

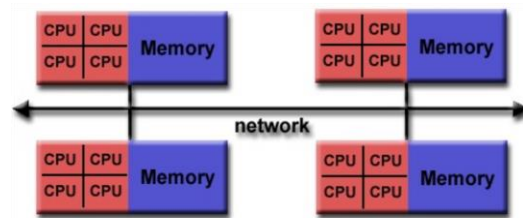
Introducción a MPI

Cómputo Concurrente

Introducción

OpenMP proporciona un conjunto de directivas de compilación que permiten a los programadores especificar la "región paralela". OpenMP asume que todos los hilos tienen acceso a todos los datos (modelo de datos compartidos). Por otro lado, MPI significa paso de mensajes, la memoria no necesita compartirse, como la red IO, la memoria en diferentes procesadores. MPI proporciona un conjunto de API que especifica el remitente y el receptor del mensaje y el problema específico en el que está trabajando cada trabajador. OpenMP es muy fácil de usar, pero MPI puede ser un poco difícil.

MPI se ha desarrollado en las últimas décadas. Tiene una variedad de aplicaciones en la industria y la investigación. La estructura de memoria de MPI es flexible: la memoria puede ser distribuida o compartida, puede estar en la GPU o en la CPU, etc. En una palabra, MPI ocurre dondequiera que se deba comunicar la información.



Objetivos de la Practica

1. Comprender la estructura de los programas escritos en MPI.
2. Diseñar e implementar programas más sencillos escritos en MPI.

Metodología de la Practica

Para alcanzar los objetivos presentados en la Sección anterior, se deberán seguir los siguientes pasos:

1. Instalación los paquetes de software o módulos indicados en la sección 3.
2. Ejecución del programa concurrente `hello_world`
3. Implementación del procedimiento paralelo `ping_pong`
4. Diseño e implementación del procedimiento `statistics`

Para realizar el paso uno, el alumno debe seguir las instrucciones de instalación indicadas en [1]. Después, debe bajar del repositorio del curso el código `hello_mpi.py`, el cual se encuentra en el `code/hello_mpi`. Enseguida, el alumno estudiará el código, poniendo especial énfasis en los comentarios. Después, el alumno ejecutará el código y observará su salida.

Teniendo como base el código `hello_mpi.py` y con ayuda del profesor, el alumno implementará el pseudocódigo `ping_pong` mostrado a la izquierda para lograr el paso tres. El algoritmo trata de dos tareas que generan un número entre cero y uno de tal manera que la que genera el número más grande gana. Las tareas se turnan para enviarse el resultado y registrar el resultado. El algoritmo regresa el identificador del ganador. Después, el alumno ejecutará el código y observará y explicará su salida preguntándose si ésta tiene sentido. Finalmente, el alumno diseñará e implementará un algoritmo `statistics`, el cual recibe un arreglo desordenado de n elementos y regresa el mínimo y el máximo, la mediana, la media y la desviación estándar. Los cálculos se hacen bajo la restricción de que no es posible ordenar el arreglo. Siendo así, el algoritmo que calcula la mediana necesita recibir el máximo y el mínimo, mientras que el que calcula la desviación estándar requiere la media. Con la realización de este paso, el alumno habrá completado el último punto de la metodología.

```

1: parallel_procedure ping_pong( $r$ )
2:   start_distributed_task( $\mathcal{E}$ )
3:    $Sample \leftarrow \langle \emptyset \rangle$ 
4:    $rounds \leftarrow r$ 
5:   if  $|\mathcal{E}| \neq 2$  then
6:     abort()
7:   else
8:     ping_pong( $Sample, rounds$ )
9:   end if
10:   $winner \leftarrow \text{analyze\_sample}(Sample)$ 
11:  return winner
12: end parallel_procedure

```

```

1: task ping_pong( $rounds, Sample$ )
2:   $n \leftarrow 0$ 
3:   $partner_{id} \leftarrow \text{mod}(\text{get\_id}() + 1, 2)$ 
4:  while  $n < rounds$  do
5:     $num \leftarrow \text{get\_rand}()$ 
6:    if  $\text{get\_id}() = \text{mod}(n, 2)$  then
7:      send( $num, partner_{id}, obs$ )
8:    else
9:       $msg \leftarrow \text{recv}(partner_{id}, obs)$ 
10:      $partner_{num} \leftarrow \text{get\_field}(msg, 0)$ 
11:     if  $num > partner_{num}$  then
12:       push_back( $Sample, partner_{id}$ )
13:     else
14:       push_back( $Sample, \text{get\_id}()$ )
15:     end if
16:   end if
17:    $n \leftarrow n + 1$ 
18: end while
19: if  $\text{get\_id}() = 0$  then
20:    $msg \leftarrow \text{recv}(partner_{id}, sample)$ 
21:    $Sample_{partner} \leftarrow \text{get\_field}(msg, 0)$ 
22:   push_back( $Sample, Sample_{partner}$ )
23: else
24:   send( $Sample, partner_{id}, sample$ )
25: end if
26: end task

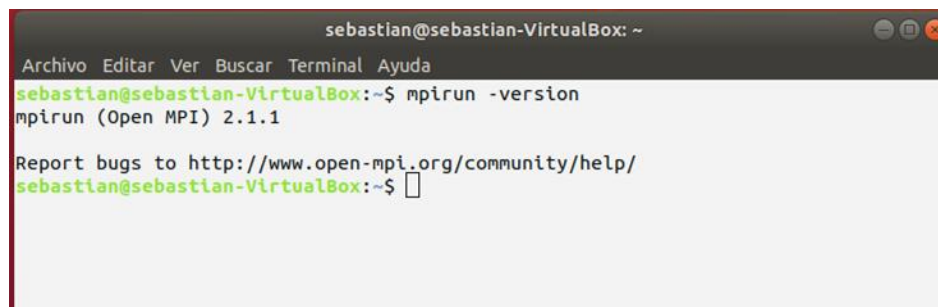
```

