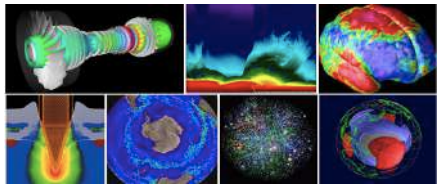
Tính toán song song và phân tán

PGS.TS. Trần Văn Lăng
langtv@vast. vn

Tài liệu: Introduction to Parallel Computing

Blaise Barney, Lawrence Livermore National Laboratory

https://computing.llnl.gov/tutorials/parallel_comp/



Lập trình song song với Python

1. Tổng quan
2. Sự khác biệt giữa Thread và Process
3. Song song dữ liệu
4. Sử dung Thread
5. Sử dụng Process
6. Đa xử lý trong Khoa học dữ liệu

Tổng quan

- Lập trình song song với Python có thể tiếp cận theo các cách như:
  - Dùng thư viện của của Python
    - Threading, multiprocessing
  - Dùng ScientificPython
    - Scientific.BSP
    - Scientific.DistributiedComputing
  - IPython
    - Interactive shell for working with clusters
  - Khác
    - pyMPI, mpi4py, pypar

- Để kiểm tra cấu hình của máy, sử dụng gói platform (sysinfor.py)

```
import platform
import multiprocessing as mp

# Subroutine
def print_sysinfo():
    print('Python version:', platform.python_version())
    print('compiler   :', platform.python_compiler())
    print('system     :', platform.system())
    print('release    :', platform.release())
    print('machine    :', platform.machine())
    print('processor  :', platform.processor())
    print('CPU count  :', mp.cpu_count())
    print('interpreter:', platform.architecture()[0])
    print('\n\n')

# To running
print_sysinfo()
```

- Để biết số core của máy, sử dung gói multiprocessing

```
import multiprocessing as mp

# Subroutine
def print_sysinfo():
    print('CPU count  :', mp.cpu_count())
    print('\n\n')
```


- Một số kết quả thử nghiệm

```
Lion:parallel lang$ python3 sysinfor.py
Python version: 3.6.5
compiler   : GCC 4.2.1 (Apple Inc. build 5666) (dot 3)
system     : Darwin
release    : 17.5.0
machine    : x86_64
processor  : i386
CPU count  : 4
interpreter: 64bit
```

```
[iami.langtv@gpu parallel]$ vi sysinfor.py
[iami.langtv@gpu parallel]$ python sysinfor.py
('Python version:', '2.6.6')
('compiler   :', 'GCC 4.4.7 20120313 (Red Hat 4.4.7-17)')
('system     :', 'Linux')
('release    :', '2.6.32-431.29.2.el6.x86_64')
('machine    :', 'x86_64')
('processor  :', 'x86_64')
('CPU count  :', 24)
('interpreter:', '64bit')
```


- Để biết tên và địa chỉ IP của máy (hostname.py)

```
import socket

hn = socket.gethostname()
ip = socket.gethostbyname(hn)
print( "Host name is %s, and IP Address is %s" % (hn,ip) )
```


- Để đo thời gian thực thi, dùng hàm time() có trong gói time (elapsetime.py)

```
import time

start = time.time()
x = 0
while x < 10000000:
        x += 1
end = time.time()

print( "Elapse time is:", end - start, "seconds" )
```

Có thể coi thêm tại https://www.tutorialspoint.com/python/python_date_time.htm

Thread và Process

- **Process** (tiến trình) là một thực thể (instance) của chương trình máy tính.
- Tiến trình sinh ra các **Threads** (luồng) hay còn gọi là **sub-processes** (tiểu tiến trình) để xử lý các nhiệm vụ phụ. Chẳng hạn, đọc các phím được nhấn (keystrokes), hiển thị ký tự vừa đọc lên màn hình, nạp một trang văn bản từ đĩa vào bộ nhớ sau đó hiển thị lên màn hình, ghi tập tin vào đĩa, ...
- **Threads** tồn tại (live) bên trong tiến trình và chia sẻ cùng không gian bộ nhớ của **Process**.

