|  |  |  |
| --- | --- | --- |
| **2º curso / 2º cuatr.**  **Grado Ing. Inform.** |  | **Arquitectura de Computadores (AC)**  **Cuaderno de prácticas.**  **Bloque Práctico 5. Optimización de código**  Estudiante (nombre y apellidos): Alberto Llamas González  Grupo de prácticas y profesor de prácticas: D3, Juan Carlos Gómez López  Fecha de entrega:  Fecha evaluación en clase: 7/06/2021 |

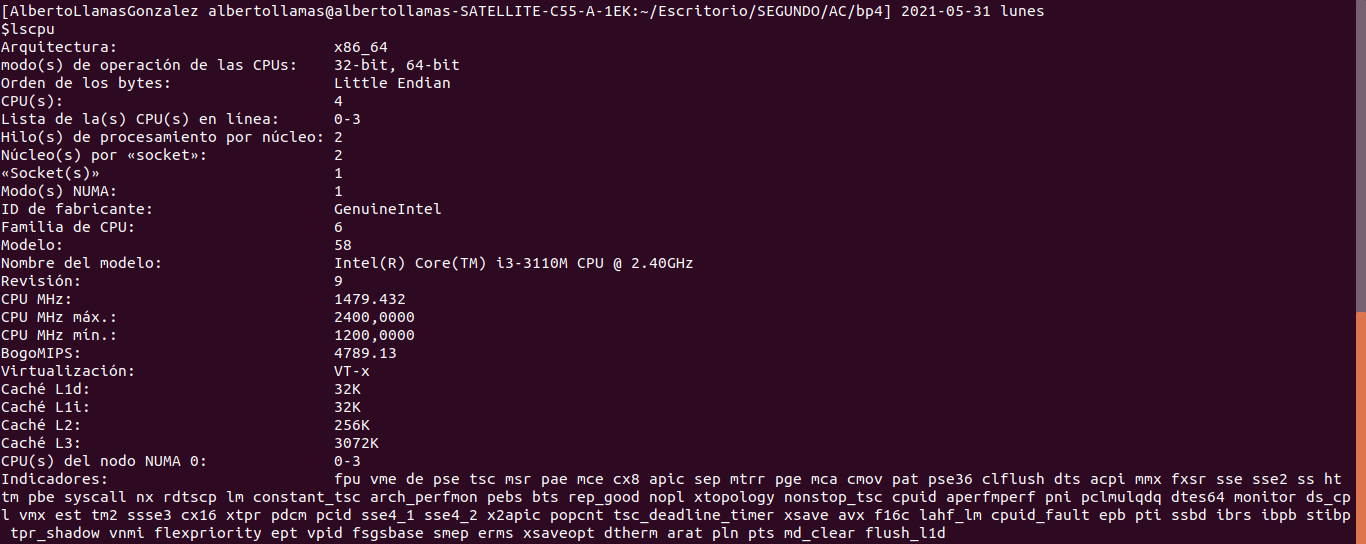
Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i3-3110M CPU @ 2.40GHz*

Sistema operativo utilizado: *Ubuntu 18.04.5 LTS "bionic"*

Versión de gcc utilizada: *gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)*

Volcado de pantalla que muestre lo que devuelve lscpu en la máquina en la que ha tomado las medidas:



1. **(a)** Implementar un código secuencial que calcule la multiplicación de dos matrices cuadradas. Utilizar como base el código de suma de vectores de BP0. Los datos se deben generar de forma aleatoria para un número de filas mayor que 8, como en el ejemplo de BP0, se puede usar drand48()).

**MULTIPLICACIÓN DE MATRICES:**

**CAPTURA CÓDIGO FUENTE**: pmm-secuencial.c

|  |
| --- |
| fos/ejer1/Captura%20de%20pantalla%20de%202021-06-04%2012-01-04.pngfos/ejer1/Captura%20de%20pantalla%20de%202021-06-04%2012-01-14.pngfos/ejer1/Captura%20de%20pantalla%20de%202021-06-04%2012-01-23.pngfos/ejer1/Captura%20de%20pantalla%20de%202021-06-04%2012-01-31.pngfos/ejer1/Captura%20de%20pantalla%20de%202021-06-04%2012-01-39.png |

**(b)** Modificar el código (solo el trozo que calcula la multiplicación) para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre -O2) a partir de la modificación realizada. Incorporar los códigos modificados en el cuaderno.

**MODIFICACIONES REALIZADAS (al menos dos modificaciones):**

**Modificación A) –explicación-:** desenrollado de bucles, con 4 cálculos por iteración, teniendo en cuenta que el tamaño N no tiene por qué ser múltiplo de 4 (desenrollamos el for hasta N-N%4 y luego calculamos de forma independiente, fuera del for, desde N%4 hasta N).

**Modificación B) –explicación-:** cambio en el orden de ejecución: i,k,j en lugar de i,j,k, para disminuir las penalizaciones por caché.

