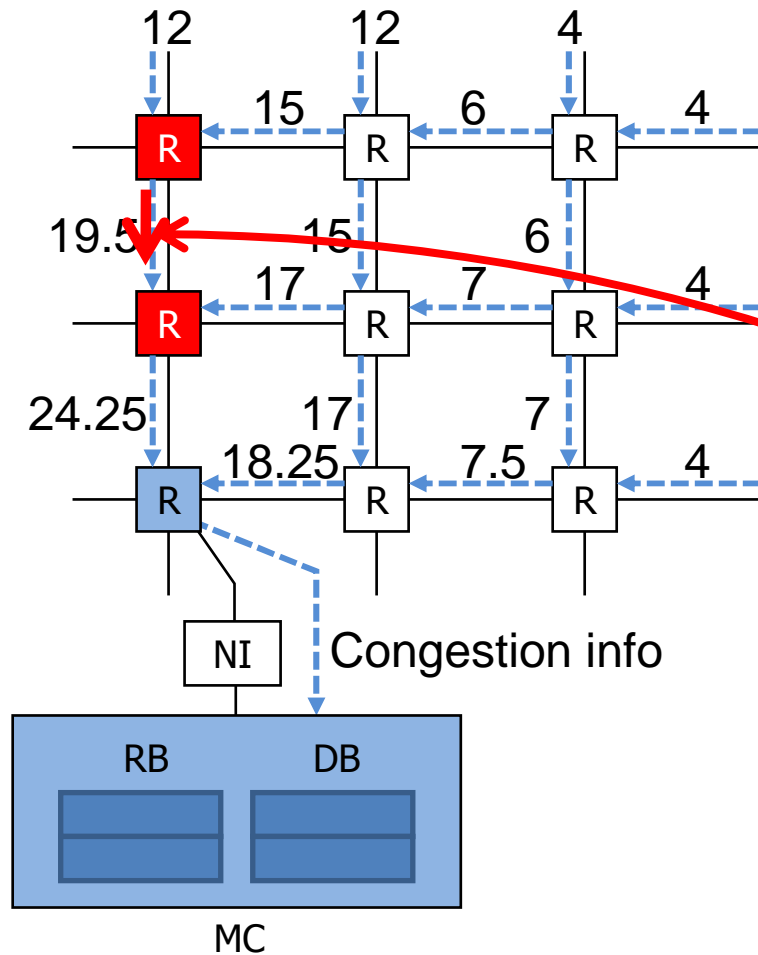
(a × non-local congestion info_{in}) +

(b × local congestion info)

* a, b : weight constant

We use non-local congestion information for scheduling in the MC

Preliminary: Non-Local Congestion Information



- **Congestion Information based on Regional Congestion Awareness (RCA) [Gratz, 2008]**

local congestion info =
of used vc + # of used xb

* vc : virtual channel

* xb : cross-bar

non-local congestion info_{out} =

(a x non-local congestion info_{in}) +
(b x local congestion info)

