

2º curso / 2º cuatr.
Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 5. Optimización de código

Estudiante (nombre y apellidos): Juan Valentín Guerrero Cano

Grupo de prácticas y profesor de prácticas: AC3 Juan José Escobar

Fecha de entrega: 09/06/21

Fecha evaluación en clase: 02/06/21

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): *(respuesta)*

Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz

Sistema operativo utilizado: *(respuesta)*

Ubuntu 20.04.1 LTS

Versión de gcc utilizada: *(respuesta)*

g++-9 9.3.0-17ubuntu1~20.04

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

```
Archivo Editar Ver Buscar Terminal Ayuda
Arquitectura: x86_64
modo(s) de operación de las CPUs: 32-bit, 64-bit
Orden de los bytes: Little Endian
Address sizes: 39 bits physical, 48 bits virtual
CPU(s): 8
Lista de la(s) CPU(s) en línea: 0-7
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»: 4
«Socket(s)»: 1
Modo(s) NUMA: 1
ID de fabricante: GenuineIntel
Familia de CPU: 6
Modelo: 142
Nombre del modelo: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz
Revisión: 12
CPU MHz: 1400.043
CPU MHz máx.: 4900.0000
CPU MHz mín.: 400.0000
BogoMIPS: 4599.93
Virtualización: VT-x
Caché L1d: 128 KiB
Caché L1i: 128 KiB
Caché L2: 1 MiB
Caché L3: 8 MiB
CPU(s) del nodo NUMA 0: 0-7
Vulnerability Itlb multihit: KVM: Mitigation: VMX disabled
Vulnerability L1tf: Not affected
Vulnerability Mds: Not affected
Vulnerability Meltdown: Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation; Enhanced IBRS, IBPB conditional, RSB filling
Vulnerability Srbds: Mitigation; TSX disabled
Vulnerability Tsx async abort: Not affected
Indicadores: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1
gb rdtscp lm constant tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfperf pni pclmulqdq dtes64 monitor ds
_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf
lm abm 3dnowprefetch cpuid_fault epb invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_ad fsgs
base tsc_adjust bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat
pln pts hwp hwp_act_window hwp_epp md_clear flush_l1d arch_capabilities
[juanvalentin@valenting-Aspire-A515-54:~]$
[juanvalentin@valenting-Aspire-A515-54:~]$
```

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

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5
6  int main(int argc, char **argv)
7  {
8      struct timespec cgt1, cgt2;
9      double t;
10
11     if (argc < 2)
12     {
13         printf("[ERROR]-Debe insertar tamaño matriz\n");
14         exit(-1);
15     }
16
17     unsigned int N = atoi(argv[1]);
18     double **A, **B, **C;
19
20     A = (double **)malloc(N * sizeof(double *));
21     B = (double **)malloc(N * sizeof(double *));
22     C = (double **)malloc(N * sizeof(double *));
23
24     if ((A == NULL) || (B == NULL) || (C == NULL))
25         printf("\nNo hay suficiente espacio para la matriz\n");
26
27     for (int e = 0; e < N; e++){
28         A[e] = (double *) malloc(N * sizeof(double));
29         B[e] = (double *)malloc(N * sizeof(double));
30         C[e] = (double *)malloc(N * sizeof(double));
31
32         if ((A[e] == NULL) || (B[e] == NULL) || (C[e] == NULL))
33             printf("\nNo hay suficiente espacio para la matriz en la columna\n");
34     }
35     int i, j, k;
36     double suma;
37
38     srand48(time(NULL));
39
40     for (i = 0; i < N; i++)
41         for (j = 0; j < N; j++)
42         {
43             if (N < 9){
44                 B[i][j] = j + 1;
45                 C[i][j] = j + 1;
46             } else {
47                 B[i][j] = drand48();
48                 C[i][j] = drand48();
49             }
50         }

```

