universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO MPI:
FUNCIONES BLOQUEANTES

Autor: Adrián Carral Martínez

1. Introducción:

En esta práctica se ha implementado un programa paralelo en lenguaje C utilizando la biblioteca MPI (Message Passing Interface). El objetivo es aplicar funciones **bloqueantes** para el envío y recepción de datos entre procesos, distribuir el trabajo de multiplicación de matrices, y analizar cómo varía el rendimiento al escalar el número de procesos utilizados (2, 4, 8, 16 y 32).

---

2. Desarrollo paso a paso:

Paso 1: Inicialización del entorno MPI

Se usaron las funciones `MPI_Init`, `MPI_Comm_size` y `MPI_Comm_rank` para establecer el entorno paralelo y obtener el número total de procesos (`size`) y el identificador único de cada uno (`rank`).

Paso 2: Preparación de matrices

Se definió que el tamaño `N` de las matrices cuadradas A y B fuera igual al número de procesos. El proceso maestro (rank 0) generó las matrices de tamaño `N x N` con valores aleatorios entre 0 y 100.

Paso 3: Distribución de datos

El maestro distribuyó una fila de la matriz A y una fila de la matriz B a cada proceso (incluido él mismo) usando `MPI_Send`. Los esclavos la recibieron con `MPI_Recv`.

Paso 4: Procesamiento local

Cada proceso multiplicó elemento a elemento su fila de A y su fila de B, y almacenó los resultados en una fila local `row_C`.

Paso 5: Recolección de resultados

Cada proceso envió su fila resultante al maestro mediante `MPI_Send`. El maestro las reunió para reconstruir la matriz C final.

Paso 6: Medición de tiempo

Se midió el tiempo total desde justo antes de la distribución de datos hasta justo después de la recolección completa, usando `MPI_Wtime()`.

---

3. Ejecución en Caléndula:

3.1 Reserva de recursos

```
[ule_formacion_10_10@frontend1 ~]$ salloc -p formacion -t 100 -C cascadelake -n
16
salloc: Pending job allocation 2319447
salloc: job 2319447 queued and waiting for resources
salloc: job 2319447 has been allocated resources
salloc: Granted job allocation 2319447
salloc: Waiting for resource configuration
salloc: Nodes cn5001 are ready for job
```

```
salloc -p formacion -t 100 -C cascadelake -n 16
```

3.2 Carga de módulo MPI

```
[ule_formacion_10_10@cn5001 ~]$ module load /soft/calendula2/modulefiles/cascadelake/openmpi_4.0.3_gcc9.2.0
Loading /soft/calendula2/modulefiles/cascadelake/openmpi_4.0.3_gcc9.2.0
  Loading requirement: cascadelake/gcc_9.2.0
```

```
module load /soft/calendula2/modulefiles/cascadelake/openmpi_4.0.3_gcc9.2.0
```

3.3 Compilación

```
[ule_formacion_10_10@cn5001 ~]$ mpicc funciones_bloqueantes.c -o funciones_bloqueantes
```

```
mpicc funciones_bloqueantes.c -o funciones_bloqueantes
```

---

4. Análisis de los resultados:

```
[ule_formacion_10_10@cn5001 ~]$ mpirun -n 2 ./funciones_bloqueantes
Tiempo de ejecución: 0.003839 segundos
[ule_formacion_10_10@cn5001 ~]$ mpirun -n 4 ./funciones_bloqueantes
Tiempo de ejecución: 0.000062 segundos
[ule_formacion_10_10@cn5001 ~]$ mpirun -n 8 ./funciones_bloqueantes
Tiempo de ejecución: 0.000135 segundos
[ule_formacion_10_10@cn5001 ~]$ mpirun -n 16 ./funciones_bloqueantes
Tiempo de ejecución: 0.000216 segundos
```

Nº de procesos	Tiempo 1	Tiempo 2	Tiempo 3	Tiempo 4	Media
2	0.000047 s	0.000052 s	0.000049 s	0.000052 s	**0.000808 s**
4	0.000071 s	0.000144 s	0.000068 s	0.000067 s	**0.000088 s**
8	0.000118 s	0.000128 s	0.000120 s	0.000113 s	**0.000120 s**
16	0.000223 s	0.000226 s	0.000216 s	0.000229 s	**0.000223 s**

- Se observa una **clara mejora** de rendimiento al pasar de 2 a 4 procesos, donde el tiempo medio se reduce drásticamente.

- A partir de 4 procesos, el beneficio se reduce: entre 4 y 8 el cambio es pequeño, y de 8 a 16 incluso empeora. Esto sugiere que la **sobrecarga de comunicación supera el ahorro por paralelismo** para tareas tan pequeñas por proceso.

5. Conclusión

Esta práctica ha sido una introducción eficaz a la programación paralela con MPI y al uso de funciones bloqueantes. He podido:

- Aprender el reparto de trabajo usando `MPI_Send` y `MPI_Recv`.
- Distribuir datos y recoger resultados de forma manual.
- Medir rendimiento y analizar el escalado de una aplicación sencilla.
- Identificar el punto en el que **añadir más procesos no compensa** debido al coste de comunicación.