

Práctica 01

Medidas de Rendimiento

Fermin Sola y Eduardo Ezponda

15 de febrero de 2023

1. Primer Apartado

c) A continuación se muestran las pantallas de CPU y Caches

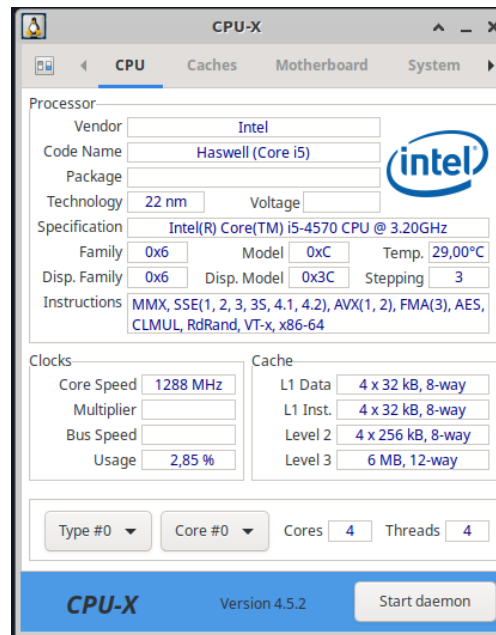


Figura 1: CPU

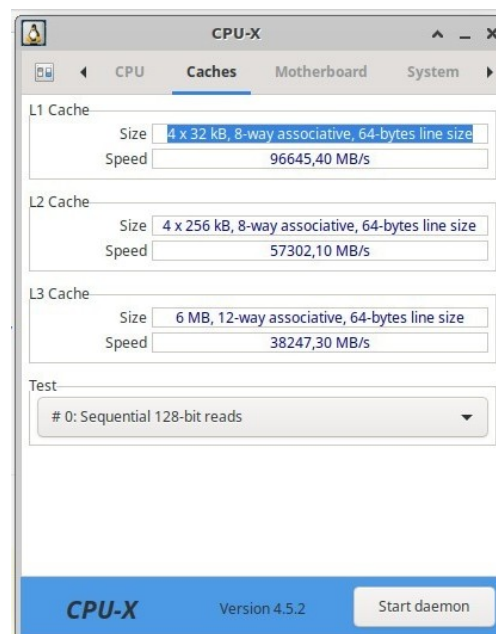


Figura 2: CACHE

2. Segundo Apartado

a) Programa SumaMatriz.c

```
1  /*
2   * Program: SumaMatriz
3   * Name: Fermin Sola en colaboracion con Eduardo Ezponda
4   * Date: 13/02/2023
5   */
6
7  #include <stdlib.h>
8  #include <stdio.h>
9  #include <time.h>
10 #include <string.h>
11
12 #define N 500
13 #define M 25000
14
15 double a[N][N], b[N][N], c[N][N];
16
17 int main(){
18     for(int i = 0; i < N; i++){
19         for(int j = 0; j < N; j++){
20             a[i][j] = drand48();
21             b[i][j] = drand48();
22         }
23     }
24
25     clock_t start = clock();
26     for(int k = 0; k < M; k++){
27         for(int i = 0; i < N; i++){
28             for(int j = 0; j < N; j++){
29                 c[i][j] = a[i][j] + b[i][j];
30             }
31         }
32     }
33     clock_t finish = clock();
34     printf("El tiempo tardado es: %ld\n", (finish-start)/CLOCKS_PER_SEC);
35
36     return 0;
37 }
```

