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 $\propto$ (IPC $\times$ Clock speed)
- Study microarchs that maximize (IPC $\times$ Clock speed)

Complexity-Effective Superscalar Microarchitectures

Motivation (cont'd.)

- Simple to measur e IPC
  - trace-driven simulation counting cycles
- Har d to measure complexity
  - full implementation to be accurate
- Need simple models f or
  - quantifying complexity
  - identifying complexity trends

Quantifying Complexity of Superscalar Processors

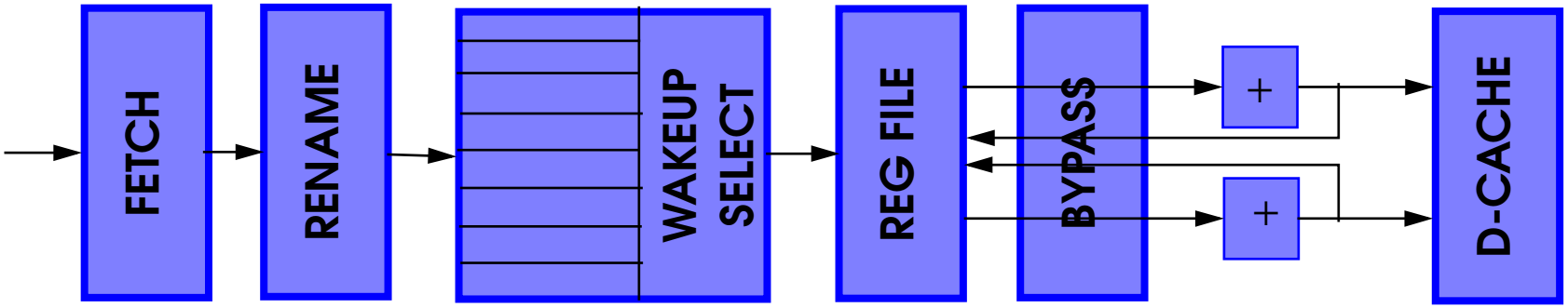
Outline

- ~~Motiv ation~~
- Measur ing complexity
  - Our approach
  - Two case studies: wakeup and bypass logic
  - Overall delay results
- Complexity-ef fective microarchitectures
  - Dependence-based microarchitecture
  - Other clustered microarchitectures
- Conclusions

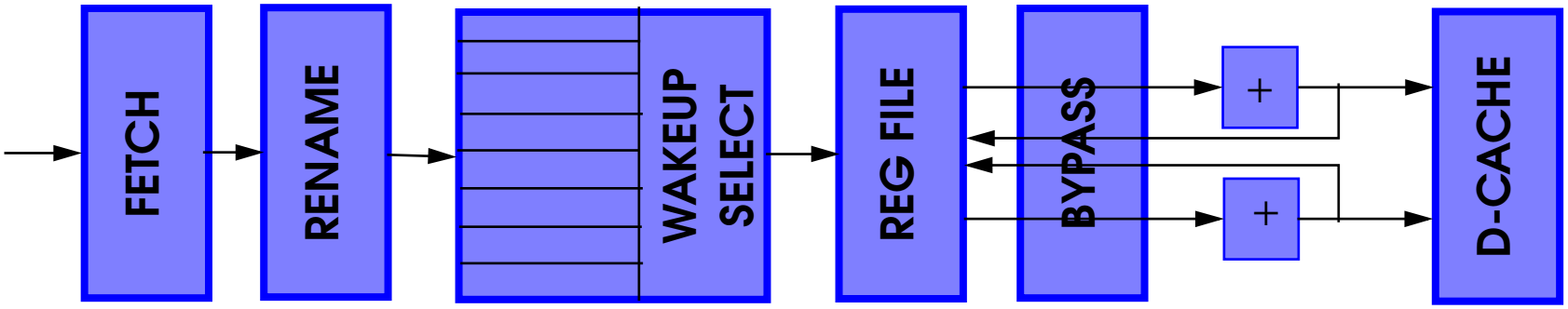
Our approach

- Concentr ate on key pipeline structures
  - delay is a function: issue width, window size
  - primarily dispatch and issue-related
  - broadcast operations over long wires
- De velop 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

