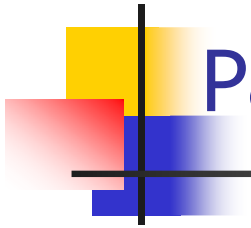
Register Integration: A Simple and Efficient Implementation of Squash Reuse

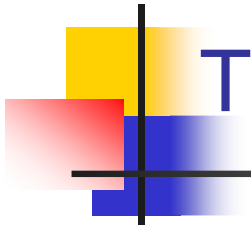
Amir Roth and Gurindar S. Sohi
University of Wisconsin-Madison

MICRO-33
Dec. 12, 2000



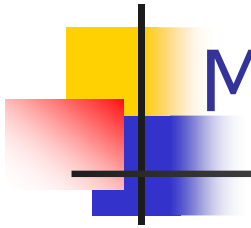
Parsing the Title

- Squash: mis-speculation? abort sequentially later work, fixup, resume
  - Problem: re-execute (squashed) mis-speculation independent work
- Reuse: salvage useful squashed results, don't re-execute instructions
- Implementation: reuse by writing saved value into register
  - determine instruction reusability: value-comparison/invalidation
- Register Integration:
  - "Recognize" and "un-squash" results from physical register file
  - Efficient: more natural "fit" for squash reuse
  - Simple: no need to read/write register values



Talk Outline

- Motivation and logical basis
- Working example
- Some implementation details
- Short performance evaluation



Motivation

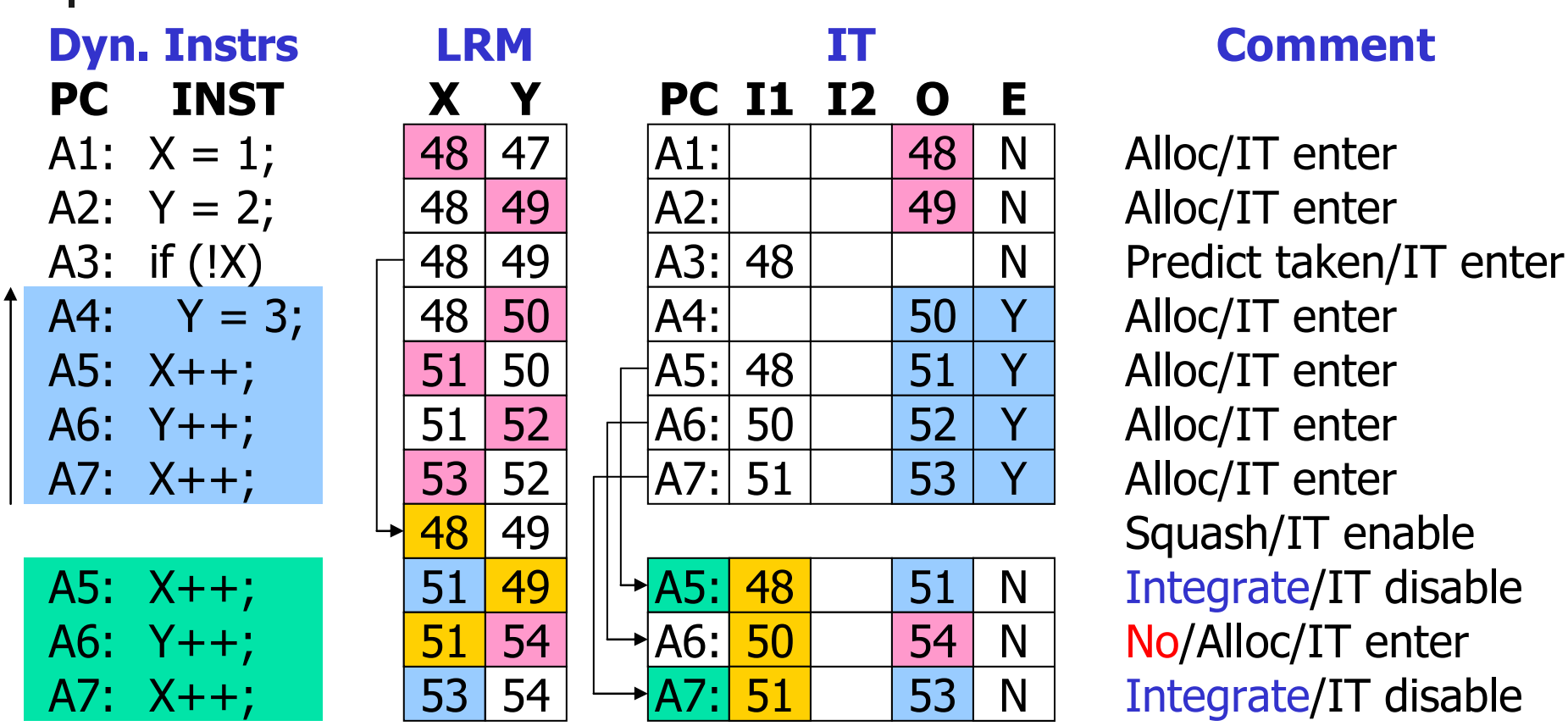
- Assume Unified Physical Register File (PRF)
  - Logical Register Map (LRM) sequentially "manages" PRF
- Conventional mis-speculation recovery
  - PR values intact
  - LRM restored to prior state, PR's become "garbage"
- "Conventional" reuse
  - Allocate new PR, write value into it
- Register Integration: why write? value is already in PR
  - To reuse: allocate PR holding squashed result to new instruction
  - Modify register-renaming to do this

