2º curso / 2º cuatr.

Grado Ing. Inform.

# Arquitectura de Computadores (AC)

## Cuaderno de prácticas.
## Bloque Práctico 5. Optimización de código

Estudiante (nombre y apellidos): Brian Sena Simons
Grupo de prácticas y profesor de prácticas: 2ºB
Fecha de entrega: ¿¿?¿
Fecha evaluación en clase: ¿?¿?

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):

AuthenticAMD – AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx.

Sistema operativo utilizado:

*Windows subsystem for Linux version 1.0.*

Versión de gcc utilizada:

gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0

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

```
1-05-19-06:05 BrianSenaSimons@Ubuntu-20.04:/proc$ cd ~
1-05-19-06:05 BrianSenaSimons@Ubuntu-20.04:~$ lscpu
uitectura:                       x86_64
o(s) de operación de las CPUs:   32-bit, 64-bit
en de los bytes:                 Little Endian
ress sizes:                      36 bits physical, 48 bits virtual
(s):                             8
ta de la(s) CPU(s) en línea:     0-7
o(s) de procesamiento por núcleo: 2
leo(s) por «socket»:             4
cket(s)»                         1
de fabricante:                   AuthenticAMD
ilia de CPU:                     23
elo:                             24
bre del modelo:                  AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx
isión:                           1
 MHz:                            2100.000
 MHz máx.:                       2100,0000
oMIPS:                           4200.00
ricante del hipervisor:          Windows Subsystem for Linux
o de virtualización:             contenedor
icadores:                        fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mm
                                  fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm pni pclmulqdq ssse3
                                 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave osxsave avx f16c rdrand hypervisor l
                                 hf_lm cmp_legacy cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw wdt topoext f
                                 gsbase bmi1 avx2 smep bmi2 rdseed adx smap clflushopt sha_ni
1-05-19-06:05 BrianSenaSimons@Ubuntu-20.04:~$
```

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`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
#endif

int main(int argc, char const* argv[])
{
  if(argc<2){
    fprintf(stderr, "[ERROR] Exec: ./{this} {size}");
    exit(-1);
  }

  unsigned int n=atoi(argv[1]);
  double **m, **a, **b;
  a = (double**)malloc(n*sizeof(double*));
  b = (double**)malloc(n*sizeof(double*));
  m = (double**)malloc(n*sizeof(double*));

  if(a==NULL | b==NULL | m == NULL){
    fprintf(stderr,"[MEM_ERROR]");
    exit(-2);
  }

  //Inicializacion
  srand(time(NULL));
  for(int i=0;i<n;++i){
    m[i] = (double*)malloc(n*sizeof(double));
    a[i] = (double*)malloc(n*sizeof(double));
    b[i] = (double*)malloc(n*sizeof(double));
    if(a[i]==NULL | b[i]==NULL |  m[i]==NULL){
    fprintf(stderr,"[MEM_ERROR]");
    exit(-2);
    } for(int j=0;j<n;j++){
      a[i][j]= drand48();
      b[i][j] = drand48();
      m[i][j] = 0;
    }
  }

```

```
28   //calculo
27   double t_ini = omp_get_wtime();
26   for(int i=0;i<n;i++){
25     for(int j = 0; j<n;j++){
24       for(int k=0; k<n;k++){
23         m[i][j] = a[i][k] * b[k][j];
22       }
21     }
20   }
19   double tiempo = omp_get_wtime() - t_ini;
18
17   printf("Tiempo(seg.): %11.9f\n", tiempo);
16
15   printf("m[%d] = %d\n",0,m[0][0]);
14   printf("m[%d] = %d\n",n-1,m[n-1][n-1]);
13
12   for(int i=0;i<n;++i){
11     free(m[i]);
10     free(a[i]);
 9     free(b[i]);
 8   }
 7
 6   free(m);
 5   free(a);
 4   free(b);
 3
 2   return 0;
 1
74 }
NORMAL  pmm-secuencial.c                                    unix | u
HardTimeOff
```

