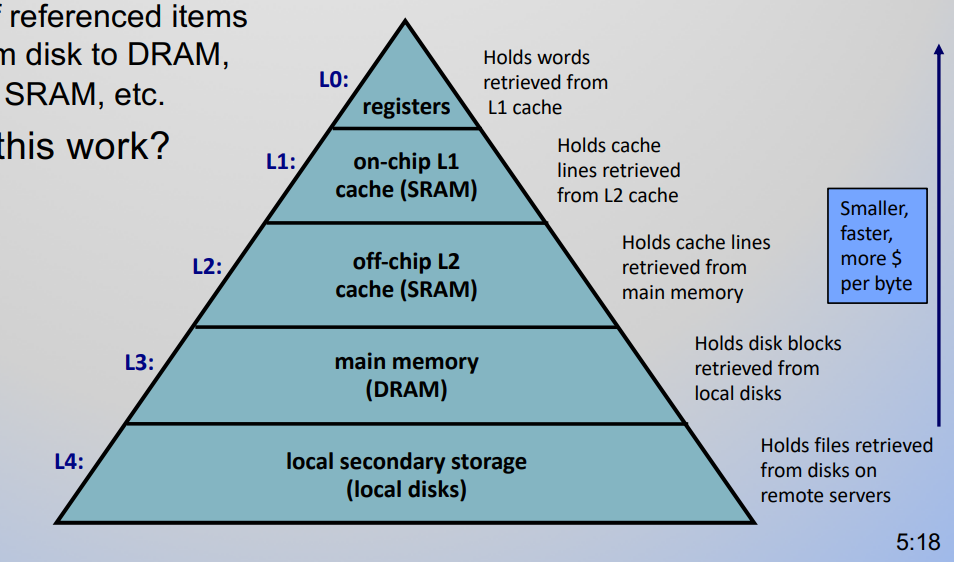
`

Eric, Alex. Noah. Bryson. Daniel. **Kevin,** Jacob

Chapter 5.

