Lecture 04: Embarrassingly Parallel, or Not

COSC 272: Parallel and Distributed Computing

Spring 2023

Announcements

1. Programming Assignment 01 Posted
   - ignore HPC cluster part of assignment for Friday
   - accounts registered, but no documentation yet
   - visit hpc.amherst.edu
   - ssh access: `[amherstid]@hpc.amherst.edu`
2. First written assignment next Friday
   - posted this weekend
3. Office Hours
   - TA (Mary Kate) Office Hours Wednesday 7–9pm, SCCE C109
   - My individual OH: Thursday 1:00–2:30

Outline

1. Lecture 03 Activity
2. Parallelism vs Concurrency
3. Embarrassingly Parallel Problem
4. Limitations of Parallelism

Lecture 03 Activity

Shared between threads

```
void increment(int[] a) {
    int i = 0;
    while (i < a.length) {
        a[i] = a[i] + 1;
        i = i + 1;
    }
}
```

thread local variable

increment ith value of a

Question 1

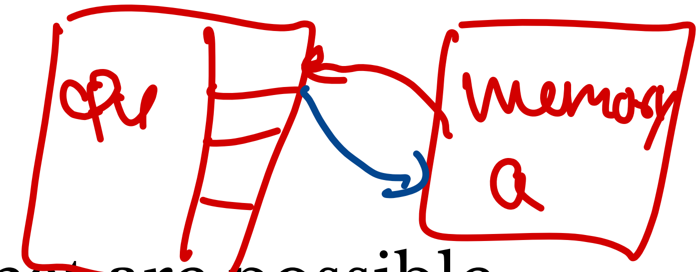
If `a = [0, 0, 0, 0]` and two threads, what are possible outcomes?

```
void increment(int[] a) {
    int i = 0;
    while (i < a.length) {
        a[i] = a[i] + 1;
        i = i + 1;
    }
}
```

[*, *, *, *]

each star could be 1 or 2

T1	T2
read a[0]	
	read a[0]
inc	
	inc
write 1	
	write 1



Question 2

If `a = [0, 0, 0, 0]` and k threads, what are possible outcomes?

```java
void increment(int[] a) {
    int i = 0;
    while (i < a.length) {
        a[i] = a[i] + 1;
        i = i + 1;
    }
}
```

each index could store any value from 1 to k

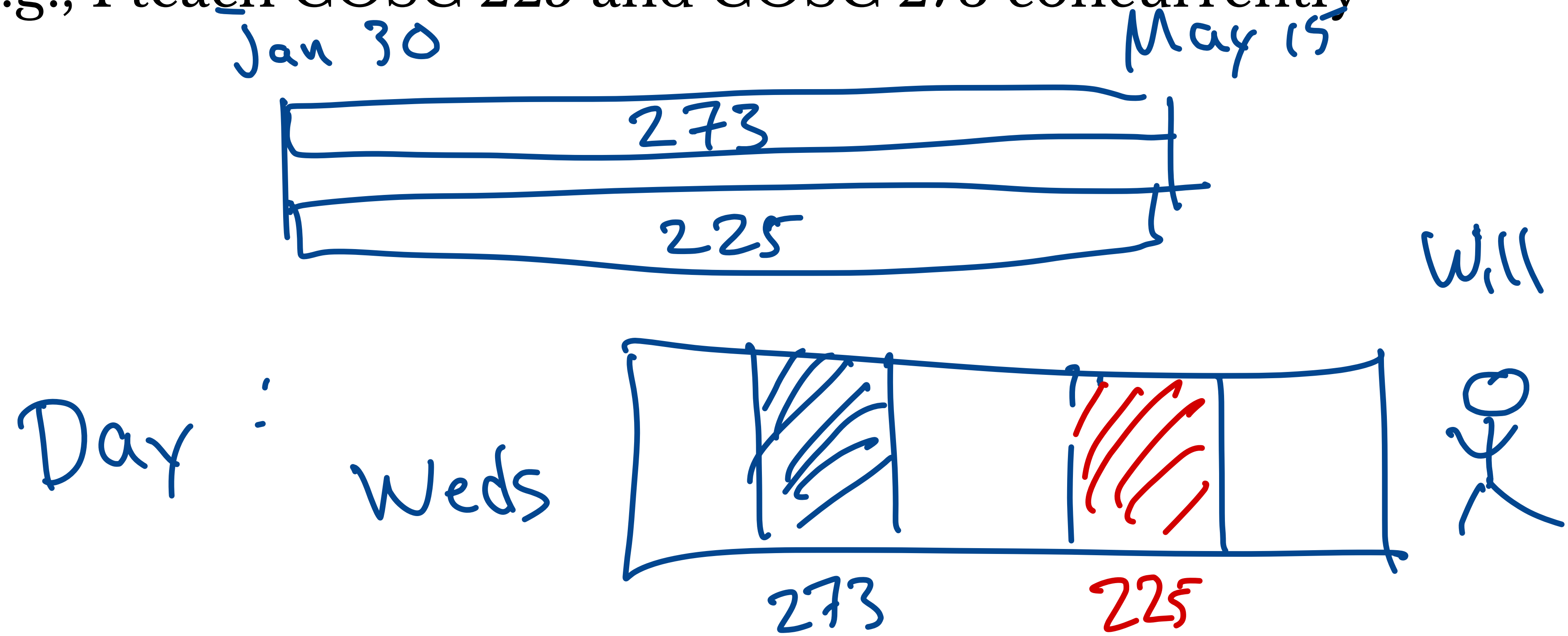
k = 10   how to get 6? (a[0])

