HY425 Lecture 11: Simultaneous Multithreading

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

November 21, 2011

Hardware multithreading

Latency overlap

- Processor maintains multiple active thread contexts
- Threads may be generated from a single or multiple programs
  - Multi-program vs. single-program thread-level parallelism
- Processor switches threads in two ways:
  - Every cycle
  - Upon a long-latency event incurred in a thread
- Multithreading overlaps latencies, improves utilization

Looking for sources of ILP

Single program

- Unrolled loops
- Frequent code paths, using branch prediction
- Threads: independent instruction streams in program
  - Blocks of independent loop iterations
  - Independent functions or other long code paths

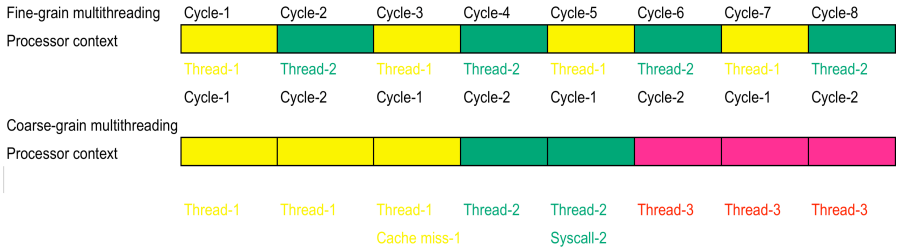
Across multiple programs

- If one program does not expose sufficient ILP, execute multiple programs simultaneously to increase overall available ILP
- Improve utilization of hardware resources, but not necessarily performance of a single program

Hardware multithreading alternatives

Fine-grain multithreading

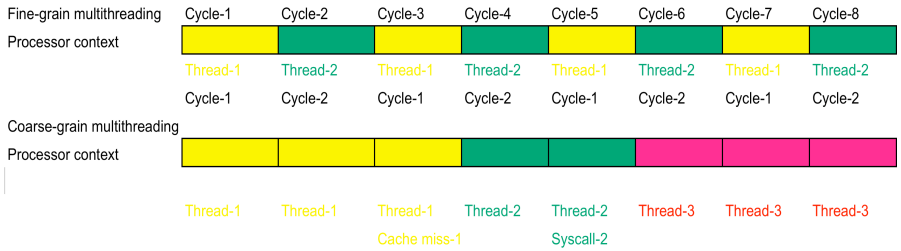
- Fine-grain multithreading switches processor context every thread cycle
- Context belongs to same address space



Hardware multithreading alternatives

Fine-grain multithreading

- Coarse-grain multithreading switches processor context upon long-latency event
- Context may belong to different address space



Fine-grain hardware multithreading

Switch every clock cycle

- Need fast HW switch between contexts
  - Multiple PCs and register files
  - Alternatively, thread ID attached to each GP register
- Implemented with round-robin scheduling, skipping stalled threads
- Hides both short and long stalls
- Delays all threads, even if they have no stalls

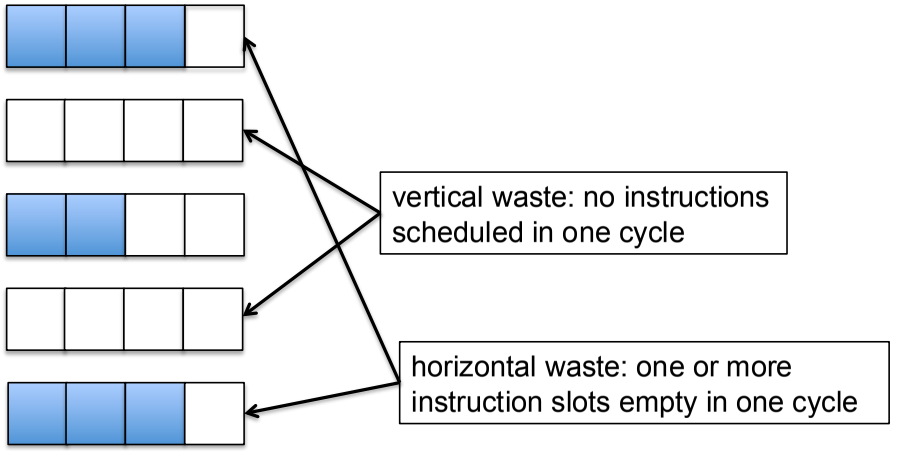
Coarse-grain hardware multithreading

Switch every clock cycle

- Can afford slower context switch than fine-grain multithreading
- Threads are not slowed down
  - Other thread runs when current thread stalls
- Pipeline startup cost upon thread switching
  - Processor issues instructions from one thread (address space)

