Introduction to MPI : C, C++ and Python

1 Basic programs

The first examples of a MPI program are C and C++ codes

```c
#include<stdio.h>
#include<mpi.h>
int main (int argc, char *argv[])
{
    int numprocs,myid,namelen;
    char processor_name[
        ↪ MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&
        ↪ numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&
        ↪ myid);
    MPI_Get_processor_name(
        ↪ processor_name,&namelen);
    printf("Hello, Process %d of %d
        ↪ on %s %d\n", myid,
        ↪ numprocs, processor_name,
        ↪ namelen);

    MPI_Barrier( MPI_COMM_WORLD);
    MPI_Finalize();
    exit(0);
}
```

=

```c
#include<iostream>
#include<mpi.h>
int main (int argc, char *argv[])
{
    int numprocs,myid,namelen;
    char processor_name[
        ↪ MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&
        ↪ numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&
        ↪ myid);
    MPI_Get_processor_name(
        ↪ processor_name,&namelen);
    std::cout<<"Hello, Process" <<
        ↪ myid<<" of "<<numprocs<<"
        ↪  on "<< processor_name<<
        ↪ std::endl;
    MPI_Barrier( MPI_COMM_WORLD);
    MPI_Finalize();
    exit(0);
}
```

1. Compile these codes with mpicc (and/or mpiCC) and run by using **mpirun** with a number of virtual cores equal to 2, 4, 8.

   Solution: If the number of physical cores of your computer is less than 8, one must add the option **-oversubscribe** to **mpirun**for running the code. The number of lines correspond to the number of virtual cores.

2. Python is also a language for which you can implement the MPI library. Consider the code **hello_mpi.py**.

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
```

```
processor_name=MPI.Get_processor_name()
print ("hello, Process ",rank," of ",size," on ",processor_name)
```

Run this code in a terminal by using the command un by using **mpirun -n np python3 hello_mpi.py** with a number **np** of virtual cores equal to 2, 4, 8.

Solution: Similar behavior, but as expected, the Python code is simpler.

2 Communications

2.1 Broadcast

These codes illustrate the broadcast communication

```c
#include<stdio.h>
#include<mpi.h>
int main (int argc, char *argv[])
{
    int numprocs,myid,namelen;
    char processor_name[
        ↪ MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&
        ↪ numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(
        ↪ processor_name,&namelen);
    double reel=(double) myid;
    printf("Hello, Process %d of %d on %
        ↪ s valeur de reel %e\n", myid,
        ↪  numprocs, processor_name,
        ↪ reel);
    MPI_Bcast(&reel,1, MPI_DOUBLE,3,
        ↪ MPI_COMM_WORLD);
    MPI_Barrier( MPI_COMM_WORLD);
    printf("Hello, Process %d of %d on %
        ↪ s valeur de reel %e\n", myid,
        ↪  numprocs, processor_name,
        ↪ reel);
    MPI_Finalize();
    exit(0);
}
```

```c
#include<iostream>
#include<mpi.h>
using namespace std;
int main (int argc, char *argv[])
{
    int numprocs,myid,namelen;
    char processor_name[
        ↪ MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&
        ↪ numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&
        ↪ myid);
    MPI_Get_processor_name(
        ↪ processor_name,&namelen);
    double reel=(double) myid;
    cout<<"Before " <<myid<<" of "<<
        ↪ numprocs<<" on "<<
        ↪ processor_name<<" integer
        ↪  value "<<reel<<endl;
    MPI_Bcast(&reel,1, MPI_DOUBLE,3,
        ↪  MPI_COMM_WORLD);
    MPI_Barrier( MPI_COMM_WORLD);
    cout<<"After " <<myid<<" of "<<
        ↪ numprocs<<" on "<<
        ↪ processor_name<<" integer
        ↪  value "<<reel<<endl;
    MPI_Finalize();
    exit(0);
}
```

1. Compile these codes with mpicc (and mpiCC) and run by using mpirun with a number of virtual cores equal to 2, 4, 8.

   Solution: The variable is changed after broadcasting.

2. Python is also a language for which you can implement the MPI library. Consider the code **broadcast.py**.

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
processor_name=MPI.Get_processor_name()
num=10
```

```
numsent = comm.bcast(num, root=0)
print ("hello, Process ",rank," of ",size," on ",processor_name," ",numsent)
```

Run this code in a terminal by using the command un by using **mpirun -n np python3 broadcast.py** with a number **np** of virtual cores equal to 2, 4, 8.

