Architectural Core Salvaging in a Multi-Core Processor for Hard-Error Tolerance

Michael D. Powell, Arijit Biswas,
Shantanu Gupta[†], and Shubu Mukherjee

SPEARS Group,
Intel Massachusetts

[†]EECS,
University of Michigan

Motivation

- Hard errors in logic are an increasing risk
- Errors manifest at manufacture time or in field:
  - Manufacture: more cores, bigger die -> lower yield
  - Field: wearout failure

- Large SRAMs (cache) regular, easily protected
  - Manufacture: spare lines, field: line disable
- Remainder of die (cores) not as easily protected
  - Focus on manufacture, but same applies to field

How do we tolerate core defects?

Tolerating defective cores

- Defective core options: ***disable*** or ***salvage***
  - Disabling wastes entire core even for minor defect
  - Salvaging uses *redundancy* to maintain correctness

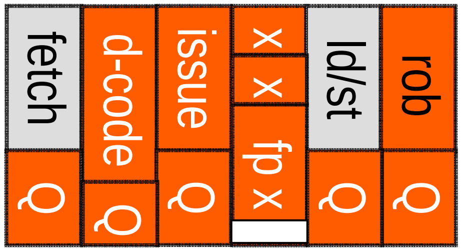
- Salvage by using redundancy to tolerate a defect
  - ***μarchitectural:*** use another resource in the core
  - ***Architectural:*** use another core

Architectural salvaging covers against more defects

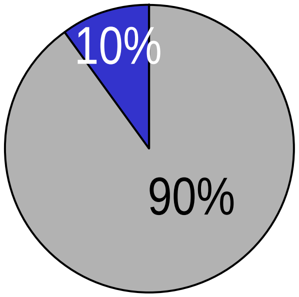
µarchitectural salvaging:

- Natural method of defect-tolerance
  - Both others and we have studied it
- Protects only resources w/ intra-core redundancy
  - Small-array entry: other entries
  - Execution logic: other logic w/ same function

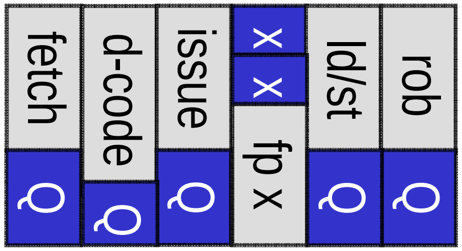
perceived µarch salvaging area coverage



µarch salvaging area coverage limit



Core area

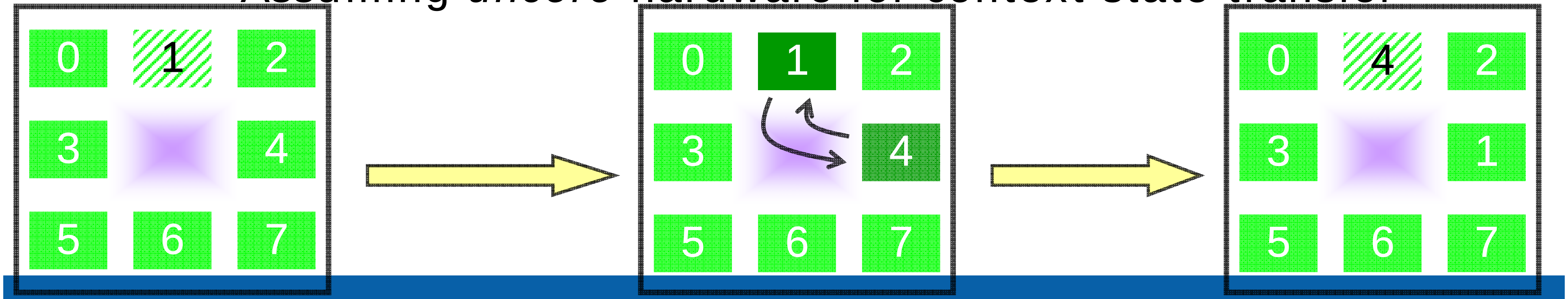


–Difficult to cover >> 10% of core area

Coverage not as much as might be expected

Architectural core salvaging

- Key observation
  - In CMP, *die* must support all instructions, but
  - individual *cores* need *not* support all instructions
- Architectural salvaging
  - Threads can be dynamically migrated (swapped) between cores to guarantee progress
    - On demand context switch (CS)
    - Cores with critical defects in exec. can still be used
    - Assuming *uncore* hardware for context state transfer