Programa SumaMatriz.s

```
1      .text
2      .comm      a,2000000,32
3      .comm      b,2000000,32
4      .comm      c,2000000,32
5      .section   .rodata
6  .LC0:
7      .string     "El tiempo tardado es: %ld\n"
8      .text
9      .globl      main
10     .type       main, @function
11  main:
12  .LFB6:
13     .cfi_startproc
14     endbr64
15     pushq       %rbp
16     .cfi_def_cfa_offset 16
17     .cfi_offset 6, -16
18     movq        %rsp, %rbp
19     .cfi_def_cfa_register 6
20     subq        $48, %rsp
21     movl        $0, -36(%rbp)
22     jmp         .L2
23  .L5:
24     movl        $0, -32(%rbp)
25     jmp         .L3
26  .L4:
27     call        drand48@PLT
28     movq        %xmm0, %rax
29     movl        -32(%rbp), %edx
30     movslq      %edx, %rdx
31     movl        -36(%rbp), %ecx
32     movslq      %ecx, %rcx
33     imulq       $500, %rcx, %rcx
34     addq        %rcx, %rdx
35     leaq        0(,%rdx,8), %rcx
36     leaq        a(%rip), %rdx
37     movq        %rax, (%rcx,%rdx)
38     call        drand48@PLT
39     movq        %xmm0, %rax
40     movl        -32(%rbp), %edx
41     movslq      %edx, %rdx
42     movl        -36(%rbp), %ecx
```

```

43      movslq      %ecx, %rcx
44      imulq      $500, %rcx, %rcx
45      addq       %rcx, %rdx
46      leaq       0(,%rdx,8), %rcx
47      leaq       b(%rip), %rdx
48      movq       %rax, (%rcx,%rdx)
49      addl       $1, -32(%rbp)
50  .L3:
51      cmpl       $499, -32(%rbp)
52      jle        .L4
53      addl       $1, -36(%rbp)
54  .L2:
55      cmpl       $499, -36(%rbp)
56      jle        .L5
57      call       clock@PLT
58      movq       %rax, -16(%rbp)
59      movl       $0, -28(%rbp)
60      jmp        .L6
61  .L11:
62      movl       $0, -24(%rbp)
63      jmp        .L7
64  .L10:
65      movl       $0, -20(%rbp)
66      jmp        .L8
67  .L9:
68      movl       -20(%rbp), %eax
69      cltq
70      movl       -24(%rbp), %edx
71      movslq     %edx, %rdx
72      imulq     $500, %rdx, %rdx
73      addq      %rdx, %rax
74      leaq      0(,%rax,8), %rdx
75      leaq      a(%rip), %rax
76      movsd     (%rdx,%rax), %xmm1
77      movl      -20(%rbp), %eax
78      cltq
79      movl      -24(%rbp), %edx
80      movslq     %edx, %rdx
81      imulq     $500, %rdx, %rdx
82      addq      %rdx, %rax
83      leaq      0(,%rax,8), %rdx
84      leaq      b(%rip), %rax
85      movsd     (%rdx,%rax), %xmm0
86      addsd     %xmm1, %xmm0
87      movl      -20(%rbp), %eax

