

# **Process Parallelization**

Lu Liu, Kyungchan Lim


# Test Questions

---

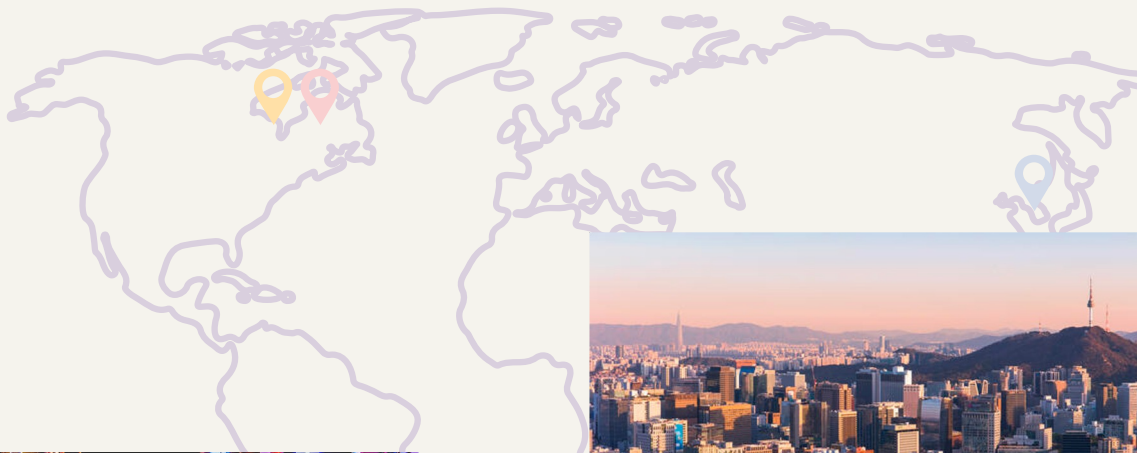
1. What is the purpose of Amdahl's Law?
2. What is the possible issue with parallel processing?
3. Which method during the implementation gave the best result?

# Kyungchan Lim




- Second year Ph.D. student
  - Advisor: Dr. Doowon Kim
  - Research Area: Data Driven Security and Measurement Study
- 

# Places where I've been



 **Seoul,  
South Korea**  
Originally from South Korea

 **Royal Oak, MI**  
Went to high school in Royal  
Oak, MI

 **Albany, NY**  
Graduated B.S. and M.S. from  
University at Albany in NY





# About Kyungchan Lim

Christian

Have one wife and two cats

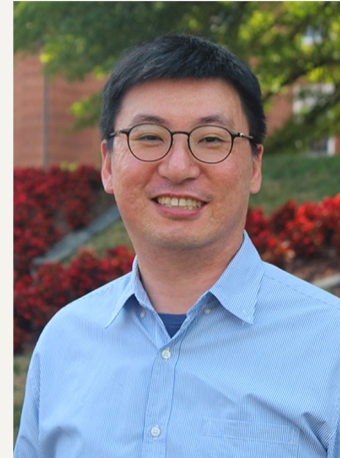
Love to eat and cook

Loved cars

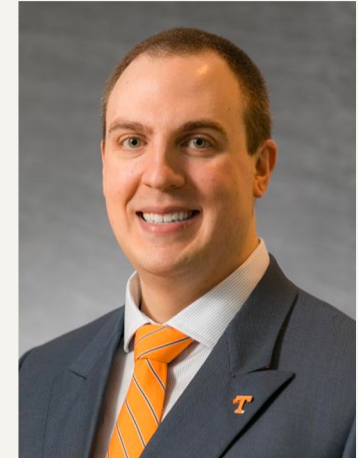


# Lu Liu

- From Denver, CO. Originally from Jinan, Shandong, China



Advised by Dr. Doowon Kim



Working for Dr. Stephen Marz as GTA

- First year Ph.D. student in Computer Science.
- Research focuses on cybersecurity and computer networks, especially data-driven and usable security.



