ETH zürich

Self-Optimizing Memory Controllers: A Reinforcement Learning Approach - ISCA 2008

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

[1] Cornell University

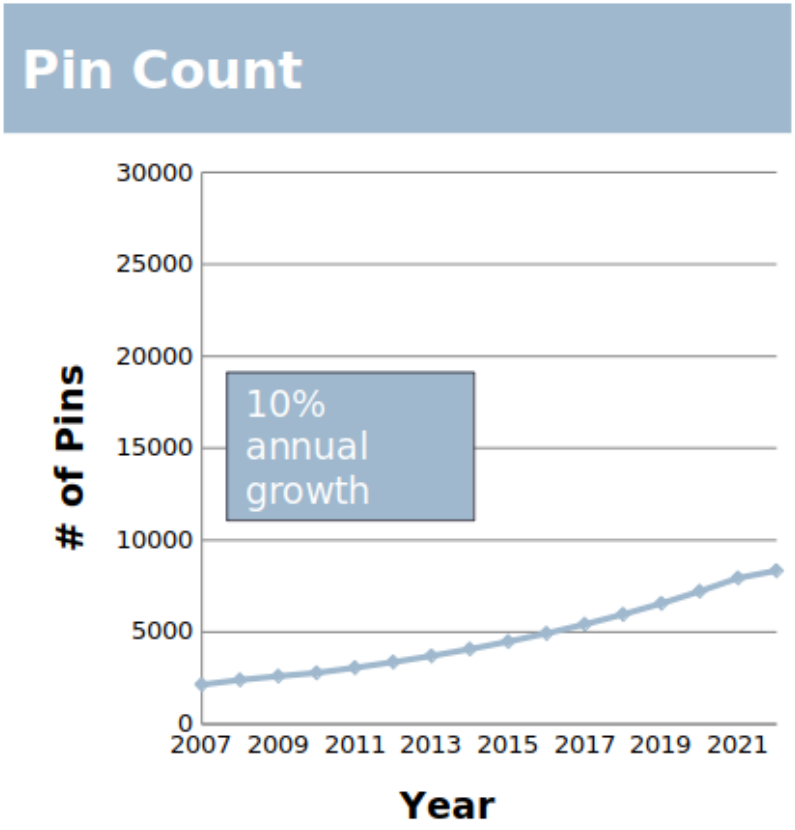
[2] Microsoft Research

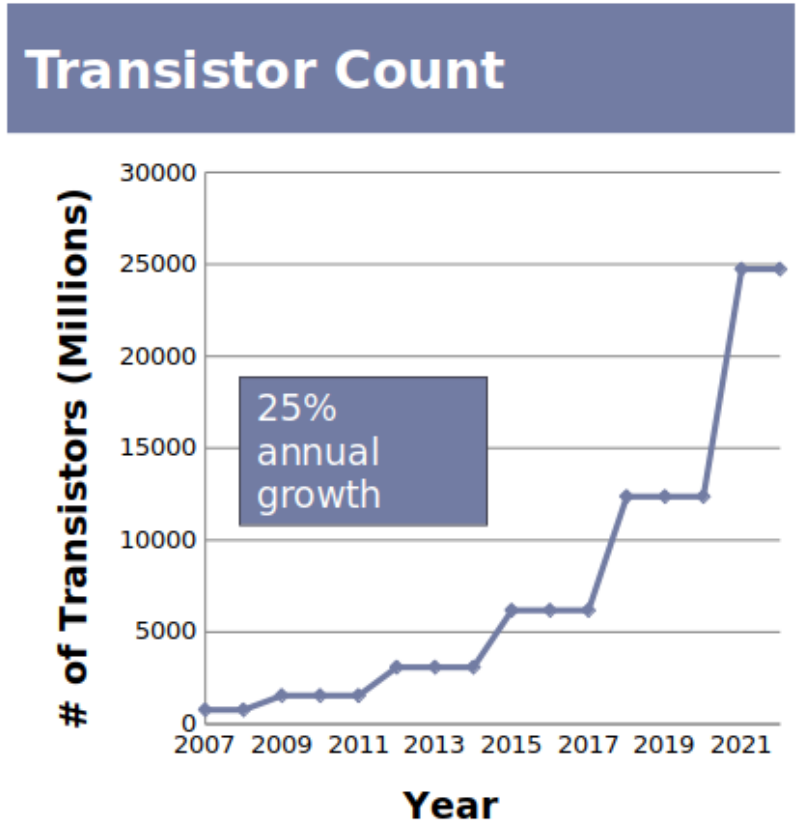
Presented by Marco Zeller

Outline

1. Motivation and Background
2. Mechanisms
3. Results
4. Summary
5. Discussion

Moore's Law and Memory Controller





Projection according to ITRS 2007 Executive Summary

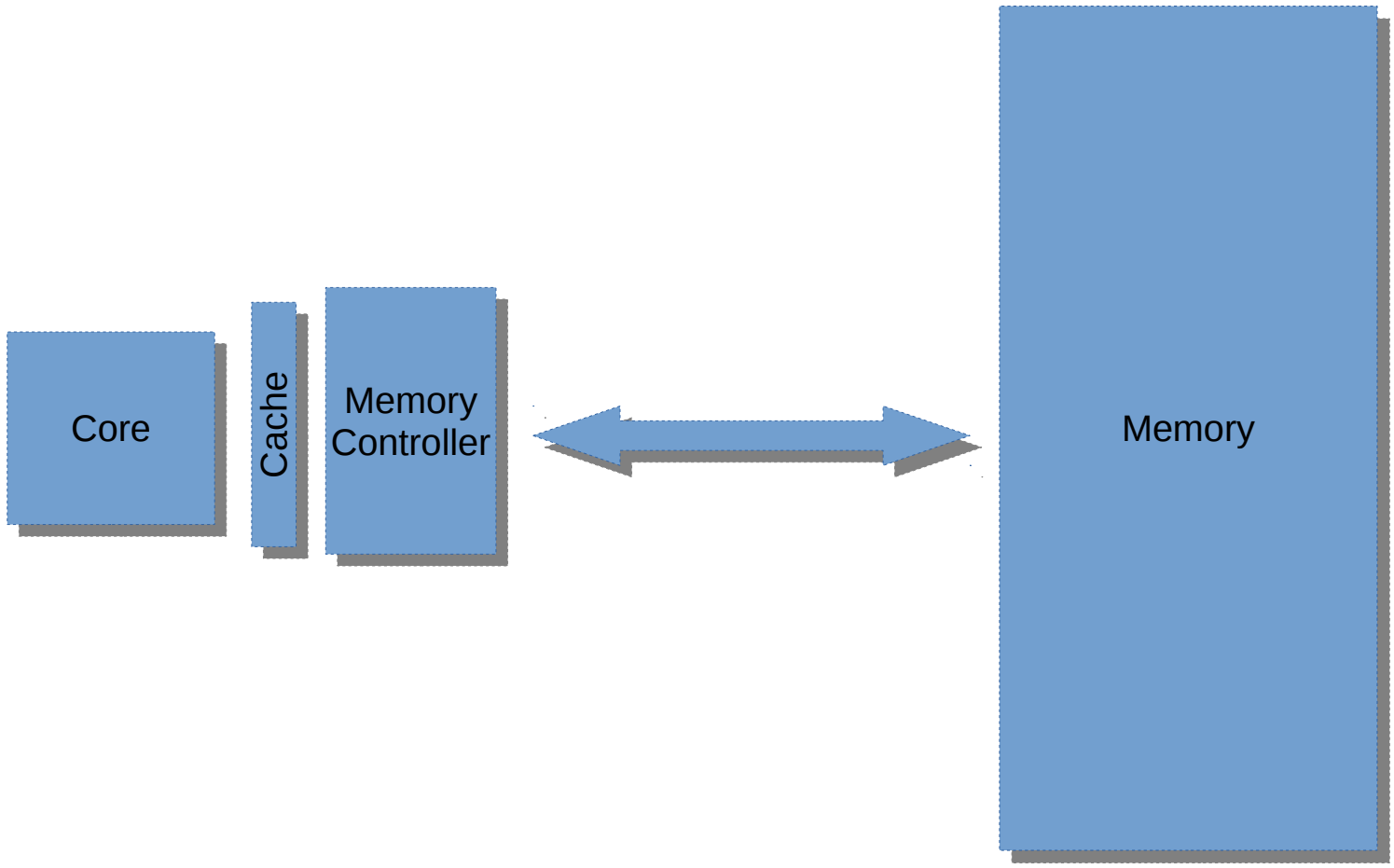
Off-Chip Memory Bandwidth Observations

Workload is increasing faster than the available bandwidth

Higher pressure on off-chip interface with each new technology generation

