Metodología de la Programación Paralela
Curso 2024/25

Práctica 1:

Paralelización con OpenMP

En esta práctica se va a llevar a cabo la paralelización del algoritmo evolutivo proporcionado en la Práctica 0 mediante el uso de directivas ofrecidas por OpenMP, estándar *de facto* para la programación paralela en sistemas con memoria compartida. Para ello, se proponen una serie de cuestiones, que se han de realizar de forma progresiva analizando en cada momento la mejora que se obtiene en el rendimiento. Esta práctica representa el 10% de la calificación de la asignatura.

La fecha de entrega de la práctica será el **18 de octubre a las 23:55**. Entregas posteriores no se tendrán en cuenta. La entrega se realizará a través de la tarea habilitada en el Aula Virtual, adjuntando un fichero .zip o .tgz que contenga:

- El código fuente paralelizado y los ficheros *Makefile* y *run.sh* para su compilación y ejecución. Además, el código ha de incluir los comentarios necesarios para ayudar a comprender los cambios realizados.

- Un documento en formato *.pdf* que describa cómo se han resuelto las cuestiones propuestas, justificando las decisiones de paralelización tomadas en cada caso y analizando los resultados obtenidos. El código que se muestre debe coincidir con el que se entregue. Asimismo, el documento ha de incluir una sección en la que se indique cómo ha sido la coordinación, el reparto del trabajo entre los miembros del grupo y el tiempo dedicado.

Es recomendable, aunque no obligatorio, el uso de la herramienta *GitLab/GitHub* como repositorio software para ir almacenando los cambios realizados durante el desarrollo de la práctica. Si en alguna cuestión es necesario comparar varias versiones paralelas, se crearán diferentes ramas (una para cada versión). La rama máster contendrá la versión final del código con la que se obtienen las mejores prestaciones.

En aquellos casos en los que sea necesario mostrar la evolución de las prestaciones, se recomienda estructurar la información en tablas que incluyan el tiempo de ejecución y otras métricas de rendimiento. El formato de cada tabla podría ser el siguiente:

n	*m*	*n_gen*	*tam_pob*	*n_threads*	*t_ejec*	*speed-up*	*eficiencia*

donde $speed-up = \frac{t_{secuen.}}{t_{paral.}}$ y $eficiencia = \frac{speed-up}{p}$, con p = número de elementos de proceso (hilos en memoria compartida).

Se recomienda también la creación de gráficos a partir de las series de valores más relevantes de las tablas, en los que se muestre la evolución de las métricas al variar el número de hilos.

Describir el sistema computacional utilizado con la ayuda de la herramienta: `lstopo -f -p --no-legend --no-io sistema.svg`.

A continuación, se muestran las cuestiones a realizar indicando de forma orientativa su distribución en cada semana.

1ª Semana

Cuestión 1. Justificar qué funciones y/o zonas del código son paralelizables. No es necesario indicar las directivas OpenMP que sería necesario utilizar.

Cuestión 2. Con el fin de obtener un código más eficiente en la parte de memoria compartida y poder optimizar el proceso de mejora de la población de individuos: reemplazar las llamadas a la función *rand* por *rand_r* (que es "*thread safe*") para evitar, por un lado, que se genere la misma secuencia de valores aleatorios en diferentes hilos y, por otro, que se pierda paralelismo durante la generación de dichos valores debido a la secuencialidad que introduce la función *rand*. Además, añadir el código necesario para implementar correctamente la generación aleatoria a lo largo del algoritmo.

Para ello, una posibilidad sería crear en *mh.c* una variable global *seed* que se hará *threadprivate* y que se pasará como argumento por referencia a *rand_r*. Así, además, cada hilo podrá inicializar *seed* en paralelo de forma independiente (por ejemplo, en función del tiempo del sistema, de su *pid* o una combinación de ambos) en una primera región paralela situada justo antes de generar la población inicial de subconjuntos *B* del problema MDP (bucle *for* que crea cada individuo, que identificaremos como FOR_INI). Es en esta generación aleatoria de individuos donde el efecto sobre el paralelismo de *rand_r* en lugar de *rand* es claramente apreciable.

Lanzar ejecuciones y tomar tiempos del bucle FOR_INI, representando en una misma gráfica dos series de tiempos variando el número de hilos: serie (a) usando *rand*, y serie (b) usando *rand_r* y las modificaciones del código previamente indicadas. Analizar los resultados obtenidos.