```

```

88      cltq
89      movl      -24(%rbp), %edx
90      movslq    %edx, %rdx
91      imulq     $500, %rdx, %rdx
92      addq      %rdx, %rax
93      leaq      0(,%rax,8), %rdx
94      leaq      c(%rip), %rax
95      movsd     %xmm0, (%rdx,%rax)
96      addl      $1, -20(%rbp)
97  .L8:
98      cmpl      $499, -20(%rbp)
99      jle       .L9
100     addl      $1, -24(%rbp)
101  .L7:
102     cmpl      $499, -24(%rbp)
103     jle       .L10
104     addl      $1, -28(%rbp)
105  .L6:
106     cmpl      $24999, -28(%rbp)
107     jle       .L11
108     call      clock@PLT
109     movq      %rax, -8(%rbp)
110     movq      -8(%rbp), %rax
111     subq      -16(%rbp), %rax
112     movq      %rax, %rcx
113     movabsq    $4835703278458516699, %rdx
114     movq      %rcx, %rax
115     imulq     %rdx
116     sarq      $18, %rdx
117     movq      %rcx, %rax
118     sarq      $63, %rax
119     subq      %rax, %rdx
120     movq      %rdx, %rax
121     movq      %rax, %rsi
122     leaq      .LC0(%rip), %rdi
123     movl      $0, %eax
124     call      printf@PLT
125     movl      $0, %eax
126     leave
127     .cfi_def_cfa 7, 8
128     ret
129     .cfi_endproc
130  .LFE6:
131     .size      main, .-main
132     .ident      "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0"

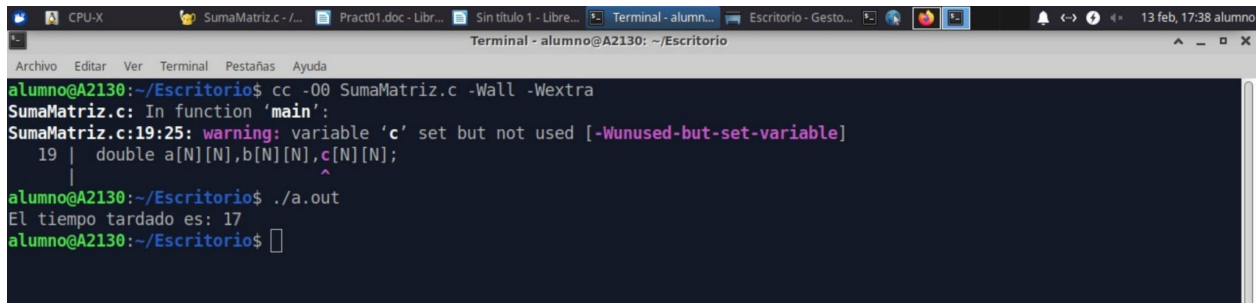
```

```

133     .section          .note.GNU-stack,"",@progbits
134     .section          .note.gnu.property,"a"
135     .align 8
136     .long             1f - 0f
137     .long             4f - 1f
138     .long             5
139 0:
140     .string           "GNU"
141 1:
142     .align 8
143     .long             0xc0000002
144     .long             3f - 2f
145 2:
146     .long             0x3
147 3:
148     .align 8
149 4:

```

b) Resultados de la función Clock() en nuestro SumaMatriz.c

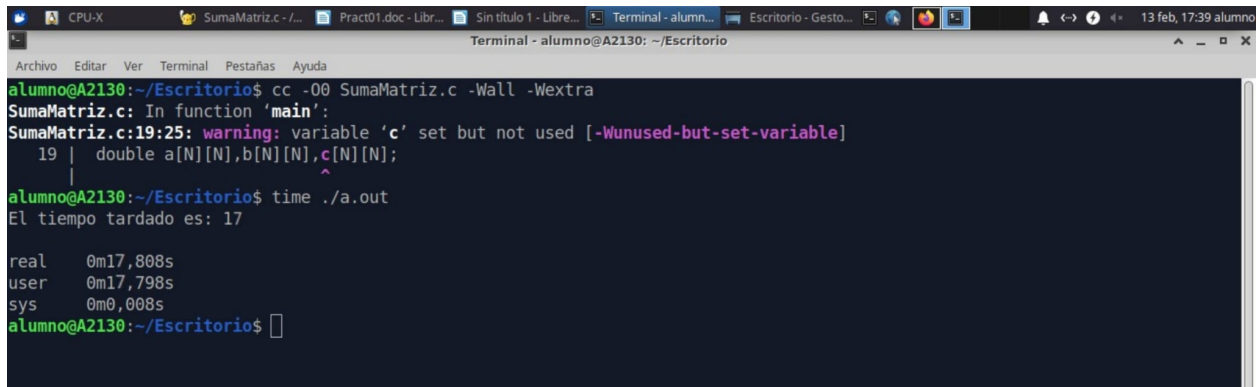


```

alumno@A2130:~/Escritorio$ cc -O0 SumaMatriz.c -Wall -Wextra
SumaMatriz.c: In function 'main':
SumaMatriz.c:19:25: warning: variable 'c' set but not used [-Wunused-but-set-variable]
   19 | double a[N][N],b[N][N],c[N][N];
      |                        ^
alumno@A2130:~/Escritorio$ ./a.out
El tiempo tardado es: 17
alumno@A2130:~/Escritorio$

