Complexity-Effective Superscalar Processors

Subbarao Palacharla, Norm Jouppi,[†] Jim Smith

24th International Symposium on Computer Architecture

Tuesday, June 3rd, 1997

University of Wisconsin-Madison

[†]DEC Western Research Lab

Motivation

- Wide, homogeneous superscalar will not scale well
  - Longer wires increase delay
  - Smaller feature sizes accentuate wire delays

  → Potentially slow clock
- Performance $\propto$ $(\mathrm{IPC} \times \mathrm{Clock\ speed})$
- Study microarchs that maximize (IPC $\times$ Clock speed)

Complexity-Effective Superscalar Microarchitectures

Motivation (cont'd.)

- Simple to measure IPC
  - trace-driven simulation counting cycles
- Hard to measure complexity
  - full implementation to be accurate
- Need simple models for
  - quantifying complexity
  - identifying complexity trends

Quantifying Complexity of Superscalar Processors

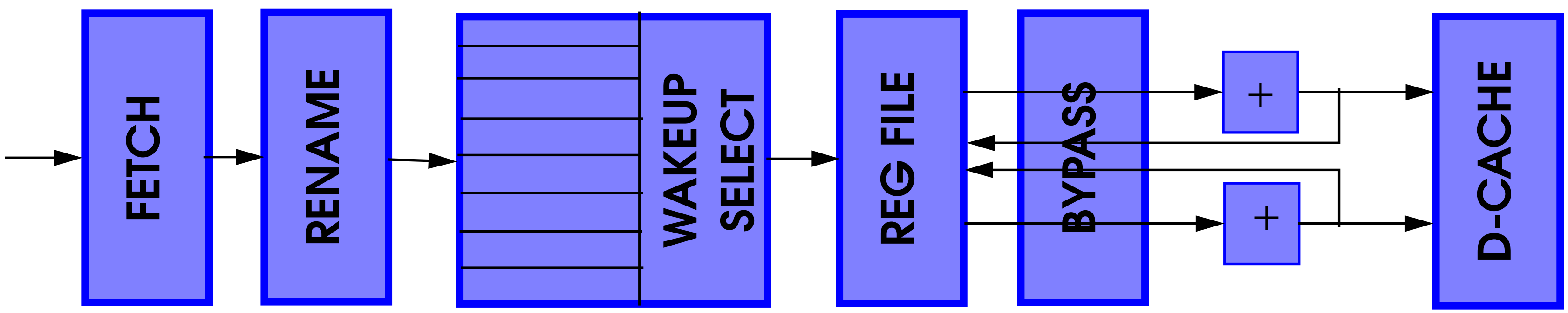
Outline

- ~~Motivation~~
- Measuring complexity
  - Our approach
  - Two case studies: wakeup and bypass logic
  - Overall delay results
- Complexity-effective microarchitectures
  - Dependence-based microarchitecture
  - Other clustered microarchitectures
- Conclusions

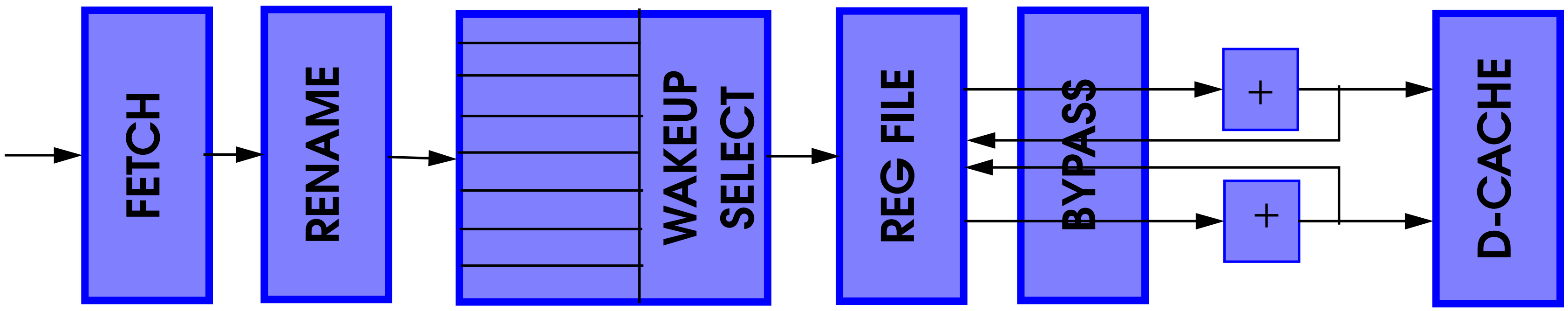
Our approach

- Concentrate on key pipeline structures
  - delay is a function: issue width, window size
  - primarily dispatch and issue-related
  - broadcast operations over long wires
- Develop simple delay models

Baseline superscalar model



Key structures



STRUCTURE	DELAY
Fetch logic	f(IW)
Rename logic	f(IW)
Window wakeup logic	f(IW,WINSIZE)
Window select logic	f(WINSIZE)
Bypass logic	f(IW)
Register file	f(IW)
Cache	~f(IW)

