CSC 631: High-Performance Computer Architecture

Spring 2022

Lecture 9: Multithreading

Thread-Level Parallelism (TLP)

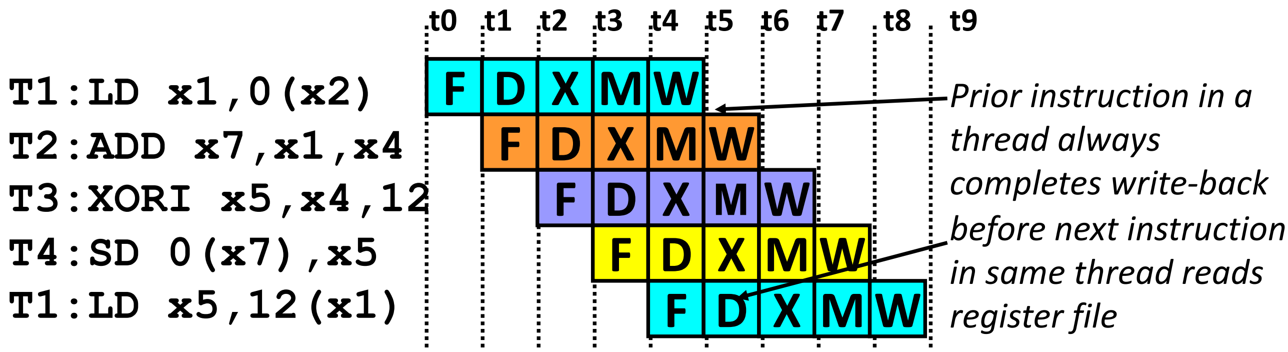
- Difficult to continue to extract instruction-level parallelism (ILP) from a single sequential thread of control
- Many workloads can make use of thread-level parallelism:
  - TLP from ***multiprogramming*** (run independent sequential jobs)
  - TLP from ***multithreaded*** applications (run one job faster using parallel threads)
- Multithreading uses TLP to improve utilization of a single processor

Multithreading

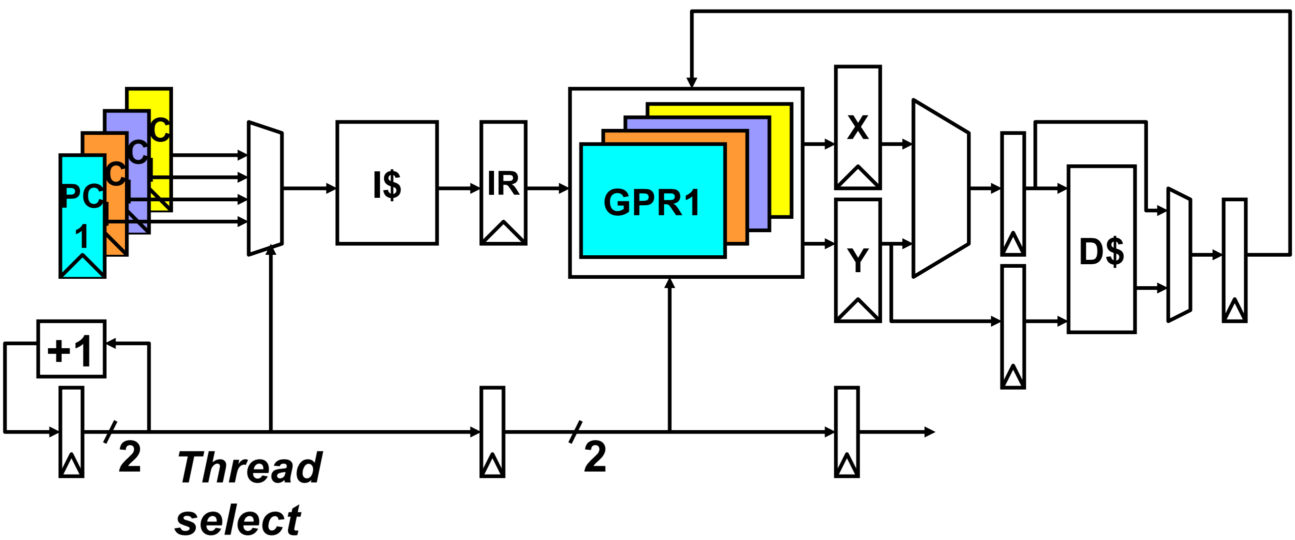
How can we guarantee no dependencies between instructions in a pipeline?

One way is to interleave execution of instructions from different program threads on same pipeline

Interleave 4 threads, T1-T4, on ***non-bypassed*** *5-stage pipe*



Simple Multithreaded Pipeline



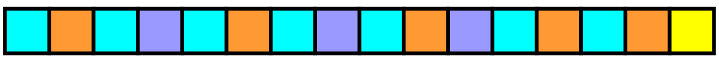
- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs

Multithreading Costs

- Each thread requires its own user state
  - PC
  - GPRs
- Also, needs its own system state
  - Virtual-memory page-table-base register
  - Exception-handling registers
- *Other overheads:*
  - Additional cache/TLB conflicts from competing threads
    - or add larger cache/TLB capacity
  - More OS overhead to schedule more threads (where do all these threads come from?)