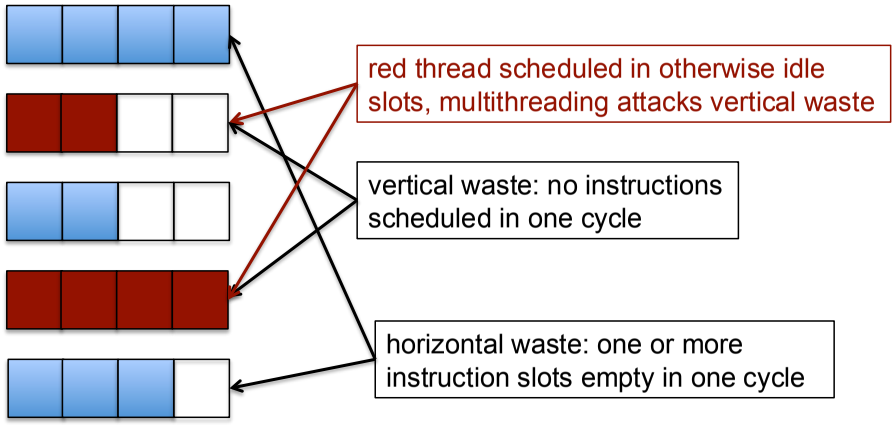
ILP waste

Single-thread processor with horizontal waste



ILP waste

Multi-threaded processor with vertical waste

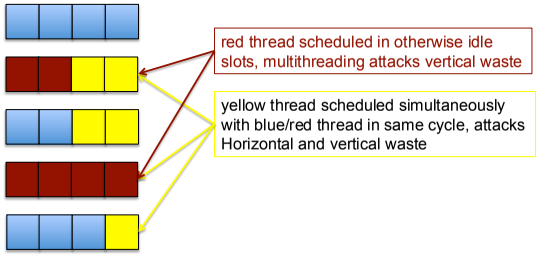


Simultaneous multithreading

Fine-grain multithreading in same cycle

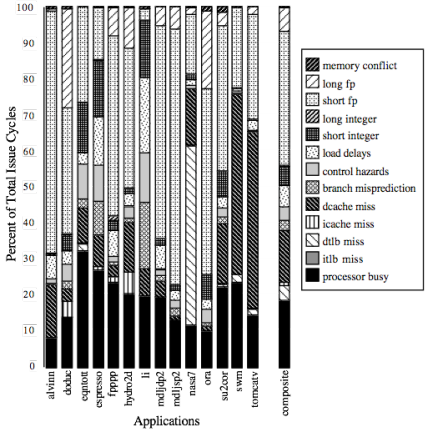
- Instructions fetched from multiple threads in same cycle
- Attacks horizontal waste
- Fine-grain or coarse-grain multithreading to attack vertical waste

SMT concept



Wasted instruction slots in ILP processor

Unused cycles in 8-issue processor



Cast ILP processor to exploit TLP

Reuse ILP hardware for TLP

- TLP and ILP exploit two different forms of parallelism
- TLP and ILP both need multiple functional units
- TLP can fill up functional units left idle by ILP

Dynamically scheduled processor base

Shared HW mechanisms

- Large set of virtual registers can hold register sets of independent threads
- Renaming provides unique register identifiers to different threads
- Out-of-order completion of instructions from different threads allowed
  - No cross-thread RAW, WAW, WAR hazards
- Separate reorder buffer per thread

SMT design issues

Extensions over superscalar

- Instruction issue logic
  - Fetch instructions from > 1 threads simultaneously
  - Policy choice for selecting ready threads to select instructions
  - Instructions per thread issued is policy parameter
- Hardware resource sharing
  - Active threads may share caches, TLBs, branch predictors, ...
  - Active threads may interfere between each other, thus affecting performance
  - Constructive “symbiotic” co-scheduling
    - Active threads cooperatively improve hardware utilization

Replicated state

Thread context

- PC and register file
- Address space (private or shared)
- Switching context
  - 100s to 1000s of cycles in software
  - More expensive between address spaces
  - Less expensive between threads in same address space
- Hardware resources other than PC and registers can be shared

SMT workload considerations

Multi-programming (MP) workload

- One hardware thread per address space
- Multiple address spaces executing simultaneously
  - Performance limited by contention for shared resources

Multi-threaded parallel workload (TLP)

- Many hardware threads per address space
- Parallelized program
  - Performance limited by contention for shared resources between threads of same program
  - Performance limited by serialization bottlenecks, such as synchronization, limited availability of parallelism