```

Figura 3: Compilación y ejecución



```

alumno@A2130:~/Escritorio$ cc -O0 SumaMatriz.c -Wall -Wextra
SumaMatriz.c: In function 'main':
SumaMatriz.c:19:25: warning: variable 'c' set but not used [-Wunused-but-set-variable]
   19 | double a[N][N],b[N][N],c[N][N];
      |                        ^
alumno@A2130:~/Escritorio$ time ./a.out
El tiempo tardado es: 17

real    0m17.808s
user    0m17.798s
sys     0m0.008s
alumno@A2130:~/Escritorio$

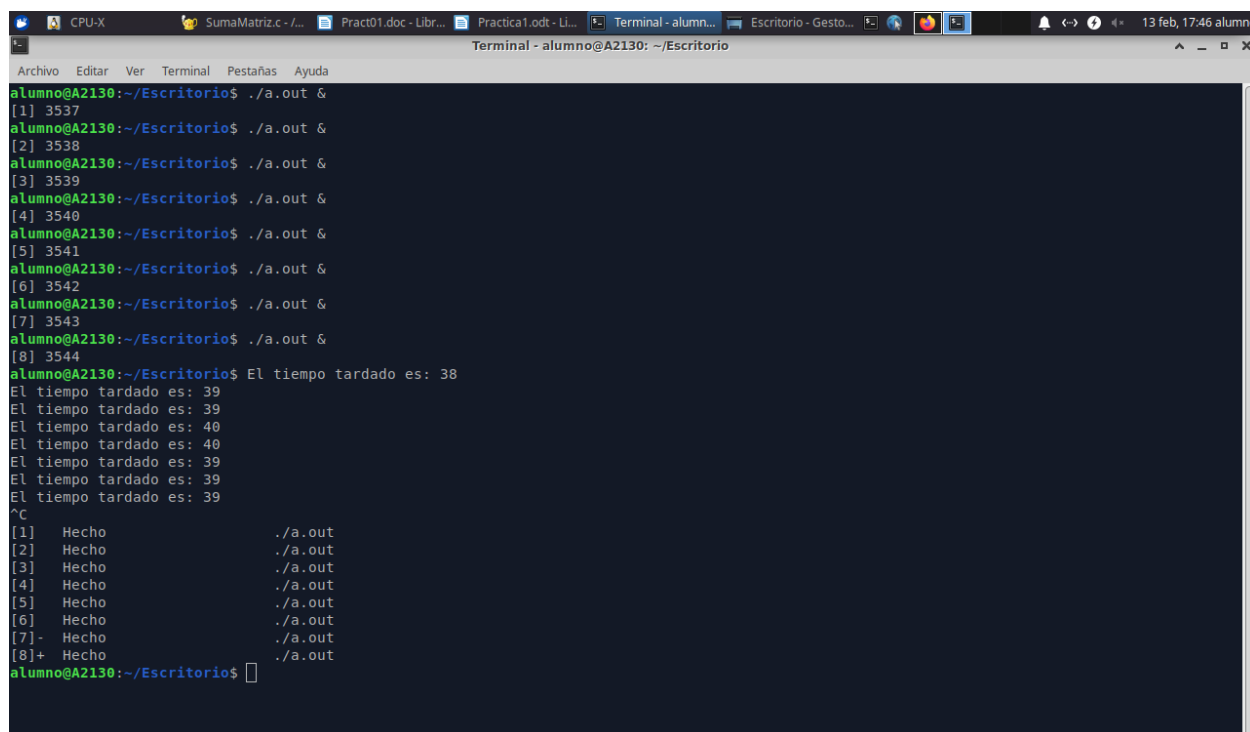
```

Figura 4: Compilación y ejecución con time

La sección analizada del programa tarda 17 segundos como se indica en las imágenes. Es posible que los tiempos no sean siempre exactos puesto que esto depende de que la CPU en ese momento esté más o menos libre.