Thread Scheduling Policies

- Fixed interleave *(CDC 6600 PPUs, 1964)*
  - Each of N threads executes one instruction every N cycles
  - If thread not ready to go in its slot, insert pipeline bubble
- Software-controlled interleave *(TI ASC PPUs, 1971)*
  - OS allocates S pipeline slots amongst N threads
  - Hardware performs fixed interleave over S slots, executing whichever thread is in that slot



- Hardware-controlled thread scheduling *(HEP, 1982)*
  - Hardware keeps track of which threads are ready to go
  - Picks next thread to execute based on hardware priority scheme

Denelcor HEP
(Burton Smith, 1982)

First commercial machine to use hardware threading in main CPU

- 120 threads per processor
- 10 MHz clock rate
- Up to 8 processors
- precursor to Tera MTA (Multithreaded Architecture)

Coarse-Grain Multithreading

- Tera MTA designed for supercomputing applications with large data sets and low locality
  - No data cache
  - Many parallel threads needed to hide large memory latency

- 256 mem-ops/cycle * 150 cycles/mem-op = 38K instructions-in-flight…
  - Tera was not very successful, 2 machines sold.
  - Changed their name back to Cray!

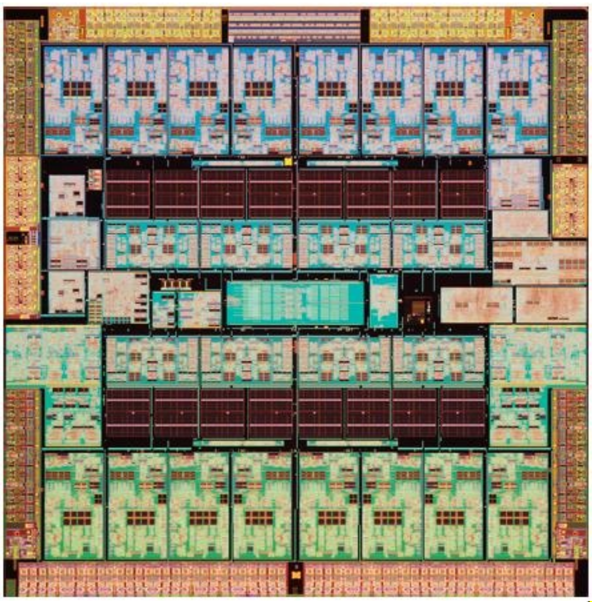
- Other applications are more cache friendly
  - Few pipeline bubbles if cache mostly has hits
  - Just add a few threads to hide occasional cache miss latencies
  - Swap threads on cache misses

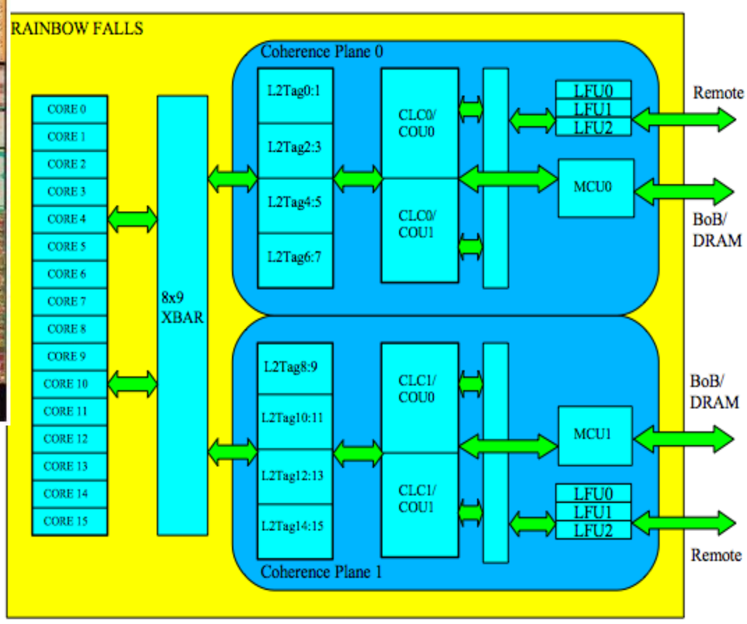
Oracle/Sun Niagara processors

- Target is datacenters running web servers and databases, with many concurrent requests
- Provide multiple simple cores each with multiple hardware threads, reduced energy/operation though much lower single thread performance

- Niagara-1 [2004], 8 cores, 4 threads/core
- Niagara-2 [2007], 8 cores, 8 threads/core
- Niagara-3 [2009], 16 cores, 8 threads/core
- T4 [2011], 8 cores, 8 threads/core
- T5 [2012], 16 cores, 8 threads/core
- M5 [2012], 6 cores, 8 threads/core
- M6 [2013], 12 cores, 8 threads/core

Oracle/Sun Niagara-3, "Rainbow Falls" 2009





Oracle SPARC M6 Processor (2013)

The Next Oracle Processor: SPARC M6

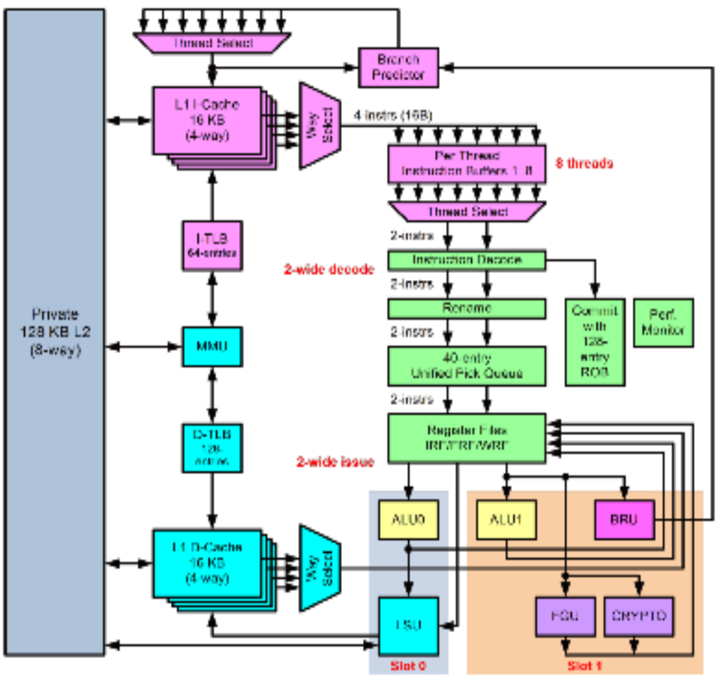
	nm	Cores	Threads	L3$	Memory per Socket	PCIe	Max. Sockets
T4	40	8	64	4MB	0.5TB	2*G2	4
T5	28	16	128	8MB	0.5TB	2*G3	8
M5	28	6	48	48MB	1TB	2*G3	32
M6	28	12	96	48MB	1TB	2*G3	96