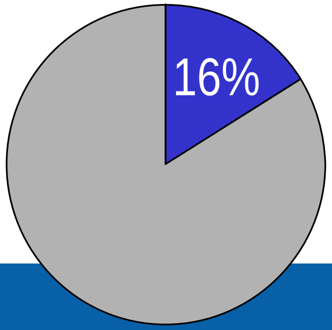


Low overhead if defective units used infrequently

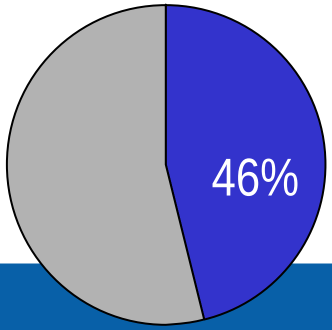
Architectual Core Salvaging Contributions

- Better performance than core disabling
  - Most workloads get useful work from defective core
- Exploit architectural redundancy
  - exceed limitations of μarch. redundancy
- Cover > portion of core w/ less invasive technique

μarch salvaging: max exec-unit coverage



arch salvaging: demonstrated exec-unit coverage

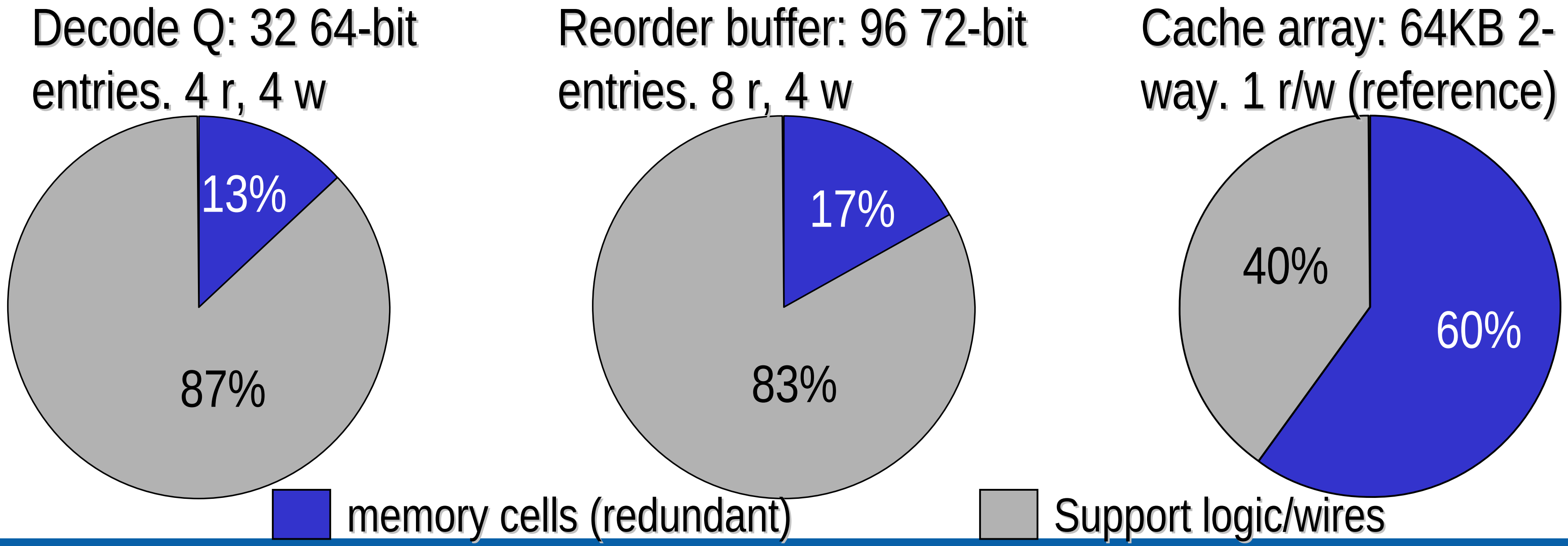


Outline

- Introduction
- Limitations of µarchitectural salvaging
- Architectural salvaging
- Methodology and performance results
- Conclusion

μarch salvaging: small arrays

- Small RAMs, CAMs occupy substantial core area
  - Buffers, queues, regfiles: too small for spare arrays
  - May protect using spare entries or by reducing size
- Covers only memory cells; not decoder, mux, sense amp
- memory-cell fraction decreases w/ array size


