Power-Efficient Redundant Execution for Chip Multiprocessors

P. Subramanyan[1] V. Singh[1] K. K. Saluja[2] E. Larsson[3]

[1]Indian Institute of Science, Bangalore, India
[2]University of Wisconsin-Madison, USA
[3]Linköping University, Sweden

3rd Workshop on Dependable and Secure Nanocomputing
29 June, 2009

Outline

Outline

Introduction

The Reliability Problem

"Future designs will consist of 100 billion transistors, 20 billion of which are unusable due to manufacturing defects; 10 billion will fail over time due to wear-out, and regular intermittent errors will be observed."

[Borkar, 2005]

A system of 2048 HP AlphaServer ES45s was affected by an average of 24.0 parity errors every week determined to be caused by radiation strikes.

[Michalak et al., 2005]

The Reliability Problem

Moore's law is expected to apply for the next 10 years giving us smaller and faster devices with reduced power. But, there is a downside:

- Smaller devices make ICs more susceptible to transient faults
- Wearout and drift effects more prominent
  - e.g., negative bias temperature instability (NBTI), electromigration (EM), gate oxide integrity (GOI)
- Increased process variations also causing faults

The upshot of decreased reliability is the need for architectural mechanisms for fault tolerance.

Requirements of a Hardware Reliability Solution

- Reliability mechanisms need to have low cost
  - small area overhead
  - low performance overhead
  - small additional power consumption
- Mechanism must be configurable at runtime
  - Switched off for users who do not require reliability
  - Switched off for applications that are inherently redundant
- Transparent to software
  - *i.e.*, must work with existing software

The Power Problem

- Power and peak temperature are key performance limiters in CMPs [Isci et al., 2006]
  - Since power budget for a chip is fixed, decreasing the power for a single core increases available power for, and hence performance of, other cores
- Decreasing operating temperatures leads to a significant increase in device reliability [Parulkar et al., 2008]
  - Decreasing temperature from $105\,^{\circ}\mathrm{C}$ to $66\,^{\circ}\mathrm{C}$ increased GOI time-to-breakdown by a factor of 9; average failure rate reduced by a factor 17
  - NBTI degradation decreased by 29%, equivalent to an eight-fold increase in lifetime

Moral Methods for decreasing the power overhead of fault tolerance mechanisms are required

Outline

Fault Tolerance Based on RMT

Several approaches based on Redundant Multithreading (RMT);

- Examples: AR-SMT, SRT, CRT[1]
- Redundancy by running two copies of the same program
- Performance overhead is between $20 - 30\%$

But base assumption is a SMT processor:

- Lee and Brooks [2005] study efficency of SMT
  - Efficient SMT requires wide and deep pipeline
  - Leads to higher area, power requirement
- Comparison with CMPs shows that [Li et al., 2005]:
  - CMPs better for CPU bound workloads
  - SMT better for memory bound workloads
- Interference between threads [Mathis et al., 2005]

[1] [Mukherjee et al., 2002; Reinhardt and Mukherjee, 2000; Rotenberg, 1999]

Fault Tolerance Based on CMPs: Fingerprinting

Question: When to do output comparison?

- Chip external pins comparison increases error detection latency
- Comparing cache/register updates requires large bandwidth

Idea: Compress hash of register updates, load/store addresses, store values using CRC codes to obtain a fingerprint and compare fingerprints instead.

- Comparison bandwidth is minimal
- Error detection latency is also very small

[Smolens et al., 2004]

