Laboratorio 4
Paralelismo en sistemas de memoria distribuida

Objetivos:

- Aprender a paralelizar una aplicación en un sistema de memoria distribuida utilizando el estilo de programación paralela mediante "paso de mensajes".
- Estudiar el API de MPI y aplicar distintas estrategias de paralelismo en su aplicación.
- Aplicar métodos y técnicas propios de esta asignatura para estimar las ganancias máximas y
- las eficiencias del proceso de paralelización.

Introducción

En esta práctica se va a continuar trabajando sobre el problema tratado en la práctica anterior. Deberéis, por tanto, paralelizar la aplicación secuencial dada para mejorar el desempeño.

Podréis encontrar más información acerca de las interfaces de paso de mensajes, y el modelo de programación (tal y como se ha visto en teoría), además de las funciones de comunicación colectiva, en los siguientes vídeos preparados por la Universitat Politècnica de València (UPV):

- https://youtu.be/uiZU05CfZUg?si=oS5gU-AqjUGzRKl3
- https://youtu.be/y1yB7LTn6oA?si=dO7IdLa8QmGMS9CI

Especificación MPI

MPI (Message-passing interface) es una especificación que establece las funciones de una biblioteca para el paso de mensajes entre múltiples procesadores. Esto permite la comunicación vía paso de mensajes entre nodos de un clúster de computadores que constituyen sistemas multicomputador de memoria distribuida (véase Unidad 1, Taxonomía de Flynn). En este escenario los datos utilizados por un proceso se mueven desde el espacio de direcciones de este al de otro proceso, de forma cooperativa, mediante las instrucciones recogidas en el estándar MPI.

Podemos distinguir 4 tipos de instrucciones definidas en la especificación de MPI:

1. Instrucciones para abrir, gestionar y cerrar las comunicaciones entre procesos ejecutándose en diferentes nodos.
2. Instrucciones para transferir datos entre 2 procesos (uno a uno).
3. Instrucciones para transferir datos entre múltiples procesos.
4. Instrucciones que permiten al usuario definir tipos de datos.

Las principales ventajas del uso de MPI son que permite establecer un estándar de paso de mensajes en arquitecturas multicomputadora portable, fácil de utilizar y que permite abstraer los detalles de bajo nivel propios de la gestión de este tipo de comunicaciones.

Existen multitud de librerías que implementan el estándar MPI (mpich, OpenMPI, …) en diferentes lenguajes como c, c++, Fortran, Python etc. En este enlace podéis encontrar un tutorial con ejemplos de uso de MPI de la Universidad de Granada:

- https://lsi2.ugr.es/jmantas/ppr/tutoriales/tutorial_mpi.php

Tarea 4.1: Trabajo previo: familiarizarse con MPI

Como trabajo previo para el uso de MPI en el código secuencial de la práctica anterior, es necesario familiarizarse con el modo de funcionar de las librerías de MPI, así como del proceso para lanzar múltiples procesos en máquinas distintas del laboratorio.

NOTA: Es muy importante seguir las siguientes instrucciones (Tarea 4.2) al pie de la letra, y utilizar los ordenadores del laboratorio para realizar y probar

En grupo, deberéis realizar un pequeño programa que lance procesos en varias máquinas, y probar que esto se realice de forma correcta, para, posteriormente, pasar a hacerlo con el código modificado de la práctica anterior.

El ejemplo más básico, sería este "Hello World" que se ejecuta en múltiples procesos:

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

int sched_getcpu();

int main(int argc, char *argv[]) {
  int numprocs, rank, 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, &rank);
  MPI_Get_processor_name(processor_name, &namelen);
  printf(" PID: %d Hello from process %d out of %d on %s processor %d \n",
           getppid(), rank, numprocs, processor_name, sched_getcpu());
  if (rank==2)
  {
    printf("¡Hola! desde el procesador %d\n", rank);
  }
  MPI_Finalize();
}
```

Un ejemplo algo más avanzado, lo podéis encontrar en el tutorial de la Universidad de Granada, como es el caso del cálculo de pi (tal como se ha visto en clase de teoría):

- https://lsi2.ugr.es/jmantas/ppr/tutoriales/tutorial_mpi.php?tuto=03_pi

Entregable para esta tarea: pensad un problema sencillo paralelizable con MPI, que contenga llamadas de comunicación colectiva, y realizad su implementación mediante la librería, tal como habéis visto en los ejemplos.

Tarea 4.2: Pasos para compilar y ejecutar en el laboratorio

1. Todas las máquinas deben tener el **mismo usuario**. Todos los ejecutables deben tener el **mismo nombre** en cada máquina y estar en la misma carpeta.
2. Compilar empleando mpicc o mpic++:
   ```
   mpicc programa.c -o programa
   ```
   Recuerde que el programa, junto con sus dependencias, deben copiarse y ser accesibles en cada nodo de su multicomputador.

3. Crear **el archivo de hosts** con las IPs o nombres de cada nodo de la red y copiar al resto de máquinas en el mismo directorio de trabajo donde se encuentran los ejecutables. El archivo de hosts es un fichero de texto plano que especifica el nombre o la IP de las máquinas que forman el supercomputador. **Cada nombre o IP en una línea**.
4. Cree un certificado de clave pública ssh en uno de los nodos:
   `ssh-keygen -t rsa` *(esto crea la carpeta oculta* `~/.ssh`*)*
5. Copiar el archivo `~/.ssh/id_rsa.pub` en la carpeta `~/.ssh` del resto de máquinas y cambiar su nombre a `authorized_keys`. Si no existe la carpeta oculta `~/.ssh` en una máquina, crearla previamente ejecutando el comando `ssh-keygen -t rsa`.
6. Ejecutar el programa en la máquina donde se creó el certificado:
   ```
   mpirun -mca plm_rsh_no_tree_spawn 1
   -hostfile <archivo_hosts> -n <núm_procesos> ./programa
   ```

Observaciones:

- Usar el mismo directorio en todas las máquinas: p.ej. Carpeta personal (`$HOME`)
- Abrir sesión con **el mismo usuario en todos los nodos** del cluster
- Puede ser necesario modificar los permisos de los archivos:
  - Archivo de hosts: `chmod 600 <archivo_hosts>`
  - Archivo ejecutable para permitir ejecución remota: `chmod o+x programa`

Entregable de esta tarea: Deberéis mostrar al profesor de prácticas el funcionamiento de vuestro problema de la Tarea 4.1 funcionando en paralelo en los ordenadores del laboratorio. **Fecha límite: mostrar el código funcionando, al inicio de la tercera sesión para esta práctica.**

Tarea 4.3: Paralelización del código de la práctica anterior

Tal y como se ha visto en las tareas anteriores, y partiendo de la versión secuencial (sin OpenMP o hebras con `async`), modificar el código de la práctica anterior para paralelizarlo mediante MPI.

Recordad los pasos vistos en clase de teoría:

- Descomposición en tareas independientes:
  - Detectar el paralelismo implícito del código (esto se ha hecho en la práctica anterior).
- Asignar las tareas a procesos:
  - Es decir, decidir qué tareas se van a poder ejecutar en paralelo
- Redacción del código paralelo:
  - Es decir, utilizando en este caso las llamadas necesarias de la librería MPI
- Evaluación de las prestaciones:
  - Cálculos de ganancia con respecto a la versión secuencial, y a la versión con OpenMP y/o hebras.

Entregable de esta tarea: Se entregará el código paralelizado con MPI, y una memoria que lo acompañe donde se explicará:

- Qué partes del código han sido paralelizadas y por qué. Justificad muy bien la respuesta, en base a experimentos en los que se pruebe a paralelizar diversas partes del código, documentando los resultados en relación al tiempo de ejecución.
- Comparativa de tiempos, y cálculo de las ganancias de la versión MPI respecto a la versión secuencial y a la de la práctica anterior (OpenMP y/o hebras con `async`).
- **Fecha de entrega: antes de la última sesión de prácticas.**