* What is the principle of locality?
  + Programs access a small subset of address space at any time. This is necessary for mem hierarchy to work
  + What is temporal locality?
    - Accessing the same data over and over.
    - The data being accessed are close together in time
  + What is spatial locality?
    - Accessing nearby data over and over again.
    - The data being accessed are close together in physical memory
* What is a *memory hierarchy*?
  + Put most expensive fastest memory at the top (near CPU). The slower, cheaper, and bigger the memory the lower it is in the hierarchy.
  + It is the memory construct designed to give the illusion of having a lot of fast memory. It consists of various levels of memory.
  + If the processor is at the top, where in the hierarchy is the fastest memory?
    - The register file is the fastest and at the top of the memory
  + Where is the cheapest memory (lowest cost per bit) in the hierarchy?
    - At the bottom
  + Where is the largest memory in the hierarchy?
    - Also bottom
  + What is the goal of a memory hierarchy? Are you on campus?
  + To give the illusion of all memory being as fast as our fastest memory.
    - The illusion of both fast and large memory.
      * Fast memory is small
      * Large memory is slow
  + Why do designers use memory hierarchies?
    - See previous. It is also very efficient.
* In the context of a memory hierarchy, what do the following terms mean?
  + Block, hit rate, miss rate, hit time, miss penalty
    - Block (or line) - the minimum unit of information that can be either present or not present in a cache. (390)
    - Hit Ratio - hits/accesses
    - Hit - the desired block is present
    - Hit Rate - the fraction of memory accesses found in a level of the memory hierarchy. (390)
    - Miss - the processor wants a block of memory but it's not in the cache
    - Miss Ratio = (1 - hit Ratio)
    - Data transfer - entire block or line is copied. Typically multiple words at a time.
    - Miss penalty ‐ The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor. (390)
* What must programmers know about memory to get good performance?
  + They too often think of memory as one big block. They need to know how hierarchies work in order to maximize performance.
* How do memory hierarchies exploit temporal locality?
  + Memory hierarchies take advantage of temporal locality by keeping more recently accessed data items closer to the processor (391)
* How do they exploit spatial locality?
  + Memory hierarchies take advantage of spatial locality by moving blocks consisting of multiple contiguous words in memory to upper levels of the hierarchy. (391)
  + This works for lots of things that exhibit natural spacial locality such as instruction memory, program code that executes sequentially, data sets, other arrays, etc.
* In general, if data are in a level *i* of a hierarchy, will they also be found in level *i+1*?
  + Yes, they have to be if it is a true hierarchy. (391)
* Is most of the cost of a memory hierarchy at the highest level (closest to the processor)?
  + No. The highest level is the most expensive **per bit** but not usually the most expensive overall. (Page e-2, answer to check yourself on page 391)
* What are the four primary technologies used today in memory hierarchies?
  + In order from highest cost per bit to lowest:
    - SRAM semiconductor memory - 0.5–2.5ns - $500–$1000
    - DRAM semiconductor memory - 50–70ns - $3–$6
    - Flash semiconductor memory - 5,000–50,000ns - $0.06–$0.12
    - Magnetic disk - 5,000,000–20,000,000ns - $0.01–$0.02
  + What are the important characteristics of each?
    - SRAM - (static random access memory) (393)
      * Doesn’t need to refresh and so the access time is very close to the cycle time is the time between memory accesses.
      * Typically use six to eight transistors per bit to prevent the information from being disturbed when read.
      * Needs only minimal power to retain the charge in standby mode
    - DRAM - (dynamic random access memory) (393)
      * The value kept in a cell is stored as a charge in a capacitor. A single transistor is then used to access this stored charge, either to read the value or to overwrite the charge stored there.
      * It cannot be kept indefinitely and must periodically be refreshed. Impersistence is why this memory structure is called dynamic, in contrast to the static storage in an SRAM cell.
      * DRAMs use a two-level decoding structure, and this allows us to refresh an entire row (which shares a word line) with a read cycle followed immediately by a write cycle.
      * To improve performance, DRAMs buffer rows for repeated access.
      * To improve the interface to processors further, DRAMs added clocks and are properly called synchronous DRAMs or SDRAMs. The advantage of SDRAMs is that the use of a clock eliminates the time for the memory and processor to synchronize.
    - Flash (395)
      * Writes can wear out flash memory bits. To cope with such limits, most flash products include a controller to spread the writes by remapping blocks that have been written many times to less trodden blocks. This technique is called wear leveling.
      * Such wear-leveling lowers the potential performance of flash, but it is needed unless higher-level software monitors block wear. Flash controllers that perform wear leveling can also improve yield by mapping out memory cells that were manufactured incorrectly.
    - Magnetic disk (395)
      * Physical discs that are read/written to by a mechanical moving magnetic arm.. called the head. The time it takes the arm to move to different tracks (other cylinders of the disc) is known as seek time. When the head is in the right spot it must wait for the disc to rotate to find the specific data you want. This is the rotational delay or rotational latency.
