Multithreaded Architectures
Overview

- Multithreaded Software
- Multithreaded Architecture
- Multithreaded Micro-Architecture
- Conclusions
Multithreading
Basics

- **Process**
  - Each process has its unique address space
  - Can consist of several threads

- **Thread** – each thread has its unique execution context
  - Thread has its own PC (Sequencer) + registers + stack
  - All threads (within a process) share same address space
  - Private heap is optional

- Multithreaded app’s: process broken into threads
  - #1 example: transactions (databases, web servers)
    - Increased concurrency
    - Partial blocking (your transaction blocks, mine doesn’t have to)
    - Centralized (smarter) resource management (by process)
MultiThreaded Architecture (MTA) Goals

- Minimal impact on single-thread performance
- Performance gain on multithreaded applications
  - Shared resources → faster communication/synchronization
- Improved throughput on multiple thread workloads
  - Multiple thread = multithreaded or multi-processes workload
- Good cost/throughput
  - “utilization”
Scalar Execution

Dependencies reduce throughput/utilization
Superscalar Execution

Generally increases throughput, but decreases utilization
Predication

Generally increases utilization, increases throughput less (much of the utilization is thrown away)
CMP – Chip Multi-Processor

Low utilization / higher throughput
Blocked Multithreading

May increase utilization and throughput, but must switch when current thread goes to low utilization/throughput section (e.g. L2 cache miss)
Fine Grained Multithreading

Increases utilization/throughput by reducing impact of dependences
Simultaneous Multithreading (SMT)

Increases utilization/throughput
Blocked Multithreading
(SOEMT- Switch On Event MT, aka’ – “Poor Man MT”)

- Critical decision: when to switch threads
  - When current thread’s utilization/throughput is about to drop (e.g. L2 cache miss)
- Requirements for throughput:
  - (Thread switch) + (pipe fill time) << blocking latency
    - Would like to get some work done before other thread comes back
  - Fast thread-switch: multiple register banks
  - Fast pipe-fill: short pipe
- Advantage: small changes to existing hardware
- Drawback: high single thread performance requires long thread switch
- Examples
  - Macro-dataflow machine
  - MIT Alewife
  - IBM Northstar
  - Rumors – AMD?, SUN?
Interleaved (Fine grained) Multithreading

- Critical decision: none?
- Requirements for throughput:
  - Enough threads to eliminate data access and dependencies
    - Increasing number of threads reduces single-thread performance

Superscalar pipeline

Multithreaded pipeline
Interleaved (Fine grained) Multithreading (cont.)

- **Advantages:**
  - *(with flexible interleave:)* Reasonable single thread performance
  - High processor utilization (esp. in case of many thread)
- **Drawback:**
  - Complicated hardware
  - Multiple contexts (states)
  - *(with inflexible interleave:)* limits single thread performance
- **Examples:**
  - HEP Denelcor: 8 threads (latencies were shorter then)
  - TERA: 128 threads
  - MicroUnity - 5 x 1GZ threads = 200 MHz like latency
- **Became attractive for new Gfx/MM/Comm processors**
Simultaneous Multi-threading (SMT)

- Critical decision: fetch-interleaving policy
- Requirements for throughput:
  - Enough threads to utilize resources
    - Fewer than needed to stretch dependences
- Examples:
  - Compaq Alpha EV8 (cancelled)
  - Intel Pentium® 4 Hyper-Threading Technology
SMT Case Study: EV8

- 8-issue OOO processor (wide machine)

SMT Support
- Multiple sequencers (PC): 4
- Alternate fetch from multiple threads
  - Separate register renaming for each thread
- More (4) physical registers
- Thread tags on all sequential resources
  - ROB, LSQ, BTB, etc.
  - Allow per thread flush/trap/retirement
- Process tags on all address space resources: caches, TLB’s, etc.
- Notice: none of these things are in the “core”
  - Instruction queues, scheduler, wakeup are SMT-blind
Basic EV8

Fetch | Decode/Map | Queue | Reg Read | Execute | Dcache/Store Buffer | Reg Write | Retire

PC | Register Map | Regs | Dcache | Regs |

Thread-blind
SMT EV8

Thread-blind

Thread Blind? Not really
Performance Scalability

Multiprogrammed Workload

Decomposed SPEC95 Applications

Multithreaded Applications
Multithread

I’m afraid of this

Is CMP a solution?
Performance Gain – when?