Important to utilize available memory bandwidth efficiently

The Memory Controller



Memory Scheduling

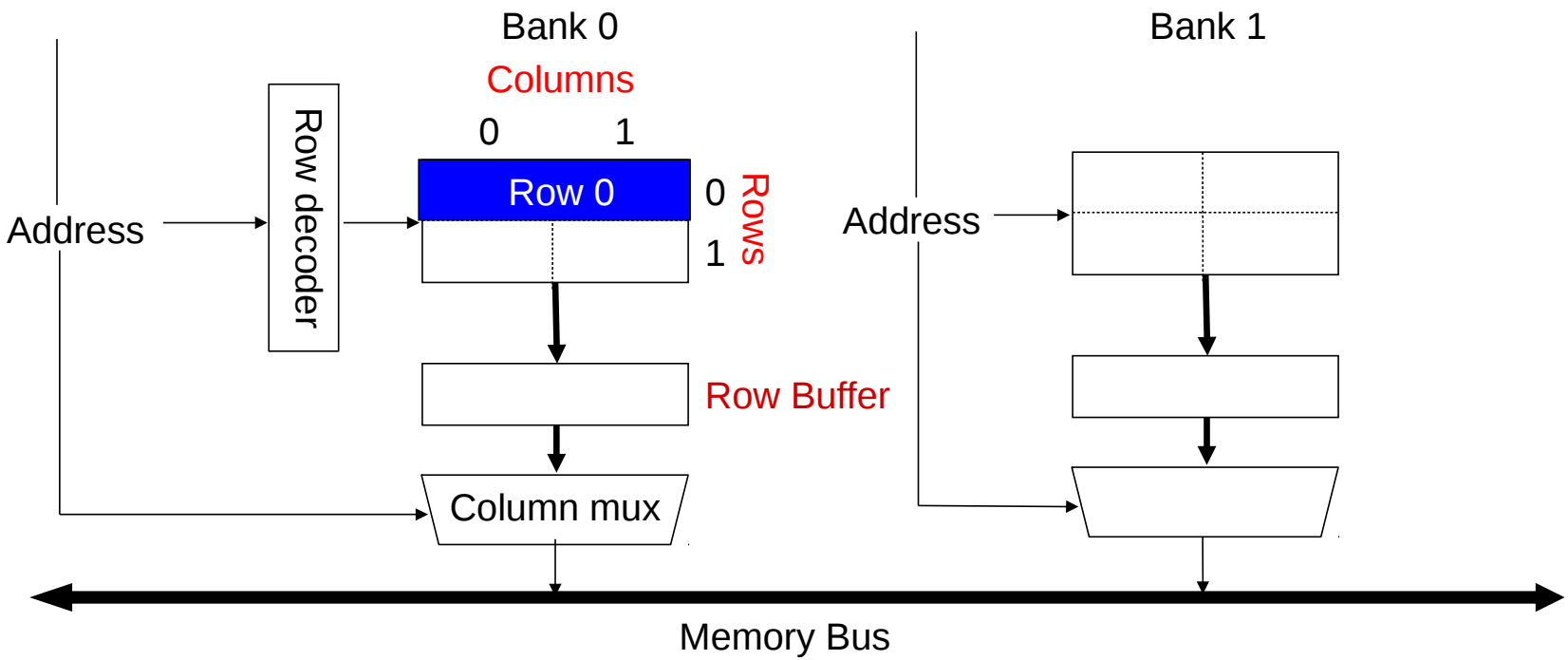
We consider 4 DRAM-Interface commands:

1. Activate (row) 'Open'
2. Precharge (row) 'Close'
3. Read
4. Write

The paper distinguishes 5 commands (performance)

Row Buffer

Activate



State of the Art Memory Controller

FR-FCFS (first-ready first-come first-serve) policy

Provides the best average performance

Drawbacks:

- Designed for average-case application behavior
- Does not consider long-term performance impacts
- Does not adapt its scheduling policy

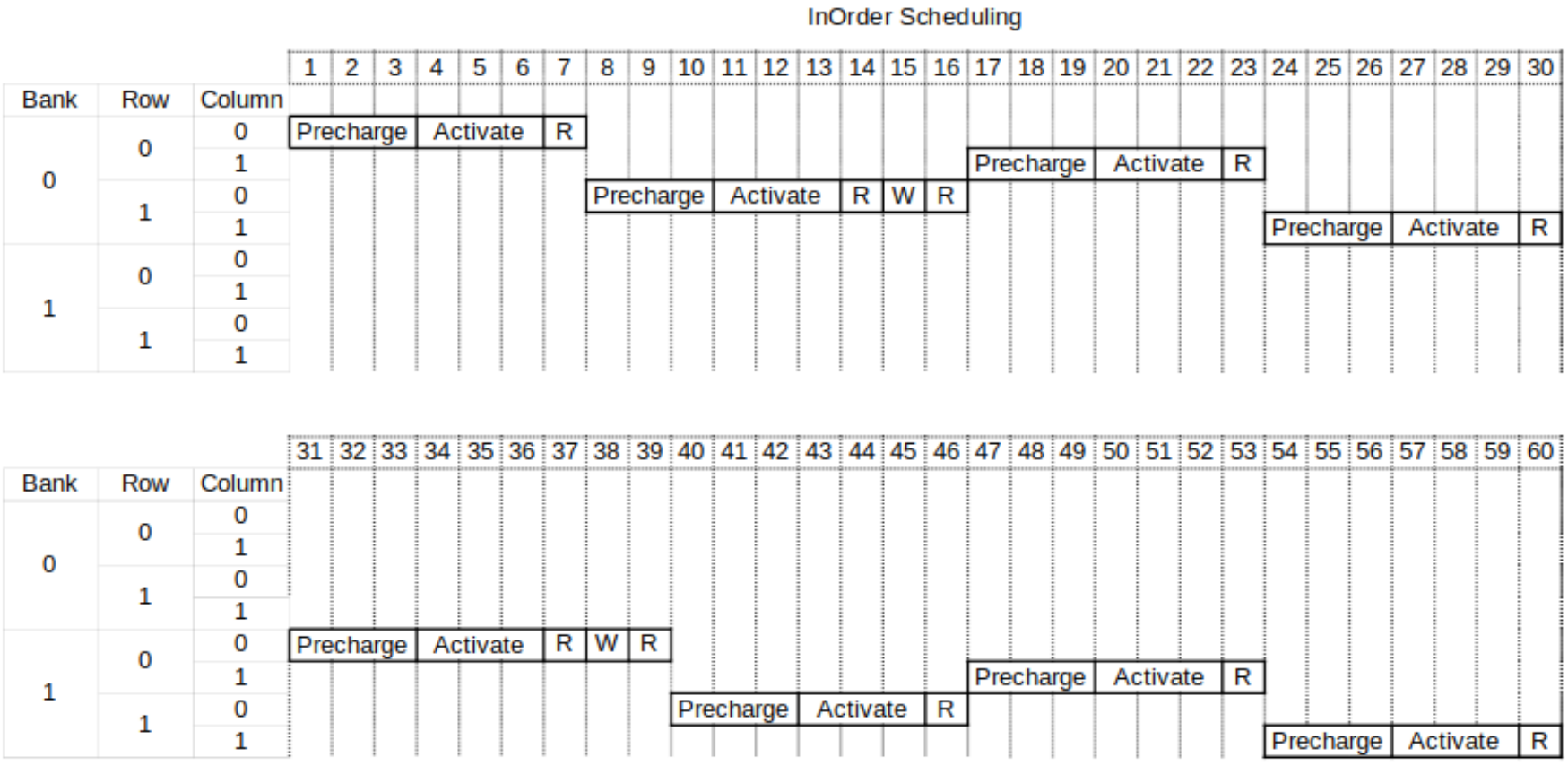
Assume the following workload: action (bank, row, column)

