Computer Architecture

- Each computer can have multiple CPUs, each CPU has multiple cores (e.g., two quad-core CPUs)
- All the CPUs are connected to memory (e.g., 64G memory)
- CPU cores can execute in parallel
Multicore Programming

- Execute tasks simultaneously on many CPU cores

[parallel processing]

[serial processing]
What is a thread?

- Multiple threads share the memory.
- Don’t need inter-process communication.
- They are “light-weighted” (not much overhead to fork multiple threads)
What is a process?

- Processes are “share nothing”
  (independent executing without sharing memory or state)
- Easier to turn into a distributed application.
Python threads

- Package “threading”
- Unfortunately, python only allows a single thread to be executing at once
  
  (due to GIL (global interpreter lock))
- Usually no or little speedup
  
  only useful when you want to interleave I/O and CPU execution
Python processes

- Package “multiprocessing”
- You can create multiple processes
  - Automatically run on multiple CPU cores
  - Default no shared memory, each process has its own memory space (larger memory overhead)
  - Can declare some part of memory to be shared (but often harder to use)
- You can also check some tutorials:
  - http://sebastianraschka.com/Articles/2014_multiprocessing.html
  - https://pymotw.com/2/multiprocessing/basics.html
import multiprocessing as mp

def helloworld(x):
    print "Hello World %d\n"%x

# Setup a list of processes
plist = []
for x in range(4):
    plist.append(mp.Process(target=helloworld, args=(x,)))

# Run processes
for p in plist:
    p.start()

# Exit the completed processes
for p in plist:
    p.join()
Output of the program:

Hello World 0
Hello World 1
Hello World 2
Hello World 3
Basic functions

- (Check https://docs.python.org/2/library/multiprocessing.html)
- “Process(target=helloworld, args=(x,))”:
  - Specify the target function to run (helloworld)
  - Specify the input argument of the function (only one argument x)
  - Create an object belongs to “Process” type

- The process will run when execute “process.start()”
- The process will terminate when execute “process.join()”
import multiprocessing as mp

def f(x, q):
    q.put(x**2)
    return

q = mp.Queue()
processes = []
for x in range(4):
    processes.append(mp.Process(target=f, args=(x, q)))

for p in processes:
    p.start()
for p in processes:
    p.join()

while (q.empty==False):
    print q.get()
Example: Exchanging objects using Queue (Output)

Output of the program:

0
1
4
9
Use of Queue

- mp.Queue is a concurrent and "first in first out" data structure
- Can be used to communicate, or gather the results from the processes
- Queue.put(): insert an object to the end of queue
- Queue.get(): remove the first element in the queue
- Queue.empty: check whether the queue is empty
Using Pool

- Pool class is another and more convenient approach for parallel processing in python.
- Use “mp.Pool(processes=4)” to create 4 processes.
- Use “\([r_1, r_2, \cdots, r_k] = \text{pool.map}(f, [x_1, x_2, \cdots, x_k])\)” to run multiple processes and get the results.
  - \(f\) is the function to run for the processes.
  - \([x_1, \cdots, x_k]\) are the input arguments we want to run for the function (this is a size \(k\) list).
  - \([r_1, \cdots, r_k]\) are the output arguments we get after running the functions for each input (this is a size \(k\) list).
  - \(k\) may be larger than number of processes.
Example: using Pool

```python
import multiprocessing as mp

def f(x):
    return x**2

pool = mp.Pool(processes=4)
results = pool.map(f, range(4))
print(results)

Output of the program:

[0, 1, 4, 9]
```
Example: Compute sum of square

```python
import multiprocessing as mp

def f(x):
    return x**2

pool = mp.Pool(processes=4)
results = pool.map(f, range(100))
print(sum(results))

Output of the program:

328350
```
Coming up

- Intro to distributed computing

Questions?