- When threads do not “step” on each other
  “predicted state” $\Rightarrow$ $\$, Branch Prediction, data prefetch….
- If ample resources are available
- Scare slow resource, can hamper performance
Scheduling in SMT

- What if one thread gets “stuck”?  
  - Round-robin: eventually it will fill up the machine (not good)  
  - ICOUNT: thread with fewest instructions in pipe has priority  
    - Translation: thread doesn’t get to fetch until it gets “unstuck”  
  - Other policies have been devised to get best balance, fairness, and overall performance:  
    - Flush, Flush++, DCRA, ...

- Variation: what if one thread is spinning?  
  - Not really stuck, gets to keep fetching  
  - Have to stop/slow it artificially (QUIESCE)  
    - Sleep until a memory location changes  
    - On Pentium 4 – use the Pause instruction
MT Application Data Sharing

- **Shared memory apps**
  - Decoupled caches = Private caches
    - Easier to implement – use existing MP like protocol
  - Shared Caches
    - Faster to communicate among threads
    - No coherence overhead
    - Flexibility in allocating resources

![Diagram of shared memory with multiple ways](image-url)
SMT vs. CMP

- How to use 2X transistors?
  - Better core+SMT or CMP?

- SMT
  - Better single thread performance
  - Better resource utilization

- CMP
  - Much easier to implement
  - Avoid wire delays problems
  - Overall better throughput per area/power
  - More deterministic performance

- Rough numbers
  - SMT: ST Performance/MT performance for 2X transistors = 1.3X/1.5-1.7X
  - CMP: ST Performance /MT performance for 2X transistors = 1X/2X
Summary (1)

- **Multithreaded Software**
- **Multithreaded Architecture**
  - Advantageous cost/throughput …
  - Blocked MT - long latency tolerance
    - Good single thread performance, good throughput
    - Needs fast thread switch and short pipe
      OOO execution reduces the advantage!
  - Interleaved MT – ILP increase
    - Bad single thread performance, good throughput
    - Needs many threads, requires several loaded contexts…
  - Simultaneous MT – ILP increase
    - OK MT performance, good single thread performance,
      Good utilization …
    - Need fewer threads
Summary (2)

- Multithreading complicates the architecture  ➔ but VLSI technology is already there
  - Advantageous price/performance…
  - Real value yet to be seen!

- A lot effort to parallelize new apps  ➔ Incompatibility with existing binaries
  - Compiler technology is not ready yet
  - OS has to distinguish between MT and CMP
Example: Pentium® 4 Hyper-Threading

- Executes two tasks simultaneously
  - Two different applications
  - Two threads of same application
- CPU maintains architecture state for two processors
  - Two logical processors per physical processor
- Implementation on Intel® Xeon™ Processor
  - Two logical processors for < 5% additional die area
- The processor pretends as if it has 2 cores in an MP shared memory system
uArchitecture impact

- Replicate resources
  - All; per-CPU architectural state
  - Instruction Pointers, Renaming logic
  - Some smaller resources - e.g. return stack predictor, ITLB, etc.

- Partition resources (share by splitting in half per thread)
  - Several buffers –
    Re-Order Buffer, Load/Store Buffers, queues, etc

- Share most resources
  - Out-of-Order execution engine
  - Caches
Hyper-Threading Performance

- Performance varies as expected with:
  - Number of parallel threads
  - Resource utilization
- Less aggressive MT than EV8 ➔ Less impressive scalability
  - Machine optimized for single-thread.

**Intel Xeon Processor MP platforms are prototype systems in 2-way configurations**
**Applications not tuned or optimized for Hyper-Threading Technology**