SMT design considerations

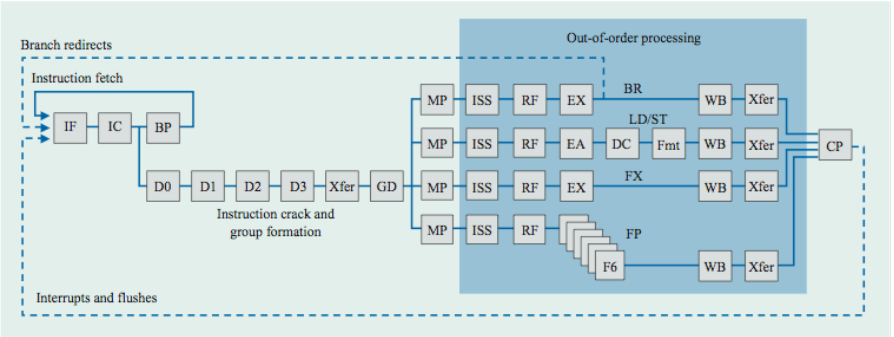
Resource scaling

- Impact of fine-grained scheduling on single thread performance
  - Priority scheduling approaches to improve single-thread performance
- More registers to hold multiple threads
- Complexity in instruction issue
  - More instructions from >1 threads to be considered
  - More complex selection logic (scheduling algorithm) for instructions
- Complexity in instruction commit (choice among threads)
- Cache and TLB conflicts between threads

Comparison between SMT and superscalar

IBM Power4

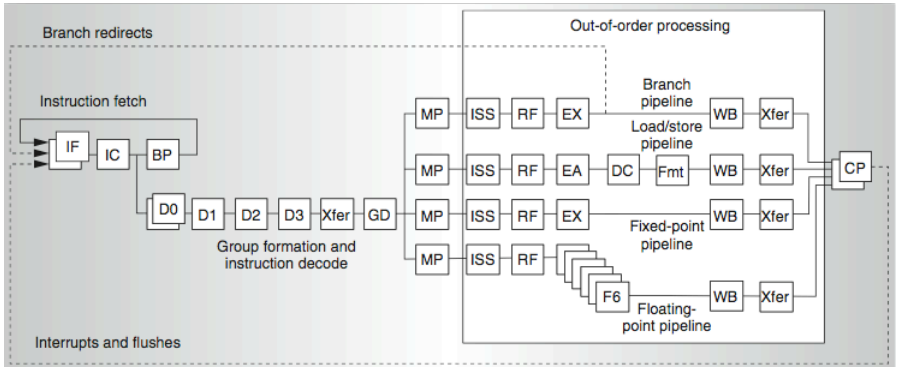
Out-of-order execution processor with 8 execution units



Comparison between SMT and superscalar

IBM Power5 vs. Power4

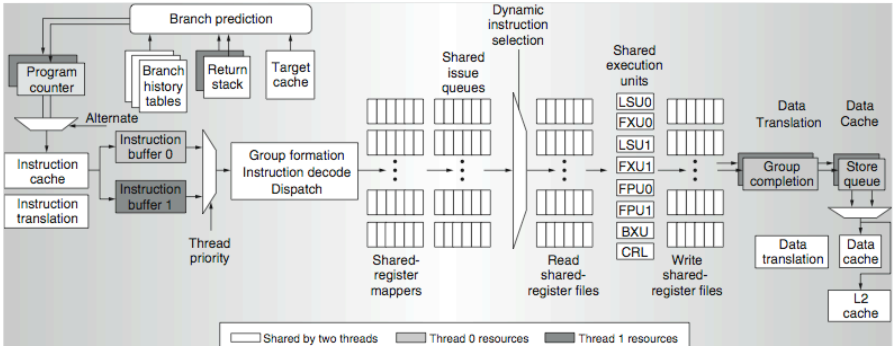
Duplication of PC and commit logic



Power5 data flow

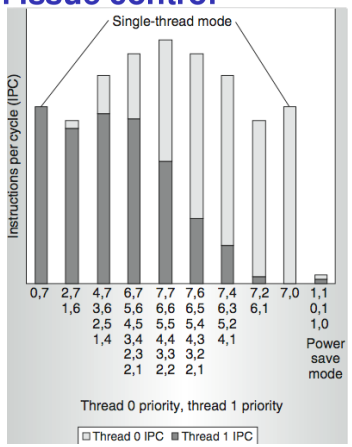
Shared vs. private resources

Private instruction queues. Shared branch prediction logic, register renaming, execution units, caches.