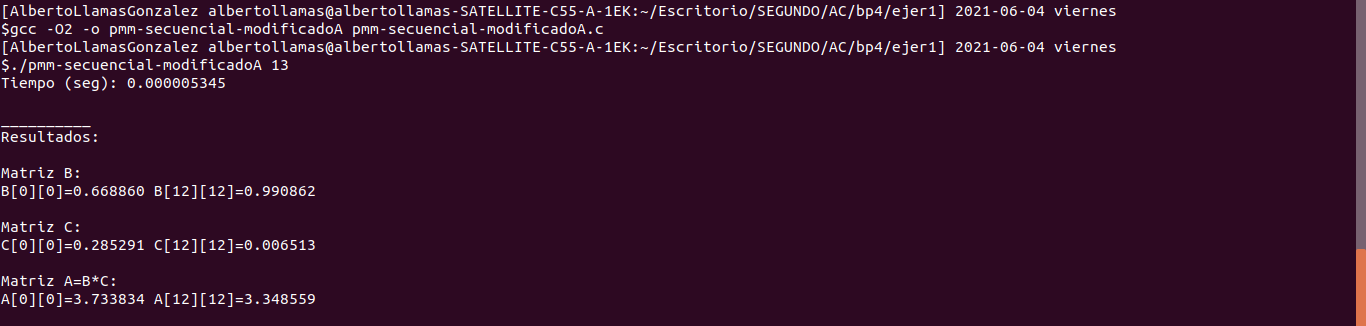
**CÓDIGOS FUENTE MODIFICACIONES**

**A) Captura de** pmm-secuencial-modificado\_A.c

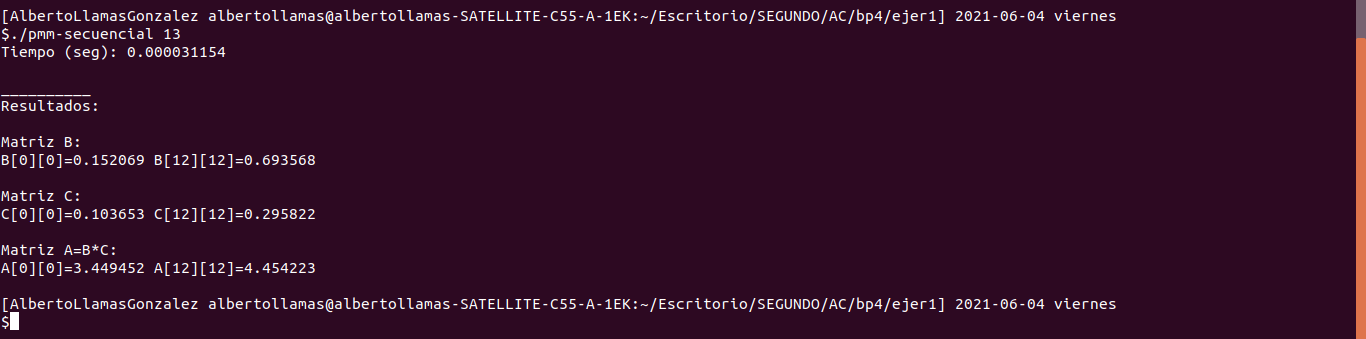
Se muestra solo el código que se ha modificado respecto al programa inicial:

|  |
| --- |
| fos/ejer1/modA/Captura%20de%20pantalla%20de%202021-06-04%2012-31-46.png |

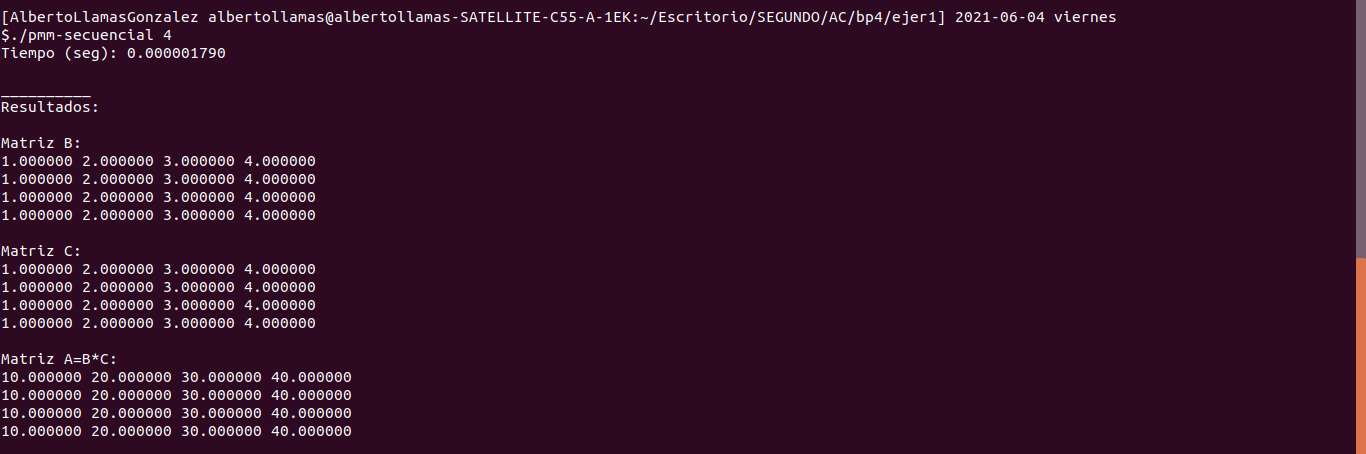
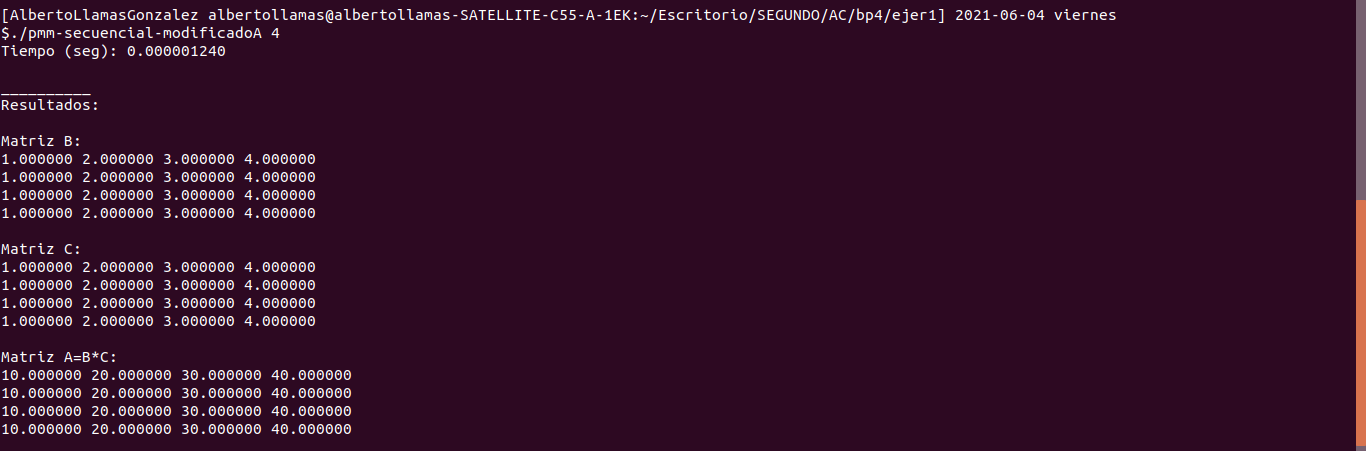
**Capturas de pantalla (que muestren la compilación y que el resultado es correcto):**