- Ví dụ, khi mở một ứng dụng soạn thảo văn bản; có nghĩa là chúng ta tạo ra một Process.
- Khi bắt đầu nhập, Process sinh ra các Threads chẳng hạn như:
  - đọc các phím bấm
  - hiển thị văn bản
  - tự động lưu tập tin lại
  - làm nổi bật (highlight) các lỗi chính tả khi nhập

- Bằng cách sinh ra nhiều Thread, Chương trình soạn thảo văn bản tận dụng thời gian nhàn rỗi của CPU (chờ đợi các phím bấm hoặc tải tập tin về) qua đó làm cho công việc có năng suất hơn.

- Như vậy về việc sử dung bộ nhớ: Threads và Processes là khác nhau:
  - Threads dùng shared-memory, trong khi đó Processes thì có thể không.
  - Sự đồng bộ hóa về dữ liệu là cần thiết đối với các Threads, nhưng với Processes là có thể không cần

Process

- Một Process có thể có nhiều Thread
- Process được tạo ra bởi hệ điều hành để thực thi chương trình
- Hai Process có thể thi hành đồng thời một đoạn code trong chương trình Python
- Khi mở và đóng Process tốn nhiều thời gian hơn so với mở đóng Thread
- Trong trường hợp không chia sẻ không gian bộ nhớ, thì việc chia sẻ thông tin giữa các Process chậm hơn giữa các Thread

Threads

- Thread như là những tiến trình nhỏ (mini-processes) tồn tại bên trong Process
- Thread chia sẻ không gian bộ nhớ nên các biến được dung chung trong việc đọc ghi
- Hai Threads không thể thực thi đồng thời cùng một đoạn code trong chương trình Python

Ví dụ về sự khác nhau

- Hàm (myFunc) xuất ID của thread đang hoạt động.
- Khi gọi hàm này nhiều lần, ID này được lưu trữ lại trong một mảng toàn cục **A**.

```
import threading
import time
import random

A = [ ]
print( "Length of A = %d. Thread ID: %d" % (len(A),threading.get_ident()) )
print( "A =", A )

def myFunc(n):
    s = random.randint( 1,3 )
    time.sleep( s )
    m = threading.get_ident()
    A.append( m )
    print( "n = %2d. Time to sleep = %d. Th.ID: %d. Main Th.ID: %d"
          % (n,s,m,threading.main_thread().ident) )
```


- Kích hoạt 10 lần để cho Thead gọi hàm **myFunc()** đã tạo ở trên.
- Việc kết thúc một thread để chuyển sang thread khác là không đặt ra ngay khi một thread được start; chính vì vậy các thread cần được lưu trữ vào trong mảng **th** để sử dụng về sau.

```
th = [ ]
for i in range(10):
    t = threading.Thread( target = myFunc, args=(i,) )
    th.append( t )
    t.start()
```

- Sau đó mới lần lượt đợi cho các thread này hoàn thành công việc bằng hàm join().
- Sau khi xong, kết quả có trong biến toàn cục A được xuất ra

```
for i in th:
    i.join()    # Waiting for all threads to terminate

print( "Length of A = %d. Thread ID: %d" % (len(A),threading.get_ident()) )
print( "A =", A )
```


Kết quả thực thi chương trình với Thread

```
Lion:nttu lang$ python3 ThreadProcess.py
Length of A = 0. Thread ID: 140735566779264
A = []
n =  0. Time to sleep = 1. Th.ID: 123145378865152. Main Th.ID: 140735566779264
n =  2. Time to sleep = 1. Th.ID: 123145389375488. Main Th.ID: 140735566779264
n =  3. Time to sleep = 1. Th.ID: 123145394630656. Main Th.ID: 140735566779264
n =  7. Time to sleep = 2. Th.ID: 123145415651328. Main Th.ID: 140735566779264
n =  4. Time to sleep = 2. Th.ID: 123145399885824. Main Th.ID: 140735566779264
n =  9. Time to sleep = 2. Th.ID: 123145426161664. Main Th.ID: 140735566779264
n =  1. Time to sleep = 3. Th.ID: 123145384120320. Main Th.ID: 140735566779264
n =  5. Time to sleep = 3. Th.ID: 123145405140992. Main Th.ID: 140735566779264
n =  8. Time to sleep = 3. Th.ID: 123145420906496. Main Th.ID: 140735566779264
n =  6. Time to sleep = 3. Th.ID: 123145410396160. Main Th.ID: 140735566779264
Length of A = 10. Thread ID: 140735566779264
A = [123145378865152, 123145389375488, 123145394630656, 123145415651328, 123145
399885824, 123145426161664, 123145384120320, 123145405140992, 123145410396160,
123145420906496]
Lion:nttu lang$
```


