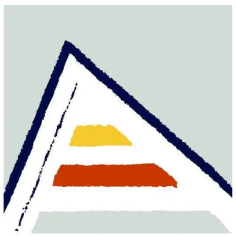
INGENIERÍA DE LOS COMPUTADORES
UNIVERSIDAD DE ALICANTE
CURSO 2023/2023



Universitat d'Alacant
Universidad de Alicante

PRÁCTICA 2/LABORATORIO 2

Estudio y propuesta de paralelización de una aplicación

PROFESOR
Pau Climent Pérez, pau.climent@gcloud.ua.es

AUTORES
Alberto Vicente Cánovas, avc86@gcloud.ua.es
Javier Sala Pérez, jsp67@gcloud.ua.es

ÍNDICE

1.- Búsqueda/Desarrollo de un buen candidato a ser paralizado

En esta tarea, nos enfocaremos en encontrar un problema que se preste para la paralelización. En este contexto, exploraremos la resolución de sistemas de ecuaciones lineales utilizando una variante paralela del método de Gauss.

1.1.-Objetivo

Nuestro objetivo es desarrollar una solución paralela eficiente para resolver sistemas de ecuaciones lineales. Esta técnica es fundamental en áreas como la ingeniería, la física y la ciencia de datos, y buscamos optimizar su rendimiento utilizando la paralelización.

1.2. Implementación

Nuestro programa recibirá una matriz de coeficientes y un vector de términos independientes que definen el sistema de ecuaciones lineales. A partir de estos datos, aplicaremos una versión paralela del método de Gauss para encontrar la solución del sistema.

1.2.1. Inicio

Se genera una matriz mat con valores aleatorios para simular un sistema de ecuaciones lineales. Los valores de la matriz se llenan con números enteros aleatorios entre 0 y 2000.

1.2.2. Proceso Secuencial/Paralelizado

Eliminación Gaussiana (Paralelizado) : En la etapa de eliminación gaussiana del programa, se realiza una serie de pasos para transformar la matriz original del sistema de ecuaciones en una forma triangular superior. Este proceso implica iterar a través de las filas de la matriz y realizar operaciones secuenciales en las filas para eliminar los elementos por debajo de la diagonal principal. A medida que avanzas en las filas, se calculan las relaciones y se aplican las operaciones necesarias para lograr la forma triangular superior.

Sustitución hacia Atrás (Secuencial): Una vez que la matriz se ha transformado en una forma triangular superior, se procede a la sustitución hacia atrás. En esta etapa, se calculan las soluciones para las incógnitas del sistema de ecuaciones. El proceso de sustitución hacia atrás se realiza de manera secuencial, comenzando desde la última ecuación y trabajando hacia atrás. Para cada ecuación, se calcula el valor de una incógnita utilizando

los valores de las incógnitas previamente encontradas y los coeficientes en la fila correspondiente de la matriz.

