ECE 571 – Advanced Microprocessor-Based Design Lecture 12

Vince Weaver

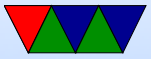
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

28 September 2020

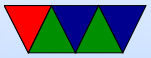
Announcements

- HW#4 was posted. Branch Prediction.



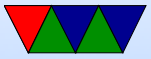
Some last branch predictor things

- Can turn off branch prediction on some machines. Most notably on the ARM1176 chip in a Raspberry Pi.



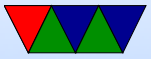
Branch Predictor Energy

- How much Energy does a branch predictor take?
- Often modeled as memory. Only in more complex setups does the logic take much space.
- How much die area (how many bits) (leakage)
- How often are they updated
- Do you need to store branch history on context switch?
- What happens if you turn off branch predictor? Will your code still run?



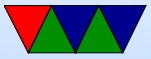
Branch Predictor Energy

- Parikh, Skadron, Zhang, Barcella, Stan
- 4 concerns:
  1. Accuracy. Not affect power, but performance
  2. Configuration (may affect power)
  3. Number of lookups
  4. Number of updates
- Tradeoff power vs time.
- brpred can be size of small cache, 10% of power
- Can use banking to mitigate



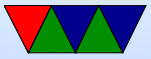
Branch Predictors

- can watch icache, not activate predictor if no branches
- Pipeline gating, keep track of each predicted branch confidence. If confidence hits certain threshold, stop speculating. Show this may or may not be good.
- Integer code, large predictors good
- FP, tight loops, predictors not as important.



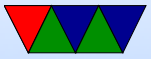
Branch Predictor Evaluation

- (Strasser, 1999). Simulation, small branch predictor can help energy.
- (Co, Weikle, Skadron) Formula for break even point. Leakage matters, what brpred hides is stall cycles.
- SEPAS: A Highly Accurate Energy-Efficient Branch Predictor (Baniasadi, Moshovos. ISLPED 2004).
  Once a branch prediction reaches steady state (unlikely to change) stop accessing/updating predictor, saving



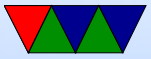
energy.

- Low Power/Area Branch Prediction Using Complementary Branch Predictors (Sendag, Yi, Chuang, Lija. IPDPS 2008)
  Complementary Branch Predictor to handle the tough cases.



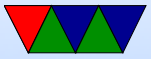
Branch Predictors and Code Type

- What type of code is easiest to predict?
  Regular Loops
  Often found in large scientific "floating-point" workloads
- What is hard to predict? User input, random data/parsing with lots of conditional branches.
  "integer benchmarks", things like compilers, parsers, compression



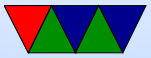
Branch Predictor History

- IBM Stretch, 1950s
- 2-bit counters first proposed late 1970s
- Two-level, 1991



Branch Predictor Implementations

- Agner Fogg document, interesting. Found a bug in 2-bit saturating counters on P1
- Haswell re-written from scratch, unknown. Too many branches close together can cause problems



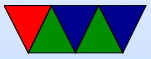
Caches

"Almost all programming can be viewed as an exercise in caching." – **Terje Mathisen**

First Data Cache: IBM System/360 Model 85, 1968

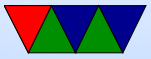
Good survey paper, Ajay Smith, 1982

Computer Architects don't like to admit it, but no amazing breakthroughs in years. Mostly incremental changes.



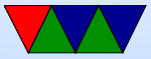
What is a cache?

- Small piece of fast memory that is close to the CPU.
- "caches" subsets of main memory
- Managed automatically by hardware (can you have a software controlled cache? Scratchpad memory? Why aren't they used more? Hard to do right.)



