Lecture 26: Memory and Locality
Announcements

- In-person Final Scores released
- Final Resubmission open—will close **Wednesday June 7th at 11:59PM**
  - NO LATE SUBMISSIONS
- EX7 deadline was inconsistent across the platforms, so it is now officially due tonight
  - Grades will be released this weekend
  - Re-grades will open this weekend and close **Wednesday June 7th**

---

![Histogram Image]

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Median</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.0</td>
<td>55.0</td>
<td>100.0</td>
<td>56.39</td>
<td>20.98</td>
</tr>
</tbody>
</table>

Slido Event #3731178  
https://app.sli.do/event/8QNyh3w8FPmi6b9qRf8vxS
Memory and Locality

Memory Usage and Movement
**Review: Binary, Bits and Bytes**

**Binary**
A base-2 system of representing numbers using only 1s and 0s
- vs decimal, base 10, which has 9 symbols

**Bit**
The smallest unit of computer memory represented as a single binary value either 0 or 1

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Decimal Break Down</th>
<th>Binary</th>
<th>Binary Break Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>((0 \times 10^0))</td>
<td>0</td>
<td>((0 \times 2^0))</td>
</tr>
<tr>
<td>1</td>
<td>((1 \times 10^0))</td>
<td>1</td>
<td>((1 \times 2^0))</td>
</tr>
<tr>
<td>10</td>
<td>((1 \times 10^1) + (0 \times 10^0))</td>
<td>1010</td>
<td>((1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0))</td>
</tr>
<tr>
<td>12</td>
<td>((1 \times 10^1) + (2 \times 10^0))</td>
<td>1100</td>
<td>((1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0))</td>
</tr>
<tr>
<td>127</td>
<td>((1 \times 10^2) + (1 \times 10^1) + (2 \times 10^0))</td>
<td>01111111</td>
<td>((0 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0))</td>
</tr>
</tbody>
</table>

**Byte**
The most commonly referred to unit of memory, a grouping of 8 bits
Can represent 265 different numbers \((2^8)\)
1 Kilobyte = 1 thousand bytes (kb)
1 Megabyte = 1 million bytes (mb)
1 Gigabyte = 1 billion bytes (gb)
public int sum1(int n, int m, int[][] table) {
    int output = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            output += table[i][j];
        }
    }
    return output;
}

public int sum2(int n, int m, int[][] table) {
    int output = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            output += table[j][i];
        }
    }
    return output;
}

What do these two methods do?
What is the big-$\Theta$ $\Theta(n \times m)$
Why not time code?

Actual time to completion can vary depending on hardware, state of computer and many other factors.

These graphs are of times to run add and contains on structures of various sizes of N and you can see inconsistencies in individual runs which can make determining the overall relationship between the code and runtime less clear.

You can find the code to run these tests on your own machine on the course website!
Incorrect Assumptions

Accessing memory is a quick and constant-time operation

Sometimes accessing memory is cheaper and easier than at other times

Sometimes accessing memory is very slow
RAM (Random-Access Memory)

RAM is where data gets stored for the programs you run. Think of it as the main memory storage location for your programs.

RAM goes by a ton of different names: memory, main memory, RAM are all names for this same thing.
RAM can be represented as a huge array

RAM:
- addresses, storing stuff at specific locations
- random access

Arrays
- indices, storing stuff at specific locations
- random access

This is a main takeaway

If you’re interested in deeper than this: [https://www.youtube.com/watch?v=fpnE6UAfbtU](https://www.youtube.com/watch?v=fpnE6UAfbtU) or take some EE classes?
A quick aside: Types of memory

**Arrays - contiguous memory:** when the “new” keyword is used on an array the operating system sets aside a single, right-sized block of computer memory

```java
int[] array = new int[3];
array[0] = 3;
array[1] = 7;
array[2] = 3;
```

**Nodes - non-contiguous memory:** when the “new” keyword is used on a single node the operating system sets aside enough space for that object at the next available memory location

```java
Node front = new Node(3);
front.next = new Node(7);
front.next.next = new Node(3);
```

More on how memory impacts runtime later in this course...
Memory Architecture

**CPU Register**
- What is it? The brain of the computer!
- Typical Size: 32 bits
- Time: ≈free

**L1 Cache**
- Extra memory to make accessing it faster
- Typical Size: 128KB
- Time: 0.5 ns

**L2 Cache**
- Extra memory to make accessing it faster
- Typical Size: 2MB
- Time: 7 ns

**RAM**
- Working memory, what your programs need
- Typical Size: 8GB
- Time: 100 ns

**Disk**
- Large, longtime storage
- Typical Size: 1 TB
- Time: 8,000,000 ns
Memory Architecture

Takeaways:

- the more memory a layer can store, the slower it is (generally)
- accessing the disk is very slow

Computer Design Decisions

- Physics
- Speed of light
- Physical closeness to CPU
- Cost
- “good enough” to achieve speed
- Balance between speed and space
Locality

How does the OS minimize disk accesses?

