Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek[1,2] **Onur Mutlu**[2] **José F. Martínez**[1] **Rich Caruana**[1]

[1]Cornell University, Ithaca, NY 14850 USA
[2] Microsoft Research, Redmond, WA 98052 USA

35th International Symposium on Computer Architecture (**ISCA 2008**), June 21-25, 2008, Beijing, China.

Executive Summary

Executive Summary

Problem: Scheduling policies

- Can not anticipate the long term effects of their scheduling decisions

Executive Summary

Problem: Scheduling policies

- Cannot anticipate the long term effects of their scheduling decisions
- Cannot take lessons from the consequences of their past actions

Executive Summary

Problem: Scheduling policies

- Cannot anticipate the long term effects of its scheduling decisions
- Cannot take lessons from the consequences of its past actions

Solution: RL-based controller computes learning, far-sighted policy $\rightarrow$ efficient bandwidth utilization

Executive Summary

Problem: Scheduling policies

- Cannot anticipate the long term effects of its scheduling decisions
- Cannot take lessons from the consequences of its past actions

Solution: RL-based controller computes learning, far-sighted policy $\rightarrow$ efficient bandwidth utilization

Results:

- 19% speedup, 21% more bandwidth utilization over best static policy

Executive Summary

Problem: Scheduling policies

- Cannot anticipate the long term effects of its scheduling decisions
- Cannot take lessons from the consequences of its past actions

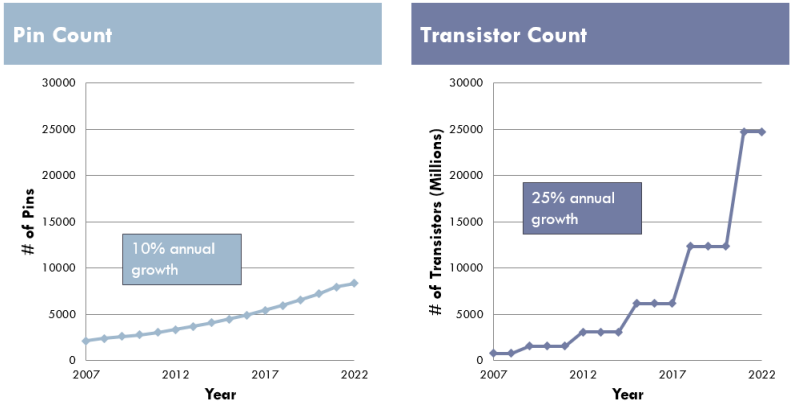
Solution: RL-based controller computes learning, far-sighted policy $\rightarrow$ efficient bandwidth utilization

Results:

- 19% speedup, 21% more bandwidth utilization over best static policy
- Scales as well as the best static policy

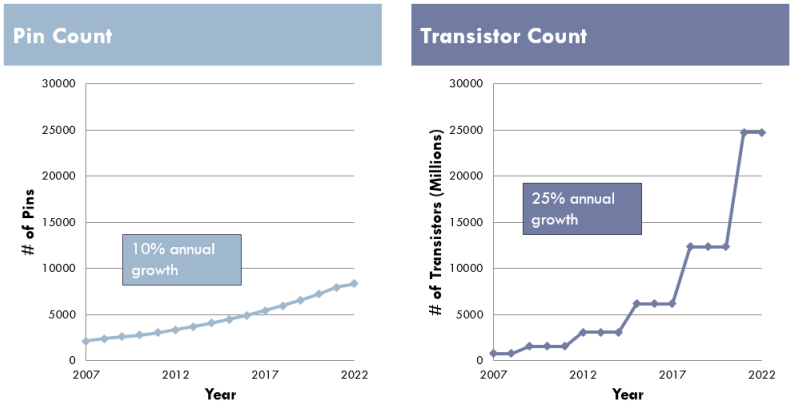
Problem, Background and Goal

DRAM bandwidth is the bottleneck



ITRS 2007 Executive Summary

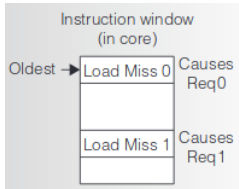
Goal: Efficiently utilize DRAM bandwidth



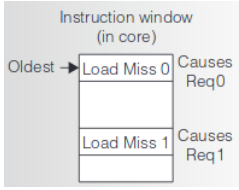
ITRS 2007 Executive Summary

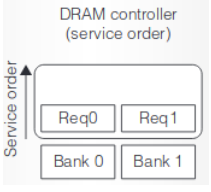
The Memory Controller

The Memory Controller

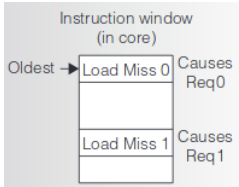


The Memory Controller

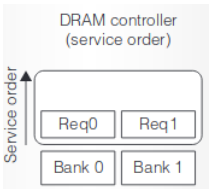




The Memory Controller

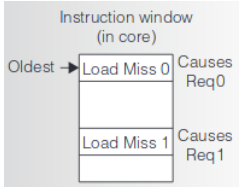


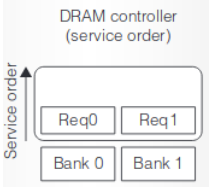
- Accepts cache misses and write-back requests, puts them in the memory transaction queue



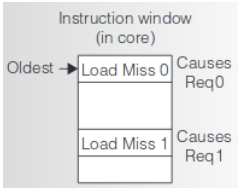
The Memory Controller

- Accepts cache misses and write-back requests, puts them in the memory transaction queue
- Issues *activate, read, write* and *precharge* commands to satisfy these requests

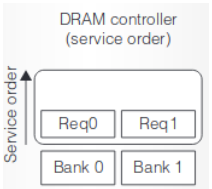




The Memory Controller



- Accepts cache misses and write-back requests, puts them in the memory transaction queue
- Issues *activate, read, write* and *precharge* commands to satisfy these requests
- Must obey many local and global DRAM timing constraints



[0]Onur Mutlu and Thomas Moscibroda, "Parallelism-Aware Batch Scheduling: Enabling High-Performance and Fair Memory Controllers" IEEE Micro, 2009

FR-FCFS Scheduling Policy

FR-FCFS Scheduling Policy

- Provides the best average performance

FR-FCFS Scheduling Policy

- Provides the best average performance
- (1) Prioritizes read/write over activate/precharge.