Memory Wall

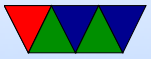
- Processors getting faster (and recently, more cores) and the memory subsystem cannot keep up.
- Modern processors spend a lot of time waiting for memory
- "Memory Wall" term coined by Wulf and McKee, 1995



Exploits Program Locality

- Temporal – if data is accessed, likely to be accessed again soon
- Spatial – if data is accessed, likely to access nearby data

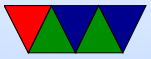
Not guaranteed, but true more often than not



Memory Hierarchy

There's never enough memory, so a hierarchy is created of increasingly slow storage.

- Older: CPU $\rightarrow$ Memory $\rightarrow$ Disk $\rightarrow$ Tape
- Old: CPU $\rightarrow$ L1 Cache $\rightarrow$ Memory $\rightarrow$ Disk
- Now?: CPU $\rightarrow$ L1/L2/L3 Cache $\rightarrow$ Memory $\rightarrow$ SSD Disk $\rightarrow$ Network/Cloud

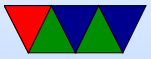


Cache Types

- Instruction (I$) – holds instructions, often read only
  (what about self-modifying code?)
  can hold extra info (branch prediction hints, instruction decode boundaries)

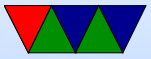
- Data (D$) – holds data

- Unified – holds both instruction and data
  More flexible than separate

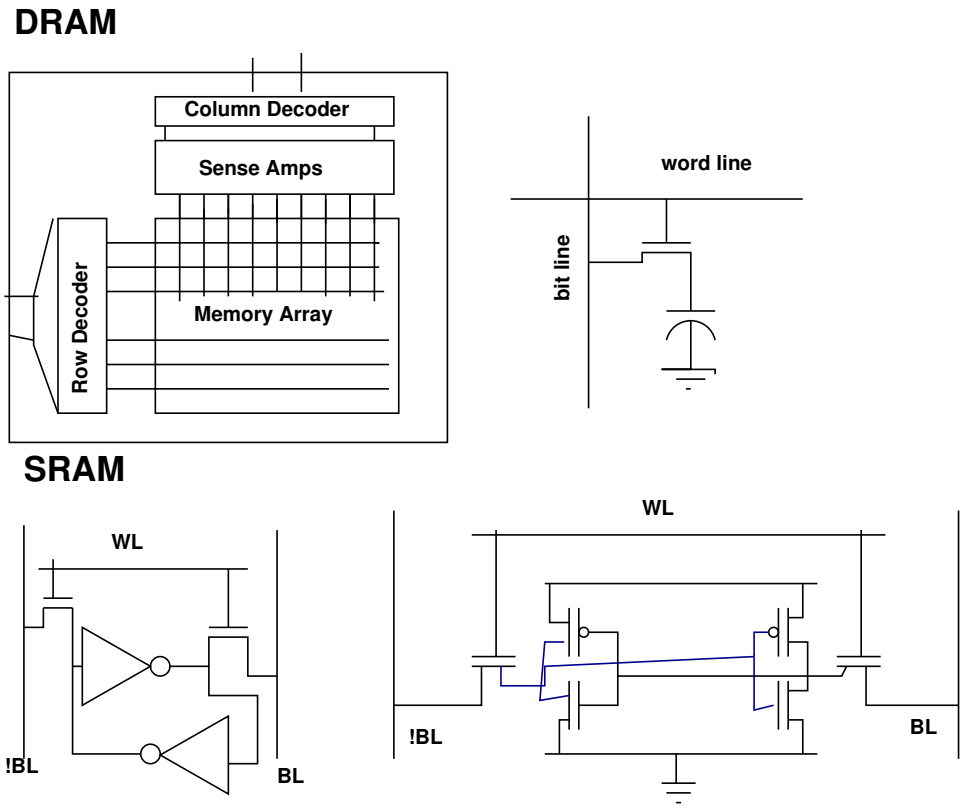


Cache Circuitry

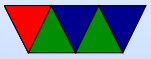
- SRAM – flip-flops, not as dense
- DRAM – fewer transistors, but huge capacitors
  chips fabbed in DRAM process slower than normal CPU logic