Nhận xét

- Do thời gian sleep khác nhau, nên các thread kết thúc không giống nhau, dẫn đến thứ tự công việc không theo thứ tự của câu lệnh lặp for
- Thread theo cơ chế sử dụng bộ nhớ chia sẻ, nên biến A có hiệu lực trong thread (ID...64) chính và các thread khác (ID...52, ID...88, ..., ID...60). Vì vậy mảng A ban đầu rỗng, sau khi thực hiện mảng A gồm 10 phần tử là các Thread ID.

- Như vậy, biến toàn cục A được sử dụng chung giữa các thread với nhau (shared-memory)
- Với chương trình như trên; nhưng thay vì dùng **Thread**, dùng **Process** thì biến toàn cục A này không tác động giữa các process
- Cần import

```
import multiprocessing
import time
import random
import os
```

- Hàm để các process thực hiện

```
A = [ ]
print( "Length of A = %d. Process ID: %d" % (len(A),os.getpid()) )
print( "A =", A )

def myFunc(n):
    s = random.randint( 1,3 )
    time.sleep(s)
    p = os.getpid()
    pp = os.getppid()
    A.append( p )
    print( "n = %d. Time to sleep = %d. Process ID: %d.
           Parent Process ID: %d" % (n,s,p,pp) )
```


- Thực thi tất cả các process

```
pr = [ ]
for i in range( 10 ):
    t = multiprocessing.Process( target = myFunc, args=(i,) )
    pr.append( t )
    t.start()

for i in pr:
    i.join()

print( "Length of A = %d. ProcessID: %d" % (len(A),os.getpid()) )
print( "A =", A )
```


Kết quả thực thi chương trình với Process

```
Lion:nttu lang$ python3 ThreadProcess.py
Length of A = 0. Process ID: 22873
A = []
n = 3. Time to sleep = 1. Process ID: 22877. Parent Process ID: 22873
n = 7. Time to sleep = 1. Process ID: 22881. Parent Process ID: 22873
n = 6. Time to sleep = 1. Process ID: 22880. Parent Process ID: 22873
n = 9. Time to sleep = 1. Process ID: 22883. Parent Process ID: 22873
n = 0. Time to sleep = 2. Process ID: 22874. Parent Process ID: 22873
n = 1. Time to sleep = 2. Process ID: 22875. Parent Process ID: 22873
n = 2. Time to sleep = 2. Process ID: 22876. Parent Process ID: 22873
n = 4. Time to sleep = 2. Process ID: 22878. Parent Process ID: 22873
n = 5. Time to sleep = 3. Process ID: 22879. Parent Process ID: 22873
n = 8. Time to sleep = 3. Process ID: 22882. Parent Process ID: 22873
Length of A = 0. ProcessID: 22873
A = []
Lion:nttu lang$ _
```


Lưu ý

- Dùng Process nhưng để có kết quả như Thread, cần phải lưu giá trị vào trong một hàng đợi bằng cách khai báo biến Q = Queue().
- Sau đó đẩy kết quả cần lưu vào trong hàng đợi này. Chẳng hạn, Q.put(os.getpid()).
- Cuối cùng, để xuất ra và đưa vào mảng A qua lệnh A.append(Q.get())

- Chương trình viết lại như sau

```
import multiprocessing
import time
import random
import os
from multiprocessing import Queue

A = [ ]
print( "Length of A = %d. Process ID: %d" % (len(A),os.getpid()) )
print( "A =", A )

Q = Queue()
```



```
def myFunc( n ):
    s = random.randint( 1,3 )
    time.sleep(s)
    p = os.getpid()
    pp = os.getppid()
    Q.put( p )
    print( "n = %d. Time to sleep = %d. Process ID: %d.
            Parent Process ID: %d" % (n,s,p,pp) )
pr = [ ]
for i in range( 10 ):
    t = multiprocessing.Process( target = myFunc, args=(i,) )
    pr.append( t )
    t.start()
for i in pr:
    i.join()
while not Q.empty():
    A.append( Q.get() )
print( "Length of A = %d. ProcessID: %d" % (len(A),os.getpid()) )
print( "A =", A )
```