Summary of Related Work

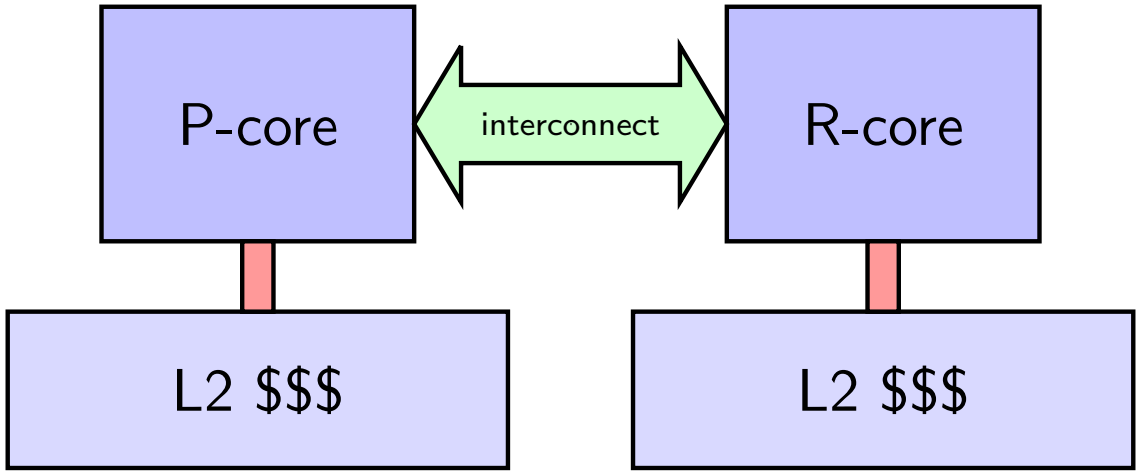
Existing Solutions Have Some Drawbacks:

- RMT has low power/performance costs, but based on SMT which might not be desirable for all applications
- Fault tolerant architectures based on CMPs like Fingerprinting, Reunion and others all have a power overhead that is nearly 100%

Therefore, this paper investigates the design of a fault tolerant architecture based on CMPs which has a low power and performance overhead.

Outline

Overview



- Two cores execute the same instruction stream
- Dedicated interconnect between each pair of cores
- One core designated as primary (P-core) and another as redundant (R-core)

Overview (contd.)

P and R cores execute the same instruction stream, and appear as a single logical processor to the application.

Input Replication:

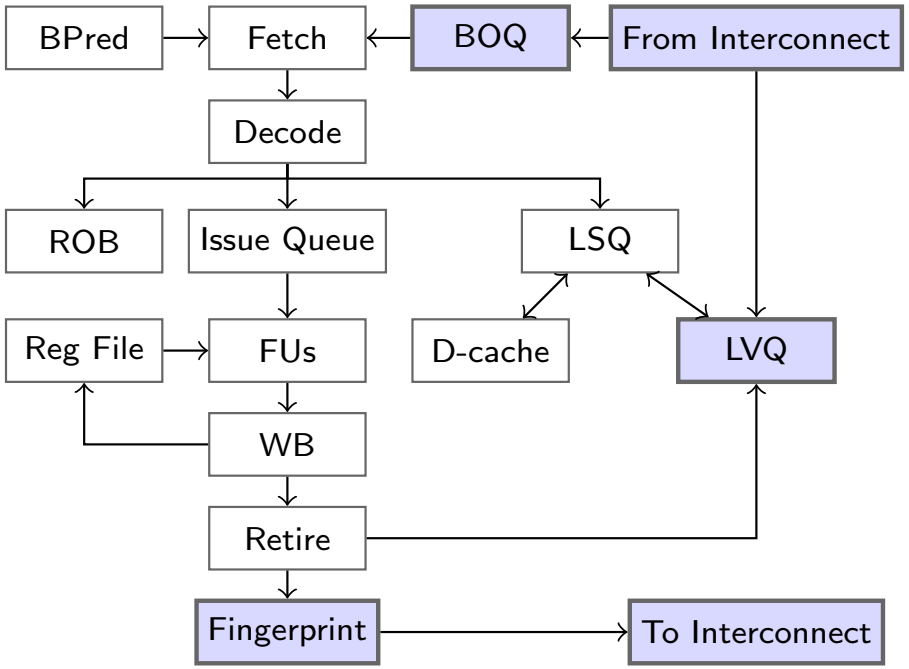
- P-core accesses memory hierarchy, R-core does not
- P-core transfers load values to R-core over interconnect
- R-core stores values in load value queue (LVQ)[2]
- R-cores accesses LVQ instead of data cache

Output comparison:

- Register updates, branch targets, load and store addresses store values hashed using CRC code to generate fingerprint[3]
- Cores compute and exchange fingerprints to detect errors

[2] Introduced by Reinhardt and Mukherjee [2000]
[3] Introduced by Smolens et al. [2004]

Block Diagram



Newly added structures are shaded and not used when redundant execution is not performed.

