Arquitectura de Sistemas Paralelos

Práctica 3

Programación en OpenMP
Comunicación de Datos

Michael Alexander Fajardo Malacatus
Gloria del Valle Cano
michael.fajardo@estudiante.uam.es & gloria.valle@estudiante.uam.es

Grado en Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
24-04-2020

Índice general

Índice de figuras

Introducción

En esta práctica investigamos una manera de paralelizar totalmente diferente. OpenMP es una forma de programar en dispositivos de memoria compartida, mientras que en el antiguo MPI se trabajaba con memoria distribuida, donde cada proceso paralelo está trabajando en su propio espacio de memoria aislado de los demás. En OpenMP el paralelismo ocurre donde cada hilo paralelo tiene acceso a todos sus datos. Veremos el rendimiento y la comparativa de estas tecnologías, así como las ventajas en juntar estas dos tecnologías en un programa híbrido.

Bloque 1: Ejercicios básicos en OpenMP

1.1. Cálculo de Pi por integración numérica

En este apartado se incluyen las siguientes versiones de la estimación de pi mediante este método.

- Versión serie.
- Versión `MPI`.
- Versión `parallel`.
- Versión `private + critical`
- Versión `WorkSharing`.
- Versión `WorkSharing + reduction`.
- Versión híbrida `MPI + OpenMP`.

Para todas estas versiones se ha tenido en cuenta un n de 10485760. Hemos realizado una comparativa final de estas versiones:

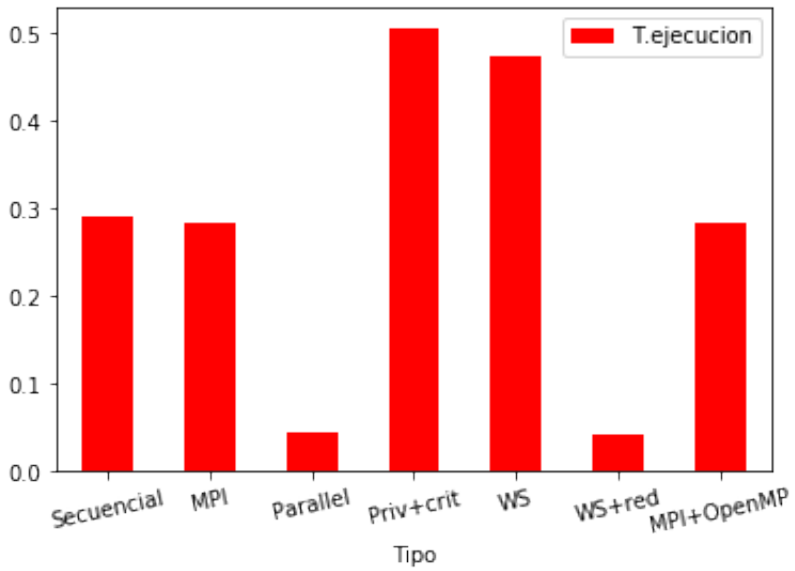

Figura 1: Speedup (Estimación de pi, método por integración)

Podemos ver que la necesidad depende del tamaño del problema. Para este caso no es necesario hacer un paralelismo híbrido, ya que sólo con `OpenMP` conseguimos buenos resultados.

1.2. Suma de N números enteros

Para la realización de las dos variantes del ejercicio en OpenMP, en ambas se ha utilizado:

1. La directiva `pragma omp parallel for`, para el `reparto equitativo` entre los hilos.
2. Para la variante con `reparto dinámico`, la cláusula `schedule` de tipo `dynamic`, con el reparto pedido por el enunciado.
3. La cláusula `reduce` con la operación de suma.
4. Las cláusulas `private`, para proteger el acceso a la variable, en nuestro caso la variable de recorrido del bucle y la cláusula `shared`, para permitir que todos los hilos puedan acceder a la información de una determinada variable, en nuestro caso el array con el que trabajar.

1.2.1. Comparación Serie, OpenMP, MPI, OpenMP + MPI

Para la comparación se ha especificado un tamaño del array de N = 1000000000.