One of the volunteers responsible for the PhD Tea Time organized by Dr. Sai Swaminathan



# Hometown - Jinan



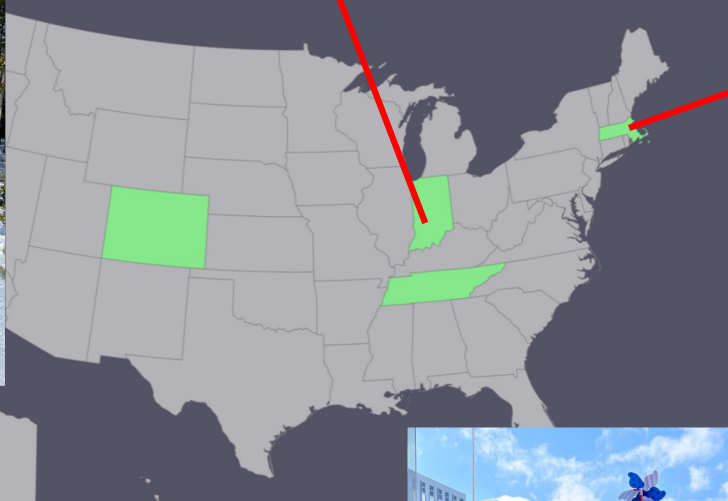
Baotu Spring

**Jinan** is the capital of Shandong province in Eastern China. With a population of 9.2 million, it is the second-largest city in Shandong.

It is also called the "City of Springs" for its famous 72 artesian springs.

Sister city: Sacramento, California (October 2, 1984)





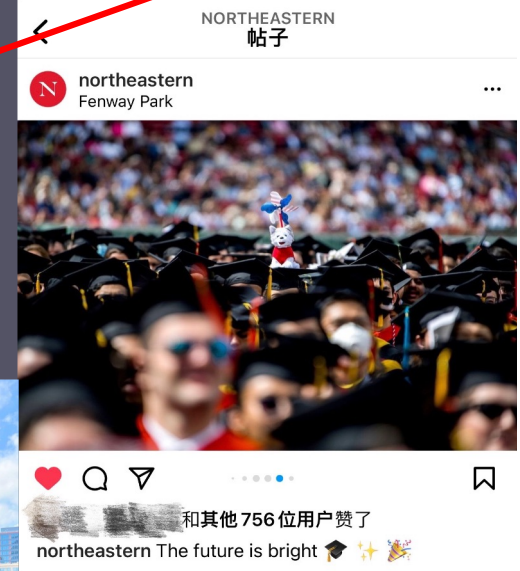
**Indiana University - B.S.  
in Business (BEPP)  
Bloomington, IN**

**Northeastern University  
- M.S in Information Systems  
Boston, MA**



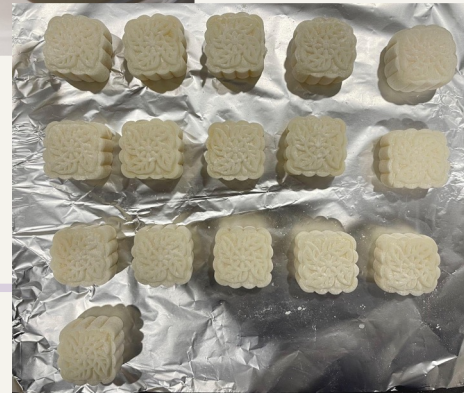
Sushi Den

- Hiking
  - Skiing
- Ikon pass holder. Favorite resorts are Copper and Aspen Snowmass
- Favorite foods: Sashimi, Unagi Don, Okonomiyaki

