# Lecture 15: More Mandelbrot and Thread Pools

COSC 273: Parallel and Distributed Computing

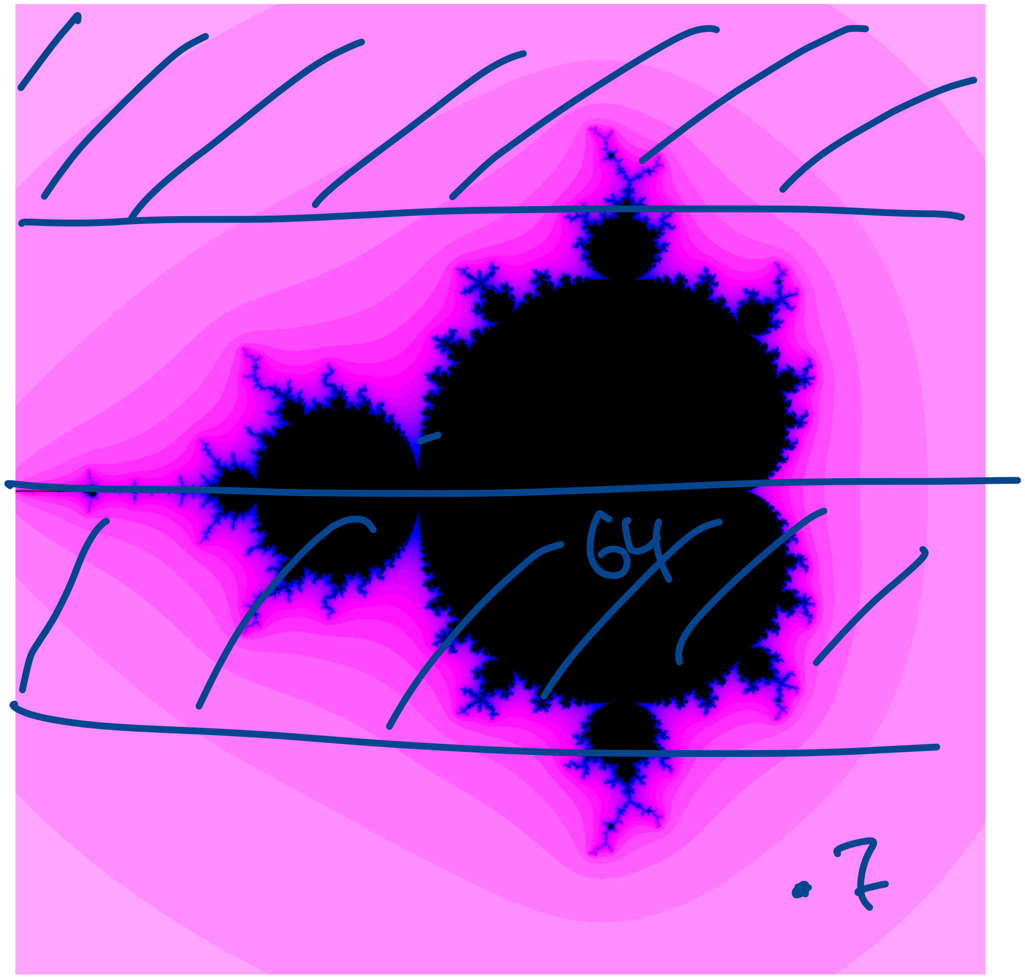
Spring 2023

# Outline

1. Mandelbrot Task
2. Thread Pools

# Mandelbrot Task

Draw this picture as quickly as possible!

