Programming Camp - High Performance Computing

Marcelo Sena

This version: September 23, 2024

Thanks to Brian Higgins, Ciaran Rogers and Diego Jimenez for sharing material.

Presentation outline

1. High Performance Computing
2. Parallel Computing
3. AI, LLMs and Coding

1 | High Performance Computing

Parallel Computing

AI, LLMs and Coding

Why do we need high performance

1. Quantitative, data-driven economic models are computationally complex
   - Finding equilibrium involves solving individual behavior and then iterating for many prices until markets clear;
   - Estimation requires solving model with many parameters guesses;
   - Games with many players; industries with strategic behavior;
   - Auctions with many goods;
   - Migration with many cities;
2. Big data can easily overwhelm our laptops
   - e.g. product level databases
   - e.g. credit card transaction data

Why do we need high performance

1. Quantitative, data-driven economic models are computationally complex
   - Finding equilibrium involves solving individual behavior and then iterating for many prices until markets clear;
   - Estimation requires solving model with many parameters guesses;
   - Games with many players; industries with strategic behavior;
   - Auctions with many goods;
   - Migration with many cities;
2. Big data can easily overwhelm our laptops
   - e.g. product level databases
   - e.g. credit card transaction data

→ A mix of hardware and software can push the boundary of what is feasible
   - Servers, with CPUs, GPUs, big RAM;
   - Parallel programming;
   - Distributed computing (many computers)

· Today: Show you the basics, and highlight what's out there

· Lots to learn from CS
   - Machine learning tools – e.g. PyTorch & GPU – can be leveraged for scientific computing
   - Worth investing at this point in PhD

GPUs are changing machine learning and computer science

NVIDIA share price

1 High Performance Computing

What type of computer cluster should you use?

- Paid services include Microsoft Azure, Amazon Web Services (AWS) and Google Cloud Platform.

	Paid Services	**University Cluster**
Monetary cost	[✗] Expensive, often paid by the hour	[✓] Free to students and faculty
Waiting times	[✓] Instantly available	[✗] Access often involves queues and waiting times
Ease of use	[✓] Low set up costs (e.g., turning on a computer + installing software)	[✗] Complicated to understand and use (high fixed costs)

1 High Performance Computing

1.1 University Clusters vs. Paid Services

1.2 Using Stanford Servers

Some Terminology

- **Computer cluster:**
  Collection of separate computer servers, called nodes, which are connected via a fast interconnect.
  Configuration allows many computers to work together.

- **Nodes:**
  Individual computers designed to accomplish specific tasks (e.g., login nodes, computing nodes).

- **Job Scheduler:** (SLURM)
  Software that organizes and assigns tasks to the computer cluster.
  Provides a framework to start, execute and monitor jobs within the cluster.

List of Stanford Servers

1. Farmshare: `rice` (login node), `wheat` (big memory node), and `oat` (gpu node).
   - Coursework, and non-sponsored research.
   - Can only connect to `rice`, and access other nodes from there.

2. Sherlock
   - Sponsored research. Need a faculty sponsor to get access to.
   - Humanities and Sciences partition available `hns`.
   - Connect to login nodes (low computing power) and schedule jobs in high performance nodes.

3. Other servers available tailored to other needs.
   - GSB server: `yen`.

Using the terminal

- Connect through SecureCRT (Windows) or the terminal (Mac).

- Connecting to Stanford servers:

```
ssh StanfordID@rice.stanford.edu
ssh StanfordID@login.sherlock.stanford.edu
```

- Some useful commands:

`pwd`	list current working directory
`cd`	change directory
`touch file.txt`	create a new file called file.txt
`vi file.txt`	open text editor in terminal
`module avail` or `ml avail`	show available modules
`module load X` or `ml X`	load module X into workspace
`module list`	list loaded modules

Using the terminal

Text Editors

- when you open the file in the server's terminal, you will be using the Vim text editor (https://www.vim.org/)
- vim is a very powerful text editor, but with relatively high entrance costs
- many modes of editing (e.g. inserting text is one of them)
  - other modes allow creative text manipulation through "motions" (e.g. delete this text until you find a closing parenthesis)
- a good starter is Vim's own tutorial: type `vimtutor` in the terminal
- the first couple of times can feel slow and frustrating, but with practice you will hardly want to use anything else
- NeoVim (https://neovim.io/) is an alternative option (a fork/rebuild of Vim with a more modern interface)
- other nice text editors (more user friendly) include: Nano, Emacs, and Sublime Text

Transferring files to/from servers

- **Windows:**
  - Use SecureCRT + SecureFX (paid software, free for Stanford students). More information click here.
- **Mac/Linux:**
  - Use Cyberduck (free) or other paid software (e.g., Transmit).
  - Through the terminal also works: see `scp`.
- **Farmshare:**
  - Transfer through AFS: `http://afs.stanford.edu`
  - If working in batch jobs, store in `/farmshare/user_data/SUNetID`
- **Sherlock:**
  - Sherlock is pretty flexible. Once you need more information click here.

