

# Federation: Repurposing Scalar Cores for Out-of-Order Execution

Michael Boyer, David Tarjan, Kevin Skadron  
University of Virginia  
boyer@cs.virginia.edu



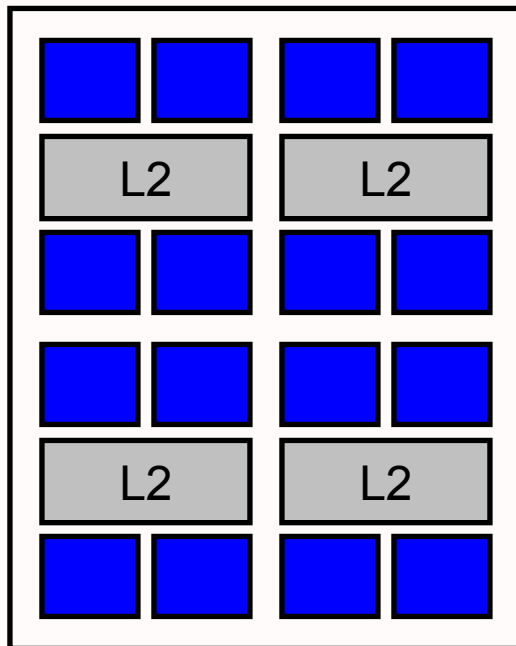
# Outline

- Motivation
- Overview of Federation
- Architectural details:
  - Issue queue (IQ)
  - Load-store queue (LSQ)
- Performance, power, and area impact
- Conclusion

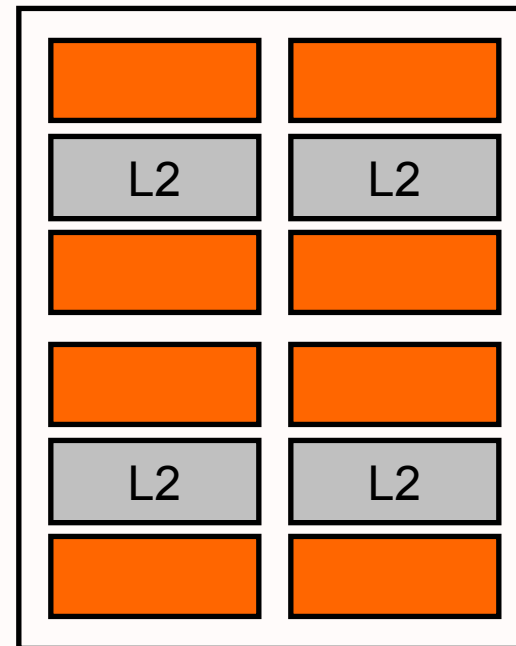
# Motivation

Best architecture depends on workload:

Lots of parallelism



Limited parallelism

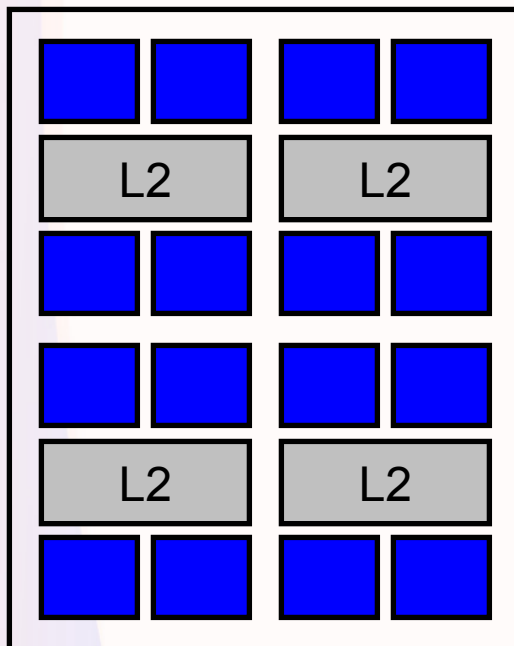


How do we choose at design time without knowing the workload characteristics?

