Assignment Description

1. MPI Execution Environment

The goal of this part of the assignment is becoming familiar with the environment and be able to compile and execute MPI programs.

We will use the following MPI program, which implements the typical "hello world" program (hello.c).

```
#include "mpi.h"
#include <stdio.h>

int main(int argc, char **argv)
{
  MPI_Init(&argc, &argv);

  printf("Hello World!\n");

  MPI_Finalize();
}
```

You will note that you need to include "mpi.h" and that it is required to start the MPI programs with `MPI_Init` and end it them with `MPI_Finalize`. These calls are responsible for interfacing with the MPI runtime for creating and destroying the MPI processes.

The following items illustrate the main necessary steps to compile and execute the "hello world" using MPI:

- Compile the program `hello.c` with mpicc (more details can be found in the documentation).
- Execute the hello world MPI program through SGE. A sample script for the execution of the hello world program using 8 MPI processes is provided below.

```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N hello
#$ -o hello.out.$JOB_ID
#$ -e hello.err.$JOB_ID
#$ -pe orte 8

mpirun -np 8 ./hello
```

The "-pe orte 8" option indicates that it is necessary 8 cores for the execution of the job (these will be assigned to more than one node un the UOC cluster). The mpirun command interfaces with the MPI runtime to start the execution of the program. The "-np 8" option indicates the MPI runtime to start 8 processes and the name of the program (binary) is provided at the end of the command. You can explore additional mpirun options by yourself.

Questions:

1. **What is the result of the execution of the MPI program discussed above? Why?**
2. **Explain what happens if you use the mpirun option "-np 4" instead of "-np 8" with the example above.**

2. MPI processes

The code shown below includes the typical library calls in MPI programs. Note that the variable "rank" is essential in the execution of MPI programs as it allows identifying the number of MPI process within a communicator (in this case MPI_COMM_WORLD). Please check the documentation for additional information.

```
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  int rank, numprocs;
  char hostname[256];

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);

  gethostname(hostname,255);

  printf("Hello world! I am process number %d of %d MPI processes on host %s\n", rank,
numprocs, hostname);

  MPI_Finalize();

  return 0;
}
```

Questions:

3. **What is the result of the execution of the MPI program above? Why?**

3. MPI point to point communications

In this section we address the first programming exercise. Another example of an MPI program ("hellompi.c") is provided along with this assignment description. This program is designed to be executed with only 2 MPI processes and is responsible for sending messages between the two MPI processes.

You are requested to provide a variant of the "hellompi.c" program such that each of the MPI processes send an MPI message to the other processes.

An output example with 3 MPI processes is shown below:

```
Proc #1 sending message to Proc #0
Proc #1 sending message to Proc #2
Proc #1 received message from Proc #0
Proc #1 received message from Proc #2
Proc #0 sending message to Proc #1
Proc #0 sending message to Proc #2
Proc #0 received message from Proc #1
Proc #0 received message from Proc #2
Proc #2 sending message to Proc #0
Proc #2 sending message to Proc #1
Proc #2 received message from Proc #0
Proc #2 received message from Proc #1
```

Questions:

4. **Provide the code of the variant of hellompi.c as described above.**

4. Order of the write messages in the standard output

In this section you are requested to implement an MPI program that passes messages between MPI processes following a ring, i.e., each MPI process sends a message to the next rank and receives one from the previous rank (except the first and last to close the ring).

Given N processes MPI, it is expected that the MPI process with rank 0 will send a message to the process with rank 1, the process with rank 1 will receive the message the process with rank 0 and send of a message in the process with rank 2, and so on until reaching the process with rank N-1 that you will receive a message from the process N-2 and will send a message to the process with rank 0.

You can use the following line as an example for the output of your program.

```
Proc 2: received message from proc 1 sending to 3
```

Questions:

5. **Provide the code of your ring program using MPI.**
6. **What is the order of the messages in the output? Why?**

5. Basic MPI programming

In this section you are requested to provide different variants of a simple problem using MPI and to parallelize a sequential code as described below.