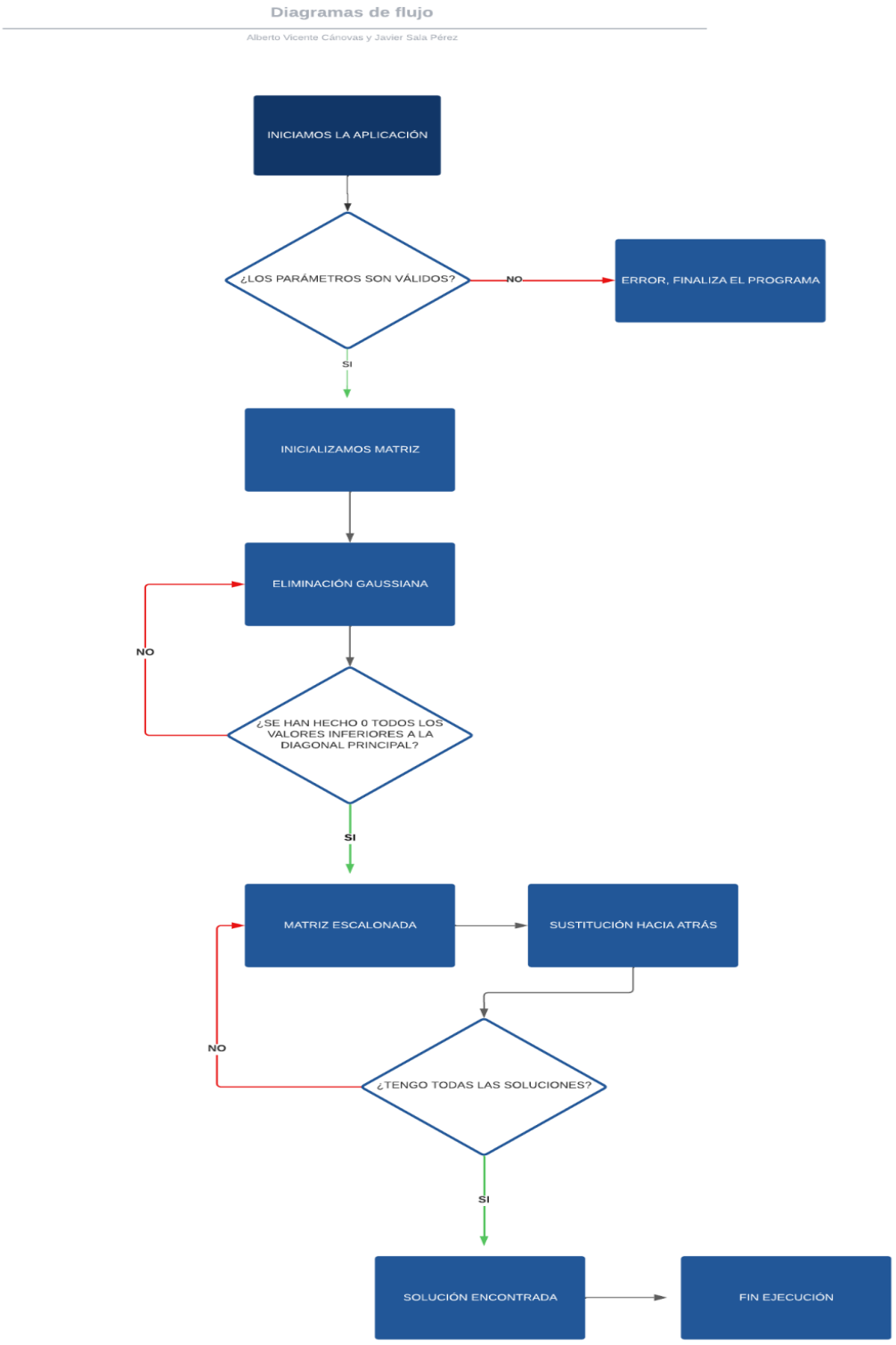
1.2.3. Resolución

Resultados: Una vez que se han encontrado las soluciones del sistema de ecuaciones, el programa imprime estas soluciones en la salida estándar. Las soluciones se muestran con un formato que incluye el valor de cada incógnita. Cada solución se imprime en una línea diferente junto con su etiqueta correspondiente (por ejemplo, "x0 = valor").

Medición del Tiempo de Ejecución: Para medir el tiempo de ejecución del programa, se utiliza la función clock() al comienzo y al final del programa. El tiempo transcurrido entre estas dos llamadas a clock() se calcula y se convierte en segundos. El tiempo de ejecución se muestra en la salida del programa para proporcionar información sobre cuánto tiempo llevó la ejecución.

2.- Estudio del problema previo a su paralelización

2.1.-Grafo de Dependencias entre tareas



2.2.-Análisis de Acceso a Variables y Optimización para Ejecución en Paralelo

1. Variable matriz con nombre “Mat” que se define con el valor del parámetro que se introduzca por consola y ese es su tamaño. Después se inicializa con la función rand()%2001 con número entre 0 y el 2000.
2. Variable auxiliar A que coge el valor por el que tenemos que multiplicar la fila i+1 para que con la resta de i - i+1 realizar 0.
3. Tres variables contadores para los bucles que son i, j y step que sirven como condiciones en los for.
4. Mat_size que representa el tamaño de la matriz.
5. Variable auxiliar sum almacena todas las soluciones del sistema.
6. Variable chrono que nos va a devolver el tiempo de ejecución haciendo una resta del tiempo end - tiempo start.

En cuanto a una estructuración más apropiada para lograr una ejecución más eficiente en una máquina paralela podríamos plantear alguna de las siguientes opciones:

-División de tareas. En la inicialización, al conocer un valor de la matriz que unos hilos se encarguen de almacenarlo mientras que los otros continúan calculando valores. De forma similar, podría hacerse en las operaciones de eliminación gaussiana y sustitución hacia atrás.

-Balance de carga óptimo. Dividir el número de tareas a realizar entre el número de hilos que dispongamos.

Hay que tener varios aspectos a tener en cuenta a la hora de uso de la memoria caché, uno de los más importantes en el tamaño de la caché ya que este es limitado y quizá no pueda manejar las necesidades del programa. Otro de los problemas más comunes son los conflictos de caché, cuando varios hilos intentan acceder a posiciones de memoria que mapean en la misma línea de caché.

