

```
#pragma omp parallel for num_threads(NT)
for (i=imin;i<imax;i++) coll[i-imin] = dt*(coll_term_f (i,I, F,g));
if (mpi_rank > 0) {
  MPI_Send(coll,N1,MPI_DOUBLE,0,0,MPI_COMM_WORLD)
} else {
  while (count < mpi_size) {
    MPI_Recv(tmp,N1,MPI_DOUBLE,MPI_ANY_SOURCE,0,MPI_COMM_WORLD,&mpi_status);
    sender = mpi_status.MPI_SOURCE;
    count++;
  }
}
```

Introduction to parallel programming (for physicists)

FRANÇOIS GÉLIS & GRÉGOIRE MISGUICH, IPhT courses, June 2019.



université
PARIS-SACLAY

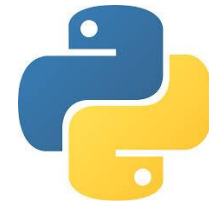


IPhT, CEA-Saclay
Itzykson room
courses.ipht.fr

1. Introduction & hardware aspects (FG)
2. A few words about Maple & Mathematica
3. Linear algebra libraries
4. Fast Fourier transform
5. Python Multiprocessing
6. OpenMP
7. MPI (FG)
8. MPI+OpenMP (FG)

These slides (GM)

Parallel programming in Python



Threads *versus* Processes

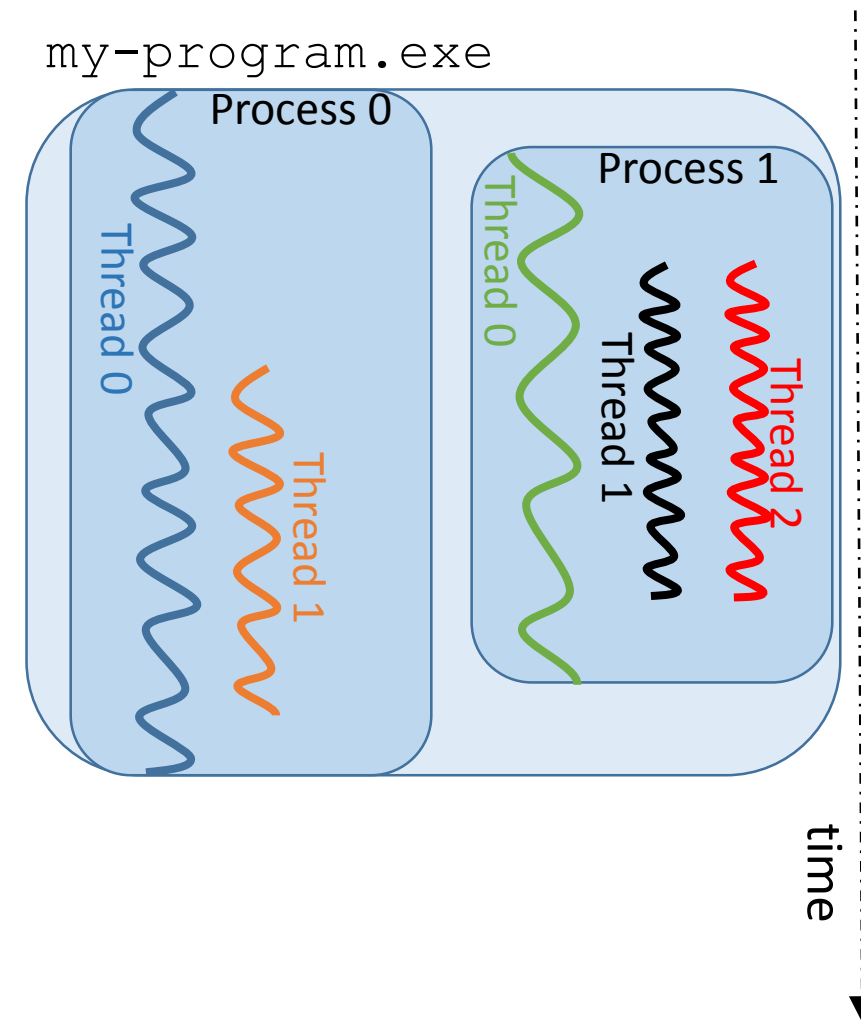
A given application/program may run different processes and different threads

Process

- Each process has its own memory space.
→ they are somewhat independent
- Switching from one process requires some interaction with the operating system → slow switching

Thread

- Threads (associated to a given process) share the same memory space
→ Threads can share information easily /quickly
- There is no memory “protection” between the threads of the same process → responsibility of the programmer
- Threads have little information of their own
→ faster to create than processes



Example of application using several processes

The screenshot shows the Windows Task Manager window titled "Gestionnaire des tâches de Windows". The "Processus" tab is selected, displaying a list of running processes. A red rectangular box highlights a group of 15 processes, all of which are instances of "Brave Browser" (brave.exe) running under the user "misguich". The processes are listed with their names, user names, processors, and memory usage. Below the list, there is a checkbox for "Afficher les processus de tous les utilisateurs" which is checked, and a button labeled "Arrêter le processus". At the bottom of the window, system statistics are displayed: "Processus : 129", "UC utilisée : 28%", and "Mémoire physique : 63 %".