```
2021-05-19-06:41 BrianSenaSimons@Ubuntu-20.04:~$ cd Uni2/AC/BP4/ejer1/
2021-05-19-06:41 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencial 10
Tiempo(seg.): 0.000001400
m[0] = 0.040104
m[9] = 0.308990
2021-05-19-06:42 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencial 100
Tiempo(seg.): 0.000010600
m[0] = 0.186325
m[99] = 0.342855
2021-05-19-06:42 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencial 1000
Tiempo(seg.): 0.001893200
m[0] = 0.529797
m[999] = 0.150035
2021-05-19-06:42 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencial 10000
Tiempo(seg.): 0.161114400
m[0] = 0.251861
m[9999] = 0.248257
2021-05-19-06:42 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencial 30000
     ^C^C^C^C^C^C^C
2021-05-19-06:46 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$   Not a good idea ^^^^ xd
```

**(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-: (**Diap. 9 Seminario 4). Realizar el bucle de 4 en 4.

**Modificación B) –explicación-:** (Diap. 16 Seminario 4). Localidad de los accesos, cambiamos los índices de acceso a las matrices.

**...**

**CÓDIGOS FUENTE MODIFICACIONES**

**A) Captura de** `pmm-secuencial-modificado_A.c`

```
El Problema es cuando N no es divisible por 4.
```

```
18        m[i][j] = 0;
17      }
16    }
15
14    //calculo
13    double t_ini = omp_get_wtime();
12    for(int i=0;i<n;i++){
11      for(int j=0; j<n;j+=4){
10        for(int k=0; k<n;k++){
 9          m[i][j] = a[i][k] * b[k][j];
 8          m[i][j+1] = a[i][k] * b[k][j+1];
 7          m[i][j+2] = a[i][k] * b[k][j+2];
 6          m[i][j+3] = a[i][k] * b[k][j+3];
 5        }
 4      }
 3    }
 2    double tiempo = omp_get_wtime() - t_ini;
 1
60    printf("Tiempo(seg.): %11.9f\n", tiempo);
 1
 2    printf("m[%d] = %f\n",0,m[0][0]);
 3    printf("m[%d] = %f\n",n-1,m[n-1][n-1]);
 4
 5    for(int i=0;i<n;++i){
 6      free(m[i]);
 7      free(a[i]);
```

```
Modificación B:
```

```
3   }
4
5   //calculo
6   double t_ini = omp_get_wtime();
7   for(int i=0;i<n;i++){
8     for(int j=0; j<n;j++){
9       for(int k=0; k<n;k+=4){
10        m[i][k] = a[i][j] * b[j][k];
11        m[i][k+1] = a[i][j] * b[j][k+1];
12        m[i][k+2] = a[i][j] * b[j][k+2];
13        m[i][k+3] = a[i][j] * b[j][k+3];
14      }
15    }
16  }
17  double tiempo = omp_get_wtime() - t_ini;
18
19  printf("Tiempo(seg.): %11.9f\n", tiempo);
20
21  printf("m[%d] = %f\n",0,m[0][0]);
```

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

**Modificación A:**

```
021-05-19-07:06 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ nvim pmm-secuencialMod.c
021-05-19-07:08 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ gcc -O2 -fopenmp -o pmm-secuencialMod pmm-secuenci
d.c
021-05-19-07:08 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencialMod 10
iempo(seg.): 0.000001300
[0] = 0.040104
[9] = 0.308990
unmap_chunk(): invalid pointer
bortado (`core' generado)
021-05-19-07:08 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencialMod 100
iempo(seg.): 0.000005600
[0] = 0.186325
[99] = 0.342855
021-05-19-07:08 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencialMod 1000
iempo(seg.): 0.001773800
[0] = 0.529797
[999] = 0.150035
021-05-19-07:08 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencialMod 10000
iempo(seg.): 0.157440300
[0] = 0.251861
[9999] = 0.248257
021-05-19-07:08 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$
```

**Modificación B:**