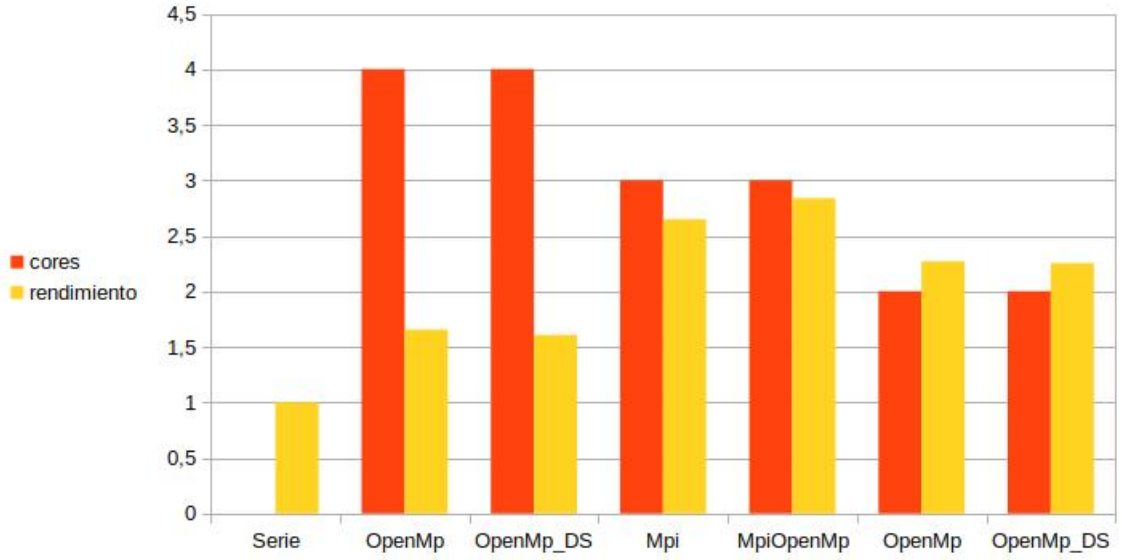

Figura 2: Speedup (Suma)

Se puede observar que MPI tiene un mejor rendimiento respecto a las demás tecnologías. Se puede ver que el rendimiento en `OpenMP` con y sin reparto `dinámico(OpenMP_DS)` es muy similar. Aunque se puede ver claramente que al unir `OpenMP` y `MPI` el rendimiento aumenta aún más que con MPI solo.

1.3. Estimar el valor de pi a partir de números aleatorios.

Para la realización de este ejercicio, se ha utilizado:

- **pragma omp for**: para el reparto equitativo.
- **private(p)**: para proteger el acceso a los puntos x e y.
- **reduction(+:k)**: para hacer una copia por hilo de esta operación.
- **schedule(static)**: para designar la operación como estática.

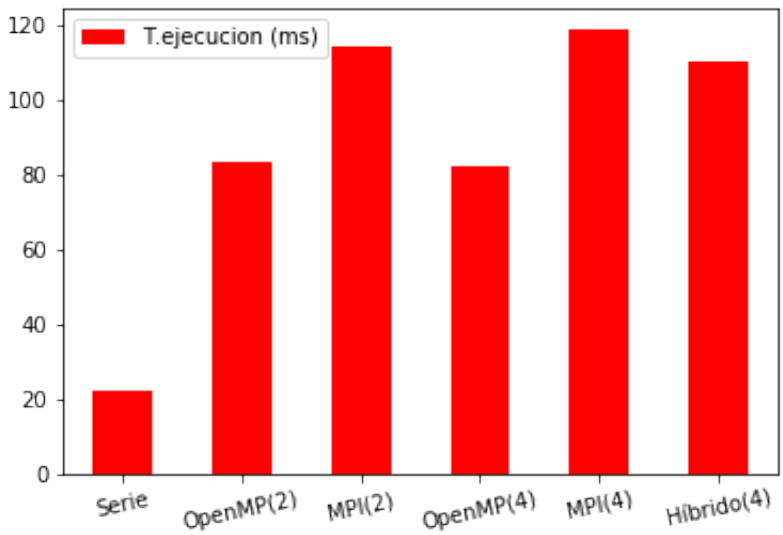

Figura 3: Comparativa en tiempo de ejecución

Podemos ver en este caso que `OpenMP` funciona mejor que `MPI`, y que la tecnología funciona mejor de manera independiente. Dado el problema, el paralelismo ocurre en el *for*, donde cada hilo paralelo tiene acceso a todos sus datos, siendo `OpenMP` más idóneo para este caso.

