Performance of Programs

- “Complexity” of algorithms
- How good/efficient is your algorithm
  - Measure using Big-Oh notation: $O(N \log N)$
- Next question: How well is the code executing on the machine??
  - Actual time to run the program
    - What are the factors that come into play
    - Where is the program and data stored
    - What are the actual machine instructions executed
  - Why is some HW better than others for different programs?
  - What factors of system performance are HW related
  - How does machine instruction set affect performance
  - What are the technology trends and how do they play a role?

Processor time: how to measure?

- Number of clock cycles it takes to complete the execution of your program
- What is your program
  - A number of instructions
    - Different types: load, store, ALU, branch
  - Stored in memory
  - Executed on the CPU
Aspects of CPU Performance

CPU time = Seconds = Instructions x Cycles x Seconds
Program Program Instruction Cycle

CPU = IC * CPI * Clk

Improving Performance of processors: ...

- Are “real” processors like LC 3?
- Design principles to improve the performance of the processor?
  - Pipelined processors: overlap execution of instructions
  - Superscalar processors: have multiple pipelined execution units
  - Multi-threaded processors: execute multiple threads
  - Multi-core: executed multiple threads and multiple programs on many cores/processors on a single chip

Back to our old friend: CPU time equation

- Recall equation for CPU time

cpu time = seconds = instructions x cycles x seconds
program program instruction cycle

= IC * CPI * Clk

- So what are we doing by pipelining the instruction execution process?
  - Clock?
  - Instruction Count?
  - CPI?
  - How is CPI affected by the various hazards?

Next: Memory

- Key component in a computer?
- Memory

- How are “real” memory systems organized?
  - How do they affect performance?
The computer is composed of input devices, a central processing unit, a memory unit and output devices.

Recall Technology Trends: The CPU-Memory Gap
- The increasing gap between DRAM, disk, and CPU speeds.

Memory Technology
- Random access memory
  - Can read from any location by supplying address of data
- Memory Comes in Many Flavors
  - Main RAM memory Key features
    - RAM is packaged as a chip
    - Basic storage unit is a cell (one bit per cell)
    - Multiple RAM chips form a memory
    - SRAM (Static Random Access Memory) or DRAM (Dynamic Random Access Memory)
  - ROM, EPROM, EEPROM, Flash, etc. – Non-Volatile
    - Read-only memories – store OS
  - "Secondary memory" Disks, Tapes, Flash etc.
- Difference in speed, price and “size”
  - Fast is small and/or expensive
  - Large is slow and/or cheap
**Computer Architecture 1**

**How is memory really organized?**

- Many types of memory with different speeds
- Processor speed and memory speed mismatched
  - Data transferred between memory and processor
    - Instructions or data
- What does processor do while waiting for data to be transferred?
  - Idle – processor is stalled leading to slowdown in speed and lower performance
- Why can’t we have memory as fast as processor
  - Technology, cost, size
- What is the solution then?

**“Memory” organization**

- Items on a desktop (register) or in a drawer (cache) are more readily accessible than those in a file cabinet (main memory) or in a closet in another room
- Keep more frequently accessed items on desktop, next frequent in drawer, etc.
- And things you need a lot less often in the closet in another room!

**Memory Hierarchies**

- Organize memory as a memory hierarchy.
- Key Principles
  - Locality – most programs do not access code or data uniformly
  - Smaller hardware is faster
- Goal
  - Design a memory hierarchy "with cost almost as low as the cheapest level of the hierarchy and speed almost as fast as the fastest level"!
  - This implies that we be clever about keeping more likely used data as "close" to the CPU as possible
- Levels provide subsets
  - Anything (data) found in a particular level is also found in the next level below.
  - Each level maps from a slower, larger memory to a smaller but faster memory
Key Concept-- Locality

- Principle of Locality:
  - Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
  - Temporal locality: Recently referenced items are likely to be referenced in the near future.
  - Spatial locality: Items with nearby addresses tend to be referenced close together in time.
- Nearby in time
- Nearby in space

Locality: Example

```
sum = 0;
for (i = 0; i < n; i++)
  sum += a[i];
return sum;
```

- Data
  - Reference array elements in succession (stride-1 reference pattern): Spatial locality
  - Reference sum in each iteration: Temporal locality
- Instructions
  - Reference instructions in sequence: Spatial locality
  - Cycle through loop repeatedly: Temporal locality

Next: The Memory Hierarchy . . .

