Escuela de Ciencia de la Computación
Universidad Nacional de San Agustín Arequipa

Milagros Celia Cruz Mamani

COMPUTACIÓN PARALELA Y DISTRIBUIDA

Laboratorio 1

PRUEBAS CON LA MEMORIA CACHÉ

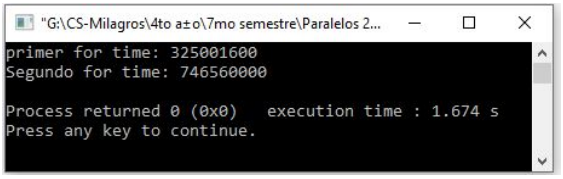
https://github.com/milagroscm/COMPUATACIO-PARALELA

Resumen

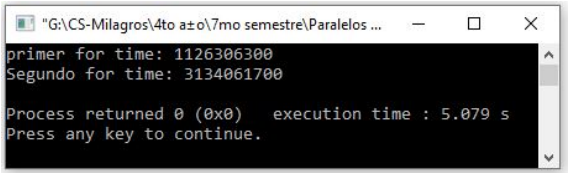
El alumno debe realizar el informe en formato articulo donde la implementación, resultados y análisis de la ejecución para los siguientes problemas.

1. Ejercicios

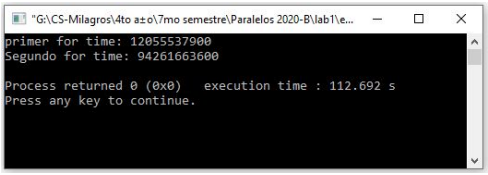
1.1. Implementar y comparar los 2-bucles anidados FOR presentados en el Capitulo 2 del libro pag.22



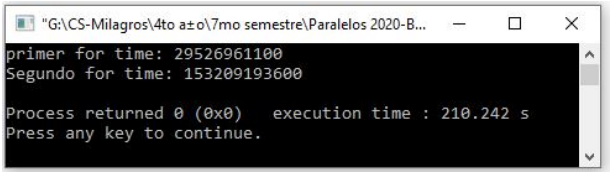
(a) Matriz con 5000 datos



(b) Matriz con 10000 datos



(c) Matriz con 30000 datos



(d) Matriz con 40000 datos

1.1.1. Explicación:

Para la comparación de los 2-bucles anidados se realizo las pruebas bajo las siguientes premisas:

- El código fue implementado usando la librería `chrono` para calcular el tiempo.
- Se trabajo con matrices de 5000, 10000, 30000 y 40000 datos.
- El primer bucle produce menos cache misses debido que C++ es un lenguaje “row-major", quiere decir que en el caso de las matrices, la segunda fila está ubicada después de la primera fila, la tercera fila después de la segunda fila y así sucesivamente como si se tratara de un arreglo lineal más grande. Debido a esto el primer bucle aprovecha mejor la localidad espacial.

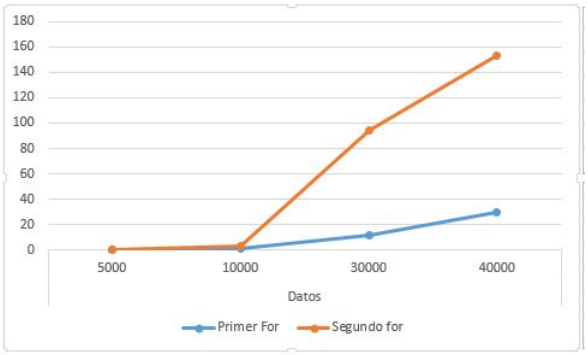
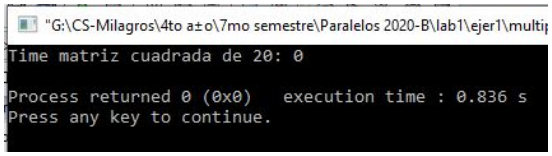
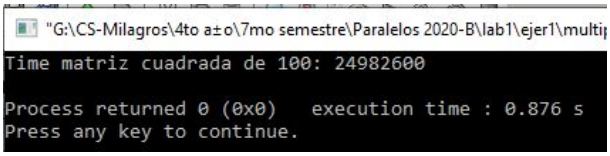


Figura 1: Comparación de los 2-bucles anidados

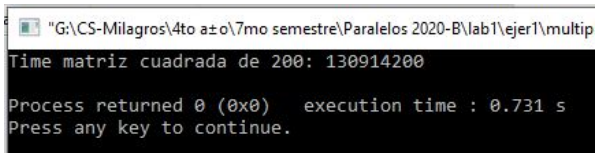
1.2. Implementar en C/C++ la multiplicación de matrices clásica, la versión de tres bucles y evaluar su desempeño considerando diferentes tamaños de matriz



