Parallel processing in Python

David Arroyo Menéndez

November 12, 2019

My processors!

How many maximum parallel processes can you run?

```
import multiprocessing as mp
print("Number of processors: ", mp.cpu_count())
```

```
$ cat /proc/cpuinfo
```

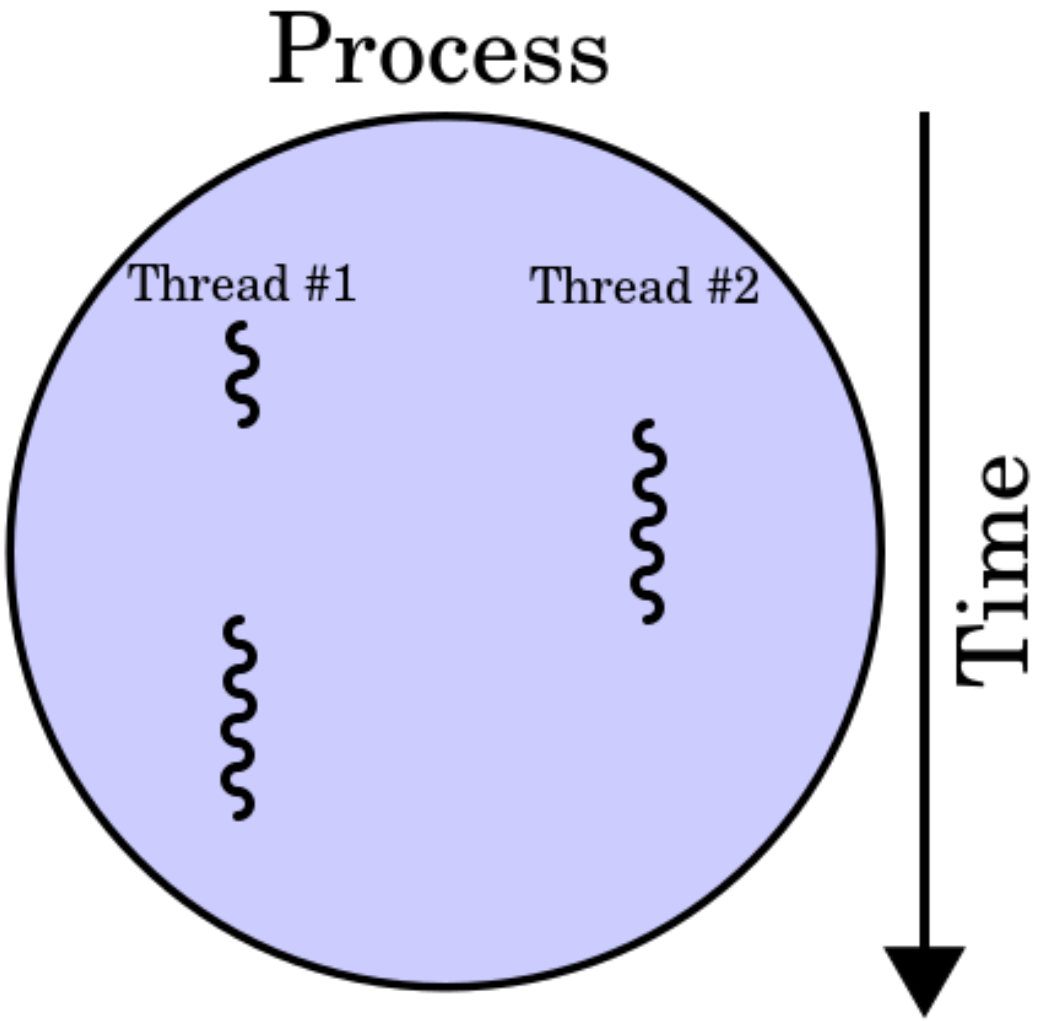
Multiprocesses

```
$ python3 poolapply.py
$ python3 poolmap.py
$ python3 pool.py
```

Thread Definition

A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.
The threads of a process share its executable code and the values of its dynamically allocated variables and non-thread-local global variables at any given time.