- Brief Overview of Memory Design
  - Focus on Cache design
    - How does Cache memory work?
    - How are addresses mapped to Cache
    - How to rewrite code to get better cache performance?—code optimization
  - How do disks work?
  - Later in Operating Systems: Virtual memory—what is it?
Link to Performance?

- Simplified model:
  - Processor is (1) in execution or (2) waits for memory
    - "effective" (Real) CPI increases
  - Execution time = (execution cycles + memory stall cycles) \times cycle time

Memory Access time and Performance?

- \[ CPU = IC \times CPI \times Clk \]

- How does memory access time fit into CPU time equation?

- More stall cycles = increase in CPI

Performance: Simplified Goal?

- Improve performance = decrease stall cycles
  - Decrease time to access memory
  - How?
    - Organize memory as a hierarchy!
    - Most of the time you are accessing data in the fastest memory
Memory Unit

- An ordered sequence of storage cells, each capable of holding a piece of data.
- Address space
  - Size of memory: N bit address space = $2^N$ memory locations
- Addressability
  - Size of each memory location – k bits
- Total memory size = $k \cdot 2^N$ bits
- Assumption thus far: Processor/CPU gets data or instruction from some memory address (Inst fetch or Load/Store instruction)
  - Time depends on how is memory actually organized?
  - Can everything we need fit into a memory that is close to the CPU?

Typical Bus Structure Connecting CPU and Memory

- A bus is a collection of parallel wires that carry address, data, and control signals.
- Buses are typically shared by multiple devices.

Memory Read Transaction (1)

- CPU places address A on the memory bus.

Memory Read Transaction (2)

- Main memory reads A from the memory bus, retrieves word x, and places it on the bus.
Memory Read Transaction (3)

- CPU read word x from the bus and copies it into register %eax.

Load operation: `Load R0, A` or in Intel: `movl A, R0`

Memory Write Transaction (1)

- CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.

Store operation: `STORE R1,A` or in Intel: `movl R1, A`

Memory Write Transaction (2)

- CPU places data word y on the bus.

Store operation: `STORE R1, A` or in Intel: `movl R1, A`

Reality…Memory access
What happens if not found in main memory ...read from Disk

CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

Data not found in memory

Reading a Disk Sector (2)

Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

Reading a Disk Sector (3)

When the DMA transfer completes, the disk controller notifies the CPU with an interrupt (i.e., asserts a special "interrupt" pin on the CPU)
Why Cache?

- Gap between main memory speed and processor speed
  - Reading data/inst from memory will take more than 1 processor cycle
  - Increases time to execute program
- What?: Place a small but fast memory close to the processor
- How?: Why does this work
  - Principle of Locality

Recall...Locality

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

Locality Example:
- Data
  - Reference array elements in succession
    (stride-1 reference pattern): Spatial locality
  - Reference sum on each iteration: Temporal locality
- Instructions
  - Reference instructions in sequence: Spatial locality
  - Cycle through loop repeatedly: Temporal locality

Cache and Main Memory

Simple Model of Memory Hierarchy...

- Sequence of addresses
  - How many?
- CPU generates request for memory location
  - i.e., an address
    - How long does it take to get this data?
    - Depends where it is in the Memory hierarchy
- Simplified Model for memory hierarchy:
  - small amount of Fast On-chip Cache memory
  - Larger amount of off-chip Main memory
  - Huge Disk
**How does Cache memory work?**

- Address space = $2^N$ words each of some size $K$ bits
  - N bit address
- Memory addresses go from 0 to $2^N$-1
  - These are the addresses that the processor requests in the Load or Store instructions, or when fetching a new instruction (value in PC)
- Some of these memory locations are placed in the cache
  - If you see it in the cache then don’t need to go all the way to memory to read them
    - Faster time to read/write inst/data!