1.4 Multiplicación de matrices NxN

Para la realización de la multiplicacion de matrices se ha realizado la paralelización por filas. Para ello se han utilizado:

1. La directiva **pragma omp parallel for**, para el reparto equitativo entre los hilos de la matriz.
2. Las cláusula **private**, para proteger el acceso a la variable, en nuestro caso las variable de recorrido de los 3 bucles y la cláusula **shared**, para permitir que todos los hilos puedan acceder a la información de una determinada variable, en nuestro caso los arrays de las matrices con las que trabajar.

1.4.1 Comparación Serie, OpenMP, MPI

Para la comparación se ha especificado un tamaño de 400 para la matriz.

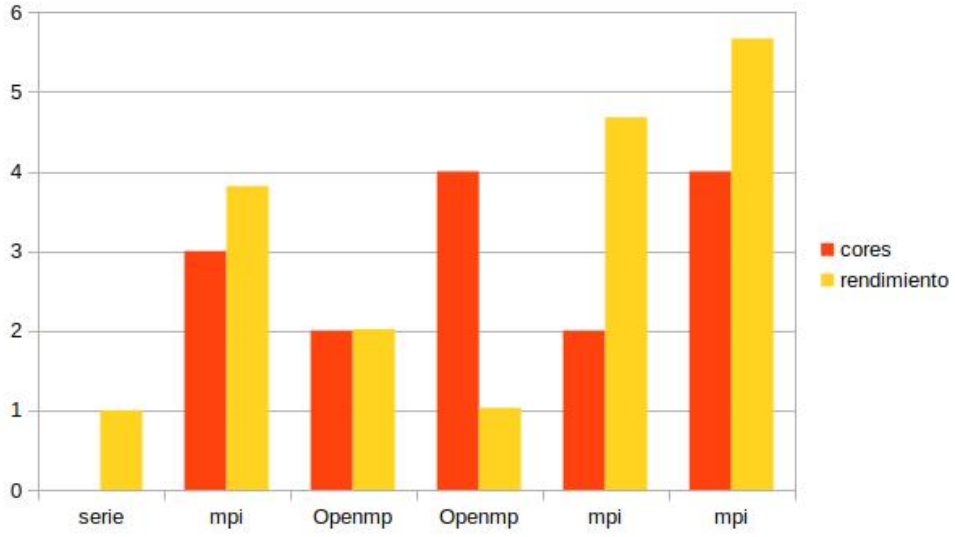

Figura 4: Speedup (Multiplicacion Matrices)

Como se puede apreciar en la figura de arriba la tecnología que mayor rendimiento tiene es `MPI` que se incrementa cuanto mayor sea el número de procesadores. Cabe mencionar que con 3 procesadores `MPI` baja ligeramente el rendimiento, lo que puede deberse al reparto, puesto que solo se contaría realmente con 2 procesadores para realizar el trabajo (esclavos) y el maestro, sin embargo cuando el número de procesadores es par el maestro también tiene trabajo que hacer y por tanto se tarda menos. En cuanto a `OpenMP` cuanto mayor es el número de threads menor es el rendimiento, se piensa que es por que tarda más en gestionar los hilos cuanto mayor es el número de hilos y eso hace que su rendimiento empeore.

Bloque 3: Programación Híbrida MPI + OpenMP

3.1. Suma de N enteros

Para la realización del programa con `MPI` junto con `OpenMP`, se ha hecho uso de las funciones **scatter y reduce** por parte de **MPI** y las directivas utilizadas en el apartado **1.2** por parte de `OpenMP`.

3.2. Estimación de Pi a partir de números aleatorios

Se incluye la realización del programa híbrido partiendo del apartado **1.4**, utilizando **firstprivate** para el punto y la distancia, con la operación **reduce** para la acumulación de la variable k.