      * In summary, the two primary differences between magnetic disks and semiconductor memory technologies are that disks have a slower access time because they are mechanical devices—flash latency is 1000 times as fast and DRAM is 100,000 times as fast—yet they are cheaper per bit because they have very high storage capacity at a modest cost—disks are 6 to 300 times cheaper.
  + Which are volatile?
    - Volatile is lost when it loses power. Non-volatile doesn't lose data when power is lost.
    - Flash and disc memory are both non-volatile while SRAM and DRAM are volatile.
  + Which technology is used to implement caches?
    - Caches use SRAM as it’s faster than DRAM. (392)
  + Which technology is used to implement main memory?
    - DRAM as they have larger capacity for the same amount of silicon (392)
  + Which technology is used as secondary memory in Personal Mobile Devices?
    - Flash as it is very stable and rugged. It can handle being jostled constantly. (392)
  + Which technology is commonly used as secondary memory in servers?
    - Magnetic disk. (392)
* Why does DRAM require refresh?
  + It loses charge over time. It would lose its data if not refreshed constantly. This is because it uses capacitors to store charges. A row is refreshed in two cycles by reading it and writing it back to itself.
  + What is buffered internally in DRAM?
    - DRAM will buffer rows for repeated access. By changing the address, random bits can be accessed while waiting for the next row access.
  + What is the advantage of having multiple banks within a DRAM chip?
    - Sending an address to several banks permits them all to read or write simultaneously. For example, with four banks, there is just one access time and then accesses rotate between the four banks to supply four times the bandwidth. This rotating access scheme is called address interleaving. (395)
* In the context of flash memory, what is *wear leveling*?
  + Flash memory wears out over time. You can write to a specific address only ~100k times.
  + To cope with such limits, most flash products include a controller to spread the writes by remapping blocks that have been written many times to less trodden blocks. This technique is called wear leveling.
* How is a magnetic disk organized?
  + A magnetic hard disk consists of a collection of platters, which rotate on a spindle at 5400 to 15,000 revolutions per minute.
  + The metal platters are covered with magnetic recording material on both sides, similar to the material found on a cassette or videotape
  + Each disk surface is divided into concentric circles, called tracks. There are typically tens of thousands of tracks per surface. Each track is in turn divided into sectors that contain the information; each track may have thousands of sectors. Sectors are typically 512 to 4096 bytes in size.
  + What sequence of steps is required to access a specific disk sector?
    - Three stage process: (397)
      * Seek (move head to correct spot)
      * Rotational Latency (wait for disc to turn to the right spot)
      * Transfer Time (time to transfer a block of bits)
  + Which steps in a disk access take the most time? Why?
    - Seek is usually advertised as 3ms to 13ms but can be much shorter.
    - Rotational Latency for 5400 RPM disc = ~5.6ms
  + What are the tradeoffs between flash memory and magnetic disc?
    - Magnetic disks are nonvolatile like flash, but unlike flash, there is no write wear-out problem. However, flash is much more rugged and hence a better match to the jostling inherent in personal mobile devices
    - Disc is still slower but it doesn’t wear out over time. Flash wears out but is better when moved.
* What term is used to refer to any storage that is managed in such a way as to take advantage of locality of access?
  + Cache was the name chosen to represent the level of the memory hierarchy between the processor and main memory in the first commercial computer to have this extra level. The memories in the datapath in Chapter 4 are simply replaced by caches. Today, although this remains the dominant use of the word **cache**, **the term is also used to refer to any storage managed to take advantage of locality of access. (398)**
* What computer systems today include memory hierarchies with caches?
  + every general-purpose computer built now from servers to low-power embedded processors includes caches.
* What is a direct-mapped cache?
  + There are a fixed number of sets with one block per set. Usually a power of 2 for the number of sets.
  + A cache structure in which each memory location is mapped to exactly one location in the cache. (398)
  + The cache is obviously smaller than the memory so this does not go both ways. Each memory location is mapped to exactly one location in the cache but each cache location will have many memory locations mapped to it. This is why in HW 10 we see accesses overwriting things we have in the cache even though the cache is far from full.
  + What determines the location(s) in a cache where a block can be placed?
    - The index bits in the address to give the set. The tag identifies which block to be accessed, LRU or random is used to replace an old block with a new one if there is a miss (replacing blocks with LRU only happens with multiple associativoty, else, there is only one block to replace)
    - Almost all direct-mapped caches use this mapping to find a block: (Block address modulo (Number of blocks in the cache)
    - Thus, an 8-block cache uses the three lowest bits (8 = 2^3 ) of the block address.
  + What is added to the cache to determine which block is in each cache location?
