# ECE 571 – Advanced Microprocessor-Based Design Lecture 5

Vince Weaver

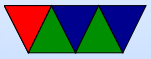
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

13 September 2022

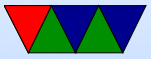
# Announcements

- Homeworks
  - HW#1 graded
  - HW#2 due Friday (a reading)
- Optional Readings
  - Pipeline Discussion: Computer Organization (RiscV) / Patterson and Hennesey
    Section 4.11 "Real Stuff: The ARM Cortex-A53 and Intel Core i7 Pipelines"
  - Power/Energy: Computer Architecture / Hennesey



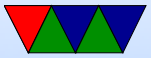
and Patterson

Section 1.5 “Trends in Power and Energy in Integrated Circuits”

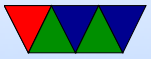


# HW#1 Review – Aggregate counts

- bzip2 benchmark – what does it do?
- 19 billion instructions +/- 1000 or so
  (this is test input maybe?)
- 12 billion cycles +/- 100 million
  why would cycles vary?
  Better than last time
- Didn't ask, but cycles/s = 3.3GHz or so (actual=2.6)
- Didn't ask, but roughly what's the IPC? 1.6 or so
- Reversed: similar for instructions but different for cycles



– HW2 will show you why I asked that



# HW#1 Review – Sampling

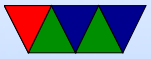
- Perf record: 6.4s (why slower?)

```
39.19%  bzip2    bzip2                  [.] mainSort
30.53%  bzip2    bzip2                  [.] mainGtU
12.54%  bzip2    bzip2                  [.] handle_compress.isra.0
 8.10%  bzip2    bzip2                  [.] generateMTFValues
 7.72%  bzip2    bzip2                  [.] BZ2_compressBlock
```

- Valgrind, 1m28s == roughly 20 times slower?

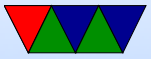
```
8,150,804,034 (41.35%)  blocksort.c:mainSort
5,619,003,640 (28.51%)  blocksort.c:mainGtU
2,434,222,854 (12.35%)  bzlib.c:handle_compress.isra.0
1,601,050,270 ( 8.12%)  compress.c:generateMTFValues
1,511,856,286 ( 7.67%)  compress.c:BZ2_compressBlock
```

- Gprof, 4.6s



# Different results, using function entry instead of exact instruction count for sampling?

| time | seconds | seconds | calls | s/call | s/call | name |
|---|---|---|---|---|---|---|
| 51.13 | 1.82 | 1.82 | 53 | 0.03 | 0.05 | mainSort |
| 20.65 | 2.56 | 0.74 | 89518573 | 0.00 | 0.00 | mainGtU |
| 13.48 | 3.04 | 0.48 | 53 | 0.01 | 0.01 | generateMTFVal |
| 7.02 | 3.29 | 0.25 | 12223 | 0.00 | 0.00 | default_bzallo |
| 5.62 | 3.49 | 0.20 | 53 | 0.00 | 0.06 | BZ2_compressBl |

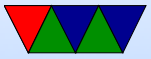


# HW#1 Review – Skid

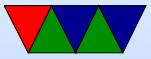
- Skid instructions – mov is more likely than sub?
- movzbl = move 8-bit byte from memory into 32-bit long register, zero extending

```
      |     n = ((Int32)block[ptr[unLo]+d]) - med;
 1.17 |       mov    (%r11),%esi
 0.78 |       lea    (%rbx,%rsi,1),%eax
 1.51 |       movzbl 0x0(%r13,%rax,1),%eax
 5.47 |       sub    %r9d,%eax
      |     if (n == 0) {
 1.77 |       test   %eax,%eax
```

instructions:uppp ppp = precise IP, how much skip 0=arbitrary, 1=constant, 2=request 0, 3=require 0

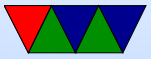


```
0.88 |       mov    (%r11),%esi
1.48 |       lea    (%rbx,%rsi,1),%eax
5.51 |       movzbl 0x0(%r13,%rax,1),%eax
1.67 |       sub    %r9d,%eax
```



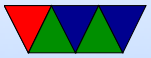
# Out-of-Order Processors

- Tries to exploit instruction-level parallelism
- Instead of being stuck waiting for a resource to become available for an instruction (cache, multiplier, etc) keep executing instructions beyond as long as there are no dependencies
- Need to insure that instructions commit in order
  Need to make sure loads/stores happen in order.