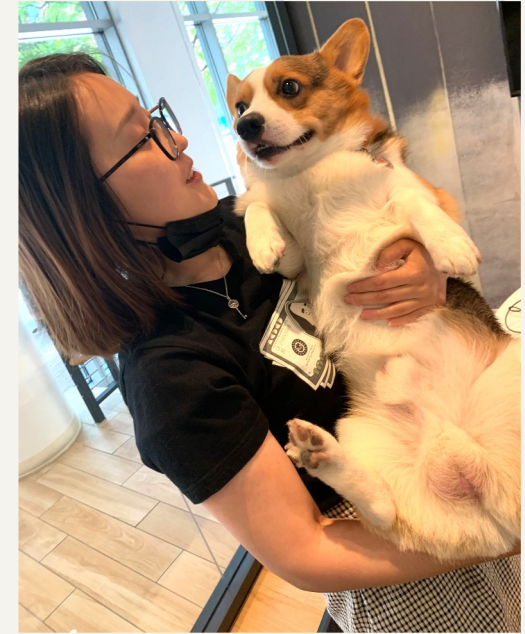




- Ice Skating
- Swimming
- Baking
- Cooking
- Cocktail







- Dog lover, particularly of the Corgi breed





- LEGO Lover
- Stand-Up Comedy
- Pop music
- Maroon 5



Jimmy O. Yang

# Outline

---

01

**Overview**

02

**History**

Form 1960s to 2020s

03

**Algorithms**

04

**Applications**

Machine Learning,  
Cryptocurrency

05

**Implementations**

Parallel Processing in Machine  
Learning

06

**Open Issues**

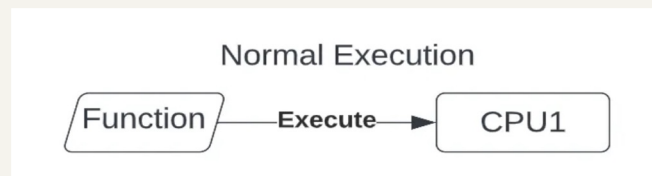


# Overview

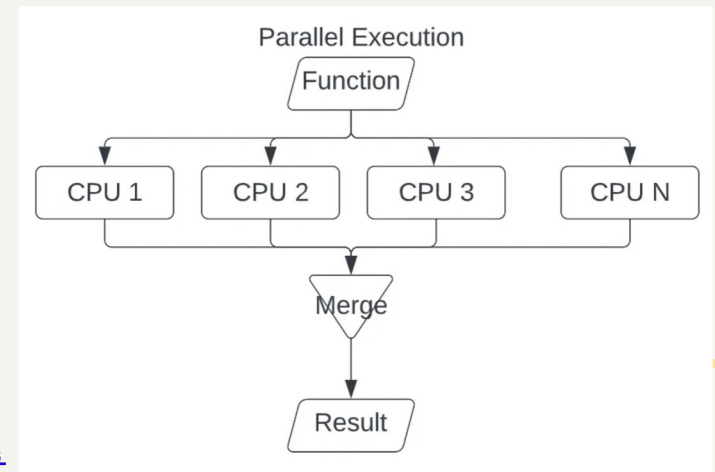
- **Parallel processing** is a computing technique when multiple streams of calculations or data processing tasks co-occur through numerous processors working concurrently from serial processors.
- **Amdahl's Law:** Predict theoretical speed up on performance improvement achievable through parallelization.
- **Vector Processing:** Performing the same operation on multiple data elements simultaneously.
- **Symmetric Multiprocessing (SMP):** Multiple processors sharing memory, working on a single task.
- **Massively Parallel Processing (MPP):** Large-scale parallel processing using thousands of processors.
- **Multicore Processors:** A single chip containing multiple processor cores.
- **Heterogeneous Computing:** Combining CPUs and GPUs for efficient parallel workloads.
- **OpenMP:** API for creating parallel programs on shared-memory architectures.

## Applications:

- High-Performance Computing (HPC)
- Graphics Processing Units (GPUs)
- Big Data
- Machine Learning
- Cryptography



Reference: <https://towardsdatascience.com/parallelization-w-multiprocessing-in-python-bd2fc234f516>



# History

## 1960s - Early ideas and research

- Researchers start exploring the idea of parallel processing to improve computer performance.

Amdahl's Law was introduced in 1967 (by Gene Amdahl)

## 1980s - Advancements and commercialization

- Connection Machine CM-1 is launched with a massively parallel architecture. (by Danny Hillis)

## 2000s - Multicore processors and GPUs

- IBM's POWER4 processor, an early commercial multicore processor, is released.

- NVIDIA's CUDA platform enables GPU parallel computing.

## 2020s - Continued development and new architectures

- Exploration of alternative approaches like neuromorphic and quantum computing.

## 1970s - First Parallel Computer

- ILLIAC IV supercomputer is completed.

- Cray-1 supercomputer introduces vector processing.

## 1990s - Multiprocessor systems and software

- Symmetric multiprocessing (SMP) systems become common.

- OpenMP API is introduced for parallel programming.

## 2010s - Heterogeneous computing and big data

- Heterogeneous computing gains traction.

- Big data and machine learning drive demand for parallel processing.