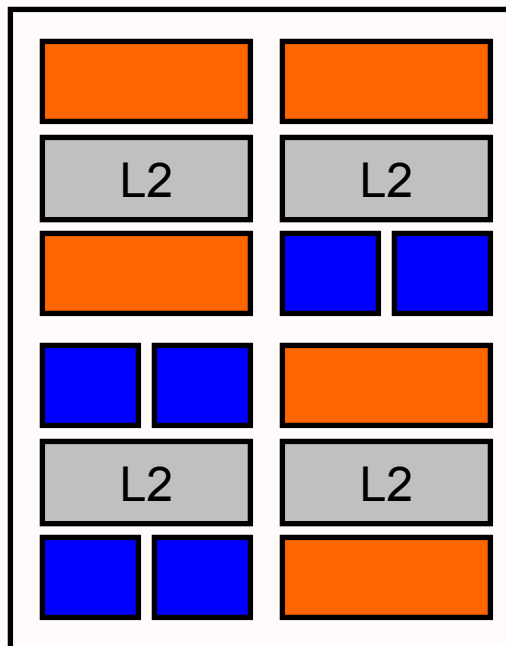
# Basic Idea

Allow the architecture to adapt *at runtime*:

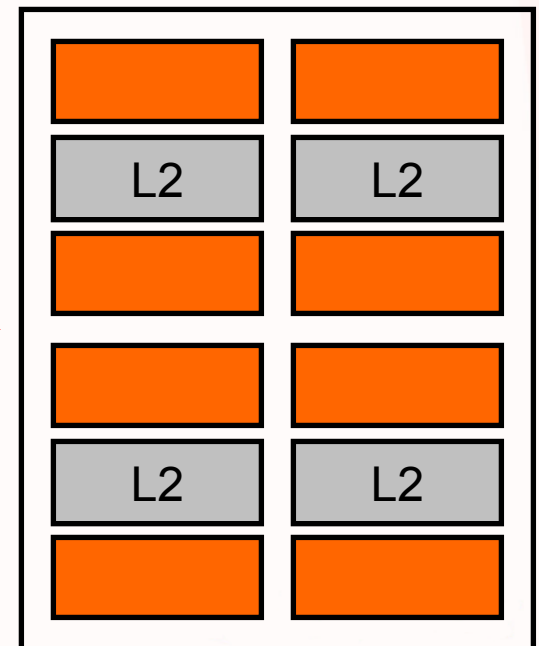
Throughput  
(Homogeneous)



Mixture  
(Heterogeneous)



Latency  
(Homogeneous)

