ECE 571 – Advanced Microprocessor-Based Design Lecture 7

Vince Weaver

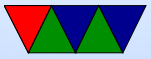
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

16 September 2020

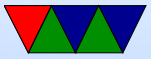
Announcements

- 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

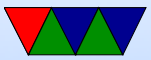
Section 1.5 “Trends in Power and Energy in Integrated Circuits”



HW#1 Review

- bzip2 benchmark – what does it do?
- 19 billion instructions +/- 1000 or so
  (this is test input maybe?)
- 13 billion cycles +/- 100 million
  why would cycles vary?
- Didn't ask, but cycles/s = 2.7GHz or so (actual=2.6)
- Didn't ask, but roughly what's the IPC? 1.5 or so
- Reversed: similar – HW2 will show you why I asked that
- Perf record: 6.4s (why slower?)

```
57.16%  bzip2    bzip2                [.] mainSort
```

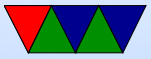


```
 17.57%  bzip2    bzip2                [.] BZ2_compressBlock
 11.90%  bzip2    bzip2                [.] mainGtU.part.0
 11.20%  bzip2    bzip2                [.] handle_compress.isra.2
  0.94%  bzip2    bzip2                [.] BZ2_blockSort
```

- Valgrind, 1m18s == roughly 20 times slower?

```
11,291,448,187  blocksort.c:mainSort [/opt/ece571/401.bzip2]
 3,381,835,437  compress.c:BZ2_compressBlock [/opt/ece571/401.bzip2]
 2,138,813,059  bzlib.c:handle_compress.isra.2 [/opt/ece571/401.bzip
 1,958,107,443  blocksort.c:mainGtU.part.0 [/opt/ece571/401.bzip2]
   165,396,105  blocksort.c:BZ2_blockSort [/opt/ece571/401.bzip2]
```

- Gprof, also 4.3s
  Different results, using function entry instead of exact instruction count for sampling?
  Also, using older gcc, newer versions it's broken on x86_64?



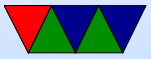
```
time   seconds   seconds    calls   s/call   s/call  name
 70.77      2.59     2.59       53     0.05     0.05  mainSort
 18.58      3.27     0.68       53     0.01     0.06  BZ2_compressBl
  8.20      3.57     0.30    12223     0.00     0.00  default_bzallo
  1.09      3.61     0.04       53     0.00     0.05  BZ2_blockSort
  0.82      3.64     0.03  1856468     0.00     0.00  add_pair_to_bl
```

- Skid instructions – mov is more likely than sub?

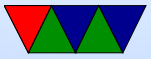
```
      |                    n = ((Int32)block[ptr[unLo]+d]) - med;
 1.17 | 5f0:   mov     (%r10),%edx
 0.61 |        lea     (%rdx,%r13,1),%eax
 1.11 |        movzbl  (%r15,%rax,1),%eax
 2.70 |        sub     %r9d,%eax
      |                    if (n == 0) {
 1.08 |        cmp     $0x0,%eax
```

instructions:uppp

```
      |                    n = ((Int32)block[ptr[unLo]+d]) - med;
```

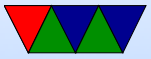


```
0.45 | 5f0:    mov     (%r10),%edx
0.86 |         lea     (%rdx,%r13,1),%eax
3.53 |         movzbl  (%r15,%rax,1),%eax
1.25 |         sub     %r9d,%eax
     |                    if (n == 0) {
1.15 |         cmp     $0x0,%eax
```



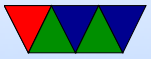
Multi-core

- More'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

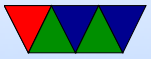


Multi-threaded

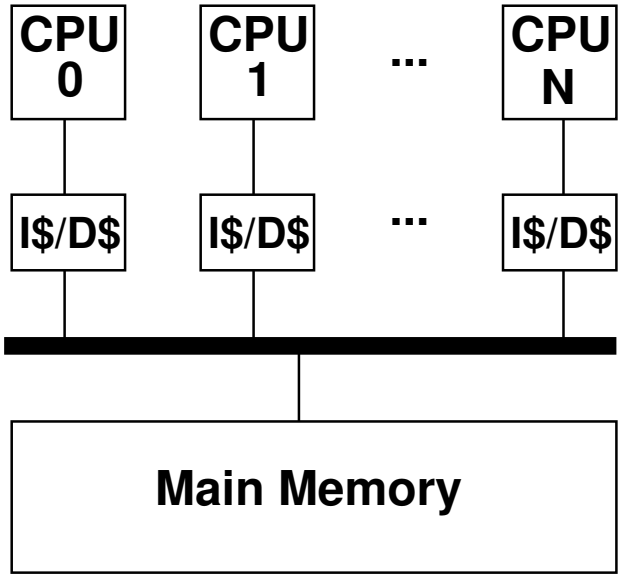
- SMT (simultaneous multithreading), Intel Hyperthreading
- Hyrbid of multi-core and multi-pipeline
- Your pipelines might not always be full, especially if waiting on memory
- Why not duplicate fetch/decode logic, and have two programs execute at once on same set of pipelines.
- If one is idle/stalled,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

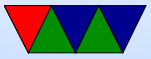


- Looks to OS as if you have two cores, but really just one with two instruction dispatch stages
- Extra logic to make sure that pipelines used fairly, the results get committed to the right register file, etc.