- first 5 threads read/inc/write in succession
- last 5 threads all read, +inc then all write

Parallelism vs Concurrency

Concurrency performing multiple tasks that occupy overlapping time intervals

- E.g., I teach COSC 225 and COSC 273 concurrently



Parallelism vs Concurrency

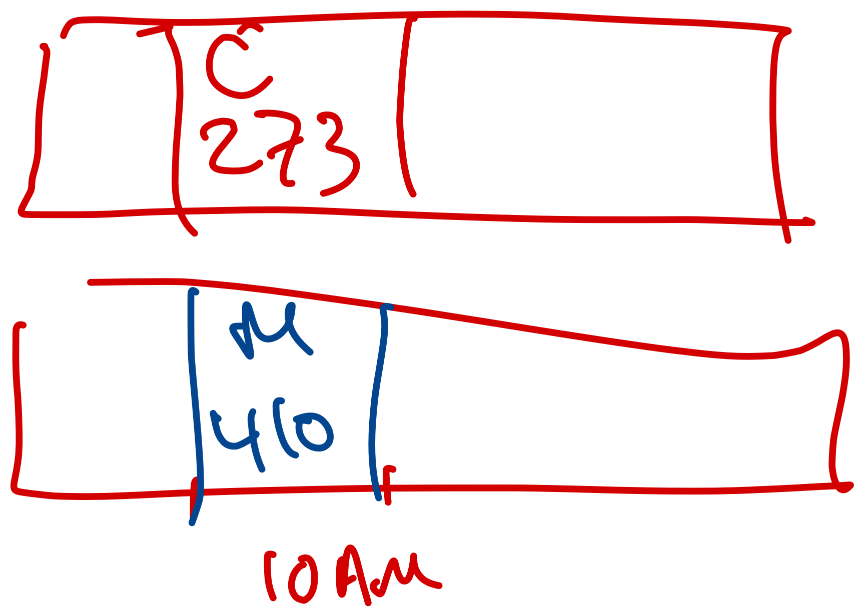
Concurrency performing multiple tasks that occupy overlapping time intervals

- E.g., I teach COSC 225 and COSC 273 concurrently

Parallelism making progress on multiple tasks at the same time

- E.g., COSC 273 and MATH 410 are taught in parallel (MWF 10-10:50)
- parallel $\implies$ concurrent

Weds



Virtues and Perils

Parallelism can give performance boost

- performance is one focus of this class

Virtues and Perils

Parallelism can give performance boost

- performance is one focus of this class

Concurrency is necessary for basic functionality of computers

- cannot execute multiple programs without concurrency
- operating system typically handles issues of concurrency
  - why you probably haven't encountered concurrency before

Virtues and Perils

Parallelism can give performance boost

- performance is one focus of this class

Concurrency is necessary for basic functionality of computers

- cannot execute multiple programs without concurrency
- operating system typically handles issues of concurrency
  - why you probably haven’t encountered concurrency before

Issues of nondeterminism exist for *concurrent* programs, not just parallel ones

Back to Counter

$O(n^2) \rightarrow O(\frac{n^2}{k})$

```
public void increment () {
    ++count;
}
```

How could we **fix** the problem of mis-counting?

- Want every increment to count!
  - Threads have own local counter
  - at end, accumulate values

“Easy” Solution

Each thread stores own private count!