    - Adding a tag since two or more addresses can map to the same location
    - The tags contain the address information required to identify whether a word in the cache corresponds to the requested word. The tag needs just to contain the upper portion of the address, corresponding to the bits that are not used as an index into the cache. (399)
  + Why is it essential to add a valid bit to every cache entry?
    - We also need a way to recognize that a cache block does not have valid information. For instance, when a processor starts up, the cache does not have good data, and the tag fields will be meaningless. Thus, we need to know that the tag should be ignored for such entries. (399)
* How is a memory address interpreted by a cache?
  + It looks at the bits in each of the three address fields.
  + What fields are the address divided into?
    - Tag, index, offset
  + What role does each field play in a cache access?
    - Index = Enough bits to represent the total number of sets in the cache.
    - Offset = The offset to select the data in the word that is desired.
    - Tag = Whatever bits are leftover in the address.
    - TODO: Check tag and offset definitions here
* What do the addresses of all bytes within a block have in common?
  + The index
* Given a memory address and a block size, can you determine the block number that that location will be a part of?
  + If you know how large the address is you know how many total bits you have.
  + Knowing the block size will tell you how many bytes it has and therefore how many bits you need for the byte offset-enough to represent all the bytes using the least significant bits.
  + I don’t see how you can actually figure out the index without being given either the cache size or the number of sets.
  + So, no?
    - You can get offset but I don’t see how you can tell where index ends and tag begins without more information - unless they mean you’ll know that from the address.
* In general, what is the impact of reducing the block size? Of increasing the block size?
  + Decreasing block size usually increases miss rate and vice versa
  + Increasing block size generally reduces miss rate. However, when the block size gets so large that it becomes a significant portion of the cache size the miss rate can go up. This is because you’ll only be able to fit a few blocks in the cache at a time and they’ll need to be replaced long before all their words are accessed, increasing the miss rate. (405)
* How can the optimizations of *early restart* and *critical word first* reduce the effects of a large miss penalty?
  + First, large miss penalties come from having larger blocks. The bigger the block, the longer it takes to transfer the block from a lower level into the cache when a miss occurs.
  + To help mitigate this, you can use an early restart method. This is where the processor resumes execution as soon as it returns the word it was looking for from memory, instead of waiting for the whole block to transfer to the cache.
    - This works best with instruction memory which is sequential. If you can pull in one new word from a block each clock cycle, you can continue to operate at normal speed with this method as long as that next word is the next instruction you need.
  + Critical word first is a more sophisticated method where the requested word is transferred first, then the rest of the block is transferred, starting at the next word and wrapping back around to the beginning of the block. It can be slightly faster but is limited by the same things as early restart. (407)
* What happens in a pipelined processor when a cache miss occurs?
  + The control unit is needed to handle a cache miss.
    - It works with a seperate controller that handles the memory access and cache filling.
  + The pipeline will simply stall until it gets the next value from memory. The specific steps are:
    - Send the original PC value to the memory.
    - Instruct main memory to perform a read and wait for the memory to complete its access.
    - Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on if it was not on already
    - Restart the instruction execution at the first step, which will re-fetch the instruction, this time finding it in the cache
* Why are writes more complicated than reads in a memory hierarchy?
  + With reads you only have to go to memory when there is a cache miss. With writes, you usually can’t only write to the cache because your change won’t be recorded in the main memory.
  + The simplest fix is the write-through scheme.
    - This method has every write also write to memory.
  + Write-buffer
    - This method uses a buffer to store what needs written to memory. When a write occurs the processor only needs to write to the cache and the buffer and can then move on while the buffer slowly writes to memory in the background. If the buffer fills up, it has to stall until there is room when a write is encountered.
  + Write-back
    - This method only writes to the cache. When that spot in the cache needs to be overwritten, only then is that data written to memory.
  + What is the advantage of a *write-through* scheme?
    - Updates cache and memory but very expensive. Requires memory for every instruction.
  + What role could a write-buffer play with a write-through cache?
    - Stores data in the write buffer, complete writes while the processor continues executing other orders
  + How are writes handled in a *write-back* cache?
    - Sets a dirty bit. The memory is updated when the block is replaced in cache.
  + What options exist for handling write-misses?
    - Using a split cache, multi-level cache,
* For the cache depicted in Fig. 5.12, what is the block size?
  + The data is listed as 512 bits which is 64 bytes or 16 words. The block offset is used to select the word in the block.