****

**Vemos que disminuye el tiempo respecto al programa sin modificar:**

****

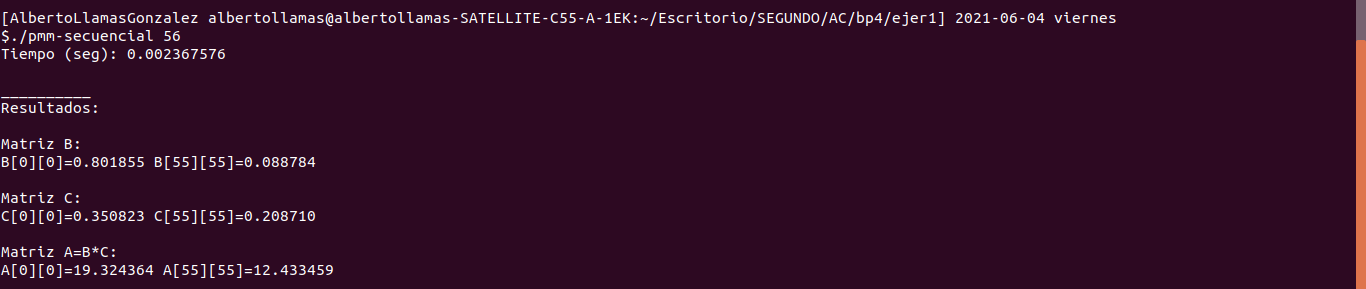
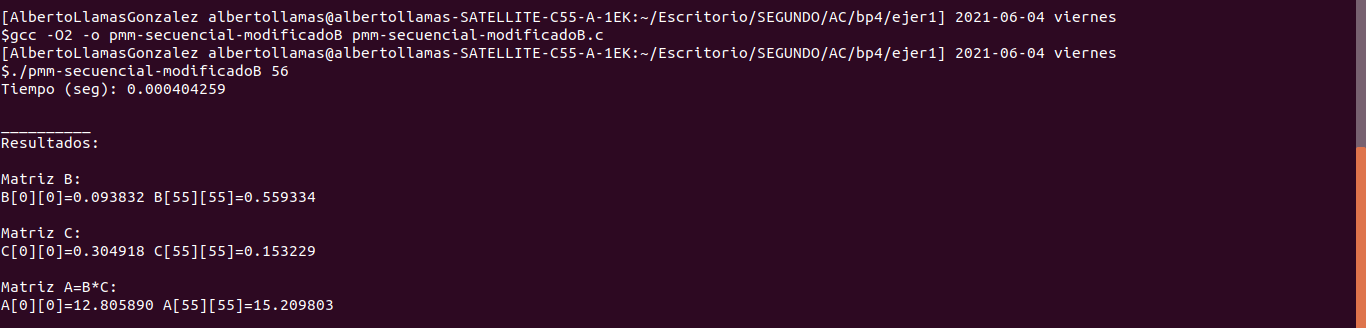
**Añadimos otra prueba con un valor < 9 para comprobar que calcula bien los resultados de hacer A=B\*C**