```

40     for (i = 0; i < N; i++)
41     for (j = 0; j < N; j++)
42     {
43         if (N < 9){
44             B[i][j] = j + 1;
45             C[i][j] = j + 1;
46         } else {
47             B[i][j] = drand48();
48             C[i][j] = drand48();
49         }
50     }
51
52     clock_gettime(CLOCK_REALTIME, &cgt1);
53
54
55     for (i = 0; i < N; i++)
56     for (j = 0; j < N; j++)
57     {
58         A[i][j] = 0;
59         for (k = 0; k < N; k++)
60             A[i][j] = A[i][j] + B[i][k] * C[k][j];
61     }
62
63     clock_gettime(CLOCK_REALTIME, &cgt2);
64
65     t = (double)(cgt2.tv_sec - cgt1.tv_sec) +
66         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
67
68
69     printf("Tiempo (seg): %0.9f\n", t);|
70
71
72     printf("\n_____ \nResultados:\n");
73
74     printf("\nMatriz B: \n");
75     if (N < 9)
76     for (i = 0; i < N; i++)
77     {
78         for (j = 0; j < N; j++)
79             printf("%f ", B[i][j]);
80         printf("\n");
81     }
82     else
83         printf("B[0][0]=%f B[%d][%d]=%f\n", B[0][0], N - 1, N - 1, B[N - 1][N - 1]);
84
85     printf("\nMatriz C: \n");
86     if (N < 9)
87     for (i = 0; i < N; i++)
88     {
89         for (j = 0; j < N; j++)

```

```

76     for (i = 0; i < N; i++)
77     {
78         for (j = 0; j < N; j++)
79             printf("%f ", B[i][j]);
80         printf("\n");
81     }
82 else
83     printf("B[0][0]=%f B[%d][%d]=%f\n", B[0][0], N - 1, N - 1, B[N - 1][N - 1]);
84
85 printf("\nMatriz C: \n");
86 if (N < 9)
87     for (i = 0; i < N; i++)
88     {
89         for (j = 0; j < N; j++)
90             printf("%f ", C[i][j]);
91         printf("\n");
92     }
93 else
94     printf("C[0][0]=%f C[%d][%d]=%f\n", C[0][0], N - 1, N - 1, C[N - 1][N - 1]);
95
96 printf("\nMatriz A=B*C: \n");
97 if (N < 9)
98     for (i = 0; i < N; i++)
99     {
100         for (j = 0; j < N; j++)
101             printf("%f ", A[i][j]);
102         printf("\n");
103     }
104 else
105     printf("A[0][0]=%f A[%d][%d]=%f\n", A[0][0], N - 1, N - 1, A[N - 1][N - 1]);
106
107 printf("\n");
108
109 for (int e = 0; e < N; e++)
110 {
111     free(A[e]);
112     free(B[e]);
113     free(C[e]);
114 }
115 free(A);
116 free(B);
117 free(C);
118
119 return 0;
120

```

(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).

He desenrollado el bucle con cuatro operaciones de cálculo por iteración teniendo en cuenta la posibilidad de que N no sea múltiplo de 4.

Modificación B) –explicación–: cambiamos el orden de la ejecución de los bucles para evitar penalizaciones de caché

...

CÓDIGOS FUENTE MODIFICACIONES**A) Captura de pmm-secuencial-modificado_A.c**

```

56     clock_gettime(CLOCK_REALTIME, &cgt1);
57
58     int mod = N % 4;
59     for (i = 0; i < N; i++)
60         for (j = 0; j < N; j++)
61         {
62             A[i][j] = 0;
63             for (k = 0; k < N - mod; k += 4)
64             {
65                 A[i][j] = A[i][j] + B[i][k] * C[k][j];
66                 A[i][j] = A[i][j] + B[i][k + 1] * C[k + 1][j];
67                 A[i][j] = A[i][j] + B[i][k + 2] * C[k + 2][j];
68                 A[i][j] = A[i][j] + B[i][k + 3] * C[k + 3][j];
69             }
70
71             if (N - mod + 2 < N)
72             {
73                 A[i][j] = A[i][j] + B[i][N - mod] * C[N - mod][j];
74                 A[i][j] = A[i][j] + B[i][N - mod + 1] * C[N - mod + 1][j];
75                 A[i][j] = A[i][j] + B[i][N - mod + 2] * C[N - mod + 2][j];
76             }
77             else if (N - mod + 1 < N)
78             {
79                 A[i][j] = A[i][j] + B[i][N - mod] * C[N - mod][j];
80                 A[i][j] = A[i][j] + B[i][N - mod + 1] * C[N - mod + 1][j];
81             }
82             else if (N - mod < N)
83             {
84                 A[i][j] = A[i][j] + B[i][N - mod] * C[N - mod][j];
85             }
86         }
87
88     clock_gettime(CLOCK_REALTIME, &cgt2);
89
90     t = (double)(cgt2.tv_sec - cgt1.tv_sec) +
91         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
92
93     printf("Tiempo (seg): %0.9f\n", t);
94
95     printf("\n_____ \nResultados:\n");
96
97     printf("\nMatriz B: \n");
98     if (N < 9)
99         for (i = 0; i < N; i++)
100         {
101             for (j = 0; j < N; j++)
102                 printf("%f ", B[i][j]);
103             printf("\n");

```

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