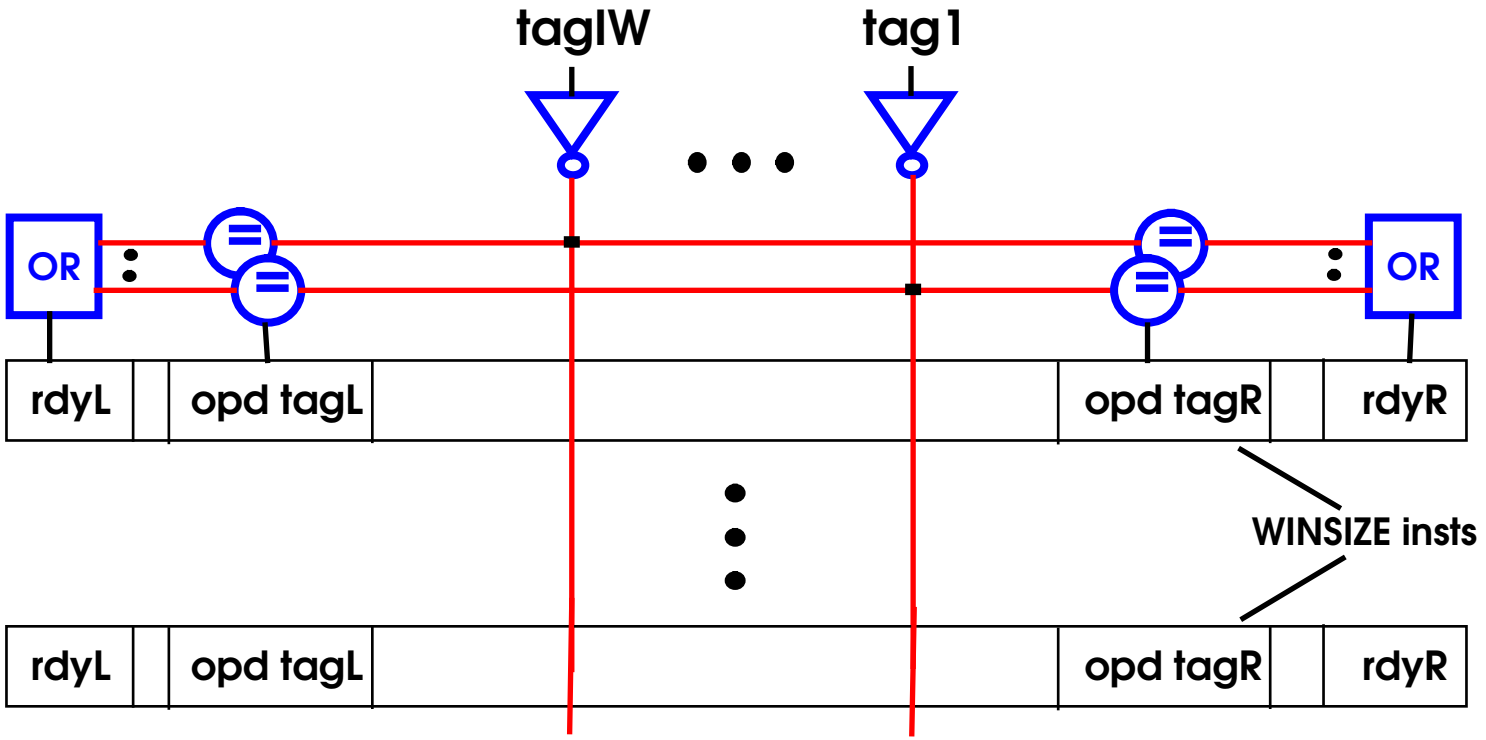
Methodology

- Repr esentative CMOS circuit
  - ISSCC proceedings
  - DEC engineers
- Optimize cir cuit
  - transistor sizing
  - reducing fan-in
  - transistor reordering to speed critical path
- Expr ess delay as function of IW and WINSIZE
- Spice sim ulate for 0.8µm, 0.35µm, 0.18µm techs
- V erify model predictions match simulations

Outline

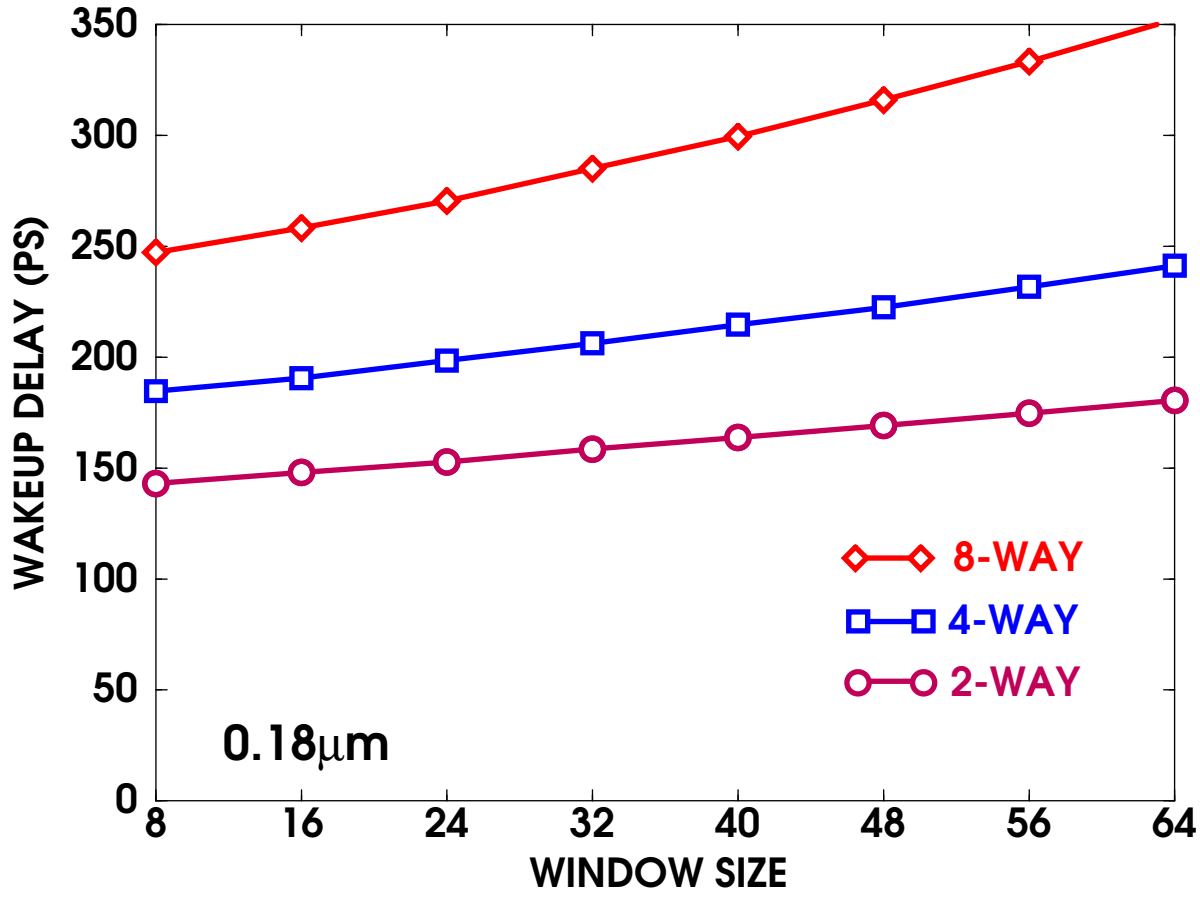
- ~~Motiv ation~~
- ~~Measuring complexity~~
  - ~~Our approach~~
  - Two case studies: wakeup and bypass logic
  - Overall delay results
- Complexity-ef fective microarchitectures
  - Dependence-based microarchitecture
  - Other clustered microarchitectures
- Conclusions