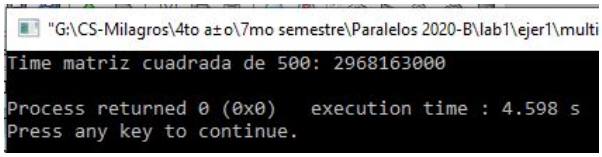
(a) Matriz cuadrada de 20 x 20



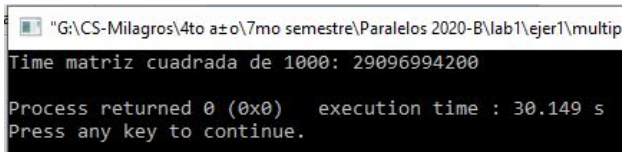
(b) Matriz cuadrada de 100 x 100



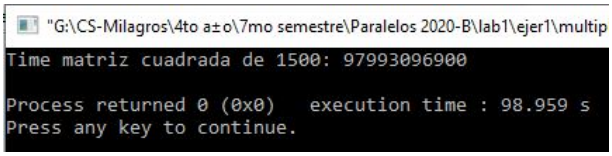
(c) Matriz cuadrada de 200 x 200



(d) Matriz cuadrada de 500 x 500



(e) Matriz cuadrada de 1000 x 1000



(f) Matriz cuadrada de 1500 x 1500

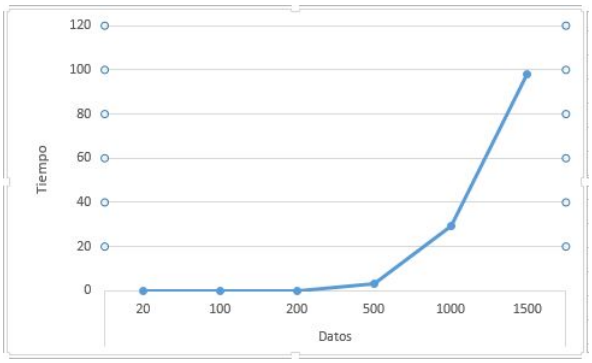
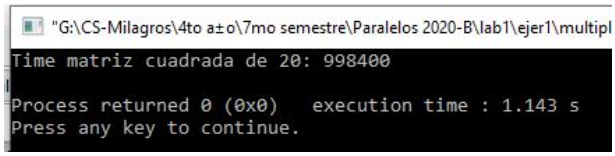
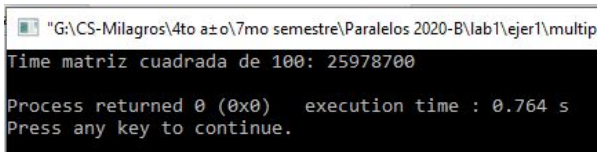


Figura 2: Desempeño de la multiplicación de matrices clásicas

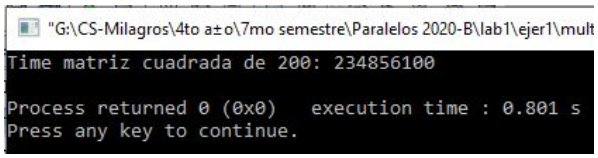
1.3. Implementar la versión por bloques , seis bloques anidados , evaluar su desempeño y compararlo con la multiplicación de matrices clásica