IW - Issue Width

WINSIZE - Window Size

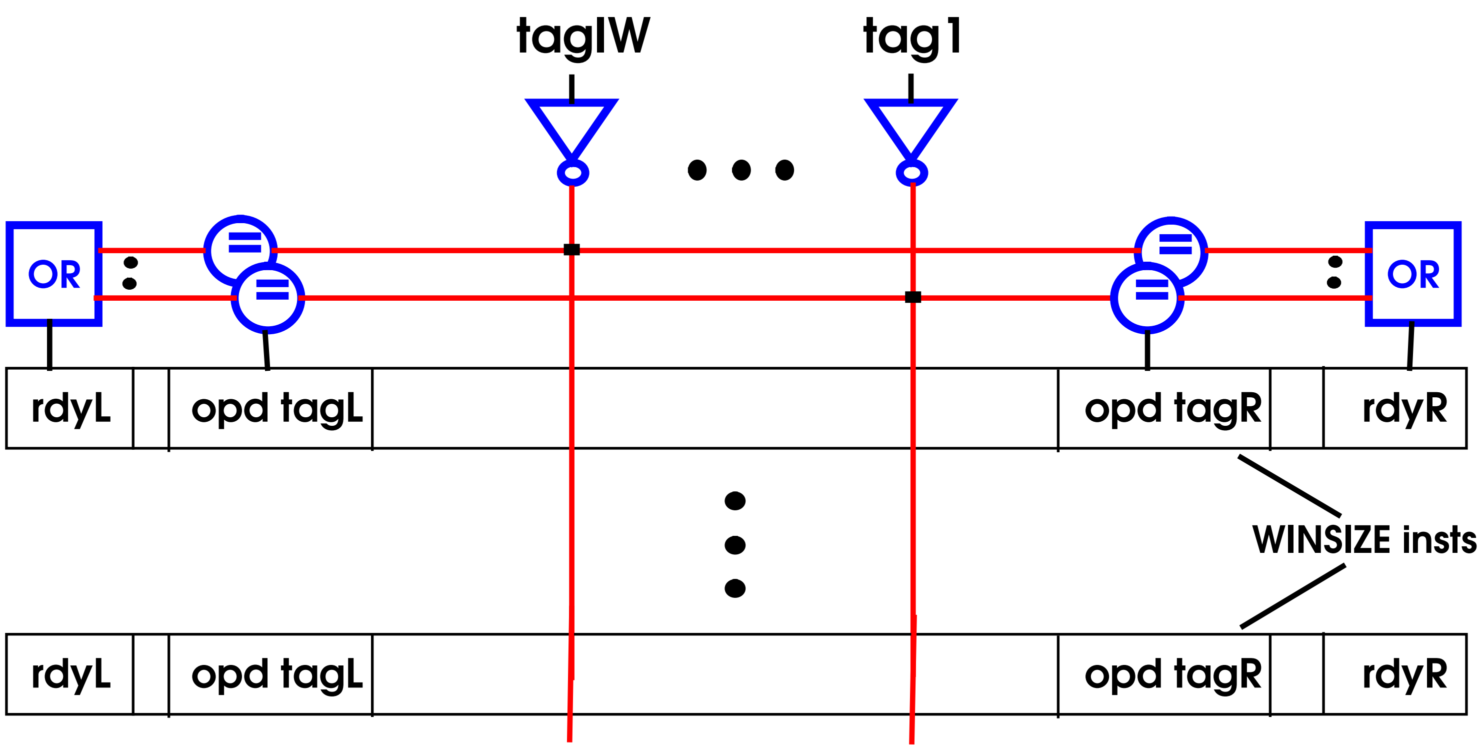
Methodology

- Representative CMOS circuit
  - ISSCC proceedings
  - DEC engineers
- Optimize circuit
  - transistor sizing
  - reducing fan-in
  - transistor reordering to speed critical path
- Express delay as function of IW and WINSIZE
- Spice simulate for 0.8μm, 0.35μm, 0.18μm techs
- Verify model predictions match simulations

Outline

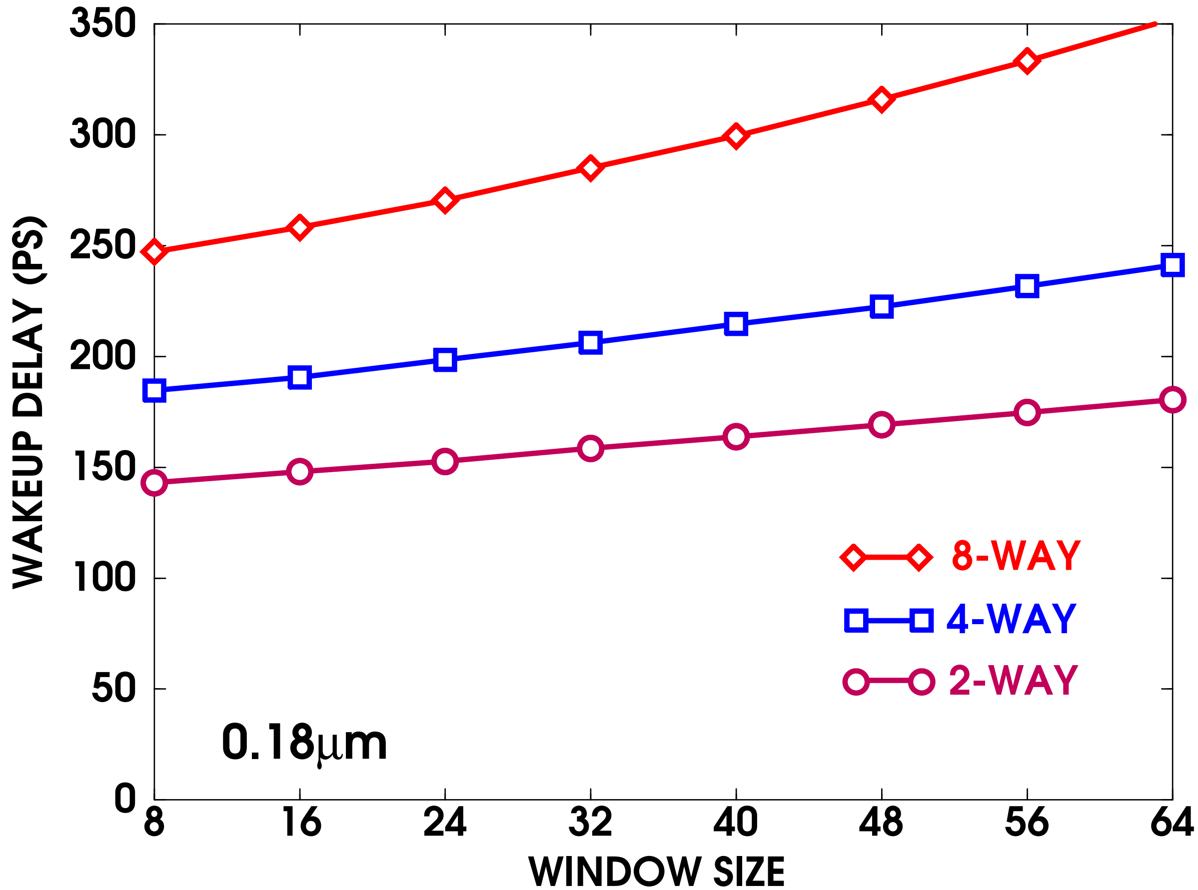
- ~~Motivation~~
- ~~Measuring complexity~~
  - ~~Our approach~~
  - Two case studies: wakeup and bypass logic
  - Overall delay results
- Complexity-effective microarchitectures
  - Dependence-based microarchitecture
  - Other clustered microarchitectures
- Conclusions