Window wakeup logic



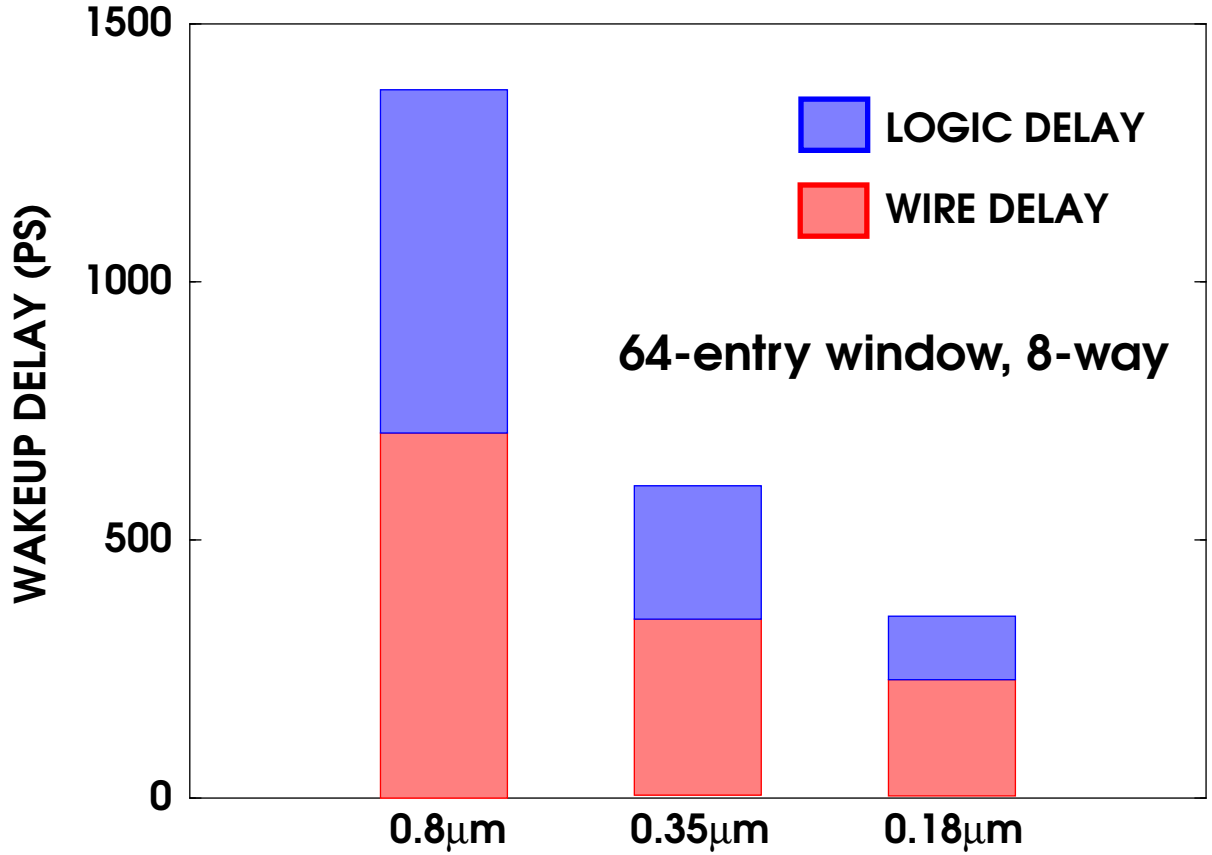
- Br oadcast result tags to waiting instructions
- Compar e result tags against source operand tags

Window wakeup logic (cont'd.)



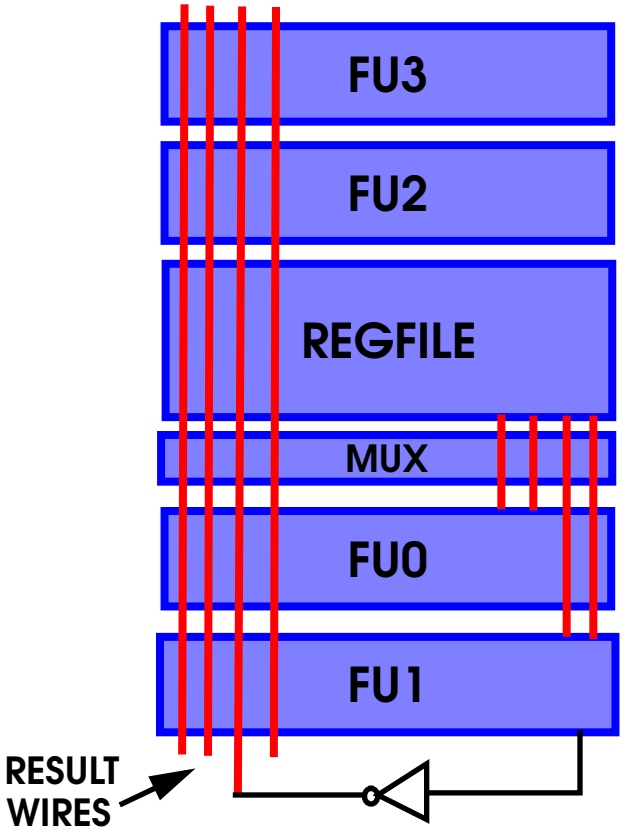
- At least linear in windo w size
- Issue width has gr eater impact

Window wakeup logic (cont'd.)