Restricciones en la Metodología

Los pasos del uno al tres se realizan individualmente y su resultado debe ser obtenido antes de que la sesión de laboratorio finalice. El último paso se realiza en equipo. Como resultado del último paso, uno de los alumnos del equipo explicará el funcionamiento de la implementación con base en su pseudocódigo. Es sumamente importante que el equipo desarrolle en primera instancia el pseudocódigo antes de la implementación. Si esto no se hace, entonces no se revisará la implementación.

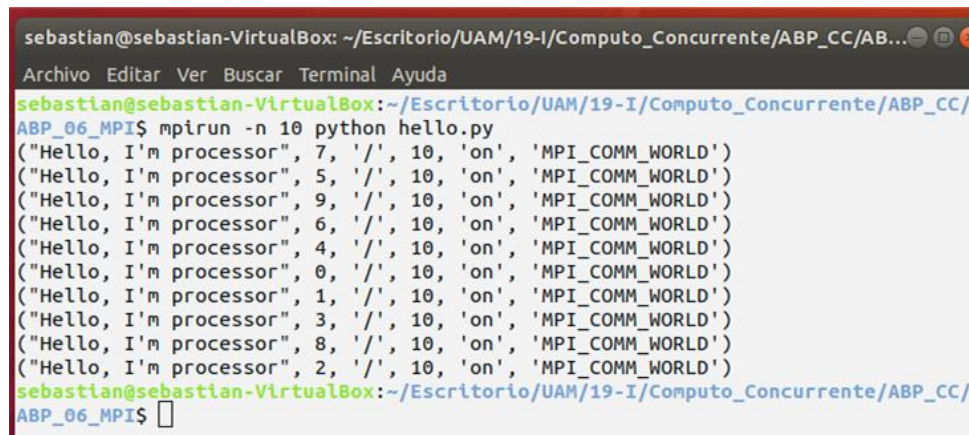
Reporte de Resultados

1. La instalación de los módulos de MPI4py fueron instalados correctamente.

A terminal window titled 'sebastian@sebastian-VirtualBox: ~' with a menu bar (Archivo, Editar, Ver, Buscar, Terminal, Ayuda). The user enters 'mpirun -version' and the output is 'mpirun (Open MPI) 2.1.1'. Below the output is a link to report bugs: 'Report bugs to http://www.open-mpi.org/community/help/'.

```
sebastian@sebastian-VirtualBox: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
sebastian@sebastian-VirtualBox:~$ mpirun -version  
mpirun (Open MPI) 2.1.1  
  
Report bugs to http://www.open-mpi.org/community/help/  
sebastian@sebastian-VirtualBox:~$
```

2. Ejecución del programa concurrente hello_world.py

A terminal window titled 'sebastian@sebastian-VirtualBox: ~/Escritorio/UAM/19-I/Computo_Concurrente/ABP_CC/AB...' with a menu bar (Archivo, Editar, Ver, Buscar, Terminal, Ayuda). The user enters 'mpirun -n 10 python hello.py'. The output shows ten lines of 'Hello, I'm processor', each followed by a unique rank number (0-9) and the MPI_COMM_WORLD identifier.

```
sebastian@sebastian-VirtualBox: ~/Escritorio/UAM/19-I/Computo_Concurrente/ABP_CC/AB...  
Archivo Editar Ver Buscar Terminal Ayuda  
sebastian@sebastian-VirtualBox:~/Escritorio/UAM/19-I/Computo_Concurrente/ABP_CC/  
ABP_06_MPI$ mpirun -n 10 python hello.py  
("Hello, I'm processor", 7, '/', 10, 'on', 'MPI_COMM_WORLD')  
("Hello, I'm processor", 5, '/', 10, 'on', 'MPI_COMM_WORLD')  
("Hello, I'm processor", 9, '/', 10, 'on', 'MPI_COMM_WORLD')  
("Hello, I'm processor", 6, '/', 10, 'on', 'MPI_COMM_WORLD')  
("Hello, I'm processor", 4, '/', 10, 'on', 'MPI_COMM_WORLD')  
("Hello, I'm processor", 0, '/', 10, 'on', 'MPI_COMM_WORLD')  
("Hello, I'm processor", 1, '/', 10, 'on', 'MPI_COMM_WORLD')  
("Hello, I'm processor", 3, '/', 10, 'on', 'MPI_COMM_WORLD')  
("Hello, I'm processor", 8, '/', 10, 'on', 'MPI_COMM_WORLD')  
("Hello, I'm processor", 2, '/', 10, 'on', 'MPI_COMM_WORLD')  
sebastian@sebastian-VirtualBox:~/Escritorio/UAM/19-I/Computo_Concurrente/ABP_CC/  
ABP_06_MPI$
```

