High-Performance Programming

Miles Erickson

October 24, 2016

Afternoon Objectives

- Compare and contrast processes vs. threads
- Compare and contrast parallelism and concurrency
- Identify problems that require parallelism or concurrency
- Implement parallel and concurrent solutions
- Measure the run time of different approaches to see the benefit of threading/parallelism.

Agenda

Morning - AWS

Afternoon

- Discuss computer resources
- Talk about processes
- Talk about threads

Motivation

- Process biggish data ($\geq$ 5GB depending on task)
- More efficient use of CPU resources
- Saves time

Computing Resources

- Central Processing Unit (CPU) Clock speed measured in GHz (cycles/second)
- Random Access Memory (RAM) Size measured in GB (gigabytes)
- Persistent Storage (disk) Size measured in GB (gigabytes)
- Graphics Processing Unit (GPU)

CPU Cores

- A CPU can have multiple cores
- Each core is a self-contained processor that can execute programs
- GPUs have many cores

Processes

An instance of a computer program that is being executed.

Each process has its own memory, program text, filehandles, permissions, etc. and can run on any core.

A computer runs many, many processes, most just waiting.

```
$ ps aux | wc -l
     233
```

Multiprocessing in python

```
from multiprocessing import Pool
import os

# Count the number of words in a file
def word_count(f):
    return len(open(f).read().split())

# Use a regular loop to count words in files
def sequential_word_count(lst_of_files):
    return sum([word_count(f) for f in lst_of_files])

# Use a multiple cores to count words in files
def parallel_word_count(lst_of_files):
    pool = Pool(processes=4)
    results = pool.map(word_count, lst_of_files)
    return sum(results)
```

Threads

Each process contains one or more **threads** of execution

Threads are lighter-weight than processes

- faster to create
- less memory overhead
- inter-thread communication easier (shared memory)
- faster to context switch

Multi-threading in python

Can we use these for parallel programming?

Multi-threading in python

Python processes have a Global Interpreter Lock (GIL) that prevents multiple thread from running at once.*

Python threads are concurrent but not parallel

Why use threads?

*In the most common implemetation

Multi-threading in python

Useful when the program has to wait on resources outside of the python code

- I/O
- Database queries
- Certain libraries (e.g., image processing)

Multi-threading in python

```
import threading

jobs = []
for i in range(num_threads):
    t = threading.Thread(target=target_function,
                         args=(arg1, arg2))
    jobs.append(t)
    t.start

# "join" will wait until the thread terminates
results = []
    t.join()
    # Access the result of the thread (if any) and append
    results.append(t.result)
```

Summary

What?	Library	Cores	Why?
Parellelism	multiprocessing	multiple	CPU-bound problems
Concurrency	threading	single	I/O-bound problems