FR-FCFS Scheduling Policy

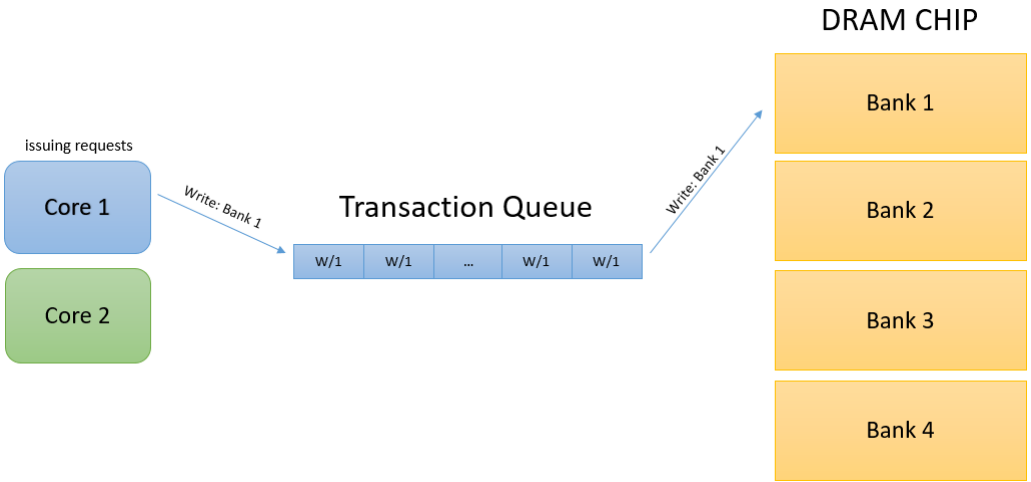
- Provides the best average performance
- (1) Prioritizes read/write over activate/precharge.
- (2) Prioritizes older commands over younger commands

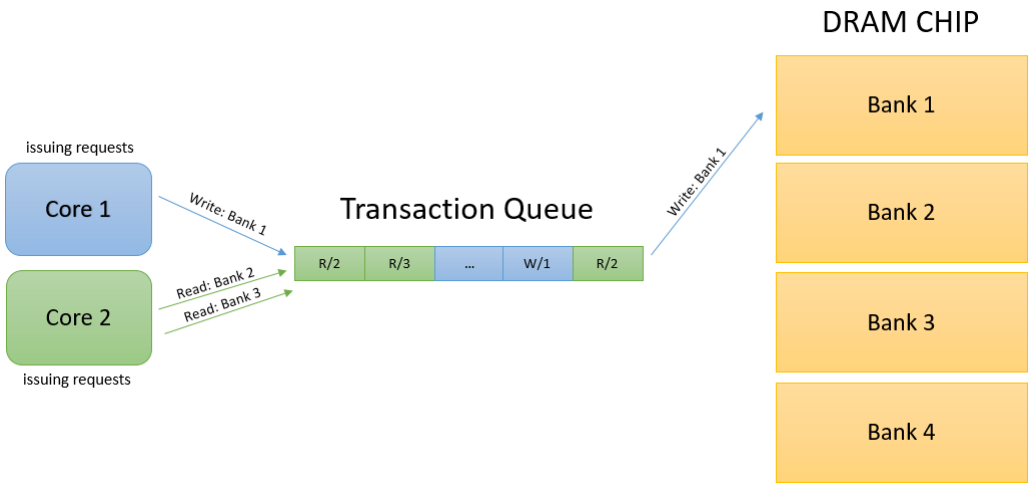
FR-FCFS Scheduling Policy

- Provides the best average performance
- (1) Prioritizes read/write over activate/precharge.
- (2) Prioritizes older commands over younger commands
- **Can't anticipate the long term effects of its scheduling decisions**

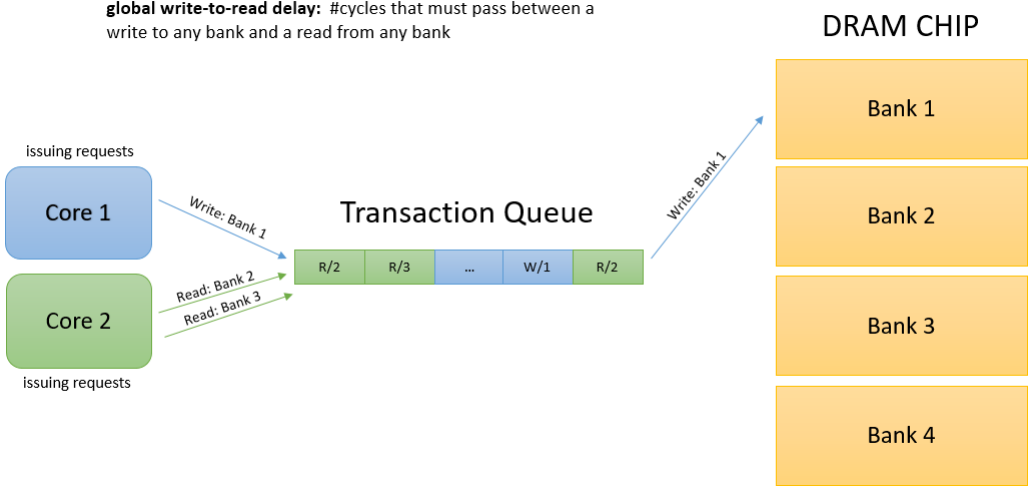
FR-FCFS Scheduling Policy

- Provides the best average performance
- (1) Prioritizes read/write over activate/precharge.
- (2) Prioritizes older commands over younger commands
- **Can't anticipate the long term effects of its scheduling decisions**
- **Can't take lessons from the consequences of its past actions to decide better in the future**

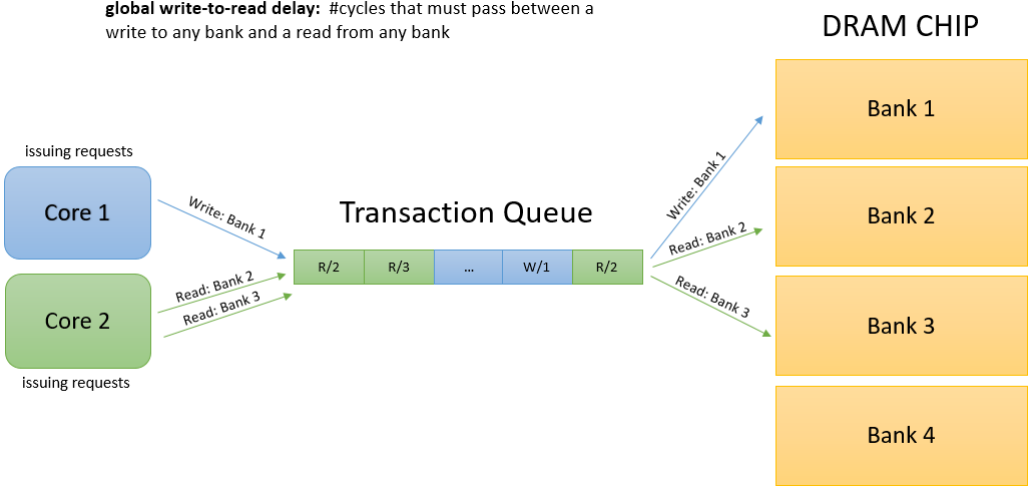

DRAM CHIP
Bank 1
Bank 2
Bank 3
Bank 4
issuing requests
Core 1
Core 2
Write: Bank 1
Transaction Queue
W/1
W/1
...
W/1
W/1
Write: Bank 1