Nom de l'image	Nom d'uti...	Processeur	Mémoire (jeu de travail p...	Description
devmonsrv.exe ...	Système	00	3 012 K	Bluetooth Device Monitor
mediasrv.exe *32	Système	00	3 360 K	Bluetooth Media Service
obexsrv.exe *32	Système	00	2 448 K	Bluetooth OBEX Service
mDNSResponde...	Système	00	2 504 K	Bonjour Service
brave.exe	misguich	00	11 640 K	Brave Browser
brave.exe	misguich	00	84 584 K	Brave Browser
brave.exe	misguich	00	10 932 K	Brave Browser
brave.exe	misguich	00	33 408 K	Brave Browser
brave.exe	misguich	00	16 676 K	Brave Browser
brave.exe	misguich	00	14 060 K	Brave Browser
brave.exe	misguich	00	32 820 K	Brave Browser
brave.exe	misguich	00	68 580 K	Brave Browser
brave.exe	misguich	00	13 788 K	Brave Browser
brave.exe	misguich	00	3 524 K	Brave Browser
brave.exe	misguich	00	3 860 K	Brave Browser
brave.exe	misguich	00	84 396 K	Brave Browser
brave.exe	misguich	00	1 964 K	Brave Browser
brave.exe	misguich	00	32 992 K	Brave Browser
brave.exe	misguich	00	8 560 K	Brave Browser
DDVDataCollect...	Système	00	12 308 K	Dell Data Vault Data Collector Service
DDVCollectorSv...	Système	00	1 580 K	Dell Data Vault Data Collector Service API
DDVRulesProce...	Système	00	6 728 K	Dell Data Vault Rules Processor
DSAPI.exe	Système	00	26 600 K	DSAPI.exe
EEventManager...	misguich	00	2 572 K	EEventManager Application
EPCP.exe	Système	00	5 740 K	Epson Customer Participation
escsvc64.exe	Système	00	1 744 K	Epson Scanner Service (64bit)
E_YATIPKE.EXE	misguich	00	4 032 K	EPSON Status Monitor 3
explorer.exe	misguich	01	43 764 K	Explorateur Windows
FUFAXRCV.exe ...	misguich	00	5 740 K	Fax Reception
FUFAXSTM.exe...	misguich	00	8 460 K	Fax Transmission

Parallel programming in Python

- **Thread library** (`import threading`)

Can start several threads, but they will not run simultaneously (because of the Global Interpreter Lock – a.k.a. GIL).

Can be useful for some I/O tasks (because the CPU will be waiting for some remote server, etc.), but not really for computations.

... will not be discussed here.

- **Multiprocessing library** (`import multiprocessing`)

Allows to perform tasks simultaneously (using *processes* instead of *threads*)

We will present a few examples using:

- `Process`, `Queue`
- `Pool`, `map`, `imap`

Python/Multiprocessing

Multiprocessing.Process

```
import os, math
from multiprocessing import Process, Queue

def my_function(r,q):
    proc = os.getpid()
    i1=r[0]**3
    i2=r[1]**3
    print('Process #{0} will sum from {1}
to {2}'.format(proc,i1,i2))
    sum=0.0
    for i in xrange (i1,i2):
        sum+=math.sin(i)
    q.put((i1,i2,sum))

ranges = [ [1,100], [201,300],
[300,400],[401,500] ]
list_of_procs = []
q=Queue()
for r in ranges:
    p = Process(target=my_function,
args=(r,q))
    list_of_procs.append(p)
    p.start()
for p in list_of_procs:
    p.join()

results=[q.get() for p in list_of_procs]
print(results)
```

q: Object where each process can write its result

Python/Multiprocessing

Multiprocessing.Process

Processes do **not** share memory, which means that the global variables are *copied*, hence their value in the original process do not change.