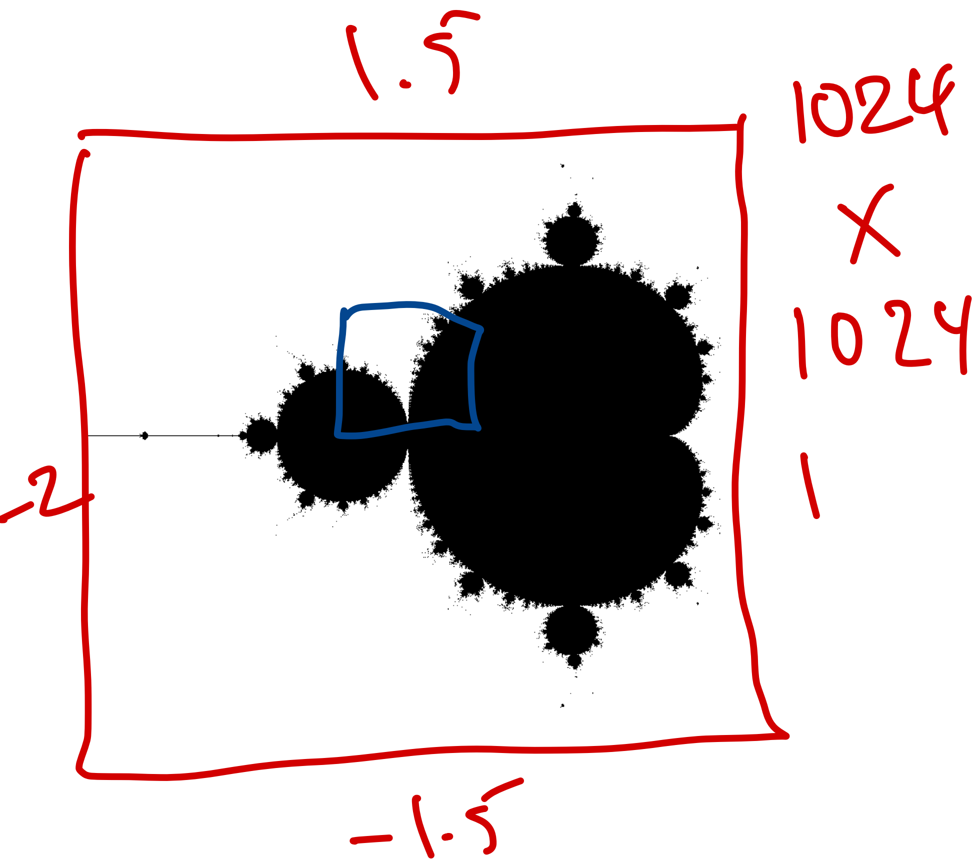


# Defining the Mandelbrot Set

To determine if $c$ is in the Mandelbrot set $M$:

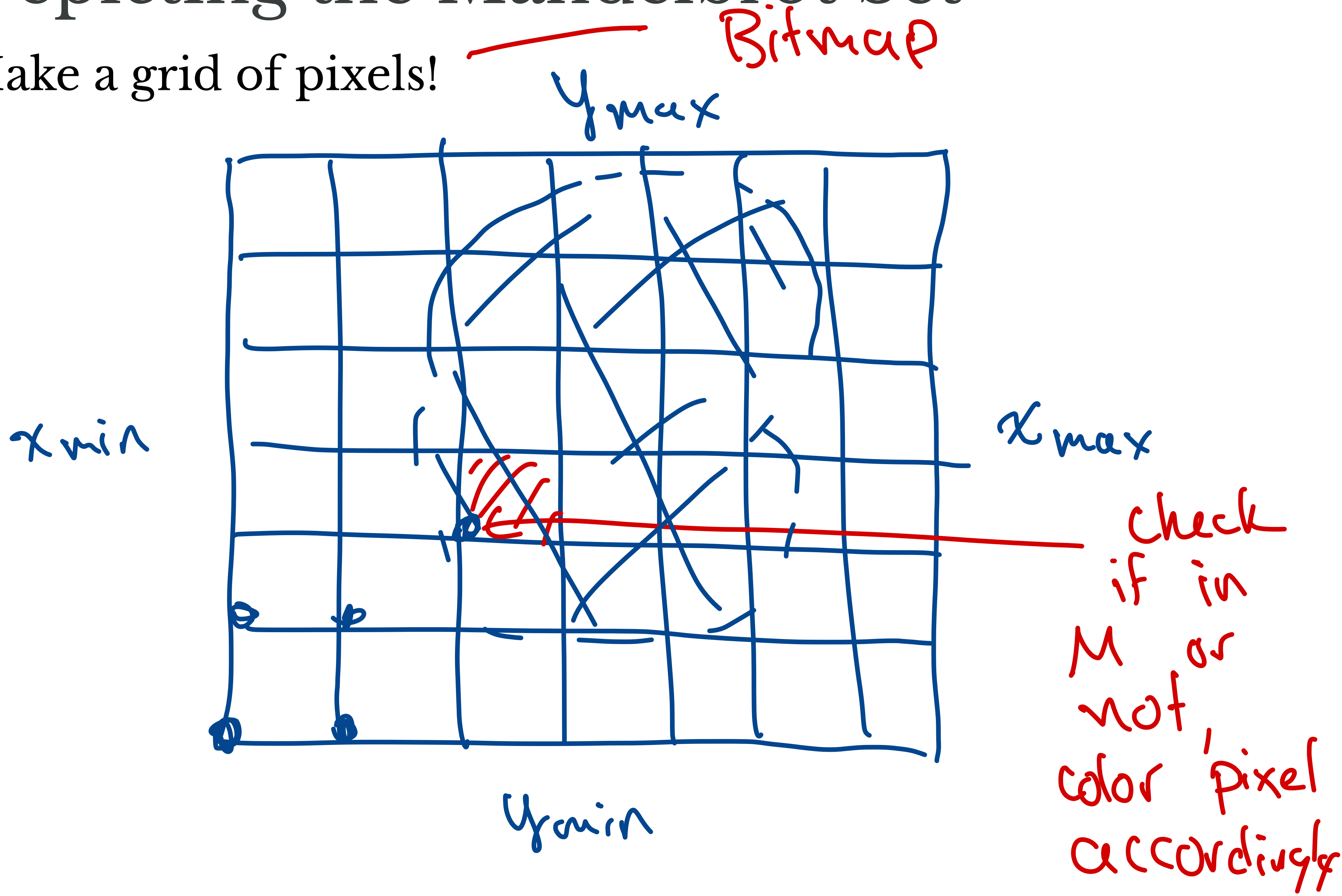
- compute $z_1 = c$
- define $z_n = z_{n-1}^2 + c$ for $n > 1$

If $z_n$ remains **bounded**, $c$ is in $M$; otherwise $c$ is not in $M$.



# Depicting the Mandelbrot Set

Make a grid of pixels!



# Computing the Mandelbrot Set

Choose parameters:

- $N$ number of iterations
- $M$ maximum modulus $(M > 2)$

length — distance from 0

Given a complex number $c$:

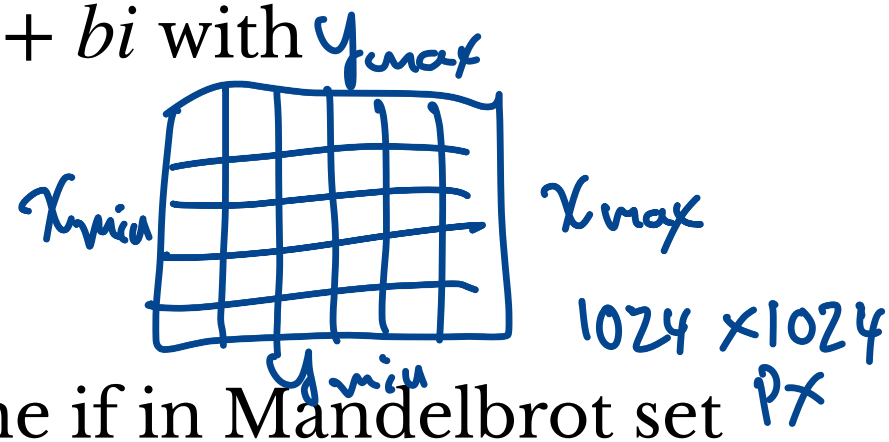
- compute $z_1 = c, z_2 = z_1^2 + c, \ldots$ until
  1. $|z_n| \geq M$
     - stop because sequence appears unbounded
  2. $N$th iteration
     - stop because sequence appears bounded
- if $N$th iteration reached $c$ is likely in Mandelbrot set