- run threads until they’re done
- aggregate local counts when threads terminate

Question

When might "easy" solution not be sufficient?

1. Might need intermediate counts
2. Counts could be on-going (no fixed termination)
3. diff threads see count during exec.

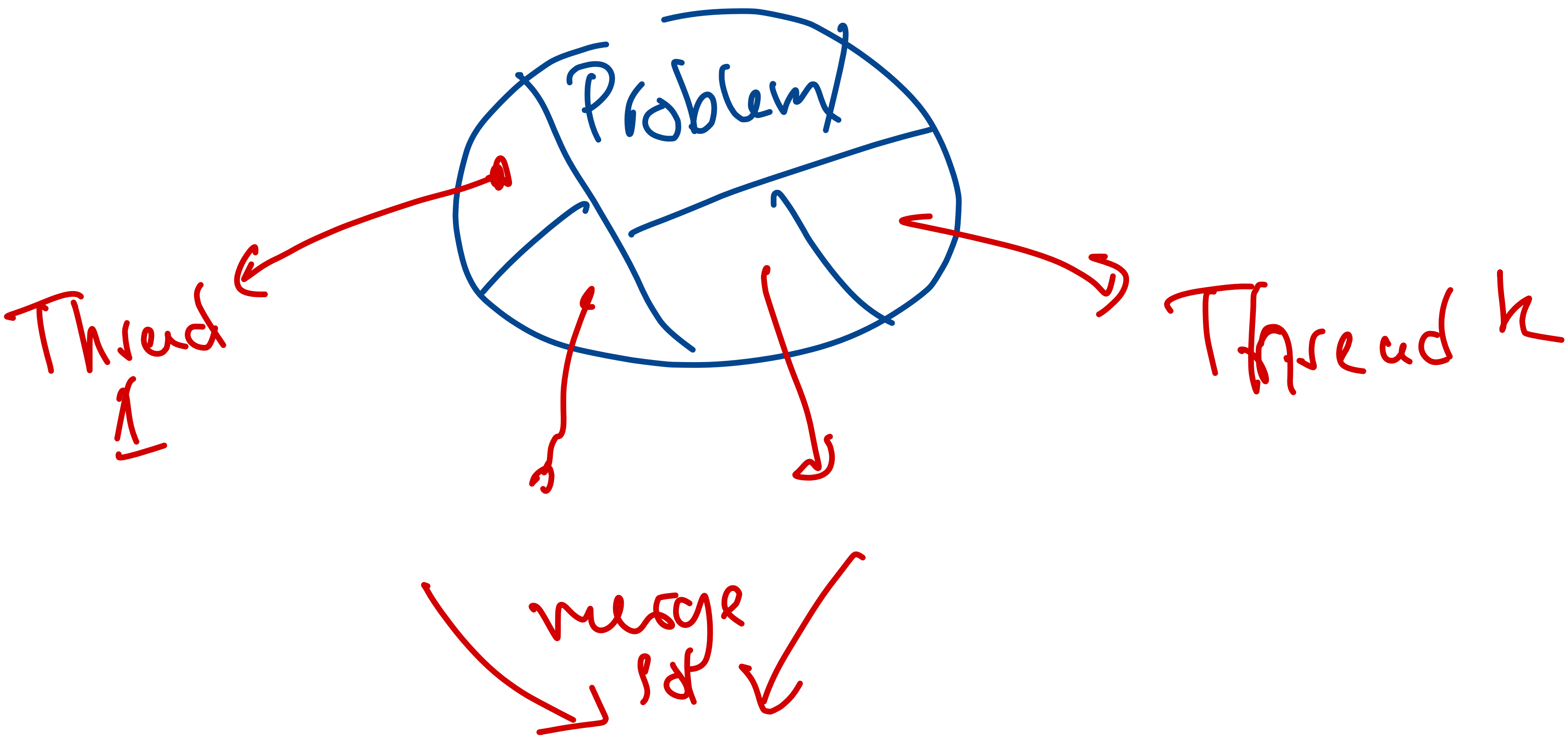
Question

When might “easy” solution not be sufficient?