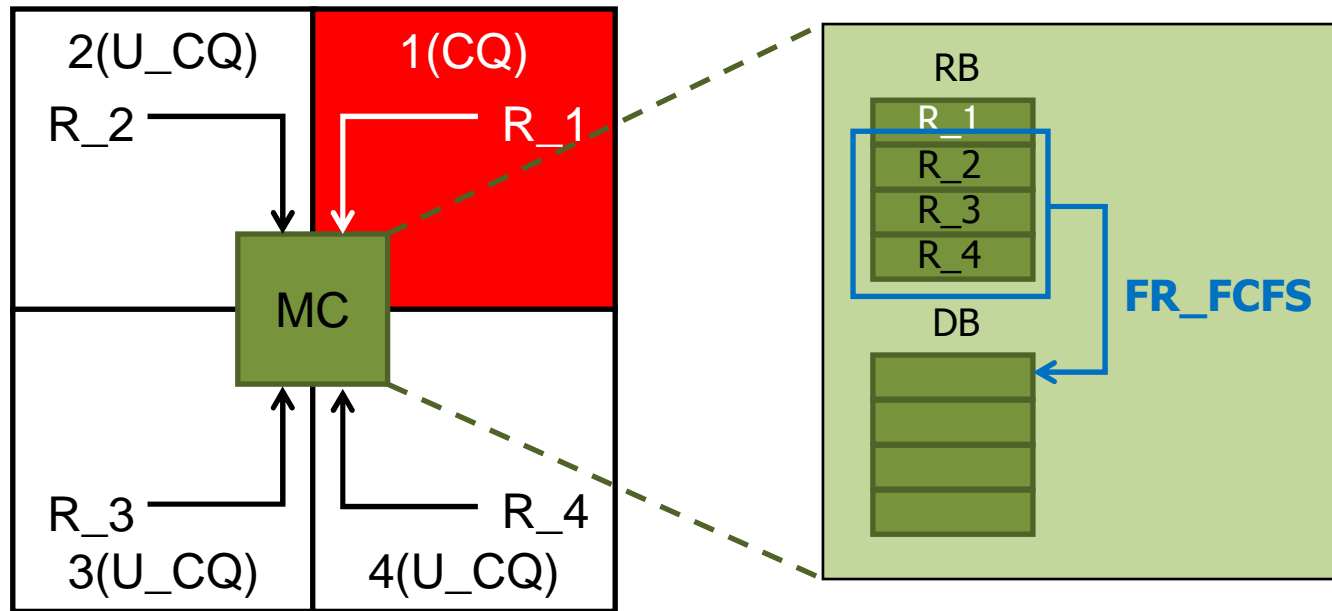
* a, b : weight constant

We use non-local congestion information for scheduling in the MC

Our Idea #1

1. Congestion-aware memory access scheduling

We can increase the memory utilization by prioritizing the requests from uncongested area than those from congested area.

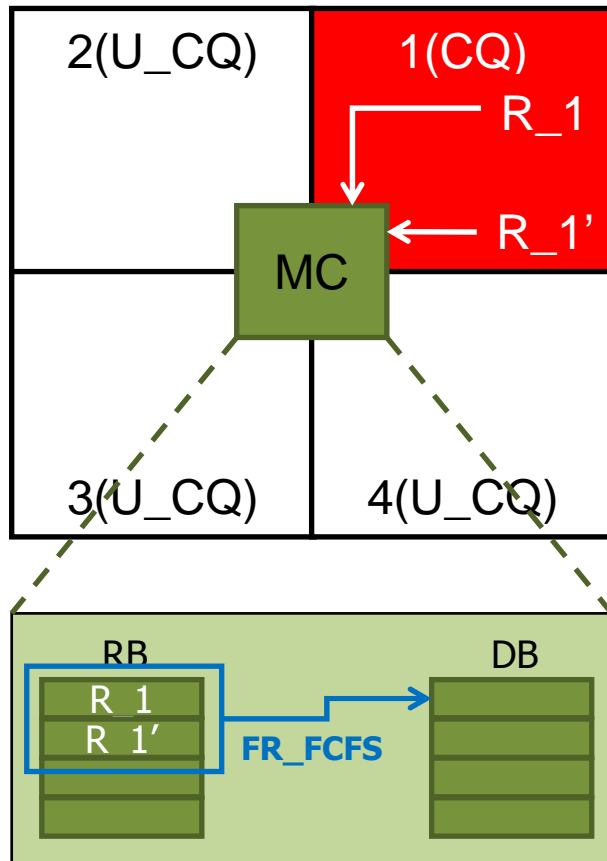


Case 1: Quadrant 1 is congested.

