MICRO 2012

MorphCore

An Energy-Efficient Microarchitecture for High Performance ILP and High Throughput TLP

Khubaib Milad Hashemi Yale N. Patt

M. Aater Suleman Chris Wilkerson

Presented by Lukas Fluri

Executive summary

- **Problem:** Modern workloads require a microarchitecture with good single- and multi-threaded performance while not wasting any energy. Current cores do not provide this as they are specialized on the execution of one of those workload-types.
- **MorphCore:** microarchitecture based on a big out-of-order core with the ability to switch to higly parallel in-order SMT execution mode
- **Results:** MorphCore
  - Performs very close to the best single-thread optimized core on single-threaded workloads
  - Achieves 2/3 of the performance improvement of the best optimized multi-threaded architecture on multi-threaded workloads
  - Performs best on average over all workloads compared to the other measured core architectures
  - Achieves the performance improvements with significally less energy than other cores

Outline

- **Background, Problem and Goal**
- **Novelty, Key approach and Ideas**
- **Mechanisms (in some detail)**
- **Key Results: Methodology and Evaluation**
- **Summary**
- **Strengths**
- **Weaknesses**
- **Takeaways**
- **Thougts, Ideas and Discussion starters**

2 important concepts for this paper

Out-of-order execution

Simultaneous Multithreading

Out-of-order execution (OOO)

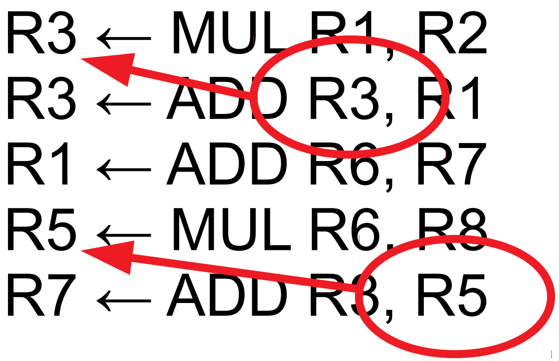
In-order execution

F	D	E	E	E	E	R	W							
	F	D	-	-	-	E	R	W						
		F	-	-	-	D	E	R	W					
						F	D	E	E	E	E	R	W	
							F	D	E	R	-	-	-	W

Out-of-order execution

F	D	E	E	E	E	R	W				
	F	D	-	-	-	E	R	W			
		F	D	E	R	-	-	-	W		
			F	D	E	E	E	E	R	W	
				F	D	-	-	-	E	R	W

Program to execute:

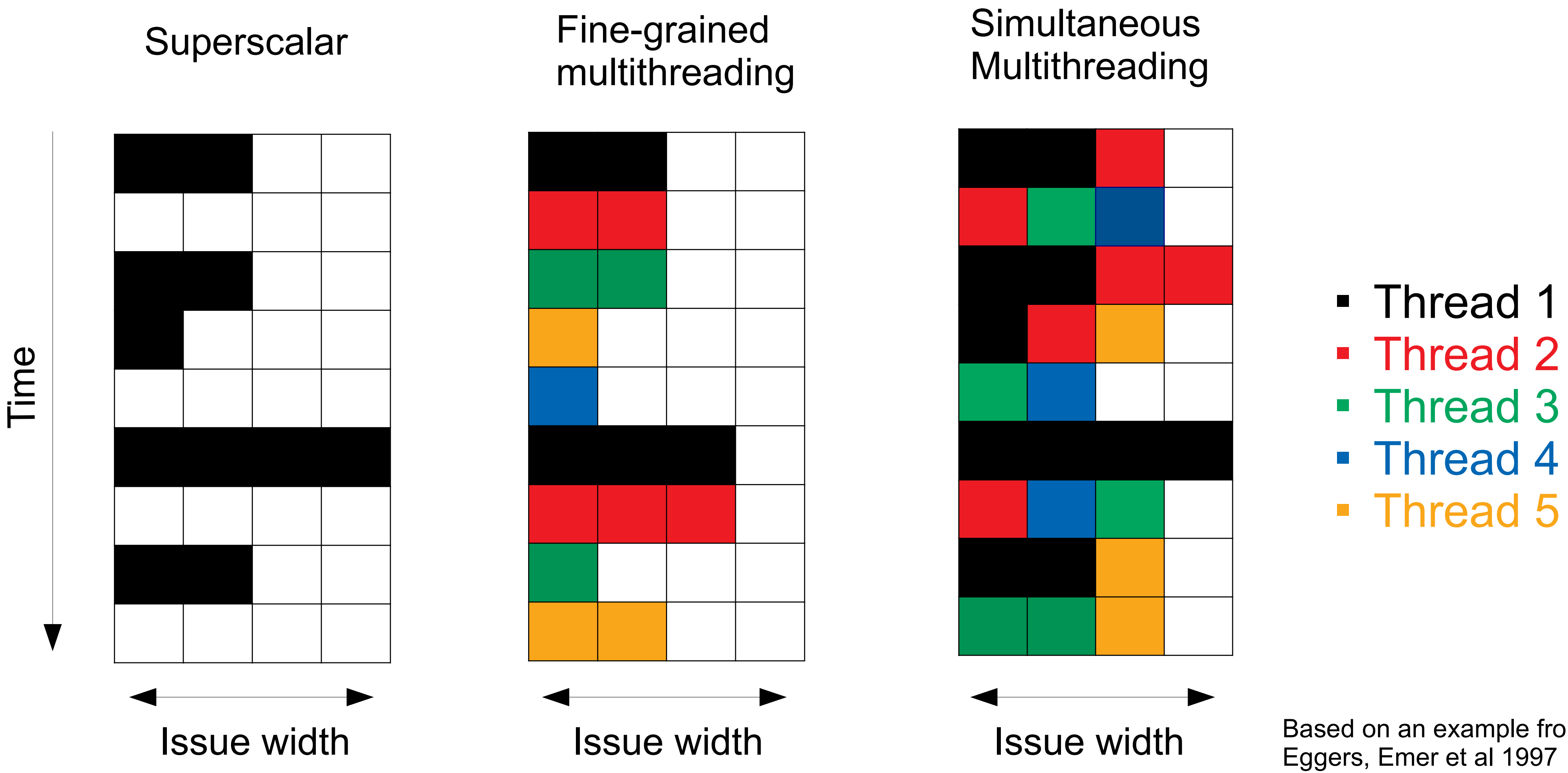


R3 ← MUL R1, R2
R3 ← ADD R3, R1
R1 ← ADD R6, R7
R5 ← MUL R6, R8
R7 ← ADD R3, R5

Dependencies!

Based on an example from: Onur Mutlu, Course ‘Design of Digital Circuits’ 2017

Simultaneous Multithreading (SMT)



Based on an example from Eggers, Emer et al 1997

Industry builds 2 types of cores

Large out-of-order cores

- Exploit Instruction-Level-Parallelism (ILP)

+ High single thread performance

- Power-inefficient for multi-threaded programs

Small cores

- Exploit Thread-Level-Parallelism (TLP)

+ High parallel Throughput

- Poor single thread performance

Problem

Modern workloads require a micro-architecture capable of both delivering good single and multi-threaded performance.

➡ Currently only possible with a big OOO-core that wastes huge amounts of energy on multi-threaded workloads.

Early approach: ACMP

Asymmetric Chip Multiprocessor

- One or few large cores for fast single-threaded execution
- Many small cores for high throughput in multi-threaded execution

- Numbers of cores fixed at design time, can’t adapt dynamically to workload

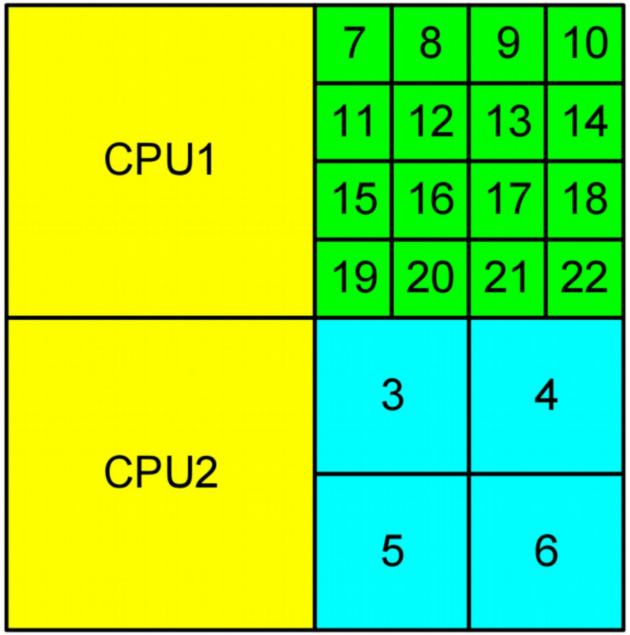


Image: Morad,Weiser et al 2005

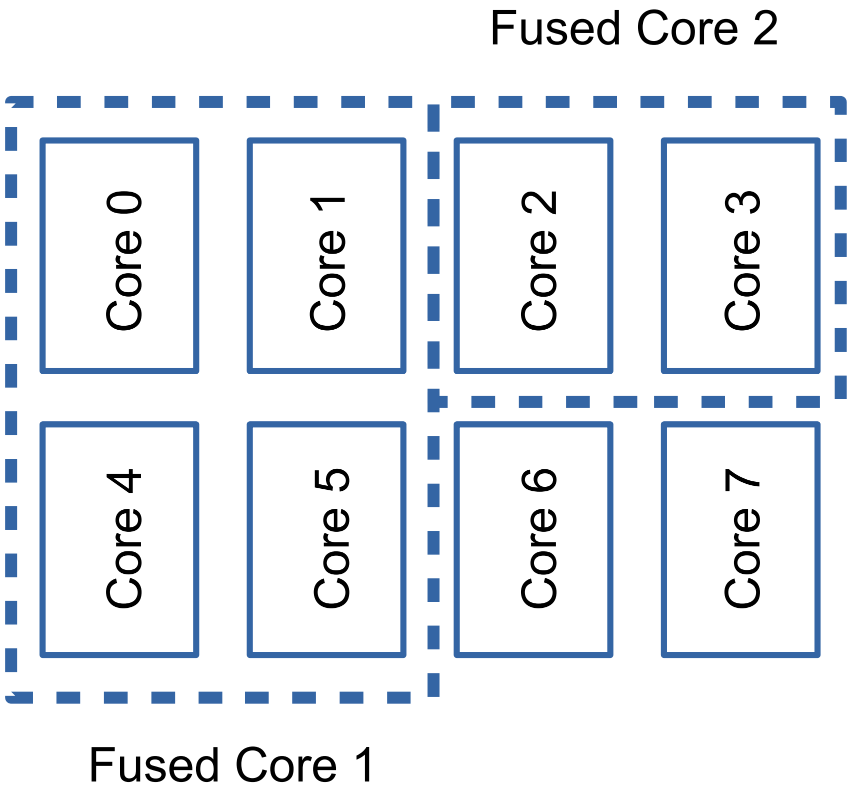
Recent approach: Core Fusion

Core Fusion

- Many small cores for high throughput in multi-threaded execution
- Ability to dynamically fuse into larger cores when executing single-threaded code

+ Can dynamically adapt to workload

- Fused cores have low performance and high power/energy consumption



Goal

Propose a Core architecture that:

- Can adapt to its workload
- Provides high performance in single-threaded execution
- Provides high parallel throughput in multi-threaded execution
- Uses no more energy/power than necessary

MorphCore

Outline

- Background, Problem and Goal
- **Novelty, Key approach and Ideas**
- Mechanisms (in some detail)
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- Weaknesses
- Takeaways
- Thougts, Ideas and Discussion starters

Key insight 1

A highly threaded in-order core can achieve the same or better performance as an out-of-order core. (While using much less energy)