```
ModB.c
2021-05-19-07:19 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencialModB 10
Tiempo(seg.): 0.000002500
m[0] = 0.040104
m[9] = 0.308990
munmap_chunk(): invalid pointer
Abortado (`core' generado)
2021-05-19-07:19 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencialModB 100
Tiempo(seg.): 0.000321300
m[0] = 0.186325
m[99] = 0.342855
2021-05-19-07:19 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencialModB 1000
Tiempo(seg.): 0.824760700
m[0] = 0.529797
m[999] = 0.150035
2021-05-19-07:19 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer1$ ./pmm-secuencialModB 10000
```

**TIEMPOS:**

| **Modificación** | **Breve descripción de las modificaciones** | **-O2** |
|---|---|---|
| Sin modificar | | *De 0,25 a 10 seg. aquí* |
| Modificación A) | Desenrollado del bucle interior | 9.5x10^-4 segs menos en media. |
| Modificación B) | Cambio de Índices del bucle + Desenrollado | No Bueno, tarda más |
| ... | | |
| | | |

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

Al utilizar un incremento de 4 en 4 nos ahorramos varias instrucciones a código máquina reduciendo el número de vueltas que da el bucle. Sobre todo para un N muy grande.

Luego el intercambio de índices era para poder jugar con el principio de localidad espacial al que se rigen las memorias. Sin embargo, con mi arquitectura no ha provocado ninguna mejora, si no que la solución de la modificación A de por si, ha sido la mejor.

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`

```c
#include <time.h>
#include <stdlib.h>
#include <omp.h>
#include <stdio.h>


int main(int argc, const char* argv[])
{

  if(argc<3){
    fprintf(stderr,"[ERROR] Exec: ./{this} {size1} {size2}");
    exit(-1);
  }

  int N = atoi(argv[1]);
  int M = atoi(argv[2]);

  struct {
    int a;
    int b;
  } S[N];

  int R[M];
  for(int i=0;i<M;i++)
    R[i] = 0;

  srand(time(NULL));

  //Inicializacion
  if(N<8){
    for(int i=0;i<N;i++){
      S[i].a = i;
      S[i].b = N-i;
    }
  }else{
    for(int i=0;i<N;i++){
      S[i].a = rand()%255;
      S[i].b = rand()%255;
    }
  }

  //calculo
  int X1, X2;
  double t_ini = omp_get_wtime();

  for(int i=0;i<M;i++){
    X1=0; X2=0;
```

```
MAL  Figura1.c                                          unix | utf-8 |
ura1.c" 59L, 909C escritos
```

```
46   }
45
44   int N = atoi(argv[1]);
43   int M = atoi(argv[2]);
42
41   struct {
40     int a;
39     int b;
38   } S[N];
37
36   int R[M];
35   for(int i=0;i<M;i++)
34     R[i] = 0;
33
32   srand(time(NULL));
31
30   //Inicializacion
29   if(N<8){
28     for(int i=0;i<N;i++){
27       S[i].a = i;
26       S[i].b = N-i;
25     }
24   }else{
23     for(int i=0;i<N;i++){
22       S[i].a = rand()%255;
21       S[i].b = rand()%255;
20     }
19   }
18
17   //calculo
16   int X1, X2;
15   double t_ini = omp_get_wtime();
14
13   for(int i=0;i<M;i++){
12     X1=0; X2=0;
11     for(int j=0;j<N;j++) X1+=2*S[i].a+i;
10     for(int j=0;j<N;j++) X2+=3*S[i].a-i;
 9
 8     if (X1<X2) R[i]=X1;
 7     else R[i]=X2;
 6   }
 5
 4   double tiempo = omp_get_wtime() - t_ini;
 3
 2   printf("Tiempo(segs.): %11.9f\n", tiempo);
 1
59 }
NORMAL  Figura1.c                                              unix | utf-8
Figura1.c" 59L, 909C escritos
```