**Memory Access times**

- memory access time
  - On-chip Cache takes 1 processor cycle
  - Main memory takes a number (10-50) processor cycles
  - Disk takes a huge amount
- Simple model we will use:
  - Memory = Cache + Main memory
  - Small size Cache = not everything fits in it
- Cache organization:
  - Cache consists of a set of blocks each of some number of bytes
  - Only a block can be fetched into and out of cache
  - Eg: if block is 16 bytes, then load 16 bytes into cache
    - Cannot load a single byte

**Memory Access times using Simplified Model**

- If data is found in Cache then time = 1
  - Called a cache hit
- Else time is Main memory access time
  - Cache miss, means read from next level
- Note: need a ‘control unit’ to determine if location is in cache or not
- Cache controller
- Why does concept of caching work?
  - Principle of Locality
    - Programs access data nearby, or data/instructions that were used recently

**Terminology Summary**

- Hit: data appears in block in upper level (i.e. block X in cache)
  - Hit Rate: fraction of memory access found in upper level
  - Hit Time: time to access upper level which consists of RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieved from a block in the lower level (i.e. block Y in memory)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Extra time to replace a block in the upper level + Time to deliver the block to the processor
- Hit Time << Miss Penalty (500 instructions on Alpha 21264)
Memory Hierarchy--Performance

- Placing the fastest memory near the CPU can result in increases in performance.
- Consider the number of cycles the CPU is stalled waiting for a memory access: memory stall cycles.
- CPU execution time = (CPU clk cycles + Memory stall cycles) * clk cycle time.
- Memory stall cycles = number of misses * miss penalty.
- Fewer misses in cache = better performance!

Cache Performance – Simplified Models

- Memory stall cycles = number of misses * miss penalty
  - = IC*(memory accesses/instruction)*missing*miss penalty
- Hit rate (hit ratio) h = (no. of requests that are hits)/(total no. requests)
  - Miss rate = 1 – (hit rate)
- Cost of memory access = h Cc + (1-h) Cm
  - Cc is cost/time from cache, Cm is cost/time when miss – fetch from memory
  - Larger the value of h, lower the cost.

Average Memory Access Time

$$\text{AMAT} = \text{HitTime} + (1 - h) \times \text{MissPenalty}$$

- Hit time: basic time of every access.
  - Always look to check if data is in cache
- Hit rate (h): fraction of access that hit
  - usually substituted by miss rate m = (1-h)
- Miss penalty: extra time to fetch a block from lower level, including time to replace in CPU
  - Access time to read/write to memory module
    - Can extend the same equation to disks/networks.

Example 1

- System = Processor, Cache, Main Memory
  - Cache time = 1 processor cycle
  - Memory access time = 50 processor cycles
- Suppose out of 1000 memory accesses (due to Load/Store and Inst fetch)
  - 40 misses in the cache
  - 960 hit in the Cache
  - Miss ratio = 40/1000 = 4%
- Average memory access time with and without cache?
Example.

- Average memory access time with and without cache?
- AMAT-cache = 1 + miss ratio * miss penalty
  - 1 + (0.04)*50 = 3
- AMAT-without-cache = 50
- What happens if miss ratio increases?

Example 2

- In reality we have an application with a instruction mix
  - CPU time = IC*CPI*Clock
  - Effective CPI = CPI + Average Stalls per instruction
- How many memory access per instruction?
  - When is memory accessed?
- Example: suppose your program has 20% Load/Store operations

Example 2

- Suppose your program has 20% Load/Store operations
  - Memory is accessed once for each instruction PLUS again for each Load/Store
  - Out of 100 instructions, we will have 120 accesses to memory
  - Average memory accesses per instruction = 120/100 = 1.2
- How many stall cycles (waiting for memory)?
  - 1.2 * 0.04*50 = 1.2*2= 2.4
  - Processor is stalled 2.4 cycles each inst