Window wakeup logic



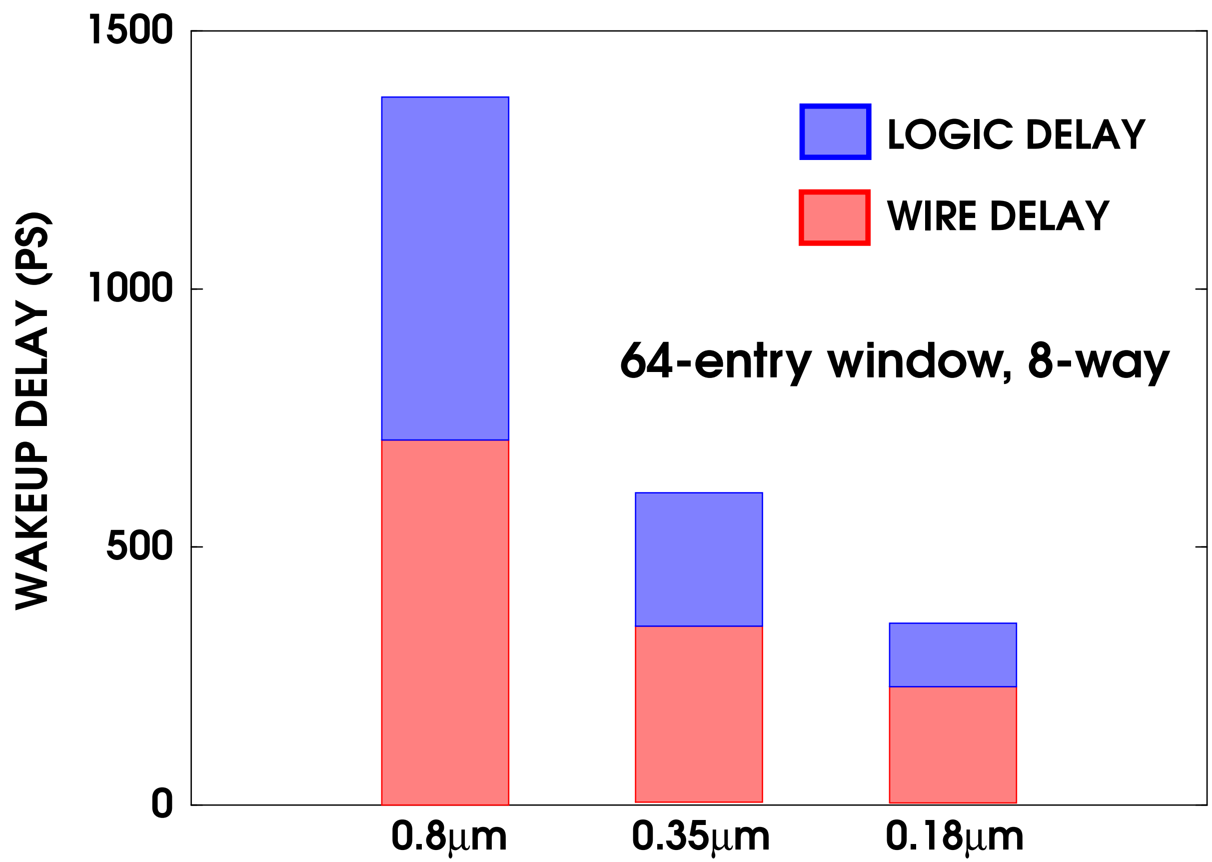
- Broadcast result tags to waiting instructions
- Compare result tags against source operand tags

Window wakeup logic (cont'd.)



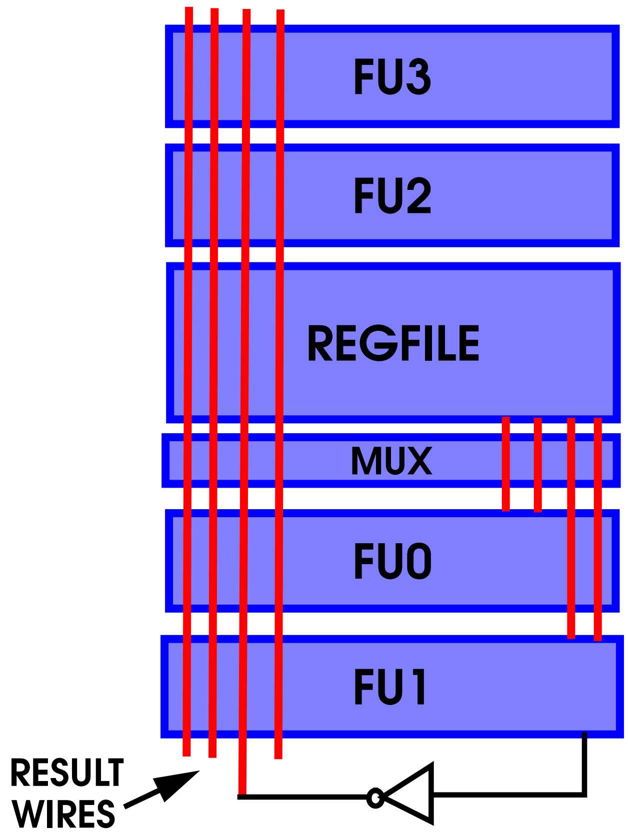
- At least linear in window size
- Issue width has greater impact

Window wakeup logic (cont'd.)



