Parallel Computing

POS6933: Computational Social Science

Jake S. Truscott, Ph.D

University of Florida
Spring 2026

Overview

- Class 2 Problem Set Review
- **Today's Goal**: Learn Skills to Improve Computational Efficiency w/ Parallel Computing
- Serial Tasks
- Parallel Tasks

General Notes Re: Class 2 Problem Set

Why Ratatouille is Unrealistic

1. Movie takes place in France but two main characters suspiciously don't have French accents?

Why Ratatouille is Unrealistic

1. Movie takes place in France but two main characters suspiciously don't have French accents?
2. Rats don't have true opposable thumbs and they can't vomit

Why Ratatouille is Unrealistic

1. Movie takes place in France but two main characters suspiciously don't have French accents?
2. Rats don't have true opposable thumbs and they can't vomit
3. Gusteau's drop from a "five-star" restaurant presumably references its Michelin rating – which actually caps at three stars

Why Ratatouille is Unrealistic

1. Movie takes place in France but two main characters suspiciously don't have French accents?
2. Rats don't have true opposable thumbs and they can't vomit
3. Gusteau's drop from a "five-star" restaurant presumably references its Michelin rating – which actually caps at three stars
4. Rats are sentient, not sapient – they can't cook food.

Why Ratatouille is Unrealistic (Cont.)

- The film's *catharsis* (emotional climax) is the scene where Remy cooks Ratatouille for the food critic, Anton Ego.

Why Ratatouille is Unrealistic (Cont.)

- The film's *catharsis* (emotional climax) is the scene where Remy cooks Ratatouille for the food critic, Anton Ego.
- Ego loves the food so much he's transported back to his childhood in the French countryside and lovingly remembers his mother's cooking.

Why Ratatouille is Unrealistic (Cont.)

- The film's *catharsis* (emotional climax) is the scene where Remy cooks Ratatouille for the food critic, Anton Ego.
- Ego loves the food so much he's transported back to his childhood in the French countryside and lovingly remembers his mother's cooking.
- Awesome – but problematic for many reasons.

Why Ratatouille is Unrealistic (Cont.)

1. The most straightforward recipe for the classical Provencal dish I could fine was from *The New York Times*.

Why Ratatouille is Unrealistic (Cont.)

1. The most straightforward recipe for the classical Provencal dish I could fine was from *The New York Times*.
2. Recipe requires *at minimum* three hours to arrange and cook, while also assuming all 12 ingredients have even been properly cut.

Why Ratatouille is Unrealistic (Cont.)

1. The most straightforward recipe for the classical Provencal dish I could fine was from *The New York Times*.
2. Recipe requires *at minimum* three hours to arrange and cook, while also assuming all 12 ingredients have even been properly cut.

   So, in less than 5 minutes of screen time, Remy secures the kitchen and (by himself) prepares, cooks, and plates the dish, and is able to have this cathartic experience about believing in yourself – **I don't buy it**

Why Ratatouille is Unrealistic (Cont.)

There's just no way Remy – by himself – could have prepared this dish in a reasonable amount of time

Why Ratatouille is Unrealistic (Cont.)

- **Solution**: Divide the labor.
- Steps:
  1. Onion: 5 min
  2. Tomato: 2 min
  3. Zucchini: 4 min
  4. Eggplant: 3 min
  5. Red Peppers: 5 min
  6. Garlic, Herbs, & Spices: 5 min
  7. Arrange: 5 min
  8. Cook: 60 min
- **Total Prep Time**: 24 minutes

Dividing the Labor (Cont.)

Remy **Linguini** **Colette** **Horst**

What's the optimal way to allocate across the (4) chefs? Discuss with a classmate

Dividing the Labor (Cont.)

Remy	**Linguini**	**Colette**	**Horst**
Onion (5 min)	Garlic, etc. (5)	Tomato (2 min) Eggplant (3 min)	Red Peppers (5 min)

Suddenly the prep time is only (5) minutes! Even adding time to *arrange*, we're still saving more than half the preparation time versus Remy doing it alone!

Model T

- Ford's Model T wasn't the first car ever built – Probably wasn't even remotely close to the "best" car available when released in 1908.
- But it was **cheap** – Easily accessible to everyday earners.
- By 1920 the Model T costed $300 – $4,800 today. **How could Ford sell so cheap?**

Model T

- Ford's ingenuity came from optimization of the factory line
- Rather than one individual (or a group) building the car from start to finish, pieces would be assembled in parallel and progressively combined into a single product at the end.
- Each worker/group would be responsible for their component (*task*), not the car as a whole.
- Less specialized workforce + faster construction time = cheaper sales price = more purchases

Putting it Together

- An optimal allocation of resources towards completing tasks often yields the quickest completion time.