Power5 thread priorities

Thread instruction issue control



Power5 thread priority control details

Thread priorities

- Hardware or software-controlled
- Priority mechanism controls instruction decoding rate for each thread
- Hardware allocates $R = 2^{|PrioT_1 - PrioT_2|+1}$ decode slots
- $R - 1$ slots to higher-priority thread, rest of the slots to other thread
- One decode slot per thread if threads have equal priority
- Hardware throttles instruction fetch for thread, if thread causes stalls to sibling thread
- Hardware policies for resource balancing (e.g. take away slots from thread that incurs many L2 cache misses)

Power5 single-thread execution

Thread switching

- Multi-threaded execution not always beneficial
- Idle threads with no work stay in dormant or inactive state
- Idle threads awaken with external (dormant) or timer (inactive) interrupt
- Dormant threads may also be waken up by active threads

Power5 extensions to support SMT

Incremental extensions to superscalar Power4

- Increased associativity of L1 instruction cache and the instruction address translation buffers
- Per thread load and store queues
- Increased size of the L2 (1.92 vs. 1.44 MB) and L3 caches
- Separate instruction prefetch and buffering per thread
- Increased number of virtual registers from 152 to 240
- Increased size of several issue queues
- The Power5 core is about 24% larger than the Power4 core because of the addition of SMT support

SMT performance in commercial processors

Multi-program workloads

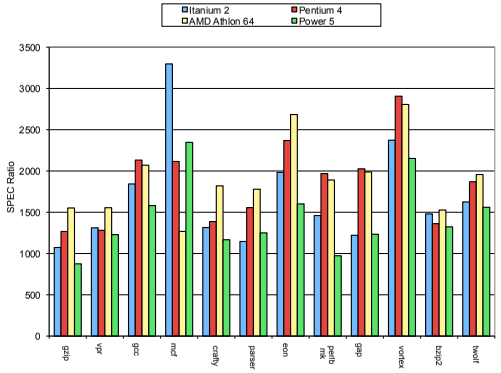
- Pentium 4 Extreme SMT achieves 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  - Pentium 4 is dual-threaded SMT
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
- Running on Pentium 4 each of 26 SPEC benchmarks paired with every other (26^2 runs) speed-ups from 0.90 to 1.58; average is 1.20
- Power 5, 8 processor server 1.23 faster for SPECint_rate with SMT, 1.16 faster for SPECfp_rate
- Power 5 running 2 copies of each app speedup between 0.89 and 1.41
  Most gained some FP apps had cache conflicts and least gains

Comparison between ILP processors

Processor	Micro architecture	Fetch / Issue / Execute	FU	Clock Rate (GHz)	Transis-tors Die size	Power
Intel Pentium 4 Extreme	Speculative dynamically scheduled; deeply pipelined; SMT	3/3/4	7 int. 1 FP	3.8	125 M 122 mm^2	115 W
AMD Athlon 64 FX-57	Speculative dynamically scheduled	3/3/4	6 int. 3 FP	2.8	114 M 115 mm^2	104 W
IBM Power5 (1 CPU only)	Speculative dynamically scheduled; SMT; 2 CPU cores/chip	8/4/8	6 int. 2 FP	1.9	200 M 300 mm^2 (est.)	80W (est.)
Intel Itanium 2	Statically scheduled VLIW-style	6/5/11	9 int. 2 FP	1.6	592 M 423 mm^2	130 W

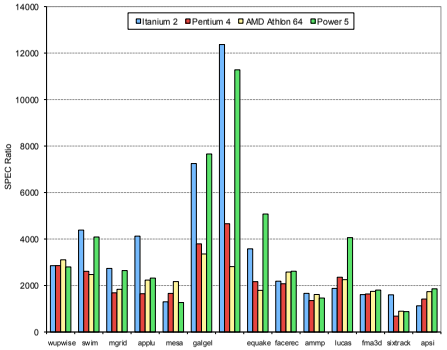
Comparison between ILP processors

SPEC INT rate



Comparison between ILP processors

SPEC FP rate



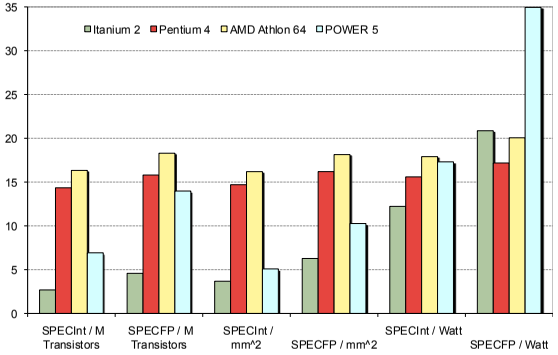
Measuring processor efficiency

Area- and power-efficiency

- Processor performance gain comes at an area/power budget cost
  - Weigh performance again against power and area increase
- Area-efficiency
  - Performance / transistor (e.g. SPECrate/million transistors)
- Power-efficiency
  - Performance / watt (e.g. SPECrate/watt)

Comparison between ILP processors

Power and area efficiency



Best ILP approach?

Results with commercial processors

- AMD Athlon most performance-efficient in INT programs
- Power5 most performance-efficient in FP programs
- Power5 most power-efficient overall
- Itanium VLIW least power-efficient and area-efficient overall