- Wire delays do not scale as well as logic delays

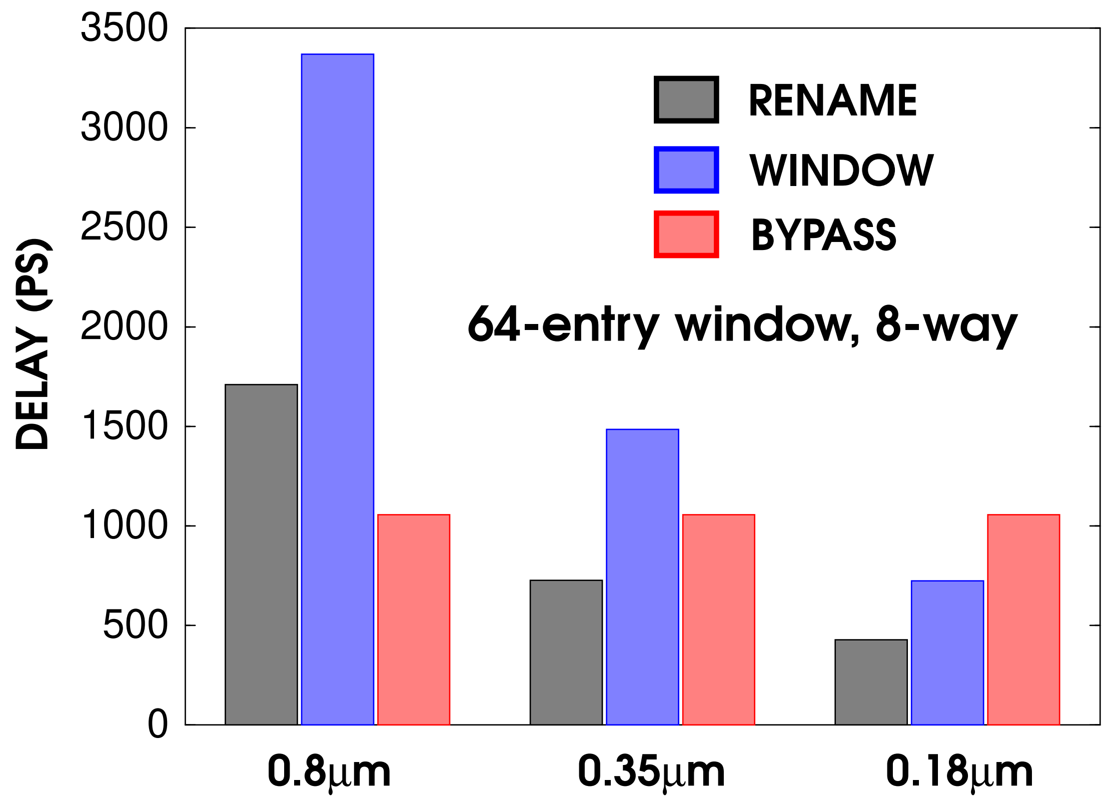
Bypass logic



- Result wire length increases linearly with issue width
- Delay increases quadratically with wire length

$\rightarrow$ Bypass delay $\propto$ IW^2

Overall delay results

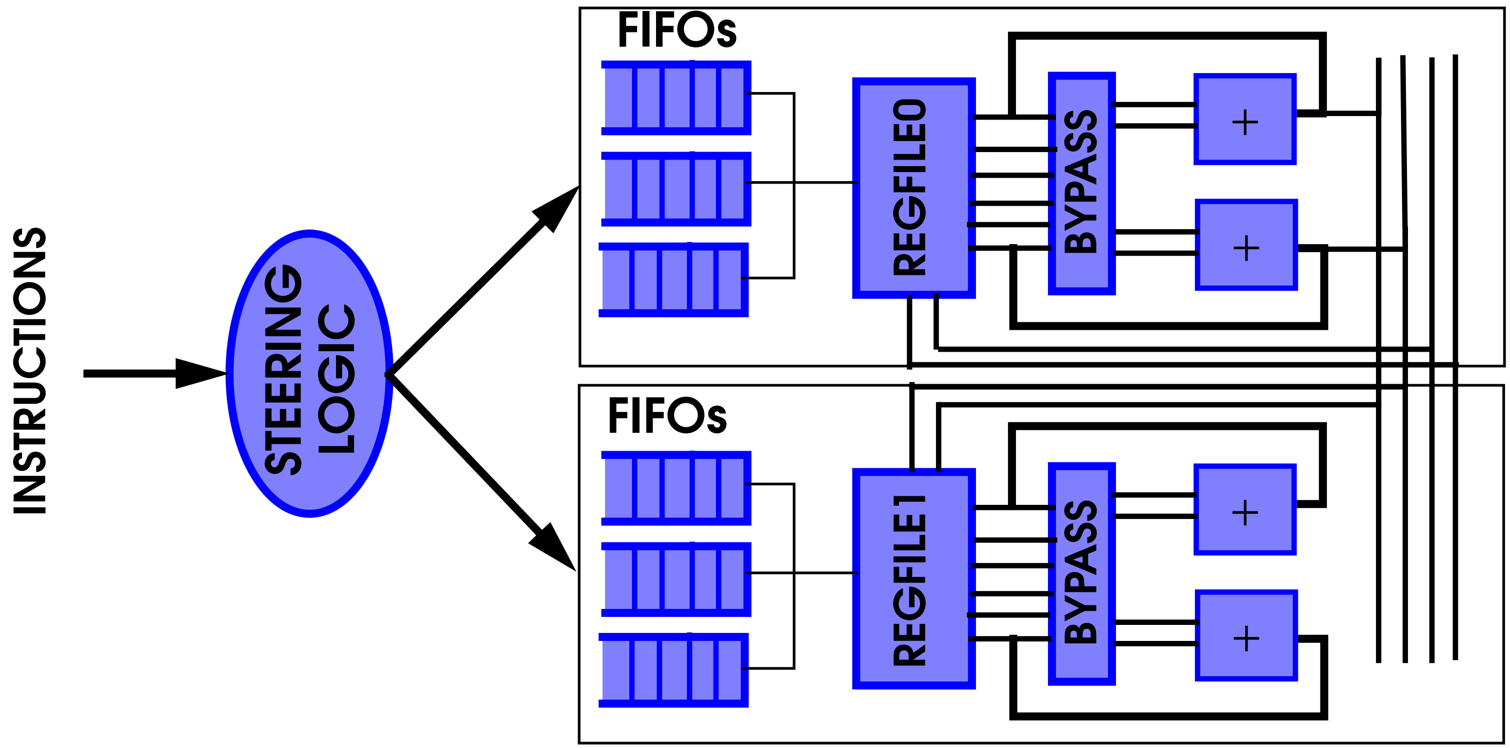


- Bypass delays do not scale with feature size
- Bypass delays: major problem in future designs
- Window logic is the next most critical