Example 2 – complete example

- Assume “ideal” CPI (without memory stalls) is 1.5
- What is the CPU time now?
- CPU = IC * CPI * Clock
  - IC*CPI is the number of CPU cycles
- CPI is number of cycles for CPU execution per instruction
- Each instruction stalls for some number of cycles
- IC * (CPI + Avg Stall cycles)*Clock
  - IC*(1.5+2.4)*clock = IC*3.9 clock cycles
Summary of Memory Hierarchy

- Multiple levels of memory
  - Cache provides ‘fast access’ times
  - More hits to cache, the lower your memory access time – better program performance!

- Next:
  - How is cache memory designed
  - How is it controlled? Hardware needed?
  - Quick look at how disks work
  - Can we come up with ways to minimize disk access time?

- Return to program performance
  - Tie in memory design to program performance

Cache Memory Hardware Design

- Main memory has $2^N$ locations
- Cache has $2^k$ locations
  - Smaller than main memory
  - How to “organize” these cache locations?
- Processor generates N bit address
- The Cache Controller hardware must look at this N bit address and decide (a) if it is in cache and (b) where to place it in cache?

Cache Memory Design: Definitions

- Cache has a total size – number of bytes in cache
- Transfers take place in blocks
  - A whole block is transferred between memory and cache
- Locating a block requires two attributes:
  - Size of block
  - Organization of blocks within the cache
- Block size (also referred to as line size)
  - Smallest usable block size is the natural word size of the processor
  - Else would require splitting an access across blocks and slow down translation

Memory viewed as blocks

- If cache block size = K bytes, then memory can be viewed as contiguous set of blocks each of size K

16 byte memory

16 byte memory, With 4 byte sized Cache blocks; 4 blocks of memory
Where can a block be placed in a cache? - Cache Organization

- If each block has only one place it can appear in the cache, it is said to be "direct mapped".
  - Mapping is usually \((\text{Block address}) \mod (\text{Number of blocks in the cache})\)
- If a block can be placed anywhere in the cache, it is said to be fully associative.
- If a block can be placed in a restrictive set of places in the cache, the cache is set associative.
  - A set is a group of blocks in the cache. A block is first mapped onto a set, and then the block can be placed anywhere within that set.
  - \((\text{Block address}) \mod (\text{Number of sets in the cache})\)
  - If there are \(n\) blocks in a set, the cache is called \(n\)-way set associative.

Cache Design: How is data found in Cache?

- Note: Time to find data is the hit time; affects performance.
- Processor generates address request – some N bit address.
- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it? In which cache block do we look?
Memory address of N bits
- This is what is requested by processor
- We need mapping from Addresses to Cache
- How to break up these N bits into fields so that the cache controller can easily determine if the address requested is already stored in cache?

Focus on Direct mapped:
- How to implement \( N \text{ Mod.} P \text{ where } P=2^c \)
- How to determine which cache block to map the address -- index
- How to determine which of the words is the one in memory -- tag
- Since we are working with blocks, how to find the specific word within the block -- offset
- N bit address is broken into these 3 fields!

Example
- 8 bit address
  - Byte addressability
- 4 byte cache blocks
  - Each cache block has 4 bytes
  - Main memory has \( 2^8 \text{=} 256 \text{ bytes} \text{....or} \text{64 blocks} \)
- Total size of cache = 16 bytes
  - 4 blocks, each of 4 bytes
- Into which cache block do we place address 01101101
- How do we know which block from memory is in the cache block? 
  - Is it 01101101 or 11001101
**Example**

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1,2,3)</td>
<td>Block 0</td>
</tr>
<tr>
<td>(4,5,6,7)</td>
<td>Block 1</td>
</tr>
<tr>
<td>(8,9,10,11)</td>
<td></td>
</tr>
<tr>
<td>(12,13,14,15)</td>
<td></td>
</tr>
<tr>
<td>(16,17,18,19)</td>
<td></td>
</tr>
</tbody>
</table>