1. Read (0,0,0)
2. Read (0,1,0)
3. Write (0,1,0)
4. Read (0,1,0)
5. Read (0,0,1)
6. Read (0,1,1)
7. Read (1,0,0)
8. Write (1,0,0)
9. Read (1,0,0)
10. Read (1,1,0)
11. Read (1,0,1)
12. Read (1,1,1)

Where:

Activate and Precharge occupy 3 DRAM-cycles

Read and Write occupy 1 DRAM-cycles


InOrder Scheduling
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Bank Row Column
0 0 0 Precharge Activate R
1 Precharge Activate R W R
1 0 Precharge Activate R
1 Precharge Activate R
1 0 0
1
1 0
1
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
Bank Row Column
0 0 0
1
1 0
1
1 0 0 Precharge Activate R W R
1 Precharge Activate R
1 0 Precharge Activate R
1 Precharge Activate R

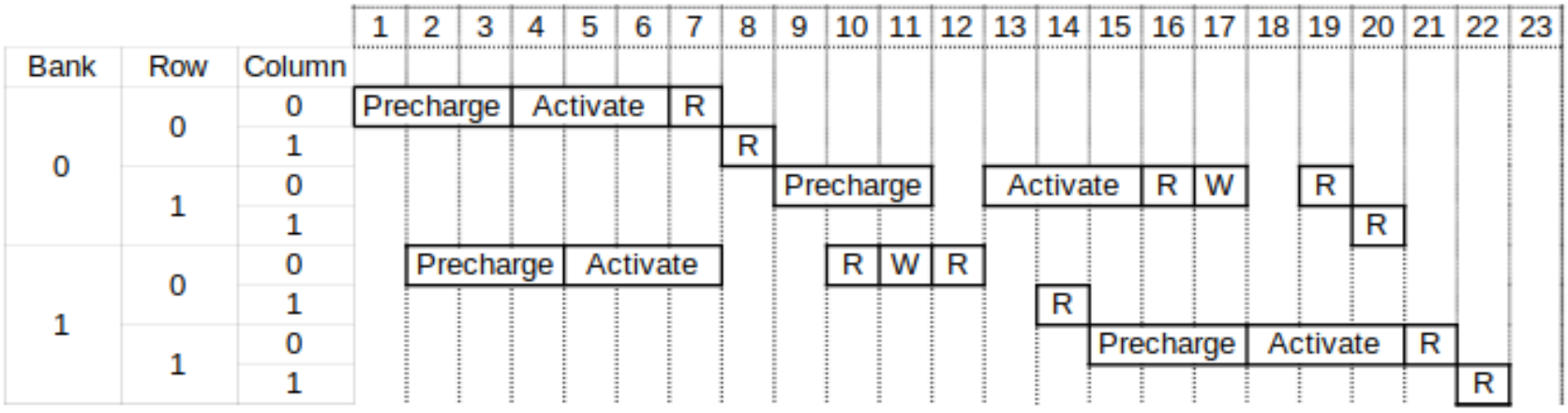
60 cycles!

FR-FCFS Scheduling

Bank	Row	Column	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0	0	Precharge			Activate			R																
		1								R															
	1	0												Precharge			Activate			R				W	R
		1																			R				
1	0	0		Precharge			Activate				R	W	R												
		1													R										
	1	0														Precharge			Activate			R			
		1																					R		

23 cycles!

Improved Scheduling



22 cycles!

How Much Room Is There for Improvement?

Used for paper: **”Optimistic Scheduler”**

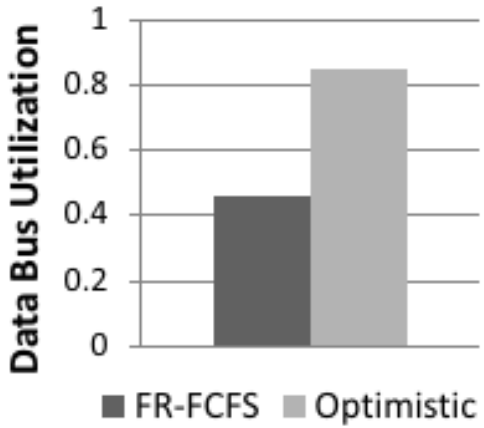
All timing constraints lifted (except DRAM data bus conflicts)

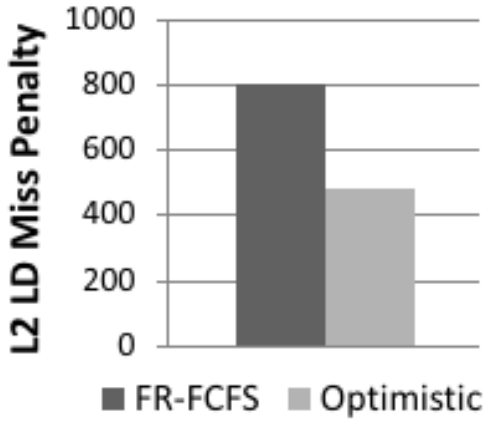
Able to use the full bandwidth at all time

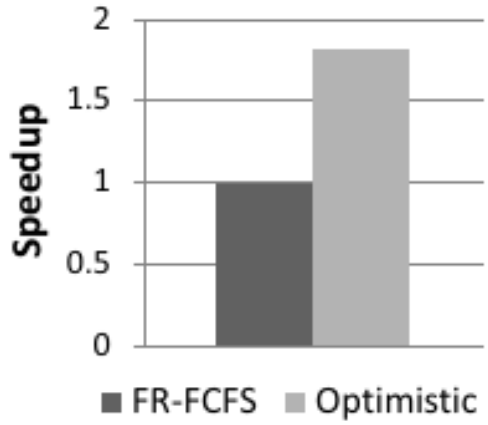
Not realizable!

- but easy to implement
- useful approximation for upper bound

FR-FCFS Compared to Optimistic Scheduler







Novelty: Self-Optimizing Memory Controller

Observation:

Efficient memory controller must be able to:

1. **do long time planning**
2. **learn**

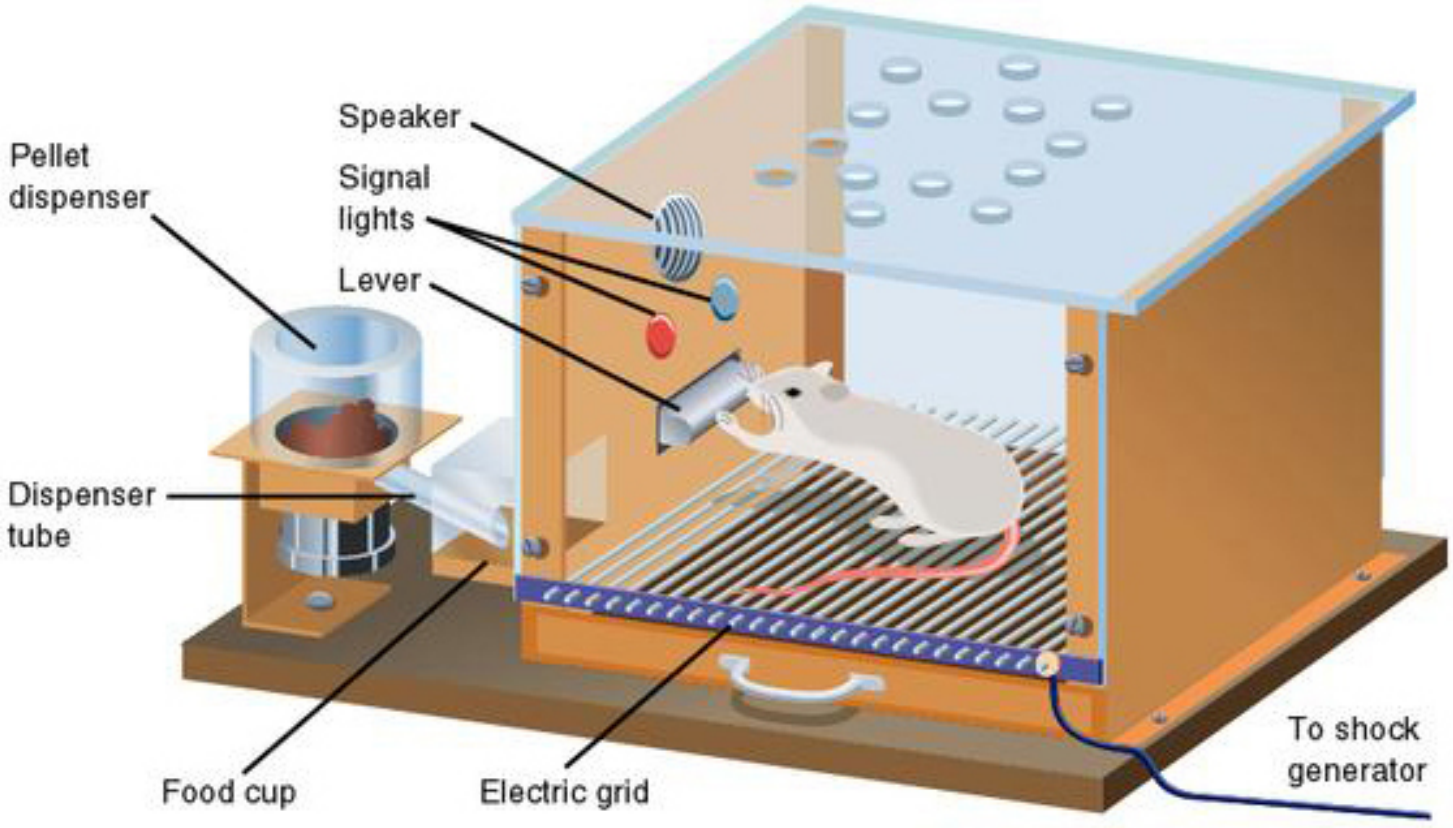
Solution:

Apply machine learning to solve a system architecture problem

Mechanisms

"Naturally formulated as an *infinite-horizon discounted Markov Decision Process*"

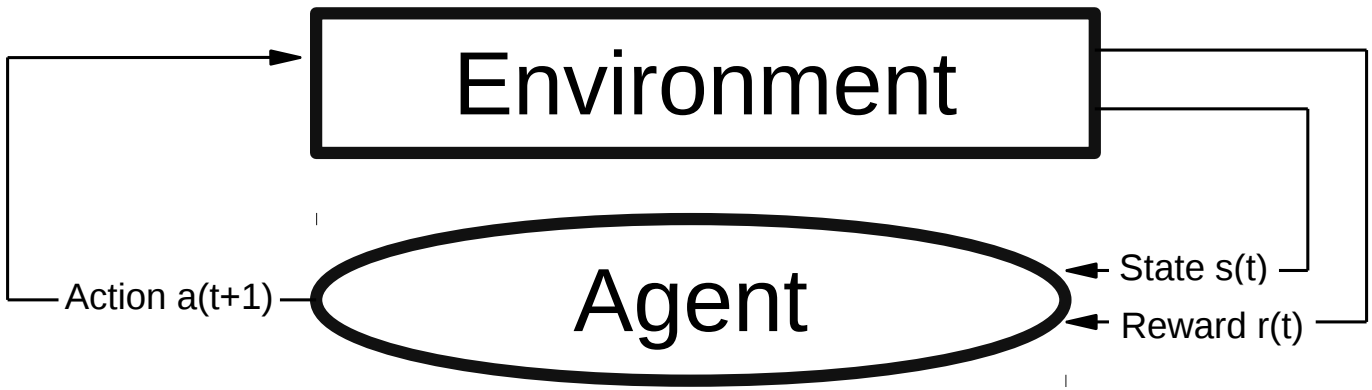
Potentially Complicated System



McLeod, S. A. (2018). Skinner - operant conditioning.
Retrieved from https://www.simplypsychology.org/operant-conditioning.html

Simple System for the Learning Agent

Reinforcement Learning Schematic



How to Obtain a Good Set of State Attributes

More state attributes $\Rightarrow$ increased hardware complexity

Handpick potential state attributes (requires some **intuition**)
in the paper 226 candidates were identified

Apply (linear) feature selection on those candidates

(Linear) Feature Selection

Automated process for finding a good (the best?) subset from N candidates

1. For every candidate (N) simulate an RL memory controller
2. Select the attribute yielding best performance
3. Simulate for every not selected candidate (N-1) an RL memory controller using the not selected attribute and the selected one
4. Repeat until preferred number of attributes reached

State Attributes (simplified)

1. Number of reads (load/store misses)
2. Number of writes (write-backs)
3. Number of reads (load misses)
4. Order of a load by core relative to other loads by the same core
5. Number of writes in the transaction queue waiting for the row
6. Number of load misses waiting for the row

Reward Function

The agent optimizes for the sum of it's future rewards

- Immediate reward of +1 every time it schedules a Read or Write
- No reward at all other times

Problem: agent's lifetime practically infinity

$\Rightarrow$ sum of the rewards will be infinity

Solution: Discounted Rewards

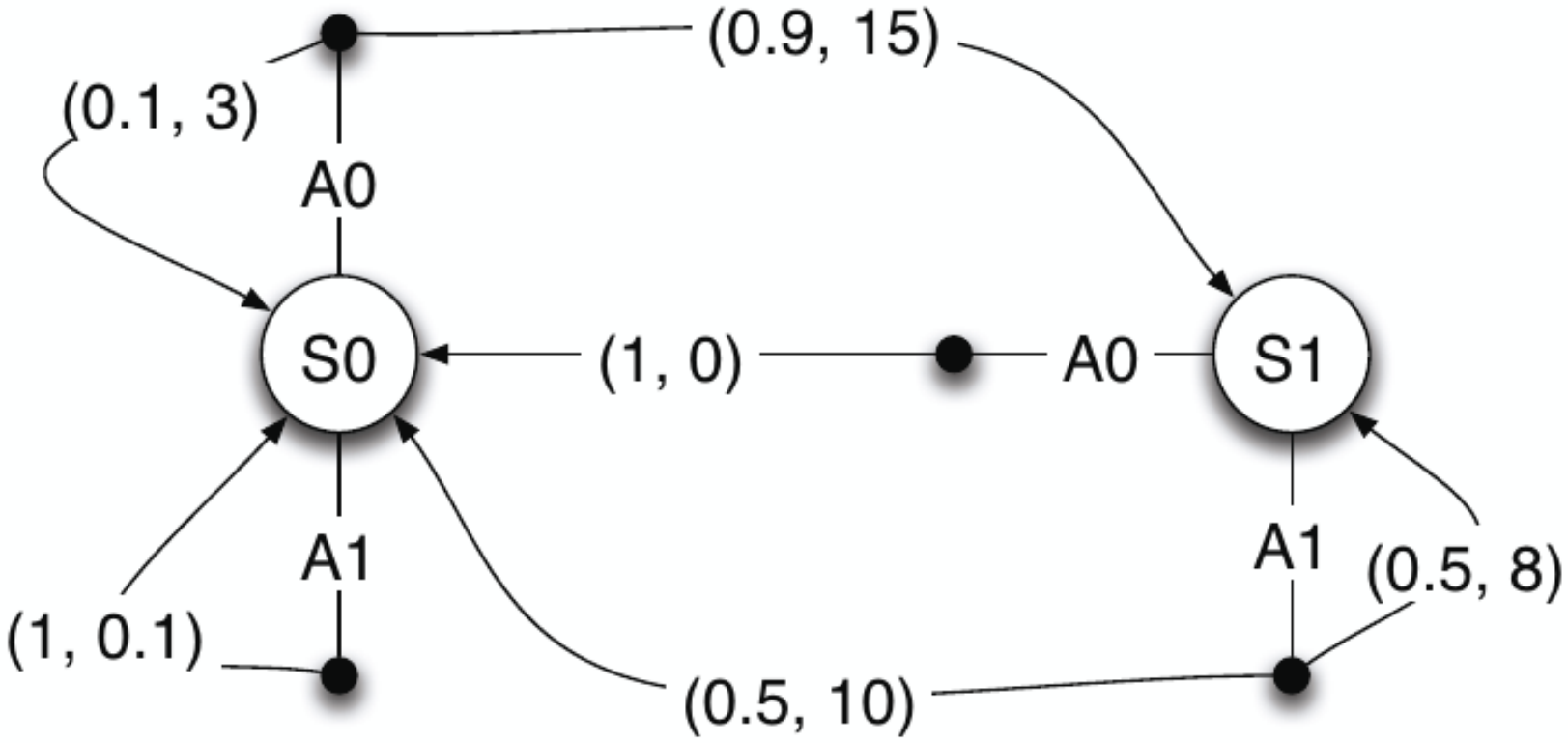
Discount parameter: $0 \leq \gamma < 1$

$$\mathbb{E}[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i}] = \mathbb{E}[r_t + \gamma^1 \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \gamma^3 \cdot r_{t+3} + ...]$$