2.3.-Estudio de variación de la carga

NUMERO DE VARIABLES	TIEMPO NORMAL
100	0,0082
250	0,0377
500	0,1744
1000	0,8693
1500	2,2593
2000	4,14016
4000	24,1433

Estas mediciones han sido realizadas con el modo normal, y podemos observar un crecimiento exponencial en el tiempo de ejecución según el tamaño del problema. Pasando

de 2000 a 4000 supone un crecimiento en la talla del 100%, pero en cuanto al tiempo vemos un incremento de casi el 600%.

2.4.-“Sintonización” de los parámetros de compilación

NUMERO DE VARIABLES	TIEMPO OPA	TIEMPO NORMAL
100	0,0072	0,0082
250	0,0327	0,0377
500	0,1577	0,1744
1000	0,7871	0,8693
1500	1,8432	2,2593
2000	3,3051	4,14016
4000	21,5678	24,1433

En cuanto al tiempo de ejecución con compilación normal, con el parámetro -O0, esto tiene un desempeño deficiente ya que el compilador no realiza ninguna opción de aceleración. Esto produce un mayor tiempo de compilación y de ejecución.
Para el caso de la ejecución OPA, optimización alta con el parámetro -03, este produce una serie de optimizaciones que mejoran la ejecución del programa. El tiempo de compilación será mayor puesto que tiene que realizar una mayor serie de tareas entre ellas: posible aumento de recursos CPU, intento de reducción del código…etc
Finalmente alguna de las optimizaciones que hace en nuestro sistema:

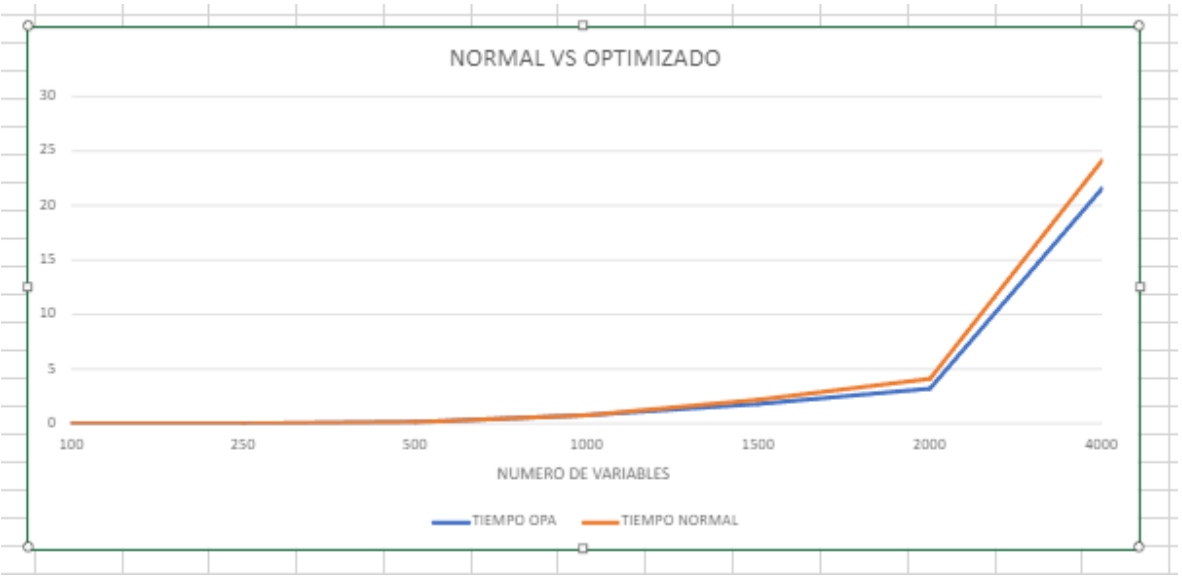
-Instrucciones y datos en Caché: Un mejor reordenamiento de las instrucciones facilita el acceso a datos y minimiza la posibilidad de error en caché.

-Vectorización: Transforma bucles para utilizar instrucciones SIMD y procesar múltiples datos en paralelo.

-Reordenamiento de instrucciones: Un mejor reordenamiento de las instrucciones puede provocar que se aprovechen mejor las unidades de la CPU minimizando los cuellos de botella.

-Optimización de registros: El compilador puede intentar asignar variables a registros de CPU en lugar de almacenarlas en la memoria. Esto reduce la latencia de acceso a la memoria, lo que a su vez mejora el rendimiento.

