CS/EE 3810: Computer Organization

Lecture 16: Memory

Anton Burtsev November, 2022

Off-Chip DRAM Main Memory

- Main memory is stored in DRAM cells that have much higher storage density
- DRAM cells lose their state over time must be refreshed periodically, hence the name Dynamic
- A number of DRAM chips are aggregated on a DIMM to provide high capacity – a DIMM is a module that plugs into a bus on the motherboard
- DRAM access suffers from long access time and high energy overhead

Memory Architecture



- DIMM: a PCB with DRAM chips on the back and front
- The memory system is itself organized into ranks and banks; each bank can process a transaction in parallel
- Each bank has a row buffer that retains the last row touched in a bank (it's like a cache in the memory system that exploits spatial locality) (row buffer hits have a lower latency than a row buffer miss)

Intel Xeon Gold 6142 Processor

- 22M Cache, 2.60 GHz
- https://ark.intel.com/content/www/us/ en/ark/products/120487/intel-xeon-gol d-6142-processor-22m-cache-2-60-gh z.html
- 6 memory channels, 2666MT/s
- Theoretical memory throughput
 - 2666 MT/s * 8 (bytes per transaction) * 6 (num channels) = 127.9 GB/s (Theoretical peak memory bandwidth)

Intel Xeon Gold 6142 Processor

- CPUs, however, communicate with DRAM in the unit of cachelines (64 bytes).
 - Therefore, to access even one byte of data the CPU will issue a cache-line read
- A single memory channel is capable of performing 333.2 millions of cache-line reads or writes per second
 - A six channel system achieves 1.9 billions of cache-line transactions
- To keep the memory subsystem saturated, the socket should issue a cache-line transaction every 0.5 nanoseconds.
- On a 16 core socket
 - Each core submits a cache-line transaction every 8 ns
 - Or 20 cycles on a 2.6GHz machine
 - Or 40 cycles if logical threads are enabled.

Virtual Memory and Page Tables

Paging

Pages





Pages



Paging idea

- Break up memory into 4096-byte chunks called pages
 - Modern hardware supports 2MB, 4MB, and 1GB pages
- Independently control mapping for each page of linear address space

- Compare with segmentation (single base + limit)
 - many more degrees of freedom

How can we build this translation mechanism?

Paging: naive approach: translation array



• Virtual address 0x410010

What is wrong?

What is wrong?

- We need 4 bytes to relocate each page
 - 20 bits for physical page number
 - 12 bits of access flags
 - Therefore, we need array of 4 bytes x 1M entries
 - 4MBs

Paging: page table























Benefit of page tables

- ... Compared to arrays?
- Page tables represent sparse address space more efficiently
 - An entire array has to be allocated upfront
 - But if the address space uses a handful of pages
 - Only page tables (Level 1 and 2 need to be allocated to describe translation)
- On a dense address space this benefit goes away
 - I'll assign a homework!

What about isolation?



• Two programs, one memory?

- Each process has its own page table
 - OS switches between them



TLB

- Since the number of pages is very high, the page table capacity is too large to fit on chip
- A translation lookaside buffer (TLB) caches the virtual to physical page number translation for recent accesses
- A TLB miss requires us to access the page table, which may not even be found in the cache – two expensive memory look-ups to access one word of data!
- A large page size can increase the coverage of the TLB and reduce the capacity of the page table, but also increases memory waste

Page translation in 64bit mode



Memory Hierarchy Properties

- A virtual memory page can be placed anywhere in physical memory (fully-associative)
- Replacement is usually LRU (since the miss penalty is huge, we can invest some effort to minimize misses)
- A page table (indexed by virtual page number) is used for translating virtual to physical page number
- The memory-disk hierarchy can be either inclusive or exclusive and the write policy is writeback

TLB and Cache

- Is the cache indexed with virtual or physical address?
 - To index with a physical address, we will have to first look up the TLB, then the cache \rightarrow longer access time
 - Multiple virtual addresses can map to the same physical address – can we ensure that these different virtual addresses will map to the same location in cache? Else, there will be two different copies of the same physical memory word
- Does the tag array store virtual or physical addresses?
 Since multiple virtual addresses can map to the same physical address, a virtual tag comparison can flag a miss even if the correct physical memory word is present

TLB and Cache

TLB and Cache

- Is the cache indexed with virtual or physical address?
 - To index with a physical address, we will have to first look up the TLB, then the cache Solonger access time
 - Multiple virtual addresses can map to the same physical address – must ensure that these different virtual addresses will map to the same location in cache – else, there will be two different copies of the same physical memory word

Does the tag array store virtual or physical addresses?
 Since multiple virtual addresses can map to the same physical address, a virtual tag comparison can flag a miss even if the correct physical memory word is present

Cache and TLB Pipeline



Virtually Indexed; Physically Tagged Cache

Virtually Indexed Caches

- 24-bit virtual address, 4KB page size → 12 bits offset and 12 bits virtual page number
- To handle the example below, the cache must be designed to use only 12 index bits for example, make the 64KB cache 16-way
- Page coloring can ensure that some bits of virtual and physical address match



Thank you!