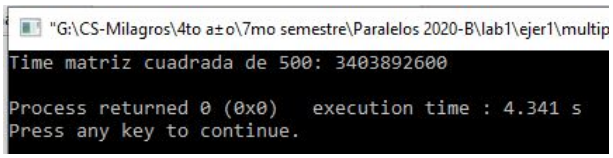
(a) Matriz cuadrada de 20 x 20



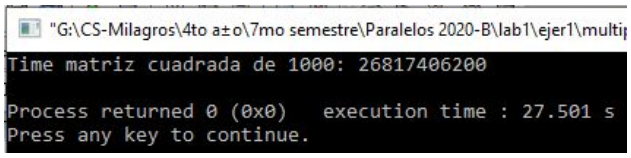
(b) Matriz cuadrada de 100 x 100



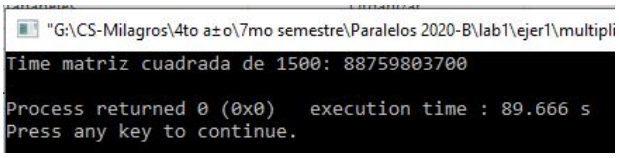
(c) Matriz cuadrada de 200 x 200



(d) Matriz cuadrada de 500 x 500



(e) Matriz cuadrada de 1000 x 1000



(f) Matriz cuadrada de 1500 x 1500

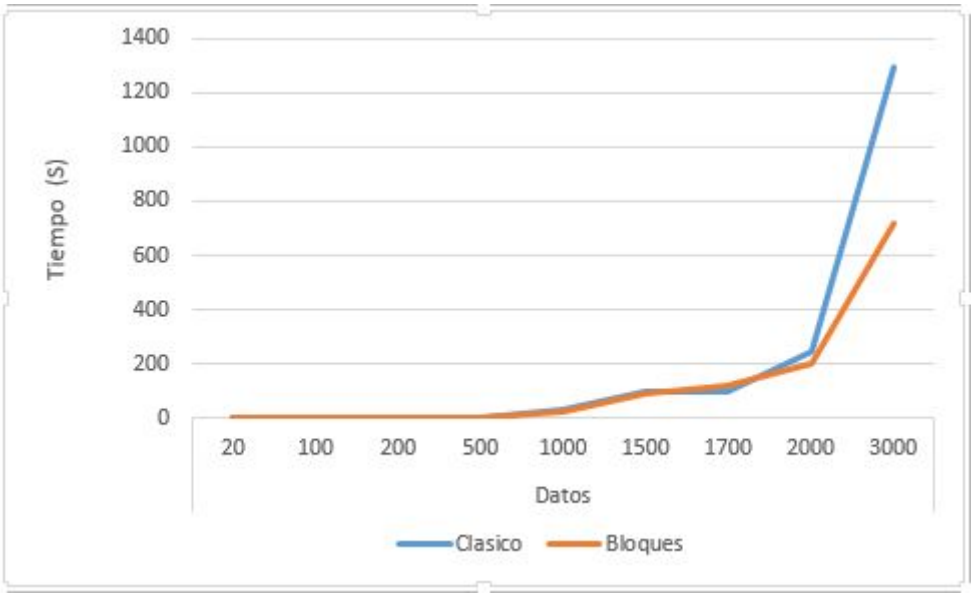


Figura 3: Comparación entre los dos tipos de multiplicación

1.3.1. Explicación:

- Primero veremos la información de muestro procesador, la arquitectura, frecuencia, los cores, threads por core mediante el comando `lscpu` y estos son los datos que muestra, según la figura **??**.

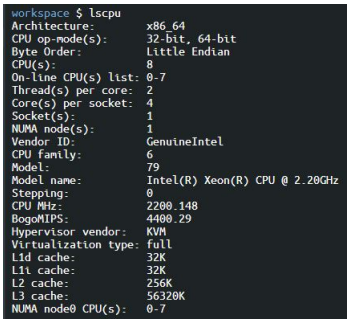


Figura 4: Comparación entre los dos tipos de multiplicación

- En la figura 3 se muestra un gráfico con el tiempo de demora de los algoritmos respecto al tamaño de las matrices , donde claramente se puede observar que el algoritmo de multiplicación con bloques es mas rápido que el algoritmo de multiplicación de 3 for cuando el tamaño aumenta. Ya que l multiplicación por bloques trata se centra en la partición imaginaria de la matriz en pequeños bloques, donde se busca que estos bloques puedan ser almacenados temporalmente en memoria caché y así evitar el desplazamiento y perder la información y así evitar los cache misses.
- En este experimento; el tamaño de la memoria de cache es de 32K , los enteros ocupan 4 bytes, y se usaron bloques de tamaño 64 que serian 2^6.

1.4. Ejecutar ambos algoritmos utilizando las herramientas valgrind y kcachegrind para obtener una evaluación más precisa de su desempeño en términos de cache misses

- Para hacer uso de las herramientas valgrind y kcachegrind se hizo uso del SO linux mediante un entorno virtual en la nube sandbox de codificación de `next.tech`.

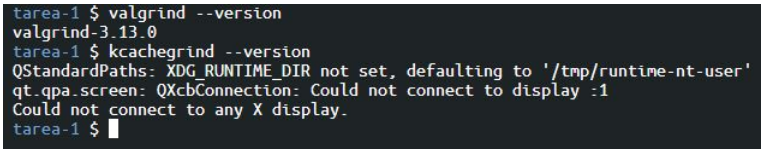
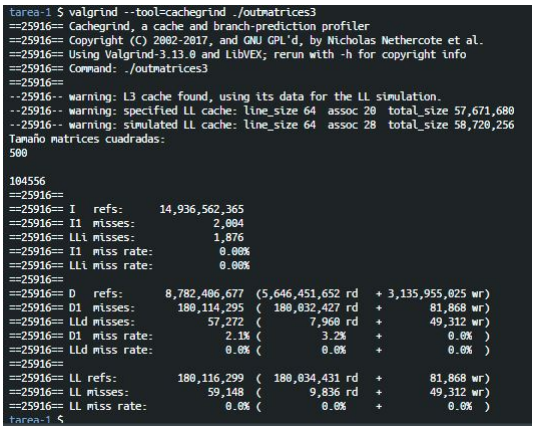