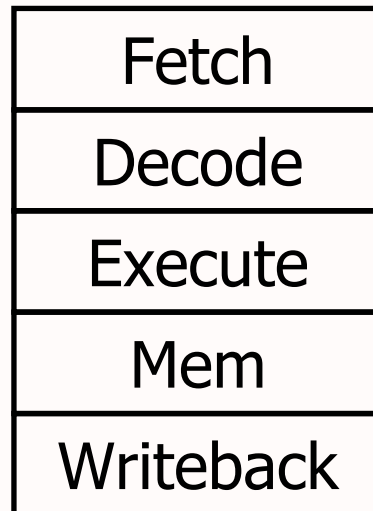


# Key Insights

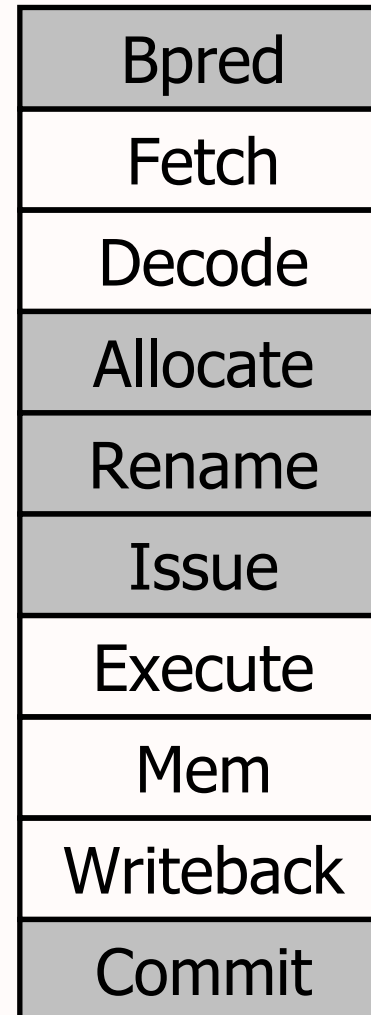
- Large, multi-threaded register files can be repurposed to support out-of-order execution
- If cores are small, single-cycle communication between neighbors is feasible
- Leverage prior work on scaling large out-of-order cores in order to reduce overhead

# Adaptive Pipeline

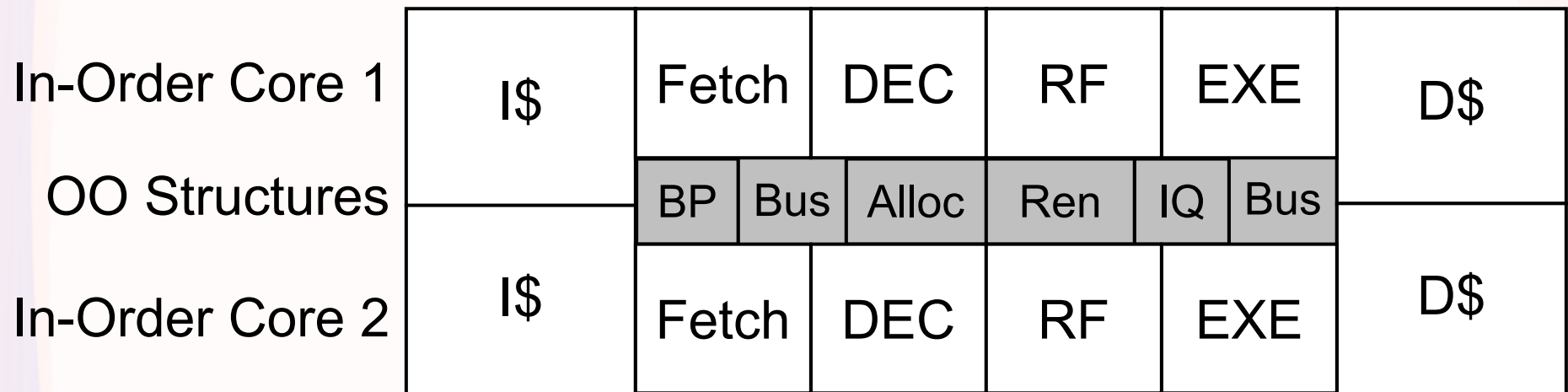
In-order



Out-of-order



# Adaptive Pipeline (2)



# Consumer-Based Issue Queue

- Consumers subscribe to their producers at rename
- Producers set ready bits of their consumers at execute
- Fixed number of subscriber slots can cause stalls
- Select uses static priority encoder based on IQ position rather than oldest-first

# Simplified Load-Store Queue

- Memory Alias Table (MAT)
- No store forwarding
- No conservative waiting on stores
- Only detect memory order violations after they have occurred and flush the pipeline when the offending instruction commits