- Wire delays do not scale as well as logic delays

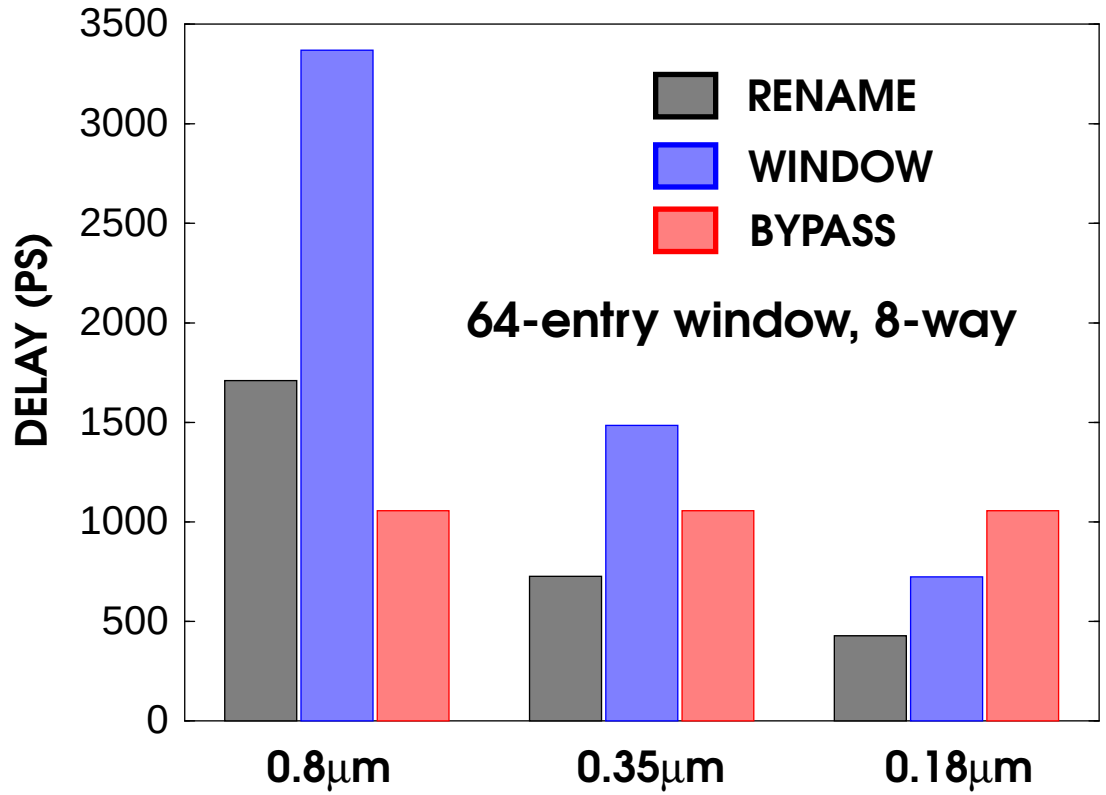
Bypass logic



- Result wir e length increases linearly with issue width
- Delay incr eases quadratically with wire length

→ Bypass delay $\propto IW^2$

Overall delay results

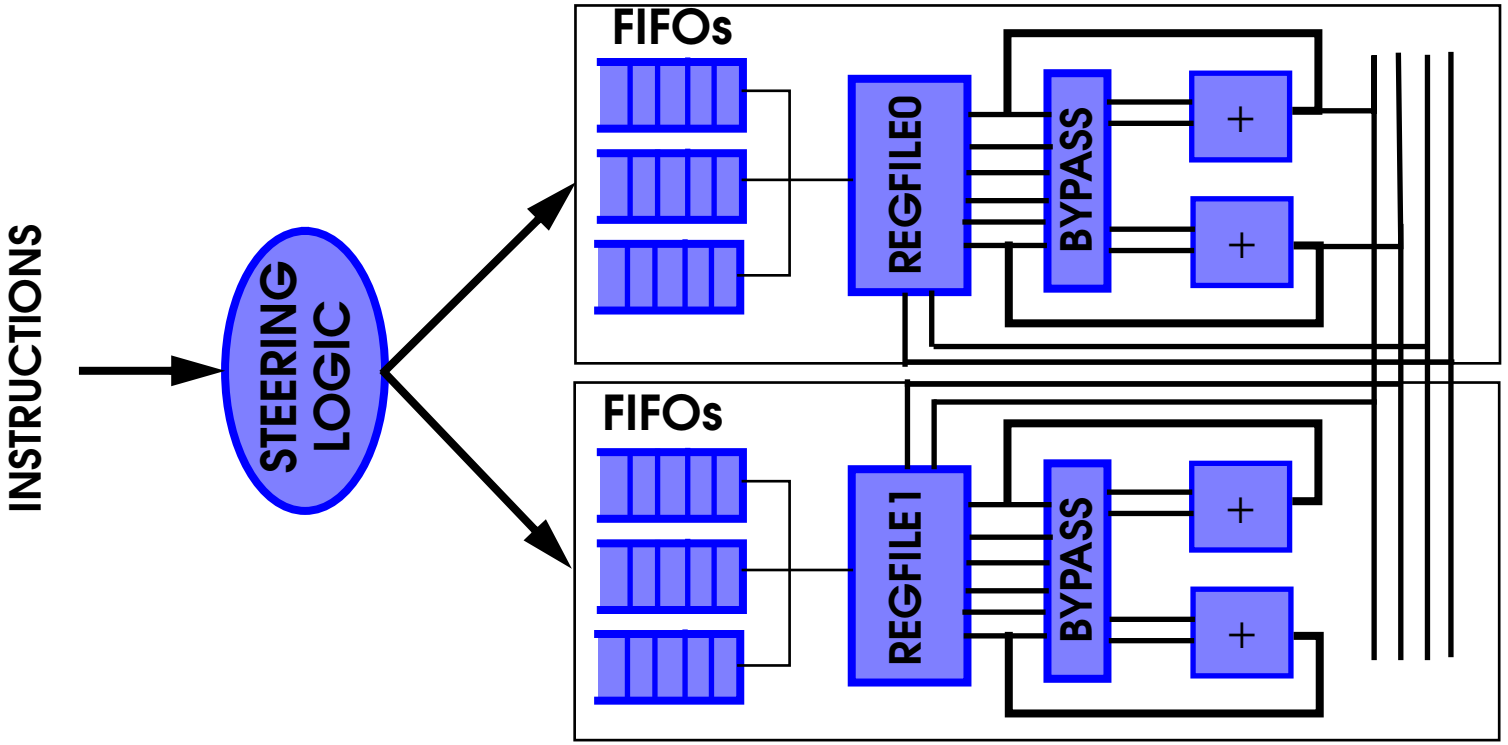


- Bypass delays do not scale with f eature size
- Bypass delays: m ajor problem in future designs
- W indow logic is the next most critical