- Branch outcomes from P-core stored in BOQ of R-core
- Used instead of branch predictor, BTB in R-core
- Fetching stalls if BOQ empty

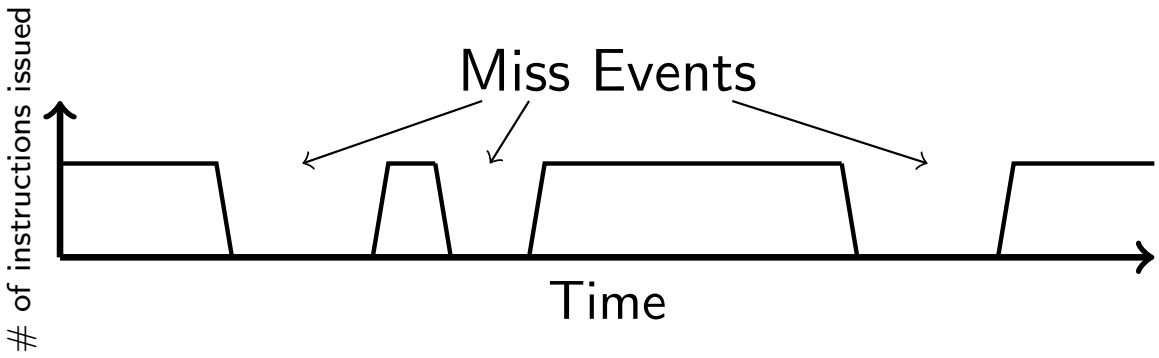
Outline

Interval Analysis: Introduction

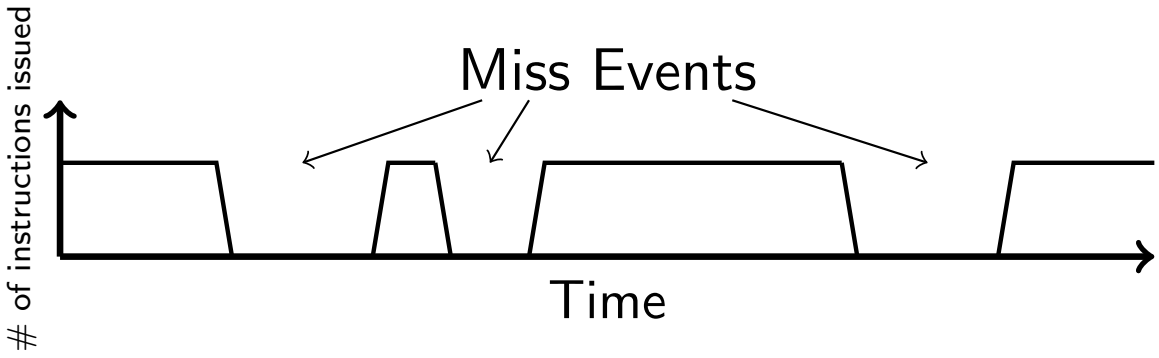
Interval Analysis

Performance of superscalar processors can be analyzed by dividing time into intervals between miss events.

[Eyerman et al., 2006]