```
2021-05-19-08:00 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1 1000 1000
Tiempo(segs.): 0.000002600
2021-05-19-08:00 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1 10000 10000
Tiempo(segs.): 0.000017200
2021-05-19-08:00 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1 50000 50000
Tiempo(segs.): 0.000083700
2021-05-19-08:00 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1 100000 100000
Tiempo(segs.): 0.000164800
2021-05-19-08:00 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$
```

**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-:** Cambiamos los operadores * por << Que son más eficientes a nivel máquina.

**Modificación B) –explicación-:** Cambiamos el struct por el tipo de acceso que estamos implementando. (Diap. 15 Seminario 4).

**Modificación C) -explicación:** Cambiamos el if/else para reducir el número de saltos.

**CÓDIGOS FUENTE MODIFICACIONES**

**A) Captura** figura1`-modificado_A.c`

```
24  }else{
23    for(int i=0;i<N;i++){
22      S[i].a = rand()%255;
21      S[i].b = rand()%255;
20    }
19  }
18
17  //calculo
16  int X1, X2;
15  double t_ini = omp_get_wtime();
14
13  for(int i=0;i<M;i++){
12    X1=0; X2=0;
11    for(int j=0;j<N;j++) X1+=(S[i].a>>1)+i;
10    for(int j=0;j<N;j++) X2+=(S[i].a>>1 + S[i].a)-i;
 9
 8    if (X1<X2) R[i]=X1;
 7    else R[i]=X2;
 6  }
 5
 4  double tiempo = omp_get_wtime() - t_ini;
 3
 2  printf("Tiempo(segs.): %11.9f\n", tiempo);
```

Modificación B:

```
13
14  int N = atoi(argv[1]);
15  int M = atoi(argv[2]);
16
17  struct {
18    int a[N];
19    int b[N];
20  } S;
21
22  int R[M];
23  for(int i=0;i<M;i++)
24    R[i] = 0;
25
26  srand(time(NULL));
27
28  //Inicializacion
29  if(N<8){
30    for(int i=0;i<N;i++){
31      S.a[i] = i;
32      S.b[i] = N-i;
```

Modificación C:

```
13   int X1, X2;
14   double t_ini = omp_get_wtime();
15
16   for(int i=0;i<M;i++){
17     X1=0; X2=0;
18     for(int j=0;j<N;j++) X1+=(S.a[i]>>1)+i;
19     for(int j=0;j<N;j++) X2+=(S.a[i]>>1 + S.a[i])-i;
20
21     /*if (X1<X2) R[i]=X1;
22     else R[i]=X2;*/
23     R[i] = (X1 < X2) ? X1 : X2;
24   }
25
26   double tiempo = omp_get_wtime() - t_ini;
```

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

```
2021-05-19-08:07 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1A 1000 1000
Tiempo(segs.): 0.000002400
2021-05-19-08:07 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1A 10000 10000
Tiempo(segs.): 0.000017100
2021-05-19-08:07 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1A 50000 50000
Tiempo(segs.): 0.000083200
2021-05-19-08:07 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1A 100000 100000
Tiempo(segs.): 0.000164700
```

```
.c
2021-05-19-08:11 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1B 1000 1000
Tiempo(segs.): 0.000002900
2021-05-19-08:11 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1B 10000 10000
Tiempo(segs.): 0.000036400
2021-05-19-08:11 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1B 50000 50000
Tiempo(segs.): 0.000124300
2021-05-19-08:11 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1B 100000 100000
Tiempo(segs.): 0.000165100
2021-05-19-08:12 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$
```

```
2021-05-19-08:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ gcc -O2 -fopenmp -o Figura1C Figura1C
.c
2021-05-19-08:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1C 1000 1000
Tiempo(segs.): 0.000002600
2021-05-19-08:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1C 10000 10000
Tiempo(segs.): 0.000017200
2021-05-19-08:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1C 50000 50000
Tiempo(segs.): 0.000082700
2021-05-19-08:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$ ./Figura1C 100000 100000
Tiempo(segs.): 0.000164600
2021-05-19-08:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer2$
```

**TIEMPOS:**

| Modificación | Breve descripción de las modificaciones | -O2 |
|---|---|---|
| Sin modificar | | *De 0,25 a 10 seg. aquí* |

| | | |
|---|---|---|
| Modificación A) | Cambiamos los operadores * por << Que son más eficientes a nivel máquina. | Pequeña mejora. |
| Modificación B) | Cambiamos el struct por el tipo de acceso que estamos implementando. (Diap. 15 Seminario 4). | Mismo tiempo de ejecución. |
| Modificación C) | Cambiamos el if/else para reducir el número de saltos. | Mismo Tiempo de ejecución. |
| | | |

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

Pues al final, en mi arquitectura al menos, los cambios realizados han tenido poco impacto en el tiempo de ejecución general de la aplicación. Primeramente hemos intentado optimizar las operaciones que realizaba la máquina, y a continuación hemos probado mejorar el acceso a memoria mediante el alineamiento de datos y abusando de la localidad temporal y espacial, y por último hemos reducido la probabilidad de saltos que podría realizar el código. Sin embargo, el tiempo de ejecución seguía siendo relativamente el mismo. Una posible explicación es que mi sistema ya optimice muchas de esas cosas internamente al compilar.

3. 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`