Putting it Together

- An optimal allocation of resources towards completing tasks often yields the quickest completion time.
- In essence – completing some tasks is better served with a lot of workers doing a little bit of work, rather than a few workers doing a lot of work

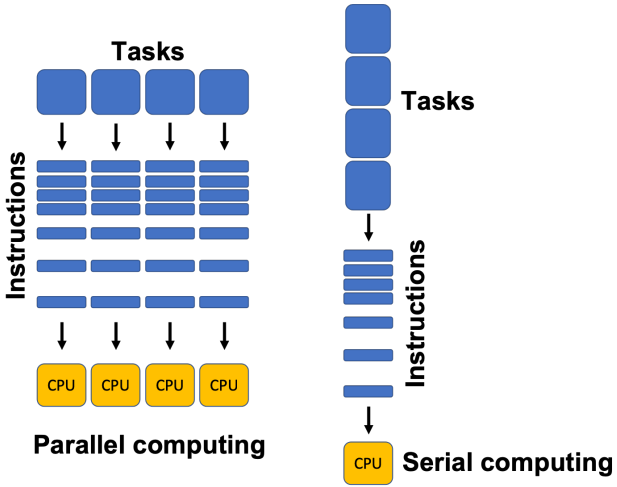
Putting it Together

- An optimal allocation of resources towards completing tasks often yields the quickest completion time.
- In essence – completing some tasks is better served with a lot of workers doing a little bit of work, rather than a few workers doing a lot of work
- We're going to clarify this in a bit but the premise serves to explain the difference between **Serial** and **Parallel** computing

The Difference

- **Serial**: A computer program's instructions are executed one at a time, in a specific, linear order, using a single processor (*Traditional/Default Computing Method*)
- **Parallel**: The practice of dividing a large computational task into smaller parts that can be executed simultaneously

The Difference (Illustrated)



Terminology

- Computers default to minimal resource allocation – i.e., it avoids trying to use resources it doesn't need to b/c it clogs memory and can otherwise slow processes.

Terminology

- Computers default to minimal resource allocation – i.e., it avoids trying to use resources it doesn't need to b/c it clogs memory and can otherwise slow processes.
- However, given sufficient resources, we spread out large & burdensome tasks – theoretically reducing the compilation time by a factor of those allocated resources.

Terminology

- Computers default to minimal resource allocation – i.e., it avoids trying to use resources it doesn't need to b/c it clogs memory and can otherwise slow processes.
- However, given sufficient resources, we spread out large & burdensome tasks – theoretically reducing the compilation time by a factor of those allocated resources.
- We just need to tell a computing system to do this!

Terminology

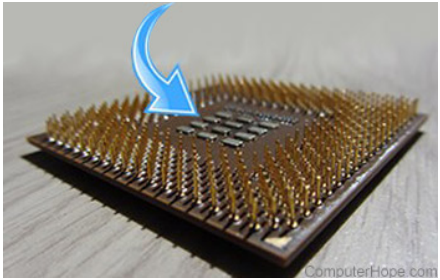
- Resources to Consider:
  1. CPU (and Processor)
  2. Thread
  3. RAM
  4. Nodes
  5. Cluster

CPU (and Processor)

- **CPU**: The "brain" of a computer that executes instructions for tasks on a computing system.
- Term generally used interchangeably with processor – refers to the piece of hardware on a computer motherboard that can contain multiple CPUs or Cores (*More Next*)
- In our Ratatoullie example, this is the kitchen – containing multiple stations for sauces, vegetables/starches, pastries, etc.

Core

- **Core**: A single processing unit within a CPU that can execute tasks independently
- In our kitchen example, cores are the number of **kitchen stoves** – each only capable of cooking one dish at a time.



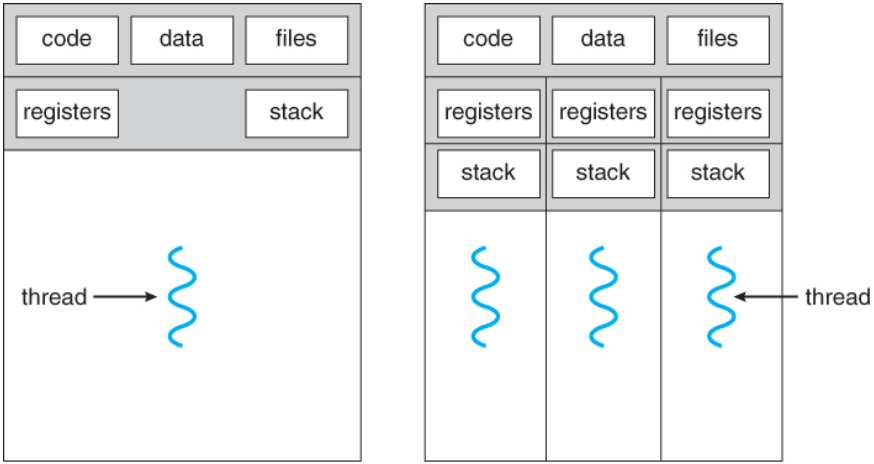
Thread

- **Thread**: A single sequence of instructions within a process that can run independently.
  - Knowing what is required, each chef will work in coordination with the others to synchronously produce their dishes.