ORACLE

Oracle SPARC M6 Core (2013)

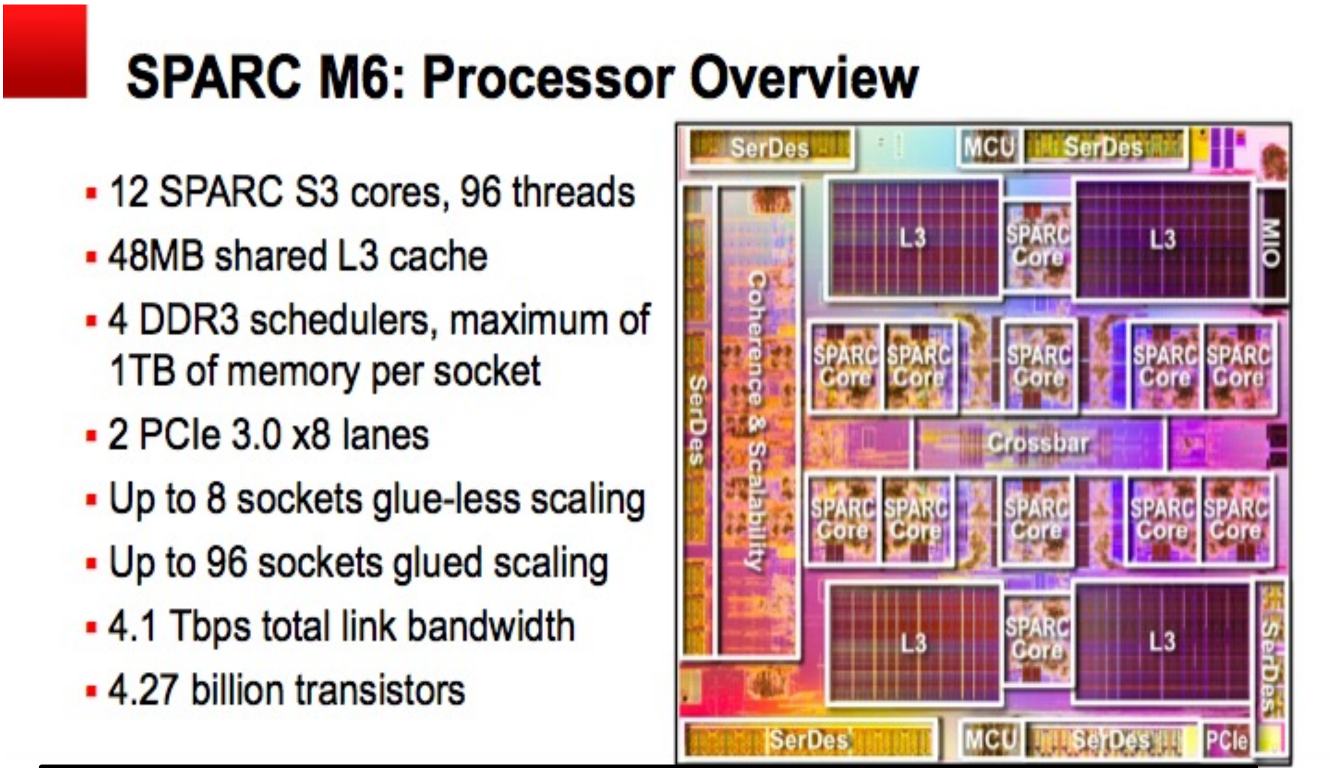
SPARC S3 Core

- Dual-issue, out-of-order
- Integrated encryption acceleration instructions
- Enhanced instruction set to accelerate Oracle SW stack
- 1-8 strands, dynamically threaded pipeline



ORACLE

Oracle SPARC M6 (2013)