```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer0] 2021-06-09 miércoles
$gcc -O2 -o pmm-secuencial-modificado_A pmm-secuencial-modificado_A.c
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer0] 2021-06-09 miércoles
$./pmm-secuencial-modificado_A 12
Tiempo (seg): 0.000007062

Resultados:

Matriz B:
B[0][0]=0.017730 B[11][11]=0.168976

Matriz C:
C[0][0]=0.026457 C[11][11]=0.832550

Matriz A=B*C:
A[0][0]=2.847989 A[11][11]=2.359433

```

B) ...

Código:

```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
    {
        if (N < 9){
            B[i][j] = j + 1;
            C[i][j] = j + 1;
        } else {
            B[i][j] = drand48();
            C[i][j] = drand48();
        }
    }

clock_gettime(CLOCK_REALTIME, &cgt1);

for (i = 0; i < N; i++)
    for (k = 0; k < N; k++)
    {
        A[i][j] = 0;
        for (j = 0; j < N; j++)
            A[i][j] = A[i][j] + B[i][k] * C[k][j];
    }

clock_gettime(CLOCK_REALTIME, &cgt2);

t = (double)(cgt2.tv_sec - cgt1.tv_sec) +
    (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));

printf("Tiempo (seg): %0.9f\n", t);

printf("\n_____ \nResultados:\n");

printf("\nMatriz B: \n");
if (N < 9)
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
            printf("%f ", B[i][j]);
        printf("\n");
    }
```

Compilación y ejecución:

```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer0] 2021-06-09 miércoles
$gcc -O2 -o pmm-secuencial-modificado_B pmm-secuencial-modificado_B.c
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer0] 2021-06-09 miércoles
$./pmm-secuencial-modificado_B 12
Tiempo (seg): 0.000006322

Resultados:

Matriz B:
B[0][0]=0.495811 B[11][11]=0.965574

Matriz C:
C[0][0]=0.687124 C[11][11]=0.417072

Matriz A=B*C:
A[0][0]=3.298858 A[11][11]=3.772472

[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer0] 2021-06-09 miércoles
$

```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	Código secuencial de cálculo de multiplicación de matrices	0.000012035
Modificación A)	Desenrollado de bucle (4 operaciones por iteración)	0.000007062
Modificación B)	Reajuste orden de bucles(evitar penalizaciones)	0.000006322
...		

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

Ambas modificaciones mejoran el tiempo de ejecución del código, un poco más el reajuste en los bucles ya que se solventa las penalizaciones por parte de la caché

2. (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


```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define MAX 1001
6
7  struct s
8  {
9      int a;
10     int b;
11 } s[MAX];
12
13 int main(int argc, char **argv)
14 {
15     if (argc < 3)
16         printf("\nIntroduzca el numero de parametros correcto N,M\n");
17
18     int N = atoi(argv[1]);
19     int M = atoi(argv[2]);
20
21     struct timespec cgt1, cgt2;
22     double t;
23
24     struct s s[N];
25     int R[M];
26
27     int i, ii, X1, X2;
28     srand48(time(NULL));
29
30
31     for (i = 0; i < N; i++)
32     {
33         if (N < 9 && M < 9){
34             s[i].a = i;
35             s[i].b = i;
36         } else {
37             s[i].a = drand48();
38             s[i].b = drand48();
39         }
40     }
41
42     clock_gettime(CLOCK_REALTIME, &cgt1);
43
44     for (ii = 0; ii < M; ii++)
45     {
46         X1 = 0;
47         X2 = 0;
48         for (i = 0; i < N; i++)
49             X1 += 2 * s[i].a + ii;
50         for (i = 0; i < N; i++)

```