## Important Contributors



- The TOP 500 project ranks 500 most powerful non-distributed computers in the world (supercomputers)

- Since June 2022, ORNL's Frontier is top performing supercomputer

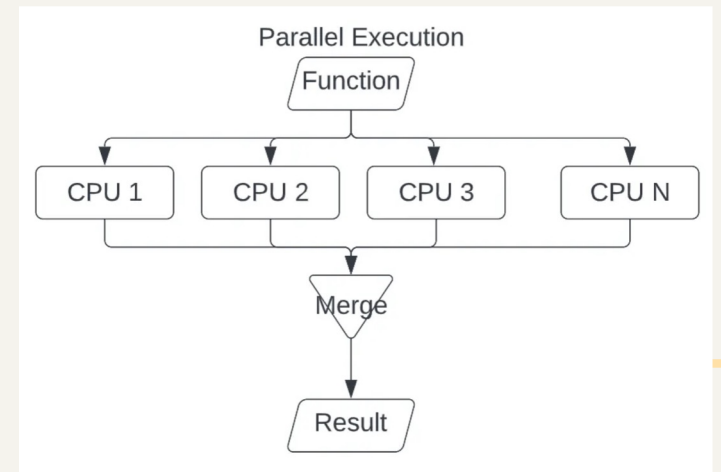
# Algorithms

- Basic idea on how Parallel processing works:
  - In a multiprocessor environment, run a serial process into multiple processes to speed up the process
- Multithreaded algorithms introduced in our textbook:
  - Multithreaded Matrix Multiplication
  - Multithreading Strassen's Method
  - Multithreaded Merge Sort
- Benefits on using Multithreaded (or Parallel) algorithms:
  - Save time on serial algorithm by processing them into parallel
- Possible Issues:
  - Mainly caused by communication between processors (e.g., Load Balancing Issues)
  - Race Condition Bug

# Algorithms

- Key steps on how parallel processing works
  - Divide parent process into multiple child process
  - Communicate between child processes to keep track which process are running
  - Load balancing between processes
- To predict theoretical speedup, Amdahl's Law can be used
  - $1/((1-P) + P/S)$
  - P is the percentage of work in parallel
  - S is number of workers

$$\frac{1}{(1 - p) + \frac{p}{s}}$$



# Algorithms (example)

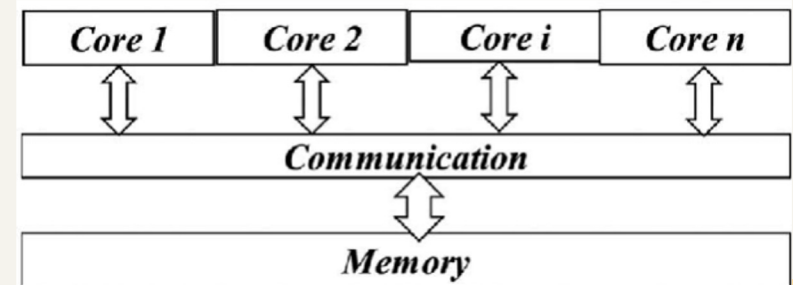
- The multiprocessing module in Python:
  - **Forking:** The multiprocessing module uses the `fork()` system call to create new processes. When a new process is created, it is an exact copy of the parent process, including all open files and resources.
  - **Process management:** The process manager maintains a list of all active processes and their state, and provides methods for starting and stopping processes, as well as for passing data between them.
  - **Interprocess communication:** The multiprocessing module uses interprocess communication with shared memory and synchronization.
  - **Parallelism:** The module provides a `Pool` class that allows the user to create a pool of worker processes, each of which can execute a target function on a separate input.
  - **Load balancing:** The multiprocessing module uses load balancing techniques to distribute tasks evenly among the available worker processes.
  - Overall, the multiprocessing module in Python is designed to provide an efficient and easy-to-use framework for managing and coordinating parallel processes. By using a combination of algorithms and techniques, it allows Python programs to take advantage of modern multi-core processors and parallel computing architectures to achieve higher performance and throughput.

# Algorithms (Benefits)

- Benefits with using parallel processing
  - Save time on serial algorithm by processing them into parallel
  - Solve more complex problems
  - Allows more resources to be used to solve problems

# Algorithms (Issues)

- Communication
  - Parallel algorithms need extra resources to communicate between processors
  - Therefore the estimated time of using two cores is not double the speed of single core
- Load Balancing Issue
  - If input size is not balanced (not even), then the work is unbalanced
  - Some processors will do more work than others
  - We are wasting processes when we have less working processors idle
- Race Condition Bug
  - If one of processors execute the code with different amount of time than expected, it can cause unexpected behavior
  - This unexpected behavior can result in security issues



# Applications

## 1. Machine Learning

### a. Training Neural Networks

- High computational demand for large datasets and complex models.
- Parallel processing accelerates training by processing data concurrently.
- Data parallelism and model parallelism

## 1. Cryptocurrency

### a. Risk calculations and cryptocurrencies in banking

- Most of today's banking processes, like credit scoring, risk modeling, are GPU-accelerated.
- One of the early adopters was JPMorgan Chase, which said in 2011 that it was switching from CPU-only processing to hybrid GPU-CPU processing. This resulted in a 40% improvement in the accuracy at its data centers in terms of risk calculations and enabled savings of 80%.