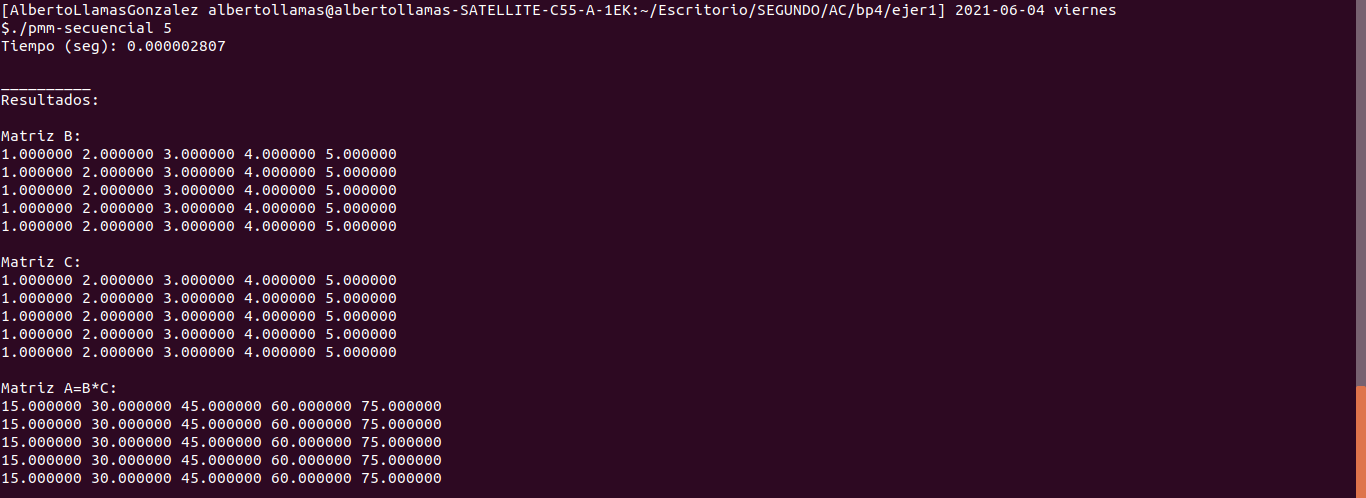
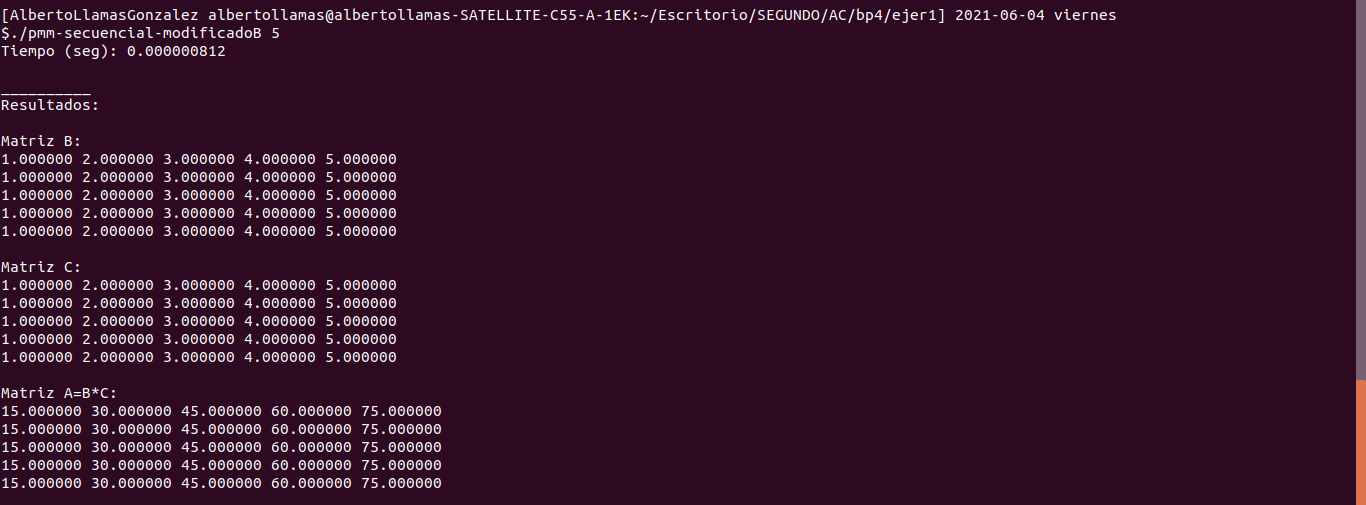
****

**B) Captura de** pmm-secuencial-modificado\_B.c

|  |
| --- |
| fos/ejer1/modB/Captura%20de%20pantalla%20de%202021-06-04%2012-44-43.png |

**Capturas de pantalla (que muestren la compilación y que el resultado es correcto):**

****

****

**TIEMPOS:**

|  |  |  |
| --- | --- | --- |
| **Modificación** | **Breve descripción de las modificaciones** | **-O2** |
| Sin modificar | *Dos tiempos para cada modificación respectivamente* | *0,000031154-0,002367576* |
| Modificación A) | Desenrollado de bucle | 0,000005345 |
| Modificación B) | Cambio de iteradores en el for | 0,000404259 |
|  |  |  |
|  |  |  |

**COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:** Es claro que las modificaciones mejoran el tiempo. De entre ellas, la que más mejora el tiempo es la modificación 2, lo cual tiene sentido por la penalización que se aprecia en caché.

1. **(a)** Usando como base el código de BP0, generar un programa para evaluar un código de la Figura 1. M y N deben ser parámetros de entrada al programa. Los datos se deben generar de forma aleatoria para valores de M y N mayores que 8, como en el ejemplo de BP0.

