Programación Paralela: Práctica 5

22 – 26 abril 2024

Las respuestas deben entregarse en un único documento (formato pdf). Este documento debe incluir el nombre completo de todos los integrantes del grupo. Si el ejercicio es de implementación, el código fuente se puede entregar en ficheros .py independientes dentro de un archivo comprimido.

Instalación de mpi4py

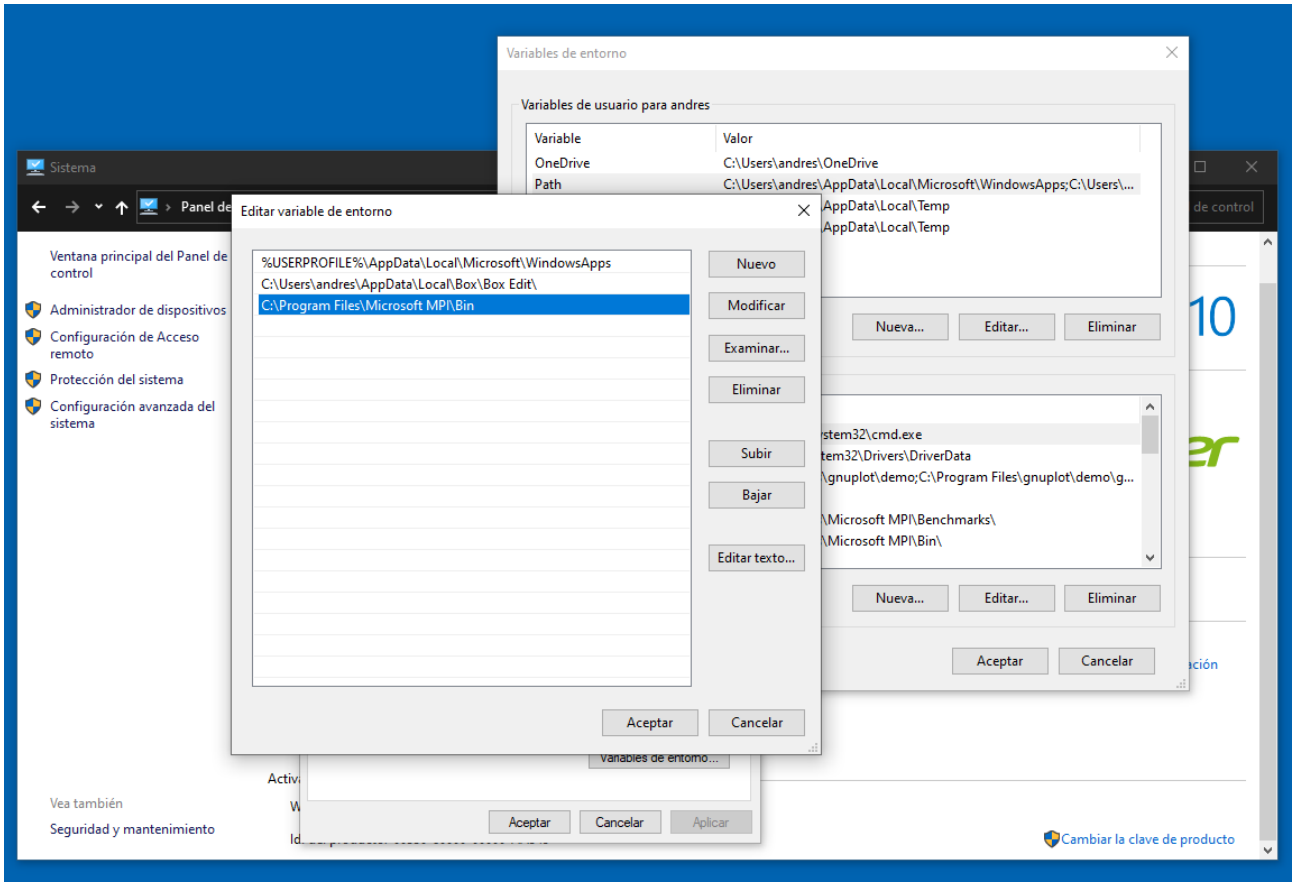
En esta práctica vamos a utilizar el interfaz `mpi4py` con el estándar MPI para intercambio de mensajes entre procesos. Primero hay que instalar una implementación de MPI en el sistema y luego la librería `mpi4py` para Python.

