Práctica 1: Memoria compartida

Sección 1: Análisis de rendimiento

En la primera sección del desarrollo de la práctica se deberá realizar un análisis de rendimiento sobre un programa. El objetivo es determinar la parte más adecuada para realizar una optimización de código. Para ello, se aplicarán las fórmulas vistas en clase, se determinará el speedup teórico máximo que se podría conseguir y se realizará una gráfica de escalabilidad junto con un análisis de la aplicación.

Primera parte: Codificación de la versión secuencial

Se propone realizar una implementación secuencial de un algoritmo de multiplicación de matrices de gran dimensión. La aplicación deberá cargar matrices de al menos 1000 filas por 1000 columnas desde fichero, y deberán validarse los resultados usando una matriz diagonal. Las matrices estarán formadas por números en formato coma flotante de 32 bits (floats). La implementación constará de las siguientes partes:

Generador de matrices de gran dimensión:

El alumno deberá codificar un programa que genere matrices de tamaño arbitrario. El número de filas y columnas será proporcionado a través de los parámetros del programa al inicio de la ejecución. Los datos se guardarán en el formato que el alumno elija (binario o texto plano. Los datos generados pueden ser aleatorios, pero también deberá poder generar una matriz identidad (matriz con ceros en todas las casillas menos en la diagonal). Se recomienda el siguiente formato de fichero:

<número filas> <número columnas> <datos>

Multiplicador de matrices

El alumno deberá realizar un segundo programa que implementará la multiplicación de matrices de tamaño arbitrario. Se deben implementar las siguientes funciones para facilitar el análisis en las siguientes secciones:

- **Lectura de las matrices desde ficheros:** El programa recibirá por parámetros el nombre de las dos matrices que se cargarán. Invocará una función que cargará las dos matrices que seguirá el siguiente esquema:

    - Leer número de filas y columnas de la primera matriz.
    - Reservar memoria dinámica para la primera matriz. Ésta se reservará como un array de punteros a arrays de floats (float **), que se inicializa de la siguiente manera:
        - Reservar un array de punteros a punteros de floats, de tamaño "número de filas".
        - Para cada posición del array de punteros a floats, reservar un array de tamaño "número de columnas.
    - Una vez reservados los datos, cargar los datos de la primera matriz, cargando en los índices [fila][columna] del array.
    - Repetir la reserva de memoria para la segunda matriz.
    - Cargar los datos de la segunda matriz "traspuestos", es decir, cargarlos en los índices [columna][fila] de la segunda matriz. Esto facilitará la multiplicación más adelante.

- **Multiplicación de matrices:** Una vez cargadas las matrices, se pasa a multiplicar. Se recomienda implementar la multiplicación de la siguiente manera:

    - Reservar espacio para la matriz resultado. En este caso puede ser un array unidimensional de tamaño "filas_matrizrx"columnas_matriz2".
    - Implementar una función que multiplique dos vectores, llamada "multiplica_vectores". Recibirá por parámetros dos arrays de floats, junto con el tamaño de cada uno de ellos. Devolverá el resultado de la multiplicación de ambos vectores.
    - Para cada fila de la primera matriz y para cada columna de la segunda matriz, invocar el método "multiplica_vectores". Al estar reservadas como arrays de punteros, sólo habrá que pasar el índice de las filas/columnas de las matrices cargadas anteriormente. El producto de las dos matrices se almacenará en la matriz resultado.

- **Escritura de resultado:** Siguiendo el formato de las matrices de entrada, se escribirá en un fichero el contenido de la matriz resultado. Primero se escribirán las filas, columnas y datos.

Segunda parte: Análisis de rendimiento

Para realizar el análisis se usará un "profiler" llamado "gprof". Éste programa realizará el análisis de uso de funciones de la implementación anterior, dando el porcentaje de uso de las partes del programa. Para poder usarlo es necesario seguir los siguientes pasos:

- Compilar con los flags "-g —pg", que permiten añadir las opciones de debug al programa.

- Hacer una ejecución "normal" del programa. Esta ejecución generará un archivo "gmon.out" que contiene los datos de llamadas a funciones en bruto.

- Ejecutar el programa gprof con el archivo gmon.out generado, para que lo interprete y muestre los resultados. El programa muestra los resultados por pantalla, por lo que se recomienda redirigir la salida a un fichero de texto.

- El modo de invocación es el siguiente:

#/home/user> gprof "nombre_aplicación" gmon.out > profiler.txt

Entre los datos mostrados está el porcentaje de uso de las funciones implementadas. El alumno deberá elegir la función más adecuada para optimizar el programa, justificándola con un análisis teórico que incluya el speedup máximo que se podría conseguir en condiciones óptimas.

Para el cálculo de speedup teórico se usará la Ley de Amdahl, que nos da el speedup teórico que se conseguiría en función de una mejora introducida:

$$\text{Speedup}_{\text{global}} = \frac{1}{(1 - \text{Fracción}_{\text{mejora}}) + \frac{\text{Fracción}_{\text{mejora}}}{\text{Speedup}_{\text{mejora}}}}$$

Dado que nuestro objetivo es mejorar la aplicación usando threads, el Speedup/mejora debería de ser el número de hilos/procesadores entre los que se dividirá la fracción del programa mejorado. Se deberá calcular el

speedup teórico usando 2, 4,6,8.... procesadores, y los datos se representarán usando una gráfica llamada "curva de escalabilidad" que mostrará en el eje horizontal el número de procesadores, y en el vertical el speedup obtenido.

Adicionalmente, se calculará el speedup teórico máximo, que representa el rendimiento máximo que se obtendría con "infinitos" procesadores.

Presentación de resultados

Para esta sección se generará una documentación que incluirá lo siguiente:

- Descripción de los datos de entrada y ejecuciones realizadas. Se deberá realizar tres ejecuciones con matrices de tamaños entre 100x100 y 10000x10000. El objetivo es poder ver como varían los Speedups teóricos en función del tamaño de datos de entrada.

- Para cada una de las tres ejecuciones anteriores:

    - Recoger los datos relevantes de porcentaje de uso con gprof.
    - Calcular la gráfica de escalabilidad para 2, 4 y 8 procesadores.
    - Calcular el speedup teórico máximo.

- Elegir la ejecución más adecuada (la que mejor speedup teórico máximo tenga) para las pruebas en las siguientes secciones de la práctica.

Junto con la documentación se presentará el código con la implementación. El programa deberá estar validado comparando las ejecuciones de multiplicación de matrices "identidad", en cuyo caso los datos de entrada deberían ser iguales a la salida.

La entrega se realizará a través de campus virtual antes del día **21 de marzo**. Se entregará un archivo zip con la documentación, código e implementación de los programas anteriormente descritos. No es necesario entregar las matrices de prueba. La entrega de la práctica se probará con el profesor el **día 21**, y se dará paso a la siguiente fase.