DRAM CHIP
Bank 1
Bank 2
Bank 3
Bank 4
issuing requests
Core 1
Core 2
issuing requests
Write: Bank 1
Read: Bank 2
Read: Bank 3
Transaction Queue
R/2
R/3
...
W/1
R/2
Write: Bank 1

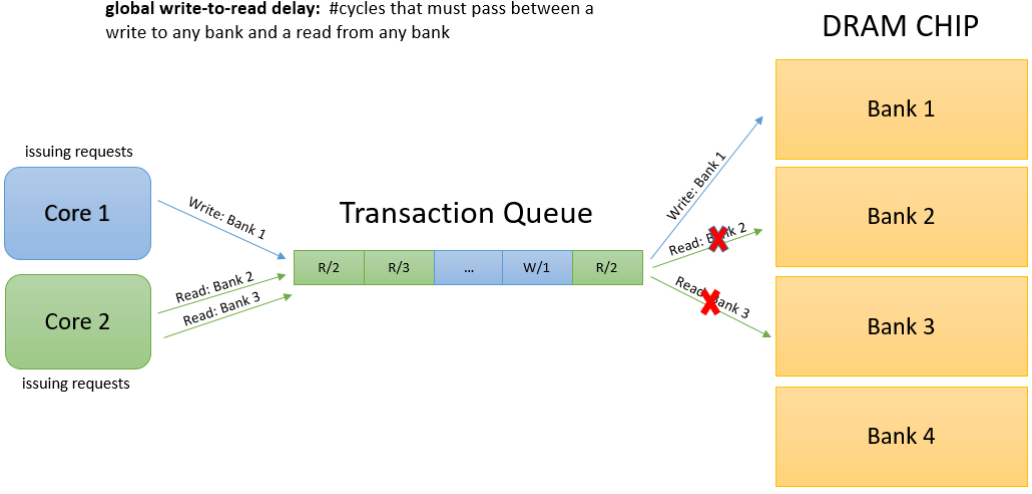
global write-to-read delay: #cycles that must pass between a write to any bank and a read from any bank



global write-to-read delay: #cycles that must pass between a write to any bank and a read from any bank

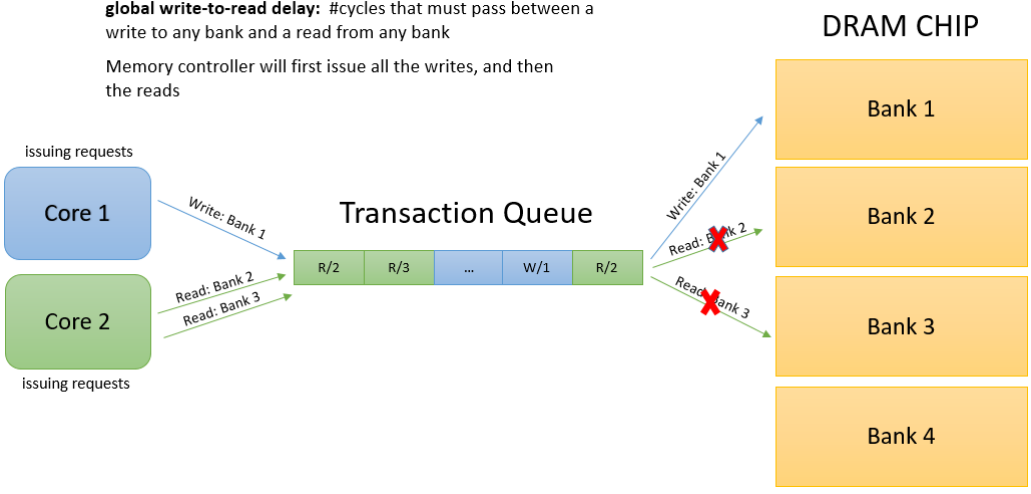


global write-to-read delay: #cycles that must pass between a write to any bank and a read from any bank



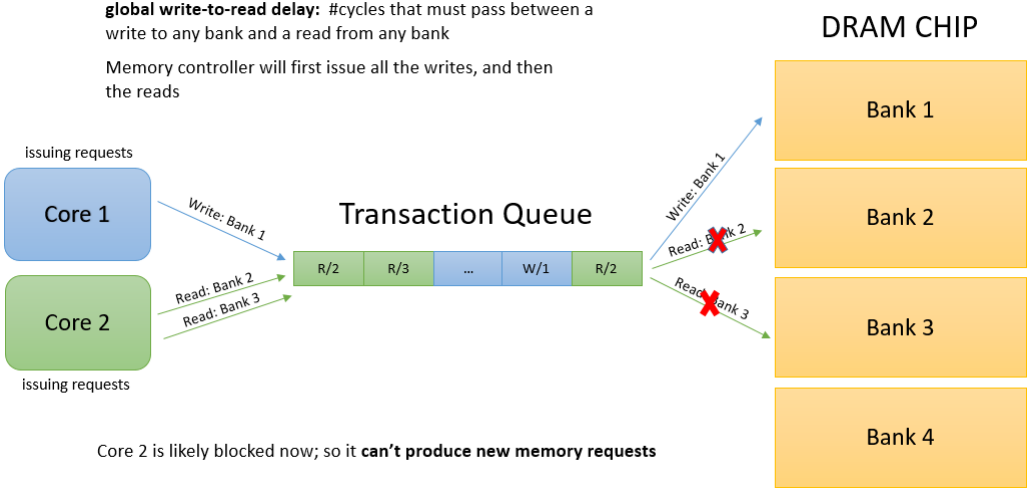
global write-to-read delay: #cycles that must pass between a write to any bank and a read from any bank

Memory controller will first issue all the writes, and then the reads



global write-to-read delay: #cycles that must pass between a write to any bank and a read from any bank