  + Is this cache direct-mapped?
    - The cache is directly map
    - This is clear because the index is 8 bits wide and there are 256 entries/blocks so each index maps to exactly one block.
  + Assuming there is a separate *set* in the cache for each possible value of the index field, how many sets are in this cache?
    - There are 256 sets in the cache (see last slide in lecture 25 for more info)
  + Why does this design use separate instruction and data caches?
    - The overall miss rate can be decreased.
    - This processor has a 12-stage pipeline. When operating at peak speed, the processor can request both an instruction word and a data word on every clock. To satisfy the demands of the pipeline without stalling, separate instruction and data caches are used. (410)
  + Can you explain why instruction miss rates are generally lower than data miss rates?
    - **There is more spatial locality.**
    - ^^^ Instructions are typically executed sequentially so the chances that the next instruction you need in already in the cache are pretty high. Only when you hit the end of a block or you jump somewhere else would you potentially have a miss.
* How does the memory hierarchy affect CPU time?
  + CPU time = (CPU execution clock cycles + Memory-stall clock cycles) x Clock cycle time
  + Memory-stall clock cycles = (Read-stall cycles Write-stall cycles)
  + Read-stall cycles Reads Program = (Reads/Program) x (Read miss rate) x (Read miss penalty)
  + What is the impact of hit time on processor performance?
    - Clearly, if the hit time increases, the total time to access a word from the memory system will increase, possibly causing an increase in the processor cycle time (pg 780) (pg 415 in my textbook–second edition)
  + What changes in a processor pipeline if the hit time is increased by one cycle?
    - An increase in hit time likely adds another stage to the pipeline, since it may take multiple cycles for a cache hit.
  + What happens in a pipelined processor when there is a cache miss?
    - **There is a memory stall. These are broken up into read stalls and write stalls.**
  + Assuming reads and writes are combined, what equations tell us the number of memory stall cycles?
    - **Memory stalls = (Reads/program\*read miss rate\*read miss penalty) + (Writes/program\*write miss rate\*write miss penalty + write buffer stalls)**
    - **We can assume that write buffer stalls are insignificant. So, we get:**
    - **Memory accesses/program\*miss rate\*miss penalty.**
  + What is the motivation for using *average memory access time* (AMAT) to compare different cache designs?
    - To capture the fact that the time to access data for both hits and misses affects performance
  + How is AMAT calculated?
    - Hit time + miss rate \* miss penalty (assume hits take one clock cycle)
    - AMAT = Average memory access time.
* What are *set-associative* and *fully associative* caches?
  + How do they differ from direct-mapped caches?
    - **Direct-mapped caches only have one block per set.**
  + In a 4-way set-associative cache, how many tags are compared on each access? Why?
    - **4 tags, because there are 4 blocks (and therefore 4 tags) in every set.**
  + In a fully associative cache, how many tags are compared on each access?
    - **In a fully associative cache, there is only one set and all the blocks are in that set. So, the tags of all the blocks are compared on each access.**
  + In general, what is the advantage of increasing the degree of associativity?
    - The miss rate goes down
  + What is the main disadvantage of increasing the associativity?
    - **They are slower. We have to compare all the tags to each other.**
  + What are the added costs of an associative cache?
  + How much of a reduction in miss rate is achieved by increasing associativity?
    - Going from one-way to two-way associativity decreases the miss rate by about 15%, but there is little further improvement in going to higher associativity.
* In what kind of caches is there a choice of which block to replace?
  + What is required to implement LRU replacement in a 2-way set-associative cache?
    - Have a single bit to tell if it was the one recently used or not
  + What is required to implement LRU replacement in a 4-way set-associative cache?
    - Block replacement algorithm to keep the ordering of accesses
  + Why does it get harder to implement LRU as the associativity grows?
    - Keeping order, diminishing returns, requires more tag bits and more comparators
* What is the principal benefit of having more than one level of cache in a memory hierarchy?
  + **Multi-level caches are used to reduce the miss penalty. L1 is level one, then L2, then L3, etc.**
* If we compare the performance of two programs, will the one with fewer instructions always be the fastest?
  + **Not necessarily. Some can take advantage of spacial locality better and so can be quicker despite having more instructions. How well a program utilizes a cache makes a BIG difference in program speed.**
  + What can be done in software to "use the memory hierarchy well"?
* What is the main idea behind a *blocked* algorithm?
  + **This might be the example from class where we talked about “cache blocking” and computing submatrices. Basically the idea was to break up a big algorithm into smaller steps that can take more advantage of temporal and spatial locality.**
* What is an *error detection code*?
  + How does Hamming's *error correction code* identify which bit is in error?
    - Uses multiple parity bits
  + How many parity bits are required for each 64-bit block of data in a SEC/DED code?
    - SEC - 8 parity bits
  + How do you compute the parity bits for a 8-bit data word?