```

31     for (i = 0; i < N; i++)
32     {
33         if (N < 9 && M < 9){
34             s[i].a = i;
35             s[i].b = i;
36         } else {
37             s[i].a = drand48();
38             s[i].b = drand48();
39         }
40     }
41
42     clock_gettime(CLOCK_REALTIME, &cgt1);
43
44     for (ii = 0; ii < M; ii++)
45     {
46         X1 = 0;
47         X2 = 0;
48         for (i = 0; i < N; i++)
49             X1 += 2 * s[i].a + ii;
50         for (i = 0; i < N; i++)
51             X2 += 3 * s[i].b - ii;
52         if (X1 < X2)
53             R[ii] = X1;
54         else
55             R[ii] = X2;
56     }
57
58     clock_gettime(CLOCK_REALTIME, &cgt2);
59
60     t = (double)(cgt2.tv_sec - cgt1.tv_sec) +
61         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
62
63     printf("Tiempo (seg): %f\n", t);
64
65
66     printf("\n\nResultados:\n");
67
68     printf("\nVector R: \n");
69     printf("R[0]=%d R[39999]=%d\n", R[0], R[M - 1]);
70
71     return 0;
72 }
73

```


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–: En el mismo bucle recorreremos a y b, sin saltos, ya que traemos el struct completo de caché una solo vez recorriéndolo de forma contigua

Modificación B) –explicación–: Añadimos a la modificación anterior un desenrollado de bucle

...

CÓDIGOS FUENTE MODIFICACIONES


A) Captura figura1-modificado_A.c

```

39
40     clock_gettime(CLOCK_REALTIME, &cgt1);
41
42     for (ii = 0; ii < M; ii++)
43     {
44         X1 = 0;
45         X2 = 0;
46         for (i = 0; i < N; i++){
47             X1 += 2 * s[i].a + ii;
48             X2 += 3 * s[i].b - ii;
49         }
50         if (X1 < X2)
51             R[ii] = X1;
52         else
53             R[ii] = X2;
54     }
55
56     clock_gettime(CLOCK_REALTIME, &cgt2);
57

```

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



```
Terminal
Archivo Editar Ver Buscar Terminal Ayuda
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer2] 2021-06-09 miércoles
$gcc -O2 -o figura1-modificadoA figura1-modificadoA.c
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer2] 2021-06-09 miércoles
$./figura1-modificadoA 1500 1500
Tiempo (seg): 0.007330

Resultados:

Vector R:
R[0]=0 R[39999]=-2248500
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer2] 2021-06-09 miércoles
$
```

B) ...

```

C figura1-modificadoB.c x
ejer2 > C figura1-modificadoB.c > ...
40
41     clock_gettime(CLOCK_REALTIME, &cgt1);
42     int mod = N % 4;
43
44     for (ii = 0; ii < M; ii++)
45     {
46         X1 = 0;
47         X2 = 0;
48         for (i = 0; i < N - mod; i+=4){
49
50             X1 += 2 * s[i].a + ii;
51             X1 += 2 * s[i + 1].a + ii;
52             X1 += 2 * s[i + 2].a + ii;
53             X1 += 2 * s[i + 3].a + ii;
54
55             X2 += 3 * s[i].b - ii;
56             X2 += 3 * s[i + 1].b - ii;
57             X2 += 3 * s[i + 2].b - ii;
58             X2 += 3 * s[i + 3].b - ii;
59
60             if (N - mod + 2 < N)
61             {
62                 X1 += 2 * s[i].a + ii;
63                 X1 += 2 * s[i + 1].a + ii;
64                 X1 += 2 * s[i + 2].a + ii;
65                 X2 += 3 * s[i].b - ii;
66                 X2 += 3 * s[i + 1].b - ii;
67                 X2 += 3 * s[i + 2].b - ii;
68             }
69             else if (N - mod + 1 < N)
70             {
71                 X1 += 2 * s[i].a + ii;
72                 X1 += 2 * s[i + 1].a + ii;
73                 X2 += 3 * s[i].b - ii;
74                 X2 += 3 * s[i + 1].b - ii;
75             }
76             else if (N - mod < N)
77             {
78                 X1 += 2 * s[i].a + ii;
79                 X2 += 3 * s[i].b - ii;
80             }
81         }
82         if (X1 < X2)
83             R[ii] = X1;
84         else
85             R[ii] = X2;
86     }
87
88
89     clock_gettime(CLOCK_REALTIME, &cgt2);

```