Kết quả

```
Lion:nttu lang$ python3 Process2.py
Length of A = 0. Process ID: 23478
A = []
n = 1. Time to sleep = 1. Process ID: 23480. Parent Process ID: 23478
n = 6. Time to sleep = 1. Process ID: 23485. Parent Process ID: 23478
n = 7. Time to sleep = 1. Process ID: 23486. Parent Process ID: 23478
n = 0. Time to sleep = 2. Process ID: 23479. Parent Process ID: 23478
n = 2. Time to sleep = 2. Process ID: 23481. Parent Process ID: 23478
n = 3. Time to sleep = 2. Process ID: 23482. Parent Process ID: 23478
n = 4. Time to sleep = 2. Process ID: 23483. Parent Process ID: 23478
n = 5. Time to sleep = 2. Process ID: 23484. Parent Process ID: 23478
n = 8. Time to sleep = 2. Process ID: 23487. Parent Process ID: 23478
n = 9. Time to sleep = 3. Process ID: 23488. Parent Process ID: 23478
Length of A = 10. ProcessID: 23478
A = [23480, 23485, 23486, 23479, 23481, 23482, 23483, 23484, 23487, 23488]
Lion:nttu lang$
```


Song song dữ liệu

- multiprocessing là một gói các API hỗ trợ để phát sinh các Process; qua đó giúp tận dụng được nhiều bộ xử lý để viết chương trình.
- Giả sử có một công việc (viết dưới dạng một hàm) được thực hiện trên nhiều dữ liệu khác nhau (Data Parallelism).

- Ví dụ: Cần tính căn bậc 2 của 3 số.
- Các câu lệnh tuần tự có thể viết như sau:

```
import math
##########
# Function to calculate square root
def g(x):
        return math.sqrt(x)
##########
for x in [1, 2, 3]:
        print( g(x) )
```


- Hoặc như sau với hàm map()

```
import math
##########
# Function to calculate square root
def g(x):
        return math.sqrt(x)
##########
print( map( g,[1, 2, 3] ) )
```


- Khi cần giao cho nhiều Process tính, chẳng hạn với 3 Process:

```
import math
from multiprocessing import Pool
##########
# Function to calculate square root
def g(x):
        return math.sqrt(x)
##########
p = Pool(3)
print( p.map(g,[1, 2, 3]) )
```


- Trong đó, đối tượng Pool cung cấp một phương tiện để song song hóa việc thực hiện một hàm qua nhiều giá trị đầu vào, và phân phối dữ liệu đầu vào cho các Process.

- Với hàm nhiều đối số.
- Ví dụ: Giải 3 phương trình bậc I dạng ax+b=0

```
def f(a,b):
        print( "Phuong trinh %fx + %f = 0" %(a, b) )
        if a != 0:
                x = -float(b)/a
                s = "Phuong trinh co 1 nghiem la " + str(x)
        else:
                if b != 0:
                        s = "Phuong trinh vo nghiem"
                else:
                        s = "Phuong trinh co vo so nghiem"
        return s
print( map(f,[2,0,0],[3,1,0]) )
```


- Lưu ý: khi dùng với đối tượng thuộc lớp Pool, hàm map() này không chuyển đổi nhiều đối số được.
- Để thuận lợi, có thể dùng một lớp Container, chẳng hạn dùng gói collections để gom các đối số của hàm thành một nhóm.
- Chẳng hạn, hàm f(a,b) có 2 đối số để giải phương trình bậc nhất như trên, có thể viết lại thành một đối số f(item)


```
def f( item ):
        print( "Phuong trinh %fx + %f = 0" %(item.a, item.b) )
        if item.a != 0:
                x = -float(item.b)/item.a
                s = "Phuong trinh co 1 nghiem la " + str(x)
        else:
                if item.b != 0:
                        s = "Phuong trinh vo nghiem"
                else:
                        s = "Phuong trinh co vo so nghiem"
        return s
```


- Trong đó, item có 2 thành phần là a và b
- Khi sử dụng, phải tạo ra kiểu đối số này:

```
import collections
Argument = collections.namedtuple( "Argument",['a','b'] )
args = (
        Argument(a = 2, b = 3),
        Argument(a = 0, b = 1),
        Argument(a = 0, b = 0),
)
```

- Khi sử dụng

```
# Thực hiện tuần tự
print( map(f,args) )

# Song song dữ liệu
from multiprocessing import Pool
p = Pool(3)
print( p.map(f,args) )
```


