# Secuencial

El algoritmo secuencial es muy similar al de la entrega 1. Primero se hace el chequeo de parámetros, la alocación y posterior inicialización de las matrices que correspondan.

La primera decisión que tomamos es a la hora de calcular R1, R2 y sus respectivos promedios.

Decidimos (por lo mostrado en la versión 2), calcular R1 y R2 en un solo recorrido de ambos, y aprovechar la estructura de control para calcular la sumatoria de las mismas (por lo mostrado en la versión 3).

Luego, calculamos el promedio de cada matriz, y guardamos la multiplicación en una sola variable para su posterior uso en el cálculo de C.

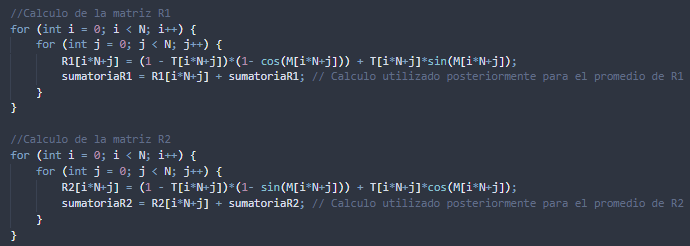
Lo que le sigue es la multiplicación de la matrices, que como vimos en la entrega 1, utilizando multiplicación por bloques es la mejor forma de realizarla. En particular usamos un tamaño de bloque de 64 (el óptimo, demostrado en la entrega 1) y organizamos A y B por columnas.

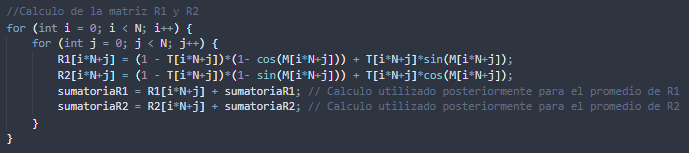
Por último queda el cálculo de C, el cual se puede hacer todo junto o separar ((R1A[i\*N+j] + R2B[i\*N+j]) \* promedioR1R2).

Como demostramos en la entrega 1, la forma más eficiente de realizarlo es todo junto, así que fue la alternativa por la que optamos.

## Versión 2

Versión 2: la versión 2 aplica el cambio a la hora de calcular R1 y R2, en la versión óptima se calculan ambas en un mismo for, en la versión 2 se separan sus cálculos:

Código versión 2:

Código versión óptima:

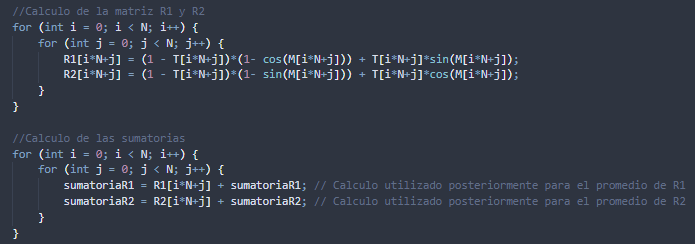
Cálculo de eficiencia:

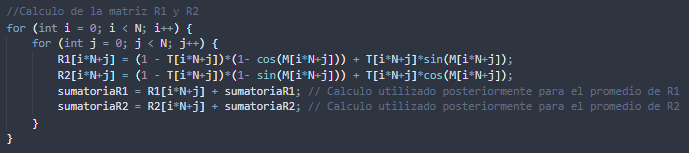
|  |  |  |
| --- | --- | --- |
| N (bs = 64) | Version 2 | Versión óptima |
| 512 | 0.587962 | 0.561895 |
| 1024 | 4.454791 | 4.352408 |
| 2048 | 35.367075 | 34.835465 |
| 4096 | 271.148718 | 269.095233 |

Conclusión: la versión óptima tiene mejor desempeño, ya que es innecesario hacer 2 for’s, se tarda ligeramente más.