```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer2] 2021-06-09 miércoles
$gcc -O2 -o figura1-modificadoB figura1-modificadoB.c
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer2] 2021-06-09 miércoles
$./figura1-modificadoB 1500 1500
Tiempo (seg): 0.003515

Resultados:

Vector R:
R[0]=0 R[39999]=-2248500
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer2] 2021-06-09 miércoles
$

```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
--------------	---	-----

Sin modificar	Código sin modificar ($N = M = 1500$)	0.009005
Modificación A)	A y B traídos de caché en el mismo bucle	0.007330
Modificación B)	Modificación A) + Desenrollado de bucle (4 operaciones)	0.003515
...		

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

Como cabía de esperar, el desenrollado de bucle hace que la modificación A sea mejor incluso. La modificación A reduce el tiempo gracias a reducir los accesos a la caché.

- 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 (*Double precision- real Alpha X Plus 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 acarrearán. 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

```
ejer3 > C daxpy.c main(int, char **)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main(int argc, char **argv)
6  {
7      if (argc < 2)
8      {
9          fprintf(stderr, "Falta num\n");
10         exit(-1);
11     }
12
13     int N = atoi(argv[1]);
14
15     struct timespec ini, fin;
16     double tiempo;
17
18     int i, a = 47;
19     int x[N], y[N];
20
21     for (i = 1; i <= N; i++)
22     {
23         x[i] = i;
24         y[i] = i;
25     }
26
27     clock_gettime(CLOCK_REALTIME, &ini);
28
29     for (i = 1; i <= N; i++)
30     {
31         y[i] = a * x[i] + y[i];
32     }
33     clock_gettime(CLOCK_REALTIME, &fin);
34
35     tiempo = (double)(fin.tv_sec - ini.tv_sec) + (double)((fin.tv_nsec - ini.tv_nsec) / (1.e+9));
36
37     printf("Tiempo(seg): %f\n", tiempo);
38 }
```

Tiempos ejec. Longitud vectores=5000	-O0	-Os	-O2	-O3
	0.000048	0.000010	0.000012	0.000005

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

