ECE 571 – Advanced Microprocessor-Based Design Lecture 5

Vince Weaver

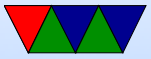
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

6 February 2018

Announcements

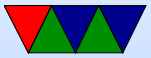
- HW#1 graded
- HW#2 due Thursday



HW#1 Review

- bzip2 benchmark – what does it do?
- 19 billion instructions +/- 400 or so
  (this is test input maybe?)
- 12.5 billion cycles +/- 200 million (more variation that last year)
- 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: 3.5s,

```
56.57%  bzip2    bzip2            [.] mainSort
16.75%  bzip2    bzip2            [.] BZ2_compressBlock
```



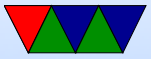
```
  12.65%  bzip2    bzip2                [.] handle_compress.isra.2
  11.61%  bzip2    bzip2                [.] mainGtU.part.0
   0.94%  bzip2    bzip2                [.] BZ2_blockSort
```

- Valgrind, 1m10.189s == roughly 20 times slower?

```
11,252,283,877  blocksort.c:mainSort [/opt/ece571/401.bzip2/bzip2]
 3,228,558,743  compress.c:BZ2_compressBlock [/opt/ece571/401.bzip2/
 2,434,211,849  handle_compress.isra.2 [/opt/ece571/401.bzip2/bzip2]
 1,887,587,824  mainGtU.part.0 [/opt/ece571/401.bzip2/bzip2]
   165,396,105  blocksort.c:BZ2_blockSort [/opt/ece571/401.bzip2/bzi
```

- Gprof, also 3.5s
  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
```



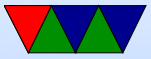
```
 71.65      2.02     2.02        53     0.04     0.04  mainSort
 19.15      2.56     0.54        53     0.01     0.05  BZ2_compressBl
  7.45      2.77     0.21     12223     0.00     0.00  default_bzallo
  1.06      2.80     0.03   1856468     0.00     0.00  add_pair_to_bl
  0.71      2.82     0.02      1272     0.00     0.00  BZ2_hbMakeCode
```

- Skid instructions – mov is more likely than sub?

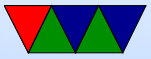
```
  0.66          movzbl (%r10,%rax,1),%eax
                          n = ((Int32)block[ptr[unLo]+d]) - med;
  4.32          sub    %r8d,%eax
                          if (n == 0) {
  1.19          cmp    $0x0,%eax
```

instructions:ppp

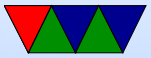
```
perf annotate:
  1.27          mov    0x0(%r13,%rdi,4),%ecx
  0.54          lea    -0x1(%r15),%edi
  0.81          lea    (%r11,%rcx,1),%eax
```



