



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 CACHE
<https://github.com/milagroscm/COMPUATACIO-PARALELA>

Resumen

El alumno debe realizar el informe en formato artículo 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 Capítulo 2 del libro pag.22

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\...  
primer for time: 325001600  
Segundo for time: 746560000  
  
Process returned 0 (0x0) execution time : 1.674 s  
Press any key to continue.
```

(a) Matriz con 5000 datos

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos ...  
primer for time: 1126306300  
Segundo for time: 3134061700  
  
Process returned 0 (0x0) execution time : 5.079 s  
Press any key to continue.
```

(b) Matriz con 10000 datos

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\...  
primer for time: 1205537900  
Segundo for time: 94261663600  
  
Process returned 0 (0x0) execution time : 112.692 s  
Press any key to continue.
```

(c) Matriz con 30000 datos

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B...  
primer for time: 29526961100  
Segundo for time: 153209193600  
  
Process returned 0 (0x0) execution time : 210.242 s  
Press any key to continue.
```

(d) Matriz con 40000 datos

1.1.1. Explicación:

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

- El código fue implementado usando la librería **chrono** para calcular el tiempo.
- Se trabajó 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.

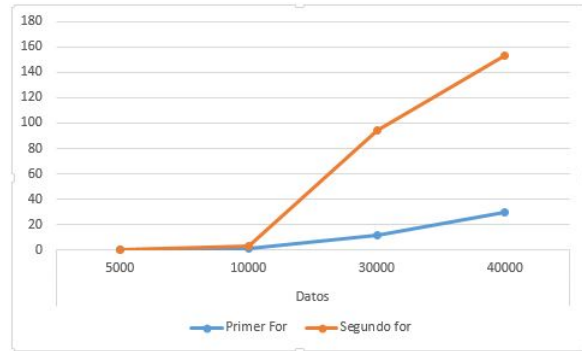


Figura 1: Comparación de los 2-bucles anidados

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

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multi
Time matriz cuadrada de 20: 0
Process returned 0 (0x0)  execution time : 0.836 s
Press any key to continue.
```

(a) Matriz cuadrada de 20 x 20

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multi
Time matriz cuadrada de 100: 24982600
Process returned 0 (0x0)  execution time : 0.876 s
Press any key to continue.
```

(b) Matriz cuadrada de 100 x 100

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multi
Time matriz cuadrada de 200: 130914200
Process returned 0 (0x0)  execution time : 0.731 s
Press any key to continue.
```

(c) Matriz cuadrada de 200 x 200

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multi
Time matriz cuadrada de 500: 2968163000
Process returned 0 (0x0)  execution time : 4.598 s
Press any key to continue.
```

(d) Matriz cuadrada de 500 x 500

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multi
Time matriz cuadrada de 1000: 29096994200
Process returned 0 (0x0)  execution time : 30.149 s
Press any key to continue.
```

(e) Matriz cuadrada de 1000 x 1000

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multi
Time matriz cuadrada de 1500: 97993096900
Process returned 0 (0x0)  execution time : 98.959 s
Press any key to continue.
```

(f) Matriz cuadrada de 1500 x 1500

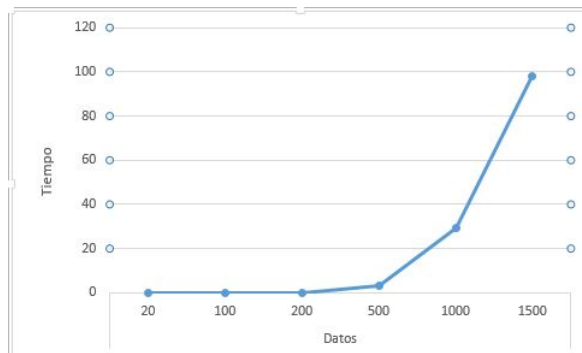


Figura 2: Desempeño de la multiplicación de matrices clásicas

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

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multipl
Time matriz cuadrada de 20: 998400
Process returned 0 (0x0)   execution time : 1.143 s
Press any key to continue.
```

(a) Matriz cuadrada de 20 x 20

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multipl
Time matriz cuadrada de 100: 25978700
Process returned 0 (0x0)   execution time : 0.764 s
Press any key to continue.
```

(b) Matriz cuadrada de 100 x 100

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multipl
Time matriz cuadrada de 200: 234856100
Process returned 0 (0x0)   execution time : 0.801 s
Press any key to continue.
```

(c) Matriz cuadrada de 200 x 200

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multipl
Time matriz cuadrada de 500: 3403892600
Process returned 0 (0x0)   execution time : 4.341 s
Press any key to continue.
```

(d) Matriz cuadrada de 500 x 500

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multipl
Time matriz cuadrada de 1000: 26817406200
Process returned 0 (0x0)   execution time : 27.501 s
Press any key to continue.
```

(e) Matriz cuadrada de 1000 x 1000