- Có thể so sánh thời gian thực hiện của chương trình song song so với tuần tự qua ví dụ giải phương trình.

```
# -*- coding: utf-8 -*-
# Function with multipe arguments with map()
# Author: A.Prof. Tran Van Lang
# Filename: equationI.py
#
import collections
import time
from multiprocessing import Pool
#
# Define data structure
#
Argument = collections.namedtuple( "Argument",['a','b'] )
args = (
        Argument(a = 2, b = 3),
        Argument(a = 0, b = 1),
        Argument(a = 0, b = 0),
)

def f( item ):
        x = 0
        while x < 10000000:
                x += 1
        print( "Phuong trinh %fx + %f = 0" %(item.a, item.b) )
        if item.a != 0:
                x = -float(item.b)/item.a
                s = "Phuong trinh co 1 nghiem: " + str(x)
        else:
                if item.b != 0:
                        s = "Phuong trinh vo nghiem"
                else:
                        s = "Phuong trinh vo dinh"
        return s

# Run tasks serially
#
start = time.time()
print( map(f,args) )
print( "Thời gian tuần tự: %4.2f giây" %(time.time() - start) )

# Data Parallelism
#
```


Sử dụng Thread

- Dùng lớp **Thread** trong gói threading.
- Chẳng hạn, cần kích hoạt một Thread để thực hiện công việc qua hàm có sẵn (giải phương trình **eqI**)

```
import myfunctions as mf
import threading as th

thread = th.Thread( target=mf.eqI, args=[2,3] )
thread.start()
thread.join()
```


Sử dụng Process

- Tương tự như **Thread**, có thể dùng lớp **Process** trong gói multiprocessing để thực thi hàm **eqI()**

```
import myfunctions as mf
import multiprocessing as mp

thread = mp.Process( target=mf.eqI, args=[2,3] )
thread.start()
thread.join()
```

- Với kết quả

```
Lion:parallel lang$ python ex_thread.py
Phuong trinh  2.00x +  3.00 = 0
Phuong trinh co 1 nghiem: -1.5
```

Threads và Processes

- Giả sử có một chương trình chỉ chờ cho hết thời gian (đặt tên là **sleeping()**), không thực hiện bất kỳ tính toán nào như sau:

```
def sleeping():
        print("Parent process: %s, PID: %s, Process name: %s, Thread name: %s" % (
                os.getppid(),
                os.getpid(),
                mp.current_process().name,
                th.current_thread().name)
        )
        time.sleep(1)
```


- Nếu run chương trình này trên tất cả các Thread của máy qua ví dụ:

```
# Run tasks using threads
start = time.time()
threads = [th.Thread(target=mf.sleeping) for _ in range(NUM_WORKERS)]
[thread.start() for thread in threads]
[thread.join() for thread in threads]
print( "==> Threads time: %5.2f secs" %(time.time() - start) )
```


4 lần gọi

- Thì thời gian thực thi là thời gian của Thread lâu nhất:

```
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: MainThread
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: MainThread
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: MainThread
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: MainThread
==> Serial time:  4.01 secs
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: Thread-1
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: Thread-2
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: Thread-3
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: Thread-4
==> Threads time:  1.01 secs
```

Các Threads của Process 6269

- Nếu run chương trình này trên tất cả các Process, mỗi Process sử dung Thread chính:

```
# Run tasks using processes
start = time.time()
processes = [mp.Process(target=mf.sleeping) for _ in range(NUM_WORKERS)]
[process.start() for process in processes]
[process.join() for process in processes]
print( "==> Parallel time: %5.2f secs" %(time.time() - start) )
```

- Trong trường hợp này, thời gian thực thi trên các Process cũng là thời gian lâu nhất của một Process nào đó trong các Process được kích hoạt

```
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: Thread-1
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: Thread-2
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: Thread-3
Parent process: 5902, PID: 6269, Process name: MainProcess, Thread name: Thread-4
==> Threads time:  1.01 secs
Parent process: 6269, PID: 6272, Process name: Process-1, Thread name: MainThread
Parent process: 6269, PID: 6273, Process name: Process-2, Thread name: MainThread
Parent process: 6269, PID: 6274, Process name: Process-3, Thread name: MainThread
Parent process: 6269, PID: 6275, Process name: Process-4, Thread name: MainThread
==> Parallel time:  1.01 secs
```