## Versión 3

La versión óptima realiza el cálculo de la sumatoria de R1 y R2 dentro del for que inicializa a esas matrices, en cambio, la versión 3 lo hace en un for aparte, una vez ya han sido calculadas.

Código de la versión 3:

Código de la versión óptima:

Cálculo de eficiencia:

|  |  |  |
| --- | --- | --- |
| N (bs = 64) | Version 3 | Versión óptima |
| 512 | 0.563209 | 0.561895 |
| 1024 | 4.366573 | 4.352408 |
| 2048 | 34.931242 | 34.835465 |
| 4096 | 269.233418 | 269.095233 |

Conclusión: es mejor realizar las sumatorias dentro del primer doble-for, sin tener que crear otro, así aprovechamos el recorrido de las matrices.

Las otras diferencias de código que se pueden aplicar son idénticas a las de la entrega 1:

* Multiplicación por bloques por filas (en lugar de por columnas)
* Multiplicación simple (en lugar de por bloques)
* Cálculo de C en 2 segmentos (en lugar de un solo doble-for). En particular el cálculo de C varía ligeramente, pero sigue cumpliendo la estructura de 2 matrices (en la nueva entrega es R1A y R2B, pero antes RA y RB) por un double (en la nueva entrega promedioR1R2, pero antes promedio R) sumado a una matriz (en ambos casos T), por lo que no va a variar el tiempo de ejecución.

En los 3 casos, las decisiones tomadas en nuestro código son las óptimas (como está demostrado en la entrega 1).

# Pthread

Pthread nos permite hacer uso de los hilos, lo cual nos va a permitir paralelizar el algoritmo y obtener mejores tiempos de ejecución.

La primera sección de código que se ve afectada es el cálculo de R1, R2 y sus respectivas sumatorias. A diferencia de en la versión secuencial, cada hilo va a resolver un conjunto de iteraciones, en nuestro caso el conjunto de iteraciones se parte en partes iguales para cada hilo, ya que son independientes entre sí.

En particular, decidimos dejar junto el cálculo de R1 y R2, por lo mostrado en la versión 2.

Otra alternativa interesante que probamos en la versión 4 fue separar la sumatoria de R1 y R2 de su inicialización.

Luego, para el cálculo de C, también probamos juntando y separando sus terminos en diferentes iteraciones, pero como era de esperar por las pruebas hechas en la entrega anterior, es mejor la resolución del cálculo en un mismo doble-for.

## Versión 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N (bs = 64) | Version 2 | | | Versión óptima | | |
| T = 2 | T = 4 | T = 8 | T = 2 | T = 4 | T = 8 |
| 512 | 0.294236 | 0.149011 | 0.078556 | 0.291988 | 0.147274 | 0.075757 |
| 1024 | 2.240393 | 1.130371 | 0.579689 | 2.230519 | 1.126372 | 0.578590 |
| 2048 | 17.680950 | 8.912965 | 4.517133 | 17.617868 | 8.898398 | 4.521590 |
| 4096 | 135.647212 | 68.280776 | 35.155932 | 135.592896 | 68.233959 | 34.857703 |

## Versión 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N (bs = 64) | Version 2 | | | Versión óptima | | |
| T = 2 | T = 4 | T = 8 | T = 2 | T = 4 | T = 8 |
| 512 | 0.292167 | 0.147304 | 0.076220 | 0.291988 | 0.147274 | 0.075757 |
| 1024 | 2.233051 | 1.126864 | 0.580074 | 2.230519 | 1.126372 | 0.578590 |
| 2048 | 17.637740 | 8.891310 | 4.522330 | 17.617868 | 8.898398 | 4.521590 |
| 4096 | 135.622566 | 68.234258 | 34.889618 | 135.592896 | 68.233959 | 34.857703 |

# OpenMP

|  |  |  |
| --- | --- | --- |
| Versión óptima | | |
| T = 2 | T = 4 | T = 8 |
| 0.292288 | 0.147484 | 0.075324 |
| 2.233557 | 1.126982 | 0.578433 |
| 17.620263 | 8.928093 | 4.526614 |
| 135.918132 | 68.412556 | 34.955603 |

## Versión 2

## Versión 3

Preguntas:

El pthreadVersion3 lo ponemos? Porque no es más eficiente y como que esa prueba ya la hicimos

El cálculo óptimo de bs nos dio 32 en vez de 64, que onda?