Logical Basis for Integration

- Key: must locate PR holding squashed value
  - Use a second mapping of PRF
  - A second LRM? No
    - Implicitly sequential, can't be "searched" using right criteria
  - Integration Table (IT): describe each PR using creating instruction
    - Operation (PC) and input PR's
    - Valid after squash (valid always)
    - Encodes "reusability criteria"

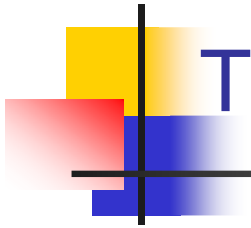
- Renaming + Integration
  - Rename an instruction, use LRM to find input PR's
  - Search IT for PR created by same instr. (PC) with same input PR's
  - Find one? Inputs haven't changed since squash! Integrate!

1 Picture == 4KB



E = Eligible (can be integrated)

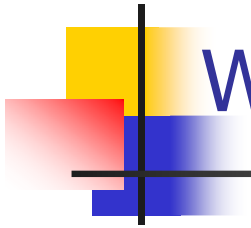
PR cannot simultaneously be mapped by two active instructions



The Tao of Integration

- Definition of "reusable" instruction: inputs unchanged since squash

- Exactly the information IT encodes
  - PR tags naturally track data-dependences (input changes)
  - Instructions integrated iff data-dependences intact
  - No need to read/compare values to perform reusability test
  - No separate invalidation/dependence-tracking mechanism

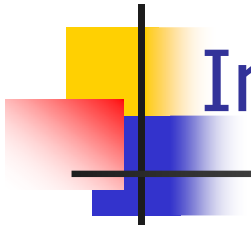


What Integration (Reuse) Accomplishes

- Improved performance (first-order effects)
  - Integrated instructions are complete*
  - Collapses data dependences
    - Chains of dependent instr's can be integrated in a single cycle
  - Integrated mis-predicted branch recovery begins immediately

- Reduced resource consumption/contention
  - No reservation-stations/scheduling/execution/writeback
  - Faster branch resolution reduces fetch demand