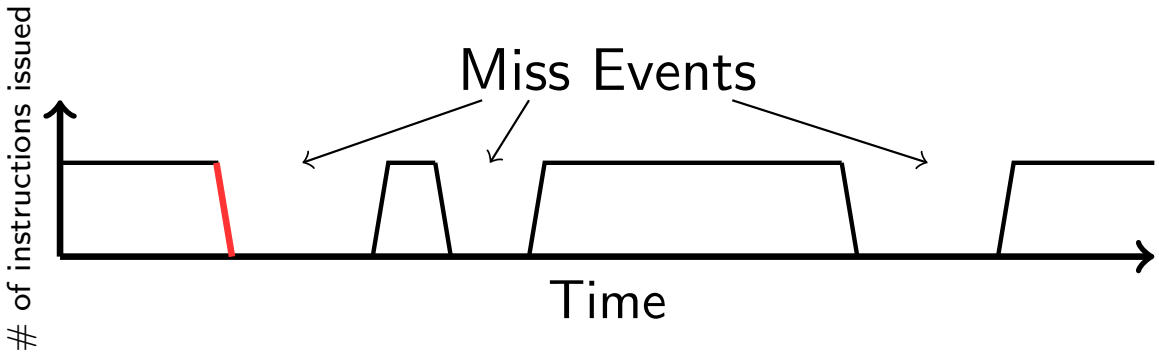


Interval Analysis: Introduction



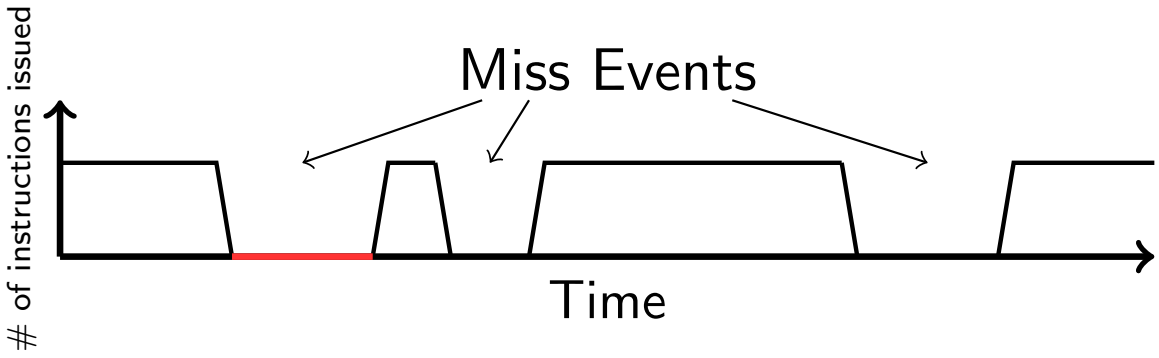
- Assumption: Superscalar processors are designed to stream instructions through pipeline at a rate equal to issue width
- This smooth flow of instructions is interrupted by miss events
- Examples of miss events: branch mispredictions, I-cache, D-cache misses, TLB misses, long latency instructions
- Effect of miss events:

Interval Analysis: Introduction



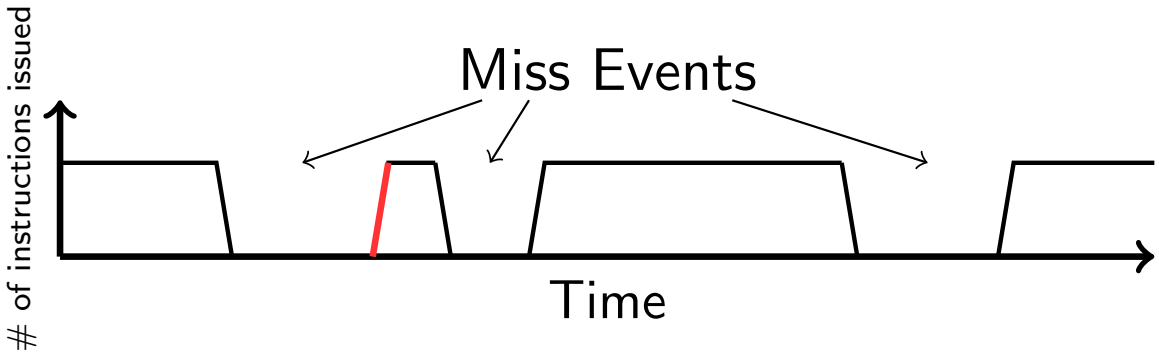
- Assumption: Superscalar processors are designed to stream instructions through pipeline at a rate equal to issue width
- This smooth flow of instructions is interrupted by miss events
- Examples of miss events: branch mispredictions, I-cache, D-cache misses, TLB misses, long latency instructions
- Effect of miss events:
  1. A decrease in the number of useful instructions issued per cycle

Interval Analysis: Introduction



- Assumption: Superscalar processors are designed to stream instructions through pipeline at a rate equal to issue width
- This smooth flow of instructions is interrupted by miss events
- Examples of miss events: branch mispredictions, I-cache, D-cache misses, TLB misses, long latency instructions
- Effect of miss events:
  1. A decrease in the number of useful instructions issued per cycle
  2. A period of no useful work

Interval Analysis: Introduction



- Assumption: Superscalar processors are designed to stream instructions through pipeline at a rate equal to issue width
- This smooth flow of instructions is interrupted by miss events
- Examples of miss events: branch mispredictions, I-cache, D-cache misses, TLB misses, long latency instructions
- Effect of miss events:
  1. A decrease in the number of useful instructions issued per cycle
  2. A period of no useful work
  3. Miss event is resolved and smooth flow of instructions resumes

Miss Events in the R-Core

What are the possible miss events in the R-core?