### a. Bitcoin

- The crypto-mining frenzy, a 2019-2020 financial trend, also put GPUs in the spotlight. Blockchain and Bitcoin don't work without parallel computing. In a serial computing world, the "chain" part of blockchain would evaporate.



# Implementations - Dataset

The dataset contains CSV files for specific Bitcoin exchanges from January 2012 to March 2021, updating OHLC (Open, High, Low, Close), BTC volume and indicated currency, and Bitcoin weighted prices per minute. The timestamp takes Unix time

length of data set: 4857377

	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
0	1325317920	4.39	4.39	4.39	4.39	0.455581	2.0	4.39
1	1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
2011-12-31 02:50:00	4.390	4.390	4.390	4.390	0.455581	2.000000	4.390000
2011-12-31 03:00:00	4.390	4.390	4.390	4.390	0.455581	2.000000	4.390000
2011-12-31 03:10:00	4.390	4.390	4.390	4.390	0.455581	2.000000	4.390000
2011-12-31 03:20:00	4.390	4.390	4.390	4.390	0.455581	2.000000	4.390000
2011-12-31 03:30:00	4.390	4.390	4.390	4.390	0.455581	2.000000	4.390000
...	...	...	...	...	...	...	...
2021-03-30 19:20:00	58657.741	58681.305	58648.662	58673.583	0.554792	32545.765983	58661.648079
2021-03-30 19:30:00	58685.027	58693.261	58677.739	58682.738	0.422230	24776.341924	58685.170224
2021-03-30 19:40:00	58633.291	58643.665	58627.750	58635.279	4.343502	254651.437773	58635.235213
2021-03-30 19:50:00	58705.824	58722.704	58694.762	58709.824	1.979674	116201.019012	58706.090970
2021-03-30 20:00:00	58767.750	58778.180	58755.970	58778.180	2.712831	159417.751000	58764.349363

486392 rows x 7 columns

4857377 x 8 → 486390 x 7

# Implementations - Random Forest

## 1 Initialize parameters

```
param_grid = {  
    'n_estimators': [200, 500, 800, 1000],  
    'max_features': ['auto', 'sqrt', 'log2'],  
    'max_depth' : [2,4,6,8,10],  
    'criterion' : ['gini', 'entropy']  
}
```

## 2 GridSearchCV compare different parameters, number of CPU

```
%%time  
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=10)  
CV_rfc.fit(X_train[:3000], y_train[:3000])  
CV_rfc.best_params_
```

```
CPU times: user 53min 19s, sys: 4.73 s, total: 53min 24s  
Wall time: 53min 24s  
{'criterion': 'entropy',  
 'max_depth': 4,  
 'max_features': 'auto',  
 'n_estimators': 1000}
```

```
%%time  
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=10, n_jobs=2)  
CV_rfc.fit(X_train[:3000], y_train[:3000])  
CV_rfc.best_params_
```

```
CPU times: user 9 s, sys: 396 ms, total: 9.4 s  
Wall time: 27min 49s  
{'criterion': 'entropy',  
 'max_depth': 4,  
 'max_features': 'auto',  
 'n_estimators': 1000}
```

## 3 Best parameters

```
CV_rfc.best_params_
```

```
{'criterion': 'entropy',  
 'max_depth': 4,  
 'max_features': 'auto',  
 'n_estimators': 1000}
```

```
%%time  
rfc_Best_Params.fit(X_train, y_train)
```

```
CPU times: user 8min 51s, sys: 128 ms, total: 8min 51s  
Wall time: 8min 51s  
RandomForestClassifier(criterion='entropy', max_depth=4, n_estimators=1000,  
                        random_state=42)
```

# Implementations - Parallel

Three methods:

- Multiprocessing Pool Parallel
- Joblib Parallel
- Pathos Pool Parallel

- Import packages
- Set parameters
- Define function

```
from multiprocessing import Pool
import time
import multiprocessing
from time import time
from joblib import Parallel, delayed
from multiprocessing import Process
from pathos.pools import ProcessPool
```

```
parameters = {'criterion': 'entropy',
              'max_depth': 4,
              'max_features': 'auto',
              'n_estimators': 1000}
```

```
def func(parameters):
    rfc = ensemble.RandomForestClassifier(random_state=42)
    return rfc.fit(X_train, y_train)
#return rfc1.get_params
```