Decode Q: 32 64-bit entries. 4 r, 4 w

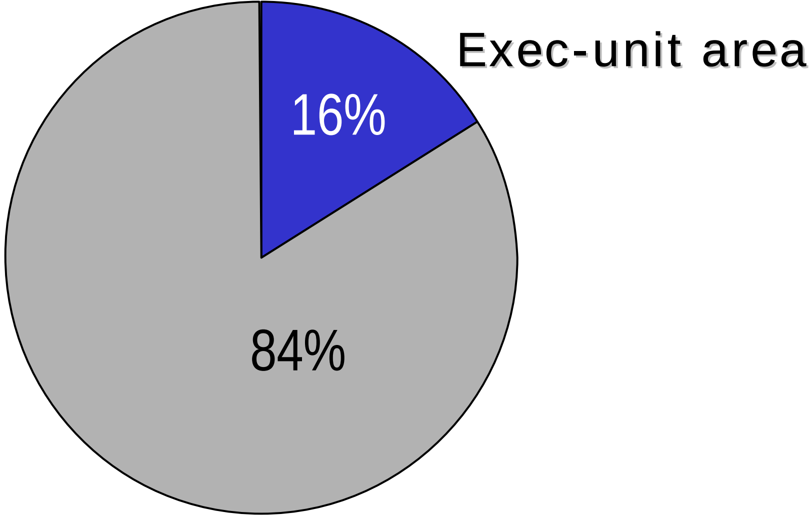
Reorder buffer: 96 72-bit entries. 8 r, 4 w

Cache array: 64KB 2-way. 1 r/w (reference)

Area truly covered can be deceptively low

μarch salvaging: execution units

- Many instruction classes are replicated
  - Canonical redundancy example; superscalar hallmark
- But less redundancy than might be expected in IA
  - Non-replicated instruction may share structure
  - Instruction replication != structure redundancy
- 16% of exec area μarch. redundant



Most exec area is for non-replicated instructions

μarch salvaging summary

- Small-array + exec coverage:
  - ~10% of non-cache core
- Not enough μarch redundancy for high coverage
- Each structure requires its own salvaging hardware
- Other redundancy needed to obtain high coverage

Outline

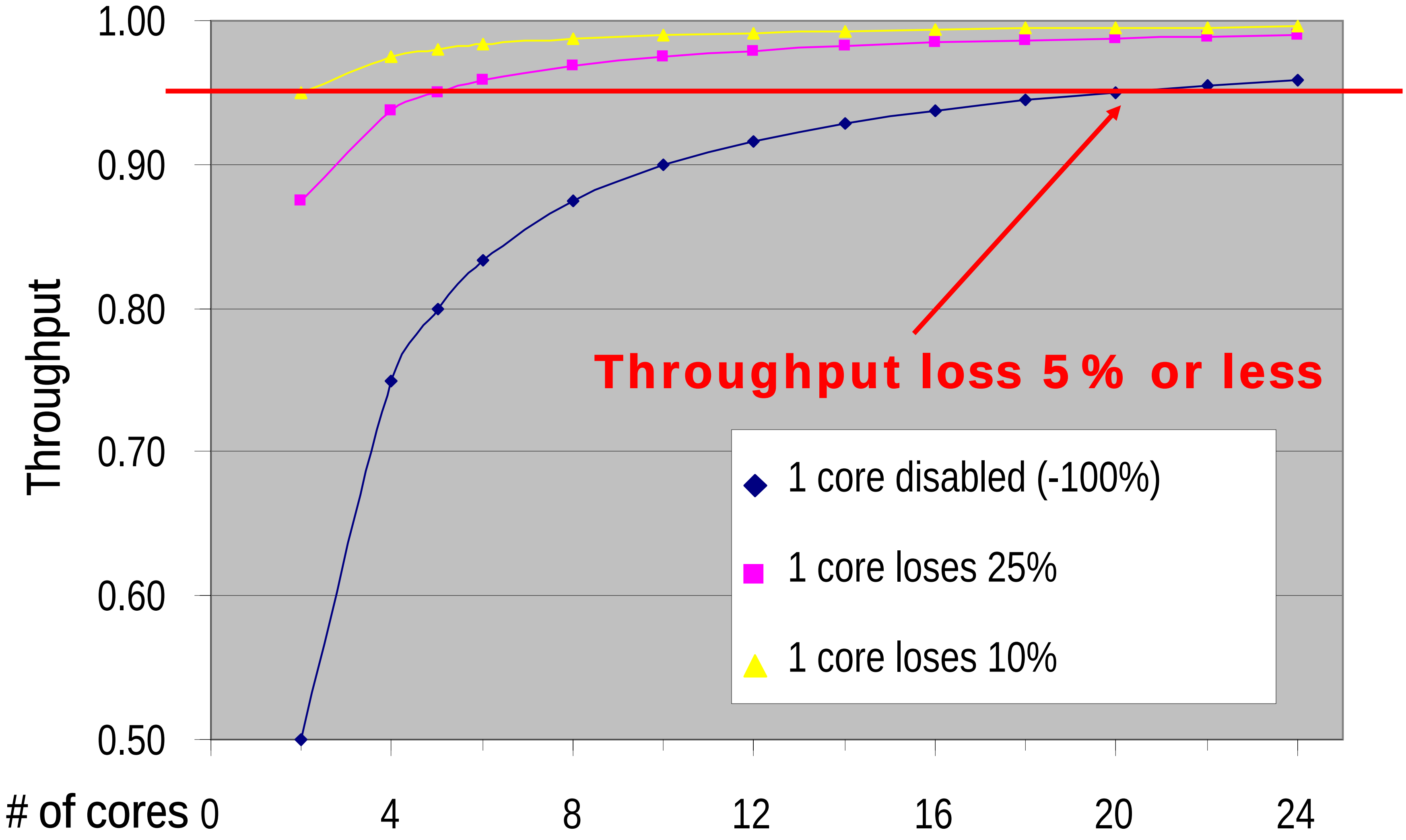
- Introduction
- Limitations of μarchitectural salvaging
- Architectural salvaging
- Methodology and performance results
- Conclusion