  + How do you decode and correct errors with a SEC/SED hamming code?
* What are *system virtual machines*?
  + What advantages do they offer?
    - The increasing importance of isolation and security in modern systems
    - The failures in security and reliability of standard operating systems
    - The sharing of a single computer among many unrelated users, in particular for Cloud computing
    - The dramatic increases in raw speed of processors over the decades, which makes the overhead of VMs more acceptable The broadest definition of VMs includes basically all emulation
  + What are the technical challenges in implementing them?
* What level in the memory hierarchy is used as a cache by *virtual memory*?
  + What are the benefits of virtual memory?
    - Uses main memory efficiently: part of program being used is in memory, while unused part can be on disk
    - Simplifies memory management: each process gets an identical virtual address space
    - Isolates address spaces: memory access bugs cannot affect your code or data; user code cannot directly access memory
  + What address translation must occur in systems with virtual memory?
    - We need a translation map to map virtual page numbers to physical page numbers. The offset does not change.
  + What is a *page fault*?
    - A miss in main memory
  + How is a specific page found in main memory?
    - Page table: The table containing the virtual to physical address translations in a virtual memory system. The table, which is stored in memory, is typically indexed by the virtual page number; each entry in the table contains the physical page
  + What information is included in a *page table*?
    - Large array with one entry per page
    - Page number is index: entry gives location of page
  + Why are tag comparisons required to locate blocks in a cache, but not pages in main memory?
    - Because virtual pages will always stay the same throughout the course of the program execution
  + What is the advantage of *multi-level* page tables?
    - Allows for much less space for the page table entries
* What is a *translation-lookaside buffer* (or TLB)?
  + What is included in each TLB entry?
    - Valid, dirty, and reference bits, tag bits, Physical page address, (Virtual page number, page table, physical memory, disk storage.)
  + Is a TLB miss the same as a page fault?
    - **From what I understand, the TLB is like a small cache for the Page table. So, we could have a TLB miss but still have a valid page table entry and not have a page fault.**
  + What happens in response to a TLB miss?
    - We read from a page table entry
  + Why are protection bits (that limit certain kinds of accesses) usually included in page table entries and TLBs?
    - **Because, for example, some things we want to limit access to. We might want to lock down some memory so it’s read-only. Some memory might only be accessible in kernel mode.**
* What are the 4 key questions in our text's common framework for memory hierarchy?
  + 1: Where Can a Block Be Placed?
  + 2: How Is a Block Found?
    - Index and search
  + 3: Which Block Should Be Replaced on a Cache Miss?
    - Random: Good for two to four degrees of associativity
    - Least Recently Used (LRU): better for higher associativities
  + 4: What Happens on a Write?
    - Write-through: The information is written to both the block in the cache and the block in the lower level of the memory hierarchy
    - Write-back: The information is written just to the block in the cache.
    - Individual words can be written by the processor at the rate that the cache, rather than the memory, can accept them
    - Multiple writes within a block require only one write to the lower level in the hierarchy
    - Misses are simpler and cheaper because they never require a block to be written back to the lower level.
* What are the three types of misses in the three Cs model?
  + **Compulsory misses: block was never in cache.** Occurs when the same memory location is not frequently accessed twice (Not many loops or iteration)
  + **Capacity misses: block used to be in cache but was replaced because cache is too small. To reduce, increase cache size.** Filling up the cache quickly. Large programs with large amount of memory.
  + **Conflict misses: block used to be in cache but was replaced because the set is too small. To reduce, increase set-associativity.** Many programs are running, accessing the same data.
* What is the *cache coherence problem* in systems with multiple processors?
  + **Because multiple processors can access the same spot in physical memory, we might end up with a case where the physical memory doesn’t match the cache for some processor. (i.e. core 1 reads address 22, then core 2 writes to address 22, and in the cache for core 1 we still have the old value for address 22).**
* What are the general characteristics of the Intel Core i7 and ARM A53 memory hierarchy?
  + Intel Core i7
    - High-end performance processor
    - Designed and manufactured by intel for computers
    - Four CPU cores
    - Power-hungry
  + i7: L1 cache
    - Organization: split instruction and data caches
    - Size: 32 KiB each for instructions/data per core
    - Associativity: Four-way (I), eight-way (D) set associativity
    - Replacement: Approximated LRU
    - Block size: 64 bytes
    - Write Policy: Write-back, no-write-allocate
    - Hit time (load use): Four clock cycles, pipelined.
  + i7: L3 cache: See slides for table
  + ARM A53
    - Popular embedded processor
    - Low power, mobile applications
    - Licensed by many companies like Samsung and Qualcomm
    - Used in smartphones, Roku, Nintendo switch, amazon fire, etc.
  + Table to compare the two near the end of the slides