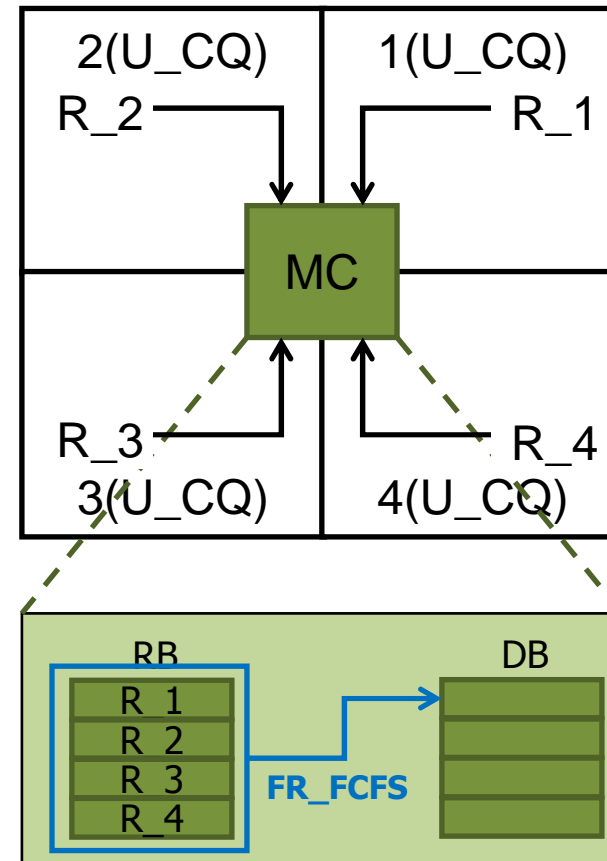
R_1, R_2, R_3 and R_4 arrive at the MC sequentially.

Our Idea #1

1. Congestion-aware memory access scheduling



Case 2: In the request buffer, there are only requests from CQ

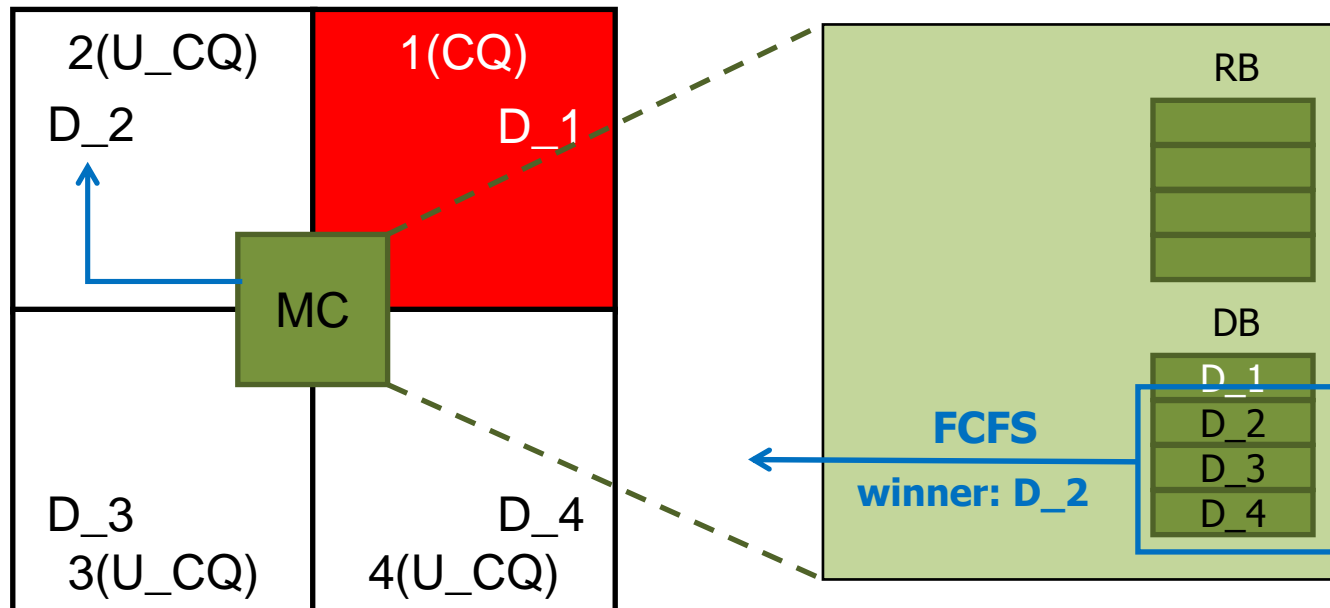


Case 3: In the request buffer, there are only requests from U_CQ

Our Idea #2

2. Congestion-aware network entry control at MC

Like the congestion-aware memory access scheduling, we prioritize read data which will be sent to the uncongested area. Therefore, we can increase data issue rate.