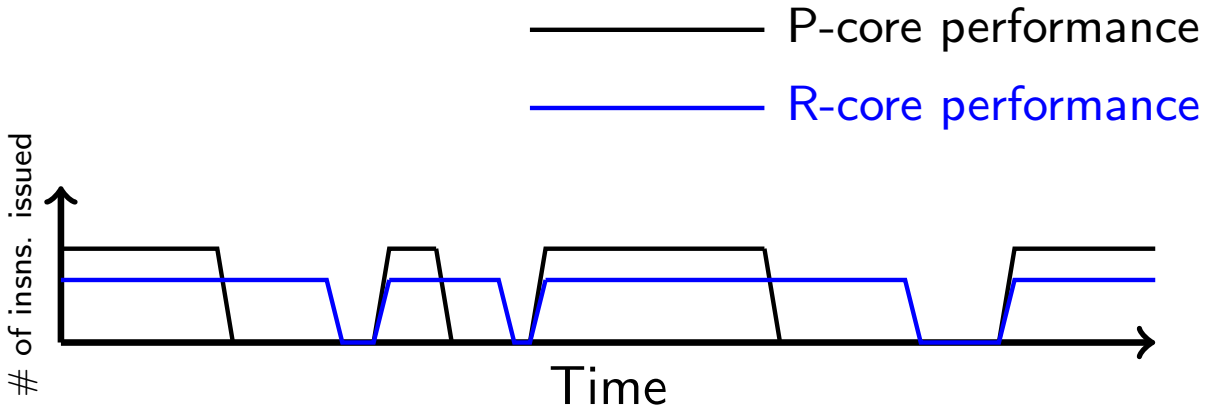
- No branch mispredictions, D-cache misses
- Only I-cache misses, TLB misses and a few others

New types of miss events:

- BOQ stall: BOQ empty, waiting for branch outcome
- LVQ miss: Load value not yet available
- Experiments indicate that above two dominate execution time

Observation: Resolution time of BOQ stalls and LVQ misses depends on P-core execution, not on R-core

DVFS in R-Core



Effect of scaling down frequency in R-core:

- Decreasing frequency of R-core increases interval length
- But miss event resolution times are unaffected
- Overall execution time need not increase if size of interval does not grow beyond the resolution time of the next miss event

Outline

Frequency Scaling

Idea

Run R-core at a lower frequency as compared to P-core; since miss event resolution times are controlled by P-core, we get a power benefit with only a small performance penalty.

Queue sizes can track program phase changes:

- Number of elements in LVQ and BOQ are a measure of how far "behind" the R-core execution is compared to P-core
- Growing queue size indicates R-core is slower than P-core
- Shrinking queue size indicates R-core is faster than necessary

DVFS Algorithm

This suggests a simple algorithm that uses queue sizes to adjust R-core frequency.

Algorithm

After every T_s seconds:

1. If $size(LVQ) > T_h$ or $size(BOQ) > T_h$:
   - Increase frequency and voltage
2. Else If $size(LVQ) < T_l$ or $size(BOQ) < T_l$:
   - Decrease frequency and voltage
3. Else do nothing

T_h is the high threshold and T_l is the low threshold. The algorithm tries to keep the queue sizes in between T_l and T_h.