```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
Sgcc -O0 -o daxpy daxpy.c
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
$./daxpy 5000
Tiempo(seg): 0.000012
y[0]=0, y[N-1]=239952
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
Sgcc -Os -o daxpy daxpy.c
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
$./daxpy 5000
Tiempo(seg): 0.000010
y[0]=0, y[N-1]=239952
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
Sgcc -O2 -o daxpy daxpy.c
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
$./daxpy 5000
Tiempo(seg): 0.000012
y[0]=0, y[N-1]=239952
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
Sgcc -O3 -o daxpy daxpy.c
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
$./daxpy 5000
Tiempo(seg): 0.000005
y[0]=0, y[N-1]=239952
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
Sgcc -O0 -o daxpy daxpy.c
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
$./daxpy 5000
Tiempo(seg): 0.000048
y[0]=0, y[N-1]=239952
[JuanValentinGuerreroCano valenting@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer3] 2021-06-09 miércoles
$

```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

En -O0 no obtenemos ningún tipo de mejora en el código ni de optimización. En -Os disminuimos el tamaño del ejecutable, siendo el peso del archivo menor. En -O2 uso de instrucciones más precisas y eficientes. En -O3 el código ensamblador es más complejo y se llama a un mayor número de subrutinas.

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)

daxpy00.s	daxpy0s.s	daxpy02.s	daxpy03.s
<pre> 69 movl \$16, %eax 70 subq \$1, %rax 71 addq %rdx, %rax 72 movl \$16, %esi 73 movl \$0, %edx 74 divq %rsi 75 imulq \$16, %rax, %rax 76 movq %rax, %rdx 77 andq \$-4096, %rdx 78 movq %rsp, %rsi 79 subq %rdx, %rsi 80 movq %rsi, %rdx 81 .L3: 82 cmpq %rdx, %rsp 83 je .L4 84 subq \$4096, %rsp 85 orq \$0, 4088(%rsp) 86 jmp .L3 87 .L4: 88 movq %rax, %rdx 89 andl \$4095, %edx 90 subq %rdx, %rsp 91 movq %rax, %rdx 92 andl \$4095, %edx 93 testq %rdx, %rdx 94 je .L5 95 andl \$4095, %eax 96 subq \$8, %rax 97 addq %rsp, %rax 98 orq \$0, (%rax) 99 .L5: 100 movq %rsp, %rax 101 addq \$3, %rax 102 shrq \$2, %rax 103 salq \$2, %rax 104 movq %rax, -128(%rbp) 105 movl -148(%rbp), %eax 106 movslq %eax, %rdx 107 subq \$1, %rdx 108 movq %rdx, -120(%rbp) 109 movslq %eax, %rdx 110 movq %rdx, %r14 111 movl \$0, %r15d 112 movslq %eax, %rdx 113 movq %rdx, %r12 114 movl \$0, %r13d </pre>	<pre> 68 movq %rsp, %r14 69 subq %rax, %rcx 70 andq \$-16, %rdx 71 movq %rcx, %rax 72 .L6: 73 cmpq %rax, %rsp 74 je .L7 75 subq \$4096, %rsp 76 orq \$0, 4088(%rsp) 77 jmp .L6 78 .L7: 79 movq %rdx, %rax 80 andl \$4095, %eax 81 subq %rax, %rsp 82 testq %rax, %rax 83 je .L8 84 orq \$0, -8(%rsp,%rax) 85 .L8: 86 leaq 3(%rsp), %r13 87 movl \$1, %eax 88 movq %r13, %r12 89 andq \$-4, %r13 90 shrq \$2, %r12 91 .L9: 92 cmpl %eax, %ebx 93 jl .L22 94 movl %eax, (%r14,%rax,4) 95 movl %eax, 0(%r13,%rax,4) 96 incq %rax 97 jmp .L9 98 .L22: 99 leaq -72(%rbp), %rsi 100 xorl %edi, %edi 101 call clock_gettime@PLT 102 xorl %eax, %eax 103 .L11: 104 incq %rax 105 cmpl %eax, %ebx 106 jl .L23 107 imull \$47, (%r14,%rax,4), %edx 108 addl %edx, 0(%r13,%rax,4) 109 jmp .L11 110 .L23: 111 xorl %edi, %edi 112 leaq -56(%rbp), %rsi 113 decl %ebx </pre>	<pre> 45 cltq 46 leaq 15(,%rax,4), %rax 47 movq %rax, %rcx 48 movq %rax, %rdx 49 andq \$-4096, %rcx 50 andq \$-16, %rdx 51 subq %rcx, %rdi 52 movq %rdi, %rcx 53 cmpq %rcx, %rsp 54 je .L4 55 .L28: 56 subq \$4096, %rsp 57 orq \$0, 4088(%rsp) 58 cmpq %rcx, %rsp 59 jne .L28 60 .L4: 61 andl \$4095, %edx 62 subq %rdx, %rsp 63 testq %rdx, %rdx 64 jne .L29 65 .L5: 66 movq %rax, %rdx 67 movq %rsp, %rcx 68 andq \$-4096, %rax 69 movq %rsp, %r15 70 subq %rax, %rcx 71 andq \$-16, %rdx 72 movq %rcx, %rax 73 .L6: 74 cmpq %rax, %rsp 75 je .L7 76 subq \$4096, %rsp 77 orq \$0, 4088(%rsp) 78 jmp .L6 79 .L29: 80 orq \$0, -8(%rsp,%rdx) 81 jmp .L5 82 .L7: 83 andl \$4095, %edx 84 subq %rdx, %rsp 85 testq %rdx, %rdx 86 jne .L30 87 .L8: 88 leaq 3(%rsp), %rbx 89 movq %rbx, %r13 90 andq \$-4, %rbx </pre>	<pre> 170 duull %eax, %ecx 171 imull \$47, (%r14,%rdx,4), %ecx 172 addl %ecx, (%rbx,%rdx,4) 173 cmpl %eax, %r15d 174 jle .L19 175 cltq 176 imull \$47, (%r14,%rax,4), %edx 177 addl %edx, (%rbx,%rax,4) 178 .L19: 179 xorl %edi, %edi 180 leaq -80(%rbp), %rsi 181 movslq %r12d, %r12 182 call clock_gettime@PLT 183 movq -72(%rbp), %rax 184 pxor %xmm0, %xmm0 185 subq -88(%rbp), %rax 186 cvtsi2sdq %rax, %xmm0 187 pxor %xmm1, %xmm1 188 movq -80(%rbp), %rax 189 subq -96(%rbp), %rax 190 cvtsi2sdq %rax, %xmm1 191 divsd .LC3(%rip), %xmm0 192 movl (%rbx,%r12,4), %ecx 193 movl 0(%r13,4), %edx 194 leaq .LC4(%rip), %rsi 195 movl \$1, %edi 196 movl \$1, %eax 197 addsd %xmm1, %xmm0 198 call __printf_chk@PLT 199 movq -56(%rbp), %rax 200 xorq %fs, %rax 201 jne .L47 202 leaq -40(%rbp), %rsp 203 xorl %eax, %eax 204 popq %rbx 205 popq %r12 206 popq %r13 207 popq %r14 208 popq %r15 209 popq %rbp 210 .cfi_remember_state 211 .cfi_def_cfa 7, 8 212 ret 213 .L12: 214 .cfi_restore_state 215 leaq -96(%rbp), %rsi 216 xorl %edi, %edi </pre>

4. (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:

Lo hacemos primero para el mismo número de componentes que en el ejercicio para observar la diferencia de tiempo