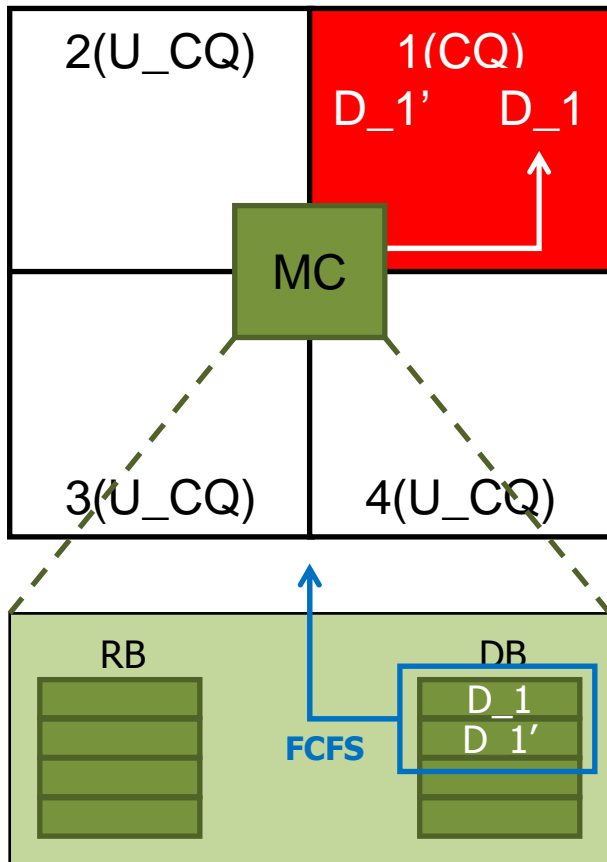


Case 1: Quadrant 1 is congested.

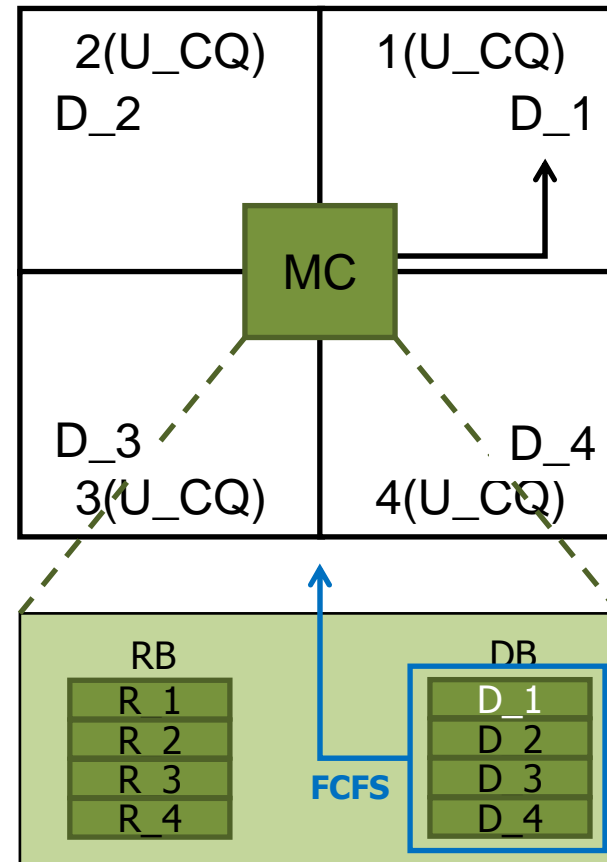
D_1, D_2, D_3 and D_4 arrive at the DB sequentially.

Our Idea #2

2. Congestion-aware network entry control at MC



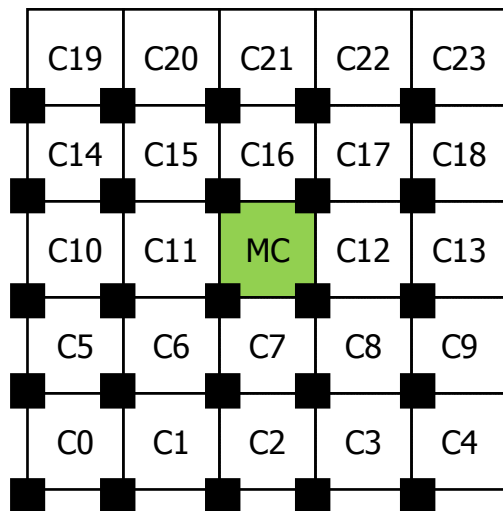
Case 2: In the data buffer, there is only data bound for CQ