Threads: same source code executable, some variables located in different memory space



Threads versus Processes

- Processes are typically independent, while threads exist as subsets of a process
- Processes carry considerably more state information than threads, whereas multiple threads within a process share process state as well as memory and other resources
- Processes have separate address spaces, whereas threads share their address space
- Processes interact only through system-provided inter-process communication mechanisms
- Context switching between threads in the same process typically occurs faster than context switching between processes

Multithreading

- Responsiveness
- Faster execution
- Lower resource consumption
- Better system utilization+
- Simplified sharing and communication
- Parallelization

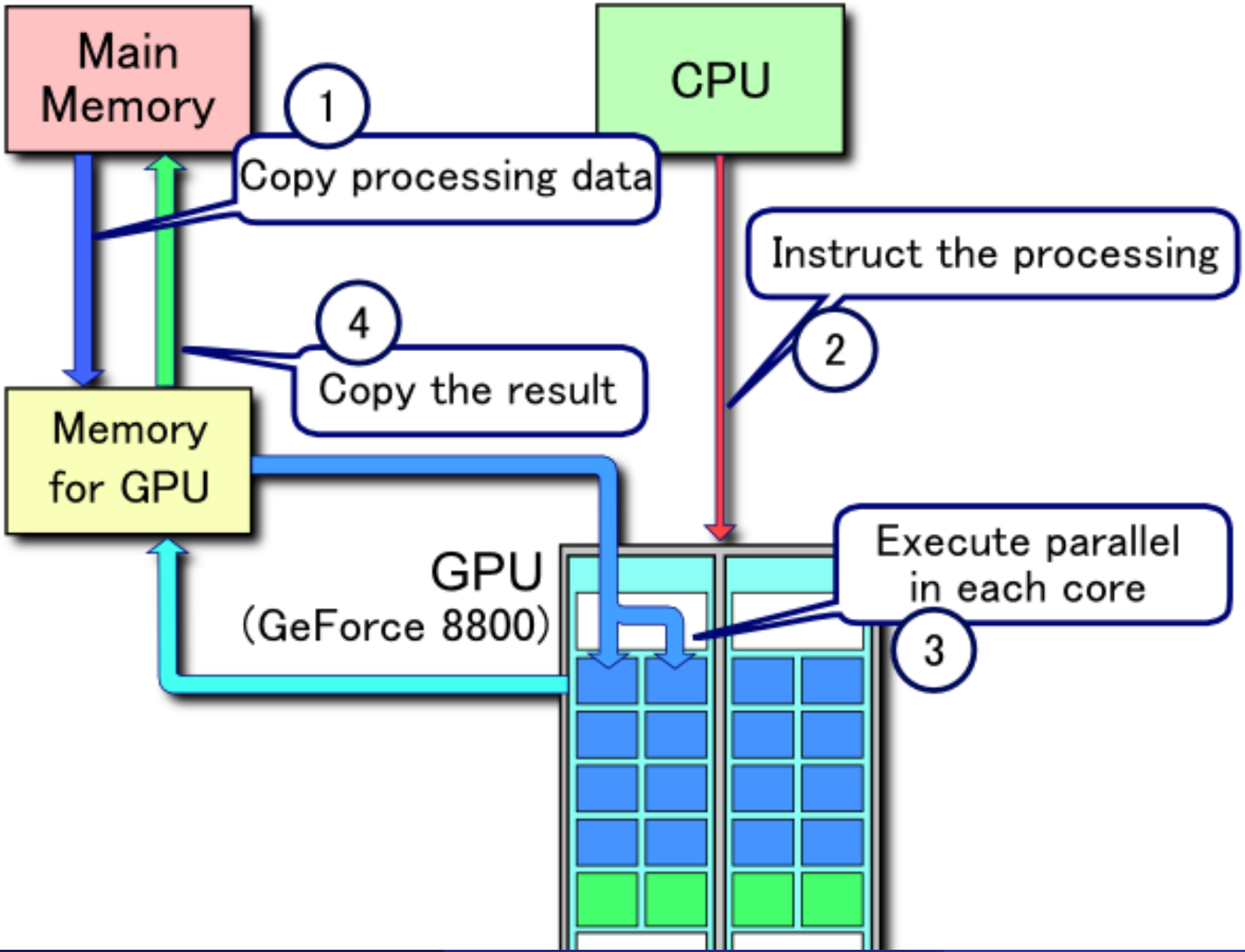
A semaphore with threads

```
import threading
import time
sem = threading.Semaphore()

def fun1():
while True:
try:
sem.acquire()
print(1)
sem.release()
time.sleep(0.25)
except (KeyboardInterrupt, SystemExit):
raise

def fun2():
while True:
try:
```

More examples

```
$ python3 multithreading.py
$ python3 thread-example.py
$ python3 threading-simple.py
$ python3 threading-timeout5.py
$ python3 threads.py
```

CUDA


Main Memory
CPU
1
Copy processing data
Instruct the processing
2
4
Copy the result
Memory for GPU
GPU
(GeForce 8800)
Execute parallel in each core
3

NUMBA: CUDA in Python

```
$ python3 numba/pi.py
```

License