Solution: Similar remark

2.2 Point-to-Point communication

These codes illustrate the point-to-point communication

```
#include<stdio.h>
#include<mpi.h>
int main (int argc, char *argv[])
{
    int numprocs,myid;
    MPI_Status status;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&
        ↪ numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid
        ↪ );
    int small=myid;
    printf("Before, Process %d of %d
        ↪ small= %d\n", myid, numprocs
        ↪ ,small);
    if(myid==0)
    {
        MPI_Send(&small,1,MPI_INT,3,10,
            ↪ MPI_COMM_WORLD);
    }
    if(myid==3)
    {
        MPI_Recv(&small,1,MPI_INT,0,10,
            ↪ MPI_COMM_WORLD,&status);
    }
    MPI_Barrier( MPI_COMM_WORLD);
    printf("After, Process %d of %d
        ↪ small= %d\n", myid, numprocs
        ↪ ,small);
    MPI_Finalize();
}
```

```
#include<iostream>
#include<mpi.h>
using namespace std;
int main (int argc, char *argv[])
{
    int numprocs,myid;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&
        ↪ numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid
        ↪ );

    MPI_Status status;
    int small=myid;
    cout<<"Before " <<myid<<" of "<<
        ↪ numprocs<<" small = "<<small
        ↪ <<endl;
    if(myid==0)
    {
        MPI_Send(&small,1,MPI_INT,3,10,
            ↪ MPI_COMM_WORLD);
    }
    if(myid==3)
    {
        MPI_Recv(&small,1,MPI_INT,0,10,
            ↪ MPI_COMM_WORLD,&status);
    }
    MPI_Barrier( MPI_COMM_WORLD);
    cout<<"After " <<myid<<" of "<<
        ↪ numprocs<<" small = "<<small
        ↪ <<endl;
    MPI_Finalize();
}
```

3. Compile and run the codes with 4, 8 virtual cores. Run with 2 cores. What happens ?

> **Solution:** For 2 cores, the execution fais because the rank 3 does not exist and the program cannot be executed

4. Python is also a language for which you can implement the MPI library. Consider the code **pointtopoint.py**.

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
processor_name=MPI.Get_processor_name()
entier=rank
print ("Before, Process ",rank," of ",size," on ",processor_name," ",entier)
if rank==0:
    comm.send(entier,dest=3, tag=11)
elif rank==3:
    entier=comm.recv(source=0,tag=11)
print ("After, Process ",rank," of ",size," on ",processor_name," ",entier)
```

Run this code in a terminal by using the command un by using **mpirun -n np python3 pointtopoint.py** with a number **np** of virtual cores equal to 2, 4, 8. Note the différences between the C code and the Python for the MPI instruction.

> **Solution:** Similar remark

3 Reduction

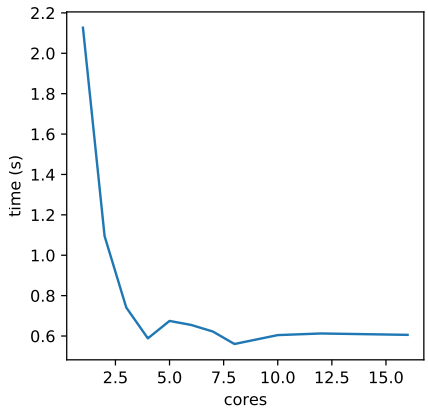
A simple example with a reduction The following code computes the π number by using a numerical evaluation of an integral by a rectangle method. Each virtual core computes a part of the loop and a reduction instruction is performed