```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer4] 2021-06-09 miércoles
$gcc -O2 -o pmm-paralelo pmm-paralelo.c
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer4] 2021-06-09 miércoles
$./pmm-paralelo 12
Tiempo (seg): 0.000001671

Resultados:

Matriz B:
B[0][0]=0.478752 B[11][11]=0.473052

Matriz C:
C[0][0]=0.791909 C[11][11]=0.701371

Matriz A=B*C:
A[0][0]=1.732631 A[11][11]=2.622170

[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer4] 2021-06-09 miércoles
$./pmm-paralelo 500
Tiempo (seg): 0.104880143

Resultados:

Matriz B:
B[0][0]=0.703566 B[499][499]=0.574740

Matriz C:
C[0][0]=0.019452 C[499][499]=0.186112

Matriz A=B*C:
A[0][0]=120.054940 A[499][499]=123.442126

```

Segunda ejecución para los mismos valores

```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer4] 2021-06-09 miércoles
$gcc -O2 -o pmm-secuencial pmm-secuencial.c
[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer4] 2021-06-09 miércoles
$./pmm-secuencial 12
Tiempo (seg): 0.000004954

Resultados:

Matriz B:
B[0][0]=0.931684 B[11][11]=0.862679

Matriz C:
C[0][0]=0.340607 C[11][11]=0.080488

Matriz A=B*C:
A[0][0]=3.892883 A[11][11]=3.466502

[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer4] 2021-06-09 miércoles
$./pmm-secuencial 500
Tiempo (seg): 0.173656392

Resultados:

Matriz B:
B[0][0]=0.639707 B[499][499]=0.967236

Matriz C:
C[0][0]=0.386511 C[499][499]=0.334385

Matriz A=B*C:
A[0][0]=129.392817 A[499][499]=121.650803

[juanvalentin@valenting-Aspire-A515-54:~/Escritorio/bp4/ejer4] 2021-06-09 miércoles
$

```


CAPTURA CÓDIGO FUENTE: pmm-paralelo.c

```
55  
56     clock_gettime(CLOCK_REALTIME, &cgt1);  
57  
58  
59     #pragma omp parallel for  
60     for (i = 0; i < N; i++)  
61         for (k = 0; k < N; k++)  
62         {  
63             A[i][j] = 0;  
64             for (j = 0; j < N; j++)  
65                 A[i][j] = A[i][j] + B[i][k] * C[k][j];  
66         }  
67  
68     clock_gettime(CLOCK_REALTIME, &cgt2);  
69  
70     t = (double)(cgt2.tv_sec - cgt1.tv_sec) +  
71         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));  
72
```

(b) RESPUESTA

Ganancia1: $G_1 = 0.000004954 / 0.000001671 = 2.964691$

Ganancia2: $G_2 = 0.173656392 / 0.104880143 = 1.655760$