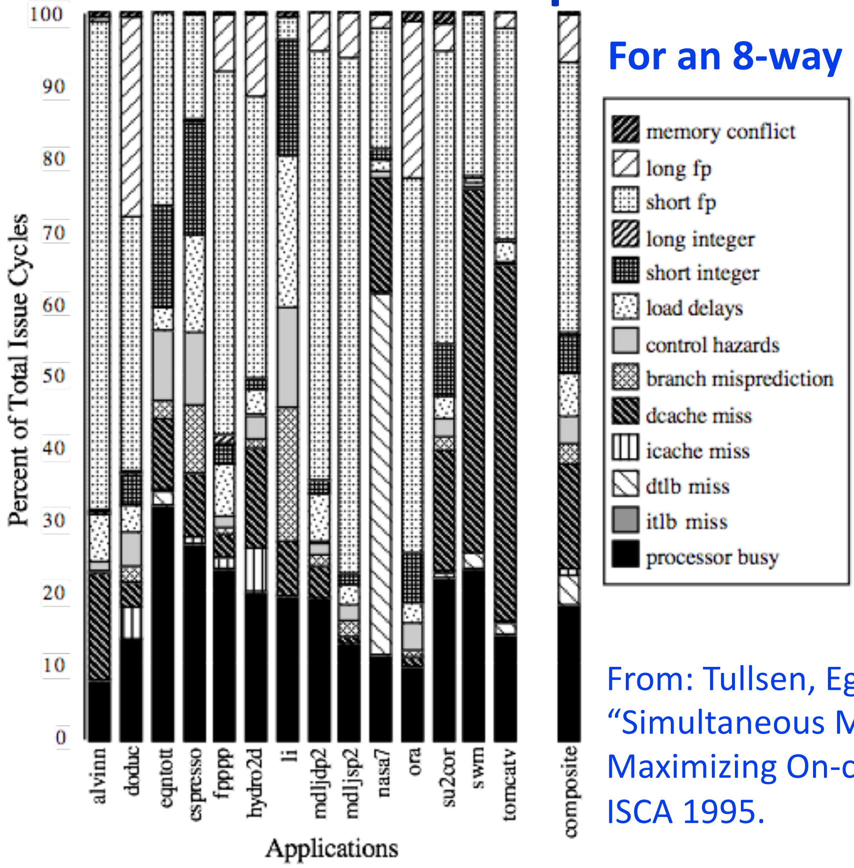


Oracle ended SPARC programs after M8 in 2017

Simultaneous Multithreading (SMT) for OoO Superscalars

- Techniques presented so far have all been "vertical" multithreading where each pipeline stage works on one thread at a time
- SMT uses fine-grain control already present inside an OoO superscalar to allow instructions from multiple threads to enter execution on same clock cycle. Gives better utilization of machine resources.

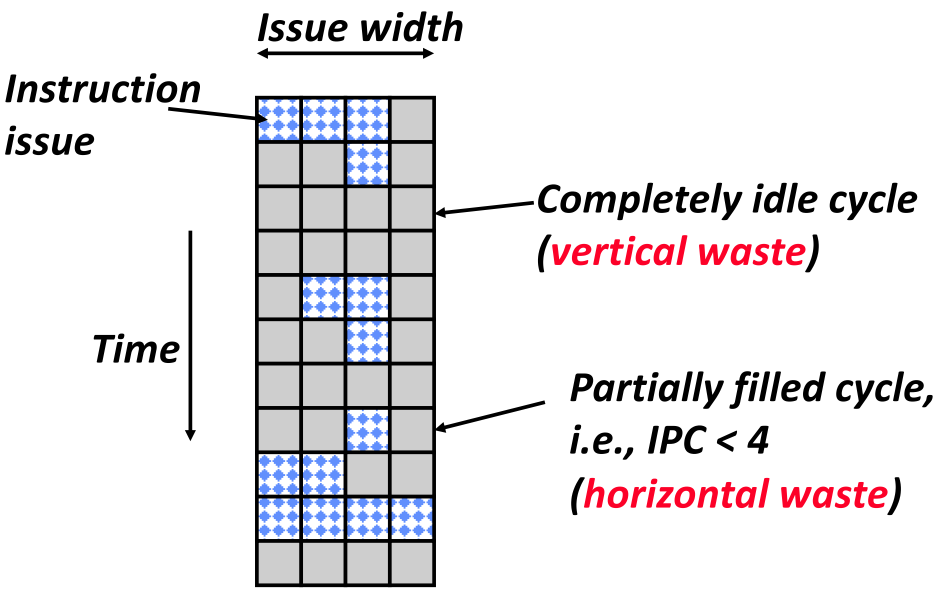
For most apps, most execution units lie idle in an OoO superscalar



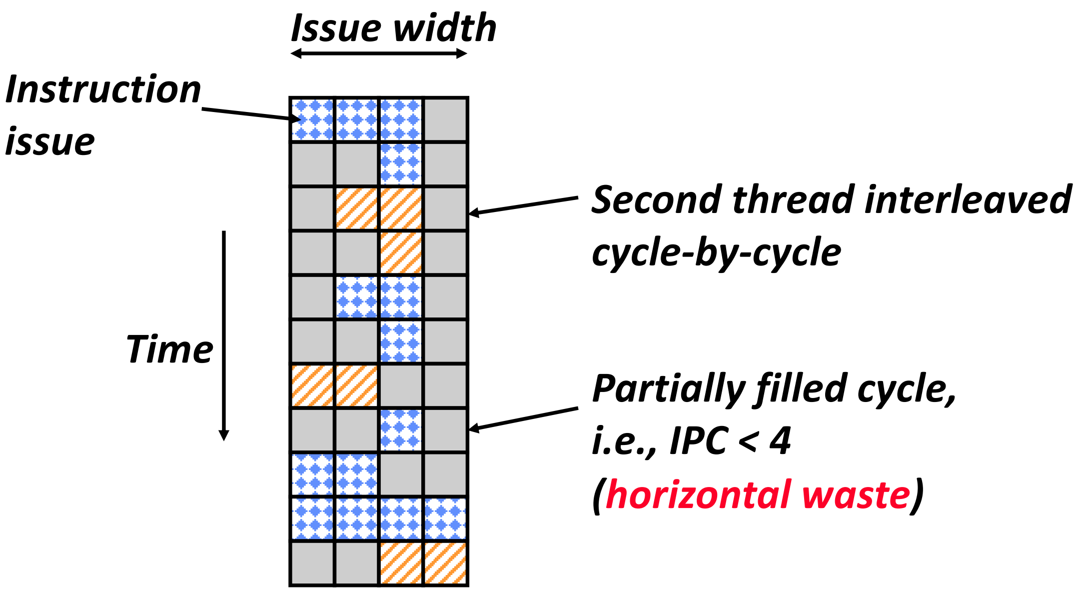
For an 8-way superscalar.

From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism", ISCA 1995.