Steps:

Say that I have a `code.sbatch` ready to run.

1. Open Fetch / SecureFX to start your connection:
   Hostname: `rice.stanford.edu`
   Username: `<SUNet ID>`
   Password: `<SUNet Password>`
2. Go to `/farmshare/user_data/<SUNet ID>`.
3. Create a `test_data` folder and copy your code files there.
4. Start your ssh connection. Open the terminal and write:
   `ssh <SUNet ID>@rice.stanford.edu`
5. From the terminal, change to the Farmshare directory.
   `cd /farmshare/user_data/<SUNet ID>/test_data`
6. Run the code on the server.
   `sbatch test_batch.sbatch` or `srun test_batch.sbatch`.
7. Check your status with `squeue -u $USER`.

Other SLURM useful commands

- Submit jobs to specific partitions:
  `#SBATCH -p hns,normal`
  (other: `bigmem, gpu, long`).

- Request more / less time than default:
  `#SBATCH --time=01:00:00`
  (less time may make your code start faster).

- Request a specific amount of RAM:
  `#SBATCH --mem=128000`
  (note: not all memory - core combinations are available).

- Switch from `rice` node to a `wheat` node:
  `srun --qos=interactive --pty /bin/bash -l`

High Performance Computing

2 Parallel Computing

AI, LLMs and Coding

Why is some code faster than others?

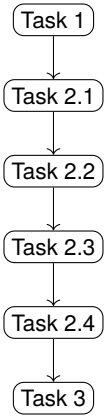
- C is typically a fast language because it is a compiled language: it compiles your written code direct to machine readable code
  - Costs more in developer time though!
- Most languages convert your code to some intermediary form which in turn converts to machine code (these are called interpreted languages)
  - e.g. Matlab, Stata, Python
  - but easier to code and debug!
- (Julia is a compiled language, reason why it can be so competitive in terms of speed).
- Built-in packages are highly optimized
  - $\beta = \mathbf{X}'\mathbf{X}^{-1}\mathbf{X}'\mathbf{Y}$ runs close to C

Parallel Computing

- Custom algorithms – like value functions w/loops — can be slow though
- Options to go faster
  1. Eliminate loops, use vectorized code
  2. Write inner loop in C
     - Matlab mex files
  3. Just-in-time compilers
     - Julia; Python-Numba, @njit
     - Run slower first time, then much faster
  4. Parallelization Today!
  5. Really parallel: GPU, multi-GPU nodes
     - Python: cupy, numba, dask, pytorch; Julia...
     - Lots of digestible material from CS

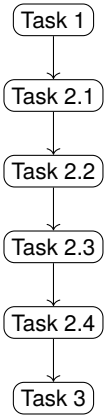
Intuition for parallel computing

We are used to running code like this:

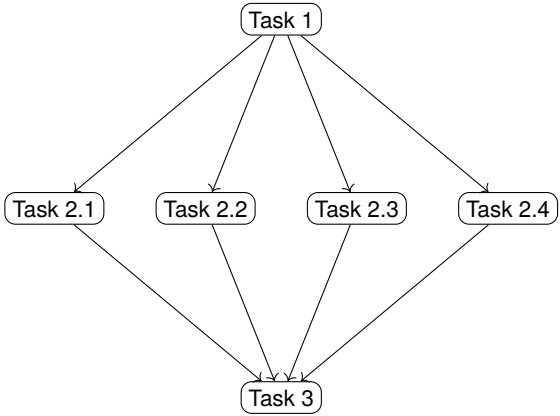


Intuition for parallel computing

We are used to running code like this:

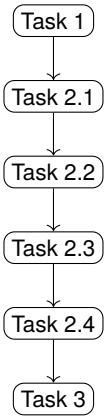


But we could do something else instead:

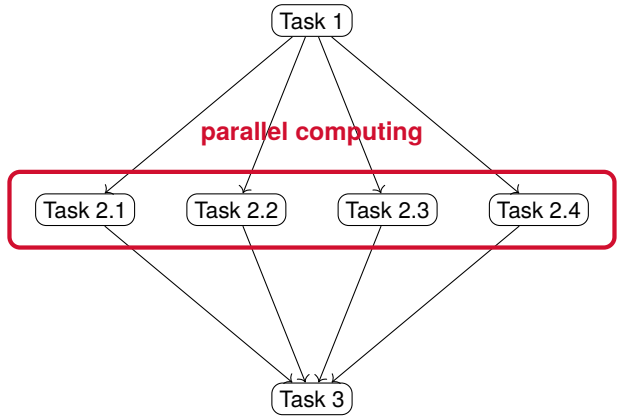


Intuition for parallel computing

We are used to running code like this:



But we could do something else instead:



Gains of going parallel

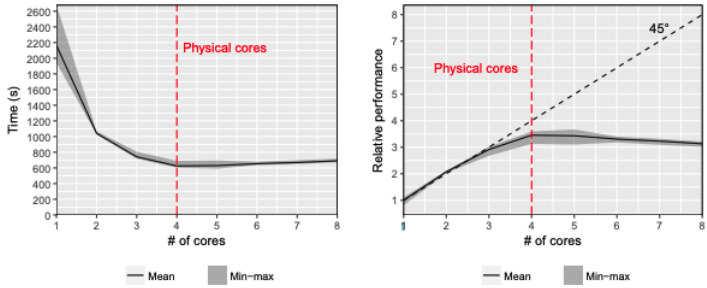
- Does our runtime increase linearly with the number of parallel processes?

NO!

- Collecting and organizing output from different “workers" also requires time
  - some packages streamlines this process for the user, see e.g. Pytorch
- Problems can be divided into
  1. scalability: how effectively we can divide the original problem into smaller tasks
  2. granularity: the amount of work required in each subtask
- A problem is well suited for parallelization if it is scalable and coarse (the latter meaning it requires more computation per subtask then communication across them)

Gains of going parallel

Moreover, bottleneck of algorithm could be elsewhere.



(a) Computing time (s)

(b) Performance gains.

Figure 12: Results in `Python` with different number of processors. Number of experiments: 20.

Figure: Solving a life-cycle problem in Python, Fernandez-Villaverde and Valencia (2018)

“Off-the-shelf” Parallelization on the CPU

- Numba allows certain for loops to be automatically parallelized by simply adding a decorator to the desired code.
- Example:

```
from numba import prange

start = time.time()
@numba.jit(nopython=True, parallel=True)
def monte_carlo_pi(nsamples):
    acc = 0
    for i in prange(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples

print(monte_carlo_pi(10_000_000))
end = time.time()
print(end - start)
```

“Off-the-shelf" Parallelization on the GPU

- If we want to run code on the GPU, one package to leverage on is `cupy`
- In a similar spirit to `Numba`, it translates `numpy` code to something GPUs can also run
- In some cases, it can be as easy as changing `np` to `cp`
- In others, it may require some re-thinking and vectorization
- Example:

```
import cupy as cp
import time

start = time.time()
def monte_carlo_pi(nsamples):
    acc = 0
    x = cp.random.rand(nsamples)
    y = cp.random.rand(nsamples)
    x = cp.power(x,2)
    y = cp.power(y,2)
    acc = cp.sum(cp.less(x + y,1.0))
    return 4.0 * acc / nsamples

print(monte_carlo_pi(10_000_000))
end = time.time()
print(end - start)
```

Writing your kernel

- We will discuss here one step further: implementing the GPU code ourselves
- What is a CUDA kernel? elementwise function to run on the “device” (aka the GPU)
- The host (aka the CPU) sends blocks to be run in parallel on GPU
- Technically, we would have to write in C CUDA (language for NVIDIA GPUS).
  - For other types of GPU’s (eg AMD), we would have to write in a different language.
  - Fortunately, there are also similar tools for them (Numba itself).
- Now “easy” to write with Numba in Python (and Julia, matlab options) thanks to “translators”.
  - typical complication: GPU memory is separate, need to initialize/end to GPU
  - have to be careful with indexing

Writing your kernel

Example of a kernel

```
from __future__ import division
from numba import cuda
import numpy
import math
# CUDA kernel
@cuda.jit
def matmul(A, B, C):
    """ Perform matrix multiplication of C = A * B """
    row, col = cuda.grid(2)
    if row < C.shape[0] and col < C.shape[1]:
        tmp = 0.
        for k in range(A.shape[1]):
            tmp += A[row, k] * B[k, col]
        C[row, col] = tmp
```