# MAT Example

st 0x13, r5  
ld r1, 0x13

MAT	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0

# MAT Example

```
st 0x13, r5  
ld r1, 0x13   EXE
```

ld executes and  
increments counter

MAT	
0	0
1	0
2	0
3	1
4	0
5	0
6	0
7	0

# MAT Example

st 0x13, r5 COM  
ld r1, 0x13

st commits and  
sets flag

MAT	
0	0
1	0
2	0
3	1 !
4	0
5	0
6	0
7	0

# MAT Example

ld r1, 0x13 COM

ld commits, sees flag,  
and flushes pipeline

MAT	
0	0
1	0
2	0
3	1 !
4	0
5	0
6	0
7	0

→ Flush

# MAT Example

ld r1, 0x13

MAT is reset and  
execution resumes

MAT	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0

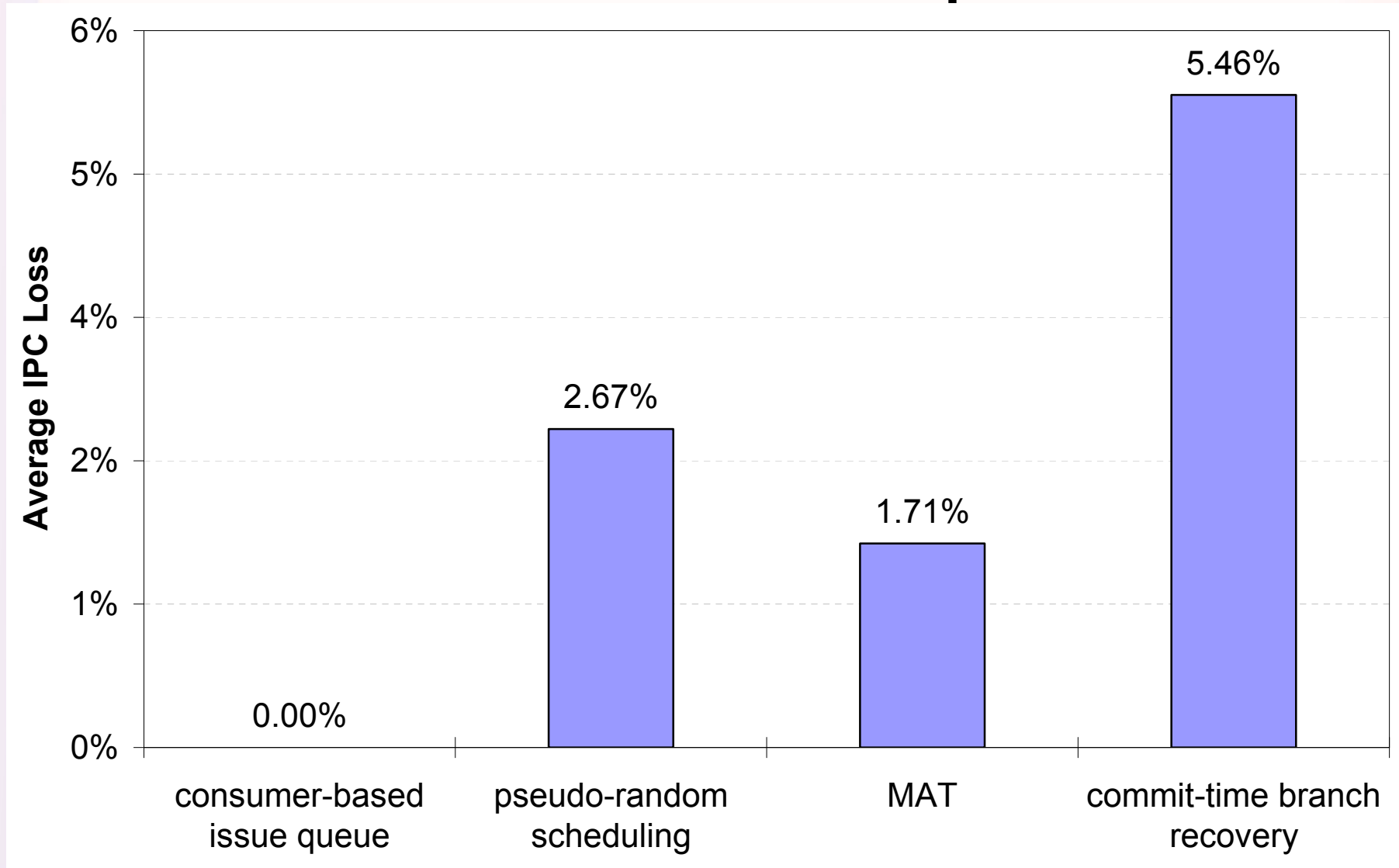
# Area Overhead

<b>Core Type</b>	<b>Area (mm<sup>2</sup>)</b>	<b>Scalar Core Cost</b>
1-way in-order	1.91	-
Federated out-of-order	3.97	0.07
2-way out-of-order	5.07	2.65
4-way out-of-order	11.19	5.85

