***Objetivos:***

* Revisar problemas y aplicaciones derivadas de cualquier rama del conocimiento, actuales o no, cuya carga computacional sea tan elevada como para justificar acometer un proceso de paralelización. Elegir una aplicación **secuencial** candidata a ser paralelizada.

La práctica que se nos propone, tiene como objetivo la búsqueda y selección de aplicaciones secuenciales cuya carga computacional sea lo suficientemente alta como para sugerir su paralelización.

La utilización de todos los recursos disponibles y la capacidad de utilizar un mayor poder de computación para resolver problemas complejos o de grandes dimensiones, son las características más destacables a la hora de seleccionar aplicaciones para su paralelización.

En nuestro caso, será el re escalado de imágenes la aplicación que trataremos y será objetivo de paralelización con posterioridad. Existen distintos algoritmos que pueden ser utilizados para el re escalado de imágenes, siendo la interpolación bilineal el que trataremos en nuestra practica.

La interpolación bilineal consiste en la interpolación del valor de los colores de los cuatro pixeles adyacentes a los nuevos pixeles a generar en el aumento del tamaño una imagen, para poder conocer el color de estos. Según la posición y disposición de los pixeles conocidos, los nuevos pixeles pueden usar tanto los ejes diagonales como los ejes-xy. Dado su funcionamiento podemos observar que es una aplicación paralelizable al consistir básicamente en el cálculo del valor del color de todos los nuevos pixeles generados.

* Defender mediante métricas ya conocidas en otras asignaturas porqué la aplicación en cuestión es idónea para su paralelización.

Como se observa, nuestra aplicación es paralelizable tanto a nivel de datos, ya que para el calculo de cada pixel se tendrá que tener en cuenta los distintos pixeles adyacentes según la posición del nuevo pixel (además de su eje según los pixeles conocidos), como a nivel funcional ya que distintas tareas (cálculos iguales o distintos) pueden ser llevados simultáneamente (de forma paralela).

Algunas de las métricas que nos aportan información sobre su posible paralelización son:

El tiempo de ejecución: Métricas como función del tamaño del problema, procesadores disponibles, número de procesos, variables dependientes del algoritmo y las de características del hardware sobre el que se ejecuta.

Ley de Amdahl: que nos dice que existe una parte secuencial del programa que no podemos paralelizar (cota inferior del tiempo de ejecución) y una parte paralelizable mediante la cual reduciremos el tiempo de ejecución. Su complejidad crece con el numero de procesadores a usar.

Ley de Gustafson (problemas escalados, donde el tamaño del problema crece): que establece que cualquier problema suficientemente grande puede ser eficientemente paralelizado.

También podemos tener en cuenta la granularidad: Conforme aumenta la granularidad, disminuye el grado de paralelismo.

La contribución de la arquitectura al aumento de las prestaciones de los computadores ha venido de la mano del paralelismo y del aprovechamiento de la localidad de acceso a los datos.

Las principales razones para la construcción de máquinas paralelas son: disminuir el tiempo total de ejecución de una aplicación, conseguir resolver problemas más complejos, de grandes dimensiones, y proporcionar concurrencia, o sea, permitir la ejecución simultánea de tareas.

Podemos considerar como programación paralela la que se hace usando varios procesadores para acelerar la resolución de los problemas con que trabajemos. Trabajaremos con algoritmos paralelos, donde se especifica la forma en que los distintos elementos de proceso comparten los datos, y cómo colaboran en la resolución de los problemas. En general consideraremos que los procesadores son todos del mismo tipo, con lo que tendremos sistemas paralelos homogéneos.

* Proponer arquitecturas idóneas para la paralelización de su aplicación.

La clasificación más popular de computadores es la taxonomía de Flynn. Esta taxonomía de las arquitecturas está basada en la clasificación teniendo en cuenta el flujo de datos e instrucciones en un sistema.

La taxonomía de Flynn distingue **cuatro tipos de arquitecturas:**

**SISD:** Flujo único de instrucciones y flujo único de datos.

**SIMD:** Flujo único de instrucción y flujo múltiple de datos. Es este tipo el que con mas probabilidad sea el más idóneo para nuestro problema.

**MISD:** Múltiples instrucciones y flujo único de datos.

**MIMD:** Múltiples instrucciones y Múltiples flujos de datos.

También podemos hablar de la estructura lógica del programa (arquitectura de software), que determinara el grado de paralelización que puede llevarse a cabo en la aplicación. En ella podríamos destacar la dependencia de datos, ya que en nuestra aplicación el cálculo de los valores de los pixeles pueden venir determinados, tanto por los pixeles en las diagonales como por aquellos que aparecen en los ejes-xy según su disposición. Estos problemas podrían resolverse limitando los parámetros de entrada de la aplicación y en algunos casos a costa de empeorar el resultado final (algunos casos solo permiten cálculo de los pixeles en el eje-x).