Instalación en Windows

Microsoft proporciona una implementación de MPI que se puede descargar de `https://www.microsoft.com/en-us/download/details.aspx?id=100593`. Atención: hay que instalar el paquete con la extensión `.exe`

Hay añadir los programas de MPI a la ruta del sistema, para ello hay que ir al:

1. Panel de control
2. Icono sistema
3. Configuración avanzada del del sistema
4. Variables de entorno
5. Seleccionar la variable `PATH` y pulsar el botón editar
6. Pulsar en examinar y elegir el directorio `bin` dentro de la instalación de MPI (generalmente `C:\Program Files\Microsoft MPI\Bin`)



Para instalar el `mpi4py` en Anaconda, como no hay un paquete precompilado hay que usar el `pip` desde la consola de Anaconda:

```
pip install mpi4py
```

Dependiendo si Anaconda fue instalado con privilegios de administrador, es posible que el comando anterior necesite los mismos privilegios. Para ejecutar la consola como administrador hay que pulsar con el botón derecho y elegir la opción en el menú desplegable.

Instalación en Linux y Mac OS X

En las distribuciones de Linux suele haber un paquete `python-mpi4py` que se puede instalar con `apt-get` o el administrador de paquetes del sistema operativo. Si Python está instalando con Anaconda solamente hay que instalar el paquete:

```
conda install -c anaconda mpi4py
```

El mismo comando debería funcionar en los sistemas Mac OS X. Si no funciona puedes probar con

```
pip install mpi4py
```

Lanzamiento de programas MPI

Los programas MPI se deben lanzar con el comando `mpiexec` desde la consola del sistema (es posible que en Windows se necesario usar la consola de Anaconda). Ejemplo:

```
mpiexec -np 4 python eje1.py
```

El parámetro -np indica el número de procesos a arrancar, dependiendo de la versión MPI es posible que haya que añadir el parámetro --oversubscribe para poder ejecutar con más procesos que núcleos tiene el procesador. También dependiendo de vuestra instalación es posible que haya que especificar la versión de Python usando el comando python3 en lugar de python.

Ejercicio 1

Todos los programas con `mpi4py` tienen un comienzo similar:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD

rank = comm.Get_rank()

size = comm.Get_size()

print("Hola mundo, soy el proceso ", rank, " de ",
size)
```

Las comunicaciones en MPI se realizan en grupos y `MPI.COMM_WORLD` es el grupo por defecto con todos procesos. En Python, cada grupo es un objeto y las operaciones con un grupo son métodos de ese objeto. Los métodos `Get_rank` y `Get_size` devuelven el número de proceso dentro del grupo (empezando por cero) y el tamaño del grupo.

- Lanza el programa anterior varias veces variando el número de procesos.
- Modifica el programa para que solamente el primer y último proceso muestren el mensaje.

Ejercicio 2

Los mensajes entre procesos se realizan con `send` y `recv`. Por defecto, el envío es asíncrono pero la recepción es bloqueante. En el siguiente programa el proceso 0 envía un mensaje al resto de procesos

```
from mpi4py import MPI


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

```
if rank == 0:
   mensaje_enviado = "Mensaje desde 0"
else:
   mensaje_enviado = ""

if rank == 0:
   for i in range(1,size):
       print("Enviando a", i)
       comm.send(mensaje_enviado, dest=i)
else:
   mensaje_recibido = comm.recv()
   print("Recibido por", rank, ":", mensaje_recibido)
```

Ejecuta el código anterior para familiarizarte con las primitivas send y recv. Una vez tengas claro cómo funcionan realiza el siguiente ejercicio.

- Escribe un programa en Python utilizando MPI que realice la siguiente tarea:
  1. Inicializa MPI y obtén el rango del proceso actual y el número total de procesos.
  2. Si el rango del proceso actual es 0, debe generar una lista de números enteros aleatorios entre 1 y 100 de longitud n y enviarla al proceso con rango 1.
  3. El proceso con rango 1 debe recibir la lista de números del proceso 0 y calcular la suma de todos los números.
  4. El proceso con rango 1 debe enviar esta suma de vuelta al proceso 0.
  5. El proceso 0 debe recibir la suma de vuelta y mostrarla por pantalla.
  6. Los procesos con rango diferente de 0 y 1 deben esperar sin hacer nada.

Nota: Puedes usar la biblioteca random de Python para generar números aleatorios.

Ejercicio 3

La función MPI.Wtime() es una función de temporización proporcionada por MPI (Message Passing Interface) para medir el tiempo en segundos. Esta función se utiliza comúnmente en programas MPI para registrar el tiempo de ejecución de secciones de código, operaciones de

comunicación, o cualquier otra tarea que se desee medir. El funcionamiento sería el siguiente:

```
start_time = MPI.Wtime()

# Sección de código a medir

end_time = MPI.Wtime()

elapsed_time = end_time - start_time

print("Tiempo transcurrido:", elapsed_time, "segundos")
```