Outline

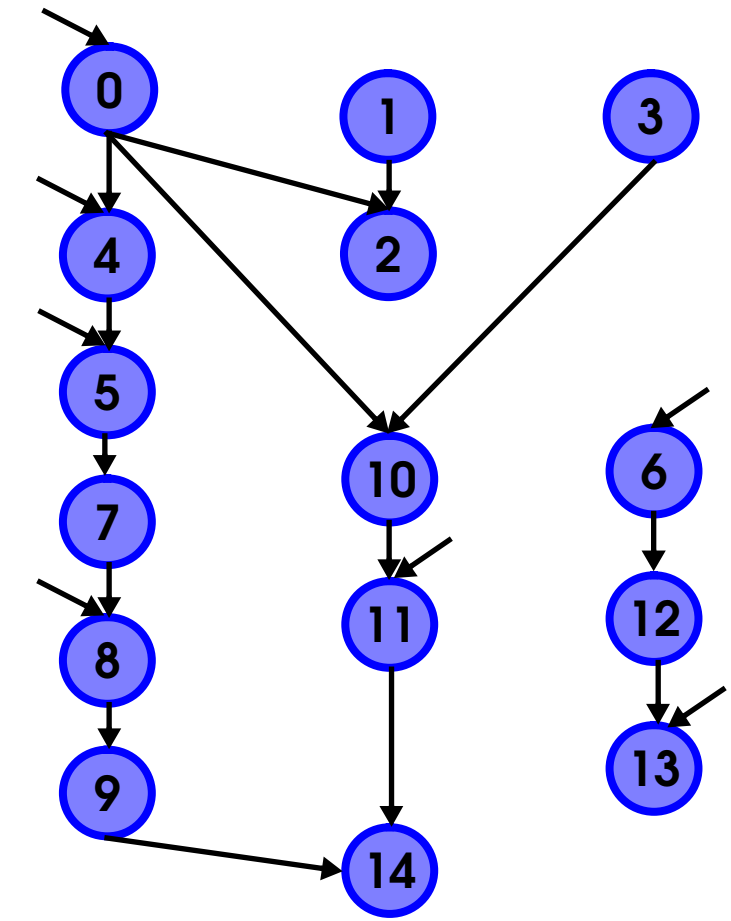
- ~~Motiv ation~~
- ~~Measur ing complexity~~
- Complexity-Ef fective microarchitectures
  - Dependence-based microarchitecture
  - Other clustered microarchitectures

Dependence-based microarchitecture

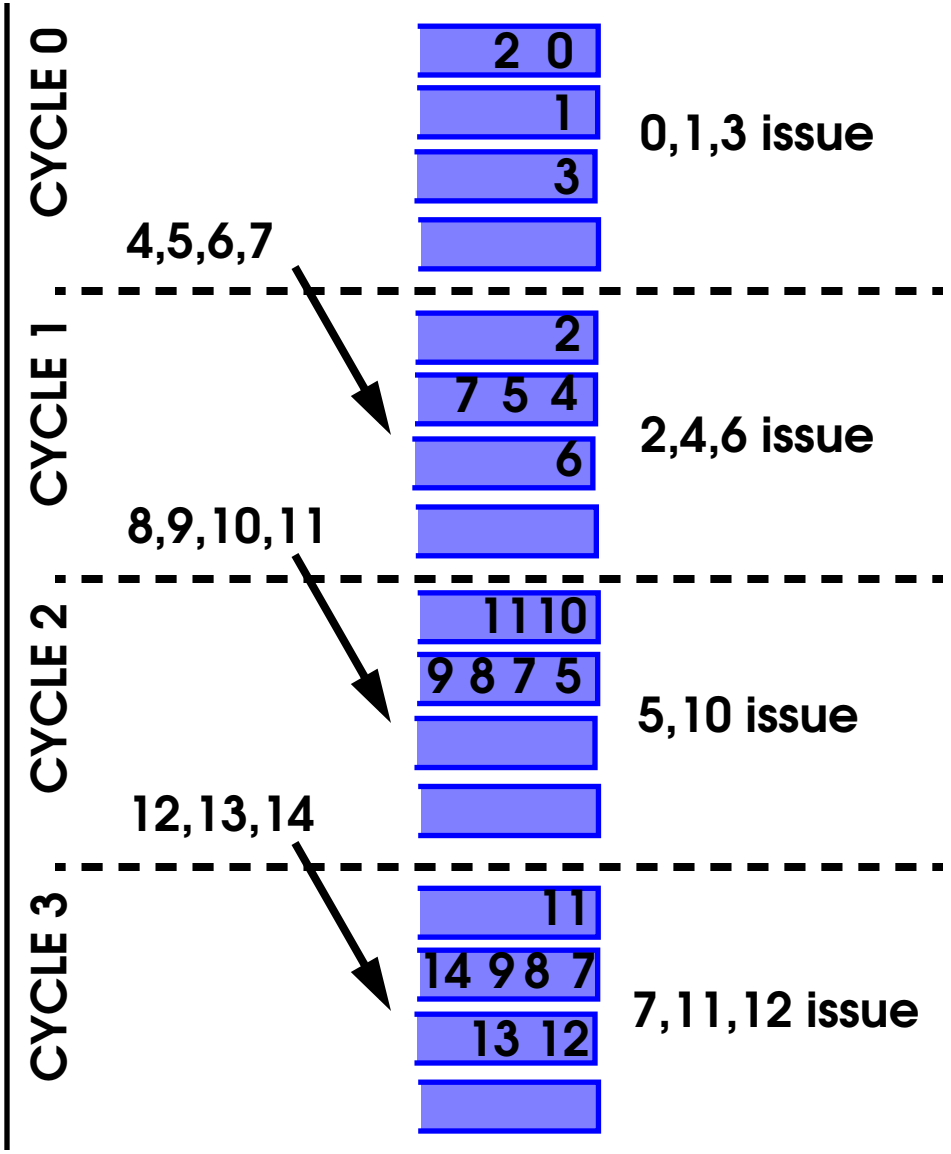


- Replace windo w with FIFOs
  - Dependent instructions steered to each FIFO
  - Window logic monitors FIFO heads only
- Cluster ed to reduce bypass delay (similar to 21264)
  - extra cycle for bypassing across clusters