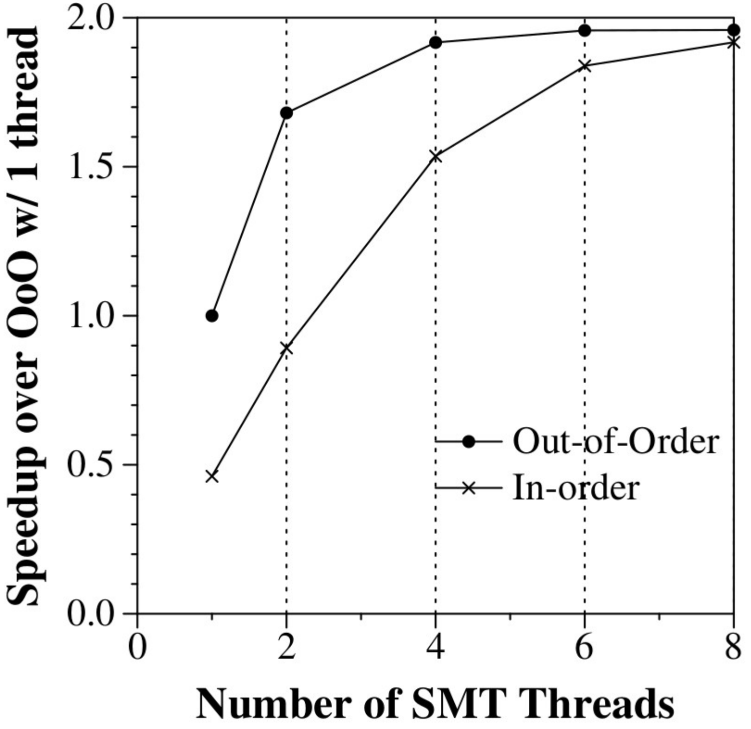


Image source: Khubaib, Suleman et al. “MorphCore”, 2012

Key insight 2

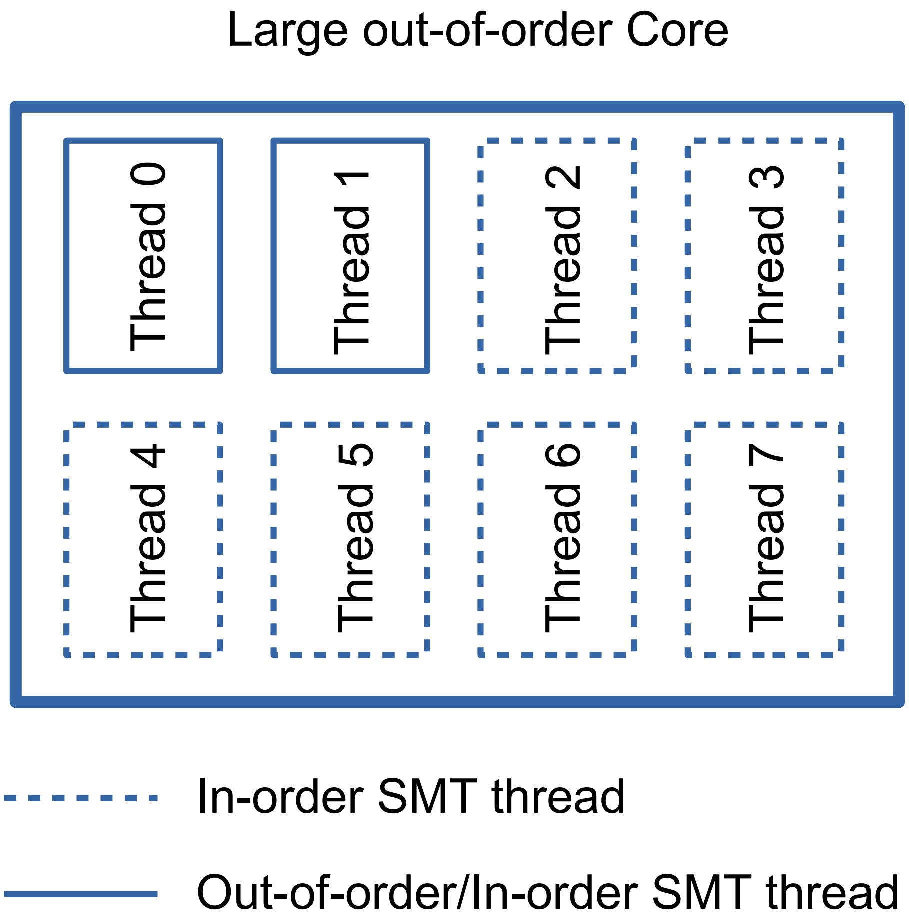
Such a core can be built using almost a subset of the hardware required to build an aggressive OOO core.

Idea

- Use a big out-of-order core as base substrate
- Add the capability to switch between out-of-order and highly threaded in-order SMT execution mode
- In the in-order SMT execution mode, turn off power-hungry OOO-structures

MorphCore

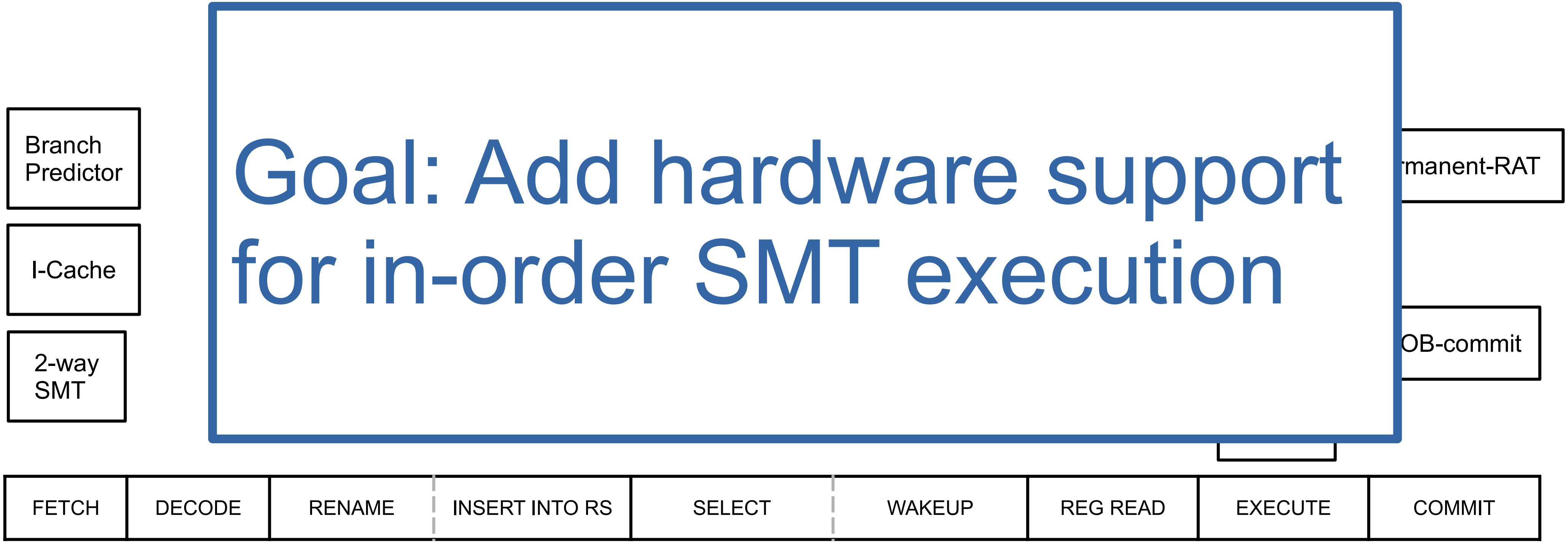
- Can switch between out-of-order and in-order SMT execution mode
  - ➔ Can dynamically adapt to different workloads
- Runs as normal OOO core in single-threaded programs
  - ➔ Provides high performance single-thread execution
- Runs as highly-threaded in-order core in multi-threaded programs
  - ➔ Provides high parallel throughput while not wasting vast amounts of energy



Outline

- Background, Problem and Goal
- Novelty, Key approach and Ideas
- **Mechanisms (in some detail)**
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- Weaknesses
- Takeaways
- Thougts, Ideas and Discussion starters

An out-of-order core microarchitecture



Goal: Add hardware support for in-order SMT execution

Branch Predictor

I-Cache

2-way SMT

rmanent-RAT

OB-commit

FETCH | DECODE | RENAME | INSERT INTO RS | SELECT | WAKEUP | REG READ | EXECUTE | COMMIT

The following slides and images are adapted from: Khubaib, Suleman et al. “MorphCore...”, 2012

Adding in-order SMT support

- Active in in-order Mode only
- Active in both Modes
- Active in out-of-order Mode only

8-way SMT

Branch Predictor

I-Cache

2-way SMT

STQ-alloc

ROB-alloc

Speculative-RAT

LDQ-alloc

RS Free List

in RS)

dep instr in RS)

File (PRF)

re Buffer
Q Lookup

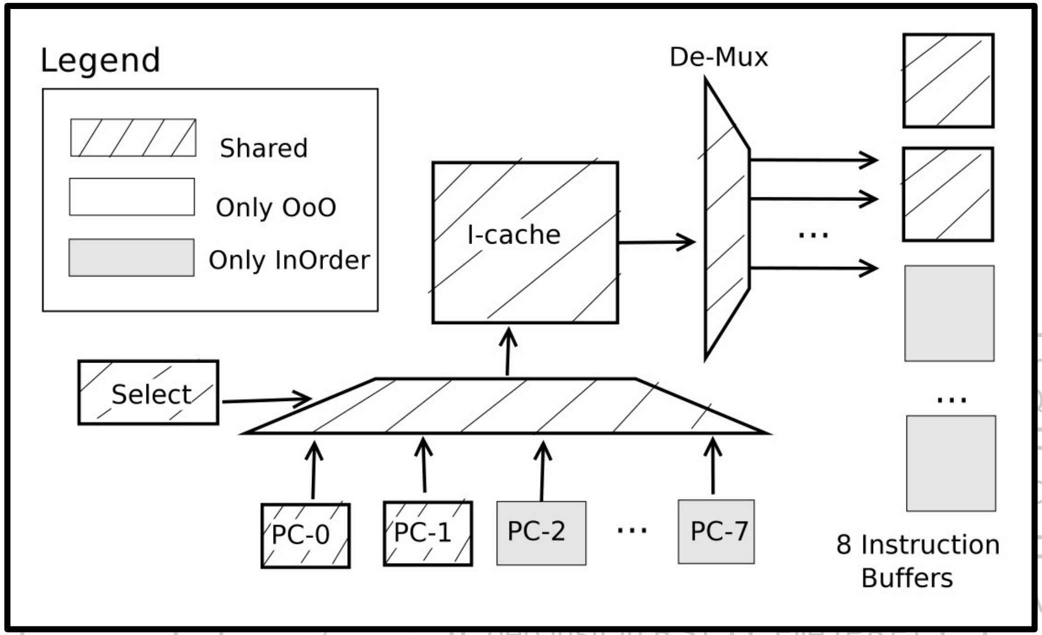
Cache

LUs

ROB-commit

Permanent-RAT

LDQ Lookup



FETCH	DECODE	RENAME	INSERT INTO RS	SELECT	WAKEUP	REG READ	EXECUTE	COMMIT

Adding in-order SMT support

- Active in in-order Mode only
- Active in both Modes
- Active in out-of-order Mode only