# Illustration

https://complex-analysis.com/content/mandelbrot_set.html

# Drawing the Mandelbrot Set

- Choose a region consisting of $a + bi$ with
  - $x_{min} \leq a \leq x_{max}$
  - $y_{min} \leq b \leq y_{max}$
- Make a grid in the region
- For each point in grid, determine if in Mandelbrot set
- Color accordingly



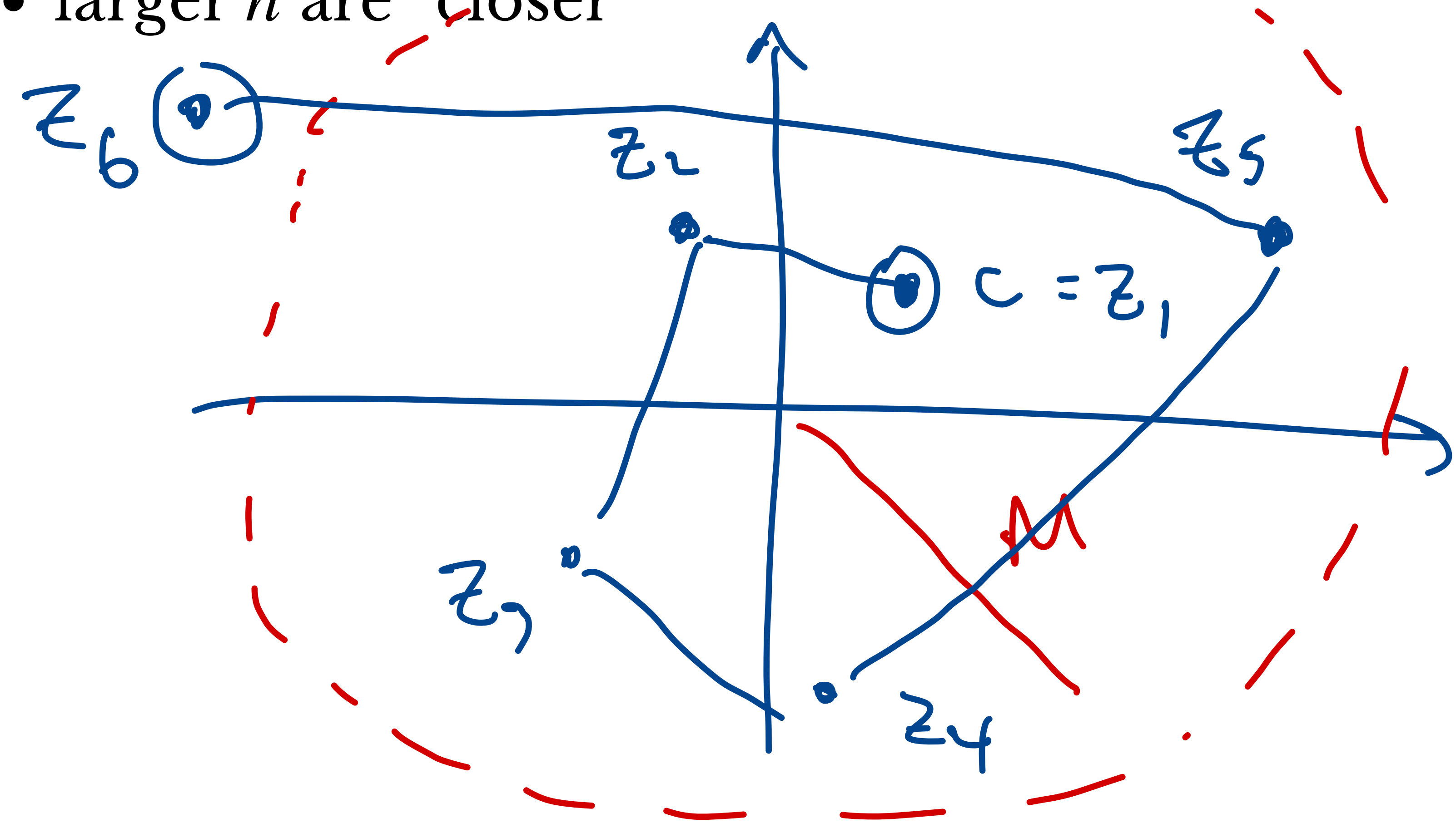
# Counting Iterations

Given a complex number $c$:

- compute $z_1 = c, z_2 = z_1^2 + c, \ldots$ until
  1. $|z_n| \geq M$
     - stop because sequence appears unbounded
  2. $N$th iteration
     - stop because sequence appears bounded
- if $N$th iteration reached $c$ is likely in Mandelbrot set

# Color by Escape Time

1. Color black in case 2 (point is in Mandelbrot set)
2. Change color based on *n* in case 1:
    - smaller *n* are "farther" from Mandelbrot set
    - larger *n* are "closer"



# Lab 03

**Input:**

- A square region of complex plane