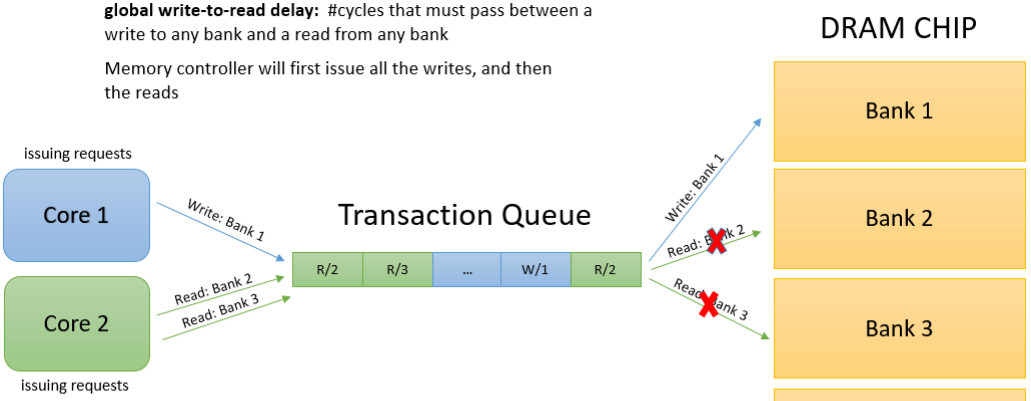
Memory controller will first issue all the writes, and then the reads



Core 2 is likely blocked now; so it **can't produce new memory requests**

global write-to-read delay: #cycles that must pass between a write to any bank and a read from any bank

Memory controller will first issue all the writes, and then the reads

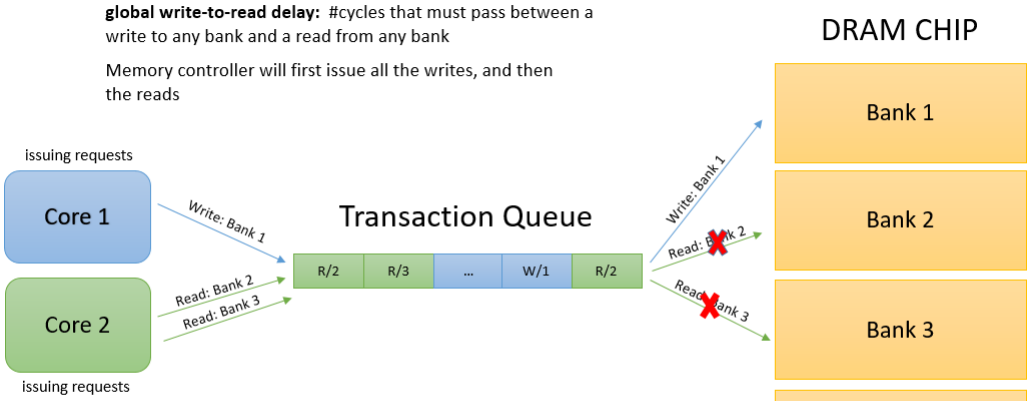


Core 2 is likely blocked now; so it **can't produce new memory requests**

DRAM bandwidth could have been better utilized

global write-to-read delay: #cycles that must pass between a write to any bank and a read from any bank

Memory controller will first issue all the writes, and then the reads



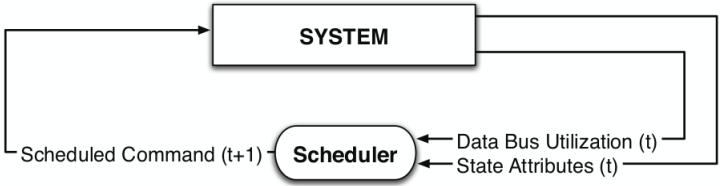
Core 2 is likely blocked now; so it **can't produce new memory requests**

DRAM bandwidth could have been better utilized

A static policy **will repeat its mistake** in every similar situation

Novelty, Key Approach & Ideas

Key Idea: Enable the Memory Controller to **Learn** From Its Actions



Benefits

Benefits

- Allows the hardware designer to focus on what specific goal the MC should accomplish

Benefits

- Allows the hardware designer to focus on what specific goal the MC should accomplish
- Enables the MC to adapt to workload changes

Benefits

- Allows the hardware designer to focus on what specific goal the MC should accomplish
- Enables the MC to adapt to workload changes
- Hopefully enables the MC to make farsighted decisions

Benefits

- Allows the hardware designer to focus on what specific goal the MC should accomplish
- Enables the MC to adapt to workload changes
- Hopefully enables the MC to make farsighted decisions
- Creates an MC which can use its experience in new, but similar situations

Benefits

- Allows the hardware designer to focus on what specific goal the MC should accomplish
- Enables the MC to adapt to workload changes
- Hopefully enables the MC to make farsighted decisions
- Creates an MC which can use its experience in new, but similar situations
- **More efficiently utilize DRAM bandwidth (21% more utilization over FR-FCFS)**

Benefits

- Allows the hardware designer to focus on what specific goal the MC should accomplish
- Enables the MC to adapt to workload changes
- Hopefully enables the MC to make farsighted decisions
- Creates an MC which can use its experience in new, but similar situations
- **More efficiently utilize DRAM bandwidth (21% more utilization over FR-FCFS)**
- **Improved system performance (19% performance improvement over FR-FCFS)**

Mechanisms

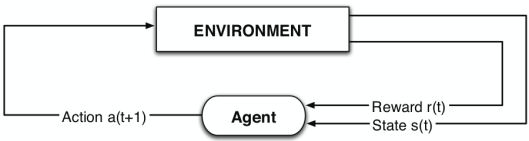
Reinforcement Learning

Reinforcement Learning

Infinite-horizon task: An endless task where we observe the state, take an action, and get a reward in each turn.

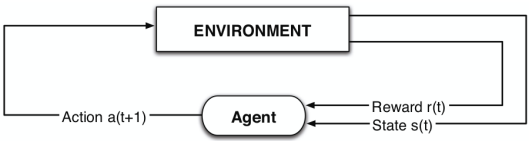
Reinforcement Learning

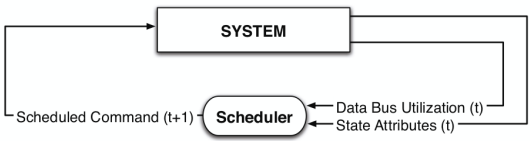
Infinite-horizon task: An endless task where we observe the state, take an action, and get a reward in each turn.



Reinforcement Learning

Infinite-horizon task: An endless task where we observe the state, take an action, and get a reward in each turn.





