Práctica 2
Estudio y propuesta de la paralelización de una aplicación

Objetivos:

- Aprender a encontrar problemas y aplicaciones derivadas de **cualquier rama del conocimiento**, cuya **carga computacional sea tan elevada** como para justificar acometer un proceso de paralelización.
- Encontrar o diseñar una aplicación **secuencial** idónea para ser paralelizada en máquinas paralelas de memoria centralizada (es decir, con aplicaciones multihebra) o distribuida (aplicaciones multiproceso).
- Justificar mediante métricas **ya conocidas en otras asignaturas** porqué la aplicación en cuestión es idónea para su paralelización.
- Proponer arquitecturas idóneas para la paralelización de su aplicación (sistemas multi-núcleo, GP-GPU, *clusters*, sistemas heterogéneos, ...).
- Estudiar cómo pueden afectar los parámetros de compilación al rendimiento de la aplicación en una máquina paralela.
- Aplicar métodos y técnicas propios de esta asignatura para estimar las **ganancias máximas** y la **eficiencia** del proceso de paralelización.

Desarrollo:

Tarea 1: ¡ Búsquese un buen problema ;-) !

En esta primera práctica los/las estudiantes deberán afrontar un proceso de **búsqueda y selección de un problema**, de **cualquier índole y rama del conocimiento** (ingeniería, matemáticas, físicas, química, psicología, etc.), cuya solución venga dada en función de una aplicación software de elevado coste computacional. El problema en cuestión puede estar formulado desde cualquier ciencia o rama del conocimiento en general, y la búsqueda de información y consultas se deberá hacer por los cauces habituales: revistas, libros, Internet, tutorías, etc.

✓ **Nota 1.1**: No es necesario entender formalmente el problema concreto (que puede pertenecer al ámbito de una disciplina distinta a la Informática), pero sí se debe entender la **estructura de su implementación** y su **funcionamiento**. El programa secuencial debe estar escrito en C o C++ y **no debe ser interactivo**, es decir, tomará unos parámetros al inicio de su ejecución (línea de órdenes o un archivo de texto) y al cabo de un tiempo (suficientemente largo) entregará los resultados. En caso de optar por un programa interactivo (por ejemplo, el juego del ajedrez), se deberán enfrentar dos instancias del mismo programa para hacer el conjunto no interactivo y poder aplicar las métricas habituales.

✓ **Nota 1.2**: Esta tarea debe finalizarse en **2-3 horas** de tiempo de prácticas, al final de las cuales deberá comunicar el problema a su profesor de prácticas para que le dé el visto bueno al problema y poder seguir con el resto de las tareas.

✓ **Nota 1.3** Al final de la segunda sesión de prácticas cada grupo deberá comunicar:

- El título y una breve descripción del problema que van a estudiar
- Lista definitiva de todos los integrantes del grupo.

Esta información **se hará pública** en el Campus Virtual (Moodle)

✓ **Nota 1.4:** Le recomendamos las actas de cualquier edición de las Jornada de Paralelismo para buscar propuestas de paralelización que podrá adaptar a esta práctica:

http://www.jornadassarteco.org/programa/?anyo=2021
http://www.jornadassarteco.org/programa/?anyo=2019
http://www.jornadassarteco.org/programa/?anyo=2018
http://www.jornadassarteco.org/programa/?anyo=2017

Tarea 2: Estudio del problema

Se trata de **estudiar la codificación de su programa**, describiendo con máximo detalle cómo se ha resuelto el problema.

Tarea 2.1 Analizar el programa mediante un **grafo de control de flujo (CFG)** (ver: https://www.youtube.com/watch?v=9N5vPeSWRfQ). ¿Revela el CFG "trozos" de la aplicación cuya ejecución pueda paralela?
Tarea 2.2 Con respecto a las variables que use su programa: estudie cómo es el acceso de lectura/escritura a cada una y cómo son las variables que se usan (ejemplo: grandes matrices de datos, datos que raramente se acceden, muchas variables automáticas, ...) ¿Se puede **estructurar** el flujo del programa de forma más apropiada para una ejecución más eficiente en una máquina paralela? ¿Podemos prever algún problema con la caché?
Tarea 2.3 Estudie cómo puede **variar la carga** del problema para poder hacer un estudio de sus implicaciones en el rendimiento del programa.
Ejemplo: La ganancia en velocidad (*speed-up*) de su aplicación puede depender fuertemente de las dimensiones de una cierta matriz, de la exactitud de alguna salida, etc. Este tipo de dependencias son fácilmente representables mediante gráficas.