**Output:**

- Escape times for a grid of points in the region
- A picture of corresponding region

**Goal:**

- Compute escape times as quickly as possible

# Mandelbrot Viewer Demo

- `mandelbrot.zip`

# Getting a Single Escape Time

```
public static float getValue (ComplexNumber c) {
    ComplexNumber z = new ComplexNumber(0, 0);
    int iter = 0;
    while (iter < MAX_ITER && z.modulus() <= MAX_MODULUS) {
        z = z.times(z).plus(c);
        iter++;
    }
    return (float) (MAX_ITER - iter) / MAX_ITER;
}
```

# Getting Many Values

```
private void updateBitmap () {
    for (int i = 0; i < BOX_WIDTH; i++) {
        for (int j = 0; j < BOX_HEIGHT; j++) {
            ComplexNumber c = getValueFromIndices(i, j);
            float val = Mandelbrot.getValue(c);
            bitmap[i][j] = colorMap(val);
        }
    }
}
```

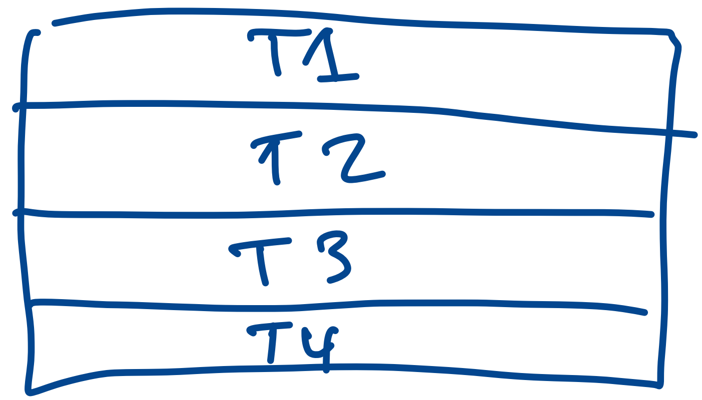
~ 1000

~ 1M iter.

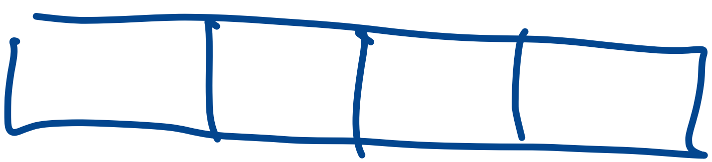
# Ideas for Improving Performance?