We’ll revisit this next week

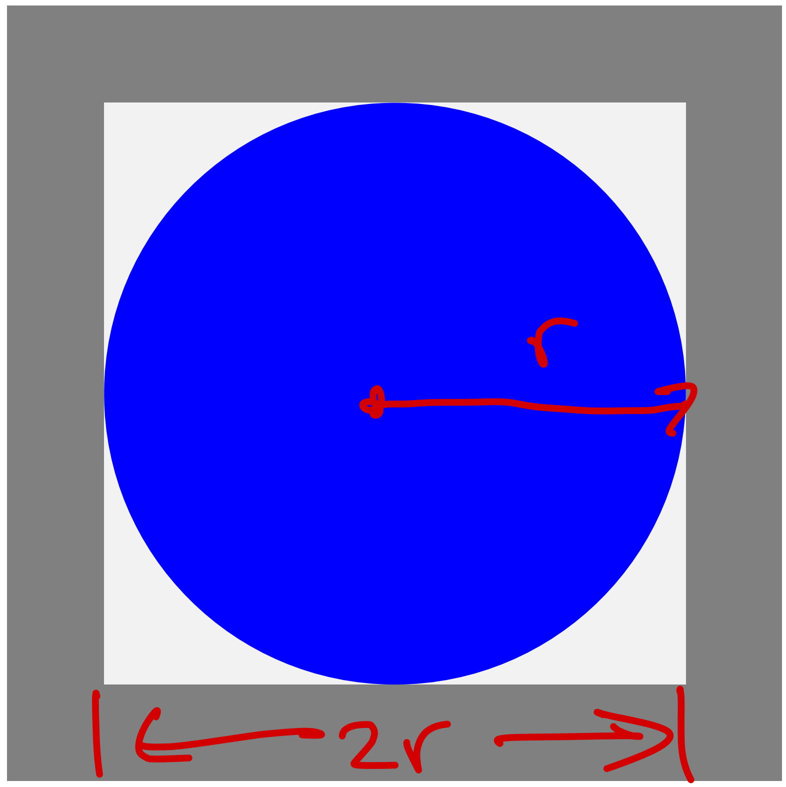
Embarrassingly Parallel Problems

A computational problem is **embarrassingly parallel** if it can be broken into many **simple** computations, (almost) all of which can be performed in parallel.



Example: Monte Carlo Estimation

A Formula from High School



Area of a disk: $A = \pi r^2$

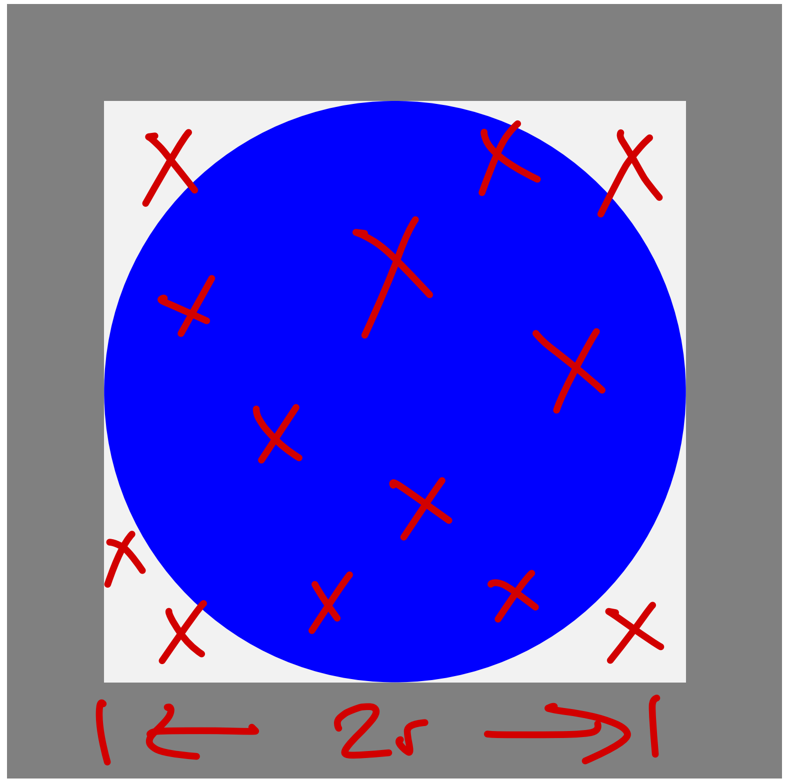
3.1415926535 . . .

An Idea from Probability

Pick a random point inside the framed region.

Probability dart hits circle

$$= \frac{\text{Area C}}{\text{Area Sq}} = \frac{\pi r^2}{4 r^2} = \boxed{\frac{\pi}{4}}$$



A square: $4r^2$

A circ: πr^2

The *probability* the point lies in the disk is proportional to the disk's area.