```
"G:\CS-Milagros\4to a±o\7mo semestre\Paralelos 2020-B\lab1\ejer1\multipl
Time matriz cuadrada de 1500: 88759803700
Process returned 0 (0x0)   execution time : 89.666 s
Press any key to continue.
```

(f) Matriz cuadrada de 1500 x 1500

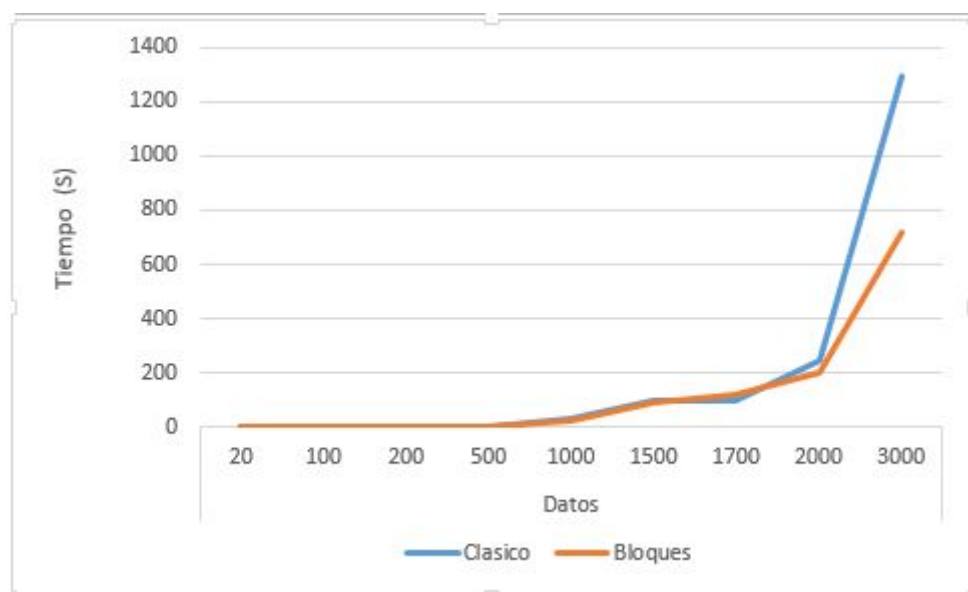


Figura 3: Comparación entre los dos tipos de multiplicación

1.3.1. Explicación:

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

```
workspace $ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                 8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):              1
NUMA node(s):          1
Vendor ID:              GenuineIntel
CPU family:             6
Model:                  79
Model name:             Intel(R) Xeon(R) CPU @ 2.20GHz
Stepping:               0
CPU MHz:                2200.148
BogoMIPS:               4400.29
Hypervisor vendor:     KVM
Virtualization type:    full
L1d cache:              32K
L1i cache:              32K
L2 cache:               256K
L3 cache:               56320K
NUMA node0 CPU(s):     0-7
```

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 más rápido que el algoritmo de multiplicación de 3 for cuando el tamaño aumenta. Ya que la multiplicación por bloques trata de centrarse 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 caché es de 32K, los enteros ocupan 4 bytes, y se usaron bloques de tamaño 64 que serían 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`.

```
tarea-1 $ valgrind --version
valgrind-3.13.0
tarea-1 $ kcachegrind --version
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-nt-user'
qt.qpa.screen: QXcbConnection: Could not connect to display :1
Could not connect to any X display.
tarea-1 $
```

Figura 5: Verificación de la instalación de valgrind y kcachegrind

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

```
tarea-1 $ valgrind --tool=kcachegrind ./outmatrices3
==25916== kcachegrind, a cache and branch-prediction profiler
==25916== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==25916== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==25916== Command: ./outmatrices3
==25916==
--25916-- warning: L3 cache found, using its data for the LL simulation.
--25916-- warning: specified LL cache: line_size 64 assoc 20 total_size 57,671,680
--25916-- warning: simulated LL cache: line_size 64 assoc 28 total_size 58,720,256
Tamaño matrices cuadradas:
500
104556
==25916==
==25916== I refs:      14,936,562,365
==25916== I1 misses:    2,004
==25916== I1L1 misses: 1,876
==25916== I1L1 miss rate: 0.00%
==25916== I1L1 miss rate: 0.00%
==25916==
==25916== D refs:      8,782,406,677 (5,646,451,652 rd + 3,135,955,025 wr)
==25916== D1 misses: 180,114,295 ( 180,032,427 rd +      81,868 wr)
==25916== L1d misses:  57,272 (    7,960 rd +    49,312 wr)
==25916== D1L1 miss rate: 2.1% (    3.2% +    0.0% )
==25916== L1dL1 miss rate: 0.0% (    0.0% +    0.0% )
==25916==
==25916== LL refs:      180,116,299 ( 180,034,431 rd +      81,868 wr)
==25916== LL misses:   59,148 (    9,836 rd +    49,312 wr)
==25916== LL miss rate: 0.0% (    0.0% +    0.0% )
tarea-1 $
```

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

```
tarea-1 $ valgrind --tool=cachegrind ./outmatrices
==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: ./outmatrices
==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== I refs:      22,453,419,192
==25946== I1 misses:    2,264
==25946== L1L misses:    2,044
==25946== I1 miss rate:  0.00%
==25946== L1L 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== L1D misses:    57,380 ( 7,988 rd + 49,312 wr)
==25946== D1 miss rate:  0.0% ( 0.0% + 0.0% )
==25946== L1D 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 ( 18,832 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.