Figura 5: Verificación de la nstalación de valgrind y kachegrind

- Para la prueba de las dos versiones se realizo sobre matrices cuadradas de 500×500 .
- **Multiplicación de matrices clásica:** El algoritmo que tiene 3 bucles anidados para realizar la multiplicación.



- **Multiplicación por bloques:** Algoritmo que tiene 6 bucles anidados para calcular la multiplicación de matrices .

```
tarea-1 $
tarea-1 $ valgrind --tool=cachegrind ./out6matrices
==25946== Cachegrind, a cache and branch-prediction profiler
==25946== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==25946== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==25946== Command: ./out6matrices
==25946==
--25946-- warning: L3 cache found, using its data for the LL simulation.
--25946-- warning: specified LL cache: line_size 64  assoc 20  total_size 57,671,680
--25946-- warning: simulated LL cache: line_size 64  assoc 28  total_size 58,720,256
Tamaño matrices cuadradas:
500
Tamaño del bloque:
64
158.462
==25946==
==25946== I   refs:      22,453,419,192
==25946== I1  misses:             2,264
==25946== LLi misses:             2,044
==25946== I1  miss rate:           0.00%
==25946== LLi miss rate:           0.00%
==25946==
==25946== D   refs:      13,183,048,822  (8,367,597,270 rd   + 4,815,451,552 wr)
==25946== D1  misses:       597,581  (      515,688 rd   +        81,893 wr)
==25946== LLd misses:        57,300  (        7,988 rd   +        49,312 wr)
==25946== D1  miss rate:        0.0% (          0.0%     +           0.0%  )
==25946== LLd miss rate:        0.0% (          0.0%     +           0.0%  )
==25946==
==25946== LL refs:           599,845  (      517,952 rd   +        81,893 wr)
==25946== LL misses:          59,344  (       10,032 rd   +        49,312 wr)
==25946== LL miss rate:          0.0% (          0.0%     +           0.0%  )
tarea-1 $
```

- **Eventos registrados** las abreviaturas de los eventos son:
  - **I:** Instrucciones ejecutadas
  - **I1:** Errores de lectura de caché
  - **D:** Lecturas de caché
  - **D1:** Errores de lectura de caché
  - **LL:** Representa a los niveles de los cache
  - **LLi:** Total de accesos a instrucciones en los otros niveles de caché
  - **LLd:** Total de accesos a datos en los otros niveles de caché.

  Kcachegrind: Con esta herramienta analizaremos los caché misses encontrados, y como se puede observar en las imágenes, en la multiplicación clásica se produce mas caché misses

  - ***Multiplicación de matrices clásica:***

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	14 877 790 149	5 502 254 570	Ir	
Data Read Access	5 501 161 685	1 751 002 016	Dr	
Data Write Access	3 250 645 533	1 000 501 023	Dw	
L1 Instr. Fetch Miss	9	8	I1mr	
L1 Data Read Miss	180 015 478	133 015 502	D1mr	
L1 Data Write Miss	15 997	1	D1mw	
ILmr	9	8	ILmr	
DLmr	0	0	DLmr	
DLmw	15 937	0	DLmw	
L1 Miss Sum	180 031 484	133 015 511	L1m	= I1mr + D1mr + D1mw

  - ***Multiplicación por bloques:***

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	22 394 637 448	7 843 395 989	Ir	
Data Read Access	8 220 522 491	2 147 356 984	Dr	
Data Write Access	4 931 922 873	1 649 356 545	Dw	
L1 Instr. Fetch Miss	15	13	I1mr	
L1 Data Read Miss	498 596	462 102	D1mr	
L1 Data Write Miss	15 997	1	D1mw	
ILmr	15	13	ILmr	
DLmr	0	0	DLmr	
DLmw	15 937	0	DLmw	
L1 Miss Sum	514 608	462 116	L1m	= I1mr + D1mr + D1mw

1.5. Conclusiones:

- La memoria caché juega un papel importante en la optimización de los programas.
- El tiempo de ejecución de un programa no solo depende de la optimización del algoritmo sino también de la arquitectura del computador.
- La multiplicación de matrices por bloques, aprovecha la localidad espacial
- Con el uso de las herramientas se puede observar la cantidad de caché misses producidos.