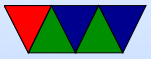
Cache Circuitry



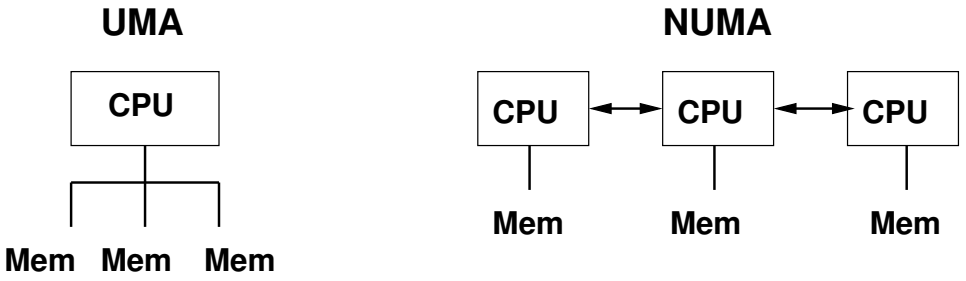
- Upside of DRAM? Smaller, can fit more.



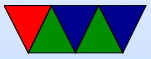
- Upside of SRAM? No need to refresh.
- Which is faster/lower energy? Used to be SRAM but not so clear anymore.
- Why not use DRAM in caches? Process tech doesn't line up well. Process for good capacitors makes for slower logic.
- Recent advances (trench capacitors, etc) have changed this a bit. IBM Power machines with large DRAM caches.



UMA and NUMA

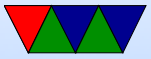


- UMA – Uniform Memory Access
  same speed to access all of memory
- NUMA – Non-Uniform Memory Access
  accesses to memory connected to other CPU can take longer



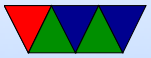
Cache Coherency

- Protocols such as MESI (Modified, Exclusive, Shared, Invalid)
- Snoopy vs Directory



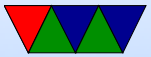
Cache Structure

	Way 0					Way 1				
line	Tag	Data				Tag	Data			
0	1	00	00	00	00	0				
1	1	00	10	00	00	1	00	00	00	00
2	0					0				
3	0					0				
4	0					0				
5	0					0				
. . .										
b	0					0				
c	0					0				
d	0					0				
e	0					0				
f	0					0				



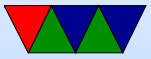
Cache Associativity

- direct-mapped – an address maps to only one cache line
- fully-associative (content-addressable memory, CAM) – an address can map to any cache line
- set-associative – an address can map to multiple "ways"
- scratchpad – software managed (seen in DSPs and some CPUs)



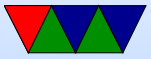
Cache Terms

- Line – which row of a cache being accessed
- Blocks – size of data chunk stored by a cache
- Tags – used to indicate high bits of address; used to detect cache hits
- Sets (or ways) – parts of an associative cache



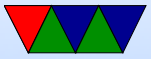
Replacement Policy

- FIFO
- LRU
- Round-robin
- Random
- Pseudo-LRU
- Spatial



Load Policy

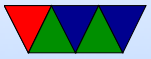
- Critical Word First – when loading a multiple-byte line, bring in the bytes of interest first



Consistency

Need to make sure Memory eventually matches what we have in cache.

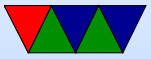
- write-back – keeps track of dirty blocks, only writes back at eviction time. poor interaction on multi-processor machines
- write-through – easiest for consistency, potentially more bandwidth needed, values written that are discarded
- write-allocate – Usually in conjunction with write-back Load cacheline from memory before writing.



Inclusiveness

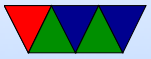
- Inclusive – every item in L1 also in L2
  simple, but wastes cache space (multiple copies)

- Exclusive – item cannot be in multiple levels at a time



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