Overview

Overview

Formulate memory access scheduling as an *infinite horizon (continuous) task*
Scheduler always has **1 DRAM clock cycle** to decide what it wants to do

Overview

Formulate memory access scheduling as an *infinite horizon (continuous) task*

Scheduler always has **1 DRAM clock cycle** to decide what it wants to do

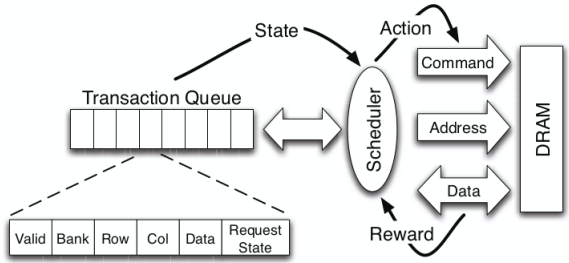


Figure 4: High-level overview of an RL-based scheduler.

Overview

Formulate memory access scheduling as an *infinite horizon (continuous) task* Scheduler always has **1 DRAM clock cycle** to decide what it wants to do

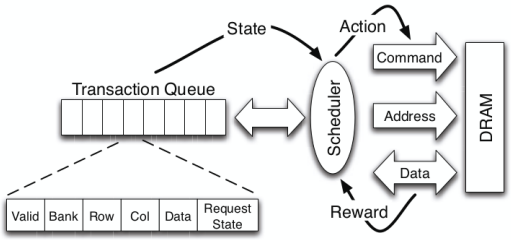


Figure 4: High-level overview of an RL-based scheduler.

Reward = 1 if a read/write command was issued. Otherwise, Reward = 0

Reward function

Reward function

At every time step, the scheduler will make the decision it thinks will maximize the reward function, which is defined as[1]

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

[1] R. Sutton and A. Barto. Reinforcement Learning. MIT Press, Cambridge, MA, 1998.

Reward function

At every time step, the scheduler will make the decision it thinks will maximize the reward function, which is defined as[2]

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$\gamma \in [0, 1]$$

$$\gamma \approx 1 \rightarrow \textit{farsighted}$$

$$\gamma \approx 0 \rightarrow \textit{greedy}$$

[2] R. Sutton and A. Barto. Reinforcement Learning. MIT Press, Cambridge, MA, 1998.

Reward function

At every time step, the scheduler will make the decision it thinks will maximize the reward function, which is defined as[3]

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Note that we also have

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma \big(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \big) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

$\gamma \in [0, 1]$

$\gamma \approx 1 \rightarrow$ *farsighted*

$\gamma \approx 0 \rightarrow$ *greedy*

[3] R. Sutton and A. Barto. Reinforcement Learning. MIT Press, Cambridge, MA, 1998.

Q-values

Q-values

- Q-values enable us to assign credit & blame to past actions

Q-values

- Q-values enable us to assign credit & blame to past actions
- We define the Q-value of a state-action pair for a specific policy as **the expected value of the reward function if we take action a in state s and follow policy π afterwards:**

Q-values

- Q-values enable us to assign credit & blame to past actions
- We define the Q-value of a state-action pair for a specific policy as **the expected value of the reward function if we take action a in state s and follow policy π afterwards:**[4]

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

[4] R. Sutton and A. Barto. Reinforcement Learning. MIT Press, Cambridge, MA, 1998.

States

For each candidate command, the associated state has 6 attributes:

States

For each candidate command, the associated state has 6 attributes:

1. # Reads in the queue

States

For each candidate command, the associated state has 6 attributes:

1. # Reads in the queue
2. # Reads in the queue that are load misses

States

For each candidate command, the associated state has 6 attributes:

1. # Reads in the queue
2. # Reads in the queue that are load misses
3. # Writes in the queue

States

For each candidate command, the associated state has 6 attributes:

1. # Reads in the queue
2. # Reads in the queue that are load misses
3. # Writes in the queue
4. # Writes in the queue waiting for the row referenced by this command

States

For each candidate command, the associated state has 6 attributes:

1. # Reads in the queue
2. # Reads in the queue that are load misses
3. # Writes in the queue
4. # Writes in the queue waiting for the row referenced by this command
5. # Load misses (which are the oldest load misses from their cores) in the queue waiting for the row referenced by this command

States

For each candidate command, the associated state has 6 attributes:

1. # Reads in the queue
2. # Reads in the queue that are load misses
3. # Writes in the queue
4. # Writes in the queue waiting for the row referenced by this command
5. # Load misses (which are the oldest load misses from their cores) in the queue waiting for the row referenced by this command
6. The order of the load relative to other loads from C (if this command is *related to* a load miss by core C)

States

For each candidate command, the associated state has 6 attributes:

1. # Reads in the queue
2. # Reads in the queue that are load misses
3. # Writes in the queue
4. # Writes in the queue waiting for the row referenced by this command
5. # Load misses (which are the oldest load misses from their cores) in the queue waiting for the row referenced by this command
6. The order of the load relative to other loads from C (if this command is *related to* a load miss by core C)

All attributes available in the controller's transaction queue $\rightarrow$ fast access

RL-Based DRAM Scheduling Algorithm

RL-Based DRAM Scheduling Algorithm

In every DRAM cycle:

RL-Based DRAM Scheduling Algorithm

In every DRAM cycle:

Issue the command you selected in the last cycle

RL-Based DRAM Scheduling Algorithm

In every DRAM cycle:

- Issue the command you selected in the last cycle
- Observe reward

RL-Based DRAM Scheduling Algorithm

In every DRAM cycle:

- Issue the command you selected in the last cycle
- Observe reward
- With small probability ϵ:
  - Select a random command #to explore the states

RL-Based DRAM Scheduling Algorithm

In every DRAM cycle:

- Issue the command you selected in the last cycle
- Observe reward
- With small probability ϵ:
  - Select a random command #to explore the states
- With probability 1 - ϵ:
  - Select the command with the highest Q-value among all legal commmands

RL-Based DRAM Scheduling Algorithm

In every DRAM cycle:

- Issue the command you selected in the last cycle
- Observe reward
- With small probability ϵ:
  - Select a random command #to explore the states
- With probability 1 - ϵ:
  - Select the command with the highest Q-value among all legal commmands
- Update the Q-value of the previous command

RL-Based DRAM Scheduling Algorithm

In every DRAM cycle:

- Issue the command you selected in the last cycle
- Observe reward
- With small probability ϵ:
  - Select a random command #to explore the states
- With probability 1 - ϵ:
  - Select the command with the highest Q-value among all legal commmands
- $Q_{prev} \leftarrow (1 - \alpha)Q_{prev} + \alpha(r + \gamma Q_{selected})$