```
4.05          movzbl (%r10,%rax,1),%eax
                        n = ((Int32)block[ptr[unLo]+d]) - med;
1.49          sub    %r8d,%eax
```

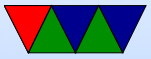


Pause to watch Falcon Heavy Launch

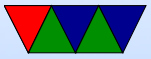


Some Real-World Pipelining/Threading Examples

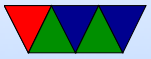
- ARM Cortex-A53 (found in Pi3)
  - Eight-stage pipeline
  - Dynamic multi-issue, two instructions
  - Static in-order pipeline
  - First 3 stags fetch two insns at a time, filling a 13-entry instruction queue (branch predictors)
  - Pipelines: one for load, one for store, two for ALU, one multiply,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



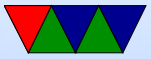
one FP/SIMD for other
  - What's the peak possible IPC?
  - Patterson and Hennesey report SPEC CPU 2006 INT results. Best case is hmmer (search for gene sequence) with IPC 1.03 (CPI 0.97). Worst is mcf (public transit vehicle scheduling) IPC 0.12 (CPI 8.56). Mostly memory constrained.
  - In-order so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



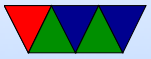
- Decodes CISC instructions to micro-ops
- Can issue up to 6 micro-ops per cycle
- 14 pipeline stages
- dynamic out-of-order with speculation
- register renaming, useful with speculation, as no need to store snapshot to undo speculation, just mark the speculated register results as invalid
- Instruction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

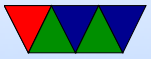


- Instruction issue. Reservation station. Up to six uops can be issued
- Finished instructions go back to reservation station and retirement unit, wait to update register state when determined it is no longer speculative.
- Once instruction hits the head of the reorder buffer, instruction commits and 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

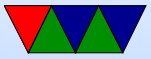


0.44). Worst, again, mcf IPC=0.37 (CPI=2.67)

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



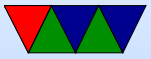
Power and Energy



Definitions and Units

People often say Power when they mean Energy

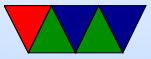
- Energy – Joules, kWH (3.6MJ), Therm (105.5MJ), 1 Ton TNT (4.2GJ), eV (1.6×10^{-19} J), BTU (1055 J), horsepower-hour (2.68 MJ), calorie (4.184 J)
- Power – Energy/Time – Watts (1 J/s), Horsepower (746W), Ton of Refrigeration (12,000 Btu/h)
- Volt-Amps (for A/C) – same units as Watts, but not same thing
- Charge – mAh (batteries) – need V to convert to Energy



Power and Energy in a Computer System

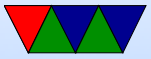
Power Consumption Breakdown on a Modern Laptop, A. Mahersi and V. Vardhan, PACS'04.

- Old, but hard to find thorough breakdowns like this
- Thinkpad Laptop, 1.3GHz Pentium M, 256M, 14" disp
- Oscilloscope, voltage probe and clamp-on current probe
- Measured V and Current. P=IIR. V=IR P=IV, subtractive for things w/o wires
- Total System Power 14-30W
- Old: no LED backlight, no SDD, etc.

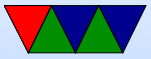


Modern results are from CUGR/REU student research.

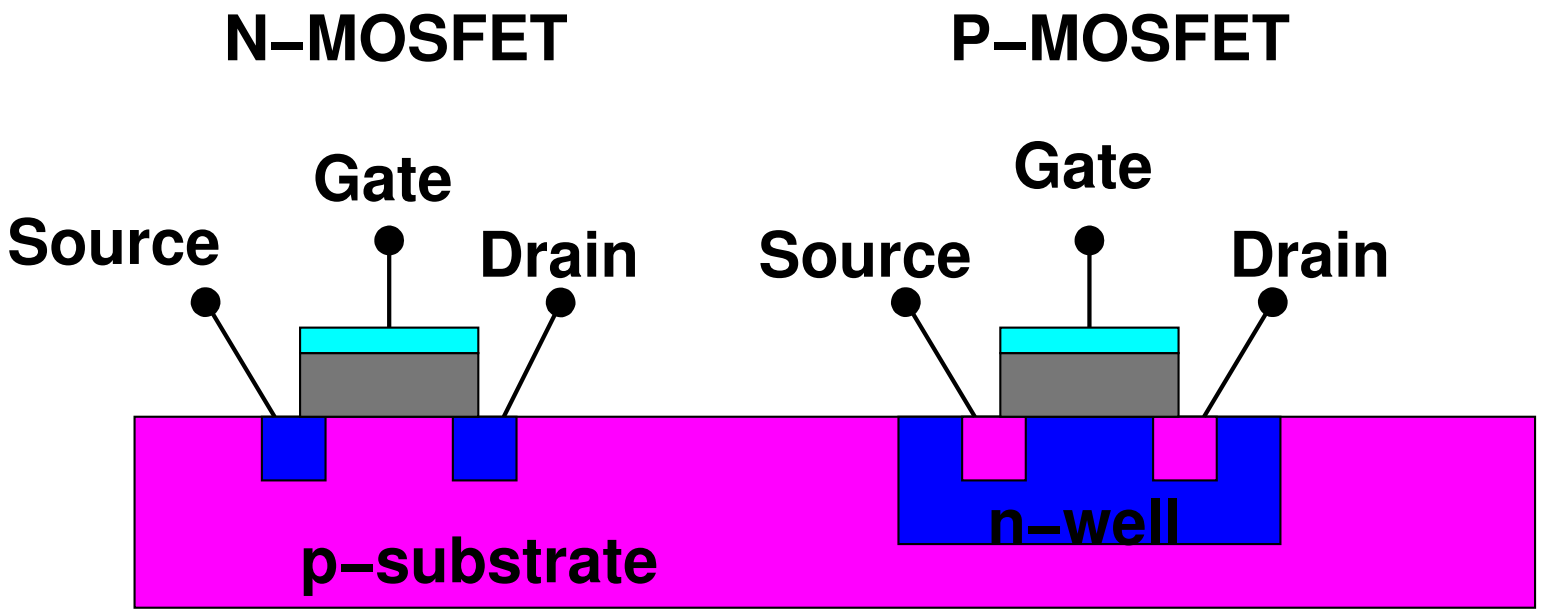
	Laptop (2004)	Modern	Server?
Hard Drive	0.5-2W	5W	
LCD	1W		
Backlight	1-4W		
CPU	2-15W	60+W	
GPU	1-5W	50+W	
Memory	0.5-1.5W	1-5W	
Power Supply	0.65W		
Wireless	0.1 - 3W		
CD-ROM	3-5W		
USB	(max 2.5W)		
USB keyboard		0.04W	
USB mouse		0.03W	
USB flash		0.5W	
USB wifi		0.5W	

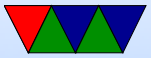


CPU Power and Energy



CMOS Transistors



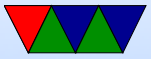


CMOS Dynamic Power

- $P = C\Delta V V_{dd} \alpha f$
  Charging and discharging capacitors big factor ($C\Delta V V_{dd}$) from V_{dd} to ground
  α is activity factor, transitions per clock cycle
  F is frequency

- α often approximated as $\frac{1}{2}$, $\Delta V V_{dd}$ as V_{dd}^2 leading to $P \approx \frac{1}{2} C V_{dd}^2 f$

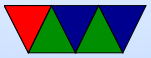
- Some pass-through loss (V momentarily shorted)



CMOS Dynamic Power Reduction

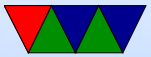
How can you reduce Dynamic Power?

- Reduce C – scaling
- Reduce V_{dd} – eventually hit transistor limit
- Reduce α (design level)
- Reduce f – makes processor slower



CMOS Static Power

- Leakage Current – bigger issue as scaling smaller. Forecast at one point to be 20-50% of all chip power before mitigations were taken.
- Various kinds of leakage (Substrate, Gate, etc)
- Linear with Voltage: $P_{static} = I_{leakage} V_{dd}$



Leakage Mitigation

- SOI – Silicon on Insulator (AMD, IBM but not Intel)
- High-k dielectric – instead of SO2 use some other material for gate oxide (Hafnium)
- Transistor sizing – make only the critical transistors fast; non-critical ones can be made slower and less leakage prone
- Body-biasing
- Sleep transistors