Superscalar Machine Efficiency

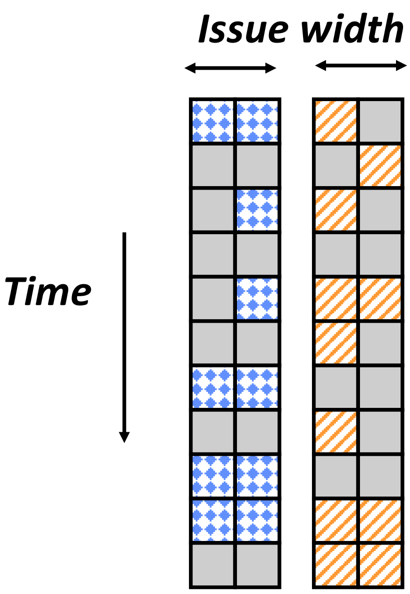


Vertical Multithreading



- Cycle-by-cycle interleaving removes vertical waste, but leaves some horizontal waste

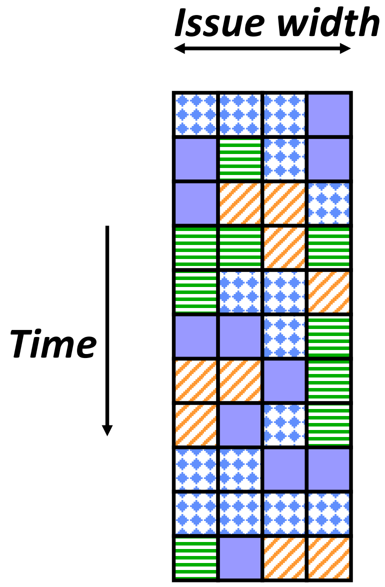
Chip Multiprocessing (CMP)



- What is the effect of splitting into multiple processors?
  - reduces horizontal waste,
  - leaves some vertical waste, and
  - puts upper limit on peak throughput of each thread.

Ideal Superscalar Multithreading

[Tullsen, Eggers, Levy, UW, 1995]



- Interleave multiple threads to multiple issue slots with no restrictions

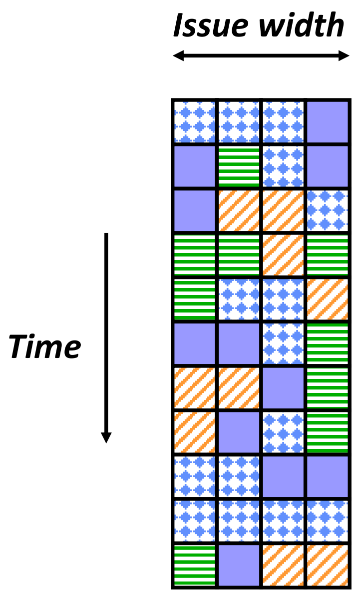
O-o-O Simultaneous Multithreading

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

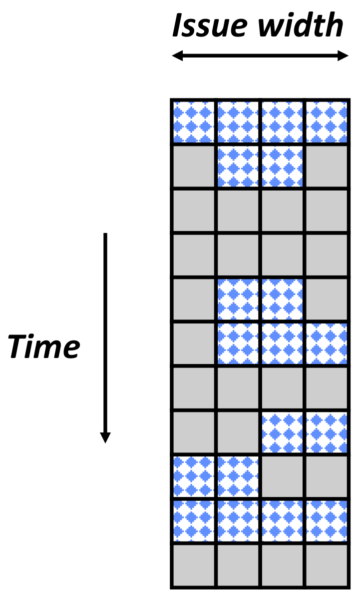
- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OOO instruction window already has most of the circuitry required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

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)

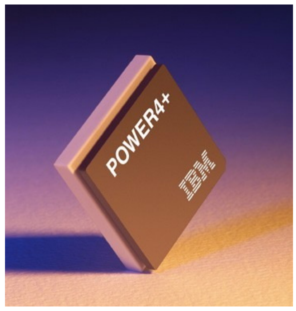


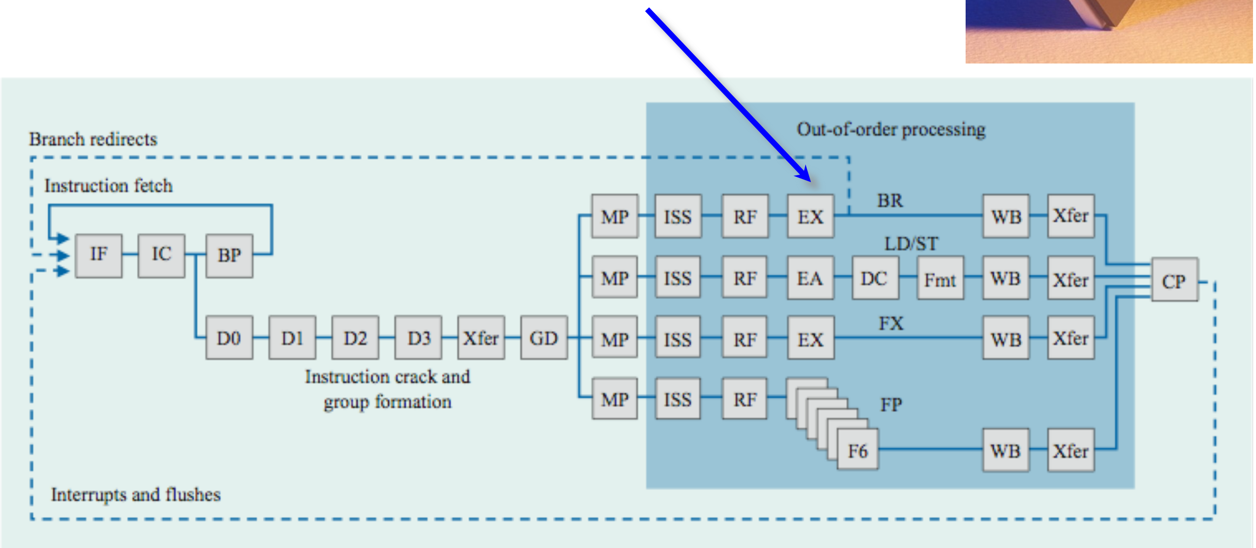
Pentium-4 Hyperthreading (2002)