**CÓDIGO FIGURA 1:**

**CAPTURA CÓDIGO FUENTE**: figura1-original.c

|  |
| --- |
| fos/ejer2/Captura%20de%20pantalla%20de%202021-06-04%2013-42-17.pngfos/ejer2/Captura%20de%20pantalla%20de%202021-06-04%2013-42-24.pngfos/ejer2/Captura%20de%20pantalla%20de%202021-06-04%2013-42-31.png |

**Figura 1** .     Código C++ que suma dos vectores. M y N deben ser parámetros de entrada al programa, usar valores mayores que 1000 en la evaluación.

|  |
| --- |
| struct {          int a;          int b;  }  s[N];    main()  {     …     for (ii=0; ii<M;ii++) {        X1=0; X2=0;        for(i=0; i<N;i++)  X1+=2\*s[i].a+ii;        for(i=0; i<N;i++)  X2+=3\*s[i].b-ii;          if (X1<X2)  R[ii]=X1  else  R[ii]=X2;     }     …  } |

**(b)** Modificar el código C (solo el trozo a evaluar) para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre -O2) a partir de la modificación realizada. En las ejecuciones de evaluación usar valores de N y M mayores que 1000. Incorporar los códigos modificados en el cuaderno.

**MODIFICACIONES REALIZADAS (al menos dos modificaciones):**

**Modificación A) –explicación-:** recorremos los vectores a y b en el mismo bucle. Por cómo están colocados en el struct, tenemos en memoria {a0, b0, a1, b1, ..., an, bn}. Con el código original estamos trayendo de caché todos los bloques del struct dos veces, para recorrer los vectores a y b, respectivamente. Sin embargo, con esta modificación sólo traemos el struct completo de caché una sola vez, porque vamos recorriendo la memoria de forma contigua, sin saltos.

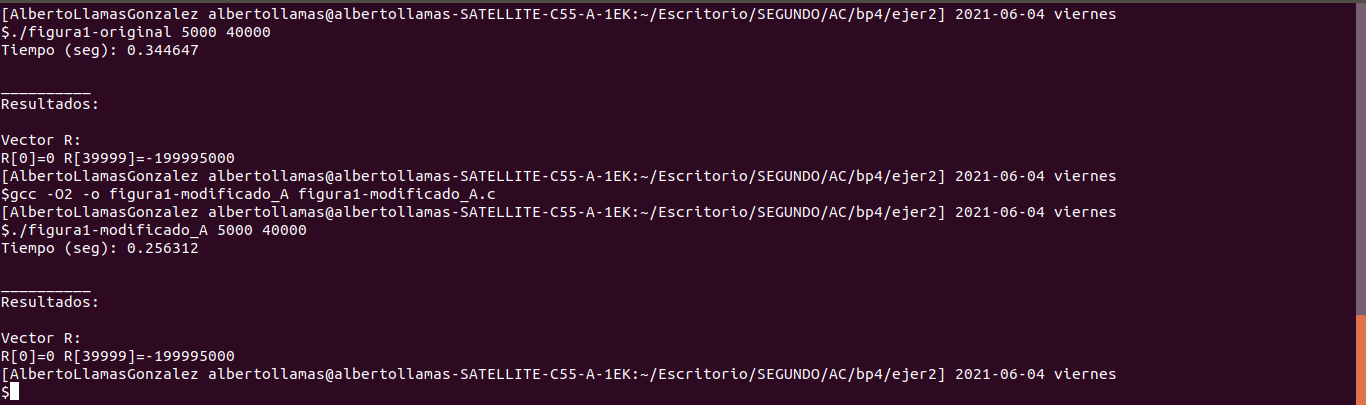
**Modificación B) –explicación-:** añadimos a la modificación A un desenrollado del for en 4 como en la modificación A del ejercicio 1

**CÓDIGOS FUENTE MODIFICACIONES**

**A) Captura** figura1-modificado\_A.c

|  |
| --- |
| fos/ejer2/Captura%20de%20pantalla%20de%202021-06-04%2013-42-06.png |

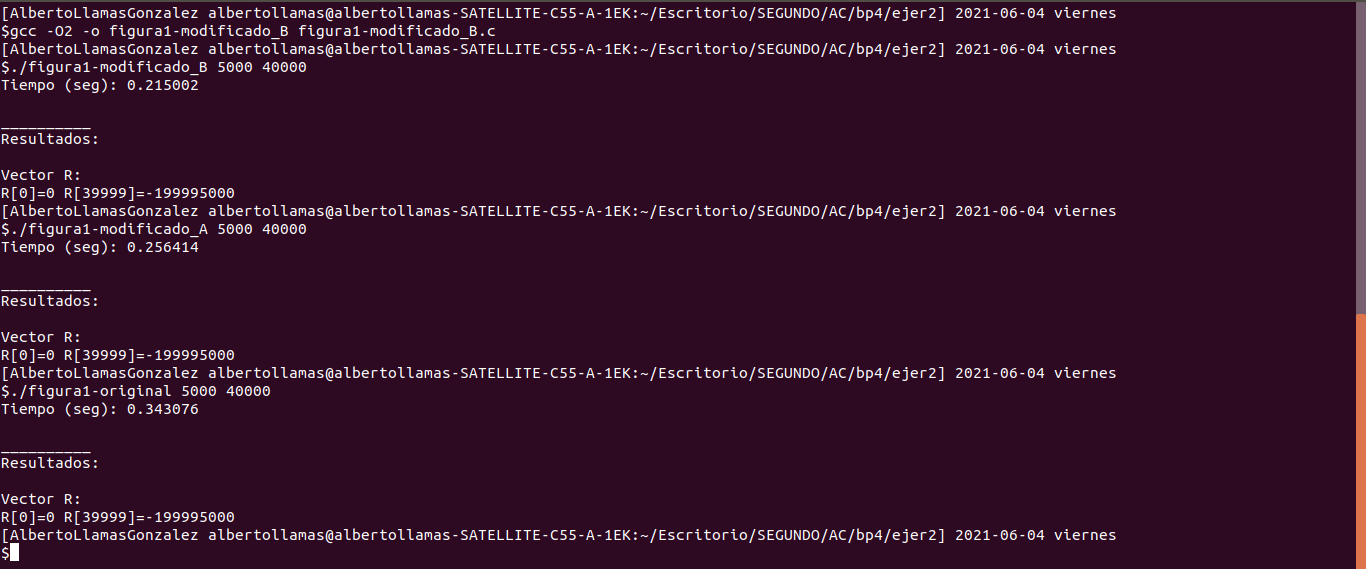
**Capturas de pantalla (que muestren la compilación y que el resultado es correcto):**