Case 3: In the data buffer, there is only data bound for U_CQ

Experimental Setup (1)

- 5x5 tile architecture (cycle-accurate model in SystemC)



Cores: ■ Tensilica LX2

- Data width: 64bit
- Instruction/Data cache size: 16KByte
- Cache line size: 64Byte
- Frequency: 400MHz

Software programs: ■ SPEC2000

- vortex, ammp, mcf, art

Routers: ■ Four-stage pipeline

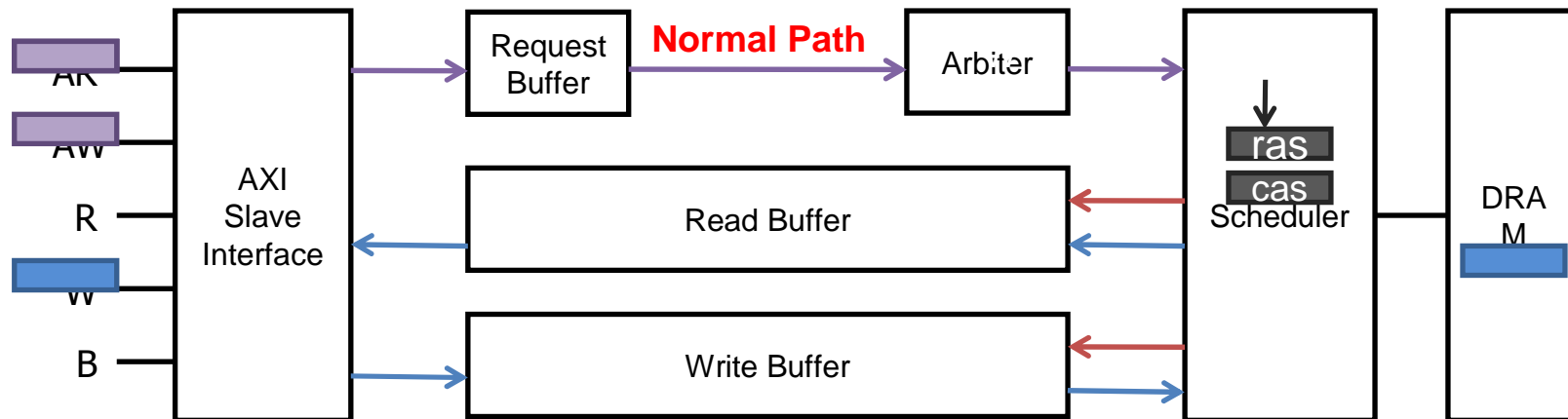
- 1st stage – input buffering
- 2nd stage – virtual channel allocation & look-ahead routing computation
- 3rd stage – switch allocation
- 4th stage – switch traversal
- 4 virtual channels per input port
- wormhole switching
- speculative allocation
- XY routing

Memory Controller: will be explained later

Experimental Setup (1)

- Memory Controller (w/ AXI interface)

- Request (Transaction ID, Address, Burst Length, Burst Size, Burst Type, Read/Write)
- Transaction ID
- Data