Architectural Salvaging

- Other cores provide redundancy, cover defects
  - Each core needs to know its defects
  - If thread needs defective resource:
    - Trap and migrate to another core
  - O/S and user transparency
    - APIC ID swapped between cores along with thread
    - Migration occurs using h/w C6 power-state array (few KB)
- Overhead and performance
  - If defective resource used frequently by *all* threads
    - Fall back to core disabling to avoid migration thrashing
    - Places upper bound on performance loss

What is design space (# of cores) for arch. salvaging?

Core salvaging: simple perf. model



Salvaging makes sense for CMPs >5 cores

Arch. salvaging: targeted instructions

- “Infrequent” instructions
  - Those used by only some applications
  - Or used in most applications, but only a few times
  - E.g., certain floating point & SIMD instructions
- Disallow salvaging “critical” instructions
  - Load, store, simple int ALU, branch
  - Defect in executing critical inst. -> disable core
- Structures used by only infrequent instructions are large fraction of execution-unit area

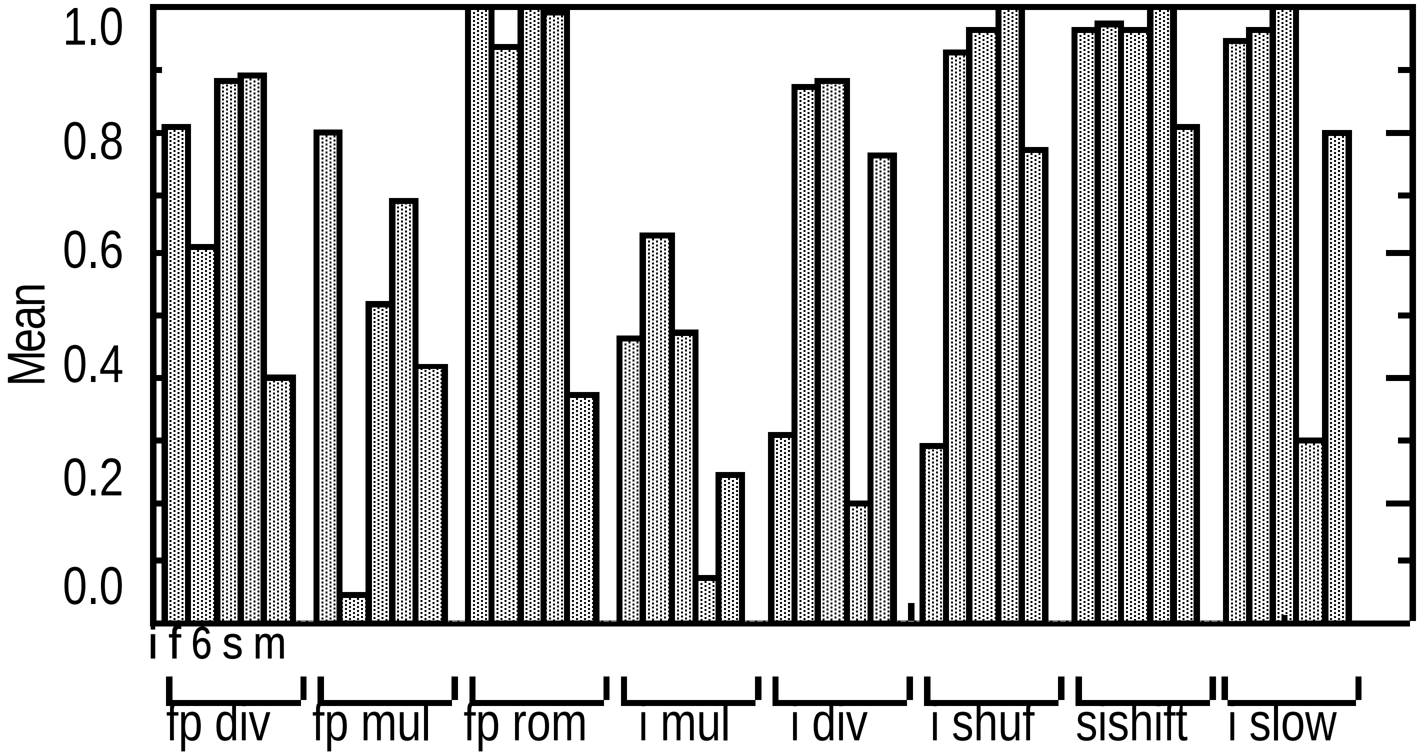
Are there enough infrequent instructions?