En la imagen con "time", el primer tiempo (real) que aparece es el tiempo de respuesta (que es mayor), que hace referencia al tiempo que tarda el programa completo en terminar. El segundo tiempo (user) es, en cambio el tiempo de CPU.

c) La función Clock() mide el tiempo de CPU. El tiempo de CPU es el tiempo real que tarda la CPU en ejecutar una sección del programa. Es decir, el tiempo que la CPU está haciendo cosas. El tiempo de respuesta sin embargo, es el tiempo que tarda todo el programa en terminar.



```
alumno@A2130:~/Escritorio$ ./a.out &
[1] 3537
alumno@A2130:~/Escritorio$ ./a.out &
[2] 3538
alumno@A2130:~/Escritorio$ ./a.out &
[3] 3539
alumno@A2130:~/Escritorio$ ./a.out &
[4] 3540
alumno@A2130:~/Escritorio$ ./a.out &
[5] 3541
alumno@A2130:~/Escritorio$ ./a.out &
[6] 3542
alumno@A2130:~/Escritorio$ ./a.out &
[7] 3543
alumno@A2130:~/Escritorio$ ./a.out &
[8] 3544
alumno@A2130:~/Escritorio$ El tiempo tardado es: 38
El tiempo tardado es: 39
El tiempo tardado es: 39
El tiempo tardado es: 40
El tiempo tardado es: 40
El tiempo tardado es: 39
El tiempo tardado es: 39
El tiempo tardado es: 39
^C
[1] Hecho ./a.out
[2] Hecho ./a.out
[3] Hecho ./a.out
[4] Hecho ./a.out
[5] Hecho ./a.out
[6] Hecho ./a.out
[7] Hecho ./a.out
[8]+ Hecho ./a.out
alumno@A2130:~/Escritorio$
```

Figura 5: Ejecuciones

Como se puede observar, cuando ejecutamos 8 veces el mismo programa a la vez, el tiempo de ejecución del mismo aumenta considerablemente puesto que la CPU está más ocupada

3. Tercer Apartado

a) Rendimiento del PC:

- El tiempo de ejecución de la sección analizada del programa es de 17 segundos. A la hora de la realización de la práctica se nos olvidó hacer el casting de la función `Clock()` para obtener el resultado del tiempo más preciso. Es por eso, que hemos tomado el tiempo de ejecución preciso del valor que nos daba la función `time`, que son 17,798 segundos.
- Dentro del bucle más interno se encuentran 31 instrucciones de ensamblador, que son las que la CPU ejecuta. Este bucle se repite $25,000 \times 500 \times 500 = 6,250,000,000$ veces

- $NI = 31 \times 6,250,000,000 = 193,750,000,000$ instrucciones
Se tarda 17,798 segundos en ejecutar 193.750.000.000 instrucciones

- frecuencia CPU = 3.20 GHz

- periodo CPU = $\frac{1}{3,20} \times 10^9 = 3,125 \times 10^{10}$ segundos por ciclo

$$CPI = \frac{17,798}{193,750,000,000} \times 3,20 \times 10^9 = 0,29395 \text{ ciclos por instrucción}$$

- $MIPS = \frac{\frac{193,750,000,000}{17,798}}{10^6} = 10,886,0546$ millones de instrucciones por segundo

- $MFLOPS = \frac{\frac{1 \times 6,250,000,000}{17,798}}{10^6} = 351,163$ millones de operaciones de punto flotante por segundo