Cuestión 3. Paralelizar la función `fitness` utilizando, por un lado, los constructores *critical* y *atomic* (de forma independiente) y, por otro, la cláusula *reduction*, indicando en cada caso qué variables serían privadas y cuáles compartidas. Ejecutar el código variando el número de hilos y comparar los resultados obtenidos respecto a versión secuencial utilizando las métricas de rendimiento.

Justificar de forma teórica las diferencias y optimizar, en lo posible, el código de critical y atomic teniendo en cuenta cómo está implementada internamente *reduction*.

Cuestión 4. Probar diferentes formas de reparto de trabajo, *scheduling* (static, dynamic, guided), variando el tamaño de asignación (*chunk_size*) en aquellos bucles en los que se considere necesario. Ejecutar el código variando el número de hilos y justificar los tiempos de ejecución y rendimiento obtenidos con cada política.

2ª Semana

Cuestión 5. Paralelizar la función `cruzar` utilizando secciones. ¿Se podría usar la cláusula *nowait*? (argumentar la respuesta). Ejecutar el código variando el número de hilos y justificar el rendimiento obtenido.

Cuestión 6. Reemplazar las llamadas a la función `qsort` por la función `mergeSort` que se muestra. A continuación, paralelizar dicha rutina mediante el uso de tareas (OpenMP *tasks*) y ejecutar el código variando el número de hilos. ¿Se reduce el tiempo de ejecución? Si no es así, explica las posibles causas. La función `mezclar` a la que invoca se proporciona en el fichero `mezclar.c`.

```
void mergeSort(Individuo **poblacion, int izq, int der)
{
	int med = (izq + der)/2;
	if ((der - izq) < 2) { return; }

	mergeSort(poblacion, izq, med);
	mergeSort(poblacion, med, der);

	mezclar(poblacion, izq, med, der);
}
```

Cuestión 7. Explicar para qué se usan en general, y si es necesario utilizar de forma explícita en alguna zona del código los siguientes constructores: *barrier*, *single*, *master*, *ordered*. Considerar también su posible uso en un código paralelo optimizado (como el requerido en la Cuestión 9).

Cuestión 8. Realizar el informe final incluyendo las conclusiones generales y valoración personal sobre el trabajo realizado.

Cuestión 9. OPCIONAL. Paralelizar la totalidad del algoritmo utilizando las directivas OpenMP que se considere mejor para cada función o bucle (y que pueden ser distintas a las usadas para resolver las cuestiones). Justificar las decisiones tomadas. De este modo, se pretende tener un código paralelo óptimo en memoria compartida que se podrá usar posteriormente para comparar con el código paralelo de paso de mensajes e híbrido implementado en las siguientes prácticas.

Observaciones

- En el algoritmo, es posible que existan funciones o bucles con poca carga computacional donde el efecto del paralelismo no sea apreciable. En estos casos, se puede aumentar el coste para apreciar mejor la reducción de tiempos de ejecución en el caso de que exista. Por ejemplo, en la Cuestión 3, se puede tomar tiempos del cálculo del *fitness* de un solo individuo, aumentar el tamaño del mismo (un valor de *m* mayor) o, incluso, tomar tiempos del bucle del cálculo del fitness de toda la población.

- Hay que tener siempre presente que, para comparar tiempos entre experimentos, se deben realizar las ejecuciones en la misma máquina y con la misma instancia de problema.

- En la Cuestión 2, para que se transmita correctamente el valor de *seed* privado de cada hilo al resto de regiones paralelas donde se requiera, es necesario generar la semilla en paralelo con el número de hilos máximo que se vaya a usar en el algoritmo (hacer una evaluación previa). Entendiéndose que el resto de regiones paralelas serán creadas con un número de hilos menor o igual que esta primera región.

- Para cada cuestión se pueden tomar tiempos como se considere mejor para reflejar el efecto del paralelismo. Justificar en cada caso las decisiones tomadas. Por ejemplo, en la Cuestión 4, se pueden tomar tiempos de cada bucle por separado o de todos a la vez (con un número de hilos, política de reparto de trabajo y tamaño de bloque asignado independiente para cada uno de ellos).