Store these for all state-action pairs

$\Rightarrow$ Q-values

Example: Infinite-Horizon Discounted Markov Decision Process



Key Results

Experimental Setup

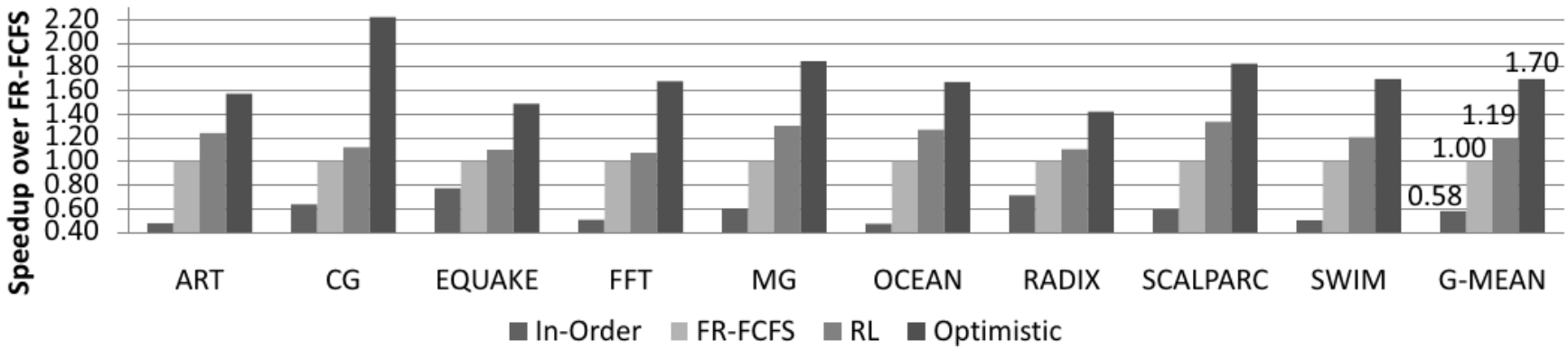
Loosely based on Intel's Nehalem processor
4 GHz Quadcore Processor
Shared L2 Cache: 4MB

Main Memory: DDR2 6.4 GB/s (1, 2, 4 Channels)
Transaction Queue: 64 Entries

Benchmarks: 9 mostly scientific memory-intensive applications
from *Data Mining*, *NAS OpenMP*, *SPEC OpenMP*, *Splash-2*

Simulator: heavily modified SESC

Speedup Compared to FR-FCFS

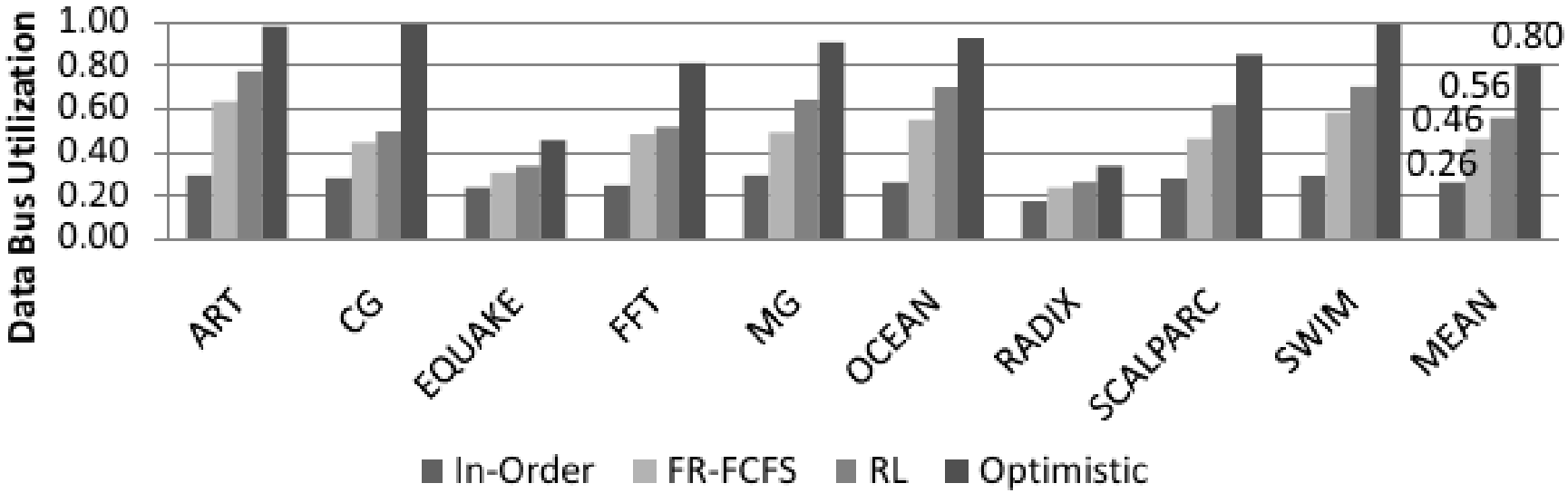


5%-33% speedup

19% speedup on average

"(27% of the possible speedup)"

Data-Bus Utilization



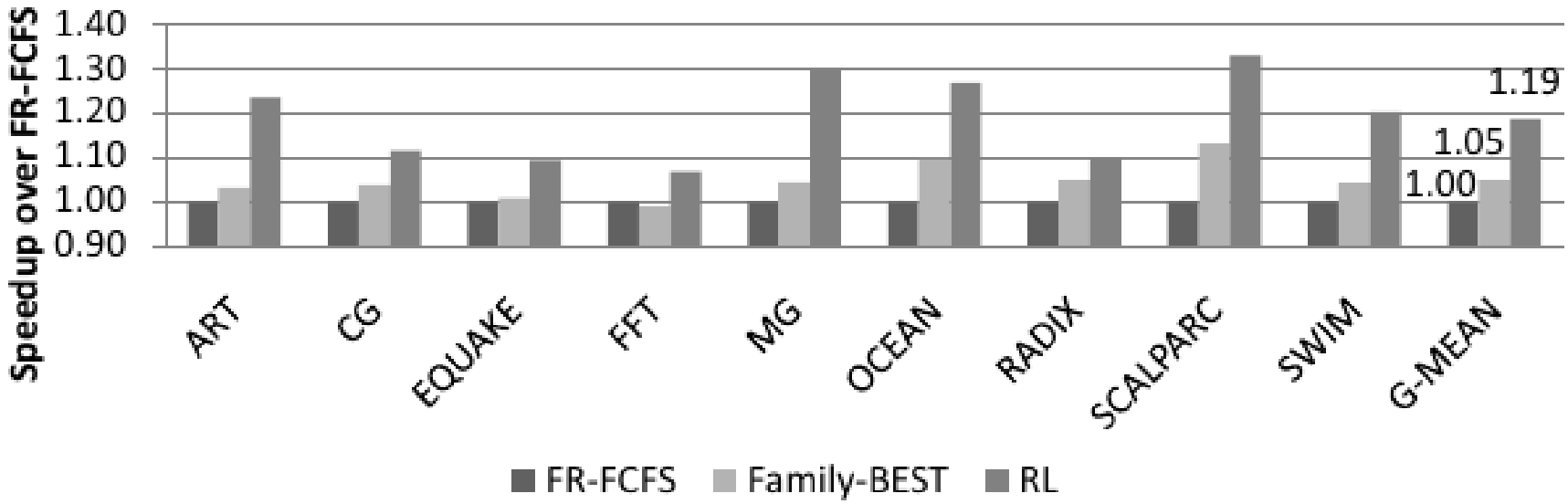
Fixed Policy With Same State Information