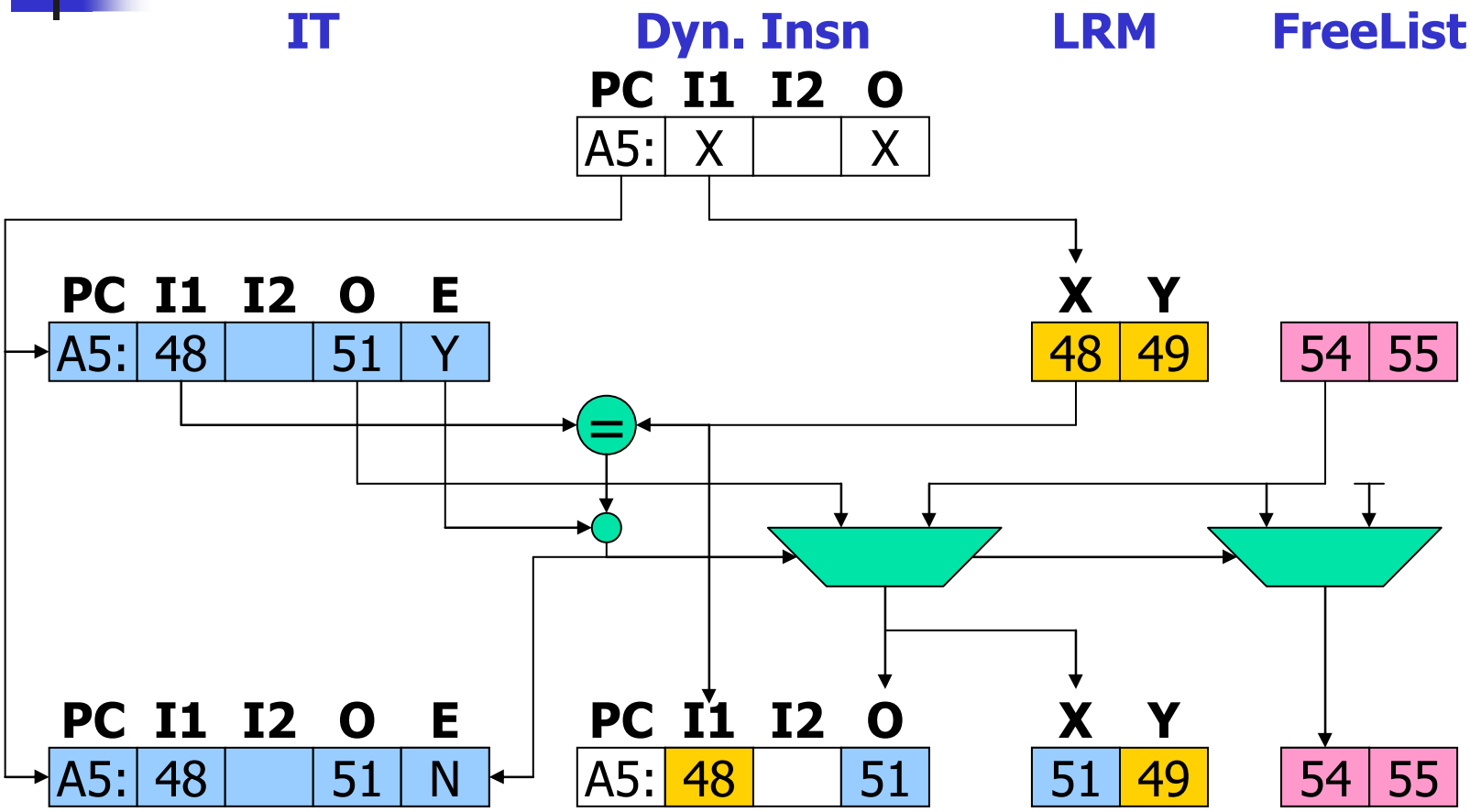
*Choose to integrate only completed instructions
  - Simplifies things, doesn't reduce benefit

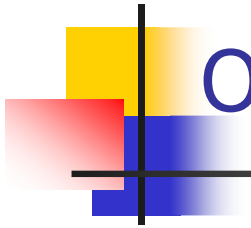


Implementation Details

- Requirements of base microarchitecture
  - Unified PRF
  - Support for load speculation (see why soon)
- Changes/Additions
  - IT
  - Integration circuit (added to renaming, next slide)
  - More PR's (keep squashed results alive longer)
  - Data-paths to LoadQ, StoreQ (see why soon)
- Non-changes
  - No datapaths to read/write PRF

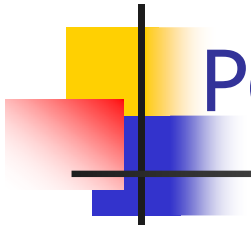
Integration Circuit





Other Implementation Issues

- Superscalar integration? Sure
  - Same parallel prefix formulation as "plain" renaming
  - Check N^2 dependences for N instructions (PR, not LR)
  - N^2M^2 if IT is M-way set-associative
- Integrating loads
  - PC + PR's not enough, previous stores are implicit inputs
  - Mis-integration: load integrated despite conflicting store
  - Add address/value fields to IT, save-from/restore-to LoadQ
  - Load speculation mechanism handles conflict after integration
  - "Snoop" IT for conflicts before integration
- More details in paper