- multithreading : 1 px ⇝ 1 thread

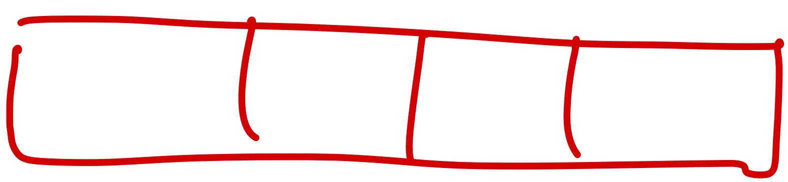
"thread pools" ?




- SIMD ops — Vector API



arithmetic ~ mask incomplete tasks



# Thread pools

# So Far

- One thread per task
- Created `Threads` and ran them in parallel
  - implmenet `Runnable` interface
  - create and `start` instances
  - `join` to wait until threads finish

# Drawbacks

- Creating new `Threads` has significant overhead
  - best performance by balancing number of threads/processors available
- Need to explicitly partition into relatively few pieces
  - partitioning may be unnatural
  - partition may be unbalanced:
    - don't know in advance how long computations will take

When tasks are fairly homogenous (e.g., computing $\pi$, shortcuts) previous approach is good

# A (Sometimes) Better Way

A nice Java feature: **thread pools**

- Create a (relatively small) pool of threads
- Assign tasks to the pool
- Available threads process tasks
  - if all threads occupied, tasks stored in a queue
  - as threads are completed, threads in pool are reused

# When are Thread Pools Better?

- Many smaller tasks
- Fixed partition of problem may be unbalanced
- “Online” problems: set of tasks not known in advance
  - e.g., processing requests for web server

# Thread Pools in Java

- Implement `Executor` interface
  - `void execute(Runnable command)` method
- More control of task handling: `ExecutorService` interface:
  - submit tasks
  - wait for tasks to complete
  - shut down pool (don't accept new tasks)

# Built-in `ExecutorService` Implementations

From `java.util.concurrent.Executors`:

- `newFixedThreadPool(int nThreads)`
  - make a pool with a fixed number of threads
- `newSingleThreadExecutor()`
  - make a pool with a single thread
- `newCachedThreadPool()`
  - make pool that creates new threads as needed (reuses old if available)
- …

# Using Thread Pools 1

## Define tasks

```
public class MyTask implements Runnable {
    ...
    public void run () {
        ...
    }
}
```

# Using Thread Pools 2

Create a pool, e.g., fixed thread pool

```
int nThreads = ...;

ExecutorService pool = Exercutors.newFixedThreadPool(nThreads);
```

Create and execute tasks

```
MyTask task = new MyTask(...);

pool.execute(task);
```

# Using Thread Pools 3

Shutting down the pool

```
pool.shutdown();
```

Wait for all pending processes to complete (like `join()` method)

```
try {

    pool.awaitTermination(Long.MAX_VALUE, TimeUnit.NANOSECONDS);

} catch (InterruptedException e) {

    // do nothing

}
```

# Example

## Shortcuts from Lab 02:

```
for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
        float min = Float.MAX_VALUE;
            for (int k = 0; k < size; ++k) {
            float x = matrix[i][k]; float y = matrix[k][j];
            float z = x + y;
            if (z < min)
                min = z;
        }
        shortcuts[i][j] = min;
    }
}
```

# A *Small* Task

For fixed row `i`, col `j`:

```
float min = Float.MAX_VALUE;
    for (int k = 0; k < size; ++k) {
    float x = matrix[i][k]; float y = matrix[k][j];
    float z = x + y;
    if (z < min)
        min = z;
}
shortcuts[i][j] = min;
```

# Two Approaches

Approach 1:

- Make a separate thread for each task
  - need `size * size` threads

Approach 2:

- Make a thread pool and let the pool decide
  - choose pool size from `availableProcessors()`

# Demo

- `executer-shortcuts.zip`