****

**B) Captura** figura1-modificado\_B.c

|  |
| --- |
| fos/ejer2/Captura%20de%20pantalla%20de%202021-06-04%2013-58-25.pngfos/ejer2/Captura%20de%20pantalla%20de%202021-06-04%2013-58-32.png |

**Capturas de pantalla (que muestren la compilación y que el resultado es correcto):**

****

**TIEMPOS:**

|  |  |  |
| --- | --- | --- |
| **Modificación** | **Breve descripción de las modificaciones** | **-O2** |
| Sin modificar |  | *0,343076* |
| Modificación A) | a y b en mismo bucle | 0,256414 |
| Modificación B) | desenrollado del bucle | 0,215002 |
|  |  |  |
|  |  |  |

**COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:**

Como vemos, el desenrollado de bucles con la unión en un mismo for es la mejor opción.

1. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina que opera con flotantes de doble precisión denominada DAXPY (***D****ouble precision- real* ***A****lpha* ***X******P****lus* ***Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

for (i=0;i<N;i++) y[i]= a\*x[i] + y[i];

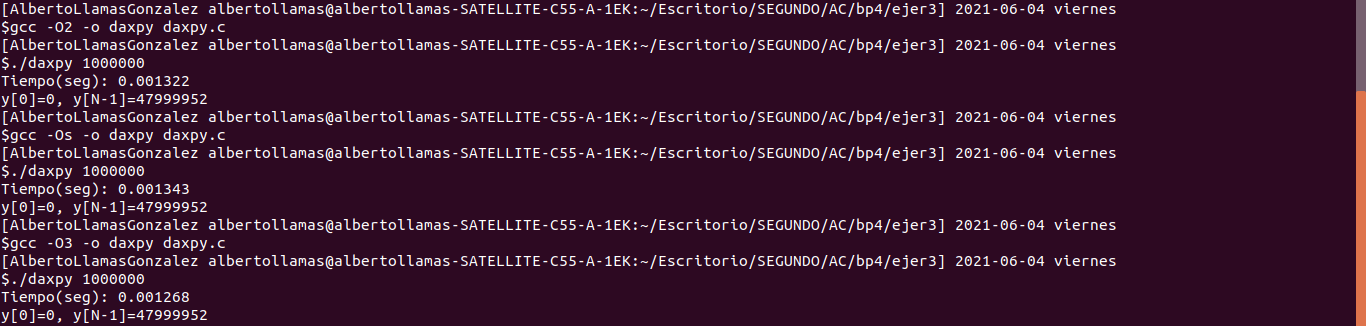
Generar los programas en ensamblador para cada una de las siguientes opciones de optimización del compilador: -O0, -Os, -O2, -O3. Explique las diferencias que se observan en el código justificando al mismo tiempo las mejoras en velocidad que acarrean. Incorporar los códigos al cuaderno de prácticas y destacar las diferencias entre ellos. Sólo se debe evaluar el tiempo del núcleo DAXPY. N deben ser parámetro de entrada al programa.

**CAPTURA CÓDIGO FUENTE**: daxpy.c

|  |
| --- |
| fos/ejer3/Captura%20de%20pantalla%20de%202021-06-04%2014-04-15.pngfos/ejer3/Captura%20de%20pantalla%20de%202021-06-04%2014-04-21.png |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tiempos ejec.**  **Longitud vectores=1000000** | **-O0** | **-Os** | **-O2** | **-O3** |
| *De 0,25 a 10 seg. aquí* | 0,001343 | 0,001322 | 0,001268 |

**CAPTURAS DE PANTALLA** (que muestren la compilación y que el resultado es correcto)**:**

****

**COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:**

**-O0:** no tenemos optimización alguna.

**-O2:** vemos que el número de instrucciones es parecido, pero observamos cambios considerables en las instrucciones utilizadas, que serán más eficientes.

**-O3**: el código ensamblador es mucho más complejo y difícil de entender, aumenta también el número de subrutinas llamadas.

**-Os:** disminuimos el tamaño del ejecutable, el código se parece mucho al de -O1 y -O2, pero podemos ver en el tamaño de archivo que este es menor.

