Multithreading Models

- Coarse-grain - switch contexts (typically several cycles) on long-latency event.
  - MIT Alewife
- Fine-grain - switch contexts every cycle.
  - HEP, Tera, Sun Niagara
- Simultaneous Multithreading
  - Compaq 21464, Intel Pentium 4, Power 5, 6

Coarse-grain multithreading

Fine-grain multithreading

Simultaneous Multithreading
Who are the arch enemies of scalability?

Scalability - Four Pronged Approach

Programming Model?
Coherent Shared Memory

- Unique features?

LimitLESS Directory

- So why scalable?

Full-Empty Bits

- Store, set full
- Load if full, set empty

Latency Tolerance
Block Multithreading
- Aka coarse-grain multithreading

Block (CG) Multithreading
- What does it do well?
- What doesn’t it do well?

Simultaneous Multithreading

Motivation
Hardware Multithreading

Conventional Processor

Multithreaded Processor

CPU

PC regs

PC regs

PC regs

Superscalar Execution

Issue Slots

Time (proc cycles)

Vertical waste

Horizontal waste

Vertical waste
Superscalar Execution

Superscalar Execution with Fine-Grain Multithreading

Superscalar Execution with Fine-Grain Multithreading

Simultaneous Multithreading
The Potential for SMT

Simultaneous Multithreading

Fine-Grain Multithreading

Conventional Superscalar

Number of Threads

Throughput Instructions per Cycle

0 1 2 3 4 5 6 7 8

The goals for this architecture are:

1. Minimize the architectural impact on conventional superscalar design.
3. Achieve significant throughput gains with many threads.

A Conventional Superscalar Architecture

- Fetch up to 8 instructions per cycle
- Out-of-order, speculative execution
- Issue 3 floating point, 6 integer instructions per cycle

An SMT Architecture

- Fetch up to 8 instructions per cycle
- Out-of-order, speculative execution
- Issue 3 floating point, 6 integer instructions per cycle
Performance of the Naïve Design

Bottlenecks of the Baseline Architecture
- Instruction queue full conditions (12-21% of cycles)
  - Lack of parallelism in the queue.
- Fetch throughput (4.2 instructions per cycle when queue not full)

Improving Fetch Throughput
- The fetch unit in an SMT architecture has two distinct advantages over a conventional architecture.
  - Can fetch from multiple threads at once.
  - Can choose which threads to fetch.
**Improved Fetch Performance**

- Fetching from 2 threads/cycle achieved most of the performance from multiple-thread fetch.
- Fetching from the thread(s) which have the fewest unissued instructions in-flight significantly increases parallelism and throughput (ICOUNT fetch policy)

**This SMT Architecture, then:**

- Borrows heavily from conventional superscalar design.
- Minimizes the impact on single-thread performance
- Achieves significant throughput gains over the superscalar (2.5X, up to 5.4 IPC)

**Multithreading Models**

- Coarse-grain – switch contexts (typically several cycles) on long-latency event.
  - MIT Alewife
- Fine-grain – switch contexts every cycle.
  - HEP, Tera, Sun Niagara
- Simultaneous Multithreading
  - Compaq 21464, Intel Pentium 4, Power 5, 6
  - Pros, cons, issues, …?
HEP Multithreading

CPU

PSW PSW PSW PSW

PSW PSW PSW PSW

Tera Multithreading

Cool Tera Features

- Full/empty bits on memory
- Randomized memory (why??)
- No bypassing
- Explicit-dependence lookahead
- LIW
- No caches
- High-bandwidth network

- Which of these are related to multithreading?

Other SMT Research (UCSD emphasis)

- Compilation issues, including instruction layout for icache performance.
- Multiple-path execution
- Symbiotic job scheduling
- Synchronization and parallelization
- Multithreading and power
- Helper threads – speculative precomputation
- Multicore/multithreaded, clustered multithreaded
- Multithreaded value prediction
- Event-driven compilation
Other interesting multithreaded research

- Speculative multithreading
- Fault tolerance
- slipstreaming

Summary

- Why simultaneous multithreading?
- Long-term solution?
- When won’t it work?
- What next?