Note: Correct operation is ensured by adding a set of extra constraints

How to store all the Q-values efficiently

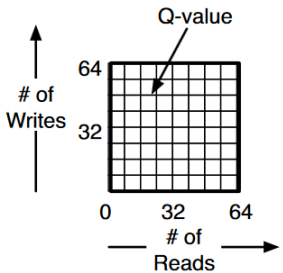


Figure: Fine-grain

How to store all the Q-values efficiently

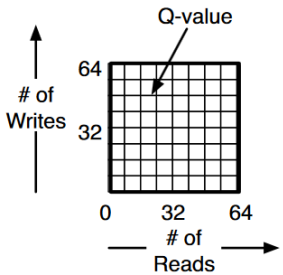


Figure: Fine-grain

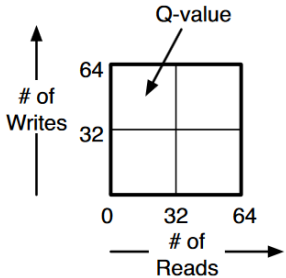


Figure: Coarse-grain

How to store all the Q-values efficiently

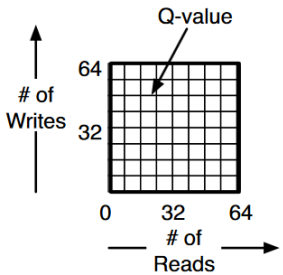


Figure: Fine-grain

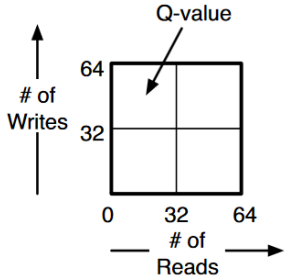


Figure: Coarse-grain

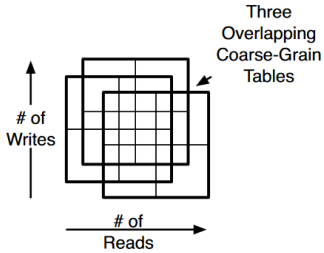


Figure: CMAC

Implementation

Implementation

$$cmd \leftarrow \text{select\_command\_with\_max\_Q-value}(C)$$

Implementation

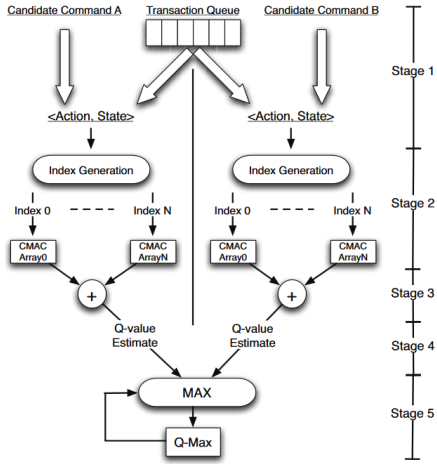
$$cmd \leftarrow \text{select\_command\_with\_max\_Q-value}(C)$$

- Assumption: Scheduler's pipe can be clocked 10 times each DRAM cycle

Implementation

$$cmd \leftarrow \text{select\_command\_with\_max\_Q-value}(C)$$

- Assumption: Scheduler's pipe can be clocked 10 times each DRAM cycle

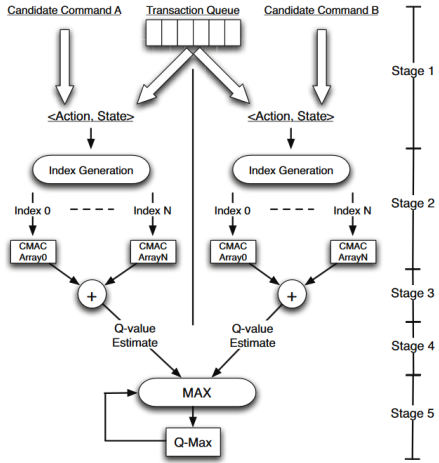


(a) 2-wide Q-value Estimation Pipeline

Implementation

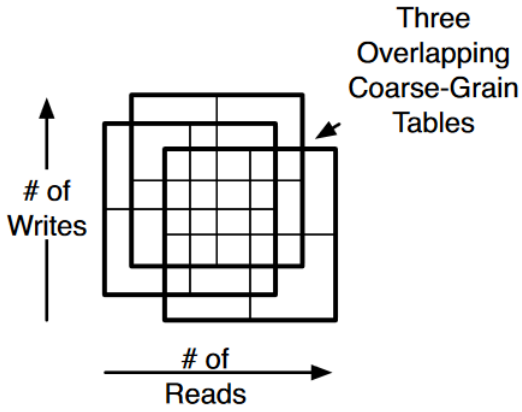
$$cmd \leftarrow \text{select\_command\_with\_max\_Q-value}(C)$$

- Assumption: Scheduler's pipe can be clocked 10 times each DRAM cycle
- Scheduler can consider 12 commands every cycle

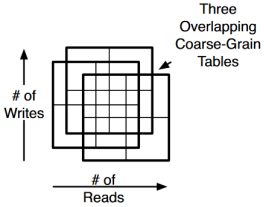


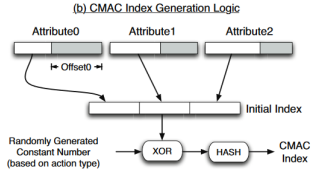
(a) 2-wide Q-value Estimation Pipeline

Implementation

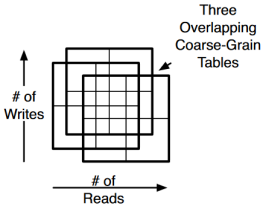


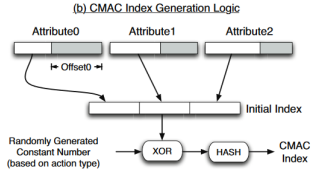
Implementation





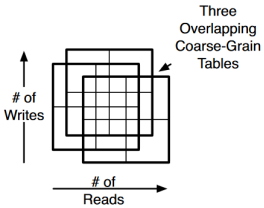
Implementation



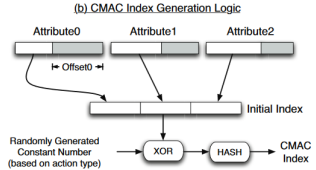


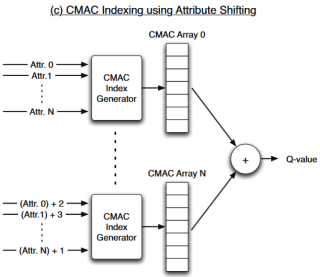
- A constant number per action type $\rightarrow$ prevent generalization across different commands