Outline

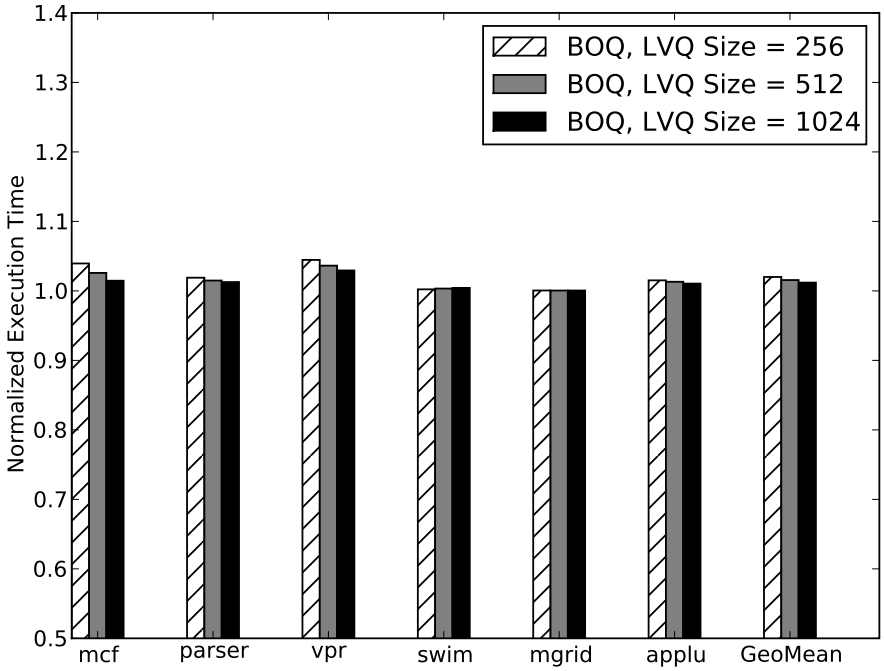
Methodology

- Modified SESC cycle accurate simulator
- Two phase simulation approach:
  - Run P-core first, collect trace of interconnect events
  - Run R-core next using generated trace
- Simple first order power model
  - Assumption: voltage scales linearly with frequency
  - Estimatation of energy consumed by R-core assuming activity is the same as P-core at reduced voltage/frequency
- Workload
  - `crafty`, `mcf` and `vpr` SPECint 2000 benchmarks with MinneSPEC reduced inputs [KleinOsowski and Lilja, 2002]
  - `ammp`, `mgrid` and `swim` SPECfp 2000 benchmarks and simulated Early SimPoints [Perelman et al., 2003]

Outline

Performance Overhead

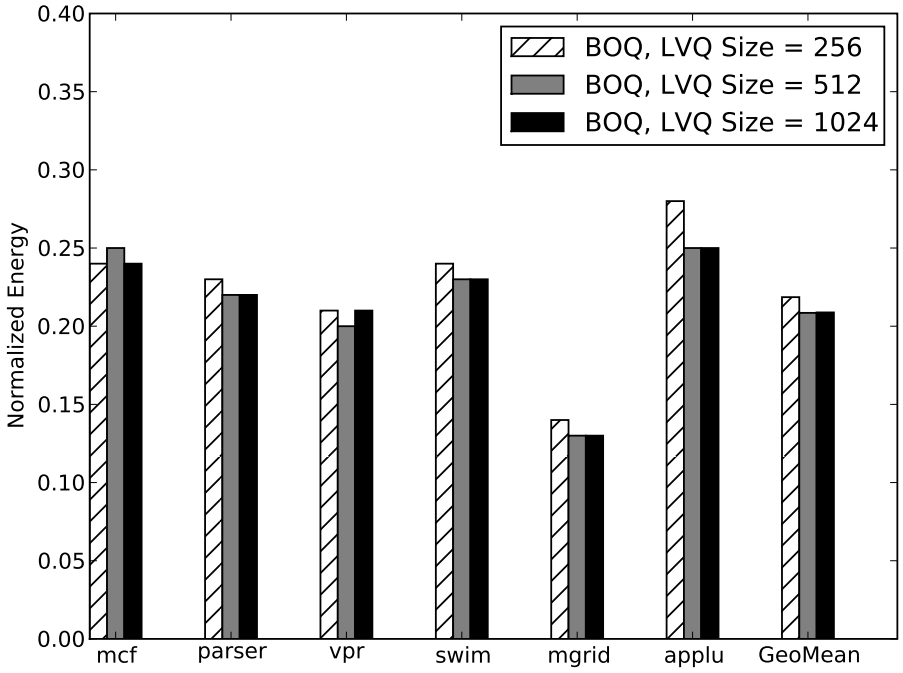
Normalized execution time = Exec. time of proposed architecture / Exec. time of non-fault tolerant execution



- Marginal improvement in performance with increased queue sizes
- Performance overhead is limited to a few percent in all cases