- In our kitchen example, threads ideally represent **both arms of a chef**. Two hands are obviously more ideal than one, but that doesn't detract from the reality that they're still standing at one stove.

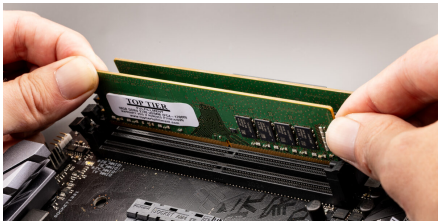
- Ex: A 16-core, 32-thread CPU usually means 16 physical cores on the CPU, each with 2 hardware threads.
  - **Key takeaway**: 32 threads does not mean 32 stoves. The extra thread mainly helps prevent idle time – e.g., think of it as the chef using their second arm to prep the next dish while the first one is cooking, so the stove is never waiting. That improves efficiency but doesn't detract from reality there's still only one stove.

Thread (Cont.)


code
data
files
registers
stack
thread
code
data
files
registers
registers
registers
stack
stack
stack
thread
single-threaded process
multithreaded process

RAM

- **RAM** (Random Access Memory): Where the computer keeps data and instructions that the CPU is actively using.
- In our kitchen example, we can think of RAM as the ingredients or recipe that the chef will want to have handy so they can prevent delays.
- *Note*: If the recipe isn't in RAM (or the ingredients aren't ready), the chef has to wait while it's fetched from the pantry (storage like a hard drive or SSD), which is much slower.



HiPerGator

- **HiPerGator** is UF's **high-performance computing (HPC)** system.
- Provides access to **thousands of CPU cores, GPUs, and large-scale storage**.
- Designed for **computationally intensive tasks** such as simulations, machine learning, and big data analysis.
- Enables research that is **too large, slow, or complex** to run on a personal computer.

Running Paralell in R

```
suppressPackageStartupMessages({
    library(parallel)
    library(doParallel)
})  # Load Parallel (Quietly)

numcores <- parallel::detectCores()
message("You Have ", numcores, " Cores Available for Use!")
```

```
You Have 14 Cores Available for Use!
```

Create Socket Cluster

- Here, we are creating *Workers* – independent `R` processes that preserves (ideally) *at minimum* a 1:1 ratio with the number of cores available.
- Your OS will decide how to best allocate resources across cores,

```
cl <- makeCluster(numcores - 1)
doParallel::registerDoParallel(cl)  # Register Socket
stopCluster(cl)  # Relieves Cluster
```

Benchmarking

```
tedious_function <- function(x) {
    y <- rnorm(1e+06)
    mean(y^2)
}

cl <- makeCluster(numcores - 1)
doParallel::registerDoParallel(cl)

serial <- system.time({
    serial_run <- lapply(1:numcores, tedious_function)
})  # Serial Run (numcores times)


parallel <- system.time({
    parallel_run <- parLapply(cl, 1:10, tedious_function)
})  # Parallel Run (numcores times)

stopCluster(cl)  # Shut Doum Parallel
```

Benchmarking (Cont.)

```
print(serial[["elapsed"]])
```

```
[1] 0.56
```

```
print(parallel[["elapsed"]])
```

```
[1] 0.06
```

Exporting Packages & Resources to Socket

- Using `cl()` does not automatically make packages loaded via *library()* or objects/functions created in the global environment available on worker nodes.
- As with socket clusters, these packages and objects must be explicitly exported to each cluster node.
- We can do that using `clusterExport()` and `clusterEvalQ()`

Exporting Packages & Resources to Socket (Cont.)

- Below is an example from one of my papers re: simulating departure behaviors of federal judges.

```
clusterExport(cl, c("predict_survival", "new_judge",
    "election_simulation", "senate_moratorium", "senate_rejection",
    "single_simulation_analysis", "electoral_outcomes_02_24",
    "departure_simulation_comparison", "d1", "c1",
    "fjc_combined_survival", "allocated_seats", "nominate",
    "senate_rejection_data"))

clusterEvalQ(cl, {
    library(dplyr)
    library(survival)
    library(stringr)
})  # Allocate Necessary Packages
```

Next Class

- Problem Set 3 due Sunday (11:59pm)
- Next Class: Intro to Text Analysis