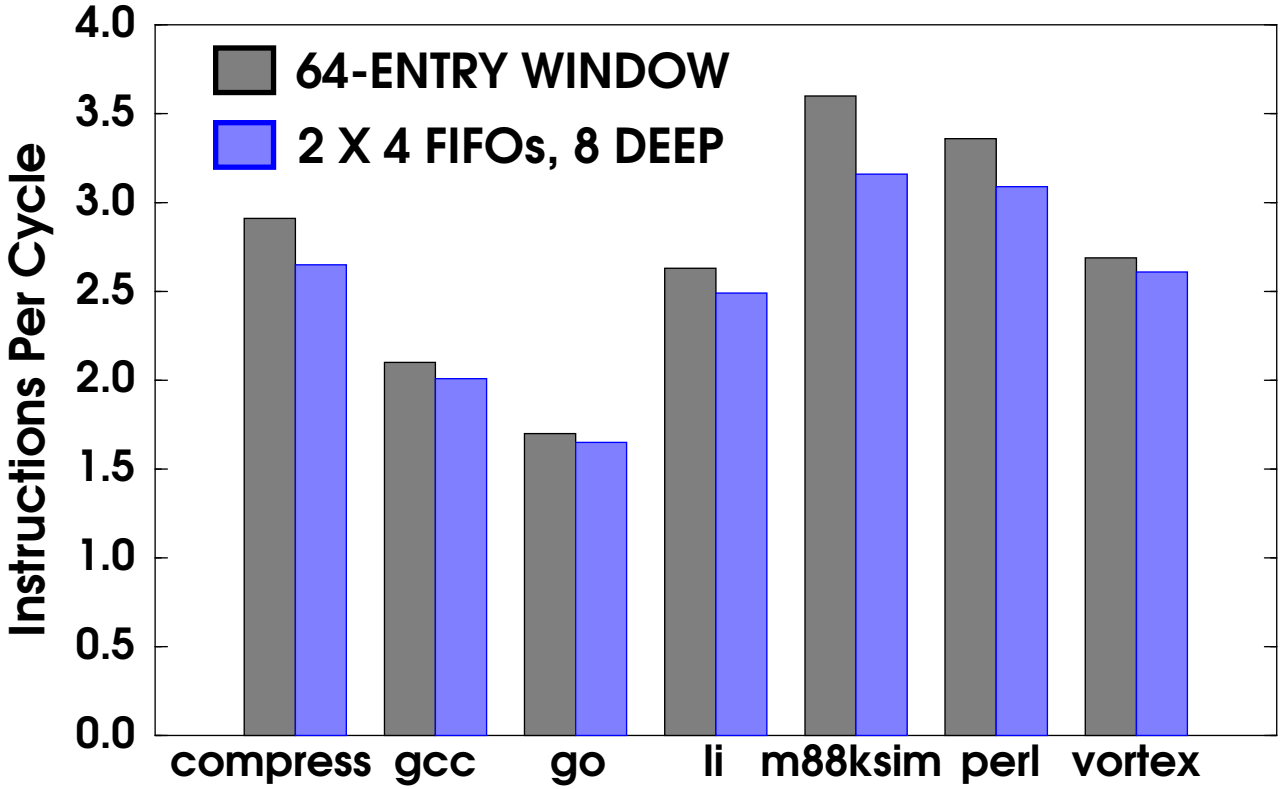
Example of steering - 4-way machine



DYNAMIC DEPENDENCE GRAPH



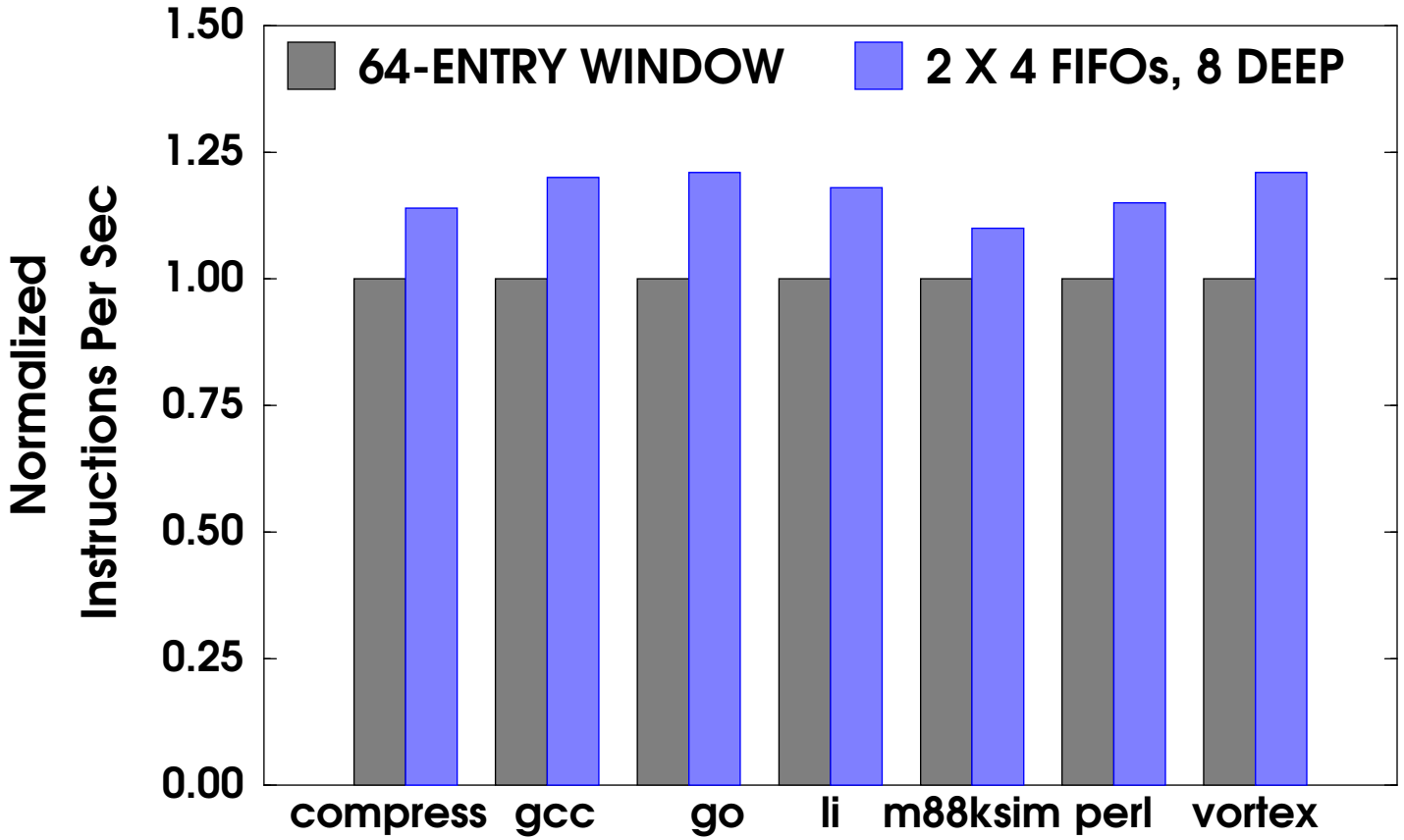
Performance results - IPCs



- 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 $\propto$ (IPC $\times$ Clock speed) !