In More Detail

- area of disk is πr^2
- area of surrounding square is $(2r)^2 = 4r^2$
- the probability that a (uniformly) random point in the square lies in the disk is: $\frac{\text{area of circle}}{\text{area of square}} = \frac{\pi r^2}{4r^2} = \frac{1}{4}\pi$.

so...

Estimation by Sampling

...to estimate π, suffices to estimate the probability that a random point point in the square lies inside the disk:

- pick a bunch of random points
- see how many lie in disk
- $p =$ proportion of points that do
- $\pi \approx 4p$

Example of **Monte Carlo method**

Question

Why is Monte Carlo estimation embarrassingly parallel?

Another Question

How much performance increase with k cores?

Another Question

How much performance increase with k cores?

- What if $k \approx$ number of samples taken?

Not So Parallel

Dependencies?

```
a1 = b1 + c1;
a2 - b2 + c2;
d = a1 * a2
```

Not So Parallel

Dependencies?

```
a1 = b1 + c1;
a2 - b2 + c2;
d = a1 * a2
```

Dependency relation: directed acyclic graph (DAG)

More Generally

Consider a program that requires

- N elementary operations
- T time to run sequentially

Suppose

- a p-fraction of operations can be performed in parallel
- $1 - p$ fraction must be performed sequentially

Question: how long could program take with n parallel machines?

Idea

With n parallel machines:

- perform p-fraction of parallelizable ops in parallel on all n machines
  - total time $\frac{T \cdot p}{n}$
- perform remaining ops sequentially on a single machine
  - total time $T \cdot (1 - p)$

Total time: $T \cdot (1 - p) + T \cdot \frac{p}{n} = T \cdot \left(1 - p + \frac{p}{n}\right)$

How Much Improvement?

The **speedup** is the ratio of the original time T to the parallel time $T \cdot \left(1 - p + \frac{p}{n}\right)$:

- $S = \frac{1}{1-p+\frac{p}{n}}$

This relation is called **Amdahl's Law**

How Much Improvement?

The **speedup** is the ratio of the original time T to the parallel time $T \cdot \left(1 - p + \frac{p}{n}\right)$:

- $S = \frac{1}{1-p+\frac{p}{n}}$

This relation is called **Amdahl's Law**

This is the best performance improvement possible **in principle**

- may not be achievable in practice!

Example

1 person can chop 1 onion per minute

Recipe calls for:

- chop 6 onions
- saute onions for 4 minutes

Note:

- chopping onions can be done in parallel
- sauteing
  - takes 4 minutes no matter what
  - must be accomplished after chopping

Example (continued)

How much can the cooking process be sped up by n cooks?

Example (continued)

- For one chef, $T = 6 + 4 = 10$
- Only chopping onions is parallelizable, so $p = 6/10 = 0.6$
- Amdahl's Law:
  - $S = \frac{1}{1-p-\frac{p}{n}} = \frac{1}{0.4+\frac{1}{n}0.6}$
- So:
  - $n = 2 \implies S = 1.43$
  - $n = 3 \implies S = 1.67$
  - $n = 6 \implies S = 2$
- Always have $S < 1/(1 - p) = 2.5$

Speedup Improvement by Adding More Processors

- Second processor: 43%
- Third processor: 17%
- Fourth processor: 9%
- Fifth processor: 6%
- Sixth processor 4%

Latency vs Number of Processors

How does latency T scale with n?

- Adding more processors has *declining marginal utility*:
  - each additional processor has a smaller effect on total performance
  - at some point, adding more processors to a computation is wasteful
- Another consideration:
  - after parallel ops have been performed, extra processors are idle (potentially wasteful!)

Remarks

The proportion of parallelizable operations p is not always obvious from problem statement

Remarks

The proportion of parallelizable operations p is not always obvious from problem statement

- Amdahl's law a valuable heuristic for general phenomena:
  1. an n-fold increase in parallel processing power does not typically give an n-fold speedup in computations
  2. adding new parallel processors becomes less helpful the more parallel processors you already have

Remarks

The proportion of parallelizable operations p is not always obvious from problem statement

- Amdahl's law a valuable heuristic for general phenomena:
  1. an n-fold increase in parallel processing power does not typically give an n-fold speedup in computations
  2. adding new parallel processors becomes less helpful the more parallel processors you already have
- Often helpful to think about scheduling subtasks (not individual operations)
- May have relationships between tasks (e.g., one must be performed before another)

Next Time

Start **Mutual Exclusion**

- How can we fix our `Counter` to work as intended if we need to maintain a running count that can be accessed by multiple threads?