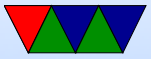
# Ensuring In-order Results with OoO

- Need to track the original order instructions would retire (graduate)
- Some chips use re-order buffer (ROB)
- Intel chips used register renaming and register alias table RAT
- historical: scoreboards and Tomosulu algorithm



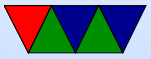
# Precise Exceptions

- One example is skid from HW#1
- What happens on exception? (interrupt, branch mispredict, etc)
- Many instructions "in-flight", which one caused exception?
- Need to find out which one did, then back things out so can restart
- A lot of trouble



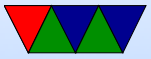
# Speculative Execution

- What do you do about branches?
  They happen often
- Stall?
- Branch prediction, guess which way things go
- What happens when you're wrong? Need to flush all the pipelines and re-start



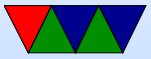
# Benefits to OoO

- Can get a nice performance boost



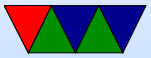
# Downside to OoO

- Security issues: spectre, meltdown
- A lot more complex than in-order
- Can waste power, especially if you re-execute or throw out code due to speculative execution



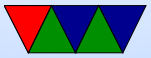
# Perf Counters related to Stalls

- Front-end stalls – fetch, decode, icache misses
- Back-end stalls – memory accesses

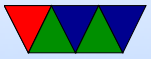


# Instruction Level Parallelism

- Using super-scalar and/or OoO (Out of Order) execution try to find parallelism within your serial code
- Chip companies want to speed up existing code. Why? (it's a pain to change, you might not have source, etc.)

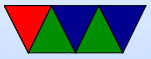


# Other Ways to get better Parallelism



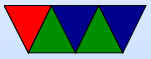
# SIMD / Vector Instructions

- x86: MMX/SSE/SSE2/AVX/AVX2
  semi-related FMA
- MMX (mostly deprecated), AMD's 3DNow! (deprecated)
- PowerPC Altivec
- ARM: Neon / A64 SIMD / "Scalabale Vector Extensions" SVE



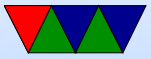
# SSE / x86

- SSE (streaming SIMD): 128-bit registers XMM0 - XMM7, can be used as 4 32-bit floats
- SSE2 : 2*64bit int or float, 4 * 32-bit int or float, 8x16 bit int, 16x8-bit int
- SSE3 : minor update, add dsp and others
- SSSE3 (the s is for supplemental): shuffle, horizontal add
- SSE4 : popcnt, dot product



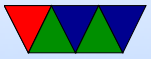
# AVX / x86

- AVX (advanced vector extensions) – now 256 bits, YMM0-YMM15 low bits are the XMM registers. Now twice as many.
  Also adds three operand instructions a=b+c
- AVX2 – 3 operand Fused-Multiply Add, more 256 instructions
- AVX-512 – version used on Xeon Phis (knights landing) and Skylake – now 512 bits, ZMM0-XMM31



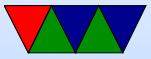
# Multi-core / Chip-Multi Processing (CMP)

- Moore's law gives you lots of transistors. Hit limit of how fast to make a single processor, so why not just put more on the die?
- Exploits multi-programmed parallelism rather than instruction-level parallelism

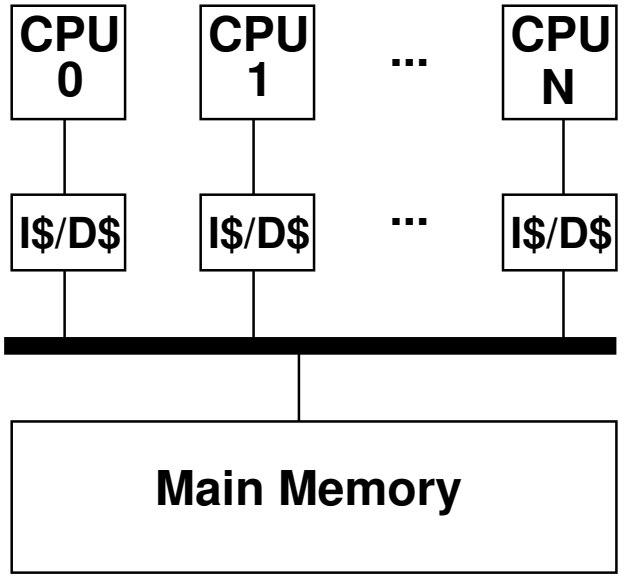


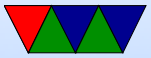
# Multi-core Downside

- Have to write parallel code, which is hard
- Otherwise need to have lots of programs running at once, which isn't always possible



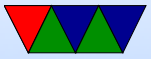
# CMP Diagram



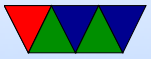



# Hardware Multi-threading

- SMT (simultaneous multithreading), Intel Hyperthreading
- Hyrbid of multi-core and multi-pipeline
- Your pipelines might not always be full, especially if waiting on memory
- Why not duplicate fetch/decode logic, and have two programs execute at once on same set of pipelines.
- If one is idle/stalled, run instructions from other thread
- Looks to OS as if you have two cores, but really just one with two instruction dispatch stages

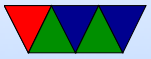


- Extra logic to make sure that pipelines used fairly, the results get committed to the right register file, etc.