This example *does not work* (i.e. output is [0,0,0,0])

```
import os, math
from multiprocessing import Process

A=[0,0,0,0]

def my_function(i):
    A[i]=A[i]+1
    return

list_of_procs = []
for r in range(4):
    p = Process(target=my_function, args=(r,))
    list_of_procs.append(p)
    p.start()

for p in list_of_procs:
    p.join()

print(A)
```


Python/multiprocessing

Multiprocessing.Pool



```
import numpy as np
import multiprocessing as mp
import numpy.random as npr
from sympy import *

def gsrn(i):
    n=200
    A = npr.randn(n,n)
    ev=np.linalg.eigvals(A)
    return i, ev[0].real

p=mp.Pool(processes=4)

print(p.map(gsrn, range(9)))

results=p.imap_unordered(gsrn, range(9))
for r in results:
    print(r)
```

Blocks until all
task are finished

Does not block

Python/multiprocessing

Multiprocessing.Pool & linear algebra:
processes versus threads

```
import numpy as np
import multiprocessing as mp
import numpy.random as npr
import time

def smallest_ev(M):
    ev=np.linalg.eigvals(M)
    return ev[0].real

p=mp.Pool(processes=2)
#Two random matrices:
n=2000
A=npr.randn(n,n)
B=npr.randn(n,n)
t0 = time.time()
```

```
results=p.imap_unordered(smallest_ev,
[A,B])
t1 = time.time() - t0
print(" Time after .imap_unordered=%0.4f
s" % (t1))
for r in results:
    print (r)
t1 = time.time() - t0
print(" Time after print results=%0.4f s"
% (t1))

t0 = time.time()
print(smallest_ev(A))
print(smallest_ev(B))
t1 = time.time() - t0
print(" Sequential time=%0.4f s" % (t1))
```

Python/multiprocessing

Multiprocessing.Pool &
linear algebra: processes versus threads

2 workers (=processes), each one
using ~12 threads (and 12 CPU cores)
[mutli-threaded linear algebra lib.]

```
misguich@totoro:~  
Tasks: 587 total,  3 running, 582 sleeping,  2 stopped,  0 zombie  
Cpu(s):  2.6%us,  0.1%sy,  0.3%ni, 96.5%id,  0.6%wa,  0.0%hi,  0.0%si,  0.0%st  
Mem:  65875860k total,  65502344k used,  373516k free,  570020k buffers  
Swap:  65535996k total,  2529000k used,  63006996k free,  58159936k cached
```

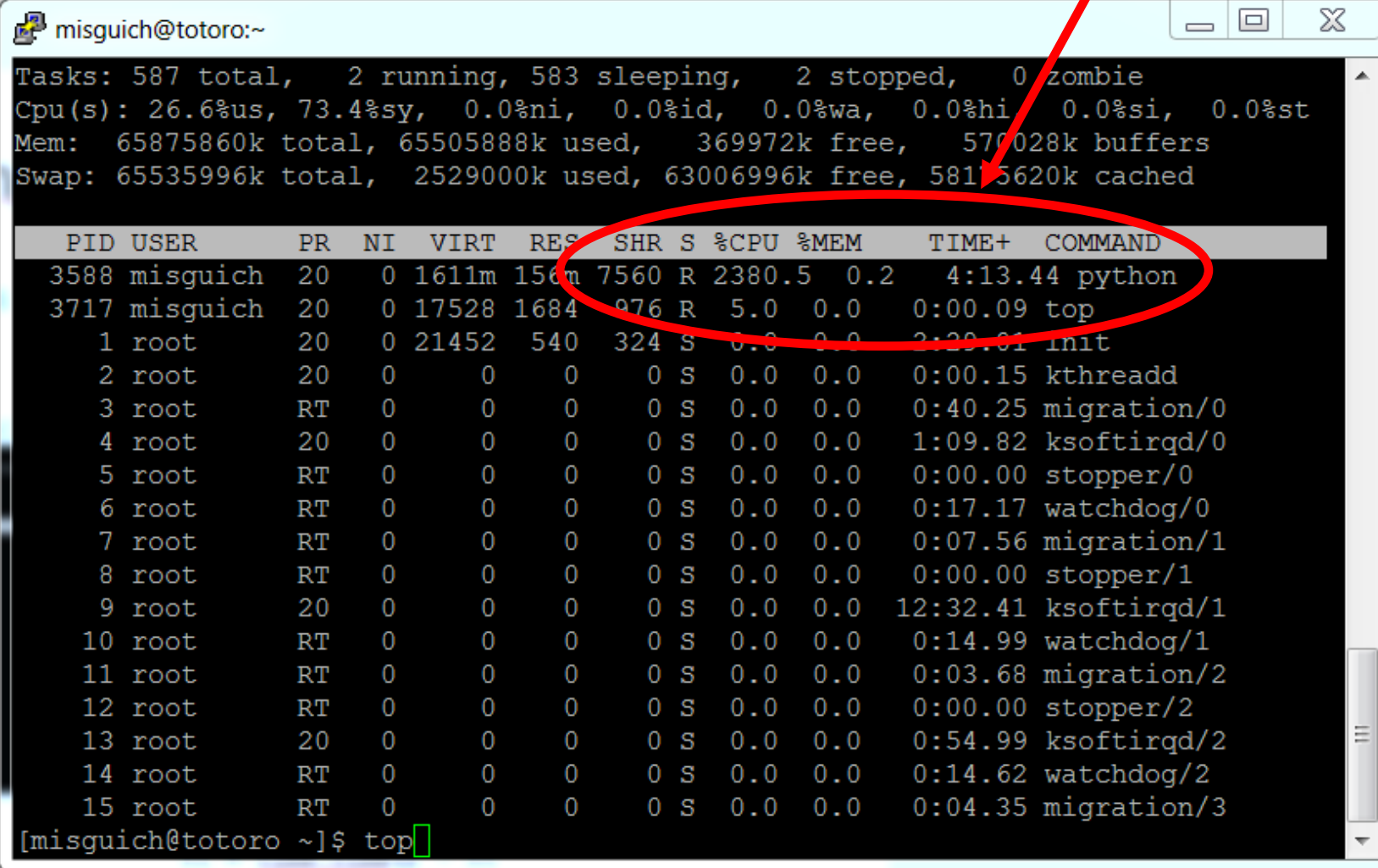
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3613	misguich	20	0	1386m	145m	2148	R	1155.7	0.2	1:05.64	python
3612	misguich	20	0	1386m	145m	2148	R	1150.2	0.2	1:11.95	python
3664	misguich	20	0	17524	1564	860	R	3.7	0.0	0:00.03	top
1	root	20	0	21452	540	324	S	0.0	0.0	2:29.01	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.15	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:40.25	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	1:09.82	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	stopper/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:17.17	watchdog/0
7	root	RT	0	0	0	0	S	0.0	0.0	0:07.56	migration/1
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	stopper/1
9	root	20	0	0	0	0	S	0.0	0.0	12:32.41	ksoftirqd/1
10	root	RT	0	0	0	0	S	0.0	0.0	0:14.99	watchdog/1
11	root	RT	0	0	0	0	S	0.0	0.0	0:03.68	migration/2
12	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	stopper/2
13	root	20	0	0	0	0	S	0.0	0.0	0:54.99	ksoftirqd/2
14	root	RT	0	0	0	0	S	0.0	0.0	0:14.62	watchdog/2

```
[misguich@totoro ~]$
```

