

ECE 571 – Advanced Microprocessor-Based Design Lecture 8

Vince Weaver

`http://www.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

11 February 2016

Announcements

- HW3 was due
- HW#4 will be posted. About branch predictors



Some last branch predictor things

- Can turn off branch prediction on some machines. Most notably on the ARM1176 chip in a Raspberry Pi.
- People seemed to like the idea of branch predictors, interesting ideas. Would make a good project. Project posted after spring break.
- Never too early to think about projects. If you think it would be fun to play with branch predictor ideas I can set you up with a branch predictor simulator. If you



are interested in power effectiveness of branch predictors (you can turn off the branch predictor on raspberry pi). If you think I'm doing a bad job of designing embedded power measurements and think you can come up with something better.



Performance Concerns – 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



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



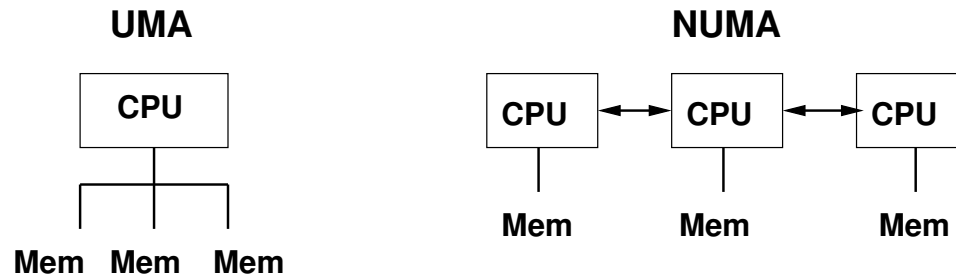
Memory Hierarchy

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

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



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 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 Coherency

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



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