- Nhưng khi có một chương trình có tính toán (đặt tên là **calculating()**)

```
def calculating():
        print("Parent process: %s, PID: %s, Process name: %s, Thread name: %s" % (
                os.getppid(),
                os.getpid(),
                mp.current_process().name,
                th.current_thread().name )
        )
        x = 0
        while x < 10000000:
                x += 1
```


- Thời gian thực thi của Process và Thread là khác biệt:

```
Parent process: 6512, PID: 6524, Process name: MainProcess, Thread name: MainThread
Parent process: 6512, PID: 6524, Process name: MainProcess, Thread name: MainThread
Parent process: 6512, PID: 6524, Process name: MainProcess, Thread name: MainThread
Parent process: 6512, PID: 6524, Process name: MainProcess, Thread name: MainThread
==> Serial time:  2.85 secs
Parent process: 6512, PID: 6524, Process name: MainProcess, Thread name: Thread-1
Parent process: 6512, PID: 6524, Process name: MainProcess, Thread name: Thread-2
 Parent process: 6512, PID: 6524, Process name: MainProcess, Thread name: Thread-3
Parent process: 6512, PID: 6524, Process name: MainProcess, Thread name: Thread-4
==> Threads time:  5.41 secs
Parent process: 6524, PID: 6527, Process name: Process-1, Thread name: MainThread
Parent process: 6524, PID: 6528, Process name: Process-2, Thread name: MainThread
Parent process: 6524, PID: 6529, Process name: Process-3, Thread name: MainThread
Parent process: 6524, PID: 6530, Process name: Process-4, Thread name: MainThread
==> Parallel time:  1.16 secs
```


Truyền và nhận dữ liệu

- Chương trình Python tạo 2 Process, Master và Slave.
- Process Master gửi một mảng 10 giá trị ngẫu nhiên đến Process Slave
- Process Slave nhận và xuất ra màn hình

- Một số import cần thiết

```
from multiprocessing import Process, Pipe
import random
import os
```

- Hàm Master() để tạo giá trị ngẫu nhiên và gửi

```
s = []
def master(conn):
    for i in range(0,10):
        s.append( random.randint(100,110) )
    print( s )
    print("Following Values were sent by PID [%d]" % ( os.getpid()))
    conn.send( s )
    conn.close()
```


- Hàm Slave() để nhận rồi xuất ra màn hình

```
r = []
def slave(conn):
    r = conn.recv()
    print( "PID",os.getpid(),"received as follow:",r )
```

- Chương trình sử dụng đối tượng kết nối được sinh ra bởi Pipe

```
if __name__ == '__main__':
    master_conn, slave_conn = Pipe()
    slave_task = Process( target=slave, args=(slave_conn,) )
    master_task = Process( target=master, args=(master_conn,) )
    slave_task.start()
    master_task.start()
    slave_task.join()
    master_task.join()
```

Một vài ví dụ

- Ví dụ cần tính tổng của *N* số thực có giá trị được tạo ngẫu nhiên trong khoảng [0,1).

```
import numpy as np
N = 10000000
a = np.random.random(N)
```


- Hàm tính tổng có thể viết

```
def psum( ib, ie ):
  s = 0.0
  for i in range(ib,ie+1):
    s += a[i]
  return s
```

- Để tính tổng tuần tự của *N* số này:

```
t0 = time.time()
psum(0,N-1)
print( "Tuần tự: Thời gian trôi qua %8.6f giây" %
(time.time()-t0) )
```


- Trong trường hợp cần dùng đối tượng Pool có trong gói Multiprocessing, chúng ta sử dụng hàm psum() nhiều lần, mỗi lần là một tiến trình tính tổng một đoạn nhỏ. Khi đó cần phải sử dụng hàm map() để tạo một dãy lặp lại.

