Seminario práctico 2

Introducción a la Interfaz de Paso de Mensajes (MPI)

El objetivo de este seminario es familiarizar al alumno con el uso de la interfaz de paso de mensajes MPI y la programación de programas paralelos sencillos en C++ usando funciones de MPI. En primer lugar, se recordarán los pasos necesarios para compilar y ejecutar programas en este entorno (lo que ya se vio en asignaturas previas del Grado). La parte principal de este seminario consistirá en un tutorial web interactivo de MPI disponible a través de la página web:
`https://lsi.ugr.es/jmantas/ppr/tutoriales/tutorial_mpi.php`

2.1. La implementación de MPI usada en este seminario

`OpenMPI` es una implementación portable y de código abierto del estándar `MPI-3`, llevada a cabo por una serie de instituciones de ámbito tanto académico y científico como industrial. Para obtener más información sobre OpenMPI, uno puede consultar la página web oficial:
`http://www.open-mpi.org/`
Los detalles de instalación de OpenMPI se pueden consultar en la sección de ayuda de la página web de la asignatura:
`http://lsi.ugr.es/ppr/ayuda/instalacion.php?ins=openmpi`

2.2. Compilación y ejecución de programas MPI.

OpenMPI ofrece varios scripts necesarios para trabajar con programas aumentados con llamadas a funciones de MPI. Los más importantes son:

- `mpicc`: para compilar y enlazar programas en C que hagan uso de MPI.
- `mpicxx`: para compilar y enlazar programas C++ que hagan uso de MPI.
- `mpirun`: para ejecutar programas MPI.

`mpicc` y `mpicxx` pueden utilizarse con las mismas opciones que el compilador de C/C++ usual, p.e.:

```
$ mpicxx -c ejemplo.c
$ mpicxx -o ejemplo ejemplo.o
```

La forma más usual de ejecutar un programa MPI es :

```
$ mpirun -np 4 ejemplo
```

El argumento `-np` sirve para indicar cuántos procesos ejecutarán el programa `ejemplo`. En este caso, OpenMPI lanzará cuatro procesos `ejemplo`. Como no se indica nada más, OpenMPI lanzará dichos procesos en la máquina local.

Para tener un mayor control sobre qué proceso se lanza en cada máquina cuando ejecutamos `mpirun`, utilizaremos un *archivo local de máquinas* en el que indicaremos, las máquinas en las que se lanzarán los procesos de nuestro programa paralelo.

La ejecución de un programa OpenMPI asumiendo un archivo local de máquinas sigue el formato:

```
mpirun [ -nolocal] -machinefile <maquinas> -np  <numero_procesos> <programa>
```

Por ejemplo, si el archivo de máquinas `machines4` contiene las siguientes líneas:

```
ei190190
ei190191
ei190192
ei190193
```

y ejecutamos el programa `ejemplo` sobre 4 procesos con la sentencia:

```
$ mpirun -machinefile machines4  -np 4  ejemplo
```

Esto significa que la primera réplica del programa `ejemplo` (el proceso 0) se ejecutará sobre la máquina *ei190190*, la segunda (el proceso 1) se lanzará en la máquina *ei190191*, la tercera en *ei190192* y la cuarta en *ei190193*.

Si se incluye el modificador `-nolocal`, no se asignará ninguna réplica a la la máquina local (desde la que se llama a `mpirun`). Como consecuencia, si se utilizara `-nolocal`, la máquina *ei190190* de nuestro archivo de máquinas (suponemos que esta es la máquina local) no participaría en el cómputo y *ei190191* invocaría 2 procesos en lugar de uno.

Existen más opciones disponibles para `mpirun` que pueden ser consultadas en la siguiente página:
`http://www.open-mpi.org/doc/v1.4/man1/mpirun.1.php`

2.3. Tutorial de MPI.

En la dirección:
`http://lsi.ugr.es/~jmantas/ppr/tutoriales/tutorial_mpi.php`
podemos acceder al tutorial de introducción a MPI. El tutorial incluye 8 ejercicios agrupados en tres niveles de dificultad (básico, medio y avanzado).