```
#include<iostream>
#include <iomanip>
#include<cmath>
#include<mpi.h>
using namespace std;
double f( double a ) {return (4.0 / (1.0 + a*a));}

int main (int argc, char *argv[])
{
    int myid, numprocs;
```

```
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    int n = 1000000000;
    double pi,sum=0.0;
    double startwtime = 0.0;
    if (myid == 0)
    {
        startwtime = MPI_Wtime();
    }
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    for (int i = myid + 1; i <= n; i += numprocs)
    {
        sum += f((i-0.5)/(double) n);
    }
    sum/= (double) n;
    MPI_Reduce(&sum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if (myid == 0)
    {
        cout<<"pi is approximately equal "<<setprecision(16) << pi <<" Error is
            ↪ "<<fabs(pi - M_PI)<<endl;
        cout<<"Wall clock time = "<<MPI_Wtime()-startwtime<<endl;
    }
    MPI_Finalize();
    exit(0);
}
```

1. Compile and run the code.
2. Increase the number of virtual cores from 1 to 8 Collect data and plot the wall time versus the number of virtual cores

> **Solution:** One notes that the time decreases with the number of cores, but saturates when the number of virtual cores becomes larger than the number of physical cores



The graph illustrates this result

3. Open a second terminal and type top. Rerun the program for a thread number of 1, 2 and 4

Solution: When the program is running, one has a number of executables equal to the number of cores

```
from mpi4py import MPI
import numpy as np
PI=np.arccos(-1)
def comp_pi(n, myrank=0, nprocs=1):
    h = 1.0 / n
    j=np.arange(1+ myrank , n + 1, nprocs)
    return (4*h*np.sum(1.0/ (1.0 +(h*(j-0.5))**2),dtype=float) )


comm = MPI.COMM_WORLD
nprocs = comm.Get_size()
myrank = comm.Get_rank()

n = 100000000
pi = np.array(0, dtype=float)
mypi = np.array(0, dtype=float)


if myrank == 0:
    wt=MPI.Wtime()


mypi=comp_pi(n, myrank, nprocs)
```

```
print("rank mypi",myrank," ",mypi)
comm.Reduce([mypi, MPI.DOUBLE], [pi, MPI.DOUBLE],
               op=MPI.SUM, root=0)
if myrank == 0:
       print ("pi is approximately %.16f, error is %.16f" % (pi, abs(pi - PI
          ↪ )))
       wt2=MPI.Wtime()
       print ("elasped time",wt2-wt)
```

4. Consider the Python code **cpi3.py**.
5. Run the code by increasing the number of virtual cores from 1 to 8
6. Rewrite the code by discarding the numpy module. Conclusion

4 Rare event statistics

Rare events are very important in nature. Global warming generates such events with huge consequences with a strong impact in the future. Statistical study of rare events is quite recent and solvable models are also rare. This exercice aims to illustrate some basic concepts. Since a good accuracy in simulation is very demanding, a second part of the exercice consists in using MPI for speed up the simulation.

One considers a exponential distribution of random numbers. Select **1000** random numbers et store the maximum value in an array. Perform **50000** independent simulation and build an histogram of all maxima. Compare your simulation results by using the gumbel function defined as

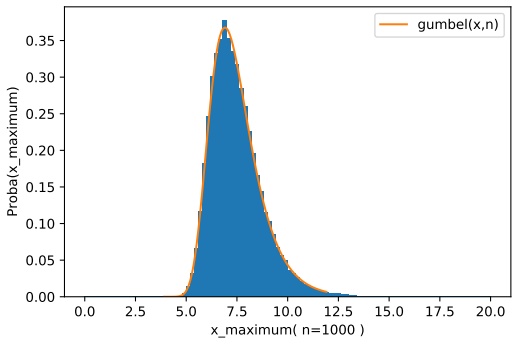
$$gumbel(x,n) = \exp(-x + \ln(n) - e^{-x+\ln(n)}) \tag{1}$$

1. Write a code (Python or C or C++) performing a simulation.

Solution:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 17 18:22:28 2021

@author: viot
"""
import numpy as np
import matplotlib.pyplot as plt
import time
def gumbel( x, n):
    return np.exp( -x +np.log(n) - np.exp(-x+np.log(n)) )
```

```
nrep = 50000 # Independent runs
maxima = np.empty( nrep )

n=1000 # Number of random numbers per simulation
start=time.time()
for i in range( nrep ):
    tab = np.random.exponential( size=n )
    maxima[i] = tab.max()
end=time.time()
print("elapsed time",end-start)

plt.hist( maxima, density=True, bins=100, range=(0.,20.))
plt.xlabel("x_maximum( n=%d )"%n )
plt.ylabel("Proba(x_maximum)" )

xvect = np.linspace( np.log(n)-3, np.log(n)+5, 100)
plt.plot( xvect, gumbel(xvect, n) , '-', label="gumbel(x,n)")
# plt.ylim( 0, 1.)
plt.legend( )

plt.savefig("gumbel.pdf")
plt.show()
```