Outline

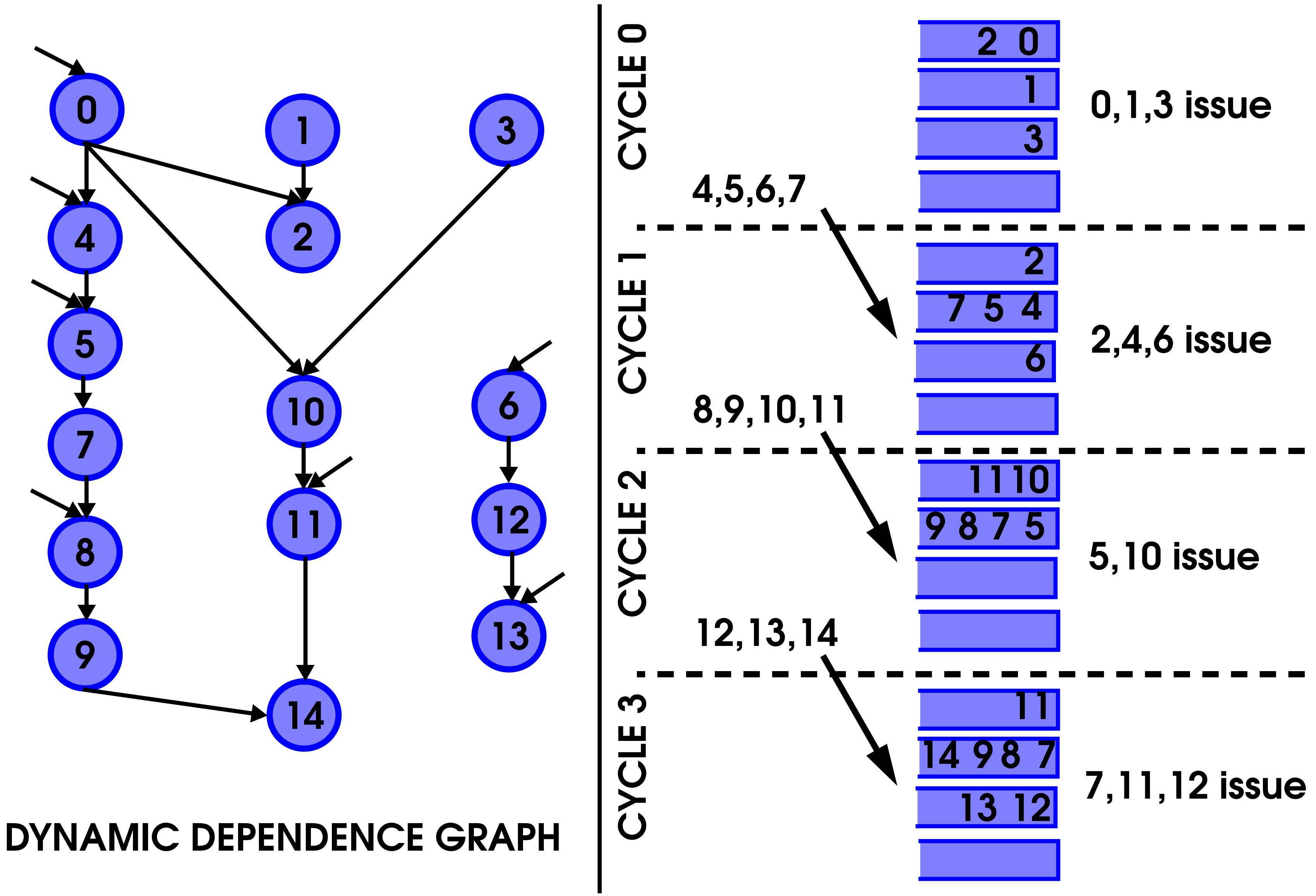
- ~~Motivation~~
- ~~Measuring complexity~~
- Complexity-Effective microarchitectures
  - Dependence-based microarchitecture
  - Other clustered microarchitectures

