Outline

• Basic Concepts
• Physical Redundancy
• Error Detecting/Correcting Codes
• Re-Execution Techniques
• Backward Error Recovery Techniques

Re-Execution

• Replicate the actions on a module either on the same module (temporal redundancy) or on spare modules (temporal & spatial redundancy)
• Good for detecting and/or correcting transient faults
  – Transient error will only affect one execution
• Analogy from real life: calling to confirm a reservation
• Can implement this at many different levels
  – ALU
  – Thread context
  – Processor
  – System

Re-Execution with Shifted Operands (RESO)

• Re-execute the same arithmetic operations, but with shifted operands (question: why shift?)
• Goal: detect errors in ALU
• Example: shift left by 2
  – Simplified example: we’re ignoring wraparound

| 0 0 1 0 | 1 0 X X |
| + 1 0 1 1 | + 0 1 X X |
| 1 0 1 0 | 1 1 X X |

• By comparing output bit 0 of the first execution and output bit 2 of the shifted re-execution, we detect an error in the ALU, since they should be equal

Re-Execution with Shifted Operands (RESO)

• How general is RESO?
• Does it work for multiplication? Division?
• What about arbitrary combinational logic?
Re-Execution With a Twist

• After adding $A + B = C$, then compute $C-B$
  – If we don’t get $A$, there’s a problem
• What new types of faults/errors does this detect?
• How general is this approach?
  – I.e., how many operations are reversible?
  – Can we extend this to higher-level operations (algorithms)?
• The devil is in the details (corner cases)
  – Overflow, underflow, divide by zero, etc.
• This type of execution checking is more frequently performed at the software level ... why?

Re-Execution with Processes

• Use redundant process to detect errors
• If we only have one processor, we must execute the two processes sequentially and then compare their results. If they differ, there’s an error.
  – Problem: slowdown factor = 2
• In a multiprocessor, we can execute copies of the same process simultaneously on 2 processors and have them periodically compare their results
  – Trend: even single chips now can contain multiple processors
  – Almost no slowdown, except for comparisons
  – Disadvantage: the opportunity cost of not using that other processor to perform non-redundant work
  – Is this an FER approach? (hint: what happens if an error occurs?)

Re-Execution with Threads

• Use redundant threads to detect/correct errors
  – A thread is like a process, except that multiple threads can share the same address space
• Many current microprocessors, like the Pentium4, are multithreaded (“hyperthreaded”, if you work for Intel)
  – Each processor can run multiple processes or multiple threads of the same process (i.e., it has multiple thread contexts)
• Can re-execute a program on multiple thread contexts, just like with multiple processors
  – Better performance than re-execution with multiple processors, since the comparison can be performed on-chip
  – Less opportunity cost to use extra thread context than extra processor
  – We’ll talk more about this soon when we get to AR-SMT paper

DIVA (Austin, Micro ’99)

• Uses small checker core to dynamically verify aggressive microprocessor core
  – Dynamic verification called online testing or concurrent testing
• Heterogeneous temporal redundancy
  – Modules are not the same → why is this a good thing?

Complicated, Speculative Microprocessor (see ECE252)
Simple Checker Core (ECE152)

possibly incorrect data
always correct data

Outside World

Fetch, decode, execute, memory
Writeback (commit)
“Razor” (Ernst et al., MICRO 2003)

• “Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation”

“AR-SMT” (Rotenberg, FTCS ’99)

• “AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors” (Rotenberg)