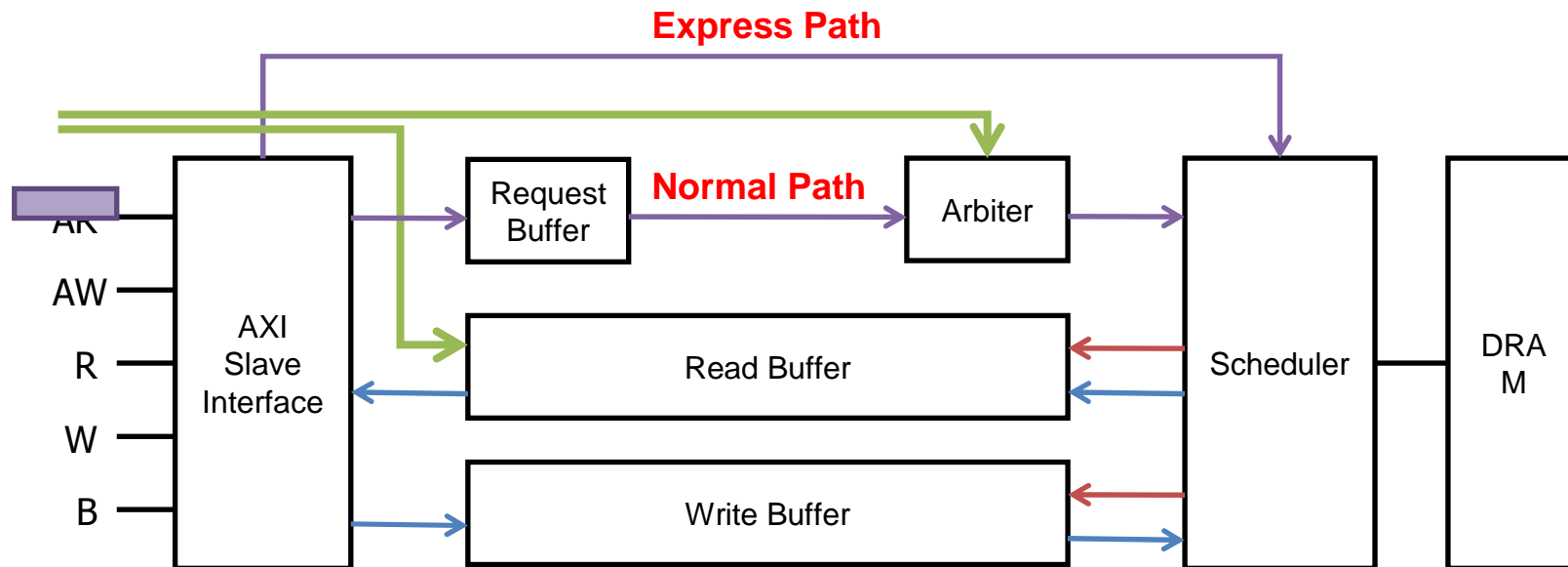


Normal MC latency (from AXI request to first DRAM command) = 3 cycle

Experimental Setup (1)

- Memory Controller (w/ AXI interface)

- Request (Transaction ID, Address, Burst Length, Burst Size, Burst Type, Read/Write)
- Transaction ID
- Data
- Congestion Information



Normal MC latency (from AXI request to first DRAM command) = 3 cycle

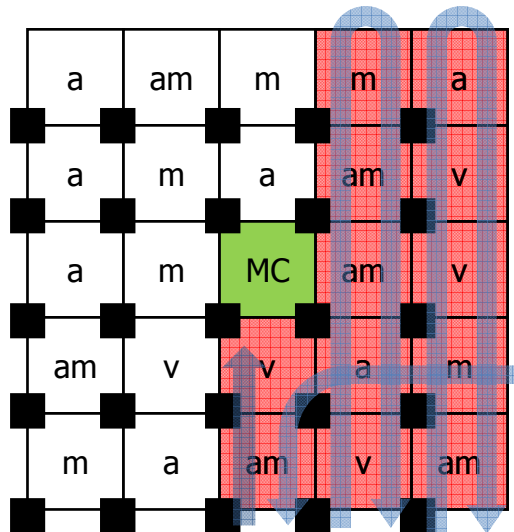
Express Path: If there is no request in the request buffer, arbiter and scheduler, the request skips request buffer and arbiter

Express MC latency = 1 cycle

Experimental Setup (2)