✓ **Nota 2.1:** La codificación de la aplicación puede ser original de cada grupo o tomada/adaptada de algún recurso bibliográfico siempre **debidamente referenciado**.

Tarea 3: "Sintonización" del compilador y análisis del rendimiento del programa

La implementación realizada tendrá que ejecutarse bajo el sistema operativo Linux (Ubuntu 20.04 de la EPS) y el compilador GCC (gcc/g++).

Tarea 3.1 Estudie y refleje en una tabla alguno de los parámetros y opciones de compilación que crea que más puedan beneficiar al rendimiento paralelo de la aplicación. Explique brevemente qué hace cada parámetro seleccionado.

Nota 3.1. Revise qué hace "`gcc –help=target`", y "`gcc --help=optimizers`" (ídem para g++ si se usa C++) en https://gcc.gnu.org/onlinedocs/gcc/Overall-Options.html

Tarea 3.2 Analicen y demuestren (mediante gráficas) qué combinación de parámetros y opciones de compilación "exprimen" o mejoran la salida que produce su compilador. Se puede lograr una cierta ganancia en velocidad si aprendemos a usar correctamente estos parámetros. En algunos problemas este efecto será más evidente que en otros. En su caso, ¿cómo es este efecto? ¿a qué cree que es debido? Ayuda: debe argumentar con conceptos como *caché*, uso más eficiente de la memoria, generación de instrucciones específica para su procesador, etc.

Tarea 3.3 ¿Cómo cambia a la ganancia en velocidad (*speed-up*) si variamos el parámetro X de nuestro problema (**X = parámetros estudiados en la Tarea 2**)? Genere gráficas del tipo: *Speed-up* o eficiencia *versus* parámetro_que_escala_nuestro_problema.

Tarea 3.4 Analice y estudie del rendimiento de la aplicación. Intente explicar a qué son debidas las variaciones en el *speed-up* como consecuencia de variar los parámetros que escalan el problema.

Tarea 5: Generación de los entregables:

Tarea 5.1 Memoria estructurada de la práctica con **TODOS** los apartados siguientes debidamente trabajados (siga las recomendaciones de su profesor de prácticas).

Es obligatorio incluir: objetivos, presentación del problema propuesto, estudio pormenorizado de la aplicación propuesta, análisis de rendimiento, defensa del programa ¿por qué es un buen candidato a paralelizar?, arquitecturas paralelas idóneas para la ejecución del programa debidamente paralelizado, bibliografía.

Tarea 5.2 Creen un fichero comprimido en ZIP con un ***Makefile*** y los fuentes necesarios para la compilación del programa (**si no es absolutamente necesario no entregue nunca binarios**).

Nota 5.1: Implemente por lo menos las siguientes reglas en su fichero m*akefile*:
`make clean` (limpiar directorio de trabajo)
`make` (construir el proyecto)
`make run` (ejecutar el programa con los parámetros por defecto).

Nota 5.2: La práctica se deberá entregar mediante el método que escoja su profesor de prácticas **antes** de la sesión de prácticas de la semana del 18-22 de octubre. Por tanto se dispondrá de 3 sesiones (6 horas) de prácticas en el laboratorio.

Nota:
Los trabajos teóricos/prácticos realizados han de ser originales. La detección de copia o plagio supondrá la calificación de "0" en la prueba correspondiente. Se informará la dirección de Departamento y de la EPS sobre esta incidencia. La reiteración en la conducta en esta u otra asignatura conllevará la notificación al vicerrectorado correspondiente de las faltas cometidas para que estudien el caso y sancionen según la legislación.