Questions:

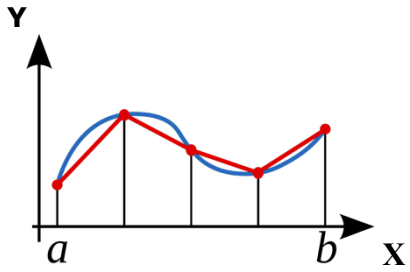
7. **Provide three simple parallel programs to sum a set of random numbers based on the variants described below. Each MPI process shall generate and sum a subset of random numbers locally, and then compute the grand total by adding the local sum from each process, following the following strategies:**

   7.a: a "master" process receives the local sums using MPI_Send.

   7.b: a "master" process receives the local sums using MPI_Gather.

   7.c: a "master" process receives the local sums using MPI_Reduce.

File "p8.c" provides the code implementing the trapezoidal rule for numerical integration. It is used to approximate the area between the graph of a function y=f(x), two vertical lines, and the x-axis. In the figure above we use n=4 subintervals.



If the endpoints of the subinterval are x_i and x_{i+1}, then the length of the subinterval is $h=x_{i+1}-x_i$. Further, if the lengths of the two vertical segments are $f(x_i)$ and $f(x_{i+1})$, then the area of the trapezoid is: $h/2\ [f(x_i)+f(x_{i+1})]$. Since we chose the n subintervals so that they would all have the same length, we also know that if the vertical lines bounding the region are x=a and x=b, then h=(b-a)/n.

If we call the leftmost endpoint x_0, and the rightmost endpoint x_n: $x_0=a$, $x_1=a+h$, $x_2=a+2h$,…, $x_{n-1}=a+(n-1)h$, $x_n=b$. Thus, the sum of the areas of the trapezoids (total area approximation) is: $h[f(x_0)/2 + f(x_1)+f(x_2)+...+f(x_{n-1})+f(x_n)/2]$.

The code provided in p8.c uses the "fragment" function, which can be useful for your implementation.

Questions:

8. **Provide a parallel version of p8.c using MPI. Explain your design decisions.**

6. Performance evaluation tools (TAU)

This section is focused on the performance evaluation of the MPI program implemented in question 8.

For this part of the assignment we will use TAU, which is a profiling and tracking toolkit for performance evaluation (https://www.cs.uoregon.edu/research/tau/home.php).

This package can be used at the user level and can be found in the following path in the UOC cluster:

```
/export/apps/tau/ (login node, /share/apps/tau/ in the compute nodes)
```

You will also need to add the following paths to the $PATH variable:

```
export PATH=$PATH:/export/apps/tau/x86_64/bin (login node)
export PATH=$PATH:/share/apps/tau/x86_64/bin (compute node)
```

The TAU package provide the necessary tools for the instrumentation and analysis of parallel applications as well as the jumpshot trace visualizer. The basic steps to use TAU and the tools associated with the package are provided below.

- **Recommended use of TAU for the assignment (dynamic instrumentation)**

In your SGE script you will need to use the following:

```
export TAU_PROFILE=1
export TAU_TRACE=1
(activates tracing)

mpirun -np NUM_PROCS tau_exec ./YOUR_MPI_PROGRAM
(dynamic instrumentation - recommended)
```

- **Analysis with pprof (you can explore additional tools, such as paraprof, if you would like to perform a more in-depth study on your own)**

You only need to execute the pprof command in the directory where your MPI program has been executed and where files named "`profile.*.*.*`" have been generated. A basic example is shown below.

```
[ivan@eimtarqso p4]$ pprof
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        3,000        3,015           1          12    3015921 .TAU application
  0.3           10           10           1           0      10235 MPI_Init()
  0.1            3            3           1           0       3450 MPI_Finalize()
  0.0        0.863        0.863           2           0        432 MPI_Barrier()
  0.0        0.545        0.545           3           0        182 MPI_Send()
  0.0        0.018        0.018           3           0          6 MPI_Recv()
  0.0        0.001        0.001           1           0          1 MPI_Comm_rank()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()

NODE 1;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        3,000        3,015           1          12    3015923 .TAU application
  0.3           10           10           1           0      10190 MPI_Init()
  0.1            3            3           1           0       3470 MPI_Finalize()
```

