Introduction to MPI

1 Basic programs

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

```cpp
#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);
}
```

```c
#include<stdio.h>
#include<mpi.h>
char processor_name[
    ↪ MPI_MAX_PROCESSOR_NAME];
int numprocs,myid,namelen;
int main (int argc, char *argv[])
{

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();
}
```

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

2 Communications

2.1 Broadcast

These codes illustrate the broadcase communication

```
#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();
}
```

```
#include<stdio.h>
#include<mpi.h>
char processor_name[
    ↪ MPI_MAX_PROCESSOR_NAME];
int numprocs,myid,namelen;
MPI_Status status;
int main (int argc, char *argv[])

{
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();
}
```

1. Compile these codee with mpicc (and mpiCC) and run by using mpirun with a number of virtual cores equal to 2, 4, 8.
2. Python is also a language for which you can implement the MPI library. Consider the code **comm_mpi.py**.

```
from mpi4py import MPI

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

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

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

2.2 Point-to-Point communication

These codes illustrate the point-to-point communication

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

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

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

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>
double f( double a ) { return (4.0 / (1.0 + a*a)); }
int main (int argc, char *argv[])
{
int myid, numprocs,namelen;
double pi, sum=0.0;
double startwtime = 0.0;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
int n = 1000000000;
double h = 1.0 / (double) n;
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){
double x = h * (i - 0.5);
sum += f(x);}
double mypi = h * sum;
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
if (myid == 0){

std::cout<<"pi␣is␣approximately␣equal␣"<<std::setprecision(16) << pi <<"␣␣Error␣
    ↪ is␣"<<fabs(pi - M_PI)<<std::endl;
std::cout<<"Wall␣clock␣time␣=␣"<<MPI_Wtime()-startwtime<<std::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
3. Open a second terminal and type top. Rerun the program for a thread number of 1, 2 and 4
4. Consider the Python code **cpi.py**.

```
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(myrank + 1, n + 1, nprocs)
        return (4*h*np.sum(1.0/ (1.0 +(h*(j+0.5))**2)))

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.fill(comp_pi(n, myrank, nprocs))
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)
```

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