3. Implementación del procedimiento paralelo ping_pong.

```
1: parallel_procedure ping_pong(r)
2:   start_distributed_task( $\mathcal{E}$ )
3:   Sample  $\leftarrow \{\emptyset\}$ 
4:   rounds  $\leftarrow r$ 
5:   if  $|\mathcal{E}| \neq 2$  then
6:     abort()
7:   else
8:     ping_pong(Sample, rounds)
9:   end if
10:  winner  $\leftarrow$  analyze_sample(Sample)
11:  return winner
12: end parallel_procedure
```

```
1: task ping_pong(rounds, Sample)
2:   n  $\leftarrow 0$ 
3:   partner_id  $\leftarrow \text{mod}(\text{get\_id}() + 1, 2)$ 
4:   while n < rounds do
5:     num  $\leftarrow$  get_rand()
6:     if get_id() = mod(n, 2) then
7:       send(num, partner_id, obs)
8:     else
9:       msg  $\leftarrow$  recv(partner_id, obs)
10:      partner_num  $\leftarrow$  get_field(msg, 0)
11:      if num > partner_num then
12:        push_back(Sample, partner_id)
13:      else
14:        push_back(Sample, get_id())
15:      end if
16:    end if
17:    n  $\leftarrow n + 1$ 
18:  end while
19:  if get_id() = 0 then
20:    msg  $\leftarrow$  recv(partner_id, sample)
21:    Sample_partner  $\leftarrow$  get_field(msg, 0)
22:    push_back(Sample, Sample_partner)
23:  else
24:    send(Sample, partner_id, sample)
25:  end if
26: end task
```

```

# Import from module mpi4py the object MPI
from mpi4py import MPI

import array

from sys import argv

import random


# Funcion para escoger un ganador
def analyze_sample(sample):
    count = 0
    for i in sample:
        if i == 1:
            count += 1

    # apply floor divition:
    # Devuelve la parte integral del cociente.
    return 1 if count > len(sample)//2 else 0


# Tarea para la simulacion de ping pong.
def ping_pong_(rounds, sample):
    comm = MPI.COMM_WORLD
    rank = comm.rank
    n = 0
    partner = (rank+1)%2
    while n < rounds:
        num = array.array('i',[random.randrange(2)])
        if rank == (n % 2):
            comm.Send([num, MPI.INT], dest = partner, tag = n)
        else:
            partner_num = array.array('i',[0])
            comm.Recv([partner_num, MPI.INT], source = partner, tag = n)
            if num[0] > partner_num[0]:
                # append the array with rank
                sample.append(rank)
            else:
                # apped array with partner
                sample.append(partner)

        n += 1
    if rank == 0:
        sample_partner = array.array('i',[0]*len(sample))

```

```

        comm.Recv([sample_partner, MPI.INT], source = partner, tag = 0)

        sample += sample_partner

    else:

        comm.Send([sample, MPI.INT], dest = partner, tag = 0)

# dar inicio a la partida usando la tarea de ping_pong_task
def ping_pong(r):
    # define the communications

    comm = MPI.COMM_WORLD

    rank = comm.Get_rank()

    size = comm.Get_size()

    # check only two participants
    assert MPI.COMM_WORLD.Get_size() == 2, "Must be two participants"

    # sample array empty
    sample = array.array('i',[])

    rounds = r

    ping_pong_(rounds, sample)

    winner = analyze_sample(sample)

    return winner

# This is the main funcion, don't forget it
if __name__ == "__main__":
    comm = MPI.COMM_WORLD

    rank = comm.Get_rank()

    size = comm.Get_size()

    # get the number of rounds
    r = int(argv[1])

    winner = ping_pong(r)

    if(rank == 0):
        print("The winner is the process " + str(winner))

```

4. Diseño e implementación del procedimiento `statistics`.

Esta implementación es la que recibe un arreglo desordenado de n elementos y regresa el mínimo y el máximo, la mediana, la media y la desviación estándar. Los cálculos se hacen bajo la restricción de que no es posible ordenar el arreglo. Siendo así, el algoritmo que calcula la mediana necesita recibir el máximo y el mínimo, mientras que el que calcula la desviación estándar requiere la media.