Network Congestion Cases

- 5x5 tile architecture



am : ammp
a : art
m : mcf
v : vortex

- **Experiment #1: A Synthetic Case**

1. Overlaying synthetic pattern

→ Arrows

issue rate = 1packet / 9cycles

= 1flit / 1cycle

2. Adjusting flit acceptance rate

→ Shaded Area

acceptance rate =

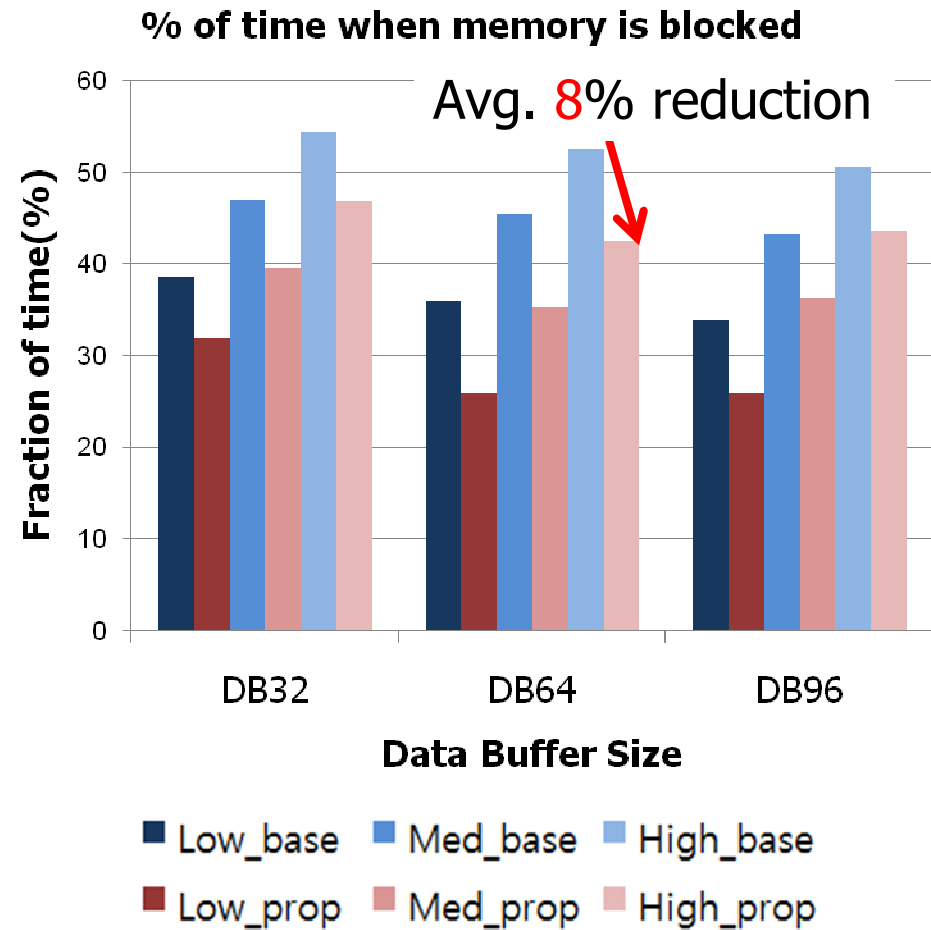
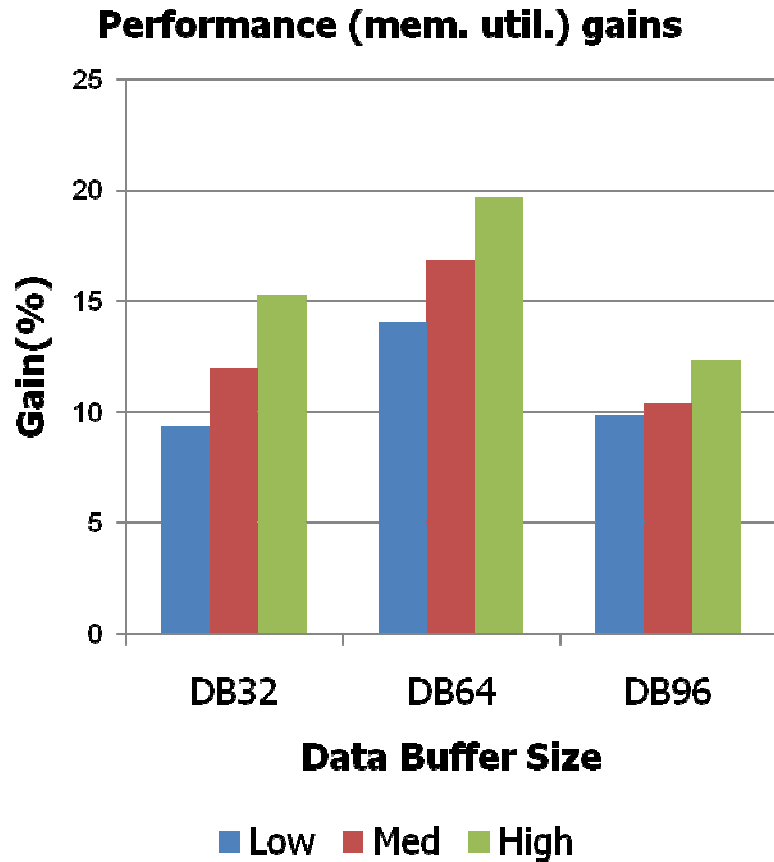
1flit / 6cycles (low)