Order of where things are: CPU > TLB > L1 > L2 > Main Mem > Disk

“I can guarantee you that I’ll have a question on the exam about cache coherence” from lecture Mar 23

No questions on matrices specifically on the code. There will be questions on locality and blocking.

Remember three misses.

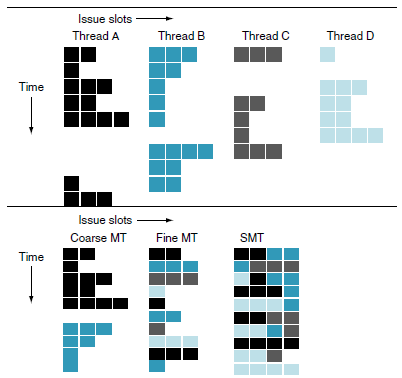
Probably won't need a calculator.

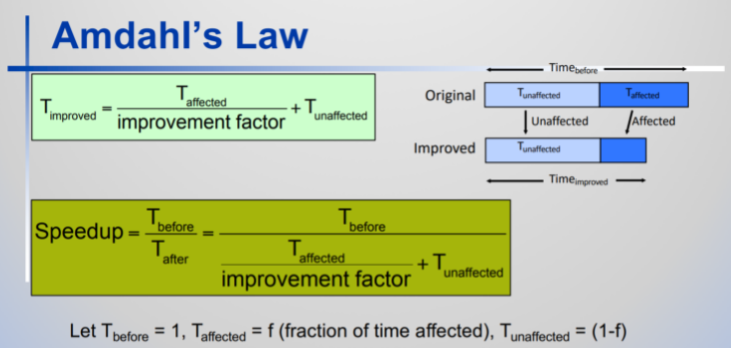
Quiz questions are good to practice.

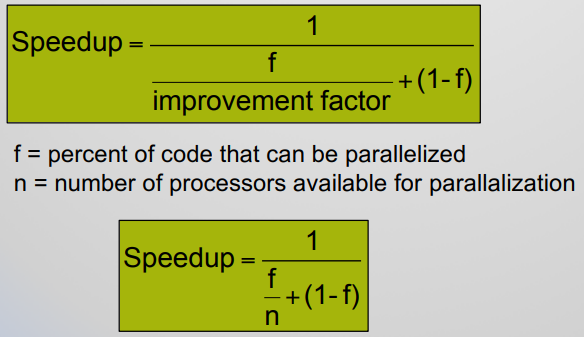
Should hopefully have a higher exam average.

CHAPTER SIX

* What is the meaning of the following classical parallel processing terms? SISD, SIMD, MISD, and MIMD
  + SISD: single instruction, single data
  + SIMD: single instruction, multiple data
  + MISD: Multiple instruction, single data
  + MIMD: Multiple instruction, multiple data
* Understand the meaning of and difference between the following terms related to parallel computing:
  + Computer cluster (6.8)
    - A bunch of separate computers on the same network working on the same problem.
  + Multicore microprocessor (6.5)
    - A multi-core processor is a computer processor on a single integrated circuit with two or more separate processing units, called cores, each of which can read and execute program instructions at the same time.
  + Hardware multithreading (6.4)

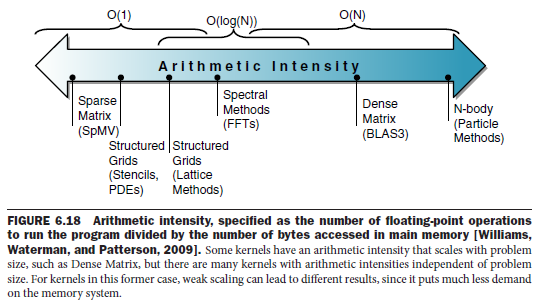


* + - Fine-grained multithreading:
      * Switches between threads on each instruction. This allows it to perform an interleaved execution of multiple threads. This way it can skip any threads that are stalled at that clock cycle.
      * The disadvantage is that it slows down the execution of individual threads since a thread that is ready to execute without stalls will be delayed by instructions from other threads.
    - Coarse-grained multithreading:
      * Created as an alternative to fine-grained. It only switches threads when encountering expensive stalls such as last-level cache misses. This means it doesn’t need to be able to switch threads as fast as fine-grained. It’s less likely to slow down individual threads since instructions from other threads will only be issued when the first thread hits a long stall.
      * It is very limited in its ability to overcome throughput losses from shorter stalls. This is because of the pipeline startup costs of coarse-grained. That cost is because when a thread stalls, the pipeline must be flushed or frozen, then the new thread must fill the pipeline before instructions begin to finish.
    - Simultaneous Multithreading (SMT):
      * A version of multithreading that lowers the cost of multithreading by utilizing the resources needed for multiple issue, dynamically scheduled microarchitecture.
      * It takes advantage of Parallelism AND Pipelining. It will process multiple threads at the same time.
  + Vector architecture (528)
    - It is used when you want to perform a single operation on every value in a vector. You use multiple processors to perform these operations at the same time (in parallel).
    - Pages 530 & 531 discuss the benefits and drawbacks of using vector architecture.
* How does Amdahl's law apply to parallel processing?
  + The amount of speedup you can get with a certain number of additional processors is limited by the amount of code that is parallelizable. Some code just has to be run sequentially and can’t be sped up with parallelism.
  + 