Dependence-based microarchitecture

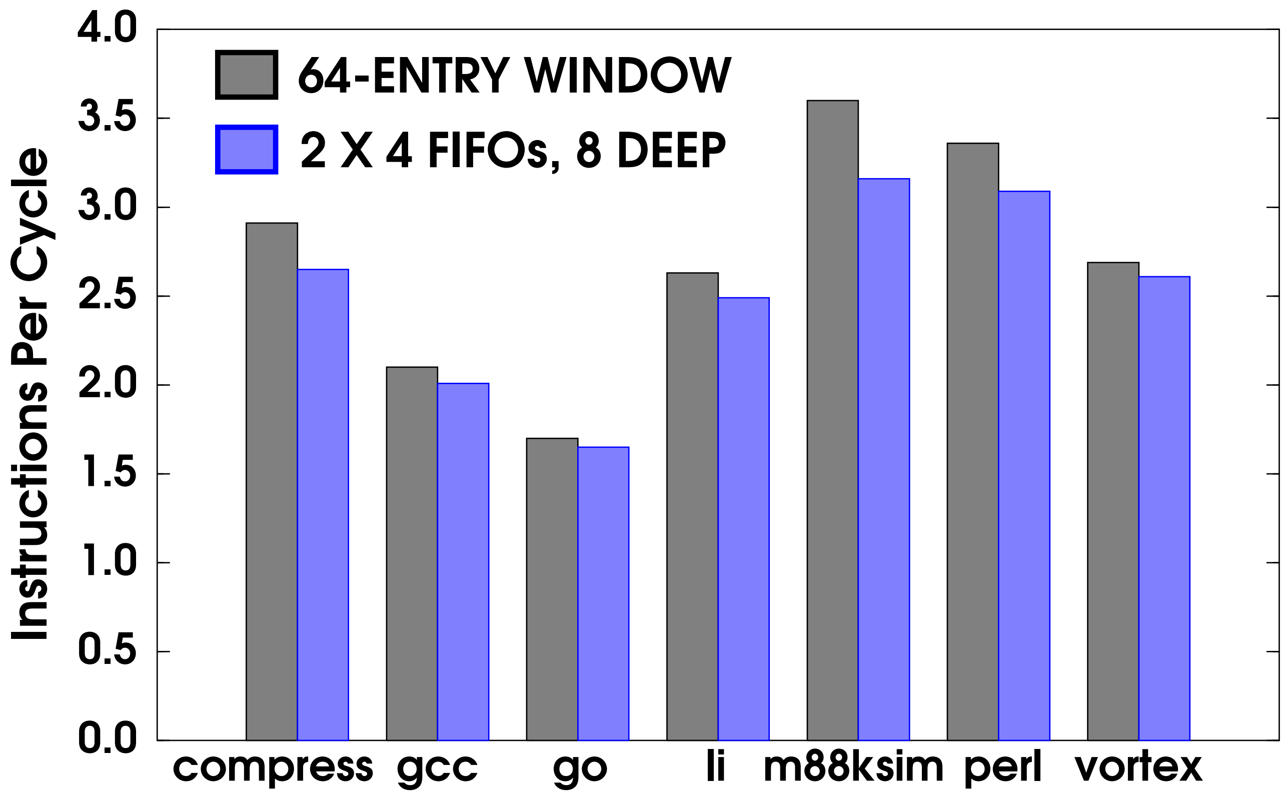


- Replace window with FIFOs
  - Dependent instructions steered to each FIFO
  - Window logic monitors FIFO heads only
- Clustered to reduce bypass delay (similar to 21264)
  - extra cycle for bypassing across clusters

Example of steering - 4-way machine



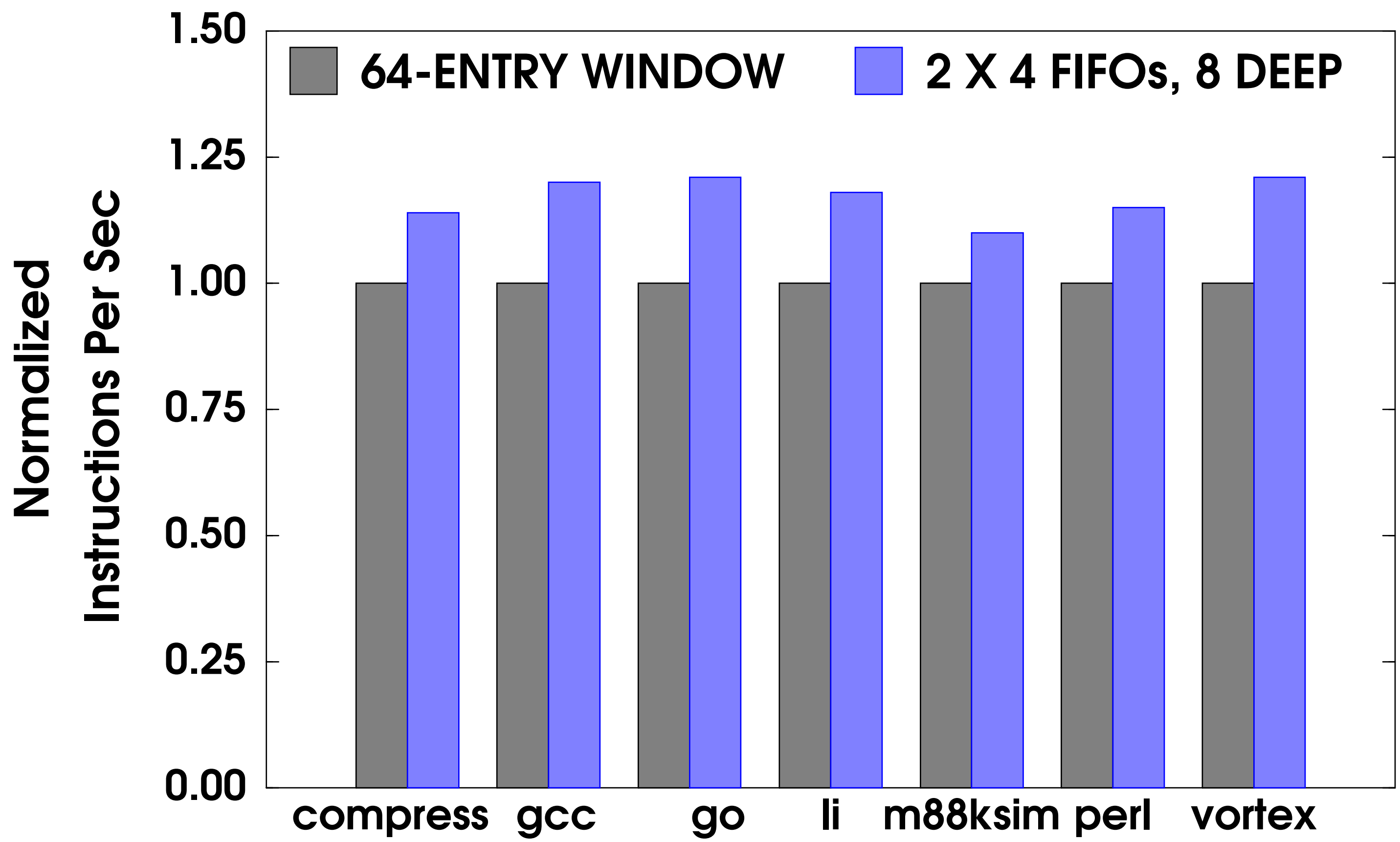
Performance results - IPCs



- Worst IPC degradation: 12% m88ksim, 9% compress due to slow (2-cycle) inter-cluster bypasses
- But, based on window delay, clock can be 25% faster

Performance ∝ (IPC × Clock speed) !