# Comparación entre modelos

Comparación de las mejores versiones de cada modelo:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N (bs = 64) | Versión secuencial | Versión Pthread | | | Versión OpenMP | | |
| T = 2 | T = 4 | T = 8 | T = 2 | T = 4 | T = 8 |
| 512 | 0.561895 | 0.291988 | 0.147274 | 0.075757 | 0.292288 | 0.147484 | 0.075324 |
| 1024 | 4.352408 | 2.230519 | 1.126372 | 0.578590 | 2.233557 | 1.126982 | 0.578433 |
| 2048 | 34.835465 | 17.617868 | 8.898398 | 4.521590 | 17.620263 | 8.928093 | 4.526614 |
| 4096 | 269.095233 | 135.592896 | 68.233959 | 34.857703 | 135.918132 | 68.412556 | 34.955603 |

# SpeedUp

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N (bs = 64) | SpeedUp - Versión Pthread | | | SpeedUp - Versión OpenMP | | |
| T = 2 | T = 4 | T = 8 | T = 2 | T = 4 | T = 8 |
| 512 | 1.924377 | 3.815303 | 7.417070 | 1.922401 | 3.809870 | 7.459707 |
| 1024 | 1.951298 | 3.864094 | 7.522439 | 1.948644 | 3.862003 | 7.524480 |
| 2048 | 1.977280 | 3.914801 | 7.704251 | 1.977011 | 3.901781 | 7.695700 |
| 4096 | 1.984582 | 3.943714 | 7.719821 | 1.979833 | 3.933418 | 7.698200 |

# Eficiencia

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N (bs = 64) | SpeedUp - Versión Pthread | | | SpeedUp - Versión OpenMP | | |
| T = 2 | T = 4 | T = 8 | T = 2 | T = 4 | T = 8 |
| 512 | 0.962189 | 0.953826 | 0.927134 | 0.961201 | 0.952468 | 0.932463 |
| 1024 | 0.975649 | 0.966024 | 0.940305 | 0.974322 | 0.965501 | 0.940560 |
| 2048 | 0.988640 | 0.978700 | 0.963031 | 0.988506 | 0.975445 | 0.961963 |
| 4096 | 0.992291 | 0.985929 | 0.964978 | 0.989917 | 0.983355 | 0.962275 |

# Conclusión

Como conclusión podemos comentar que el uso de hilos provee un incremento altísimo de la performance. En nuestro caso los valores de speedup y eficiencia se mantienen altos a medida que aumentamos la cantidad de procesadores utilizados para resolver el problema, lo que es óptimo para lograr mejores tiempos de ejecución. El poder mantener estos valores altos se debe principalmente a que en nuestro código se encuentran varias regiones paralelizables, pero no siempre vamos a correr con la misma suerte, incluso en este problema se puede apreciar que el cálculo del promedio es algo que se hace de manera secuencial.

En cuanto a las tecnologías utilizadas, se puede apreciar como los tiempos de Pthread son ligeramente mejores que los de OpenMP, lo cual es entendible ya que estamos haciendo un uso directo de las herramientas que nos provee. En cambio, cuando se utiliza la librería OpenMP, en esta conversión o pasaje a Pthread que realiza la performance se ve ligeramente afectada. Sin embargo, el código programado con OpenMP es ampliamente más fácil de escribir y de entender, lo cuál nos provee una gran ventaja a la hora de desarrollar, quedará en cada proyecto analizar cual es la conveniencia y la prioridad que se le da a la ventaja que provee cada herramienta, pero en terminos generales OpenMP se podría considerar como la mejor alternativa.