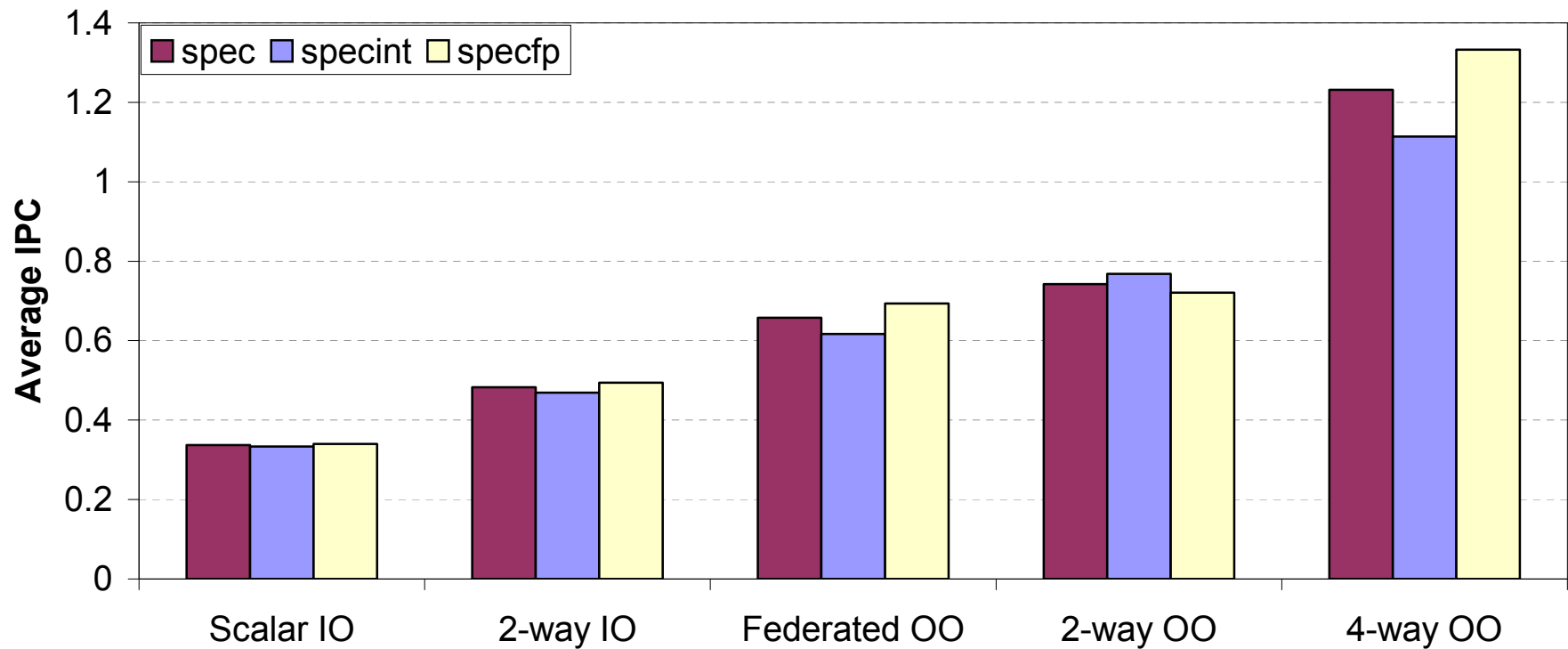
# Simulation Methodology

- Simulator:
  - SimpleScalar 3.0
  - Wattch
- Workloads:
  - SPEC2000 benchmarks
  - Simpoint: select 100 million instructions