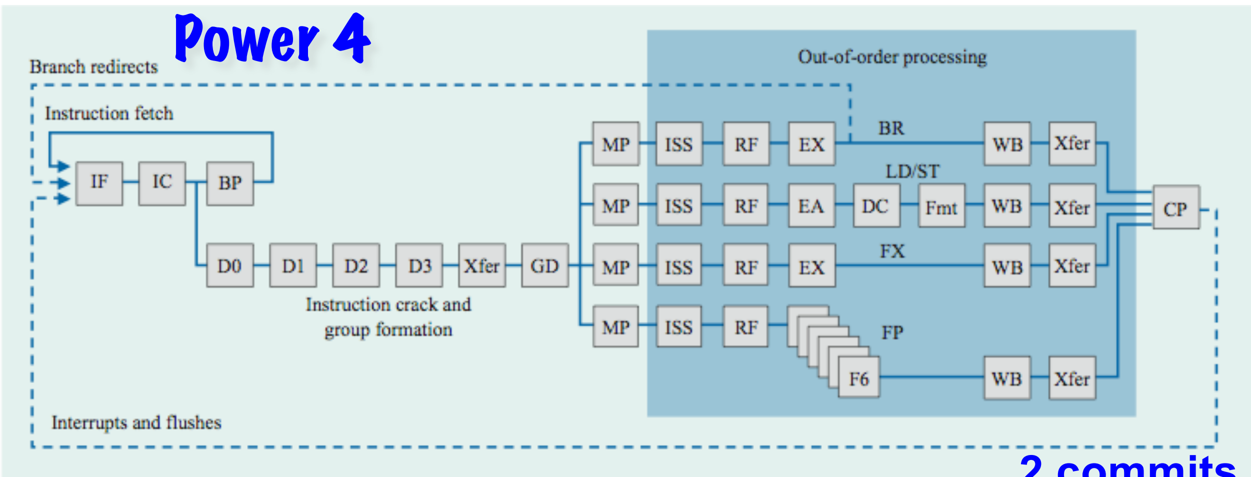
- First commercial SMT design (2-way SMT)
- Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress
  - No logical processor can 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.
- First Intel Atom (in-order x86 core) has two-way vertical multithreading
  - Hyperthreading == (SMT for Intel OoO & Vertical for Intel InO)