You can find the options for this command as follows:

```
usage: pprof [-c|-b|-m|-t|-e|-i|-v] [-r] [-s] [-n num] [-f filename] [-p] [-l] [-d] [node
numbers]
 -a : Show all location information available
 -c : Sort according to number of Calls
 -b : Sort according to number of suBroutines called by a function
 -m : Sort according to Milliseconds (exclusive time total)
 -t : Sort according to Total milliseconds (inclusive time total)  (default)
 -e : Sort according to Exclusive time per call (msec/call)
 -i : Sort according to Inclusive time per call (total msec/call)
 -v : Sort according to Standard Deviation (excl usec)
 -r : Reverse sorting order
 -s : print only Summary profile information
 -n <num> : print only first <num> number of functions
 -f filename : specify full path and Filename without node ids
 -p : suPpress conversion to hh:mm:ss:mmm format
 -l : List all functions and exit
 -d : Dump output format (for tau_reduce) [node numbers] : prints only info about all
contexts/threads of given node numbers
```

- **Trace visualization with jumpshot**

In order to better understand the possible problems related to message passing in MPI, it is often necessary to visualize the traces of the execution of programs and study them with tools that can help analyze these traces. It is worth noting that such performance evaluation techniques are typically an art that requires in-depth analysis. This assignment focus on the most basic features to familiarize with these tools.

TAU comes with the jumpshot tool, which you can use following the steps shown below.

```
tau_treemerge.pl
tau2slog2 tau.trc tau.edf -o tau.slog2
jumpshot tau.slog2
```

Using jumpshot directly to the UOC cluster can lead to very high response times and may be impractical. It is therefore recommended that you download the tool in your local environment. These tools are part of the TAU package in the following directories:

```
/export/apps/tau/x86_64/bin/tau2slog2 (login node)
/export/apps/tau/x86_64/lib/jumpshot.jar (login node)
```

You just need to have the Java runtime environment installed on your system (Linux, mac or Windows) to use the tool.

Questions:

9. **Perform a brief evaluation of your parallel implementation of p8.c using the tools introduced above. You can provide a summary of statistics, screenshots, and the comments that you consider appropriate.**

7. Performance evaluation tools (Extrae/Paraver)

This section focuses on a performance study of some of the NAS Parallel Benchmarks (NPB) - https://www.nas.nasa.gov/publications/npb.html. Specifically, you are requested to study the EP, BT and CG benchmarks (class S). The motivation behind this exercise is to understand how to perform a performance study of an applications based on their binaries and not having necessarily an understanding of their behavior.

The binaries are in the following directories of the login node:

```
/export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/ep.S.16
/export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/bt.S.16
/export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/cg.S.16
```

The binaries above are the smallest class NPBs and are designed to run with 16 MPI processes.

In this case we will use Extrae (https://tools.bsc.es/extrae), which is part of the toolkit developed by the Barcelona Supercomputing Center.

In order to use extrae you must define the following variables and execute the following:

```
export EXTRAE_HOME=/export/apps/extrae
cp /export/apps/extrae/share/example/MPI/extrae.xml YOUR_DIRECTORY
source /export/apps/extrae/etc/extrae.sh
export EXTRAE_CONFIG_FILE=YOUR_DIRECTORY/extrae.xml
export LD_PRELOAD=/export/apps/extrae/lib/libmpitrace.so
```

then, you can execute your MPI programs as usually, for example:

```
mpirun -np 16 ./filename
```

As a result of the execution of the instrumented application you will obtain a trace which is composed of three files: `filename.prv`, `filename.row` i `filename.pcf`.

In order to visualize and analyze the obtained trace you will use Paraver (https://tools.bsc.es/paraver). This tool is available for multiple platforms and you will need to install it in your system.

Questions:

10. **Perform a brief comparative study of the NPB benchmarks introduced above (EP, BT and CG) using Extrae/Paraver. You can provide a summary of statistics, screenshots, and the comments that you consider appropriate.**

Grading Policy

The correct use of the MPI programming model will be evaluated. The use of the performance evaluation tools in your studies as well as your comments will be taken into account.

Submission Format and Due Date

Please submit your assignment as a single PDF containing all the responses including the scripts, graphs etc. Brevity and concretion will be positively considered to grade the assignment.

Submitting a tar or zip file including files with your implementation is allowed (not preferred), but please do NOT use rar.

Assignments should be submitted by **16 May 2020 (extended, hard deadline)**.