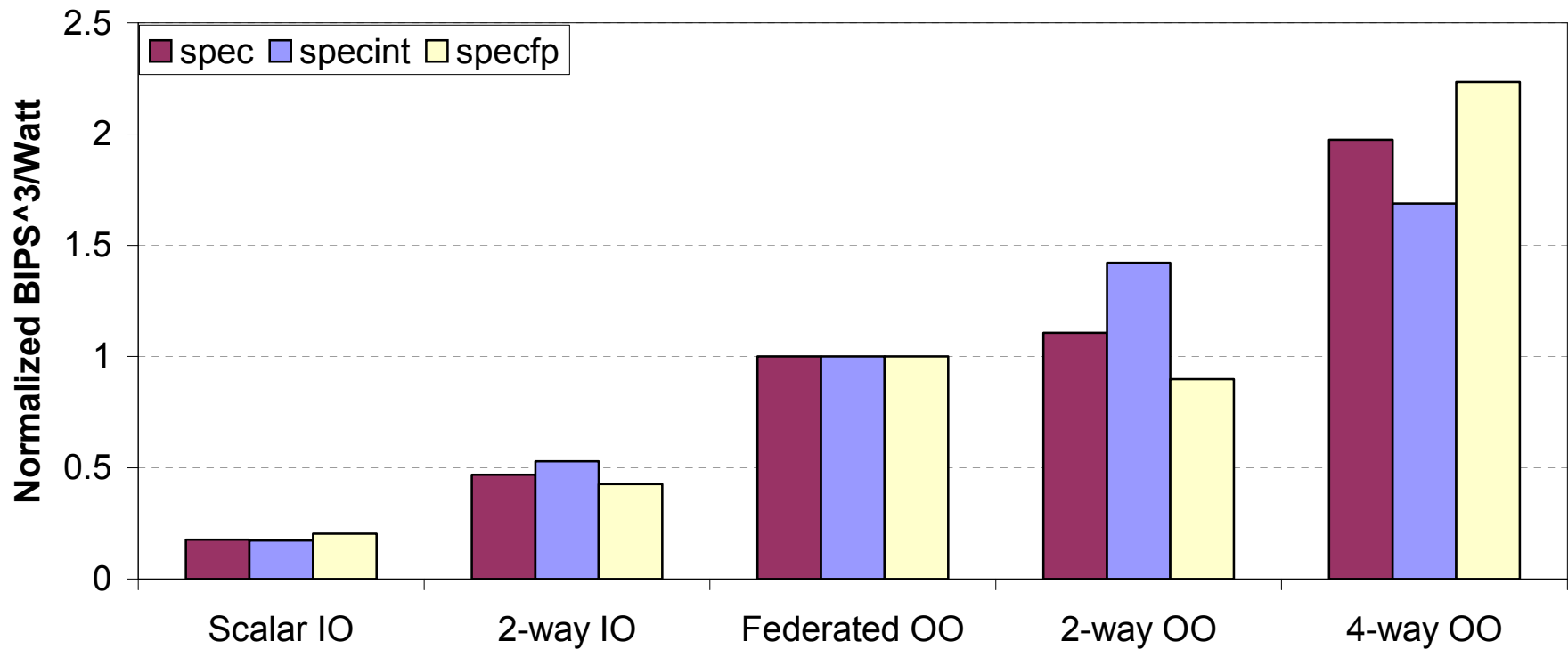
# Performance Impact



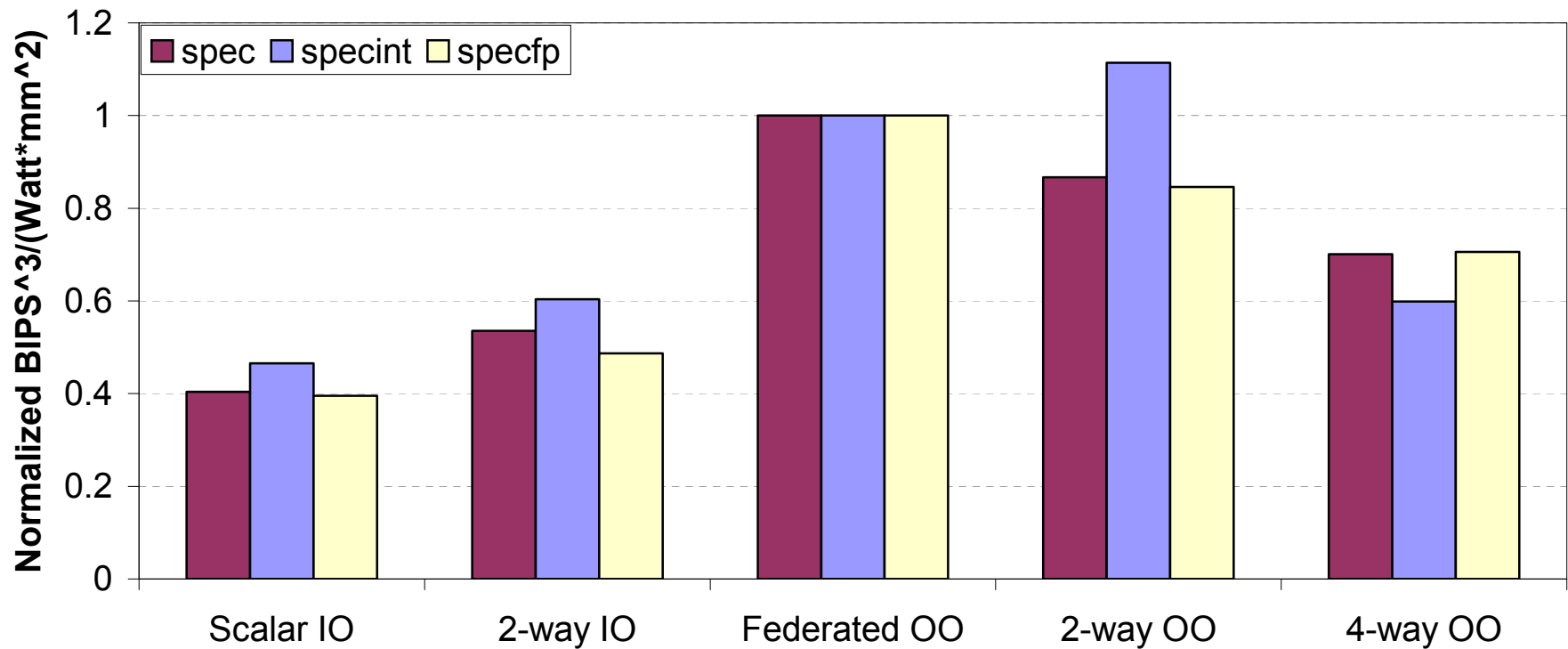
# Performance Results



# Energy Efficiency Results



# Energy-Area Efficiency Results



# Conclusions

- Two in-order cores can be Federated *at runtime* to form a two-way out-of-order core with low area overhead
- Federated core approximately doubles single-thread performance of underlying cores
- Best energy-area efficiency of options studied
- Federation allows an architecture to dynamically trade off throughput and single-thread latency

# Future Work

Software support



System-level architecture



Architecture of individual cores



Low-level implementation

# Technology Transfer

- Liaison interactions:
  - AMD: Srilatha Manne
  - Intel: Perry Wang & Jamison Collins
  - Quarterly meetings
- Publications:
  - DAC 2008
  - Longer tech report published on SRC website
  - Journal version currently in submission
- Presentations:
  - Intel, December 2007 (hosted by Pradeep Dubey)
- Internships:
  - Intel, Summer 2007 (unrelated to this work)

# Acknowledgments

This work was supported by:

SRC/AMD Fellowship

SRC grant: 2007-HJ-1607

NSF grants: CCR-0306404

CNS-0509245

IIS-0612049

Thanks to:

Michael Frank

Doug Burger

Mircea Stan

Jeremy Sheaffer