8-way SMT

Branch Predictor

I-Cache

2-way SMT

STQ-alloc

ROB-alloc

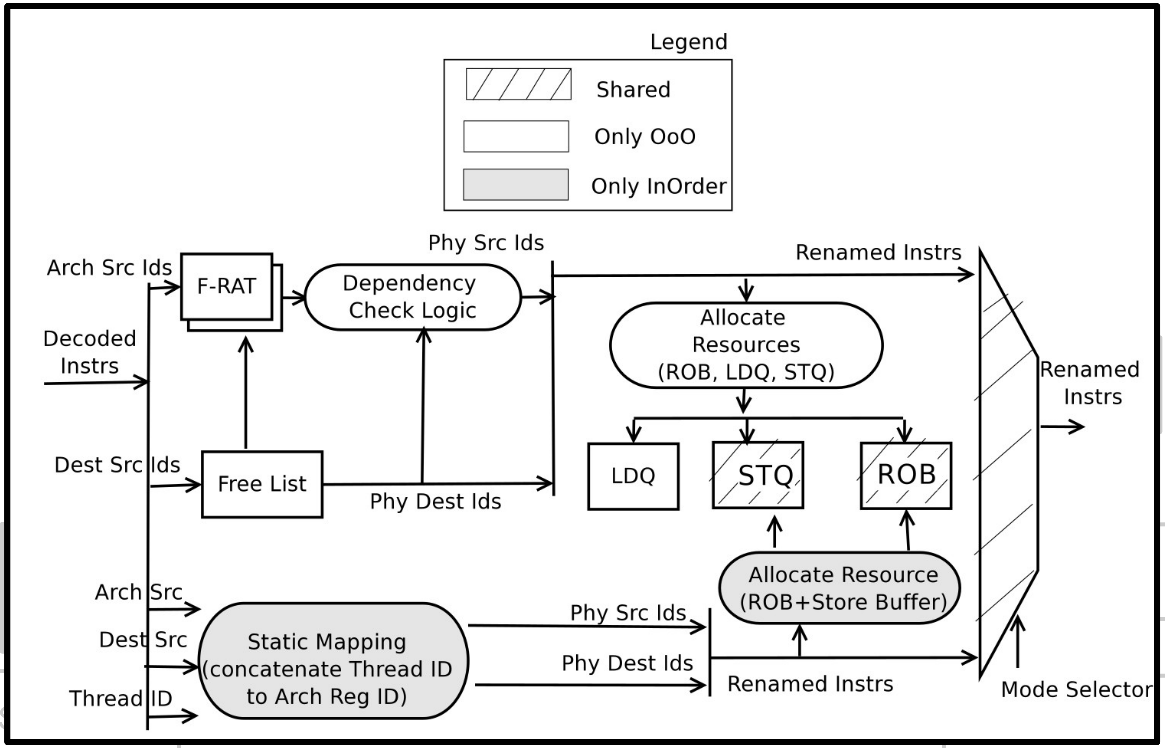
Speculative-RAT

LDQ-alloc

RS

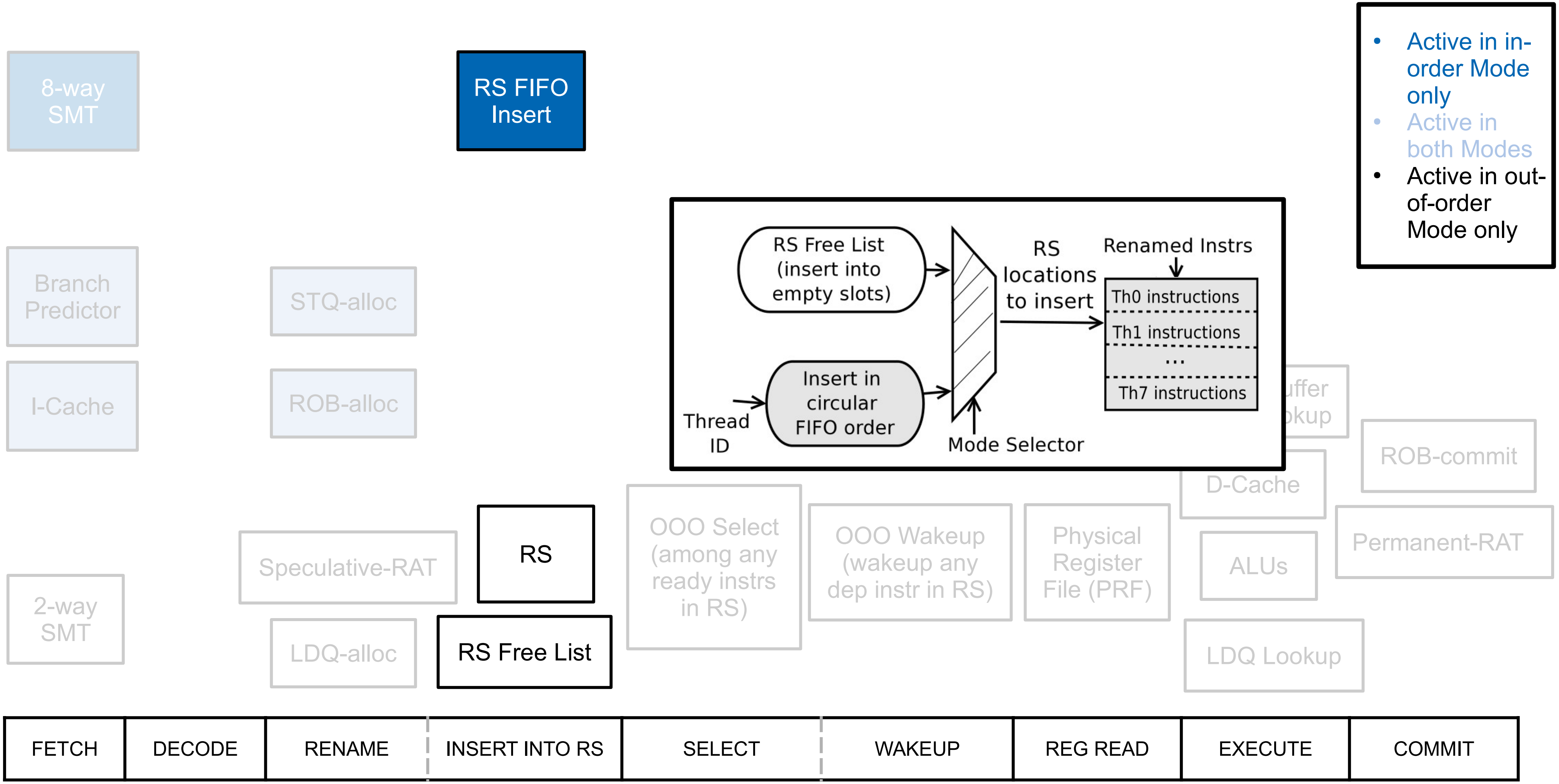
ROB-commit

Permanent-RAT



FETCH	DECODE	RENAME	INSERT INTO RS	SELECT	WAKEUP	REG READ	EXECUTE	COMMIT

Adding in-order SMT support



- Active in in-order Mode only
- Active in both Modes
- Active in out-of-order Mode only

8-way SMT

RS FIFO Insert

RS Free List (insert into empty slots)

Thread ID

Insert in circular FIFO order

RS locations to insert

Mode Selector

Renamed Instrs

Th0 instructions

Th1 instructions

…

Th7 instructions

Branch Predictor

I-Cache

STQ-alloc

ROB-alloc

2-way SMT

Speculative-RAT

LDQ-alloc

RS

RS Free List

OOO Select (among any ready instrs in RS)

OOO Wakeup (wakeup any dep instr in RS)

Physical Register File (PRF)

uffer okup

D-Cache

ALUs

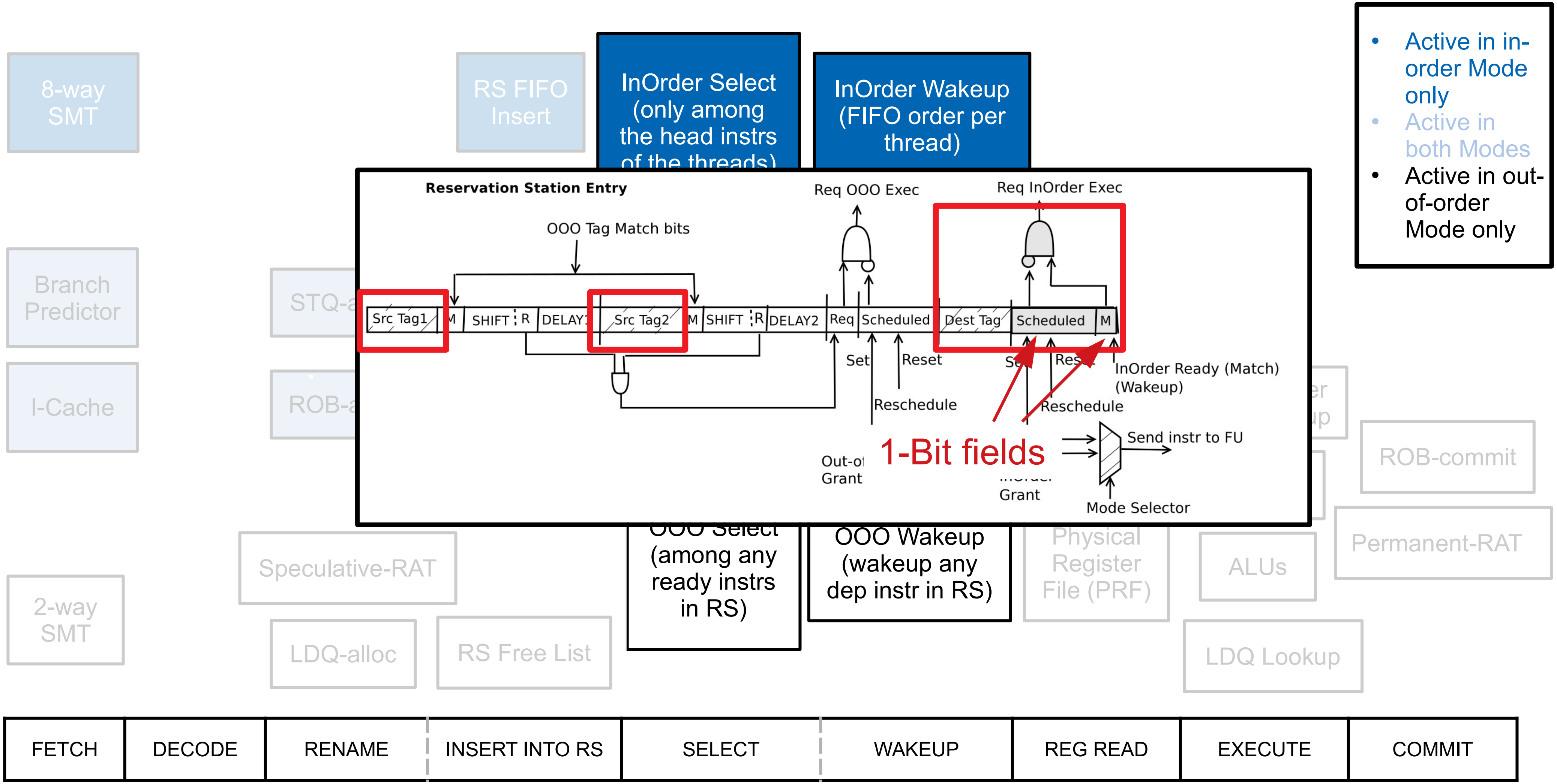
LDQ Lookup

ROB-commit

Permanent-RAT

FETCH | DECODE | RENAME | INSERT INTO RS | SELECT | WAKEUP | REG READ | EXECUTE | COMMIT

Adding in-order SMT support



- Active in in-order Mode only
- Active in both Modes
- Active in out-of-order Mode only

8-way SMT

RS FIFO Insert

InOrder Select (only among the head instrs of the threads)

InOrder Wakeup (FIFO order per thread)

Branch Predictor

I-Cache

STQ-a

ROB-a

OOO Select (among any ready instrs in RS)

OOO Wakeup (wakeup any dep instr in RS)

Physical Register File (PRF)

ALUs

ROB-commit

Permanent-RAT

Speculative-RAT

2-way SMT

LDQ-alloc

RS Free List

LDQ Lookup

er

up

1-Bit fields

FETCH	DECODE	RENAME	INSERT INTO RS	SELECT	WAKEUP	REG READ	EXECUTE	COMMIT

Adding in-order SMT support

8-way SMT

RS FIFO Insert

InOrder Select (only among the head instrs

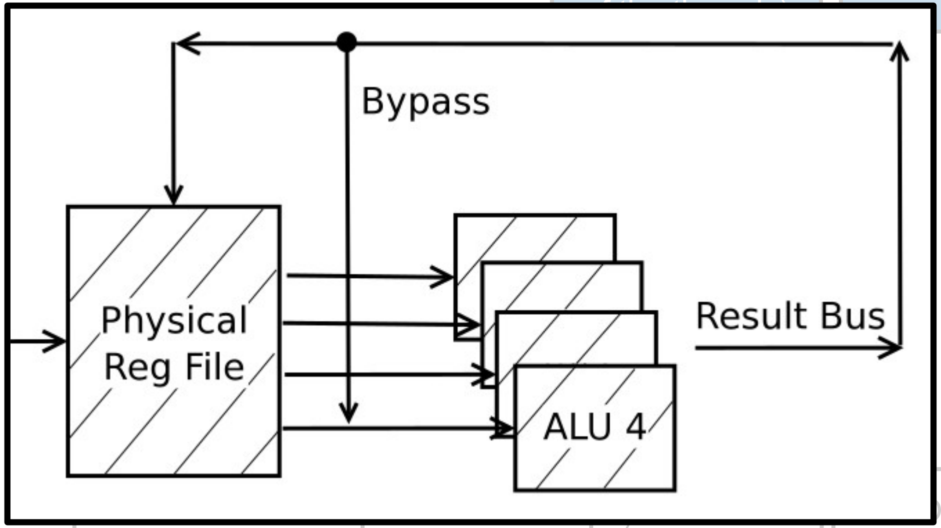
InOrder Wakeup (FIFO order per thread)