# Implementations - Multiprocessing

## Pool.map

```
if __name__ == "__main__":  
  
    nums = [1,2,4,8,16]  
    for i in nums:  
        start = time()  
        pool = multiprocessing.Pool(i)  
        pool.map(func, parameters)  
        pool.close()  
  
        print('\nPool Parallelism runs %0.3f seconds on %s CPUs.\n' % ((time() - start), i))
```

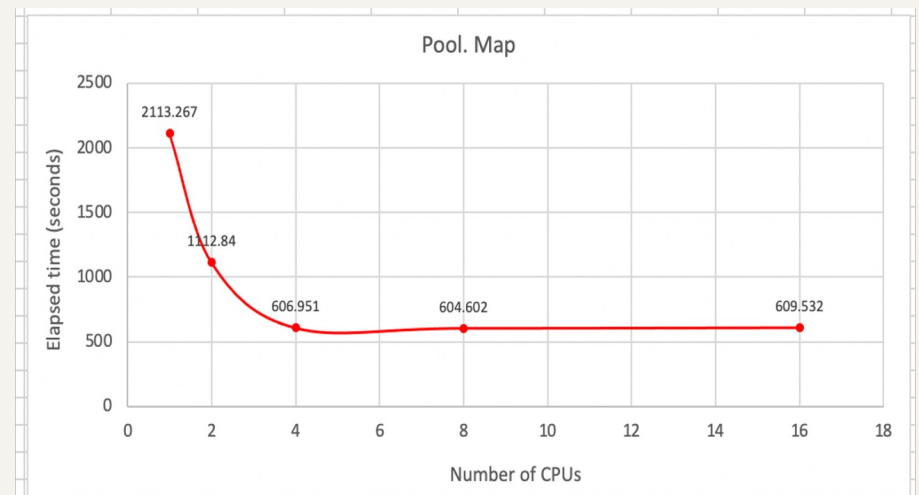
Pool Parallelism runs 2113.267 seconds on 1 CPUs.

Pool Parallelism runs 1112.840 seconds on 2 CPUs.

Pool Parallelism runs 606.951 seconds on 4 CPUs.

Pool Parallelism runs 604.602 seconds on 8 CPUs.

Pool Parallelism runs 609.532 seconds on 16 CPUs.



# Implementations - Joblib

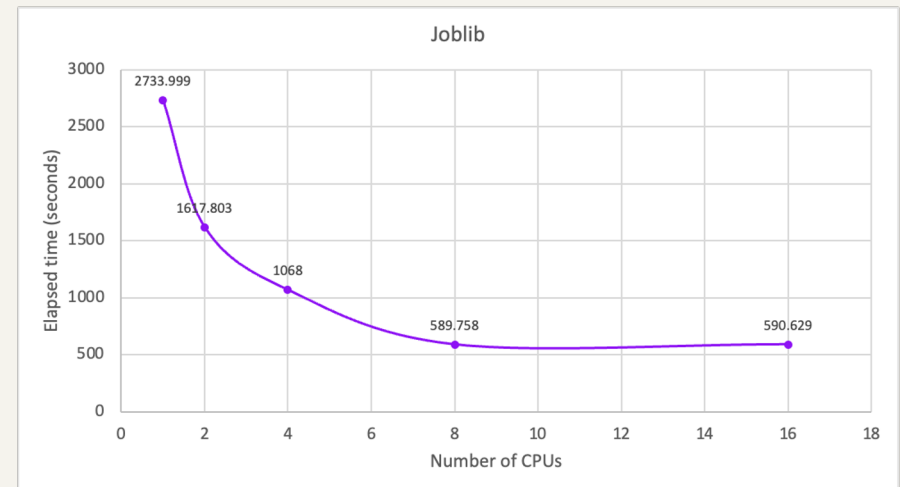
## Parallel, delayed

```
if __name__ == "__main__":  
  
    nums = [1,2,4,8,16]
```

```
t1 = time()  
Parallel(n_jobs = 1)(  
    delayed(func)(parameters)  
    for i in nums  
)  
Time = time() - t1  
print(str(Time) + 's')  
  
t1 = time()  
Parallel(n_jobs = 2)(  
    delayed(func)(parameters)  
    for i in nums  
)  
Time = time() - t1  
print(str(Time) + 's')  
  
t1 = time()  
Parallel(n_jobs = 4)(  
    delayed(func)(parameters)  
    for i in nums  
)
```

```
t1 = time()  
Parallel(n_jobs = 8)(  
    delayed(func)(parameters)  
    for i in nums  
)  
Time = time() - t1  
print(str(Time) + 's')  
  
t1 = time()  
Parallel(n_jobs = 16)(  
    delayed(func)(parameters)  
    for i in nums  
)  
Time = time() - t1  
print(str(Time) + 's')
```

```
2733.9988963603973s  
1617.8027727603912s  
1068.0003304481506s  
589.7575218677521s  
590.6288909912109s
```