Writing your kernel

This is the host code, your Python file that will call your kernel.

```
A = numpy.full((24, 12), 3, numpy.float) # matrix containing all 3's
B = numpy.full((12, 22), 4, numpy.float) # matrix containing all 4's
# Copy the arrays to the device
A_global_mem = cuda.to_device(A)
B_global_mem = cuda.to_device(B)
# Allocate memory on the device for the result
C_global_mem = cuda.device_array((24, 22))
# Configure the blocks
threadsperblock = (16, 16)
blockspergrid_x = int(math.ceil(A.shape[0] / threadsperblock[0]))
blockspergrid_y = int(math.ceil(B.shape[1] / threadsperblock[1]))
blockspergrid = (blockspergrid_x, blockspergrid_y)

# Start the kernel
matmul[blockspergrid, threadsperblock](A_global_mem, B_global_mem, C_global_mem)
# Copy the result back to the host
C = C_global_mem.copy_to_host()
print(C)
```

Challenge Exercise

- Write down a CUDA kernel for our monte-carlo estimation of π

Running GPU code on the server

To run GPU code on the server, the batch file should require the proper resources, as in example below

```
#!/bin/bash
#SBATCH -p gpu
#SBATCH -c 10
#SBATCH -G 1

cd change_to_appropriate_directory
module load relevant_softwares_and_packages
python3 run_your_code.py
```

Alternatively, you can also ask for an interactive session with GPUs (useful for debugging)

```
srun -pty -partition=gpu -gres=gpu:1 -qos=interactive $SHELL -l
```

In Farmshare this is less useful since you may have to wait a long time to access a GPU compatible node

High Performance Computing

Parallel Computing

3 | AI, LLMs and Coding

LLMs

- LLMs are Large Language Models
- In a nutshell, they are parametrized distributions of text based on some prior context
  - aka stochastic parrots
- Main use cases for economists? (as of now...)
  - coding
  - writing
  - summarizing text
- Famous LLMs out there: GPT, Llama, Bard (GPT and Llama have much better performance than Bard)

Chat GPT

Python API

- Install package `openai`
- Obtain API Key from their website

```
import openai
openai.api_key = "sk..."
models = openai.Model.list()

# print the first model's id
print(models.data[0].id)

chat_completion = openai.ChatCompletion.create(model="gpt-3.5-turbo",
                    messages=[{"role": "user", "content": "Hello world"}],
                    temperature=0.0)
```

Requires paid plan of Open AI =/

Chat GPT

Prompt Tips

Based on Andrej Karpathy's talk at Tesla

- Chain of thought
  - give examples
  - "think step by step"
- sample multiple attempts
  - prompt again
  - remove stochasticitiy
  - ask the LLM to reflect
- condition on good performance (LLMs original objectives are not yours!)
  - "you are a leading expert in this topic"/"you have IQ 120" (careful with too much though)
  - allows the LLM to place less probabilistic mass in low quality solutions
  - tell the LLM what they are not good at ("use a calculator")

Note however that prompt engineering faces similar developing-vs-run time trade-off

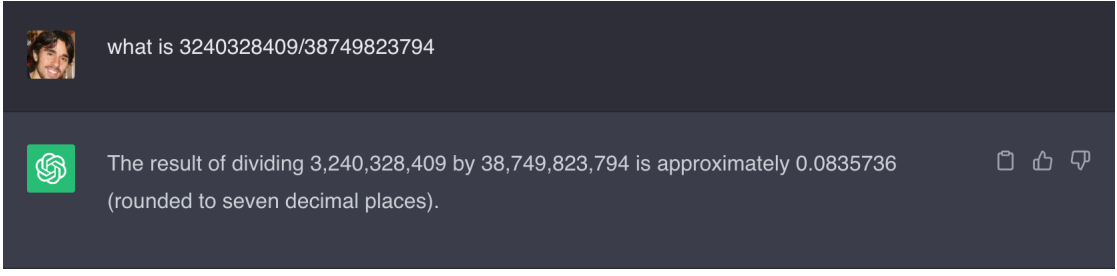
Github Co-Pilot/Chat

- If you want to speed up your coding routine, I strongly recommend Github Co-Pilot (https://github.com/features/copilot).
- It is an AI based on state-of-the-art natural language processing models that helps you complete code
- What is amazing about Github Pro is that it is not just a code completion tool, but it can also write code for you. It is not perfect, but it is a great tool to speed up your coding routine.
- For example, it can write Python code for you, and it can also write CUDA kernels for you.
- I will show you a demo of it in the next slide.
- The last 3 bullets were written by Github Co-Pilot!
- Thankfully free Stanford students!

`https://www.cursor.com/` is another great option, but paid

Personal advice on using these

- Of course it helps a lot
- But: codes that run ≠ codes that are correct
  - every output of the AI will very likely run fine!
  - its output can look correct or be almost correct



- Make sure you understand its suggestion
  - use it as a co-pilot, not a pilot