Python/multiprocessing

Multiprocessing.Pool &
linear algebra: processes versus threads

Single Python processe
using ~24 threads (and 24 CPU cores)



```
misguich@totoro:~  
Tasks: 587 total,  2 running, 583 sleeping,  2 stopped,  0 zombie  
Cpu(s): 26.6%us, 73.4%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st  
Mem:  65875860k total, 65505888k used,  369972k free,  570028k buffers  
Swap: 65535996k total, 2529000k used, 63006996k free, 5815620k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3588	misguich	20	0	1611m	156m	7560	R	2380.5	0.2	4:13.44	python
3717	misguich	20	0	17528	1684	976	R	5.0	0.0	0:00.09	top
1	root	20	0	21452	540	324	S	0.0	0.0	2:20.01	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.15	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:40.25	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	1:09.82	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	stopper/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:17.17	watchdog/0
7	root	RT	0	0	0	0	S	0.0	0.0	0:07.56	migration/1
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	stopper/1
9	root	20	0	0	0	0	S	0.0	0.0	12:32.41	ksoftirqd/1
10	root	RT	0	0	0	0	S	0.0	0.0	0:14.99	watchdog/1
11	root	RT	0	0	0	0	S	0.0	0.0	0:03.68	migration/2
12	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	stopper/2
13	root	20	0	0	0	0	S	0.0	0.0	0:54.99	ksoftirqd/2
14	root	RT	0	0	0	0	S	0.0	0.0	0:14.62	watchdog/2
15	root	RT	0	0	0	0	S	0.0	0.0	0:04.35	migration/3

```
[misguich@totoro ~]$ top
```

Python/multiprocessing

Multiprocessing.Pool & Sympy

```
import multiprocessing as mp
from sympy import *

p=mp.Pool(processes=3)
x = symbols('x')
print(p.map(integrate, [x, x**2, x**3]))
```

Python/multiprocessing

Multiprocessing.Pool & Sympy

```
import multiprocessing as mp
from sympy import *
import sys

x = symbols('x')
A= [ exp(x),sin(x),cos(x),cosh(x),sinh(x)]
def my_func(f):
    count=0
    while (count<10000) :
        f=diff(f,x)
        count=count+1
    return f
# Pass the wanted number of process as command-
line argument
np=int(sys.argv[1])
p=mp.Pool(processes=np)
results=p.map(my_func,A)
print(results)
```