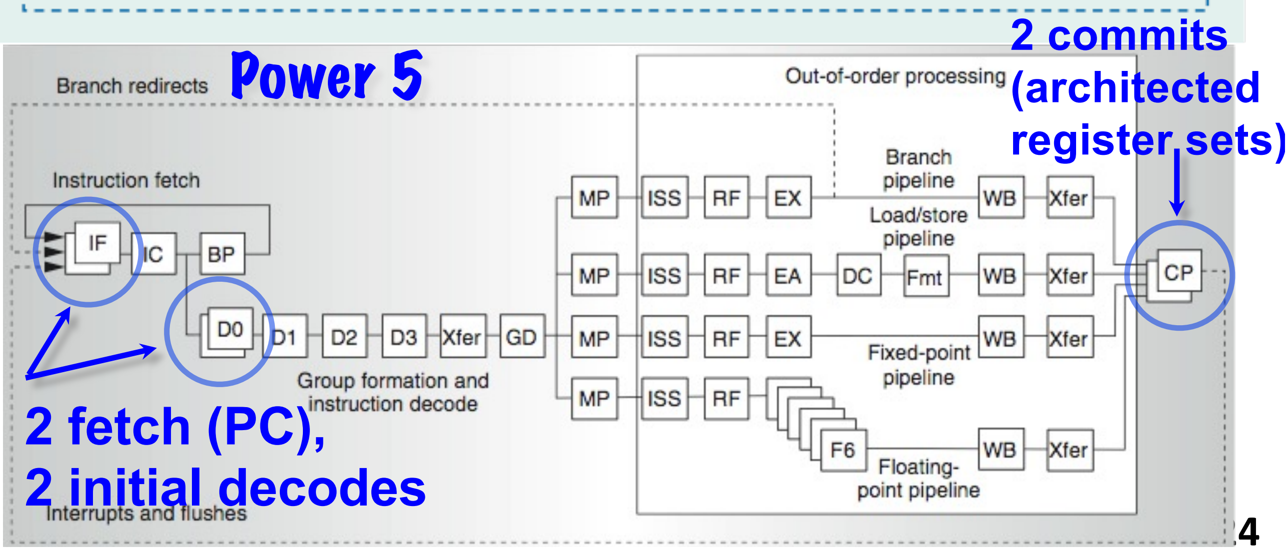
IBM Power 4

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

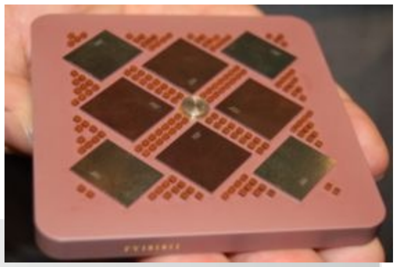


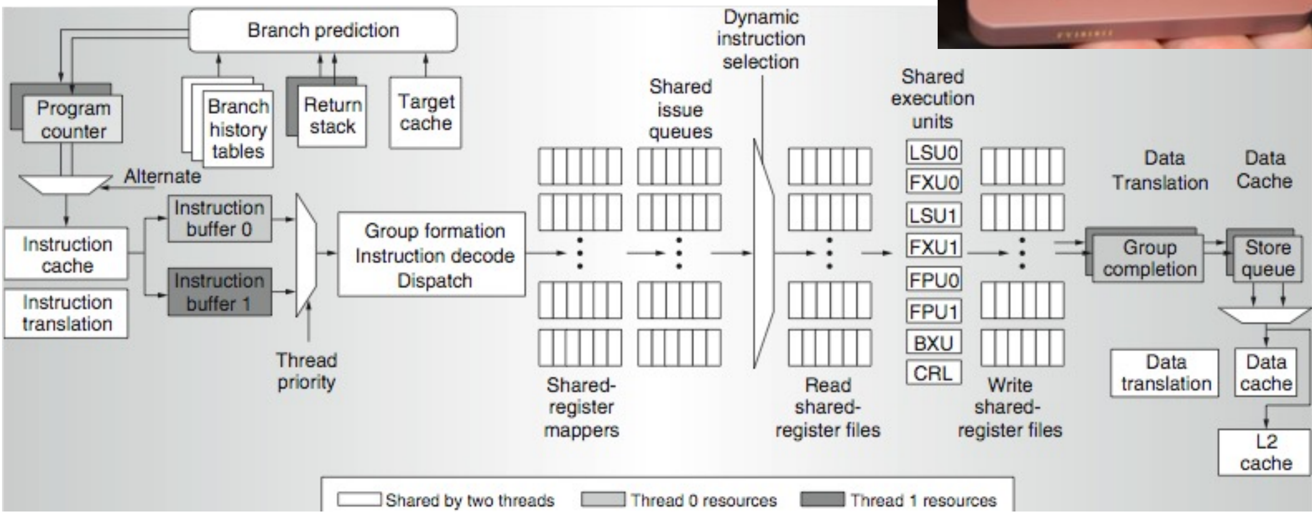






Power 5 data flow ...





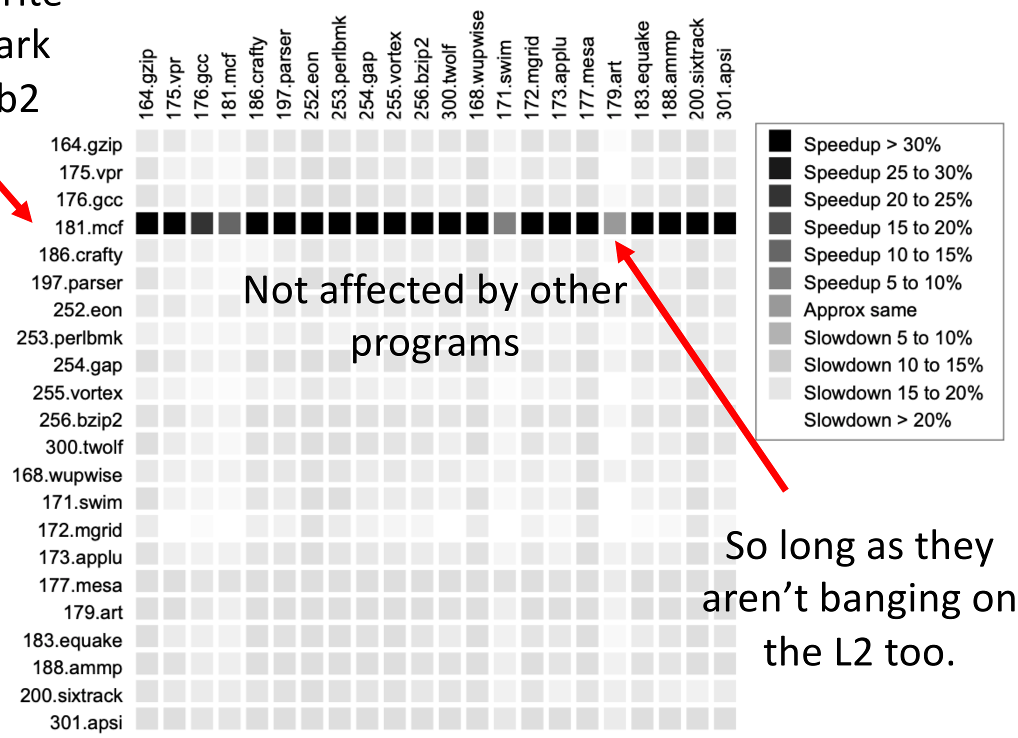
Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

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 Performance: Application Interaction