# Implementations - Pathos ProcessPool

## Pool.map

```
cores = [1,2,4,8,16]
for i in cores:
    start = time()
    pool = ProcessPool(i)
    pool.map(func, parameters)
    pool.clear()

print('\nPool Parallelism runs %0.3f seconds on %s CPUs.\n' % ((time() - start), i))
```

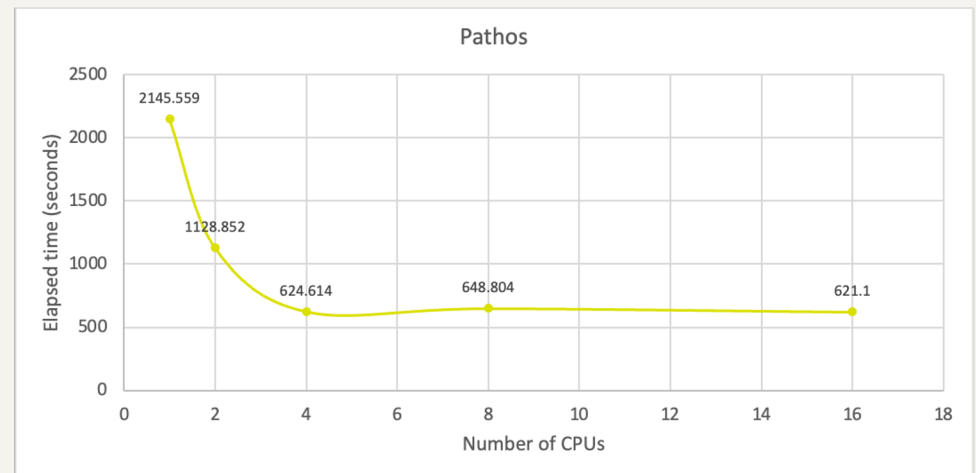
Pool Parallelism runs 2145.559 seconds on 1 CPUs.

Pool Parallelism runs 1128.852 seconds on 2 CPUs.

Pool Parallelism runs 624.614 seconds on 4 CPUs.

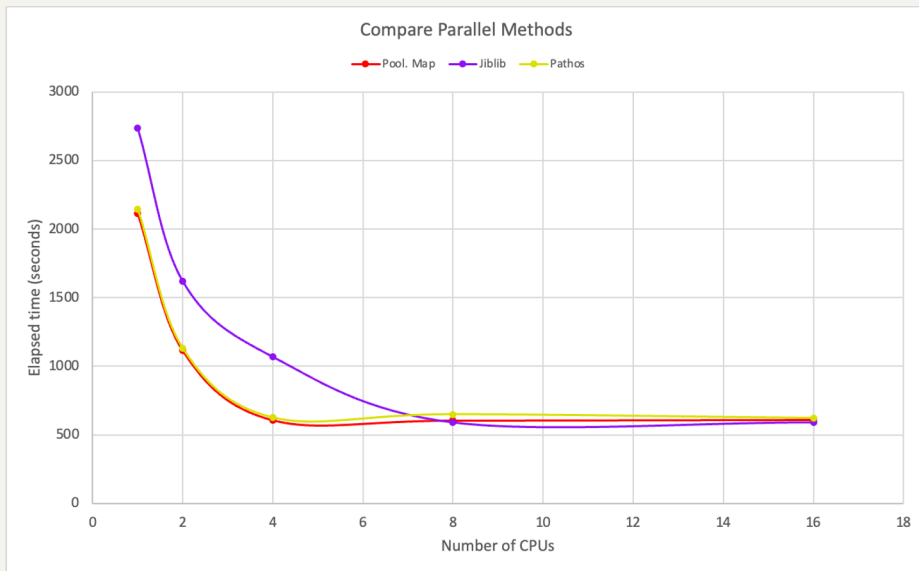
Pool Parallelism runs 648.804 seconds on 8 CPUs.