Spatial Locality
Computers try to partition memory you are likely to use close by
- Arrays
- Fields

Temporal Locality
Computers assume the memory you have just accessed you will likely access again in the near future
Leveraging Spatial Locality

When looking up address in “slow layer”

- bring in more than you need based on what’s nearby
- cost of bringing 1 byte vs several bytes is the same
- Data Carpool!
Memory and Locality

Memory Usage and Movement
Solution to Mercy’s traveling problem

If we know Mercy is going to keep eating tuna ... why not buy a bunch during a single trip and save them all somewhere closer than the store?

Let’s get Mercy a refrigerator!
CPU – kind of like the home / brain of your computer. Pretty much all computation is done here and data needs to move here to do anything significant with it (math, if checks, normal statement execution).

Data travels between RAM and the CPU, but it’s slow.
After CPU

Cache!

Bring a bunch of food back when you go all the way to the store

Bring a bunch of data back when you go all the way to RAM

RAM
Cache

Rough definition: a place to store some memory that’s smaller and closer to the CPU compared to RAM. Because caches are closer to the CPU (where your data generally needs to go to be computed / modified / acted on) getting data from cache to CPU is a lot quicker than from RAM to CPU. This means we love when the data we want to access is conveniently in the cache.

Generally we always store some data here in hopes that it will be used in the future and that we save ourselves the distance / time it takes to go to RAM.

Analogy from earlier: The refrigerator (a cache) in your house to store food closer to you than the store. Walking to your fridge is much quicker than walking to the store!
Bring a bunch of food back when you go all the way to the store

This is a big idea!

Bring a bunch of data back when you go all the way to RAM

After CPU

Cache!
How is a bunch of memory taken from RAM?

- Imagine you want to retrieve the 1 at index 4 in RAM.
- Your computer is smart enough to know to grab some of the surrounding data because computer designers think that it’s reasonably likely you’ll want to access that data too.
  - (You don’t have to do anything in your code for this to happen – it happens automatically every time you access data!)
- To answer the title question, technically the term / units of transfer is in terms of ‘blocks’.

0 99 21 24 1 22 5 2 3 1 1 1 0 0 5 1 2 22 21 4

This is a big idea (continued)!
How is a bunch of memory taken from RAM? (continued)

CPU

original data (the 1) we wanted to look up gets passed back to the cpu

all the data from the block gets brought to the cache

| 0 | 99 | 21 | 24 | 1 | 22 | 5 | 2 | 3 | 1 | 1 | 1 | 0 | 0 | 5 | 1 | 2 | 22 | 21 | 4 |
How does this pattern of memory grabbing affect our programs?

This should have a major impact on programming with arrays. Say we access an index of an array that is stored in RAM. Because we grab a whole bunch of contiguous memory even when we just access one index in RAM, we’ll probably be grabbing other nearby parts of our array and storing that in our cache for quick access later.

Imagine that the below memory is just an entire array of length 13, with some data in it.

```
0 99 21 24 1 22 5 2 3 1 1 1 0
```

Just by accessing one element we bring the nearby elements back with us to the cache. In this case, it’s almost all of the array!
Leveraging Temporal Locality

When looking up address in “slow layer”

Once we load something into RAM or cache, keep it around or a while

But these layers are smaller
  ● When do we “evict” memory to make room?
Moving Memory

Amount of memory moved from disk to RAM
- Called a “block” or “page”
  - ≈4kb
- Smallest unit of data on disk

Amount of memory moved from RAM to Cache
- called a “cache line”
  - ≈64 bytes

Operating System is the Memory Boss controls
- page and cache line size
- decides when to move data to cache or evict
Thought Experiment

public int sum1(int n, int m, int[][] table) {
    int output = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            output += table[i][j];
        }
    }
    return output;
}

public int sum2(int n, int m, int[][] table) {
    int output = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            output += table[j][i];
        }
    }
    return output;
}

Why does sum1 run so much faster than sum2?
sum1 takes advantage of spatial and temporal locality
Java and Memory

What happens when you use the “new” keyword in Java?

Your program asks the Java Virtual Machine for more memory from the “heap”
- Pile of recently used memory

If necessary the JVM asks Operating System for more memory
- Hardware can only allocate in units of page
- If you want 100 bytes you get 4kb
- Each page is contiguous

What happens when you create a new array?
- Program asks JVM for one long, contiguous chunk of memory

What happens when you create a new object?
- Program asks the JVM for any random place in memory

What happens when you read an array index?
- Program asks JVM for the address, JVM hands off to OS
- OS checks the L1 caches, the L2 caches then RAM then disk to find it
- If data is found, OS loads it into caches to speed up future lookups

What happens when we open and read data from a file?
- Files are always stored on disk, must make a disk access
Array v Linked List

Is iterating over an ArrayList faster than iterating over a LinkedList?

Answer:

LinkedList nodes can be stored in memory, which means the don’t have spatial locality. The ArrayList is more likely to be stored in contiguous regions of memory, so it should be quicker to access based on how the OS will load the data into our different memory layers.
Questions?
That’s all!