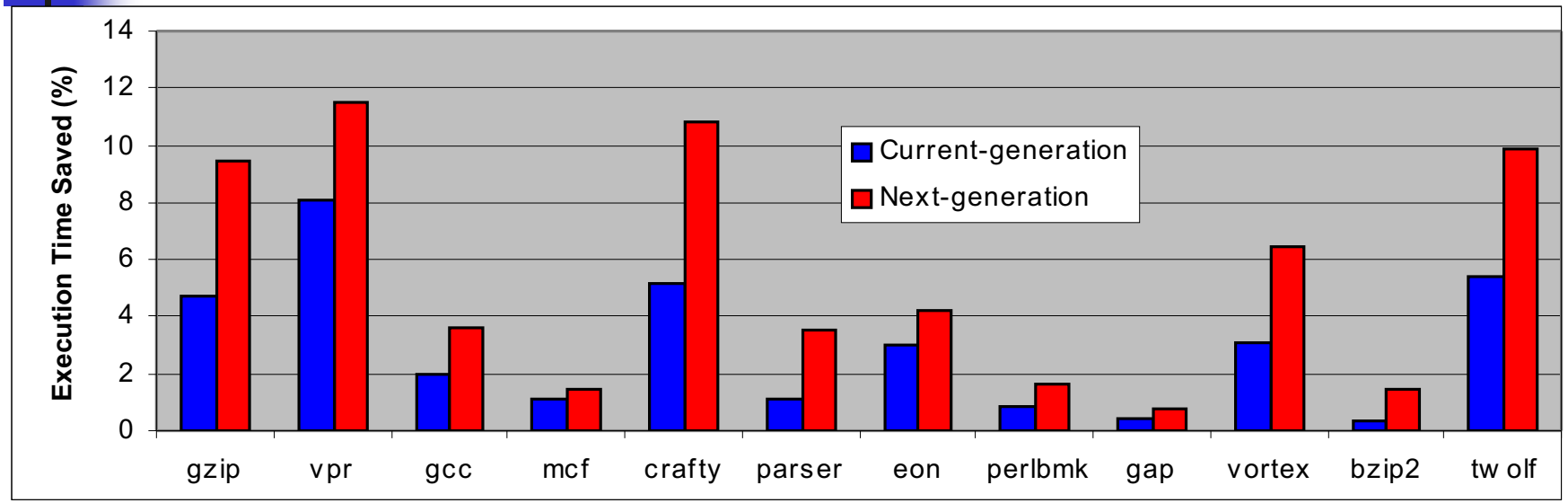


Performance Evaluation

- SPEC2000 benchmarks, Alpha EV6, -O2 –fast
- Simplescalar simulator

- 8-wide superscalar, OoO, speculative, load speculation
- 256-entry, direct-mapped IT, #PR's = 64+ROB+256
- 32KB 2-way I-Cache, 64KB 2-way D-Cache, 1MB 4-way L2
- 2 base pipeline configurations
  - Current-generation:
    - 128 ROB (448 PR's), 64 LoadQ, 32 StoreQ
    - Pipe: 3 fetch, 2 decode/rename, 2 schedule/reg-read, 3 load
  - Next-generation: (faster clock, 2MB L2)
    - 256 ROB (576 PR's), 128 LoadQ, 64 StoreQ
    - Pipe: 5 fetch, 3 decode/rename, 4 schedule/reg-read, 4 load

Performance vs. Base Microarchitecture



- Integration more effective as microarchitecture more aggressive
  - More speculative buffering+longer pipe:
    - more instructions completed along mis-speculated paths
    - more integrated instructions
  - Deeper pipeline, each integrated instruction saves more work

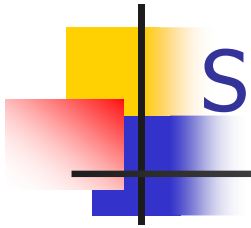
A Closer Look

- Current generation microarchitecture, every second benchmark

	Vpr	Mcf	Parser	Perl	Vortex	Twolf
Integrated/committed (%)	15.9	6.1	6.5	4.7	1.6	8.6
Integrated/squashed (%)	46.7	24.0	28.3	22.4	7.3	41.4
Fetched instr. saved (%)	6.6	3.7	1.9	1.1	4.8	4.8
Executed instr. saved (%)	15.3	7.0	5.6	4.4	15.1	9.2
Execution Time Saved (%)	8.1	1.1	1.1	0.9	3.1	5.6

- 4-15% reduction in instructions executed, 1-7% in fetched
  - Performance correlated with fetch reduction
  - Integrated instructions still fetched (leave "bubbles")

- Some other results
  - IT size matters a little, IT associativity less (thankfully)



Summary

- Integration: new implementation of squash reuse
  - Based on data-dependences, not values/invalidations
  - Reuse: improves performance, reduces resource contention
  - Simple: requires only LRM manipulations, no PR reads/writes
  - Efficient: implementation matches definition of reuse