Small result buffer

- Active in in-order Mode only
- Active in both Modes
- Active in out-of-order Mode only

Branch Predictor

I-Cache



Speculative-RAT

(among any ready instrs in RS)

Wakeup (wakeup any dep instr in RS)

Physical Register File (PRF)

Store Buffer STQ Lookup

D-Cache

ALUs

LDQ Lookup

ROB-commit

Permanent-RAT

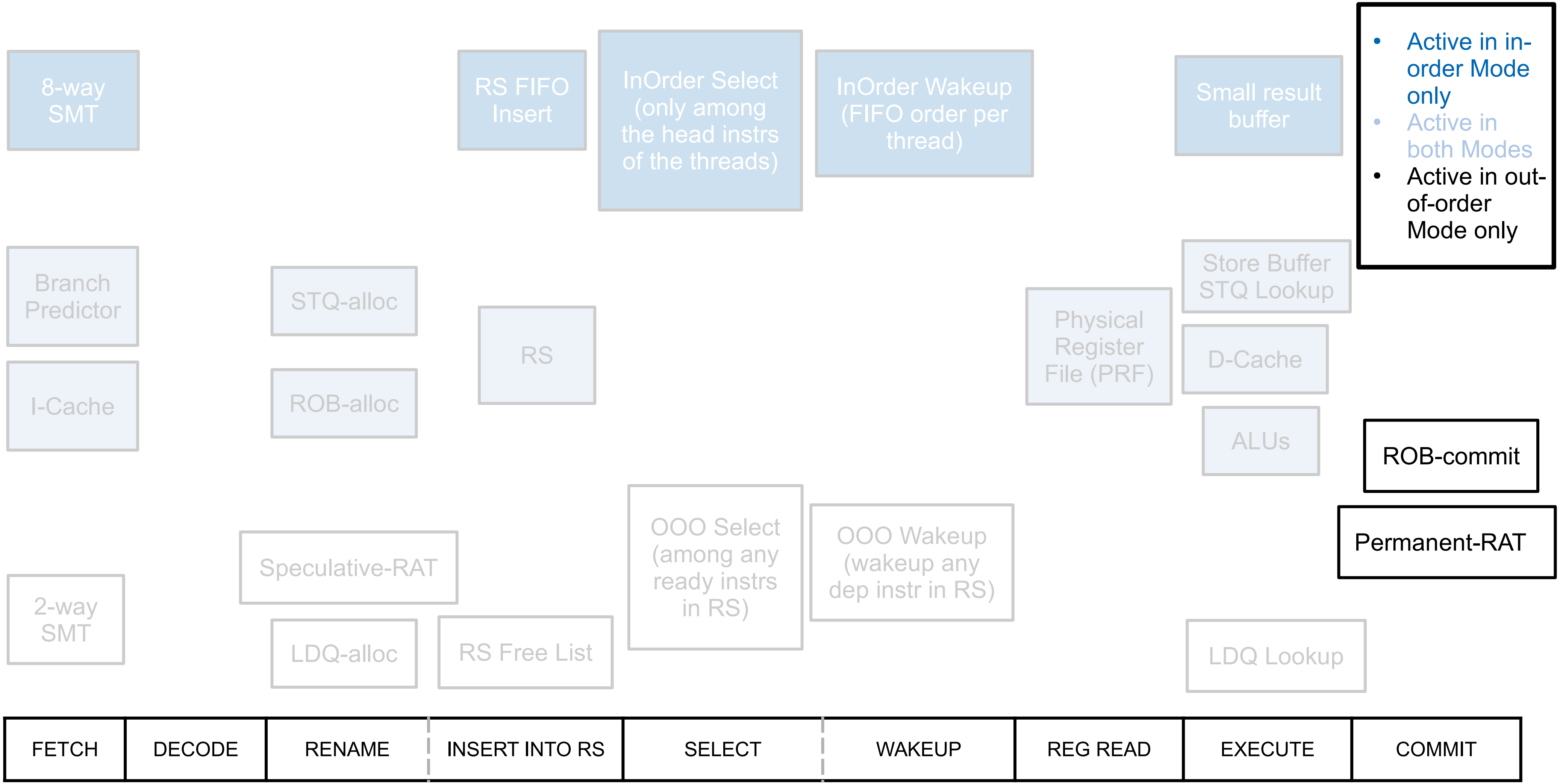
2-way SMT

LDQ-alloc

RS Free List

FETCH	DECODE	RENAME	INSERT INTO RS	SELECT	WAKEUP	REG READ	EXECUTE	COMMIT

Adding in-order SMT support



- Active in in-order Mode only
- Active in both Modes
- Active in out-of-order Mode only

8-way SMT

Branch Predictor

I-Cache

2-way SMT

STQ-alloc

ROB-alloc

Speculative-RAT

LDQ-alloc

RS FIFO Insert

RS

RS Free List

InOrder Select (only among the head instrs of the threads)

OOO Select (among any ready instrs in RS)

InOrder Wakeup (FIFO order per thread)

OOO Wakeup (wakeup any dep instr in RS)

Physical Register File (PRF)

Small result buffer

Store Buffer STQ Lookup

D-Cache

ALUs

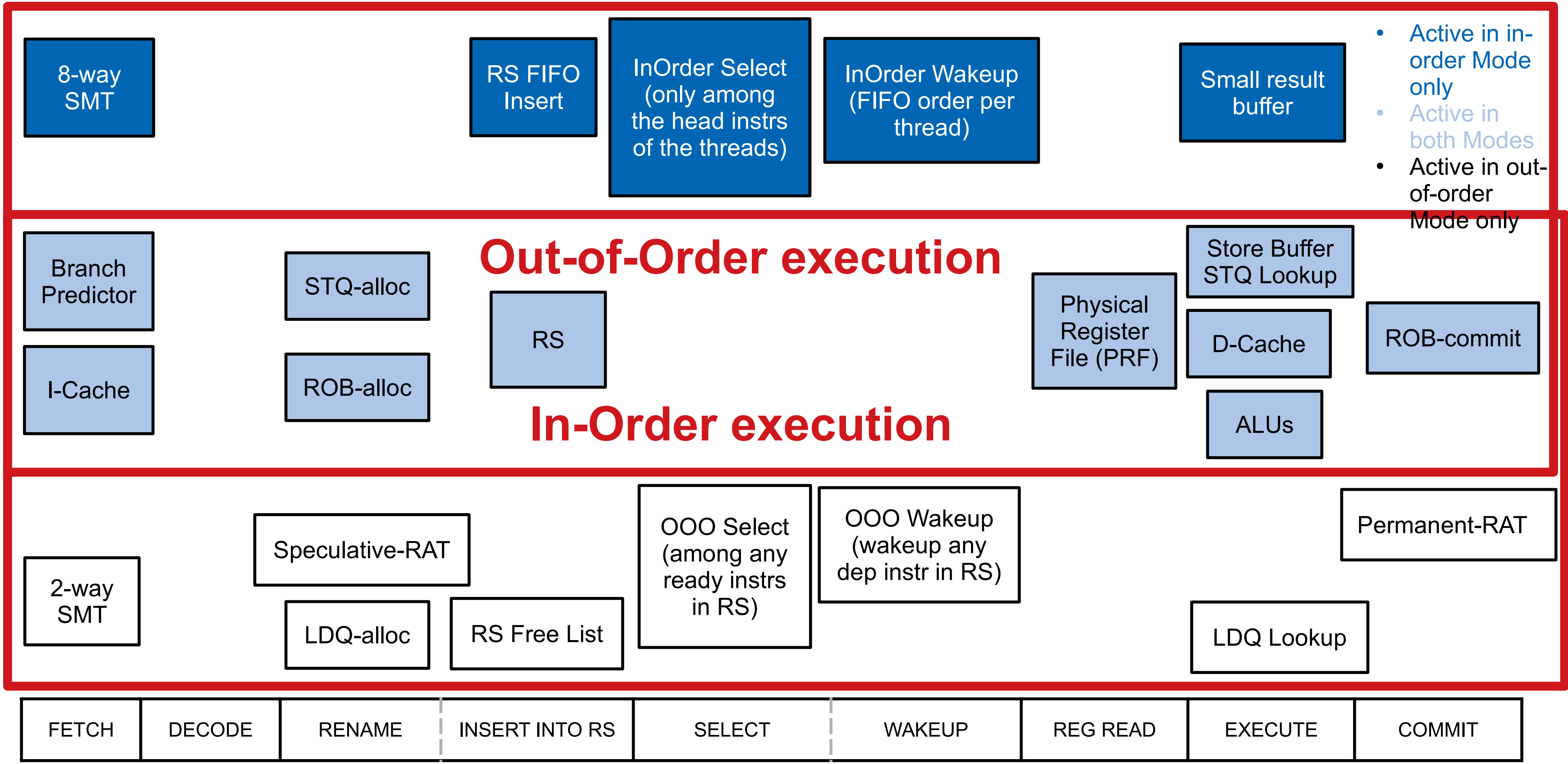
LDQ Lookup

ROB-commit

Permanent-RAT

FETCH	DECODE	RENAME	INSERT INTO RS	SELECT	WAKEUP	REG READ	EXECUTE	COMMIT

Adding in-order SMT support



- Active in in-order Mode only
- Active in both Modes
- Active in out-of-order Mode only

8-way SMT

RS FIFO Insert

InOrder Select (only among the head instrs of the threads)

InOrder Wakeup (FIFO order per thread)

Small result buffer

Out-of-Order execution

In-Order execution

Branch Predictor

I-Cache

STQ-alloc

ROB-alloc

RS

Physical Register File (PRF)

Store Buffer STQ Lookup

D-Cache

ALUs

ROB-commit

2-way SMT

Speculative-RAT

LDQ-alloc

RS Free List

OOO Select (among any ready instrs in RS)