Instruction Occurrence

Fraction of non-overlapping 100K-inst. windows that do not contain an instruction class for 5 workloads

i: spec int 2K **f**: spec fp 2K **6**: spec 2006

s: server **m**: multimedia



Many (large-area) instructions quite infrequent

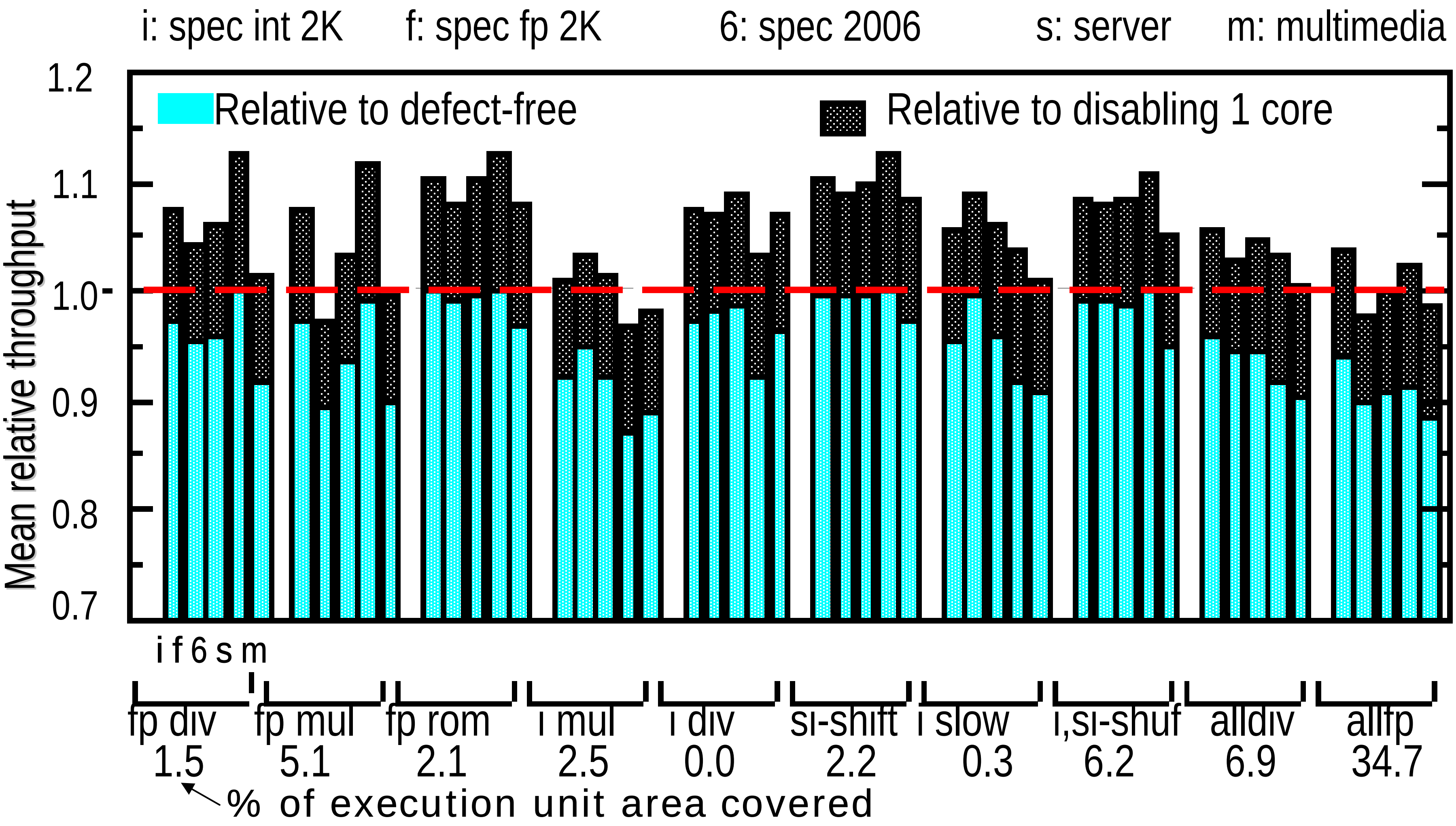
Outline

- Introduction
- Limitations of μarchitectural salvaging
- Architectural salvaging
- Methodology and performance results
- Conclusion

Methodology

- Modeled architecture: Intel® Core-2™ like
  - 8 cores; 8 MB shared last-level-cache
  - 4-issue out-of-order
  - Each core: 64KB i-cache, 64KB d-cache, 1MB L2
- Assume 1000-cycle thread migration overhead
  - Fall back to disabling for 150K cycles if > 2 migrations in 40K cycles
- Workloads: spec00, spec06, server, multimedia

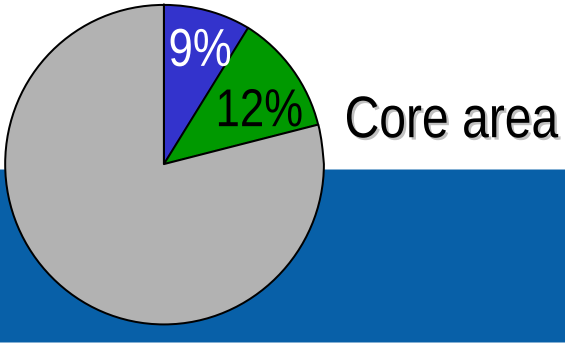
Core salvaging performance
(8 core die)

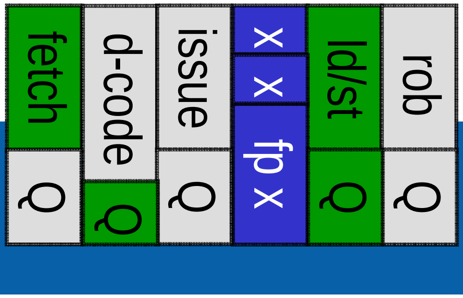


Average 5-7% better throughput than disabling

Architectural salvaging coverage

- Execution-unit case study:
  - uarch covered @ max 16% of execution-unit area
  - We show proof-of-concepts for arch. covering 46%
  - Accounts for 9% of vulnerable core area vs 3%
- Core level:
  - μarch covered max 10.6% of core
  - Arch covers nearly that much in exec. units
  - Combine exe w/ hybrid h/w salvaging (shown in paper), cover 21% of vulnerable core area





Outline

- Introduction
- Limitations of μarchitectural salvaging
- Architectural salvaging
- Methodology and performance Results
- Conclusion

Conclusions

- Hard errors in logic are an increasing risk
- Architectural vs μarch. core salvaging
  - Cover > portion of core w/ less invasive technique
  - Cover 46% of execution units vs 16% for μarch
  - Covered exec units: 9% of vulnerable core area
- Apply salvaging at manufacture or in field
- Better performance than core disabling
  - Core with minor defect -> nearly full performance

Architectural Core Salvaging in a Multi-Core Processor for Hard-Error Tolerance

Michael D. Powell, Arijit Biswas,
Shantanu Gupta[†], and Shubu Mukherjee

SPEARS Group,
Intel Massachusetts

[†]EECS,
University of Michigan