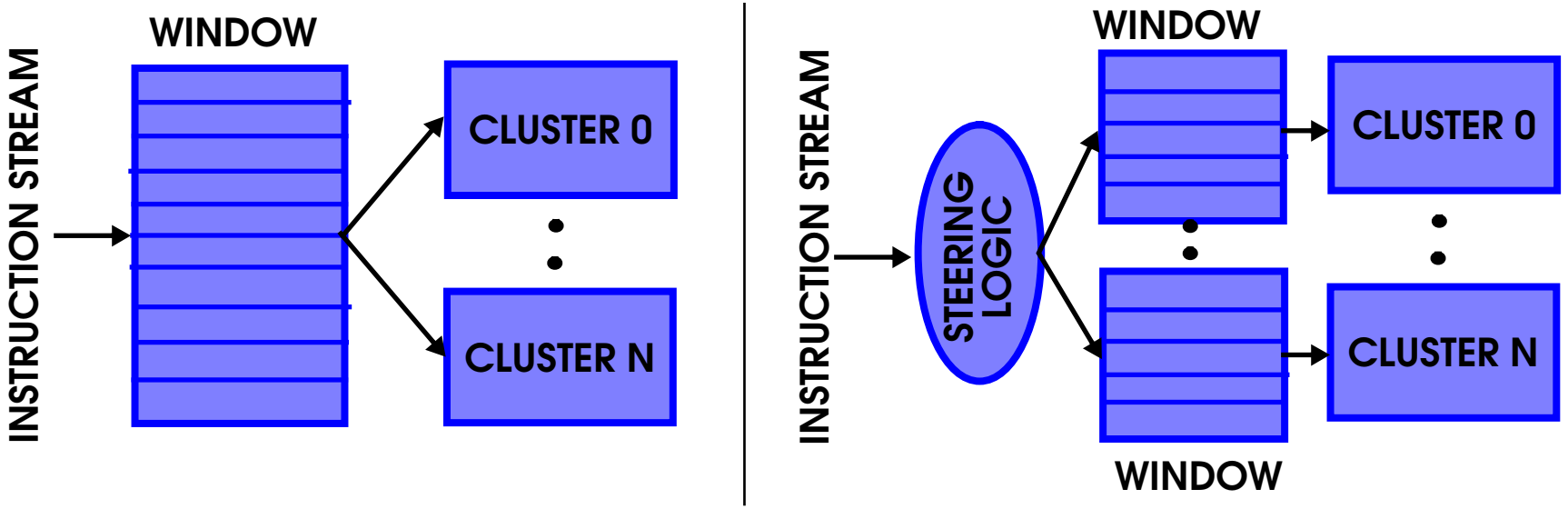
Performance results - Normalized Instructions Per Sec.