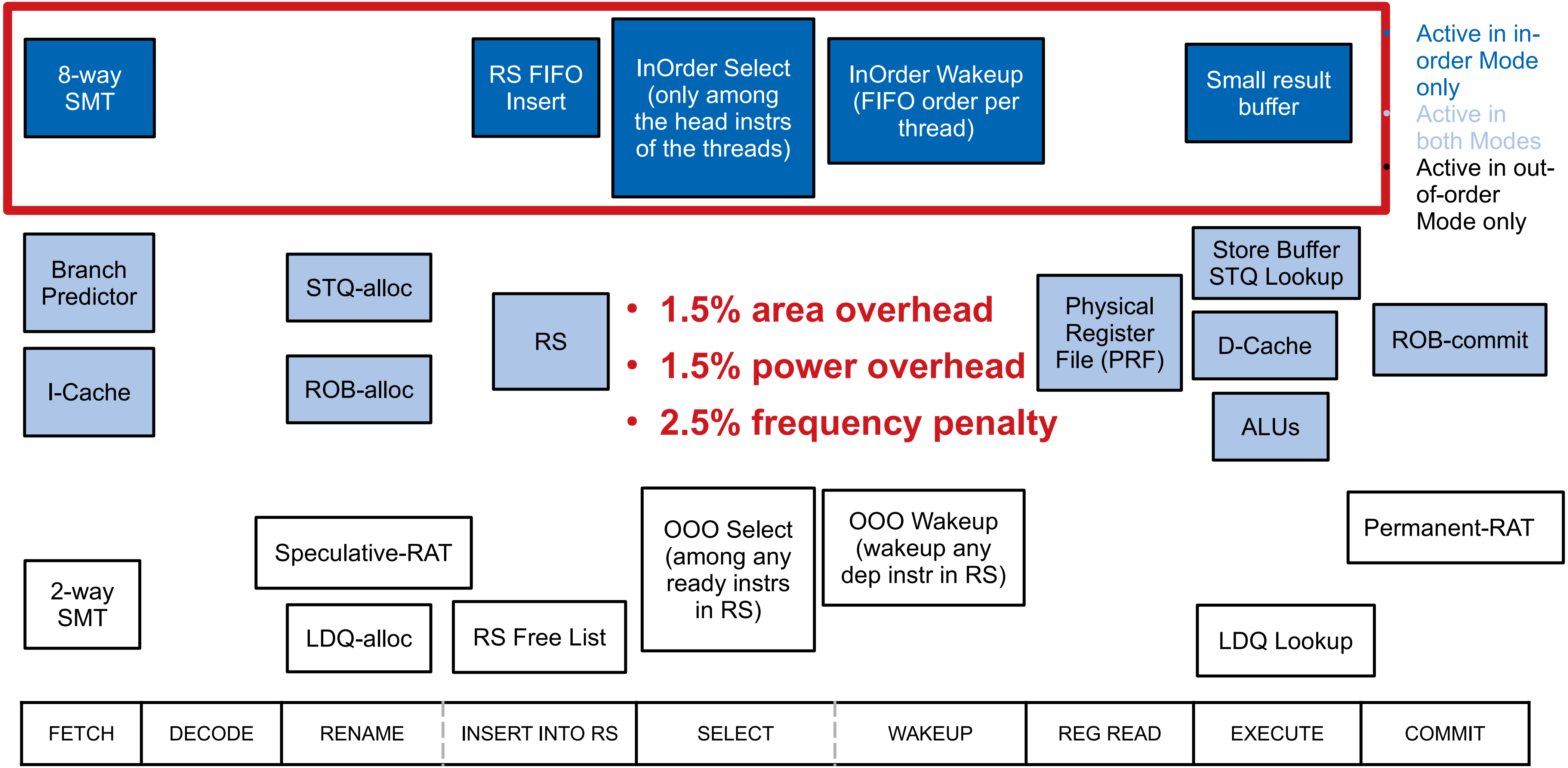
OOO Wakeup (wakeup any dep instr in RS)

Permanent-RAT

LDQ Lookup

FETCH | DECODE | RENAME | INSERT INTO RS | SELECT | WAKEUP | REG READ | EXECUTE | COMMIT

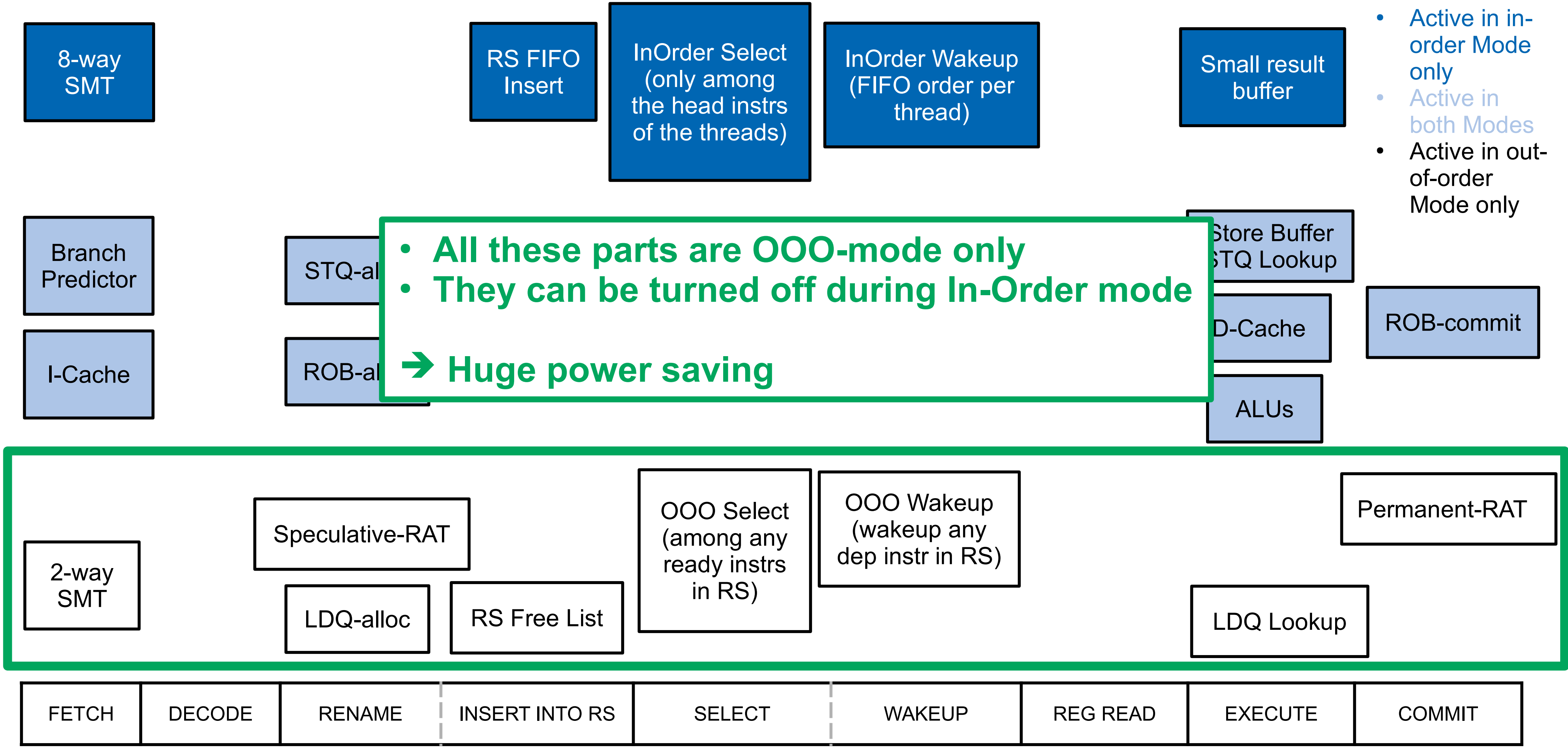
Area, Power & Frequency Overhead



- **1.5% area overhead**
- **1.5% power overhead**
- **2.5% frequency penalty**

- Active in in-order Mode only
- Active in both Modes
- Active in out-of-order Mode only

Area, power & frequency overhead



- Active in in-order Mode only
- Active in both Modes
- Active in out-of-order Mode only

- **All these parts are OOO-mode only**
- **They can be turned off during In-Order mode**

➔ Huge power saving

When to switch between modes?

- Based on number of active threads
- Threshold t = 2
- When # active threads <= t, switch to OOO-mode
- When # active threads > t switch to In-Order-mode
- Uses MONITOR/MWAIT, 2 already existent ISA instructions to get info about waiting threads

➔ No changes to operating systems, compilers or ISAs, and no recompilation of programs necessary!

Switching from OOO to in-order

- Handled by a micro-code routine that performs the following tasks:
  1) Drains the core pipeline
  2) Spills the architectural registers of all threads (into reserved memory regions)
  3) Turns off Renaming unit, OOO-Wakeup and Select blocks and Load Queue (clock-gated)
  4) Fills register values back into each thread’s PRF partitions

Switching from in-order to OOO

- Handled by a micro-code routine that performs the following tasks:
  1) Drains the core pipeline
  2) Spills the architectural registers of all threads. Store pointers to the architectural state of the inactive threads in the Active Thread Table
  3) Turns on Renaming unit, OOO-Wakeup and Select blocks and Load Queue
  4) Fills the architectural registers of only the active threads into pre-determined locations in PRF, and updates the speculative- and permanent RAT

Overhead of changing the mode

Two main contributors to overhead:

- Draining of the pipeline (dependent on instructions still in pipeline)
- Spilling of architectural register state of the threads (~250 cycles)

Outline

- Background, Problem and Goal
- Novelty, Key approach and Ideas
- Mechanisms (in some detail)
- **Key Results: Methodology and Evaluation**
- Summary
- Strengths
- Weaknesses
- Takeaways
- Thougts, Ideas and Discussion starters

The cores

Core	Type	Freq (Ghz)	Issue-width	Num of cores	SMT threads per core	Total Threads	Total Norm . Area	Peak ST throughput	Peak MT throughput
OOO-2	OOO	3.4	4	1	2	2	1	4 ops/cycle	4 ops/cycle
OOO-4	OOO	3.23	4	1	[illegible]	[illegible]	[illegible]	4 ops/cycle	4 ops/cycle
MED	OOO	3.4	2	3	[illegible]	[illegible]	[illegible]	2 ops/cycle	6 ops/cycle
SMALL	in-order	3.4	2	3	2	6	0.97	2 ops/cycle	6 ops/cycle
MorphCore	OOO or In-order	3.315	4	1	OOO: 2 In-order: 8	2 or 8	1.015	4 ops/cycle	4 ops/cycle