* Estudiar cómo pueden afectar los parámetros de compilación al rendimiento de la aplicación en una máquina paralela.

Sabemos por Arquitectura de los computadores (asignatura predecesora) que los programas reales pueden afectan al rendimiento de, en este, caso nuestra aplicación. Dentro de los programas reales podemos encontrar los compiladores y sus opciones de compilación, que pueden ayudar a mejorar el rendimiento de una aplicación.

Las prioridades de un compilador en el momento de traducir lenguajes de lato nivel son: en primer lugar la exactitud de la traducción, en segundo lugar la velocidad del código generado, en tercer lugar el tamaño del código, en cuarto lugar la velocidad del compilador, y en quinto lugar el soporte a la depuración.

Tratando las optimizaciones a nivel de arquitecturas podemos ver las diferencias entre las arquitecturas CISC y RISQ teniendo en cuenta que cuanto más registros, mas fácil es optimizar el rendimiento, para lo que el compilador deben conocer los costes relativos entre las diversas instrucciones y elegir la mejor secuencia de instrucciones dado que CISC consta ofrece más alternativas, mientras RISC intenta limitarlas y conseguir una longitud constante en ellas.

Las ayudas más importantes que la arquitectura de un procesador puede prestar al diseño del compilador son: Regularidad del repertorio: simplificar la generación de código mediante la ortogonalidad de los elementos en él y proporcionar funciones primitivas.

A nivel de paralelismo podemos conseguir optimizaciones reordenando las operaciones para permitir que múltiples cálculos se produzcan en paralelo, ya sea en la instrucción, la memoria o a nivel de hilo.

Tratando el tema que nos ocupa hablaremos sobre las optimizaciones a nivel de parámetros en GCC.

Sin ninguna opción de optimización, el objetivo del compilador es el de reducir el coste de compilación y producir los resultados esperados. Muchas de las optimizaciones en GCC solo están habilitadas al introducir -0 en la línea de comandos (como si fuesen niveles). Dependiendo del “target” y de como GCC haya sido optimizado pueden activarse diferentes optimizaciones.

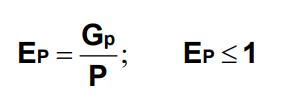
El objetivo de estas opciones, en general, es el de mejorar el espacio y tiempo mediante el sacrificio de otros aspectos, como el tiempo de compilación. Mas adelante trataremos una serie de parametros que nos pueden ser de ayuda en nuestra aplicación.

* Métodos y técnicas propios de esta asignatura para estimar las ganancias máximas y la eficiencia del proceso de paralelización.

**Medidas de rendimiento:**

* **Eficiencia:** Cuando hablamos de medición de la eficiencia para esta asignatura, debemos recordar lo visto en su predecesora Arquitectura de los computadores. En esta ultima vimos como podíamos medir la eficiencia mediante el tiempo de ejecución (o respuesta) de una aplicación y la productividad (número de aplicaciones por unidad de tiempo).

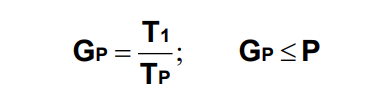
En el material proporcionado en esta nueva asignatura podemos encontrar las siguientes formulas para su cálculo, haciendo uso de la ganancia, además de un repaso a los citados métodos vistos en AC.



También podemos introducir nuevas conceptos como el speed-up, el speed-up es usado para indicar el grado de ganancia de velocidad de una computación paralela, mientras la eficiencia mide la porción útil del trabajo total realizado por los procesadores.

El speed-up es usado en los cálculos de la ley de Amdahl y la ley de Gustafson, anteriormente citadas. La parte paralelizable de la aplicación debe ser lo mayor posible según estas leyes, dado que es en esta donde se produce la mejora del rendimiento.

* **Ganancia:** Es la medida que usamos para representar el aumento de velocidad obtenido en la parte paralelizable del problema (cuanto mayor mejor).



* Bibliografia:

http://www.uv.es/varnau/OC\_T4.pdf

<http://dis.um.es/~domingo/09/ERBASE/minicursoespanol.pdf>

<http://www.uv.es/varnau/OC_T4.pdf>

<https://www.infor.uva.es/~bastida/Arquitecturas%20Avanzadas/General.pdf>

<https://es.wikipedia.org/wiki/Wikipedia>

<http://www.dte.eis.uva.es/Docencia/ETSII/SMP/BAK/tema4/ANEXO4.pdf>

http://www.fdi.ucm.es/profesor/jjruz/WEB2/Temas/EC4.pdf

Diapositivas de la asignatura proporcionadas por los profesores y de la asignatura AC.