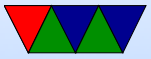
# SMT Variations

- Fine-grained – rotate threads every cycle
- Coarse-grained – rotate threads only if long latency event happens (cache miss)
- Simultaneous – issue from any combination of threads, to maximize use of pipeline (have to be superscalar)

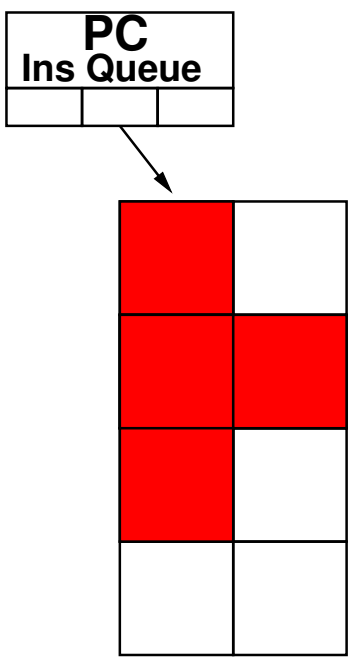


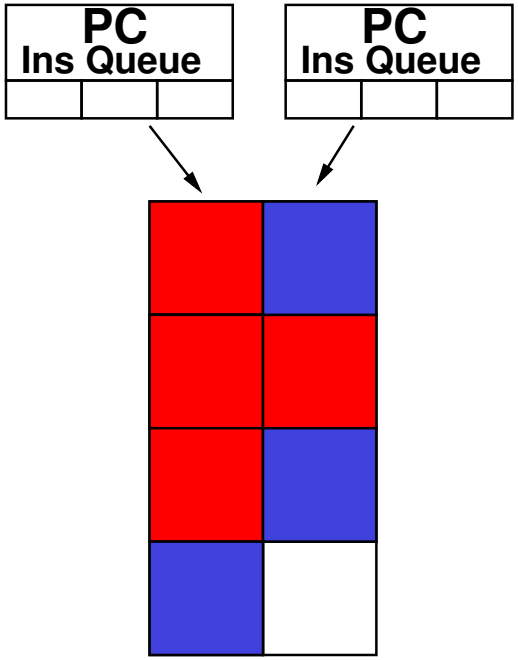
# SMT Downsides

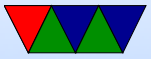
- Can actually slow down code (especially if both threads trying to use same functional units, also if both using memory heavily as cache is often shared)
- Security? Information Leakage?



# SMT Diagram

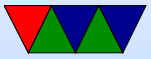


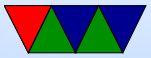
# Cache Coherency

- How do you handle data being worked on by multiple processors, each with own cache of main memory?
- Cache coherency protocols.
- Many and varied. MESI is a common one
- Directory vs Snoopy



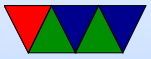
# MESI

- Modified, Exclusive, Shared, Invalid

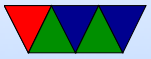


# Real-World Pipelining Examples (from P&H)

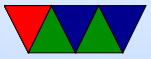
- ARM Cortex-A53 (found in Pi3)
  - Eight-stage pipeline
  - Dynamic multi-issue, two instructions
  - Static in-order pipeline
  - First 3 stages fetch two insns at a time, filling a 13-entry instruction queue (branch predictors)
  - Pipelines: one for load, one for store, two for ALU, one multiply, one divide, one FP/SIMD (mul/div/sqrt)



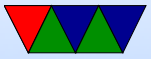
one FP/SIMD for other
  - What's the peak possible IPC?
  - Patterson and Hennesey report SPEC CPU 2006 INT results. Best case is hmmer (search for gene sequence) with IPC 1.03 (CPI 0.97). Worst is mcf (public transit vehicle scheduling) IPC 0.12 (CPI 8.56). Mostly memory constrained.
  - In-order so depends a lot on compiler to get good performance.
  - 100mW (1 core at 1GHz)
- Intel Core i7 920 (Nehalem, 2008)



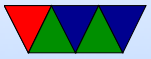
- Decodes CISC instructions to micro-ops
- Can issue up to 6 micro-ops per cycle
- 14 pipeline stages
- dynamic out-of-order with speculation
- register renaming, useful with speculation, as no need to store snapshot to undo speculation, just mark the speculated register results as invalid
- Instruction fetch, fetches 16 bytes. If wrong, 15 cycle penalty
- Predecode instruction buffer – transform 16 bytes (x86 insns 1-15 bytes) into x86 insns



- 18-instruction instruction queue.
- Micro-op decode – three decoders handle decode of instructions that map to 1 uop. One other handles microcode engine that produces longer sequences, up to 4uops a cycle.
- Can also do micro-op fusion (fuse two different insns into one uops, such as cmp/branch)
- Micro-ops go ins a 28-entry uop buffer
  Loop Stream Detector – if code is in tight loop (less than 28 insns) it can execute from this buffer and not need to fetch.



- Instruction issue. Reservation station. Up to six uops can be issued
- Finished instructions go back to reservation station and retirement unit, wait to update register state when determined it is no longer speculative.
- Once instruction hits the head of the reorder buffer, instruction commits and is removed from re-order buffer
- Even though 6 uops can issue, only 4 can be finished a turn? What's the peak IPC? (4)
- Again, SPECCPU. Best is libquantum IPC=2.2 (CPI



0.44). Worst, again, mcf IPC=0.37 (CPI=2.67)

- Where do the wasted cycles go? Stalls? But also mis-speculation where work is done and then thrown out.
- 130 Watts (2.66GHz)