In a classic VLIW, compiler is responsible for avoiding all hazards -> simple hardware, complex compiler.

Later VLIWs added more dynamic hardware interlocks, which reduce relative hardware benefits

Use loop unrolling and software pipelining for loops, trace scheduling for more irregular code

Static scheduling difficult in presence of unpredictable branches and variable latency memory

VLIW has failed in general-purpose computing, but still used in deeply embedded processors and DSPs
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

T1: LD x1,0(x2)  
T2: ADD x7,x1,x4  
T3: XORI x5,x4,12  
T4: SD 0(x7),x5  
T1: LD x5,12(x1)  

Prior instruction in a thread always completes write-back before next instruction in same thread reads register file
CDC 6600 Peripheral Processors
(Cray, 1964)

- First multithreaded hardware
- 10 “virtual” I/O processors
- Fixed interleave on simple pipeline
- Pipeline has 100ns cycle time
- Each virtual processor executes one instruction every 1000ns
- Accumulator-based instruction set to reduce processor state
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)
Tera/Cray MTA (1990-2015)

- Up to 256 processors
- Up to 128 active threads per processor
- Processors and memory modules populate a sparse 3D torus interconnection fabric
- Flat, shared main memory
  - No data cache
  - Sustains one main memory access per cycle per processor
- GaAs logic in prototype, 1KW/processor @ 260MHz
  - Second version CMOS, MTA-2, 50W/processor
  - Newer version, XMT, fits into AMD Opteron socket, runs at 500MHz
  - Newer versions (2011), XMT-2, has higher memory bandwidth and capacity
  - Discontinued 2015?
MTA Pipeline

- Every cycle, one VLIW instruction from one active thread is launched into pipeline
- Instruction pipeline is 21 cycles long
- Memory operations incur ~150 cycles of latency

Assuming a single thread issues one instruction every 21 cycles, and clock rate is 260 MHz...

What is single-thread performance?

Effective single-thread issue rate is $\frac{260}{21} = 12.4$ MIPS
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

- 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
MIT Alewife (1990)

- Modified SPARC chips
  - register windows hold different thread contexts
- Up to four threads per node
- Thread switch on local cache miss
IBM PowerPC RS64-IV (2000)

- Commercial coarse-grain multithreading CPU
- Based on PowerPC with quad-issue in-order five-stage pipeline
- Each physical CPU supports two virtual CPUs
- On L2 cache miss, pipeline is flushed and execution switches to second thread
  - short pipeline minimizes flush penalty (4 cycles), small compared to memory access latency
  - flush pipeline to simplify exception handling
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

<table>
<thead>
<tr>
<th></th>
<th>nm</th>
<th>Cores</th>
<th>Threads</th>
<th>L3$</th>
<th>Memory per Socket</th>
<th>PCIe</th>
<th>Max. Sockets</th>
</tr>
</thead>
<tbody>
<tr>
<td>T4</td>
<td>40</td>
<td>8</td>
<td>64</td>
<td>4MB</td>
<td>0.5TB</td>
<td>2*G2</td>
<td>4</td>
</tr>
<tr>
<td>T5</td>
<td>28</td>
<td>16</td>
<td>128</td>
<td>8MB</td>
<td>0.5TB</td>
<td>2*G3</td>
<td>8</td>
</tr>
<tr>
<td>M5</td>
<td>28</td>
<td>6</td>
<td>48</td>
<td>48MB</td>
<td>1TB</td>
<td>2*G3</td>
<td>32</td>
</tr>
<tr>
<td>M6</td>
<td>28</td>
<td>12</td>
<td>96</td>
<td>48MB</td>
<td>1TB</td>
<td>2*G3</td>
<td>96</td>
</tr>
</tbody>
</table>
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 SPARC M6 (2013)

SPARC M6: Processor Overview

- 12 SPARC S3 cores, 96 threads
- 48MB shared L3 cache
- 4 DDR3 schedulers, maximum of 1TB of memory per socket
- 2 PCIe 3.0 x8 lanes
- Up to 8 sockets glue-less scaling
- Up to 96 sockets glued scaling
- 4.1 Tbps total link bandwidth
- 4.27 billion transistors

Oracle ended SPARC programs after M8 in 2017
CS152 Administrivia

- PS 3 due today
- PS 4 out Wednesday March 17, due Monday April 5
- Lab 3 due Monday April 5
- Midterm 1 regrade requests by Tuesday March 16 11:59PM

- No lectures or sections next week!
  - Spring Break (March 22-26)
CS252 Administrivia

- Readings on OoO architectures this Thursday
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.

Superscalar Machine Efficiency

- **Issue width**
  - **Instruction issue**
  - **Completely idle cycle** (vertical waste)
  - **Partially filled cycle, i.e., IPC < 4** (horizontal waste)
Vertical Multithreading

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

**Issue width**

- Instruction issue

- Second thread interleaved cycle-by-cycle

- Partially filled cycle, i.e., IPC < 4 (horizontal waste)
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 4

- 2 fetch (PC), 2 initial decodes
- 2 commits (architected register sets)

Power 5

- 2 fetch (PC), 2 initial decodes
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
  - FlPt. apps had most cache conflicts and least gains
SMT Performance: Application Interaction

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

Not affected by other programs

Bulpin et al, “Multiprogramming Performance of Pentium 4 with Hyper-Threading”
SMT Performance: Application Interaction

Your favorite benchmark from Lab2

Bulpin et al, “Multiprogramming Performance of Pentium 4 with Hyper-Threading”
SMT Performance: Application Interaction

<table>
<thead>
<tr>
<th></th>
<th>164.gzip</th>
<th>175.vpr</th>
<th>176.gcc</th>
<th>181.mcf</th>
<th>186.crafty</th>
<th>197.parser</th>
<th>252.eon</th>
<th>253.perlbmk</th>
<th>254.gap</th>
<th>255.vortex</th>
<th>256.bzip2</th>
<th>300.twolf</th>
<th>168.wupwise</th>
<th>171.swim</th>
<th>172.mgrid</th>
<th>173.applu</th>
<th>177.mesa</th>
<th>179.art</th>
<th>183.equate</th>
<th>188.ammp</th>
<th>200.sixtrack</th>
<th>301.apsi</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.gzip</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>175.vpr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>176.gcc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>181.mcf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>186.crafty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>197.parser</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>252.eon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>253.perlbmk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>254.gap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>255.vortex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256.bzip2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300.twolf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>168.wupwise</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>171.swim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>172.mgrid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>173.applu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>177.mesa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>179.art</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>183.equate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>188.ammp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200.sixtrack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>301.apsi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Very sensitive to second program

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?
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

- **Superscalar**
- **Fine-Grained**
- **Coarse-Grained**
- **Multiprocessing**
- **Simultaneous Multithreading**

**Thread Color Code:**
- Blue: Thread 1
- Red: Thread 2
- Yellow: Thread 3
- Green: Thread 4
- Purple: Thread 5
- White: Idle slot
Acknowledgements

This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:

– Arvind (MIT)
– Scott Beamer (UCB)
– Joel Emer (Intel/MIT)
– James Hoe (CMU)
– John Kubiatowicz (UCB)
– David Patterson (UCB)