Pool Parallelism runs 621.100 seconds on 16 CPUs.



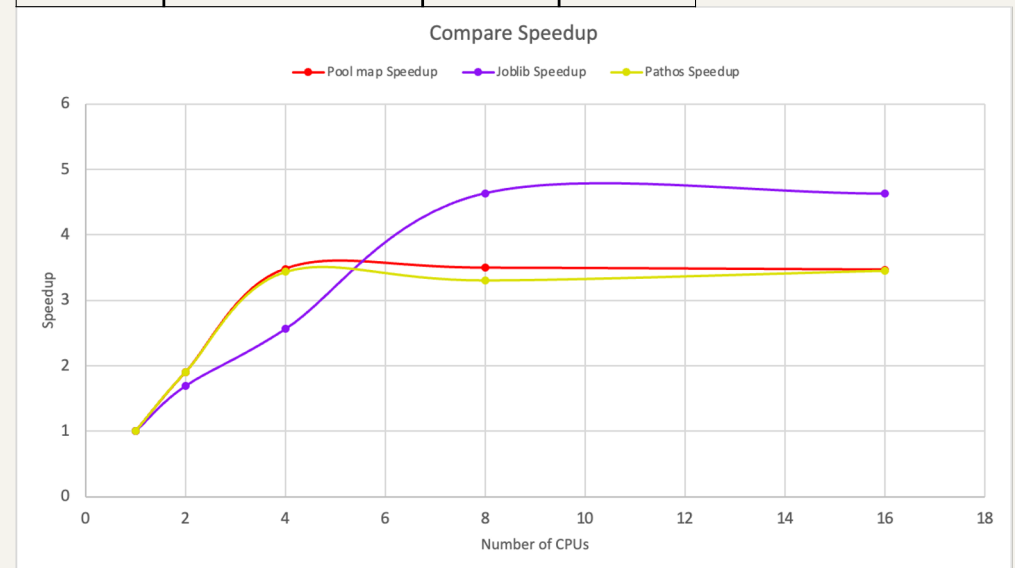
# Implementations - Visualization

Parallel Performance Visualization



Number of CPU	Multiprocessing	Jolib	Pathos
1	1	1	1
2	1.89899	1.68995	1.90066
4	3.48178	2.55992	3.43502
8	3.49530	4.63580	3.30694
16	3.46703	4.62896	3.45445

Speedup Visualization



# Open Issues

1. Communication
  - ❑ Challenges: communication requires overhead, processor may need to wait for other process
  - ❑ Possible Solution: use shared memory method to share a memory space to communicate
2. Load Balancing Issue
  - ❑ Challenges: If input size is not balanced (not even), then the work is unbalanced
  - ❑ Possible Solution: use a static/dynamic load balancing (dynamic has advantage but requires monitor the performance)
3. Race Condition Bug
  - ❑ Challenges: If one of processors execute the code with different amount of time than expected, it can cause unexpected behavior
  - ❑ Possible Solution: use synchronization mechanisms to keep processors synchronized



# References

---

<https://towardsdatascience.com/parallelization-w-multiprocessing-in-python-bd2fc234f516>

<http://webdocs.cs.ualberta.ca/~paullu/C681/parallel.timeline.html>

[https://en.wikipedia.org/wiki/Parallel\\_computing#:~:text=Michael%20J.,now%20known%20as%20Flynn's%20taxonomy.](https://en.wikipedia.org/wiki/Parallel_computing#:~:text=Michael%20J.,now%20known%20as%20Flynn's%20taxonomy.)

<https://computerscience.chemeketa.edu/cs160Reader/ParallelProcessing/AmdahlsLaw.html>

<https://www.sitepoint.com/python-multiprocessing-parallel-programming/>

[http://www.umsl.edu/~siegelj/CS4740\\_5740/Overview/Background.html](http://www.umsl.edu/~siegelj/CS4740_5740/Overview/Background.html)

<https://computer.howstuffworks.com/parallel-processing.htm>

<https://docs.python.org/3/library/multiprocessing.html>

<https://www.spiceworks.com/tech/iot/articles/what-is-parallel-processing/>

# Discussion

# Test Questions

---

1. What is the purpose of Amdahl's Law?
2. What is the possible issue with parallel processing?
3. Which method during the implementation gave the best result?