RL uses more information about the system than FR-FCFS

Generate a family of policies based on additional state-attributes

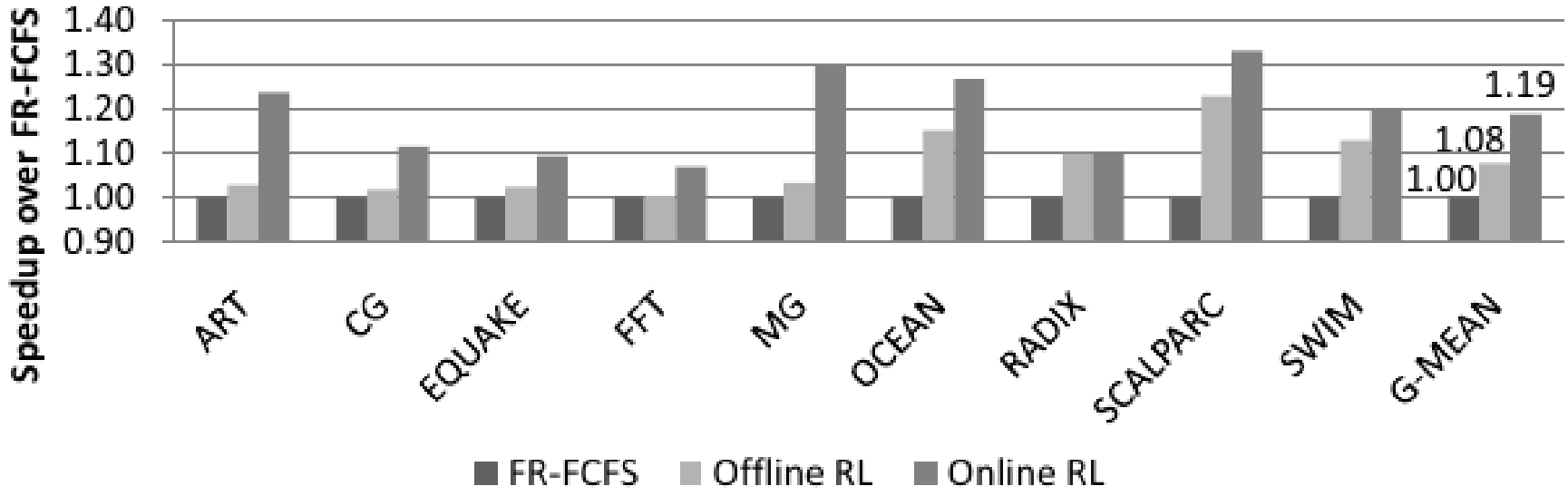
Pick the one that performs best in simulations
$\Rightarrow$ **Family-BEST**

Fixed Policy With Same State Information



using more state-attributes to improve FR-FCFS yields **only 5% speedup** on average

Speedup Compared to a Static RL Policy



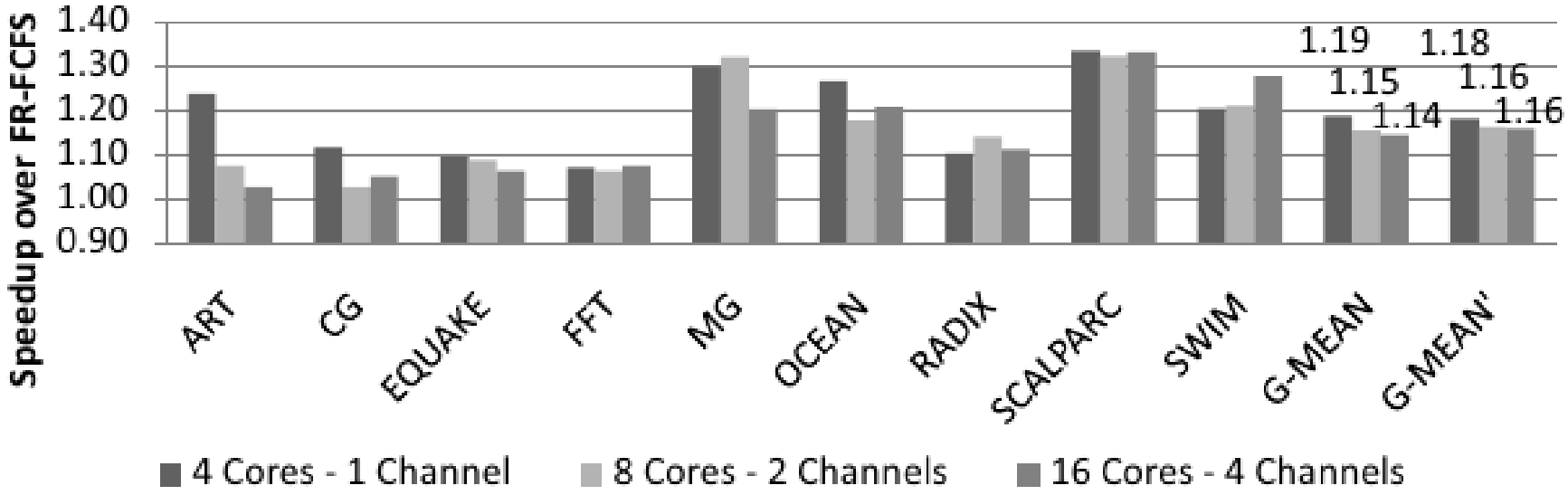
using an offline RL Memory Controller

yields **only 8% speedup** on average

⇒ online learning is essential

More Than Two Cores?

Can the Q-values successfully converge in the **presence of potential interactions** between schedulers? Yes!



Speedup Compared to Two Memory Channels

Hardware overhead:

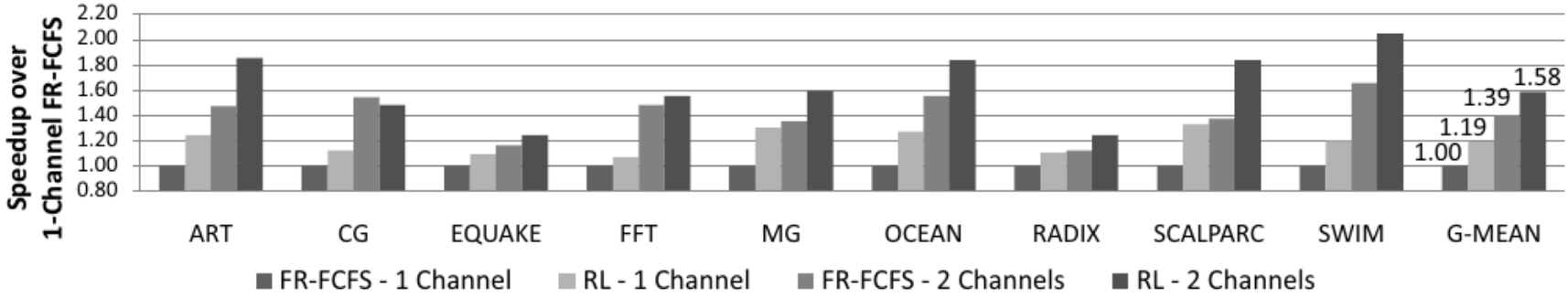
1. logic required to compute state attributes
2. logic required to estimate and update Q-values
3. SRAM arrays required to store Q-values.

8192 distinct Q-values (32bits each)

$\Rightarrow$ **32kB of on-chip storage**

Speedup Compared to Two Memory Channels

How about **increasing the memory bandwidth** instead of deploying a RL Memory Controller?



RL Memory Controller can deliver **50% of the performance increase** of doubling the bandwidth

$\Rightarrow$ lower cost than an over-provisioned system

Summary

Problem:	Off-chip memory bandwidth bottleneck
Goal:	Improve usage of available bandwidth
Observation:	Fixed policy memory controller are not optimal
Key idea:	Apply RL to a computer architecture problem **self-optimizing memory controller**
Key result:	**19% average speedup** compared to state-of-the-art controller (between 5% and 33% speedup for every tested application) Scalability to more processors/memory-channels

Strengths and Weaknesses

Strengths

Improves the memory bandwidth usage and therefore the performance.