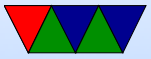
CMP Diagram





Hardware Multi-Threading

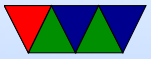
- Idea is to re-use a pipeline to execute multiple threads at once, *without* fully replicating the entire CPU (so less than multicore)
- You will have to replicate some things (program counter for each, etc)
- Usually they appear to the CPU as full separate processors even though they are not.
- Various ways to do this:



- 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)

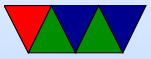
- Why do this? Often on superscalar running only one thread will leave parts idle, try to make use of these.

- Bad side effects?
  - Can actually slow down code (especially if both threads

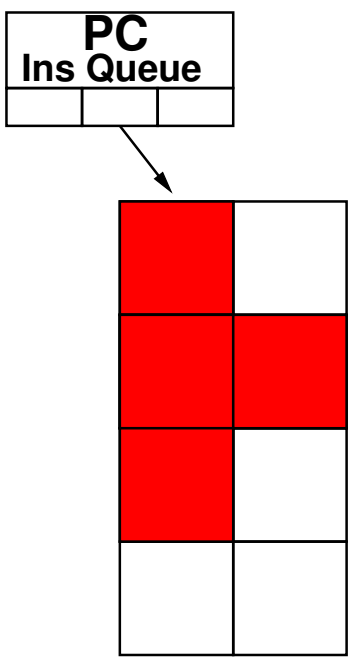


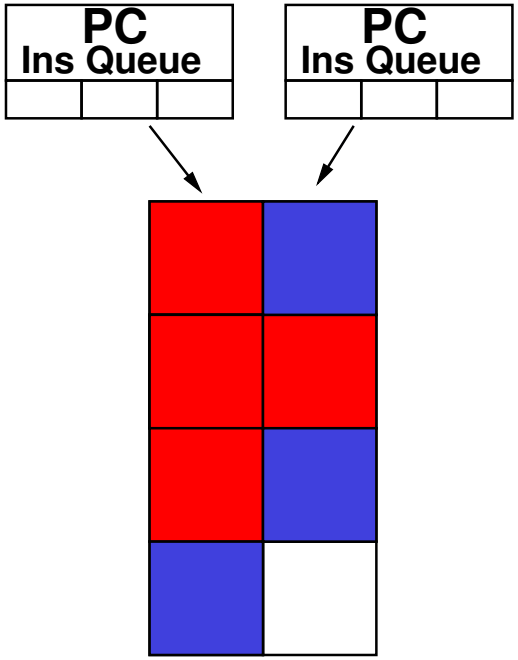
trying to use same functional units, also if both using memory heavily as cache is often shared)
  - Security? Information Leakage?

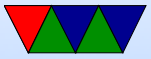
- Sometimes see it talked about as SMT (Simultaneous Multithreading), Intel Hyperthreading is more or less the same thing



SMT Diagram

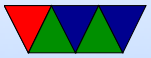






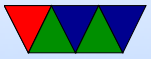
Cache Coherency

- 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



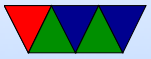
MESI

- Modified, Exclusive, Shared, Invalid

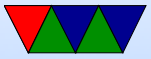


Real-World Pipelining Examples (from P&H)

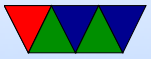
- ARM Cortex-A53 (found in Pi3)
  - Eight-stage pipeline
  - Dynamic multi-issue, two instructions
  - Static in-order pipeline
  - First 3 stags fetch two insns at a time, filling a 13-entry instruction queue (branch predictors)
  - Pipelines: one for load, one for store, two for ALU, one multiply,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



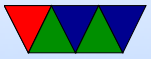
one FP/SIMD for other
  - What's the peak possible IPC?
  - Patterson and Hennesey report SPEC CPU 2006 INT results. Best case is hmmer (search for gene sequence) with IPC 1.03 (CPI 0.97). Worst is mcf (public transit vehicle scheduling) IPC 0.12 (CPI 8.56). Mostly memory constrained.
  - In-order so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



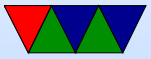
- Decodes CISC instructions to micro-ops
- Can issue up to 6 micro-ops per cycle
- 14 pipeline stages
- dynamic out-of-order with speculation
- register renaming, useful with speculation, as no need to store snapshot to undo speculation, just mark the speculated register results as invalid
- Instruction 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)