**CÓDIGO EN ENSAMBLADOR** (no es necesario introducir aquí el código como captura de pantalla, ajustar el tamaño de la letra para que una instrucción no ocupe más de un renglón)**:**

**(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)**

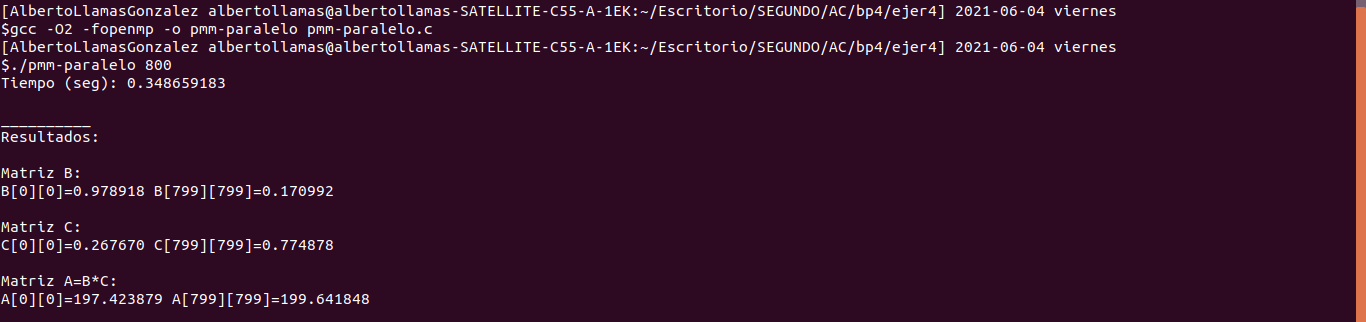
*Nota: He cambiado el formato de las tablas para que se vean bien las imágenes.*

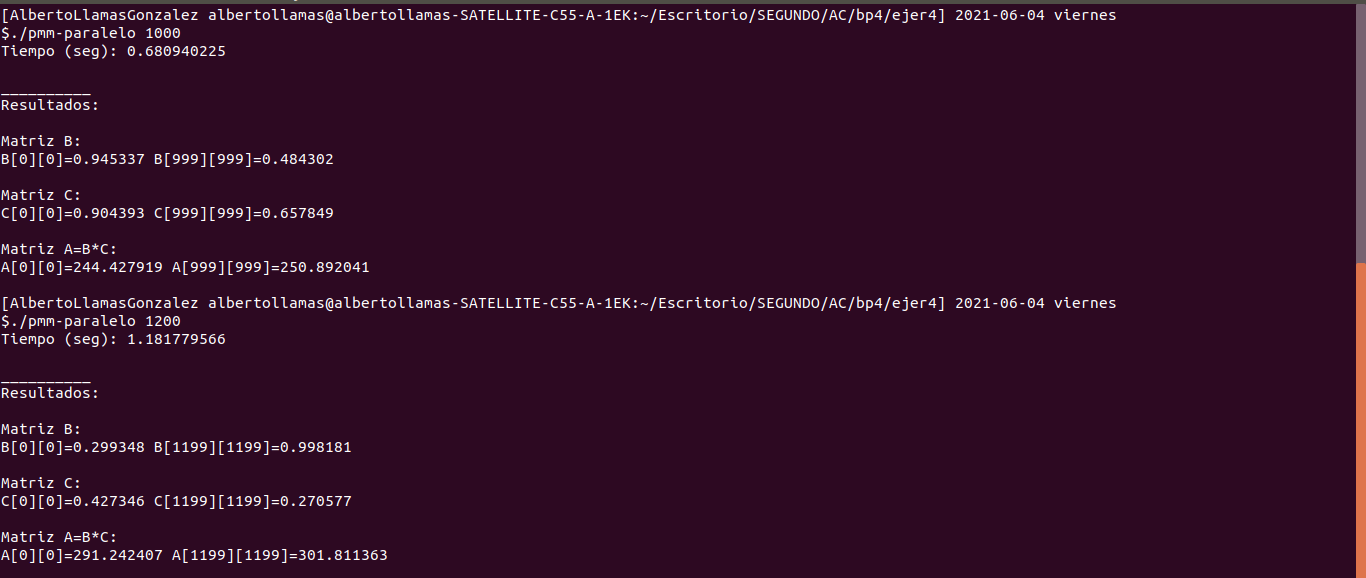
|  |
| --- |
| daxpyO0.s |
| ../../../../../../../../var/folders/pd/r_2y4gh96vx5xvwqxgkchhx40000gn/T/TemporaryItems/NSIRD_screencaptureui_Uqn0Cw/Captura%20de%20pantalla%20 |
| daxpyOs.s |
| ../../../../../../../../var/folders/pd/r_2y4gh96vx5xvwqxgkchhx40000gn/T/TemporaryItems/NSIRD_screencaptureui_W8PQDs/Captura%20de%20pantalla%20 |
| daxpyO2.s |
| **../../../../../../../../var/folders/pd/r_2y4gh96vx5xvwqxgkchhx40000gn/T/TemporaryItems/NSIRD_screencaptureui_zinXYd/Captura%20de%20pantalla%20** |
| daxpyO3.s |
| **../../../../../../../../var/folders/pd/r_2y4gh96vx5xvwqxgkchhx40000gn/T/TemporaryItems/NSIRD_screencaptureui_TfTFLu/Captura%20de%20pantalla%20**  **../../../../../../../../var/folders/pd/r_2y4gh96vx5xvwqxgkchhx40000gn/T/TemporaryItems/NSIRD_screencaptureui_nclgCm/Captura%20de%20pantalla%20** |