Reduces the human design effort for the memory controller.
Black-box-model

Well written.

Weaknesses

State attributes attribute candidates identified **based on intuition**
Relatively high number (226) of candidates

The (linear) feature selection process used does not take into account potentially important interactions between attributes

The reward function proposed in the paper might not be ideal
$\Rightarrow$ Bias with all machine learning algorithms

The reward function proposed in the paper does not easily generalize

Thoughts and Ideas

My Opinion about the Paper

It is worthwhile reading.

Although ten years old still relevant (more than ever)

Interesting symbioses of two not often related topics (Machine Learning and Micro-architecture)

Detailed description about possible implementation

Suggested reading: MORSE Multi-objective Reconfigurable Self-optimizing Memory Scheduler - 2011

Takeaways

Combined knowledge from two little related fields

The authors took ideas from the field of **Data-Processing** and applied them to design a ’superior’ **Micro-Architecture**.

Remember this was before the Machine Learning Hype!

"To our knowledge, this paper is the first to propose such a scheduler along with a rigorous methodology to designing self-optimizing DRAM controllers."

What can we learn from this paper?

Thinking in a unconventional way can lead to good results

For that it is important to not only be an expert in one field, one does **need to know about other fields** to get inspiration

In addition to that, one needs the ability to carry out the ideas inspired by these other fields in order for them to form something more than a just dream

Discussion

Questions . . .

Ensuring Correct Operation 1

The scheduler's decisions are **restricted to picking among the set of legal commands** each cycle

Care must be taken to **ensure that the system is guaranteed to make forward progress** regardless of the scheduler's decisions

1. the scheduler is not permitted to select NOPs when other legal commands are available
2. the scheduler cannot choose to activate an arbitrary row with no pending requests
3. the scheduler is not allowed to precharge a newly activated row until it issues a read or write command to it.

Ensuring Correct Operation 2

Starvation

$\Rightarrow$ Timeout policy: any request that has been pending for a fixed (but large - in our case 10,000) number of cycles is completed in its entirety before other commands can be considered for scheduling

DRAM refresh

we do not allow the RL controller to dictate the refresh schedule. Instead, at the end of a refresh interval, the RL scheduler is disabled, the appropriate rows are refreshed, and then control is returned to the RL scheduler

Critique From the ATLAS Paper

”Other scheduling algorithms have been proposed to improve DRAM throughput in single-threaded, multi-threaded, or streaming systems. None of these works consider the existence of multiple competing threads sharing the memory controllers (as happens in a multi-core system).”

Link: ATLAS Paper

Bigger Picture: General issues with Memory

Key issues to tackle:

- Enable reliability at low cost/high capacity
- Reduce energy
- Reduce latency
- Improve bandwidth
- **Reduce waste (capacity, bandwidth, latency)**
- Enable computation close to data

according to: Memory Systems course, Technion, Summer 2018

Difficulties in Optimizing Memory Controllers

1. The controller needs to obey all DRAM timing constraints to provide correct functionality
2. the controller must intelligently prioritize DRAM commands from different memory requests to optimize system performance.

Current memory controllers use relatively simple policies to schedule DRAM accesses.

Machine Learning and its Limits

Study of computer programs and algorithms that learn about their environment and improve automatically with experience

TODO: PICTURE FUNCTION (on blackboard ?)

Applicability of RL to DRAM Scheduling

To apply RL to DRAM scheduling an agent is defined.
The agent interacts with its environment over a discrete set of time steps.
At each step, the agent senses the current state of its environment, and executes an action.
This results in a change in the state of the environment (which the agent can sense in the next time step), and produces an immediate reward.
The agent's goal is to maximize its long-term cumulative reward by learning an optimal policy that maps states to actions

Design Paradigms for the Agent

Temporal credit assignment The agent needs to learn how to assign credit and blame to past actions for each observed immediate reward

Exploration vs. exploitation Too little exploration of the environment can cause the agent to commit to sub-optimal policies early on, whereas excessive exploration can result in long periods during which the agent executes sub-optimal actions to explore its environment.

Generalization it is exceedingly improbable for the agent to experience the same state more than once over its lifetime

State-Attributes (not simplified)

1. Number of reads (load/store misses) in the transaction queue.
2. Number of writes (writebacks) in the transaction queue.
3. Number of reads in the transaction queue that are load misses.
4. If the command is related to a load miss by core C in the transaction queue, the load's order in C's dynamic instruction stream relative to other loads by C with requests in the transaction queue.
5. Number of writes in the transaction queue waiting for the row referenced by the command under consideration.
6. Number of load misses in the transaction queue waiting for the row referenced by the command under consideration which have the oldest sequence number among all load misses in the transaction queue from their respective cores.

Experimental Setup

Processor Parameters	
Frequency	4.0 GHz
Number of cores	4
Number of SMT Contexts	2 per core
Fetch/issue/commit width	4/4/4
Int/FP/Ld/St/Br Units	2/2/2/2/2
Int/FP Multipliers	1/1
Int/FP issue queue size	32/32 entries
ROB (reorder buffer) entries	96
Int/FP registers	96 / 96
Ld/St queue entries	24/24
Max. unresolved br.	24
Br. mispred. penalty	9 cycles min.
Br. predictor	Alpha 21264 (tournament)
RAS entries	32
BTB size	512 entries, direct-mapped
iL1/dL1 size	32 kB
iL1/dL1 block size	32B/32B
iL1/dL1 round-trip latency	2/3 cycles (uncontended)
iL1/dL1 ports	1 / 2
iL1/dL1 MSHR entries	16/16
iL1/dL1 associativity	direct-mapped/4-way
Memory Disambiguation	Perfect
Coherence protocol	MESI
Consistency model	Release consistency