1flit / 8cycles (medium)

1flit / 10cycles (high)

Results (1)

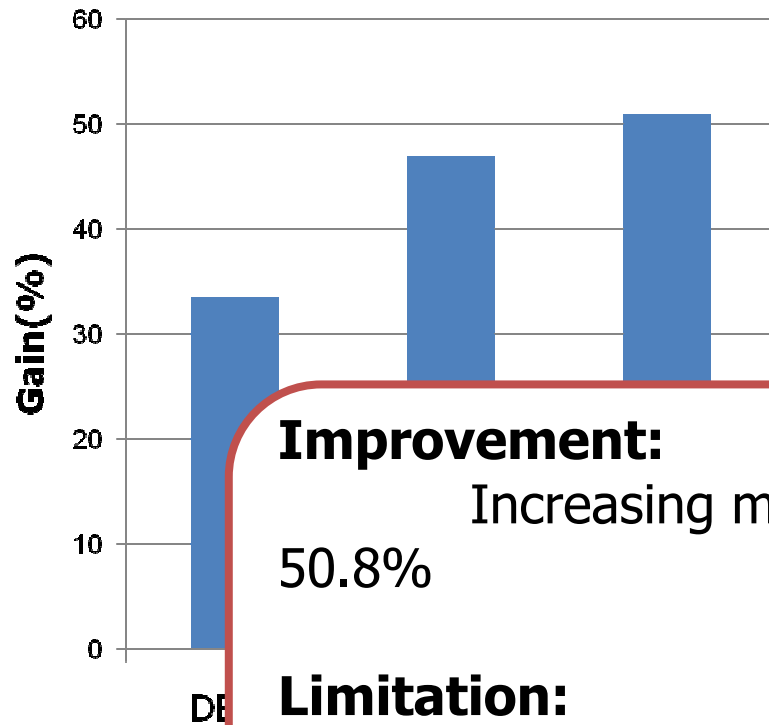
- Synthetic Case



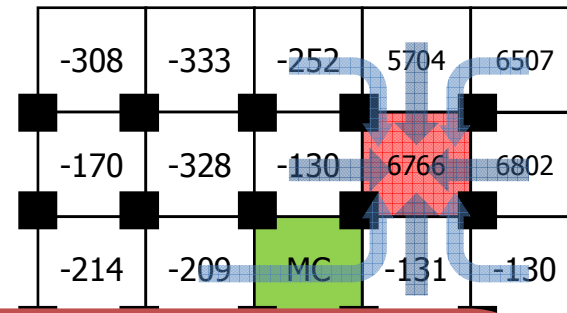
Results (2)

- Hot Spot Case

Performance (mem. util.) gains



Average latency difference per request



Improvement:

Increasing memory utilization by up to 50.8%

Limitation:

Increasing average memory access latency and fairness issue
 → On-going work

(: cycle)

Summary and On-going Work

- Problem
 - Network congestion-induced memory blocking
- Our solution
 - Prioritizing requests (data) from (to) uncongested area
 - Memory access scheduling
 - Network entry control
- Experimental results
 - The proposed memory controller presents up to 50.8% improvement in memory utilization with 5x5 tile architecture
- On-going work
 - Supporting fairness and QoS, especially, for latency
 - Congestion-aware network interface
 - Congestion-aware QoS
 - Mutual awareness between memory and network in the case of wide I/O with multiple memory channels

Thank you

Results (3)

- Memory Access Scheduling vs. Network Entry Control