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

  Average performance improvement: 16%

Other clustered microarchitectures



Single window

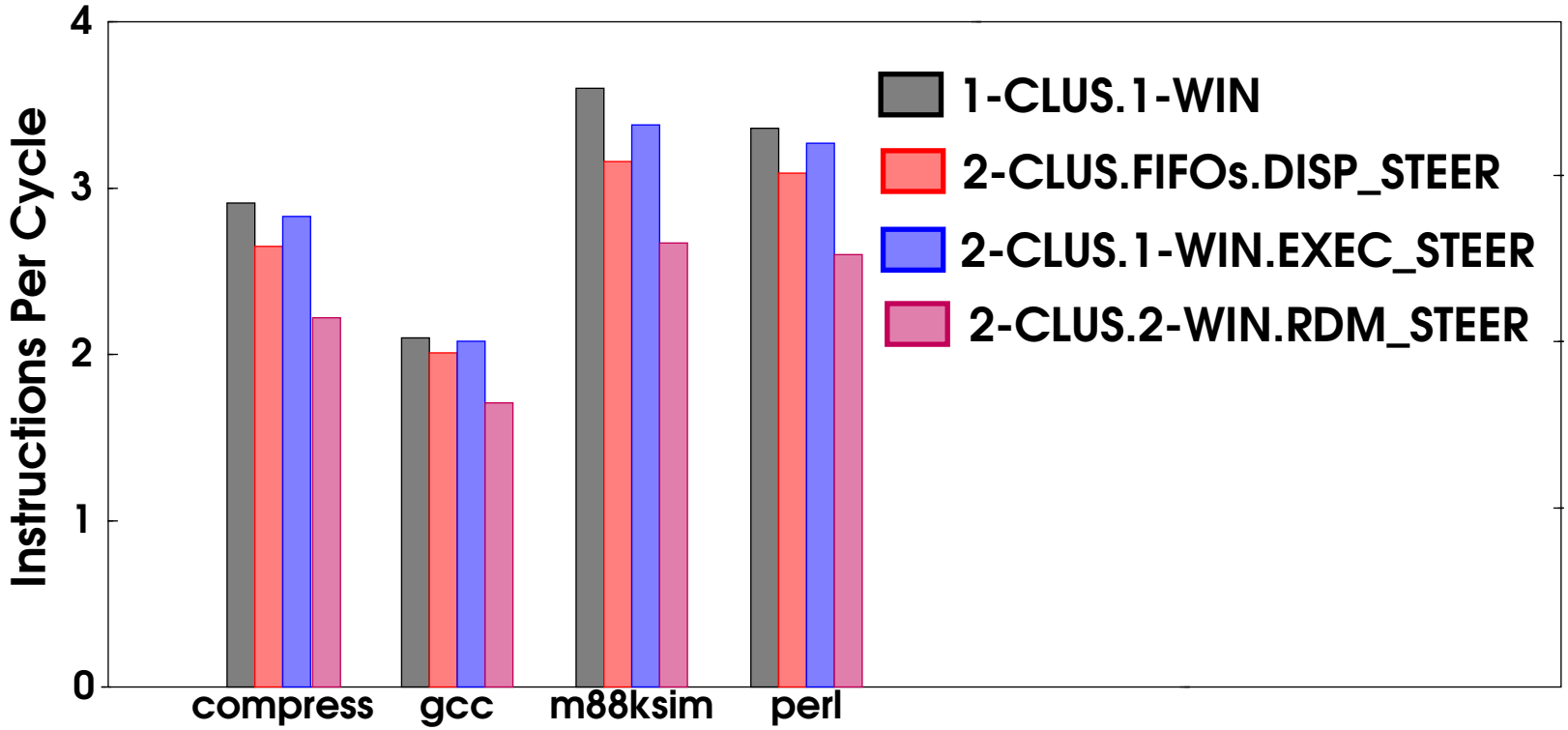
Execution steering

Mutliple windows

Dispatch steering

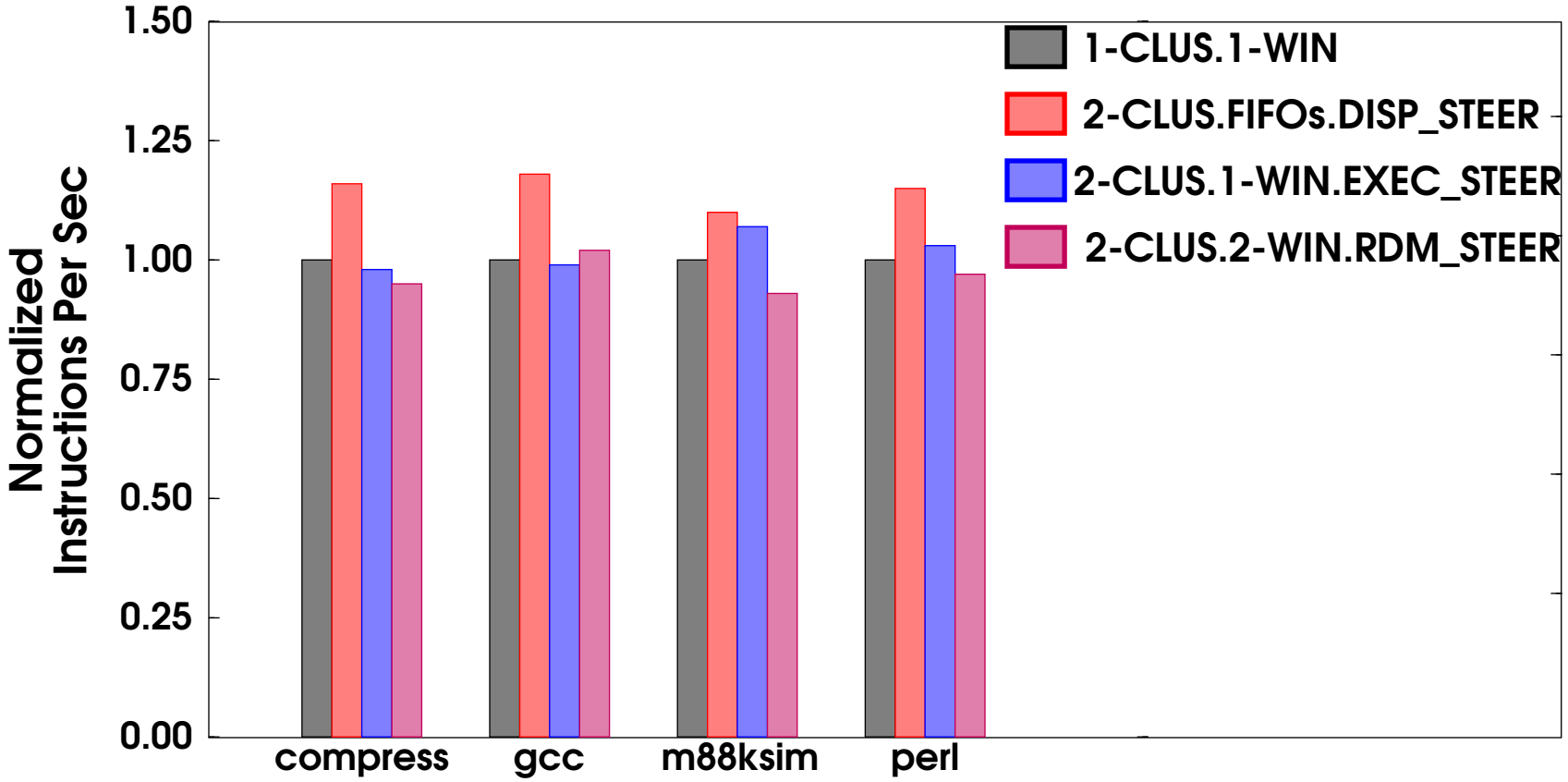
Extra cycle for inter-cluster bypasses

Performance results - IPCs



- Execution steer ing achieves high IPCs
  but steering is in critical issue path
- Random steer ing consistently performs worst
  17% to 26% IPC degradation

Performance results - Normalized Instructions Per Sec



- Dependence-based micr oarch performs best
- Random steer ing performs worse even w/ fast clock

Conclusions

- Cycle time is a cr ucial performance factor
- Detailed modeling essential

- Bypasses ar e critical performance issue

  clustering can help considerably
- Then, windo w 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 b y microprocessor vendors
  - ISSCC proceedings, DEC engineers
  - Studied alternatives for some structures
- Man y 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?

- Fr ont 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
- Register s
  - Pipeline
  - Tullsen et. al. report only 2% degradation in IPC

Using buffers to reduce wire delays

- Y es, buffers can reduce delay
  - but delay is still at least linear
  - buffers add delay and consume power
- W ires with multiple drivers need bidirectional buffers
  - not easy to switch direction fast enough
- Quadr atic 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 pr oblem only resurfaces at a smaller feature size

Steering logic complexity

- Can be done in par allel with rename
- Might need an extr a pipestage

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