Práctica 1
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 justifique** acometer un proceso de paralelización → ¿Cuándo está justificado 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** por qué 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, big.LITTLE, multiprocesadores, ...).
- Estudiar cómo pueden afectar los parámetros de compilación al rendimiento de la aplicación en una máquina paralela.
- Aplicar métodos y técnicas propias de esta asignatura para estimar las **ganancias máximas** y la **eficiencia** del proceso de paralelización.

Desarrollo:

Tarea 1: Búsqueda o desarrollo de una buena aplicación candidata a ser paralelizada

En esta primera práctica el estudiantado deberá afrontar un proceso de **búsqueda o bien de diseño 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. La búsqueda de información se deberá hacer por los cauces habituales: revistas científicas, libros, Internet, tutorías, web de las Jornadas Sarteco, repositorio de Zenodo, búsqueda en repositorios públicos como github/gitlab, etc.

✓ **Nota 1.1**: No es necesario entender formalmente el problema concreto (ya que pudiera 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. Durante este tiempo deberán ir comunicando al profesor/a de prácticas el problema que están barajando para que éste/a le dé el visto y poder seguir así 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 revisar los títulos de las contribuciones a las Jornada de Paralelismo o las Jornadas de Computación Empotrada y Reconfigurable para encontrar propuestas de paralelización que podrá adaptar a esta práctica:

http://www.jornadassarteco.org/programa/?anyo=2024
http://www.jornadassarteco.org/programa/?anyo=2023
http://www.jornadassarteco.org/programa/?anyo=2022

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

Pueden descargar algunos artículos del congreso mediante la web de ZENODO: https://zenodo.org ---> busquen artículos relacionados con las comunidades (Communities) js2019, js2021, js2022, js2023 etc.

Recomendaciones a la Tarea 1:

- Un problema susceptible de ser paralelizado debe "tardar" un tiempo suficientemente alto para que tenga sentido proceder a su paralelización. Esto es debido a que los mecanismos que introducirá en su código (en las siguientes prácticas) de creación/destrucción de hebras o procesos, y de comunicación y sincronización, supondrán una sobrecarga temporal (*time overhead*) que puede no ser despreciable, haciendo que no tenga sentido acometer su paralelización.
- La multiplicación de matrices es un buen ejemplo de un problema totalmente paralelizable, sin embargo, este tipo de problemas, donde el 100% del algoritmo es paralelizable (100% de paralelismo de datos) no nos interesan en esta práctica. Busque un problema donde, cuando se paralelice, quede equilibrado el paralelismo de datos y otros tipos de paralelismo (revise las transparencias de la unidad 3 en el Campus Virtual).
- **Recuerde:** Se permite desarrollar programas sintéticos (diseñados por cada grupo).
- La codificación de la aplicación puede ser original de cada grupo o tomada/adaptada de algún recurso bibliográfico siempre **debidamente referenciado**.

Tarea 2: Estudio del problema previo a su paralelización

Se trata de **estudiar la codificación secuencial de su programa (tarea anterior)**, describiendo con máximo detalle las principales características del código.

Ayuda: En la defensa de esta parte, se debe demostrar porqué el código escogido es un buen candidato a ser paralelizado.

Tarea 2.1 Realice el **Grafo de Dependencia entre Tareas** de su código secuencial. ¿Revela el grafo "trozos"/tareas de la aplicación cuya ejecución pueda ser implementada de forma paralela (esto es, tareas cuyo orden de ejecución secuencial podemos desordenar y dejar invariante el resultado de la aplicación)?

Ejemplo:
Grafo de Dependencias Entre Tareas: $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \ldots T_n$, donde
T_1: Inicialización del programa.
T_2: Cálculo de la convolución de la matriz A y la matriz B …
T_3: ...

Tarea 2.2 Con respecto a las variables que usa 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, ...)
- Conteste: ¿Se puede **estructurar** el flujo del programa de forma más apropiada para una ejecución más eficiente en una máquina paralela?
- ¿Podemos prever algún problema/mejora con la **caché** modificando el programa de algún modo?

Ayuda: Deberá localizar cuáles son las variables calientes (las que más se acceden) y aquellas cuyo acceso es puntual.

Tarea 2.3 Estudio de variación de la carga computacional:

Estudie cómo puede **variar la carga** de su problema y su incidencia en el rendimiento del mismo. Este tipo de dependencias son fácilmente representables mediante gráficas, **inclúyalas**. Ejemplo: ¿Cómo cambia la ganancia en velocidad (*speed-up*) si variamos un cierto parámetro X de nuestro problema.

Genere gráficas del tipo:

Speed-up o eficiencia ***versus*** `parámetro_que_escala_nuestro_problema`.

Ayuda: 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.

Tarea 2.4 "Sintonización" de los parámetros de compilación

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

Deberán:

- Presentar evidencias de cómo afectan ciertos parámetros al desempeño de su aplicación (ejemplo: cambios en los tiempos de ejecución). Recuerde que todavía no estamos paralelizando, sólo **afinando** el proceso de compilación.
- Presentar evidencias de autovectorización en algunas partes de su programa. Esto es, mostrar que el compilador usa instrucciones SIMD para realizar ciertas tareas. **Ayuda**: Use "`gcc -S programa.c`" u "`objdump -j .text -D programa.elf`" para explorar qué hace el compilador.

Ayuda: 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 2.5 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 2.6 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 4: Precalentamiento para las siguientes prácticas:

Resuelva los siguientes problemas:

- Un grifo tarda 4 horas en llenar un cierto depósito de agua y otro grifo 20 horas en llenar el mismo depósito. Si usamos los dos grifos para llenar el depósito, que está inicialmente vacío, ¿cuánto tiempo tardaremos? ¿Cuál será la ganancia en velocidad? ¿Y la eficiencia?
- Suponga que tiene ahora 2 grifos de los que tardan 4 horas en llenar el depósito. Mismas cuestiones que el punto anterior.
- Y ahora suponga que tiene 2 grifos de los que tardan 20 horas. Proceda también a realizar los cálculos.
- Ahora tiene 3 grifos: 2 de los que tardan 20 horas y 1 de los que tardan 4. ¿Qué pasaría ahora?

Tarea 5: Generación de los entregables:

Memoria estructurada de la práctica con **TODOS** los apartados anteriores 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.

Implemente por lo menos las siguientes reglas en un fichero m*akefile*:

`make clean` (limpiar directorio de trabajo)

`make` (construir el proyecto)

`make run` (ejecutar el programa con los parámetros por defecto).

Nota 3.2: La práctica se deberá entregar mediante el método que escoja su profesor de prácticas **antes** de la sesión de prácticas de la semana del 7-11 de octubre. Por tanto se dispondrá de 4 sesiones (8 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.