**Cache**

- Addresses in each block
- 4 bytes in each block
- Addresses: (0,1,2,3)

---

**Addressing -- Example**

- 8 bit address
  - Byte addressability & 256 bytes in memory
- 4 byte cache blocks
  - Each cache block has 4 bytes = need 2 bits to specify which byte within the block
  - 4 bytes in each block – memory is 256/4 = 64 blocks
- Total size of cache = 16 bytes
  - 4 blocks in cache = apply Modulo 4 mapping
- Into which cache block do we place 01101101
  - Last two LSB are offset within a block
  - Next 2 LSB are modulo 4 and specify cache block
- How do we know which block from memory is in the cache block?
  - Is it 01101101 or 11001101

- The three fields are:
  - **Tag**: the 4 most significant bits
    - Tell you which of the memory blocks are in the cache
  - **Offset** (block offset): the 2 least significant bits
    - Gives the address within the cache block
  - **Index**: the remaining 2 bits
    - Tells you into which cache block a memory address is placed

- In general: For N bit address, 2^k bytes in cache block, 2 blocks of total cache size
  - Tag of (N-k-j) bits: the most significant bits
  - Offset of K bits: the least significant k bits
  - Index of j bits

- Memory block M gets placed in cache block \((M \text{ mod}(2^j))\)
  - M is (N-k) most significant bits

---

**So how complex is the hardware required?**

- Given N bit address, the cache controller needs to determine if the requested word is in cache or not
  - If not, then send the address to the memory bus
- This process has to be built into hardware
  - Complexity of the circuit determines time taken to determine if word is in cache
  - Hit time is function of complexity of circuit
  - For direct mapped, can you think of a hardware design?
Finding cache block

- Given the i bits of address, how do you find the cache block?
  - Mod \(4\) (Mod \(2^i\) where \(2^i\) is number of cache blocks)

- What device?

Finding word within a block

- Following four addresses are all in one block
  - 01101100
  - 01101101
  - 01101110
  - 01101111

- How do you select one of these four?

Matching the tag of a memory block

- Number of memory blocks could all be mapped to the same cache block
- Following two get mapped to same block
  - 01101101
  - 01111101
- You are searching to see if 01101101 is in the cache.....
- How do you check if cache contains this block with tag 0110?

Direct Mapped Caches

- Tag match
- Block index

- Tag decoder
- Index decoder
- Multiplexer
Performance Impact of cache parameters

- Cache performance is related to cache design
  - Size of cache, size of each block, organization

- Two ways of improving performance:
  - Decreasing the miss ratio
  - Decreasing the miss penalty

Summary: Memory Access time optimization

- If each access to memory leads to a cache hit
  then time to fetch from memory is one cycle
  - Program performance is good!

- If each access to memory leads to a cache miss
  then time to fetch from memory is much larger than 1 cycle
  - Program performance is bad!

- Design Goal:
  How to arrange data/instructions so that we have as few cache misses as possible.

How about rewriting code to improve cache rates?

- This is part of what code optimization accomplishes…

Secondary Memory

- CPU fetches from memory, memory is a sequence of $2^N$ locations
- What happens if main memory (chip capacity) is less than $2^N$
- Data and programs may be stored in a non-volatile component
  - MS-Word executable is on disk
  - Your application data
- What happens if more than one process is running on the system
  - Multiprogramming
  - What is the address space available to each user?
- Need to use Disks!
The Complete memory hierarchy

- Processor has a set of registers
  - Processor instructions operate on contents in the registers
- Small, fast cache memory placed near the processor
- Main memory sitting outside the chip
  - If data is not in the cache then fetch from main memory
  - Takes longer to access main memory than cache
- Disk sitting outside the motherboard
  - If data/program not in main memory then fetch/load from disk
  - Takes much longer to access disk than main memory

Why does memory hierarchy work?

- Principle of locality!
  - Exploit at each level.

How do disks work?

- Disks consist of platters, each with two surfaces.
- Each surface consists of concentric rings called tracks.
- Each track consists of sectors separated by gaps.
Disk Geometry (Multiple-Platter View)
- Aligned tracks form a cylinder.