Implementation

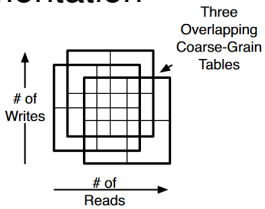


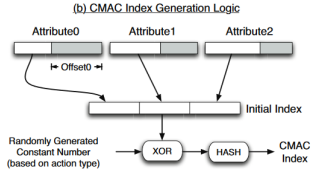
- A constant number per action type → prevent generalization across different commands

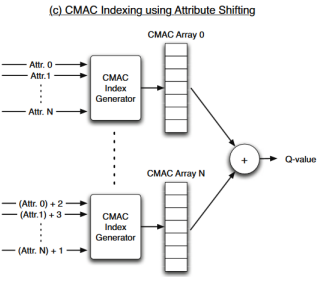




Implementation







- A constant number per action type $\rightarrow$ prevent generalization across different commands
- random shifts of attributes implement the shiftedness of CMAC arrays

(Additional) Hardware Overhead

(Additional) Hardware Overhead

1. logic to compute state attributes $\rightarrow$ counters that are updated every DRAM cycle

(Additional) Hardware Overhead

1. logic to compute state attributes $\rightarrow$ counters that are updated every DRAM cycle
2. logic to update Q-values $\rightarrow$ single pipelined 16-bit fixed-point multiplier

(Additional) Hardware Overhead

1. logic to compute state attributes $\rightarrow$ counters that are updated every DRAM cycle
2. logic to update Q-values $\rightarrow$ single pipelined 16-bit fixed-point multiplier
3. storing the Q-values $\rightarrow$ 32 kB on-chip storage

Key Results: Methodology & Evaluation

Methodology

Some important parameters of the simulated CMP:

- Frequency: 4 GHz
- #Cores: 4 (each 2-way simultaneously multithreaded)
- iL1/dL1 size: 32 kB
- Shared L2 cache: 4MB, 8-way

Methodology

Some important parameters of the simulated CMP:

- Frequency: 4 GHz
- #Cores: 4 (each 2-way simultaneously multithreaded)
- iL1/dL1 size: 32 kB
- Shared L2 cache: 4MB, 8-way

Some important parameters of the simulated DRAM:

- DDR2-800 SDRAM
- Transaction Queue: 64 entries
- Peak Data Rate: 6.4 GB/s
- DRAM Bus Frequency: 400 MHz
- Single rank with 4 DRAM chips
- #Banks: 4 per DRAM chip
- Row Buffer Size: 2 KB

Applications & Benchmarks

Benchmark	Description	Problem size
Data Mining		
SCALPARC	Decision Tree	125k pts., 32 attributes
NAS OpenMP		
MG	Multigrid Solver	Class A
CG	Conjugate Gradient	Class A
SPEC OpenMP		
SWIM-OMP	Shallow water model	MinneSpec-Large
EQUAKE-OMP	Earthquake model	MinneSpec-Large
ART-OMP	Self-Organizing Map	MinneSpec-Large
Splash-2		
OCEAN	Ocean movements	514×514 ocean
FFT	Fast Fourier transform	1M points
RADIX	Integer radix sort	2M integers

Table 3: Simulated applications and their input sizes.

Compared Memory Controllers

Compared Memory Controllers

- A conventional in-order MC

Compared Memory Controllers

- A conventional in-order MC
- MC implementing FR-FCFS

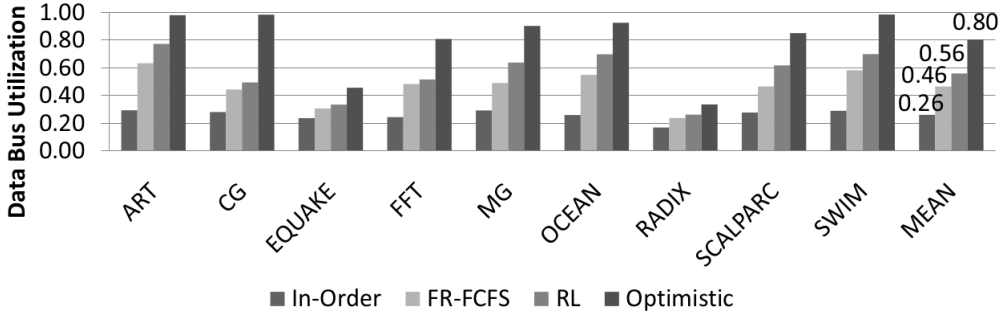
Compared Memory Controllers

- A conventional in-order MC
- MC implementing FR-FCFS
- RL-based controller proposed by this paper

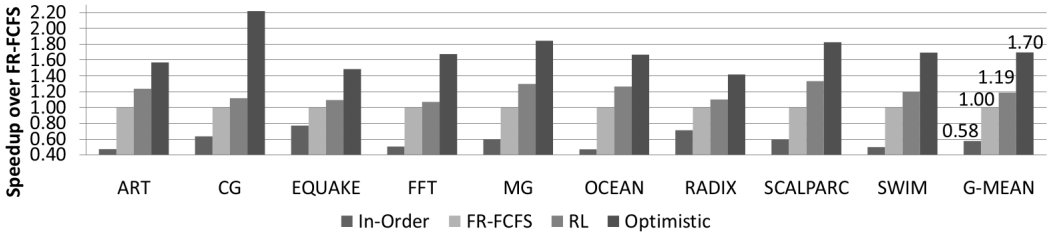
Compared Memory Controllers

- A conventional in-order MC
- MC implementing FR-FCFS
- RL-based controller proposed by this paper
- An ideal scheduler with an ideal memory that can sustain 100% peak bandwidth

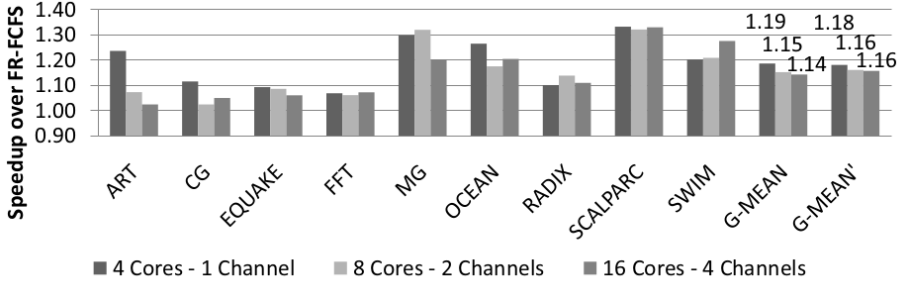
Results: Data Bus Utilization



Results: Performance Improvement



Results: Scaled to More Cores



Executive Summary

Executive Summary

Problem: Scheduling policies

- Cannot anticipate the long term effects of their scheduling decisions
- Cannot take lessons from the consequences of their past actions

Solution: RL-based controller computes learning, far-sighted policy $\rightarrow$ efficient bandwidth utilization

Results:

- 19% speedup, 21% more bandwidth utilization over best static policy
- Scales as well as the best static policy

Strengths & Weaknesses

Strengths

Strengths

- A fundamentally more powerful approach than its predecessors

Strengths

- A fundamentally more powerful approach than its predecessors
- Tries to solve an important problem that will always be relevant

Strengths

- A fundamentally more powerful approach than its predecessors
- Tries to solve an important problem that will always be relevant
- Significantly improves overall performance and data bus utilization

Strengths

- A fundamentally more powerful approach than its predecessors
- Tries to solve an important problem that will always be relevant
- Significantly improves overall performance and data bus utilization
- The paper accurately predicts that the DRAM bandwidth is going to be the main problem, 11 years ago!