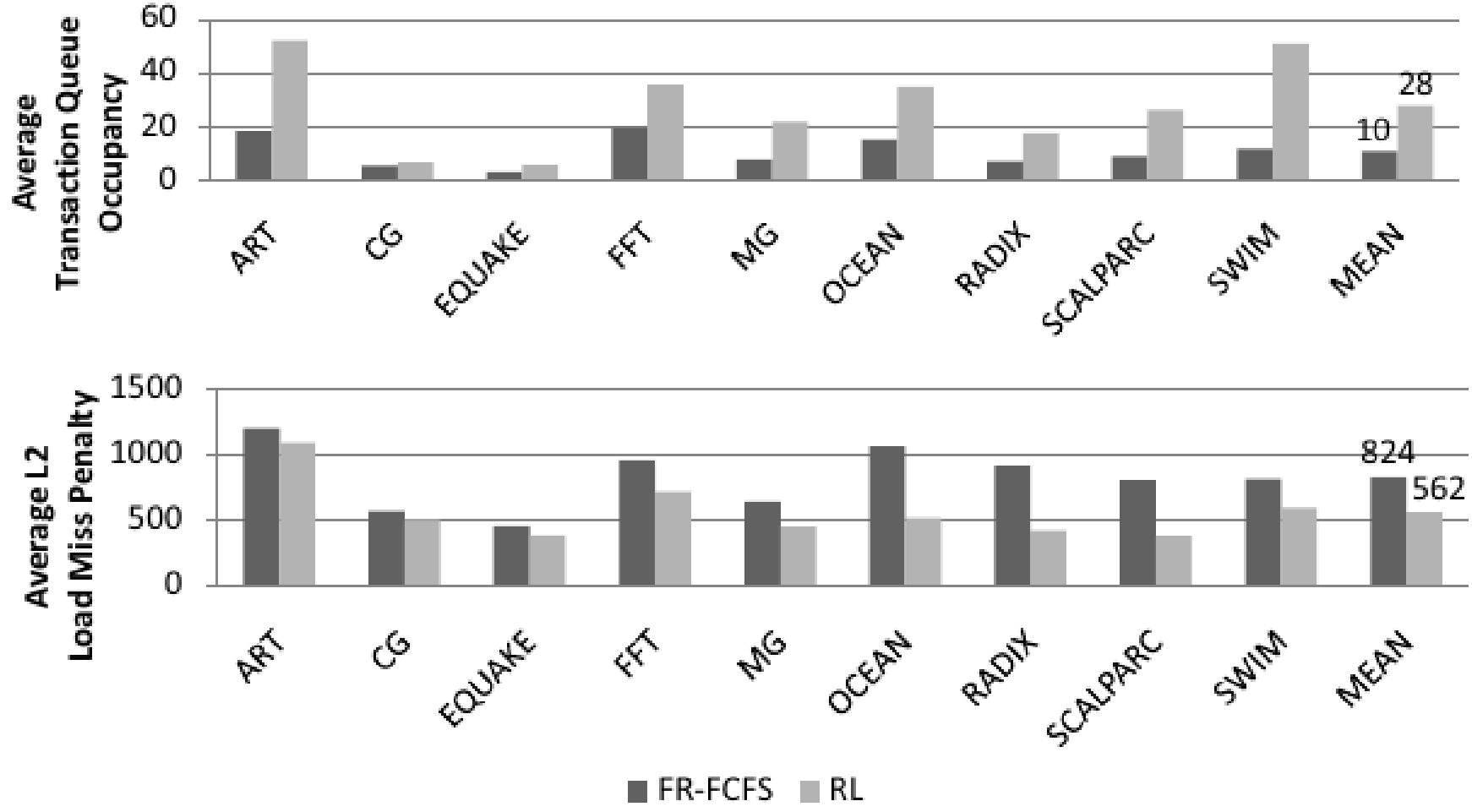
Experimental Setup

Shared L2 Cache Subsystem	
Shared L2 Cache	4MB, 64B block, 8-way
L2 MSHR entries	64
L2 round-trip latency	32 cycles (uncontended)
Write buffer	64 entries
DDR2-800 SDRAM Subsystem [26]	
Transaction Queue	64 entries
Peak Data Rate	6.4GB/s
DRAM bus frequency	400 MHz
Number of Channels	1, 2, 4
DIMM Configuration	Single rank
Number of Chips	4 DRAM chips per rank
Number of Banks	4 per DRAM chip
Row Buffer Size	2KB
Address Mapping	Page Interleaving
Row Policy	Open Page
tRCD	5 DRAM cycles
tCL	5 DRAM cycles
tWL	4 DRAM cycles
tCCD	4 DRAM cycles
tWTR	3 DRAM cycles
tWR	6 DRAM cycles
tRTP	3 DRAM cycles
tRP	5 DRAM cycles
tRRD	3 DRAM cycles
tRAS	18 DRAM cycles
tRC	22 DRAM cycles
Burst Length	8

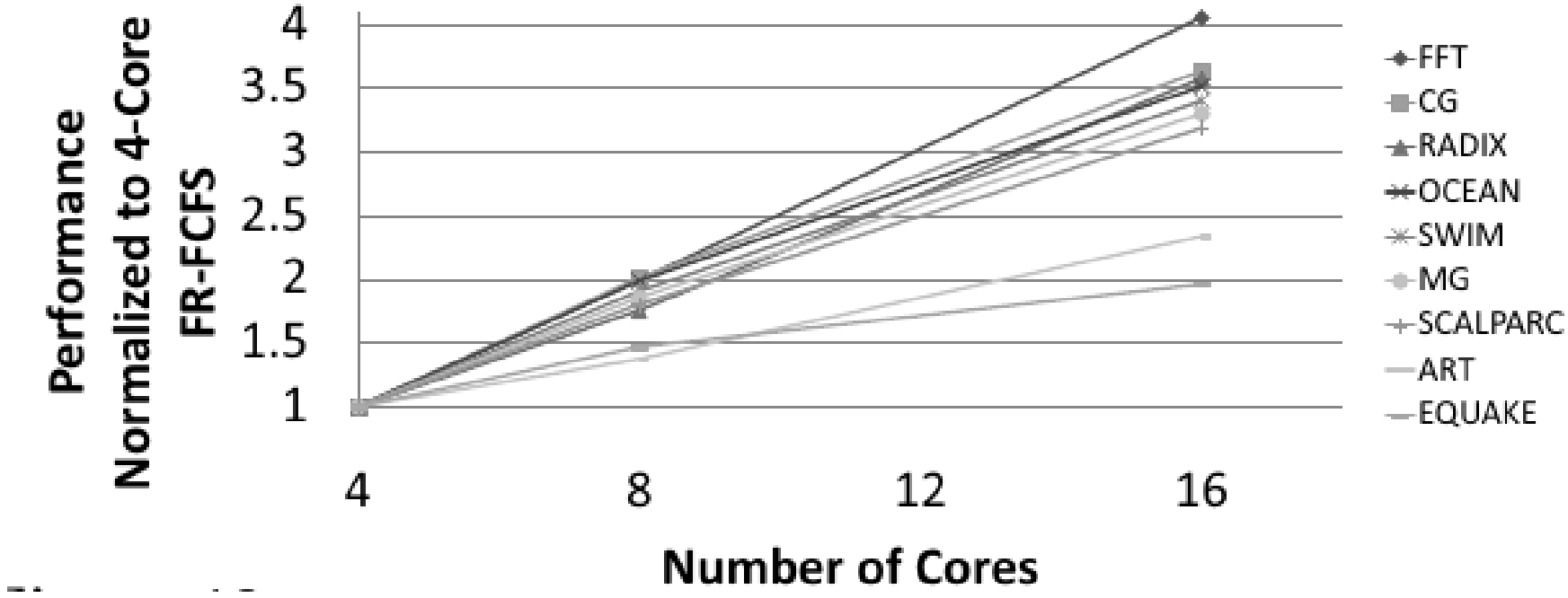
Experimental Setup

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

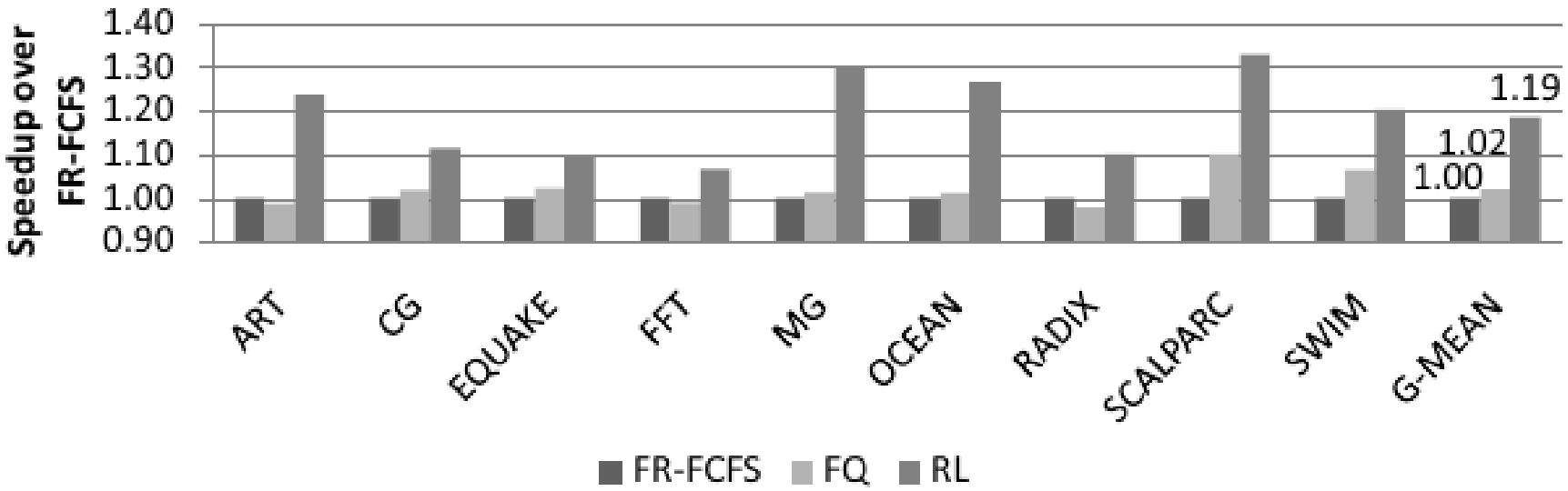
Transaction Queue and L2 Miss Penalties



More than two Cores?



Speedup compared to Fair Queuing



Finding the right Paramters