Energy Consumed By R-core

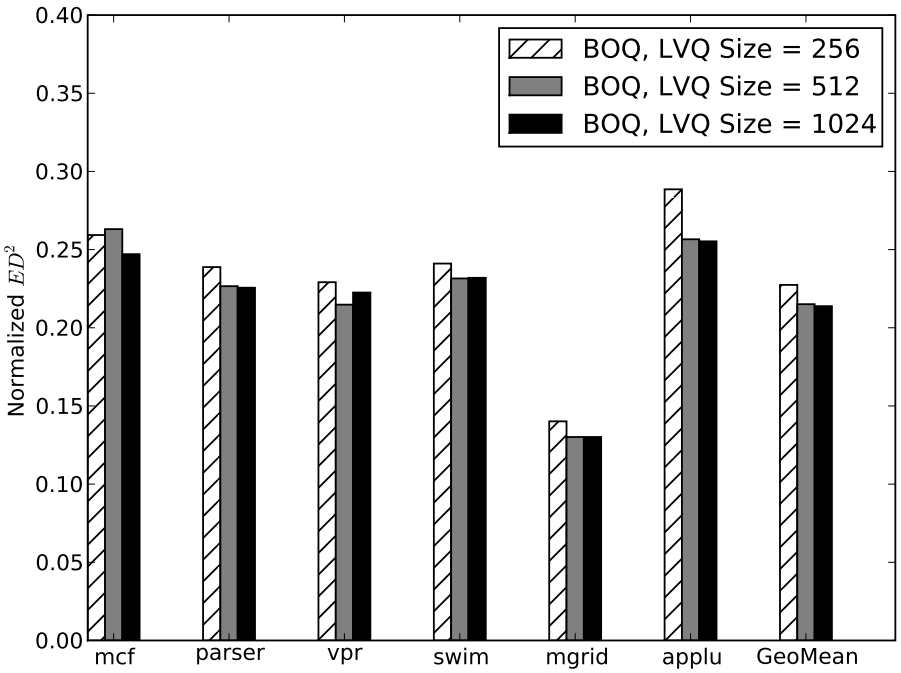
Normalized Energy = Energy of R-core / Energy of non-fault tolerant execution



- Energy overhead of redundant exection is less than 25%
- Queue sizes don't seem to have a significant effect.

ED^2 Product

$$NormalizedED^2 = NormalizedEnergy^2 \times NormalizedExecutionTime$$



ED^2 savings similar to the energy savings; mean value of about 76%.

Conclusion

An architecture for redundant execution with the following characteristics:

- Requires only minor modifications to existing CMPs
- Works with existing software
- Low performance overhead ($\sim 1\%$)
- Significant reduction of energy overhead ($\sim 77\%$)

Thank You!

S. Y. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6), 2005.

S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith. A Performance Counter Architecture for Computing Accurate CPI Components. *Proceedings of the 12th ASPLOS*, 2006.

C. Isci, A. Buyuktosunoglu, C-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. *Proceedings of the 39th MICRO*, 2006.

A. J. KleinOsowski and D. J. Lilja. MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research. *IEEE Computer Architecture Letters*, 2002.

B. Lee and B. Brooks. Effects of Pipeline Complexity on SMT/CMP Power-Performance Efficiency. *Workshop on*

Complexity Effective Design in conjunction with 32nd ISCA, Jun. 2005, 2005.

Y. Li, D. Brooks, Zhigang Hu, and K. Skadron. Performance, Energy, and Thermal Considerations for SMT and CMP Architectures. *Proceedings of the 11th HPCA*, 2005.

H. M. Mathis, A. E. Mericas, J. D. McCalpin, R. J. Eickemeyer, and S. R. Kunkel. Characterization of simultaneous multithreading (SMT) efficiency in POWER5. *IBM Journal of R&D*, 2005.

Sarah E. Michalak, Kevin W. Harris, Nicolas W. Hengartner, Bruce E. Takala, and Stephen A. Wender. Predicting the number of fatal soft errors in los alamos national laboratories's asc q supercomputer. *IEEE Trans. Device and Materials Reliability*, 2005.

S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. Detailed Design and Evaluation of Redundant Multithreading Alternatives. *Proceedings of the 29th ISCA*, 2002.

I. Parulkar, A. Wood, J. C. Hoe, B. Falsafi, S. V. Adve, and J. Torrellas. OpenSPARC: An Open Platform for Hardware Reliability Experimentation. *Fourth Workshop on Silicon Errors in Logic-System Effects (SELSE)*, 2008.

E. Perelman, G. Hamerly, and B. Calder. Picking Statistically Valid and Early Simulation Points. *Proc. of the PACT*, 2003.

S. K. Reinhardt and S. S. Mukherjee. Transient Fault Detection via Simultaneous Multithreading. *Proceedings of the 27th ISCA*, 2000.

E. Rotenberg. AR-SMT: A Microarchitectural Approach to Fault Tolerance in a Microprocessor. *Proceedings of FTCS*, 1999.

J. C. Smolens, B. T. Gold, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk. Fingerprinting: Bounding soft error detection latency and bandwidth. *Proceedings of the 9th ASPLOS*, 2004.