ECE 571 – Advanced Microprocessor-Based Design Lecture 13

Vince Weaver

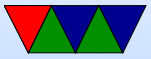
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

2 October 2024

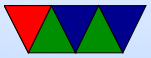
Announcements

- HW#2 grades posted
- Don't forget HW#4. A little trickier than previous as you run on 3 different machines



HW#3 Review – The Benchmarks

- sleep – does nothing
- stream – stresses memory subsystem
- matrix (ATLAS) – stresses the CPU
- iozone – disk I/O



HW#3 Review – The System

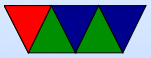
- Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz
  20MB cache, 22nm, 90W TDP

  `https://ark.intel.com/content/www/us/en/ark/products/83359/intel-xeon-processor-e5-2640-v3-20m-ca`

  `html`

- 80GB DDR4 RAM

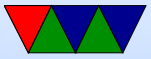
- Regular spinning hard drive



HW#3 Energy

Energy

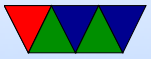
	sleep	stream	matrix	iozone
pkg	206	334	628	594
ram	27	96	54	90
time	10.0s	8.8s	3.5s	21s



HW#3 Power

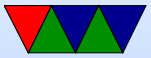
Power

	sleep	stream	matrix	iozone
pkg	20	42	180	26
ram	2.6	11	15	4.2



HW#3 Notes

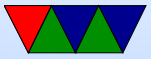
- highest power pkg: matrix
- highest power dram: matrix
- cores: always zero
  This is likely a bug with the Haswell-EP chips
  Intel slow to acknowledge this
- server class does not have integrated GPU
- guesses: old, virtual machine
- HPL (20k): pkg: 117W dram: 11W
- stream stresses memory. iozone, CPU is waiting?



HW#3 Energy-Delay

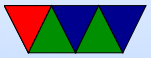
Energy-delay

	1	2	4	8	16	32	64
E	15k	9.3k	6.9k	5.0k	4.5k	4.2k	4.8k
time	296s	164s	88s	46s	35s	29s	39s
ED	4440k	1525k	607k	230k	156k	122k	187k
ED2	750M	250M	53M	11M	5.5M	3.5M	7.2M
Power	51W	57W	78W	109W	129W	145W	123W
Scaling	—	1.8x	3.3x	6.4x	8.5x	10.2x	7.6x

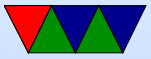


HW#3 Energy-Delay Discussion

- Interesting, this year 32 threads won everything. In past years this was not the case.
- e) scaling? only can show strong (problem size same)
  - Poorly written benchmark? (possible)
  - Not enough memory? (unlikely, benchmark from 2001)
- f) 32 threads, but only 16 cores
- g) TDP=90W for one package, but we have two

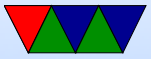


Oh No, More Caches!



Other Cache Types

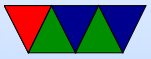
- Victim Cache – store last few evicted blocks in case brought back in again, mitigate smaller associativity
- Assist Cache – prefetch into small cache, avoid problem where prefetch kicks out good values
- Trace Cache – store predecoded program traces instead of (or in addition to) instruction cache



Virtual vs Physical Addressing

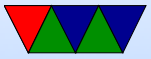
Programs operate on Virtual addresses.

- PIPT, PIVT (Physical Index, Physical/Virt Tagged) – easiest but requires TLB lookup to translate in critical path
- VIPT, VIVT (Virtual Index, Physical/Virt Tagged) – No need for TLB lookup, but can have aliasing between processes. Can use page coloring, OS support, or ASID (address space id) to keep things separate



Cache Miss Types

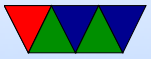
- Compulsory (Cold) — miss because first time seen
- Capacity — wouldn't have been a miss with larger cache
- Conflict — miss caused by conflict with another address (would not have been miss with fully assoc cache)
- Coherence — miss caused by other processor



Fixing Compulsory Misses

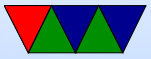
Prefetching

- Hardware Prefetchers – very good on modern machines. Automatically bring in nearby cachelines.
- Software – loading values before needed
  also special instructions available
- Large-blocksize of caches. A load brings in all nearby values in the rest of the block.



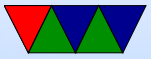
Fixing Capacity Misses

- Build Bigger Caches



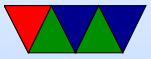
Fixing Conflict Misses

- More Ways in Cache
- Victim Cache
- Code/Variable Alignment, Cache Conscious Data Placement



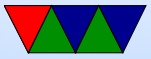
Fixing Coherence Misses

- False Sharing – independent values in a cache line being accessed by multiple cores



Capacity vs Conflict Miss

- It's hard to tell on the fly what kind of miss
- For example: to know if cold, need to keep list of every address that's ever been in cache
- To know if it's capacity, need to know if it would have missed even in a fully associative cache
- Otherwise, it's a conflict miss



Cache Parameters Example 1

32kB cache (2^{15}), direct mapped (2^0)
32 Byte linesize (2^5), 32-bit address size (2^{32})

offset = $log_2(linesize)$ = 5 bits
lines = $log_2((cachesize/\#ways)/linesize)$ = 1024 lines (10 bits)
tag = addresssize - (offset bits + line bits) = 17 bits