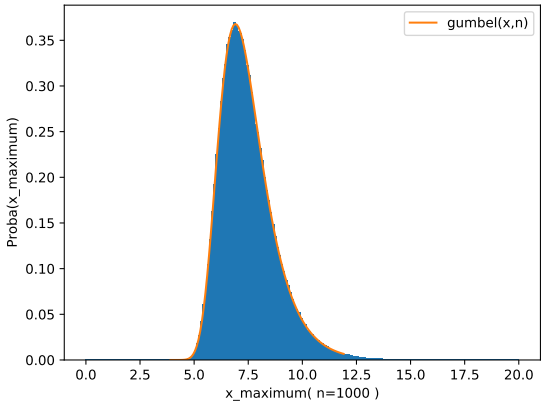
2. Write a parallel code and collect data by using the gather function. (Take care in Python, the function is different than in C or C++).

Solution:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

```
Created on Tue Aug 17 18:22:28 2021

@author: viot
"""
from mpi4py import MPI
import numpy as np
import matplotlib.pyplot as plt
import time
def gumbel( x, n):
    return np.exp( -x +np.log(n) - np.exp(-x+np.log(n)) )


comm = MPI.COMM_WORLD
nprocs = comm.Get_size()
myrank = comm.Get_rank()

nrep = 500000 # Independent runs

maxima = np.empty( nrep//nprocs )

n=1000 # Number of random numbers per simulation
if myrank==0:
    start=time.time()

for i in np.arange(0, nrep//nprocs ):
    tab = np.random.exponential( size=n )
    maxima[i] = tab.max()
#print("myrank",myrank,maxima)
maxima2=comm.gather(maxima, root=0)
maxima2=np.array(maxima2).flatten()
if myrank==0:
    end=time.time()
    print("elapsed time",end-start)
# print("myrank",myrank,maxima2)
    plt.hist( maxima2, density=True, bins=200, range=(0.,20.))
    plt.xlabel("x_maximum( n=%d )"%n )
    plt.ylabel("Proba(x_maximum)" )

    xvect = np.linspace( np.log(n)-3, np.log(n)+5, 100)
    plt.plot( xvect, gumbel(xvect, n) , '-', label="gumbel(x,n)")
# plt.ylim( 0, 1.)
    plt.legend( )

    plt.savefig("gumbelMPI.pdf")
```

```
#plt.show()
```



3. Execute your MPI code by using an increasing number of cores. (When the number of cores exceeds to number of physical core, add the option -oversubscribe with mpirun). Plot the execution time versus of the number of cores in the range $[1, 8]$.