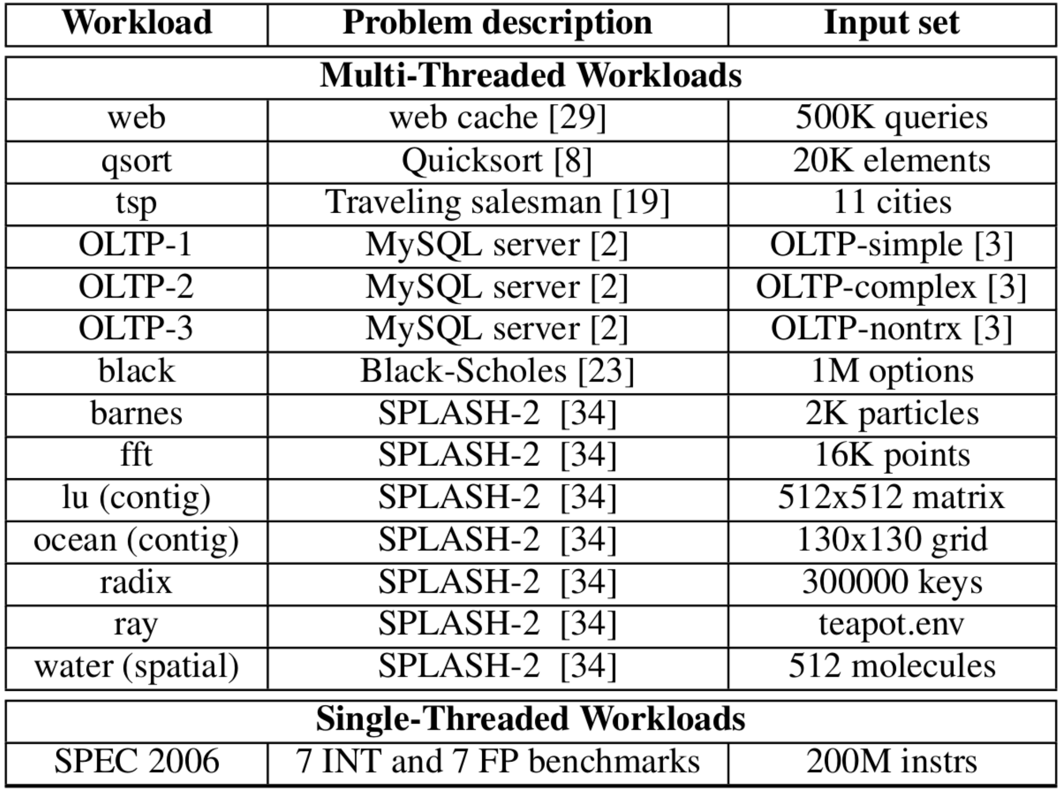
highest
aded
nce

Expected close to best in both, ST and MT

Adapted from: Khubaib, Suleman et al. "MorphCore...", 2012

The workloads

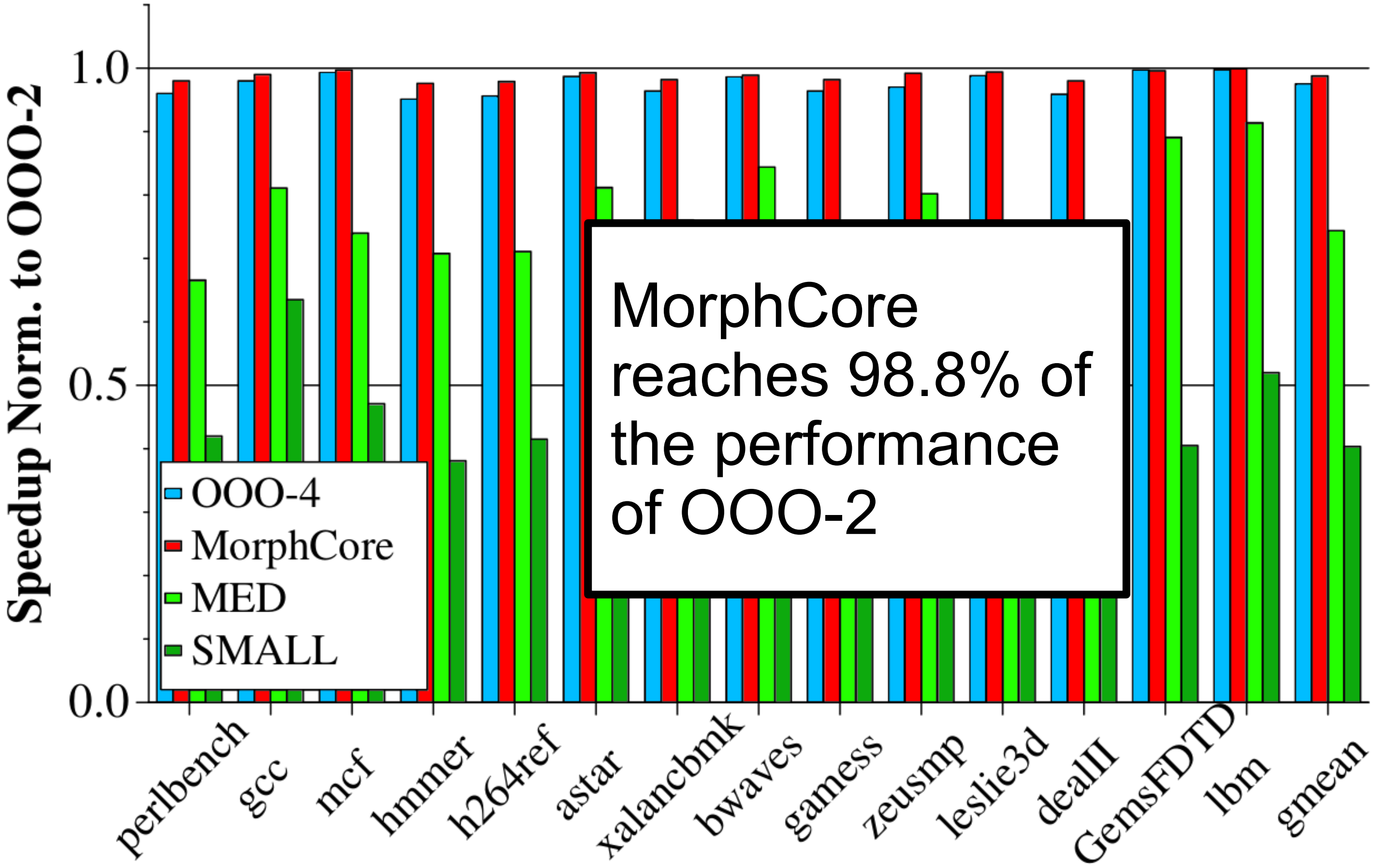
- 14 single-thread and 14 multi-threaded workloads



Workload	Problem description	Input set
Multi-Threaded Workloads		
web	web cache [29]	500K queries
qsort	Quicksort [8]	20K elements
tsp	Traveling salesman [19]	11 cities
OLTP-1	MySQL server [2]	OLTP-simple [3]
OLTP-2	MySQL server [2]	OLTP-complex [3]
OLTP-3	MySQL server [2]	OLTP-nontrx [3]
black	Black-Scholes [23]	1M options
barnes	SPLASH-2 [34]	2K particles
fft	SPLASH-2 [34]	16K points
lu (contig)	SPLASH-2 [34]	512x512 matrix
ocean (contig)	SPLASH-2 [34]	130x130 grid
radix	SPLASH-2 [34]	300000 keys
ray	SPLASH-2 [34]	teapot.env
water (spatial)	SPLASH-2 [34]	512 molecules
Single-Threaded Workloads		
SPEC 2006	7 INT and 7 FP benchmarks	200M instrs

Image source: Khubaib, Suleman et al. “MorphCore...”, 2012

Result: Single-thread workloads



MorphCore reaches 98.8% of the performance of OOO-2

Image source: Khubaib, Suleman et al. “MorphCore...”, 2012

Result: Multi-threaded workloads

- MorphCore reaches a 22% perf. Improvement over OOO-2
- Stays behind MED and SMALL (30% and 33% improv.)
- But beats MED in three workloads
- Gets beaten by OOO-4 three times

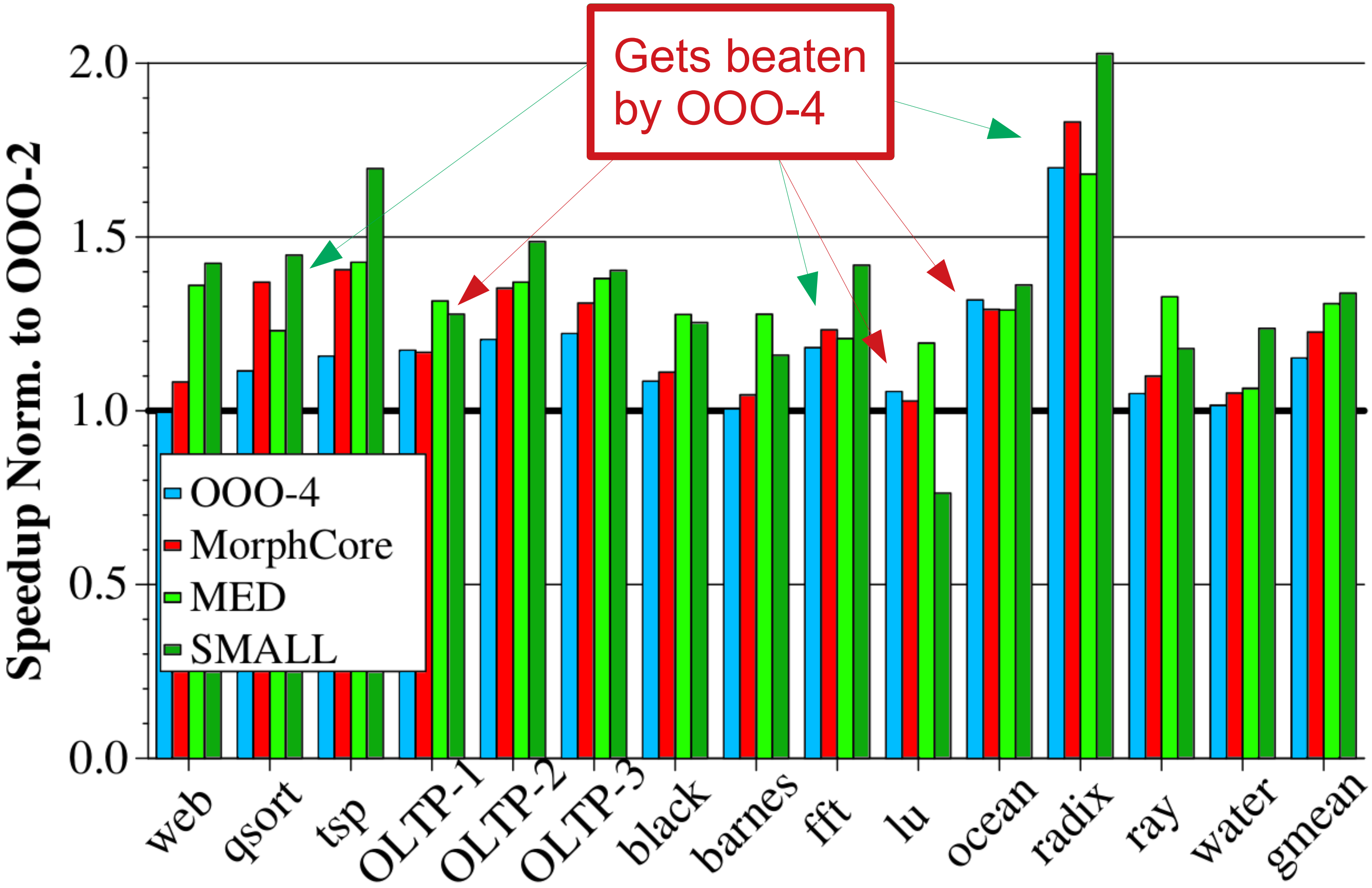
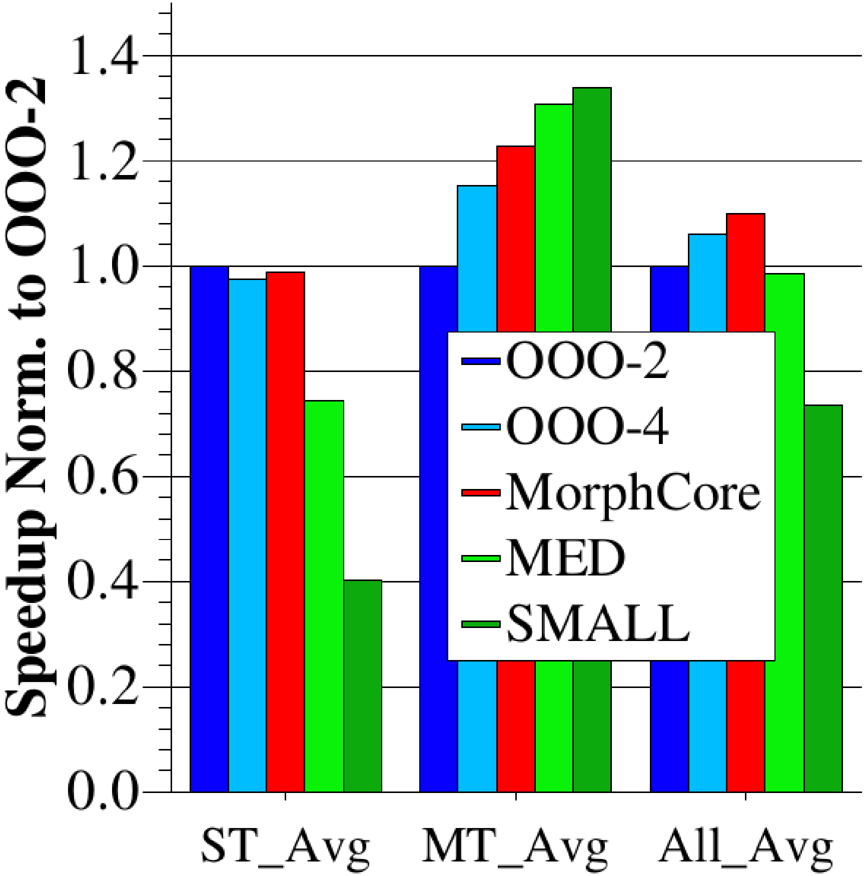