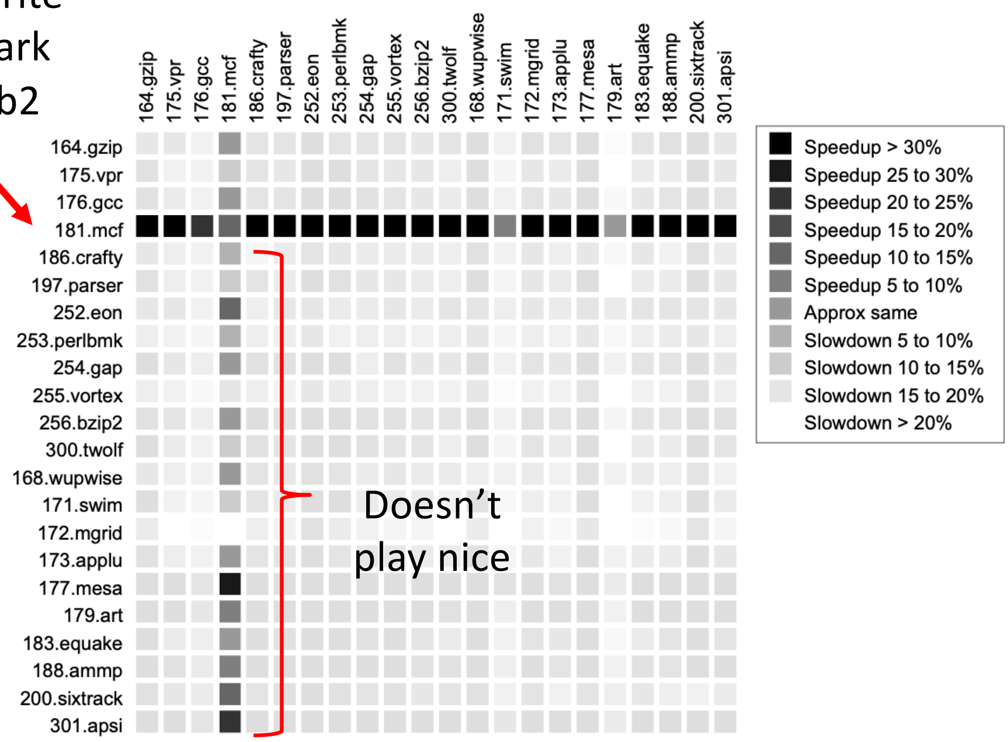
Your favorite benchmark from Lab2



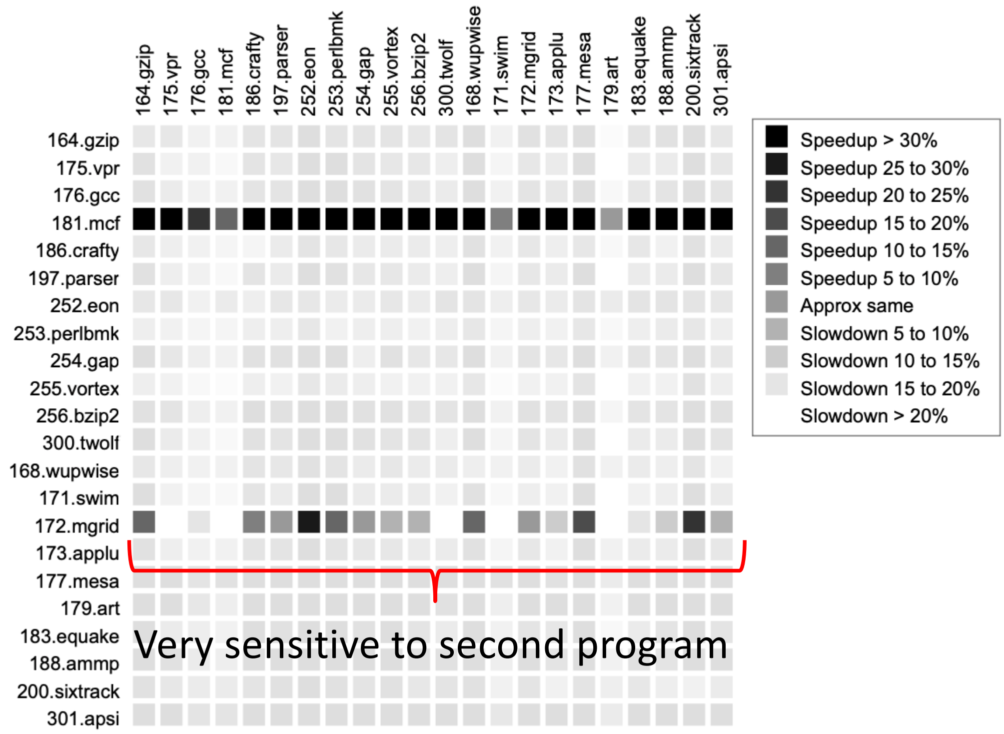
So long as they aren't banging on the L2 too.

SMT Performance: Application Interaction

Your favorite benchmark from Lab2



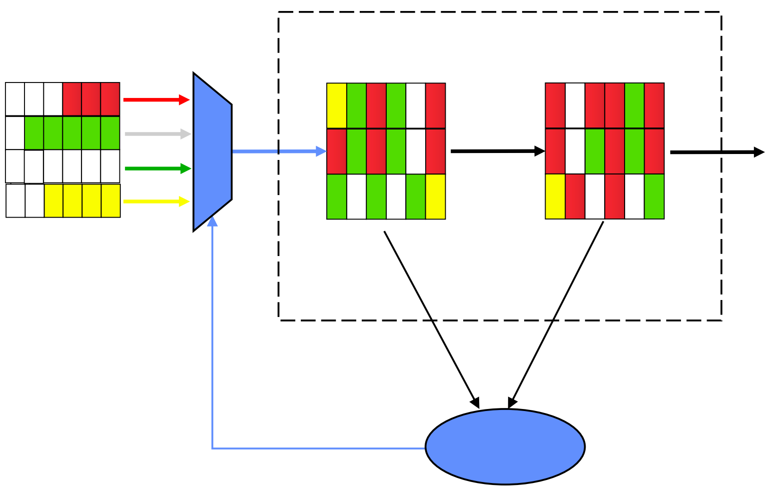
SMT Performance: Application Interaction



Bulpin et al, "*Multiprogramming Performance of Pentium 4 with Hyper-Threading*"

Icount Choosing Policy

Fetch from thread with the least instructions in flight.



Why does this enhance throughput?

SMT & Security

- Most hardware attacks rely on shared hardware resources to establish a side-channel
  - Eg. Shared outer caches, DRAM row buffers
- SMT gives attackers high-BW access to previously private hardware resources that are shared by co-resident threads:
- TLBs: TLBleed (June, '18)
- L1 caches: CacheBleed (2016)
- Functional unit ports: PortSmash (Nov, '18)

OpenBSD 6.4 → Disabled HT in BIOS, AMD SMT to follow

Summary: Multithreaded Categories