2.5.-Optimización de Parámetros de Compilación y Rendimiento: Análisis y Demostración



Como hemos comentado previamente el uso de parámetros puede mejorar el rendimiento del programa y así lo vemos reflejado en esta segunda gráfica donde está la comparación de tiempo del modo normal frente al modo optimizado. Para tamaños de problema relativamente bajos no existe prácticamente una gran diferencia en el tiempo pero a medida que sobrepasamos la cifra de 1500 se va haciendo más notable. Llegando al tamaño de 4000 donde existe una diferencia de 2,5755 segundos y hay una ganancia de casi el 11%

.

2.6.-Análisis de Rendimiento y Variación del Speed-Up en la Aplicación

El rendimiento de la aplicación se ve influenciado por el tamaño del problema y el nivel de asignación. Observamos que el tiempo de ejecución aumenta a medida que el tamaño del problema se incrementa. Además, el nivel de optimización (-O3) mejora significativamente el tiempo de ejecución respecto a la compilación sin optimizaciones (-O0)

Cálculo de speed-up con la comparación entre la compilación con y sin optimización:

Speed-Up = Tiempo sin optimización (T(O0)) / Tiempo con optimización (T(O3))

Speed-Up para 100 variables: T(O0) / T(O3) = 0.0082 / 0.0072 ≈ 1.14
Speed-Up para 250 variables: T(O0) / T(O3) = 0.0377 / 0.0327 ≈ 1.15
Speed-Up para 500 variables: T(O0) / T(O3) = 0.1744 / 0.1577 ≈ 1.11
Speed-Up para 1000 variables: T(O0) / T(O3) = 0.8693 / 0.7871 ≈ 1.10
Speed-Up para 1500 variables: T(O0) / T(O3) = 2.2593 / 1.8432 ≈ 1.23
Speed-Up para 2000 variables: T(O0) / T(O3) = 4.1401 / 3.3051 ≈ 1.25
Speed-Up para 4000 variables: T(O0) / T(O3) = 24.1433 / 21.5678 ≈ 1.12

El speed-up se calcula en función del tiempo de ejecución, por lo que valores mayores que 1 indican una mejora en el rendimiento con la compilación optimizada, lo que significa que en los casos anteriores tenemos una mejora de entre el 10-25%.

4.- 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?
Aspectos a tener en cuenta previos a resolver el problema.
El grifo A, llena un 25% por hora.
El grifo B, llena un 5% por hora.
Entonces si usamos ambos grifos al mismo tiempo estaríamos llenando un 30% por hora.
Como la capacidad es del 100%, el tiempo sería de 3,33 horas.
La ganancia de velocidad es la división entre la velocidad combinada y la velocidad del grifo más lento. (3/10)/(1/20) = 6
La eficiencia la calculamos con el cociente entre la velocidad combinada y la suma de las velocidades individuales. (3/10)/(1/4 + 1/20) = 1, es decir es un 100% más eficiente.

-Suponga que tiene ahora 2 grifos de los que tardan 4 horas en llenar el depósito. Mismas cuestiones que el punto anterior.
El grifo A, llena un 25% por hora.
El grifo B, llena un 25% por hora.
Si usamos ambos grifos, estaríamos llenado el 50% por hora lo que tardaría 2 horas en llenarse por completo.

Si usamos un grifo necesitaríamos 4 horas, al usar 2 podríamos hacerlo sólo en 2 horas.
Por lo cual es el doble de rápido.
La eficiencia es de (5/10) / (2,5/10 + 2,5/10) = 1, es decir es un 100% más eficiente.

-Y ahora suponga que tiene 2 grifos de los que tardan 20 horas. Proceda también a realizar los cálculos.
El grifo A, llena un 5% por hora.
El grifo B, llena un 5% por hora.
Si usamos ambos grifos, estaríamos llenado el 10% por hora lo que tardaría 10 horas en llenarse por completo.
Si usamos un grifo necesitaríamos 20 horas, al usar 2 podríamos hacerlo sólo en 10 horas.
Por lo cual es el doble de rápido.
La eficiencia es de (0,1/10) / (0,05/10 + 0,05/10) = 1, es decir es un 100% más eficiente.

- Ahora tiene 3 grifos: 2 de los que tardan 20 horas y 1 de los que tardan 4. ¿Qué pasaría ahora?
El grifo A, llena un 10% por hora.
El grifo B, llena un 25% por hora.
Si usamos todos los grifos, estaríamos llenado el 35% por hora lo que tardaría 2,86 horas en llenarse por completo.
La ganancia de velocidad es de 2,86.
Por lo cual es el doble de rápido.
La eficiencia es de (7/20) / (1/4 + 1/20 + 1/20) = 1, es decir es un 100% más eficiente.