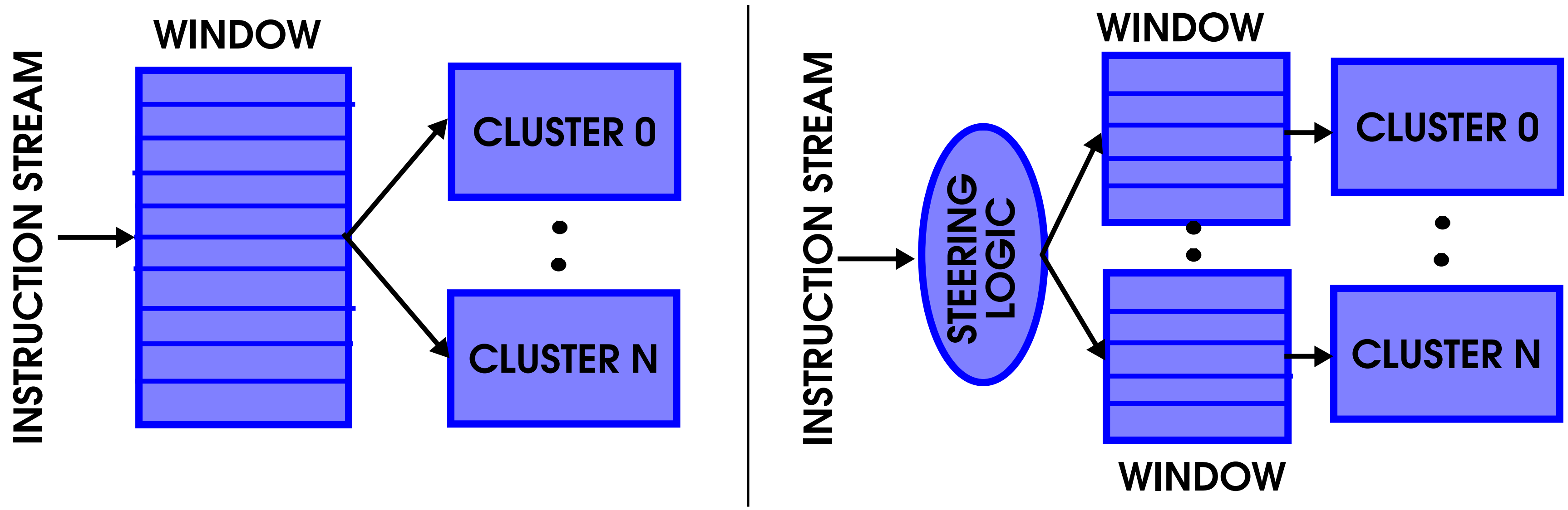
Performance results - Normalized Instructions Per Sec.