- Modifica el ejercicio anterior para que todos los procesadores distintos de 0 reciban la lista de números aleatorios. Además, vamos a medir el tiempo que tardamos en realizar las operaciones de comunicación, para poder compararlo más adelante, para ello utilizaremos la función MPI.Wtime. El tiempo de comunicación lo medirá y mostrará por pantalla el procesador 0 (que es el que realiza todos los envíos). En concreto el ejercicio realizará las siguientes tareas:
  1. Inicializa MPI y obtén el rango del proceso actual y el número total de procesos.
  2. Si el rango del proceso actual es 0, el proceso debe generar una lista de números enteros aleatorios entre 1 y 100 de longitud n (por ejemplo 1000).
  3. El proceso 0 debe medir el tiempo de ejecución para enviar esta lista a todos los otros procesos utilizando comunicaciones punto a punto (send/recv) y mostrarlo por pantalla
  4. Los procesos con rango diferente de 0 deben recibir la lista generada por el proceso 0.
  5. Cada proceso debe calcular la suma de los números recibidos y mostrarla por pantalla.

Ejercicio 4

En los ejercicios anteriores hemos mandado un mensaje desde el proceso 0 a todos los demás, una operación de difusión (broadcast). Esta operación es muy común y existe una primitiva para realizarla de forma eficiente. En general, la difusión está implementada de manera bastante sofisticada (utilizando un árbol de mensajes) y aprovechando las características de la red de comunicaciones. La difusión es muy fácil con MPI, solamente hay que

especificar el origen con el parámetro `root`.

```
mensaje_recibido = comm.bcast(mensaje_enviado, root=0)
```

Observa que a diferencia de `send` y `recv`, aquí todos los procesos tienen que ejecutar la misma llamada a `bcast`.

- Modifica el programa anterior para utilizar `bcast()` en lugar de `send()` y `recv()`. Recuerda medir el tiempo de comunicación con la primitiva MPI.Wtime()
- Compara los tiempos del ejercicio anterior y este ¿cuál es más rápido? Nota: para poder comparar, la longitud de la lista a enviar debe ser la misma en los dos casos. Prueba con diferentes tamaños de n y diferente número de procesos.

Ejercicio 5

Otra operación colectiva muy útil es la reducción, una especie de inversa a la difusión. Mientras que en la difusión enviamos un dato idéntico a todos los procesos, en la reducción cada proceso envía un dato diferente y lo recibe un único proceso. Este dato recibido es una combinación de los datos enviados, como por ejemplo la suma.

La reducción se hace en MPI con la primitiva `reduce` similar a `bcast`. En este caso el parámetro root indica el destino en lugar del origen.

El siguiente código muestra un ejemplo de uso de la operación reduce, en el que todos los procesadores generan un número aleatorio y lo envían al procesador 0 haciendo la suma de todos ellos

```
from mpi4py import MPI
import random

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

numero = random.random()
print("Procesador", rank, ":", numero)

valor = comm.reduce(numero, root=0)

print("Valor ", valor)
```

La operación con la que se combinan los mensajes en la reducción se especifica con el parámetro `op`. La operación por defecto es la suma (por lo que si no se especifica el parámetro op se hará una suma) y además están disponibles las siguientes operaciones:

MPI.MAX	Devuelve el máximo
MPI.MIN	Devuelve el mínimo
MPI.SUM	Suma los elementos
MPI.PROD	Multiplica los elementos
MPI.LAND	Y lógico
MPI.LOR	O lógico
MPI.BAND	Y bit a bit
MPI.BOR	O bit a bit
MPI.MAXLOC	Devuelve el máximo y un número asociado (índice)
MPI.MINLOC	Devuelve el mínimo y un número asociado (índice)

Ejecuta el código anterior para familiarizarte con la primitiva reduce. Una vez tengas claro cómo funciona realiza el siguiente ejercicio (puedes usar los ejercicios anteriores como base)

- Escribe un programa en Python utilizando MPI que realice la siguiente tarea:
  1. Inicializa MPI y obtén el rango del proceso actual y el número total de procesos.
  2. Si el rango del proceso actual es 0, el proceso debe generar una lista de números enteros aleatorios entre 1 y 100 de longitud n.
  3. El proceso 0 debe transmitir esta lista a todos los otros procesos utilizando la función de broadcast de MPI.
  4. Todos los procesos deben calcular la suma de los números recibidos.
  5. El proceso 0 debe comparar el tiempo de ejecución para recibir las sumas parciales calculadas por los otros procesadores utilizando comunicaciones punto a punto (send/recv) y utilizando la función de reducción de MPI
  6. El proceso 0 debe mostrar las sumas calculadas tanto con comunicaciones punto a punto como con la función de reducción de MPI, junto con el tiempo de ejecución de ambas operaciones de comunicación.

Recuerda que para medir el tiempo debes utilizar la función MPI.Wtime

- Compara los tiempos de recibir las sumas con comunicación punto a punto y con comunicación colectiva ¿cuál es más rápido? Prueba con diferentes tamaños de n y diferente número de procesos.