CSE 490/590 Computer Architecture

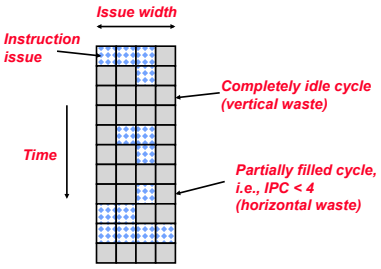
Multithreading II

Steve Ko
Computer Sciences and Engineering
University at Buffalo

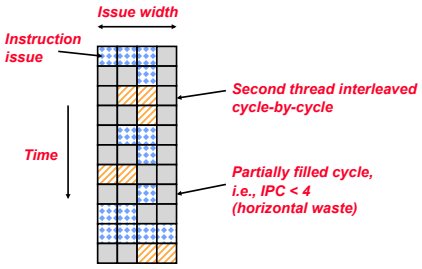
Last time…

- Multithreading executes instructions from different threads
- Coarse-grained multithreading switches threads on cache misses
- Most of the OoO superscalar units are idle.

Superscalar Machine Efficiency

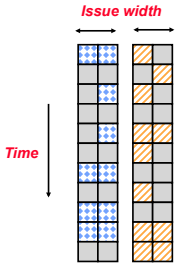


Vertical Multithreading



- What is the effect of cycle-by-cycle interleaving?

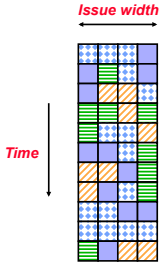
Chip Multiprocessing (CMP)



- What is the effect of splitting into multiple processors?

Ideal Superscalar Multithreading

[Tullsen, Eggers, Levy, UW, 1995]



- Interleave multiple threads to multiple issue slots with no restrictions

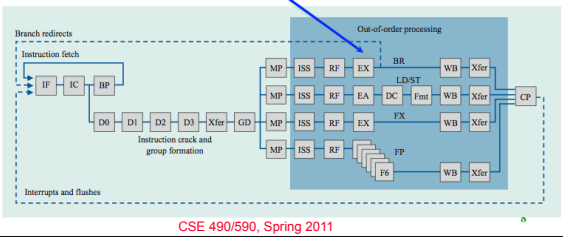
O-o-O Simultaneous Multithreading

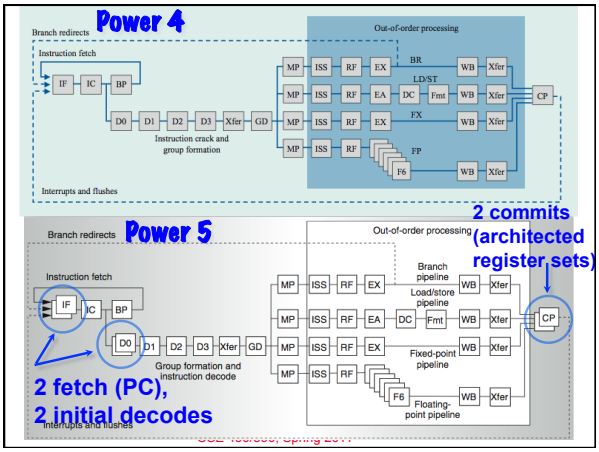
[Tullsen, Eggers, Emer, Levy, Stamm, Lo, DEC/UW, 1996]

- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OOO instruction window already has most of the circuitry required to schedule from multiple threads
- Any single thread can utilize whole machine

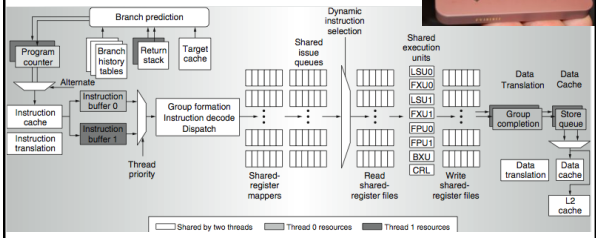
IBM Power 4

Single-threaded predecessor to Power 5. 8 execution units in out-of-order engine, each may issue an instruction each cycle.





Power 5 data flow ...



Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

Changes in Power 5 to support SMT

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

CSE 490/590 Administrivia

- Quiz 2 (next Friday 4/8): After midterm until next Monday

Pentium-4 Hyperthreading (2002)

- First commercial SMT design (2-way SMT)
  - Hyperthreading == SMT
- Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress
  - No logical processor can use all entries in queues when two threads are active
- Processor running only one active software thread runs at approximately same speed with or without hyperthreading
- Hyperthreading dropped on OoO P6 based followons to Pentium-4 (Pentium-M, Core Duo, Core 2 Duo), until revived with Nehalem generation machines in 2008.
- Intel Atom (in-order x86 core) has two-way vertical multithreading

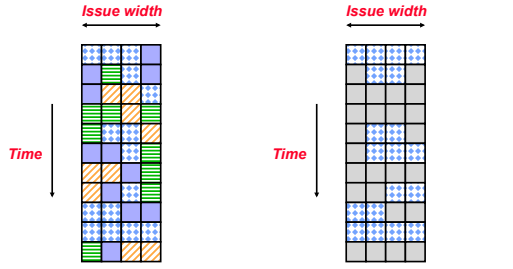
Initial Performance of SMT

- Pentium 4 Extreme SMT yields 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 was 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
  - Fl.Pt. apps had most cache conflicts and least gains