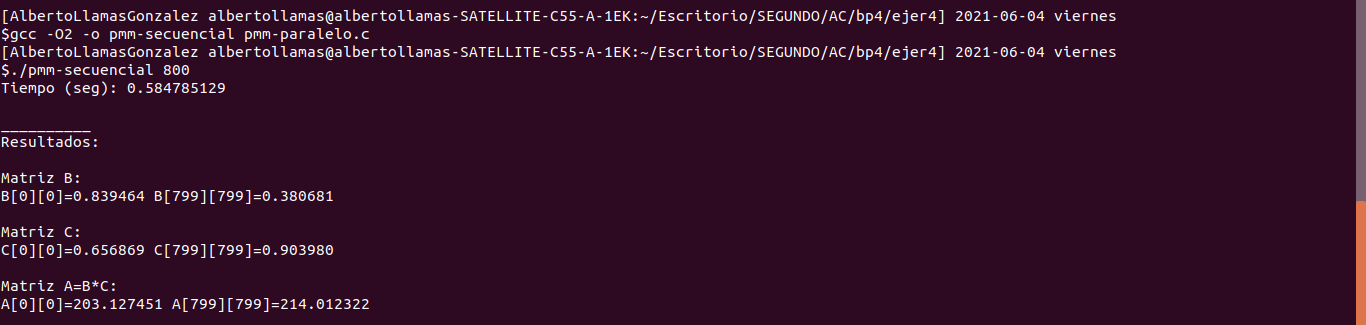
1. **(a)** Paralizar con OpenMP en la CPU el código de la multiplicación resultante en el Ejercicio 1.(b). NOTA: usar para generar los valores aleatorios, por ejemplo, drand48\_r().

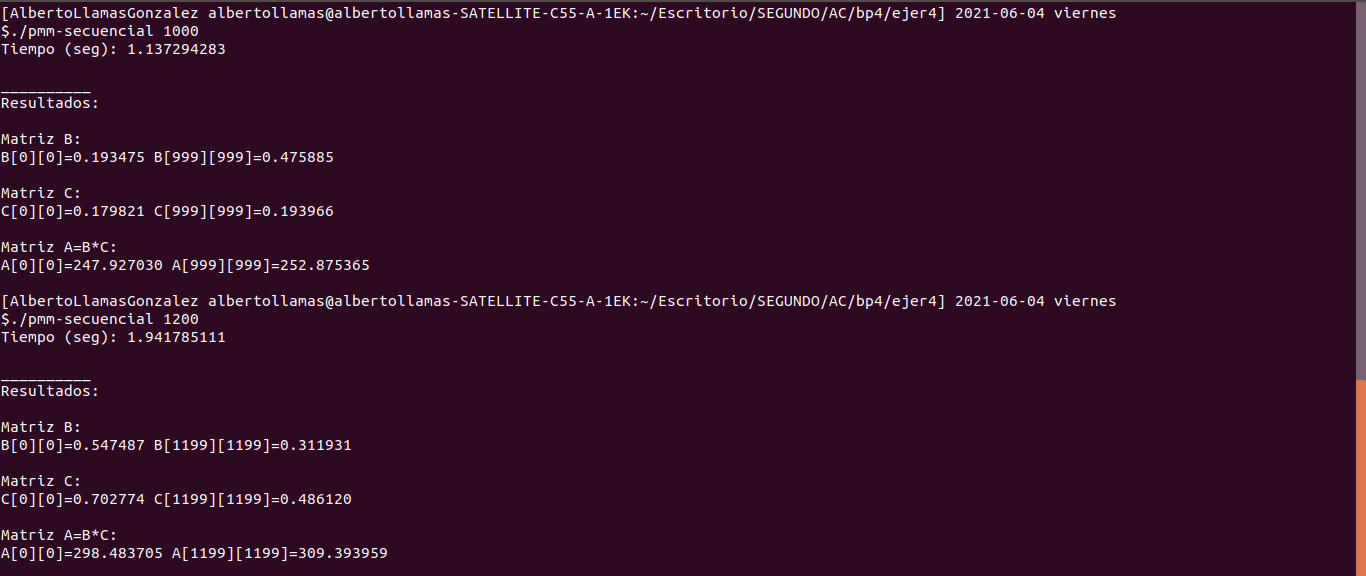
**(b)** Calcular la ganancia en prestaciones que se obtiene en atcgrid4 para el máximo número de procesadores físicos con respecto al código inicial no optimizado del Ejercicio 1.(a) para dos tamaños de la matriz.

**(a) MULTIPLICACIÓN DE MATRICES PARALELO:**

****

****

****

****

**CAPTURA CÓDIGO FUENTE**: pmm-paralelo.c

|  |
| --- |
| aa/Captura%20de%20pantalla%20de%202021-06-04%2023-13-32.png |

La única modificación realizada en el código es la sentencia añadida #pragma omp parallel for, que se ve en la imagen.

**(b) RESPUESTA**

Como podemos observar en la captura de la ejecución, podemos calcular tres ganancias, ya que hemos probado tres tamaños de matriz.

G1 = 0,584785129 / 0,348659183 = 1,6772400026

G2 = 1,137294283 / 0,680940225 = 1,6701822587

G3 = 1,94178511 / 1,181776566 = 1,6431067986

Como vemos, la ganancia oscila entre 1,64 y 1,67.