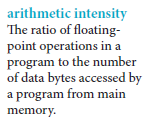


* What is the difference between weak scaling and strong scaling? (525)
  + Scaling is the concept that it is hard to get higher speedup with fixed programs. The bigger the problem the more it is sped up by adding additional processors.
  + Strong scaling: measure speed up w/ problem size fixed
  + Weak scaling: measure speedup w/ problem size proportional to the number of processors.
* What are the differences between GPU and traditional CPUs?
  + CPU = Central Processing Unit
    - Mostly just does calculations and loads to and stores from memory
  + GPU = Graphics Processing Unit
    - Mostly handles graphics. Built to print graphics rather than just calculations.
    - These are really difficult to program. One may have to code differently for each type of GPU.
    - GPUs are accelerators that supplement a CPU, so they do not need to be able to perform all the tasks of a CPU. This role allows them to dedicate all their resources to graphics. It’s fine for GPUs to perform some tasks poorly or not at all, given that in a system with both a CPU and a GPU, the CPU can do them if needed.
    - The GPU problem sizes are typically hundreds of megabytes to gigabytes, but not hundreds of gigabytes to terabytes. These differences led to different styles of architecture:
    - Perhaps the biggest difference is that GPUs do not rely on multilevel caches to overcome the long latency to memory, as do CPUs. Instead, GPUs rely on hardware multithreading (Section 6.4) to hide the latency to memory. That is, between the time of a memory request and the time that data arrive, the GPU executes hundreds or thousands of threads that are independent of that request

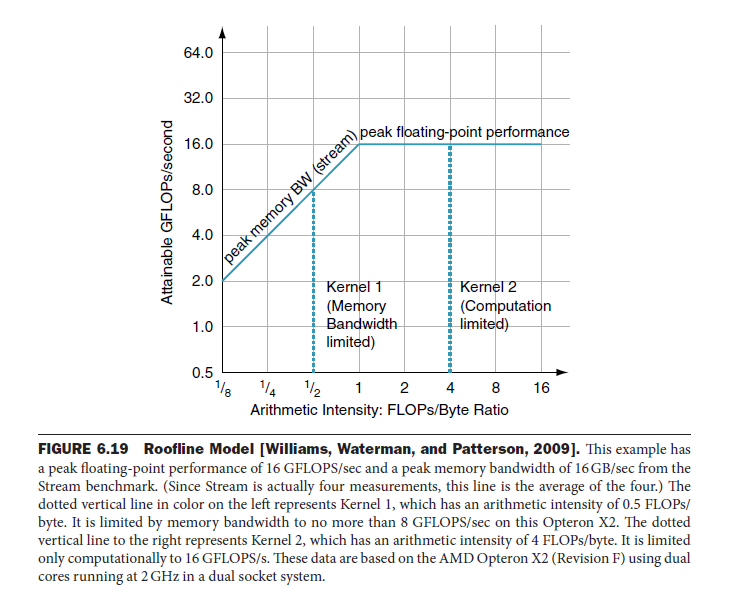
What is arithmetic intensity? (564)







What is a Roofline Model for parallel performance? (565)



Bandwidth Limited: When we **can** increase the attainable GFLOPs/Second by increasing Arithmetic Intensity (FLOPs/Byte Ratio). This is when we fall within the **sloped portion** of the graph above.

Computation Limited: This is when we **cannot** increase the attainable GFLOPs/Second by increasing Arithmetic Intensity (FLOPs/Byte Ratio). This is when we fall within the flat portion of the graph above. The only way we could increase GFLOPs/Second here is by changing the processor, or generation of processor, that we’re using (See below)