Image source: Khubaib, Suleman et al. “MorphCore...”, 2012

Speedup summary



On average, MorphCore outperforms all other cores

Image source: Khubaib, Suleman et al. “MorphCore...”, 2012

Result: Power & Energy

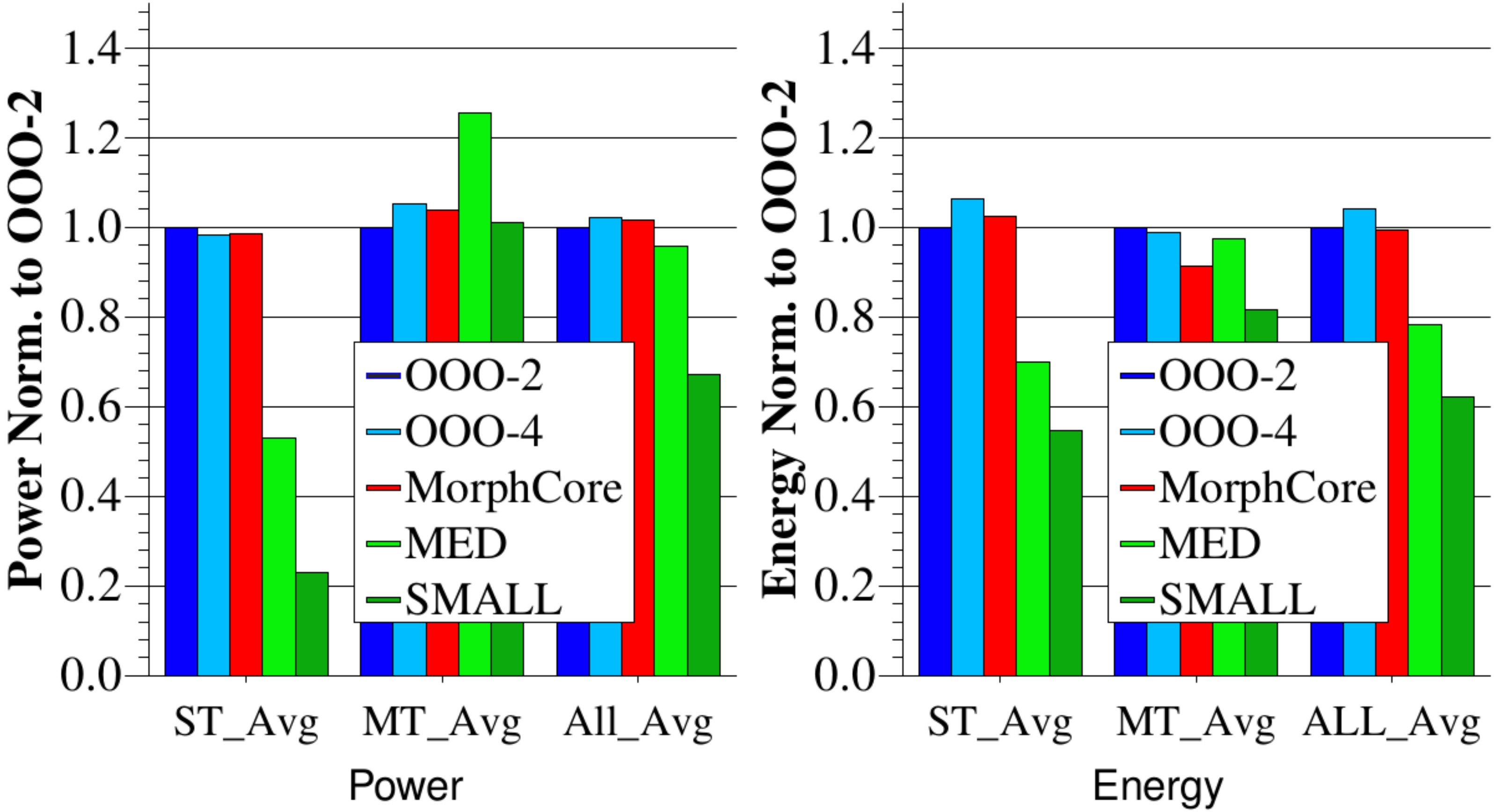
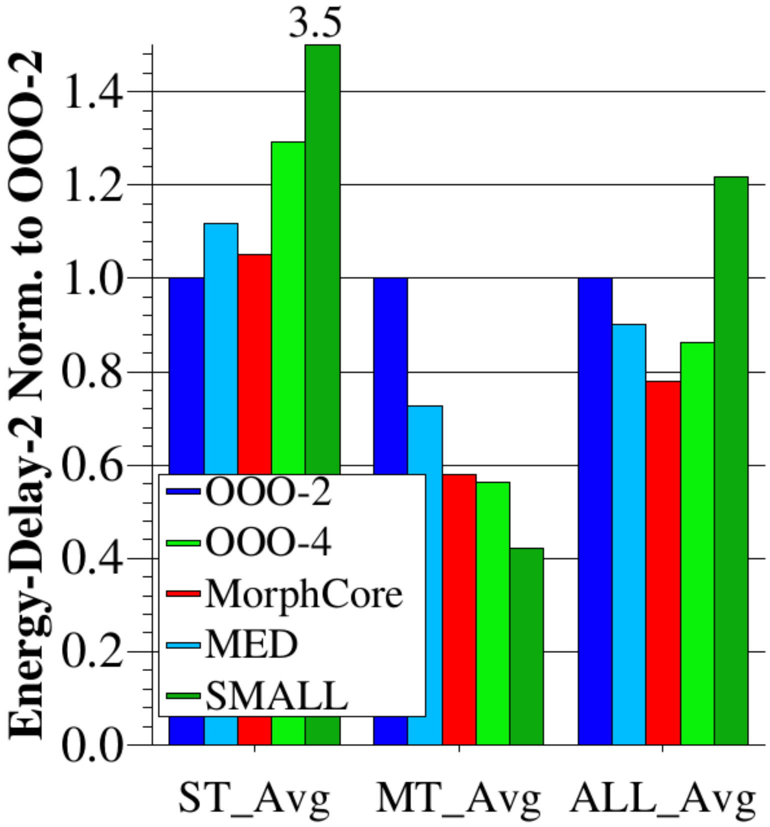


Image source: Khubaib, Suleman et al. “MorphCore...”, 2012

Overall result

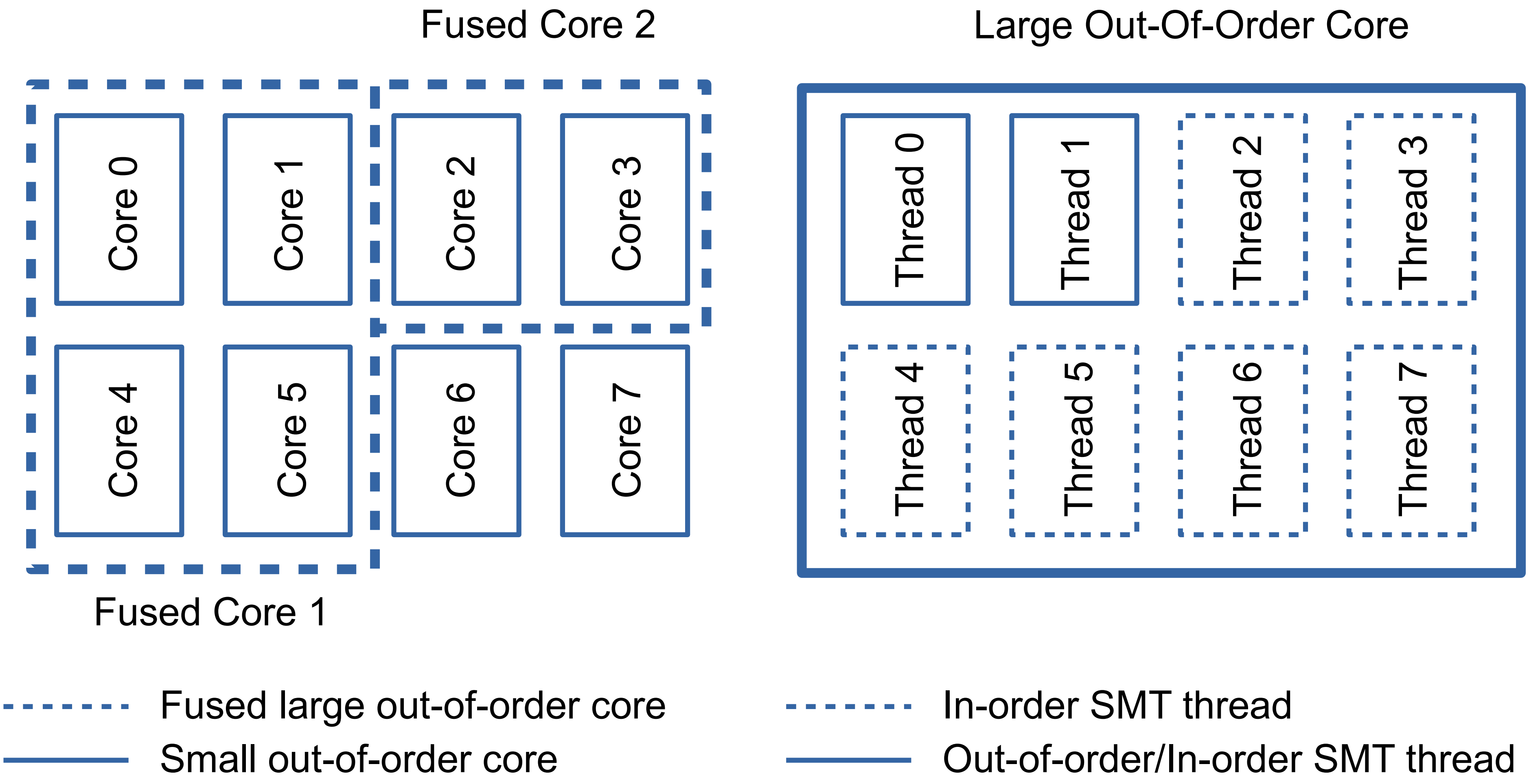
MorphCore has the lowest ED^2 being 22% lower than the baseline OOO-2



Energy-Delay-Squared (lower is better)

Image source: Khubaib, Suleman et al. “MorphCore...”, 2012

Comparison to CoreFusion



Fused Core 2

Large Out-Of-Order Core

Core 0

Core 1

Core 2

Core 3

Core 4

Core 5

Core 6

Core 7

Thread 0

Thread 1

Thread 2

Thread 3

Thread 4

Thread 5

Thread 6

Thread 7

Fused Core 1

╌╌╌ Fused large out-of-order core

─── Small out-of-order core

╌╌╌ In-order SMT thread

─── Out-of-order/In-order SMT thread

Comparison to CoreFusion

- CoreFusion is better in multi-threaded workloads (8% on aver.)
- MorphCore outperforms CoreFusion in general (5% on aver.)
- Reduces power (19%), energy (29%) and ED^2 (29%) significantly compared to CoreFusion

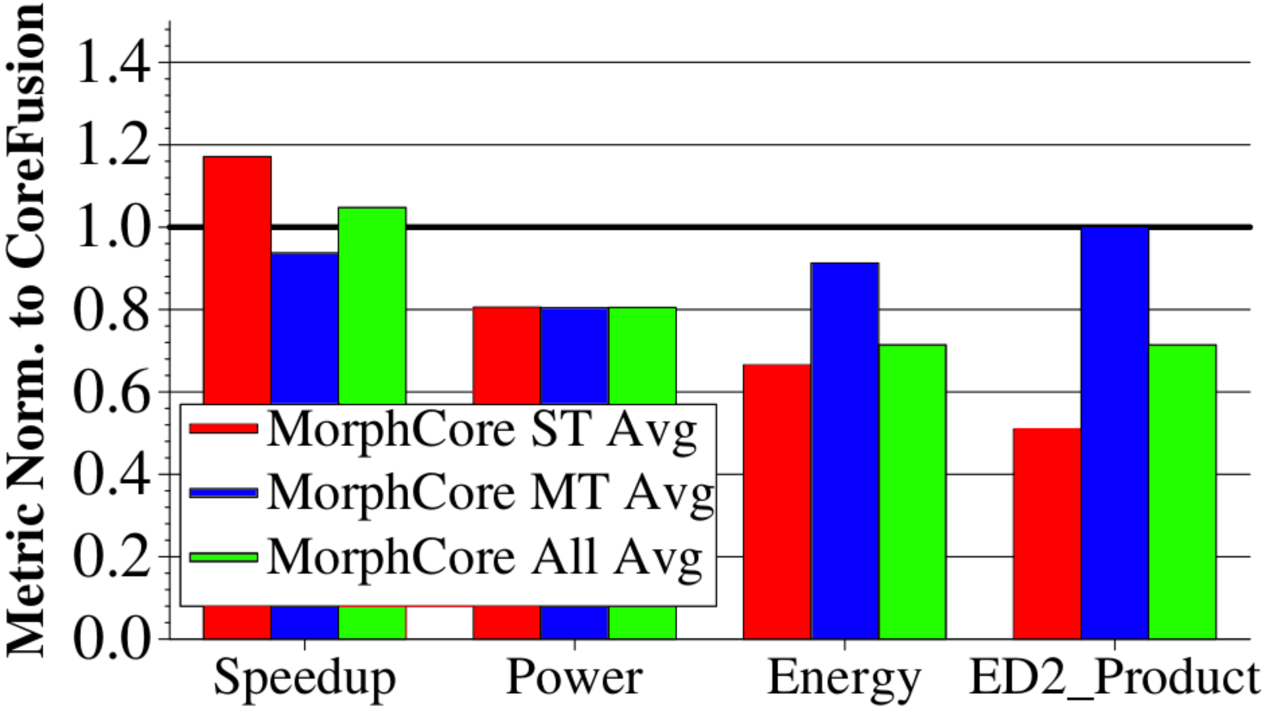


Image source: Khubaib, Suleman et al. “MorphCore...”, 2012

Outline

- Background, Problem and Goal
- Novelty, Key approach and Ideas
- Mechanisms (in some detail)
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- Weaknesses
- Takeaways
- Thougts, Ideas and Discussion starters