SMT adaptation to parallelism type

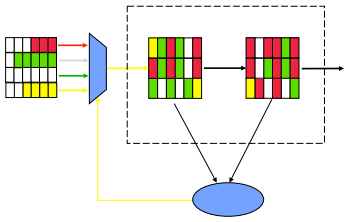
For regions with high thread level parallelism (TLP) entire machine width is shared by all threads

For regions with low thread level parallelism (TLP) entire machine width is available for instruction level parallelism (ILP)



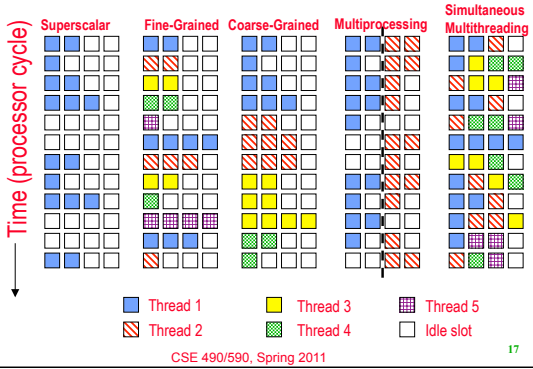
Icount Choosing Policy

Fetch from thread with the least instructions in flight.

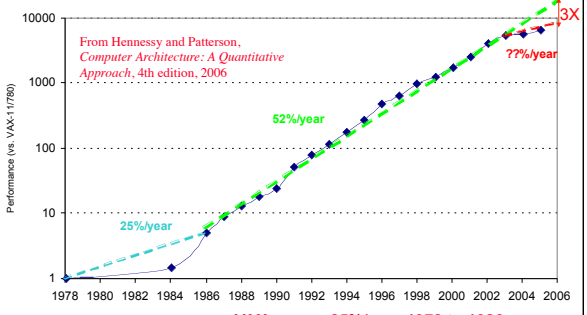


Why does this enhance throughput?

Summary: Multithreaded Categories



Uniprocessor Performance (SPECint)



- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

Parallel Processing: Déjà vu all over again?

"… today's processors … are nearing an impasse as technologies approach the speed of light.."

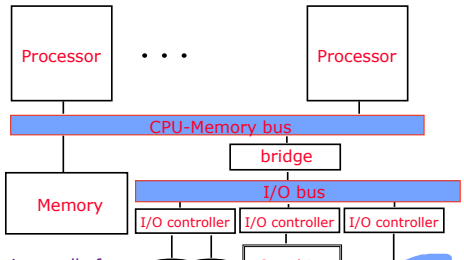
David Mitchell, *The Transputer: The Time Is Now* (1989)

- Transputer had bad timing (Uniprocessor performance↑)
  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"

  Paul Otellini, President, Intel (2005)
- All microprocessor companies switch to MP (2X CPUs / 2 yrs)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs

Manufacturer/Year	AMD/'09	Intel/'09	IBM/'09	Sun/'09
Processors/chip	6	8	8	16
Threads/Processor	1	2	4	8
Threads/chip	6	16	32	128

Symmetric Multiprocessors



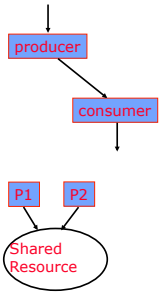
symmetric
- All memory is equally far away from all processors
- Any processor can do any I/O (set up a DMA transfer)

Synchronization

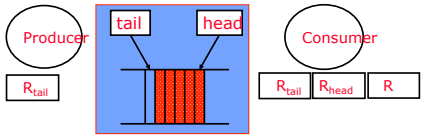
The need for synchronization arises whenever
there are concurrent processes in a system
(even in a uniprocessor system)

Producer-Consumer: A consumer process must wait until the producer process has produced data

Mutual Exclusion: Ensure that only one process uses a resource at a given time



A Producer-Consumer Example



Producer posting Item x:
Load R_{tail}, (tail)
Store (R_{tail}), x
R_{tail}=R_{tail}+1
Store (tail), R_{tail}

Consumer:
Load R_{head}, (head)
spin: Load R_{tail}, (tail)
if R_{head}==R_{tail} goto spin
Load R, (R_{head})
R_{head}=R_{head}+1
Store (head), R_{head}
process(R)

The program is written assuming instructions are executed in order.

Problems?

A Producer-Consumer Example
continued

Producer posting Item x:
Load R_{tail}, (tail)
1 Store (R_{tail}), x
R_{tail}=R_{tail}+1
2 Store (tail), R_{tail}

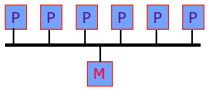
Consumer:
Load R_{head}, (head)
spin: Load R_{tail}, (tail) *3*
if R_{head}==R_{tail} goto spin
Load R, (R_{head}) *4*
R_{head}=R_{head}+1
Store (head), R_{head}
process(R)

Can the tail pointer get updated before the item x is stored?

Programmer assumes that if 3 happens after 2, then 4 happens after 1.

Problem sequences are:
2, 3, 4, 1
4, 1, 2, 3

Sequential Consistency
A Memory Model



" A system is *sequentially consistent* if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in the order specified by the program"

Leslie Lamport

Sequential Consistency =
arbitrary *order-preserving interleaving*
of memory references of sequential programs

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252