- Performs better for all benchmarks
- Net performance improvements: 10% to 22%

  Average performance improvement: 16%

Other clustered microarchitectures



Single window

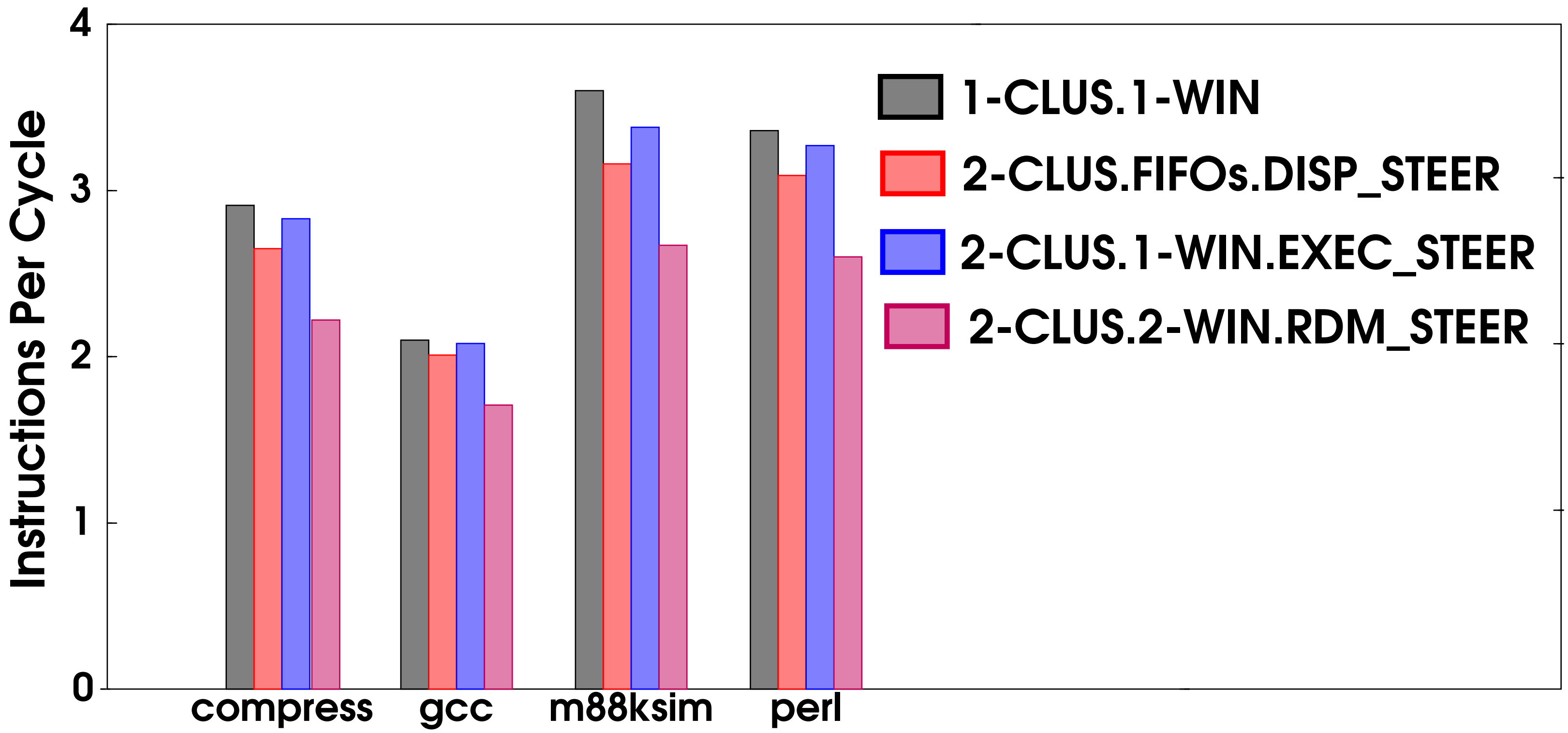
Execution steering

Mutliple windows

Dispatch steering

Extra cycle for inter-cluster bypasses

Performance results - IPCs

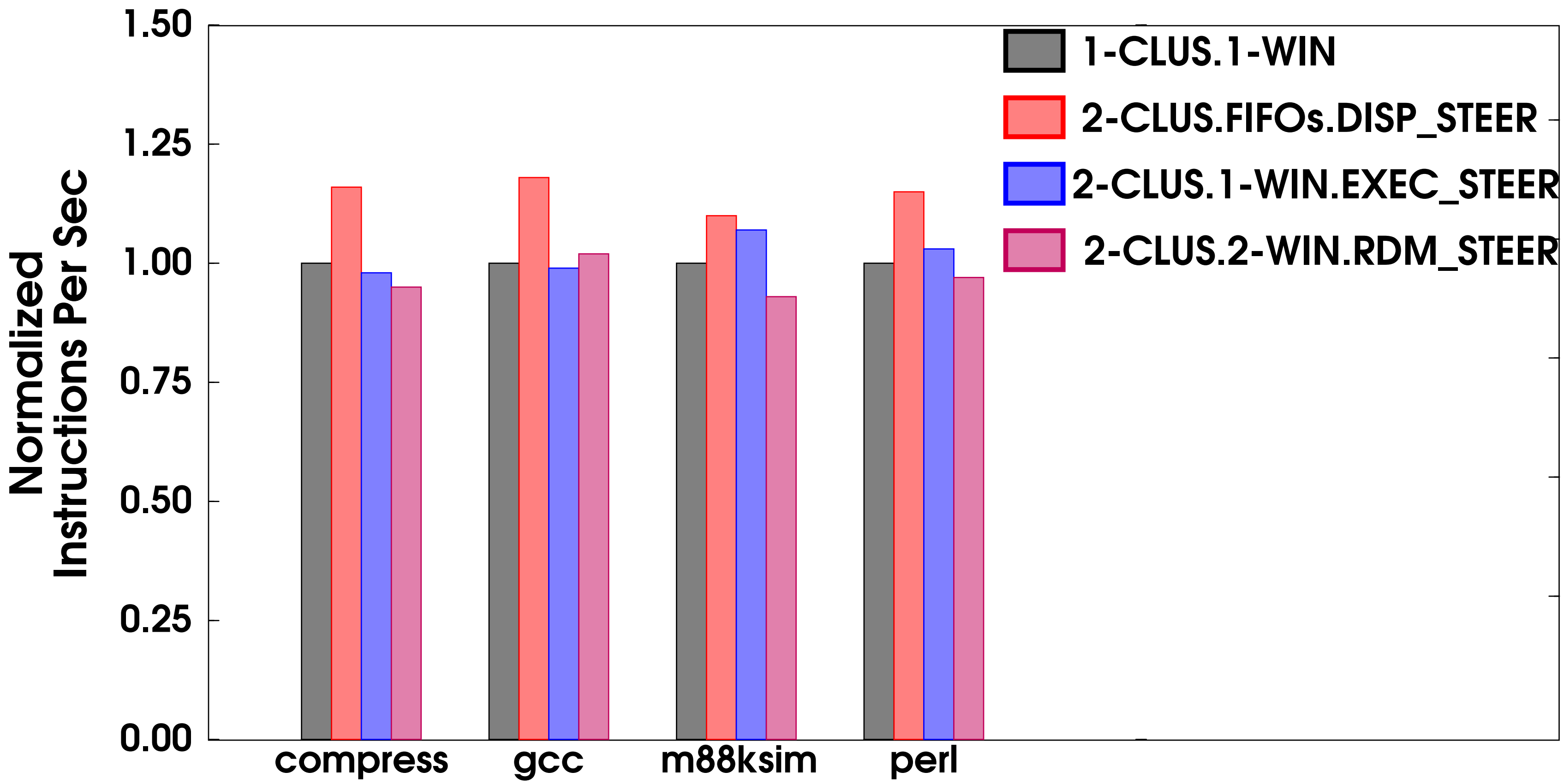


- Execution steering achieves high IPCs

  but steering is in critical issue path
- Random steering consistently performs worst

  17% to 26% IPC degradation

Performance results - Normalized Instructions Per Sec



- Dependence-based microarch performs best
- Random steering performs worse even w/ fast clock

Conclusions

- Cycle time is a crucial performance factor
- Detailed modeling essential

- Bypasses are critical performance issue

  clustering can help considerably
- Then, window logic is critical

  dependence-based processors can reduce window complexity

clustering + dependence-based == wide issue + fast clock

How good are your circuits?

- Based on design published by microprocessor vendors
  - ISSCC proceedings, DEC engineers
  - Studied alternatives for some structures
- Many circuit tricks can be used to optimize the circuits
  - relative delay times should be accurate enough
  - more interested in relationships, trends

Hard problem: study only a first effort in the direction

What about other structures?

- Front end stages
  - Pipeline at the cost of

    increased mispredict penalty

    3% IPC degradation per front-end stage

    more bypass paths
- Caches

  Size L1 to fit in a cycle

  Pipeline
- Registers

  Pipeline

  Tullsen et. al. report only 2% degradation in IPC

Using buffers to reduce wire delays

- Yes, buffers can reduce delay

  but delay is still at least linear

  buffers add delay and consume power
- Wires with multiple drivers need bidirectional buffers

  not easy to switch direction fast enough
- Quadratic increase in delay can still result
  - e.g. window wakeup logic delay

    increases at least linearly with issue width

    increases at least linearly with window size
- The problem only resurfaces at a smaller feature size

Steering logic complexity

- Can be done in parallel with rename
- Might need an extra pipestage

  3% IPC degradation per front-end stage
- Cache steering information?