Executive summary

- **Problem:** Modern workloads require a microarchitecture with good single- and multi-threaded performance while not wasting any energy. Current cores do not provide this as they are specialized on the execution of one of those workload-types.
- **MorphCore:** microarchitecture based on a big out-of-order core with the ability to switch to higly parallel in-order SMT execution mode
- **Results:** MorphCore
  - Performs very close to the best single-thread optimized core on single-threaded workloads
  - Achieves 2/3 of the performance improvement of the best optimized multi-threaded architecture on multi-threaded workloads
  - Performs best on average over all workloads compared to the other measured core architectures
  - Achieves the performance improvements with significally less energy than other cores

Outline

- Background, Problem and Goal
- Novelty, Key approach and Ideas
- Mechanisms (in some detail)
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- Weaknesses
- Takeaways
- Thougts, Ideas and Discussion starters

Strengths

- Novel but simple and elegant solution
- Low hardware overhead and low frequency penalty (1.5% & 2.5%)
- Does not need changes to software, compilers or OS; ISA remains unchanged
- Solves many of the issues of CoreFusion
- Well structured paper

Outline

- Background, Problem and Goal
- Novelty, Key approach and Ideas
- Mechanisms (in some detail)
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- **Weaknesses**
- Takeaways
- Thougts, Ideas and Discussion starters

Weaknesses

- Performance on MT-workloads is better than on other OOO-cores but still weak compared to small cores (only ~2/3 of performance)
- Mode switching policy may cause big performance overhead
- No predictable overhead of the mode switching
- Paper sometimes lacks some details

Outline

- Background, Problem and Goal
- Novelty, Key approach and Ideas
- Mechanisms (in some detail)
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- Weaknesses
- **Takeaways**
- Thougts, Ideas and Discussion starters

Takeaways

- A new michroarchitecture that can handle both, single- and multi-threaded workloads, while delivering good performance and not wasting energy
- No changes to software necessary
- Well structured paper, sometimes a bit lack of detail
- Possibility of further improvement and extensions

Outline

- Background, Problem and Goal
- Novelty, Key approach and Ideas
- Mechanisms (in some detail)
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- Weaknesses
- Takeaways
- Thougts, Ideas and Discussion starters

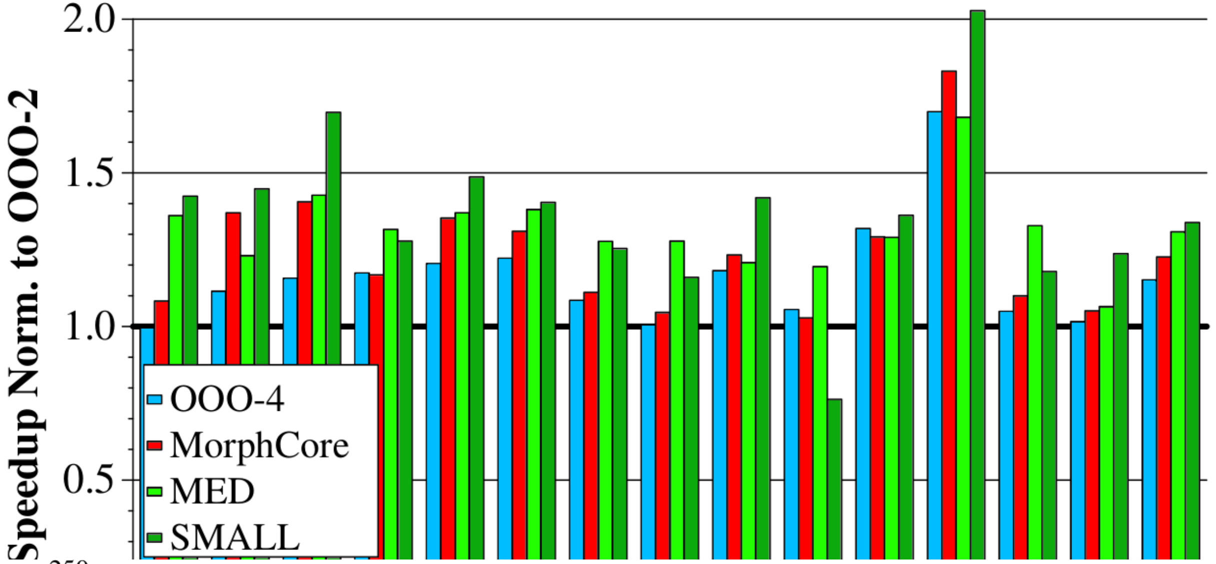
Thoughts, Ideas and Discussion starters

- Increase issue-width. Can this approach achieve a higher total peak throughput and tackle the performance gap on MT workloads between MorphCore and SMALL/MED?

  ➔ Yes, see Khubaib Ph.D. Dissertation 2014

Increase issue-width

- Increased width yields better performance
- At least almost (see lu)
- Comes at cost of higher energy cost



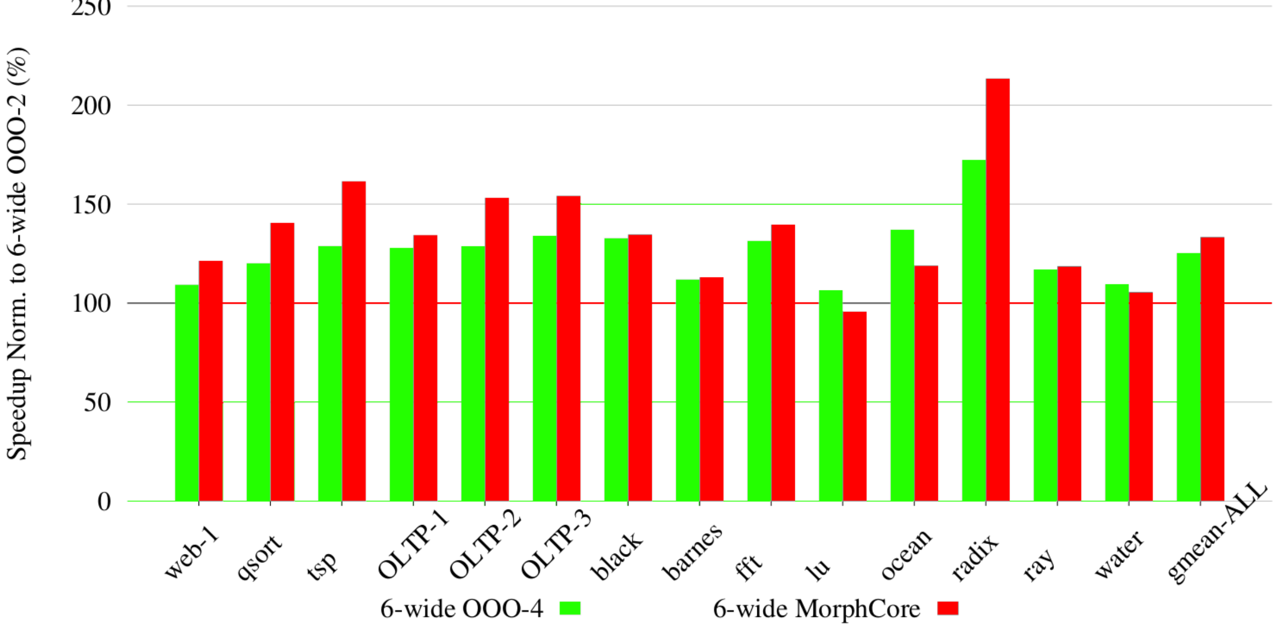


Image source: Khubaib Ph.D. thesis, 2014

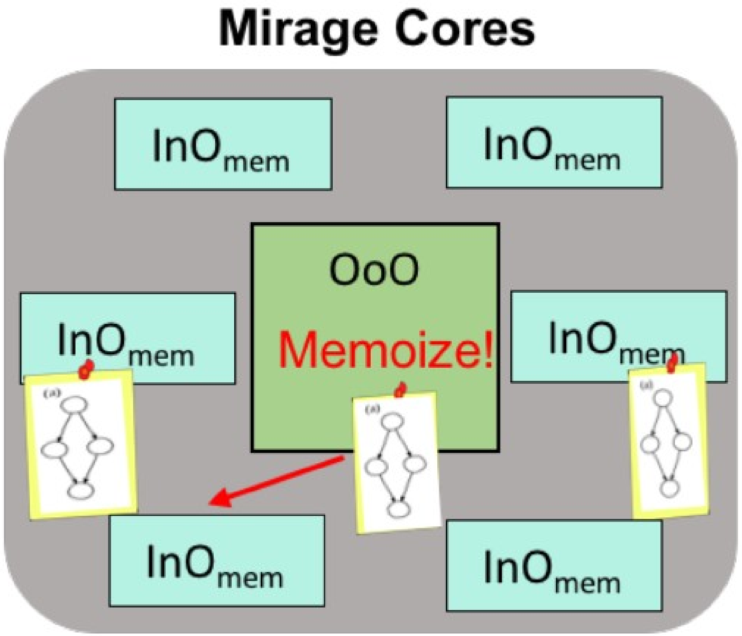
Thoughts, Ideas and Discussion starters

- Increase issue-width. Can this approach achieve a higher total peak throughput and tackle the performance gap on MT workloads between MorphCore and SMALL/MED?
  - ➔ Yes, see Khubaib Ph.D. Dissertation 2014
- Is the concept of MorphCore the only approach to the problem of providing good single- and multi-threaded performance while not wasting energy?
  - ➔ No, see Shruti Padmanabha et al. “Mirage Cores...” IEEE/ACM 2017

Mirage Cores

- Use few OOO cores to analyze the execution of a program
- Instruction schedules of parts that repeat often (e.g. loops) get saved ("memoized")
- All further executions of these parts get executed on the in-order cores

➔ In-order cores performe nearly as good as the big out-of-order cores but use less energy



- High system throughput
- Shorter execution latency

Image source: Shruti Padmanabha et al. "Mirage Cores...", 2017

Thoughts, Ideas and Discussion starters

- Increase issue-width. Can this approach achieve a higher total peak throughput and tackle the performance gap on MT workloads between MorphCore and SMALL/MED?
  - ➔ Yes, see Khubaib Ph.D. Dissertation 2014
- Is the concept of MorphCore the only approach to the problem of providing good single- and multi-threaded performance while not wasting energy?
  - ➔ No, see Shruti Padmanabha et al. “Mirage Cores...” MICRO 2017
- Fetch in each cycle from several threads instead of fetching several instructions from one thread each cycle. Can this improve SMT performance?
- Gather statistics about thread behaviour to achieve smarter mode-switching (similar to branch prediction). Is this a good approach?
- Does a frequent switch of modes lead to cache trashing?

MICRO 2012

MorphCore

An Energy-Efficient Microarchitecture for High Performance ILP and High Throughput TLP

Khubaib Milad Hashemi Yale N. Patt

M. Aater Suleman Chris Wilkerson

Presented by Lukas Fluri