Disk Operation (Single-Platter View)
- The disk surface spins at a fixed rotational rate.
- The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air.
- By moving radially, the arm can position the read/write head over any track.

Disk Access Time
- Average time to access some target sector approximated by:
  - $T_{access} = T_{avg \ seek} + T_{avg \ rotation} + T_{avg \ transfer}$
- Seek time ($T_{avg \ seek}$)
  - Time to position heads over cylinder containing target sector.
  - Typical $T_{avg \ seek} = 9$ ms
- Rotational latency ($T_{avg \ rotation}$)
  - Time waiting for first bit of target sector to pass under r/w head.
  - $T_{avg \ rotation} = \frac{1}{2} \times \frac{1}{\text{RPMs}} \times 60 \ \text{sec/1 min}$
- Transfer time ($T_{avg \ transfer}$)
  - Time to read the bits in the target sector.
  - $T_{avg \ transfer} = \frac{1}{\text{RPM}} \times \frac{1}{\text{avg # sectors/track}} \times 60 \ \text{sec/1 min}$

Accessing a Disk Page
- Time to access (read/write) a disk block:
  - seek time (moving arms to position disk head on track)
  - rotational delay (waiting for block to rotate under head)
  - transfer time (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
- Key to lower I/O cost: reduce seek/rotation delays!
  - Hardware vs. software solutions?
Placement of Data on Disk

- Placement of data on disk can affect performance of program
- Example: If you are reading an entire array, then:
  - place the data in consecutive disk blocks
  - seek time to get first disk block
  - Remaining blocks will incur no seek time
  - Time = $T_{\text{seek}} + N(T_{\text{transfer}} + T_{\text{rot}})$
  - Naïve approach: $N(T_{\text{seek}} + T_{\text{transfer}} + T_{\text{rot}})$
  - Speedup: $N^2 T_{\text{seek}} \sim O(N)$

Logical Disk Blocks

- Modern disks present a simpler abstract view of the complex sector geometry:
  - The set of available sectors is modeled as a sequence of $b$-sized logical blocks (0, 1, 2, ...)
  - Mapping between logical blocks and actual (physical) sectors
    - Maintained by hardware/firmware device called disk controller.
    - Converts requests for logical blocks into (surface, track, sector) triples.
  - Allows controller to set aside spare cylinders for each zone.
    - Accounts for the difference in "formatted capacity" and "maximum capacity".

I/O Bus

- Expansion slots for other devices such as network adapters.

Reading a Disk Sector (1)

- CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.
Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

When the DMA transfer completes, the disk controller notifies the CPU with an interrupt (i.e., asserts a special "interrupt" pin on the CPU).

Last step: Virtual Memory

- N bit address space...
- If word is not in main memory then it is on disk – need a controller to manage this...analogous to cache controller
  - Virtual memory management system
- Note: transfers take place from disk in “blocks” of M bytes (sector) – page of data
- Memory consists of a number of pages
  - Determine if the page is in main memory or not
- N bit address broken into fields which determine page number, and whether page is in the memory or disk
- If page size is 1024 bits...how to organize the N bit address ??

Virtual Memory

- Main memory can act as a cache for the secondary storage (disk)

  - Advantages:
    - Illusion of having more physical memory
    - Program relocation
    - Protection
What is multiprogramming and Why?

- Processor overlaps execution of two processes/programs
  - When one is waiting for I/O, the other is “swapped” in to the processor
    - Save the “state” of the process being swapped out
- Processes need to share memory
  - Each has its own address space
- Leads to better throughput and utilization

Memory Hierarchy: The Big Picture

- Data movement in a memory hierarchy
- Pages: virtual memory blocks
  - Page faults: the data is not in memory, retrieve it from disk
    - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
    - reducing page faults is important (LRU is worth the price)
    - can handle the faults in software instead of hardware
    - using write-through is too expensive so we use writeback
Page Tables

More on Virtual memory

- This is part of the Operating system