Strengths

- A fundamentally more powerful approach than its predecessors
- Tries to solve an important problem that will always be relevant
- Significantly improves overall performance and data bus utilization
- The paper accurately predicts that the DRAM bandwidth is going to be the main problem, 11 years ago!
- Well-written paper

Weaknesses

Weaknesses

- It does not provide fairness across multiple competing threads. It does not even provide non-starvation.

Weaknesses

- It does not provide fairness across multiple competing threads. It does not even provide non-starvation.
- More complicated hardware than FR-FCFS

Weaknesses

- It does not provide fairness across multiple competing threads. It does not even provide non-starvation.
- More complicated hardware than FR-FCFS
- Extending it is hard since hardware will get even more complicated

Weaknesses

- It does not provide fairness across multiple competing threads. It does not even provide non-starvation.
- More complicated hardware than FR-FCFS
- Extending it is hard since hardware will get even more complicated
- Heterogeneous workloads are not tested

Can we do better?

Can we do better?

- Can we solve the fairness problem?

Can we do better?

- Can we solve the fairness problem?
  - "ATLAS" [*Y. Kim, D. Han, O. Mutlu and M. Harchol-Balter, HPCA 2010*]

ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers

Yoongu Kim Dongsu Han Onur Mutlu Mor Harchol-Balter
Carnegie Mellon University

ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers

Yoongu Kim Dongsu Han Onur Mutlu Mor Harchol-Balter
Carnegie Mellon University

- Periodically order threads based on the service they have attained from the memory controllers so far

ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers

Yoongu Kim Dongsu Han Onur Mutlu Mor Harchol-Balter
Carnegie Mellon University

- Periodically order threads based on the service they have attained from the memory controllers so far
- Prioritize the threads that have attained the least service over others in each period

Can we do better?

- Can we solve the fairness problem?
  - "ATLAS" [*Y. Kim, D. Han, O. Mutlu and M. Harchol-Balter, HPCA 2010*]
  - TCM scheduling [*Y. Kim, M. Papamichael, O. Mutlu, M. Harchol-Balter, MICRO 2010*]

Thread Cluster Memory Scheduling

- Dynamically group threads into a latency-sensitive cluster and a bandwidth-sensitive cluster

Thread Cluster Memory Scheduling

- Dynamically group threads into a latency-sensitive cluster and a bandwidth-sensitive cluster
- Prioritize 1st cluster over 2nd cluster

Thread Cluster Memory Scheduling

- Dynamically group threads into a latency-sensitive cluster and a bandwidth-sensitive cluster
- Prioritize 1st cluster over 2nd cluster
- Employ different policies within each cluster

Can we do better?

- Can we solve the fairness problem?
  - "ATLAS" [*Y. Kim, D. Han, O. Mutlu and M. Harchol-Balter, HPCA 2010*]
  - TCM scheduling [*Y. Kim, M. Papamichael, O. Mutlu, M. Harchol-Balter, MICRO 2010*]
  - "BLISS" [*L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, O. Mutlu, IEEE Transactions on Parallel and Distributed Systems 2016*]

BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling

Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu

BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling

Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu

- Mark each application either as vulnerable-to-interference or as interference-causing.

BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling

Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu

- Mark each application either as vulnerable-to-interference or as interference-causing.
- Prioritize requests from 1st group over requests from 2nd group

Key Takeaways

Key Takeaways

- A novel approach to utilize the data bus

Key Takeaways

- A novel approach to utilize the data bus
- Effective in terms of performance gain

Key Takeaways

- A novel approach to utilize the data bus
- Effective in terms of performance gain
- Comes with some hardware cost

Key Takeaways

- A novel approach to utilize the data bus
- Effective in terms of performance gain
- Comes with some hardware cost
- QoS-unaware $\rightarrow$ no fairness

Key Takeaways

- A novel approach to utilize the data bus
- Effective in terms of performance gain
- Comes with some hardware cost
- QoS-unaware $\rightarrow$ no fairness
- Seemingly hard to improve

Open Discussion

Discussion

How can we solve the fairness problem while keeping our RL-based approach?

Discussion

Are there other flaws in this approach?

Discussion

Can this approach be used to solve other scheduling problems?

Discussion

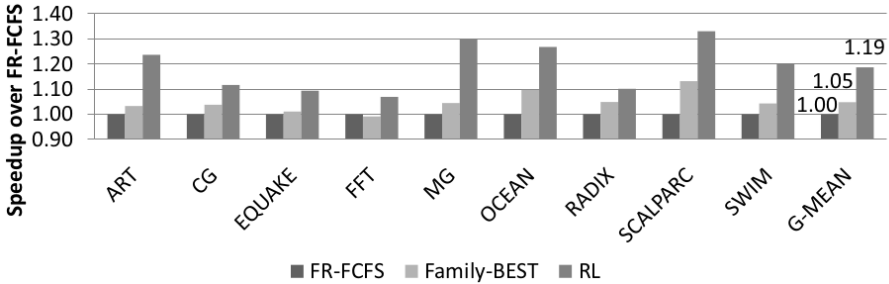
When are machine-learning based approaches applicable in computer architecture?

Backup slides

Ensuring correct operation

- the scheduler is not permitted to select NOPs when other legal commands are available
- the scheduler is only allowed to activate rows due to pending requests in the transaction queue (i.e., the scheduler cannot choose to activate an arbitrary row with no pending requests)
- the scheduler is not allowed to precharge a newly activated row until it issues a reador write command to it.

Results: RL versus Family-BEST



Results: RL versus Family-BEST

Preference relations used in Family-BEST:

- Row commands over column commands
- Older commands over younger commands
- Reads over writes
- Load misses over store misses
- More critical load misses over less critical ones, based on sequence numbers

Results: Online versus Offline