Cada ejercicio está estructurado en tres apartados:

Objetivos Se especifica qué conocimientos sobre programación con `MPI` se adquieren como resultado de la realización del ejercicio.

Enunciado Es una descripción del ejercicio que servirá como especificación del problema. También incluye una serie de pistas, así como código secuencial de partida y enlaces a la ayuda para las funciones MPI que sean necesarias. Deberemos trabajar en una solución propia antes de descargar la solución aportada por el tutorial.

Solución Incluye el código de una posible solución al problema planteado y la salida de la ejecución.

2.4. Ejercicios Propuestos

En esta primera práctica, seguiremos los ejercicios propuestos en el tutorial y realizaremos las siguientes cuatro tareas, que toman como base algunas de las soluciones propuestas en el tutorial.

1. Modificar el programa solución del ejercicio *3.2 Send Receive* del tutorial para que el proceso 0 difunda su identificador de proceso (0) al resto de procesos con identificadores pares, siguiendo un anillo de procesos pares, y el proceso 1 haga lo mismo con los procesos impares. Se deben tener en cuenta soluciones con cualquier número de procesos.

2. Modificar el programa solución del cálculo paralelo del número π (*3.3 Cálculo de PI*) para que los subintervalos de trabajo sean distribuidos por bloques en lugar de cíclicamente entre los procesos. Por ejemplo, si tuviéramos 3 procesos y $n = 11$ (número de subintervalos), el proceso 0 debería aproximar las integrales numéricas en los primeros 4 subintervalos consecutivos (1,2,3,4), el proceso 1 calcularía las integrales en los siguientes 4 subintervalos (5,6,7,8,) y el proceso 2 calcularía los últimos tres (9,10,11). Se recomienda empezar derivando matemáticamente un método general para repartir por bloques n subintervalos entre P procesos para cualquier n entero positivo. Modificarlo también la solución para que la aproximación a π se obtenga en todos los procesos.

3. Modificar el programa solución del cálculo del producto escalar de dos vectores (*4.1 Producto Escalar*) para que cada proceso inicialice por su cuenta su parte correspondiente del vector B de forma local, de tal forma que no haya necesidad de inicializar todo el vector B en el proceso 0 y repartir sus bloques entre los procesos.

4. Modificar el programa solución del ejercicio *4.4 Comunicadores* para que también se realice un `Scatter` de un vector de enteros desde el proceso 1 del comunicador global a todos los procesos impares de dicho comunicador. Los valores de este vector se escogen arbitrariamente en el proceso 0 (ojo, en el proceso con rango 0 del comunicador de rangos impares que es el proceso 1 de MPI Comm_World), pero su tamaño debe ser igual número de procesos impares en el comunicador global. El reparto asignará un elemento de dicho vector a cada proceso impar del comunicador global. Se recomienda usar el comunicador de impares para realizar esta tarea.

Como resultado de este seminario se debe realizar lo siguiente:

- Redactar una breve memoria (2 páginas como mucho) que explique las decisiones tomadas para resolver cada tarea planteada.
- Derivar los archivos fuente de las soluciones para cada tarea planteada como $p_i.cc$, $i = 1, ..., 4$.
- Obtener un archivo `Makefile` para automatizar la compilación y enlazado de los diferentes archivos, obteniendo los ejecutables como p_i, $i = 1, ..., 4$.

- Los archivos fuente, el archivo `Makefile` y la memoria (preferentemente en pdf) deben incluirse en una carpeta y deben empaquetarse (por ejemplo, utilizando `tar cz`) para subirla el archivo generado como resultado de un trabajo de la plataforma *PRADO* (el profesor creará una tarea asociada a este seminario).