```
22   srand48(time(NULL));
21   //Inicializacion
20   if(N<8){
19     for(int i=0;i<N;i++){
18       y[i] = i;
17       x[i] = N-i;
16     }
15   }else{
14     for(int i=0;i<N;i++){
13       y[i] = drand48();
12       x[i] = drand48();
11     }
10   }
 9
 8   //calculo
 7   int X1, X2;
 6   double a = drand48();
 5
 4   double t_ini = omp_get_wtime();
 3   //Código Daxpy
 2   for(int i=0;i<N;i++)
 1     y[i] = a*x[i] + y[i];
40
 1   double tiempo = omp_get_wtime() - t_ini;
 2
 3   printf("Tiempo(segs.): %11.9f\n", tiempo);
 4
 5 }
ORMAL  daxpy.c  +
ardTimeOff
```

| Tiempos ejec. Longitud vectores=100000 | -O0 | -Os | -O2 | -O3 |
|---|---|---|---|---|
| | *0.000254100* | 0.000058200 | 0.000057600 | 0.000037200 |

**(La captura no coincide con los tiempos de ejecución, ya que estos fueron calculados a continuación con varias ejecuciones para asegurar la fiabilidad de los resultados).**

```
021-05-21-12:12 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ gcc -O0 -fopenmp -o daxpyO0 daxpy.c
021-05-21-12:13 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ gcc -Os -fopenmp -o daxpyOs daxpy.c
021-05-21-12:13 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ gcc -O2 -fopenmp -o daxpyO2 daxpy.c
021-05-21-12:13 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ gcc -O3 -fopenmp -o daxpyO3 daxpy.c
021-05-21-12:13 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ ./daxpyO0 100000
iempo(segs.): 0.000314800
021-05-21-12:13 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ ./daxpyOs 100000
iempo(segs.): 0.000082900
021-05-21-12:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ ./daxpyO2 100000
iempo(segs.): 0.000058300
021-05-21-12:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ ./daxpyO3 100000
iempo(segs.): 0.000176500
021-05-21-12:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ ./daxpyO3 100000
iempo(segs.): 0.000059700
021-05-21-12:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$ ./daxpyO3 100000
iempo(segs.): 0.000043000
021-05-21-12:14 BrianSenaSimons@Ubuntu-20.04:~/Uni2/AC/BP4/ejer3$
```

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

El código generado por -O0 utiliza el puntero de pila para almacenar e ir manipulando los valores a la vez que realiza más saltos incondicionales que los demás. -Os destaca por no utilizar el puntero de pila y utiliza operaciones menos costosas para vaciar los registros como es un "xor" pero sigue fallando en el número de saltos. La opción -O2 se aprovecha de los beneficios de la -Os teniendo en cuenta ahora evitar saltos innecesarios. La -O3 presenta mayor complejidad a nivel de instrucción con intención de aumentar el rendimiento total (Éxito) cumpliendo con lo necesario para ello. (Evitar saltos, multiplicaciones….)

**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