- Nhưng do hàm psum() có 2 đối số (có 2 tham số khi gọi hàm), nên phải viết thêm một hàm để chuyển việc gọi hàm nhiều tham số về 1 tham số như sau:

```
def multirun(args):
  return psum(*args)
```


- Giả sử dùng 4 tiến trình cho việc tính toán song song này, chương trình viết như sau:

```
from multiprocessing import Pool
t0 = time.time()
l1 = [0, int(N/4), int(N/2), int(3*N/4)]
l2 = [int(N/4)-1, int(N/2)-1, int(3*N/4)-1, N-1]
l3 = zip(l1,l2)
```

```
p = Pool(4)
result = p.map( multirun, l3 )
print( "Sử dụng Pool: Thời gian trôi qua %8.6f giây"
% (time.time()-t0) )
print( "Tổng là %10.6f" % sum(result) )
```


- Lưu ý rằng,
  - Do hàm multirun() chỉ có 1 đối số, nên phải dùng hàm zip() để chuyển 2 danh sách về 1 danh sách.
  - Để biết kết quả sau cùng, có thể dùng hàm sum() của gói Python chuẩn để tính toán.
- Chúng ta có thể sử dụng đối tượng Process hoặc Thread để thực hiện công việc này. Cách thức cũng hoàn toàn như nhau:


```
import threading as th
t0 = time.time()
t1 = th.Thread(target = psum, args = [0,int(N/4)-1])
t2 = th.Thread(target = psum, args = [int(N/4),int(N/2)-1])
t3 = th.Thread(target = psum, args = [int(N/2),int(3*N/4)-1])
t4 = th.Thread(target = psum, args = [int(3*N/4),N-1])
```



```
t1.start()
t2.start()
t3.start()
t4.start()
t1.join()
t2.join()
t3.join()
t4.join()
print( "Sử dụng Thread: Thời gian trôi qua %8.6f giây"
% (time.time()-t0) )
```

- Tuy nhiên, để tính được tổng của tất cả các phần từ, hàm psum() được hiệu chỉnh như sau:

```
result = []
def psum( ib, ie ):
  s = 0.0
  for i in range(ib,ie+1):
    s += a[i]
  result.append( s )
  return s
```


- Cách thức giải quyết dùng Process cũng tương tự Thread. Tuy nhiên, do các process không chia sẻ không gian bộ nhớ, nên để có kết quả là tổng của toàn bộ các phần tử cần tính, phải sử dụng thêm đối tượng Queue để lưu trữ.

```
from multiprocessing import Queue
Q = Queue()
def psump( ib, ie ):
  s = 0.0
  for i in range(ib,ie+1):
    s += a[i]
  Q.put(s)
  return s
```


- Khi đó, việc tính tổng được xử lý sau khi các tiến trình tính toán như sau:

```
t0 = time.time()
p1 = mp.Process(target = psump, args = [0,int(N/4)-1])
p2 = mp.Process(target = psump, args = [int(N/4),int(N/2)-1])
p3 = mp.Process(target = psump, args = [int(N/2),int(3*N/4)-1])
p4 = mp.Process(target = psump, args = [int(3*N/4),N-1])
p1.start()
p2.start()
p3.start()
p4.start()
```



```
p1.join()
p2.join()
p3.join()
p4.join()
result = []
while not Q.empty():
   result.append( Q.get() )
print( "Sử dụng Process: Thời gian trôi qua %8.6f giây"
% (time.time()-t0) )
print( "Tổng là %10.6f" % sum(result) )
```

- Kết quả cả 3 trường hợp và tính toán tuần tự với N là 10 triệu như sau:

```
==> Sử dụng Pool: Thời gian trôi qua 1.813148 giây
Tổng là 5000213.687816
==> Sử dụng Thread: Thời gian trôi qua 2.870040 giây
Tổng là 5000213.687816
==> Sử dụng Process: Thời gian trôi qua 1.819063 giây
Tổng là 5000213.687816
==> Tuần tự: Thời gian trôi qua 3.025585 giây
Tổng là 5000213.687815
```


- Ví dụ tính số π đơn giản nhất là dùng hàm Arctan(x).
- Ta biết

$$\tan\left(\frac{\pi}{4}\right) = 1$$

- Suy ra

$$\arctan(1) = \frac{\pi}{4}$$

- Mà khai triển Taylor của hàm arctan() là

$$arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots + (-1)^{n-1}\frac{x^{2n-1}}{2n-1} + O(x^{2n-1})$$

- Từ đây suy ra

$$\frac{\pi}{4} \approx 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots + (-1)^{n-1}\frac{1}{2n-1} = \sum_{i=1}^{n}(-1)^{i-1}\frac{1}{2i-1}$$

- Để sử dụng lại hàm psum() ở trên, có thể viết đơn giản bằng cách tạo mảng *a*, thay vì ngẫu nhiên trong [0,1) thì *a* là các số hạng trong tổng ở trên.

```
a = []
sign = 1
for i in range(N):
  a.append(sign/(2*(i+1)-1))
  sign = -sign
```


- Ví dụ, dùng đối tượng Pool để tính, toàn bộ chương trình có thể viết như sau:

```
import time
from multiprocessing import Pool

def multirun(args):
  return psum(*args)
def psum( ib, ie ):
  s = 0.0
  for i in range(ib,ie+1):
    s += a[i]
  return s
```



```
N = 10000000
a = []
sign = 1
for i in range(N):
  a.append(sign/(2*(i+1)-1))
  sign = -sign

l1 = [0, int(N/4), int(N/2), int(3*N/4)]
l2 = [int(N/4)-1, int(N/2)-1, int(3*N/4)-1, N-1]
l3 = zip(l1,l2)
p = Pool(4)
result = p.map( multirun, l3 )
print( "Số Pi là %54.52f" % (4*sum(result)) )
```


- Ví dụ dữ liệu phân tán
- Trong các ví dụ trên, chúng ta thực hiện các thao tác để rồi cùng chia sẻ một vùng bộ nhớ chung. Chẳng hạn với Pool hoặc với Thread, dữ liệu được tính toán sau đó tập trung vào list có tên là result. Với Process, kết quả của các process được đưa vào hàng đợi Q, sau đó từ hàng đợi đưa vào result này để tính tổng trên list.

- Giả sử chương trình tính toán làm những công việc khác nhau trên cùng một dữ liệu, nhưng do mỗi process lưu trữ trong vùng bộ nhớ riêng như một hệ thống phân tán (Distributed System).
- Chẳng hạn, process đóng vai trò master chịu trách nhiệm tạo một ma trận là các giá trị thực ngẫu nhiên thuộc đoạn [0,1), sau đó gửi về cho process đóng vai trò slave. Process này chịu trách nhiệm nhân ma trận này với ma trận nghịch đảo rồi gửi chuẩn-2 của ma trận tích này về cho master để xuất ra màn hình.

- Chương trình được viết như sau:

```
import numpy as np
import multiprocessing as mp

N = 10
M = 10

def master( conn ):
  ma_a = np.random.random((N,M))
  conn.send(ma_a)
  norm = conn.recv()
  print( "Norm is %20.18f" % norm )
  conn.close()
```



```
def slave( conn ):
  sl_b = conn.recv()
  norm2 = np.linalg.norm(np.dot(sl_b,np.linalg.inv(sl_b)),2)
  conn.send( norm2 )
  conn.close()

ma_conn, sl_conn = mp.Pipe()
ma_p = mp.Process( target = master, args = (sl_conn,) )
sl_p = mp.Process( target = slave, args = (ma_conn,) )
ma_p.start()
sl_p.start()
ma_p.join()
sl_p.join()
```


Sử dụng trong Khoa học dữ liệu

- Có hai áp dụng đa xử lý trong Khoa học dữ liệu:
  - Xử lý nhập xuất: Thông thường việc ghi dữ liệu lên Data Warehouses nào đó thường mất nhiều thời gian hơn là việc đọc dữ liệu để đưa vào xử lý. Nên việc ghi được thực hiện song song sẽ hiệu quả.
  - Xây dựng mô hình huấn luyện: không phải tất cả các mô hình được huấn luyện theo cách song song, nhưng một số mô hình có thể thực hiện theo cách này. Chẳng hạn Random Forest được triển khai trên nhiều cây quyết định (Decision Tree) để lấy một quyết định tích luỹ

- Cũng lưu ý, trong thư viện sklearn có cung cấp tham số n_jobs giúp cho việc sử dụng nhiều task.

- Ví dụ minh hoạ: giả sử có công việc dowork() cần giải quyết trong 2 lần.

```
1  import time
2
3  def dowork():
4      print('Starting to do')
5      for i in range(100000000):
6          a = i
7      print('Done doing')
8
9  t0 = time.time()
10 dowork()
11 dowork()
12 t1 = time.time()
13
14 print('Elapsed time in %f seconds' %(t1-t0) )
```

```
Starting to do
Done doing
Starting to do
Done doing
Elapsed time in 5.702057 seconds
```


- Sử dụng Multi-Processing