```
14fd: 66 48 0f 7e c0
      movq   %xmm0,%rax
1502: 48 89 45 b8
      mov    %rax,-0x48(%rbp)
1506: c7 45 80 00 00 00 00
      movl   $0x0,-0x80(%rbp)
150d: eb 3e
      jmp    154d <main+0x304>
150f: 48 8b 45 a8
      mov    -0x58(%rbp),%rax
1513: 8b 55 80
      mov    -0x80(%rbp),%edx
1516: 48 63 d2
      movslq %edx,%rdx
1519: f2 0f 10 04 d0
      movsd  (%rax,%rdx,8),%xmm0
151e: 66 0f 28 c8
      movapd %xmm0,%xmm1
1522: f2 0f 59 4d b0
      mulsd  -0x50(%rbp),%xmm1
1527: 48 8b 45 98
      mov    -0x68(%rbp),%rax
152b: 8b 55 80
      mov    -0x80(%rbp),%edx
152e: 48 63 d2
      movslq %edx,%rdx
1531: f2 0f 10 04 d0
      movsd  (%rax,%rdx,8),%xmm0
1536: f2 0f 58 c1
      addsd  %xmm1,%xmm0
153a: 48 8b 45 98
      mov    -0x68(%rbp),%rax
153e: 8b 55 80
      mov    -0x80(%rbp),%edx
1541: 48 63 d2
      movslq %edx,%rdx
1544: f2 0f 11 04 d0
      movsd  %xmm0,(%rax,%rdx,8)
1549: 83 45 80 01
      addl   $0x1,-0x80(%rbp)
154d: 8b 45 80
      mov    -0x80(%rbp),%eax
1550: 3b 45 8c
      cmp    -0x74(%rbp),%eax
1553: 7c ba
      jl     150f <main+0x2c6>
```

daxpy0s.s

```
12ad: f2 0f 10 4d c0
      movsd  -0x40(%rbp),%xmm1
12b2: 31 c0
      xor    %eax,%eax
12b4: f2 0f 11 45 c8
      movsd  %xmm0,-0x38(%rbp)
12b9: 39 c3
      cmp    %eax,%ebx
12bb: 7e 1c
      jle    12d9 <main+0x179>
12bd: f2 41 0f 10 44 c5 00
      movsd 0x0(%r13,%rax,8),%xmm0
12c4: f2 0f 59 c1
      mulsd  %xmm1,%xmm0
12c8: f2 41 0f 58 04 c4
      addsd  (%r12,%rax,8),%xmm0
12ce: f2 41 0f 11 04 c4
      movsd  %xmm0,(%r12,%rax,8)
12d4: 48 ff c0
      inc    %rax
12d7: eb e0
      jmp    12b9 <main+0x159>
```

daxpy02.s

```
1299: f2 0f 10 4d b0
      movsd  -0x50(%rbp),%xmm1
129e: 31 c0
      xor    %eax,%eax
12a0: f2 0f 11 45 b8
      movsd  %xmm0,-0x48(%rbp)
12a5: 0f 1f 00
      nopl   (%rax)
12a8: f2 41 0f 10 44 c5 00
    Movsd 0x0(%r13,%rax,8),%xmm0
12af: f2 0f 59 c1
      mulsd  %xmm1,%xmm0
12b3: f2 0f 58 04 c3
      addsd  (%rbx,%rax,8),%xmm0
12b8: f2 0f 11 04 c3
      movsd  %xmm0,(%rbx,%rax,8)
12bd: 48 83 c0 01
      add    $0x1,%rax
12c1: 41 39 c6
      cmp    %eax,%r14d
12c4: 7f e2
      jg     12a8 <main+0x148>
```

daxpy03.s

```
1370: 45 85 ed
      test   %r13d,%r13d
1373: ba 01 00 00 00
      mov    $0x1,%edx
1378: 41 0f 4f d5
      cmovg  %r13d,%edx
137c: f2 0f 11 45 b8
      movsd  %xmm0,-0x48(%rbp)
1381: 7e 3f
      jle    13c2 <main+0x262>
1383: 41 83 fd 01
      cmp    $0x1,%r13d
1387: 74 39
      je     13c2 <main+0x262>
1389: f2 0f 10 4d b0
      movsd  -0x50(%rbp),%xmm1
138e: d1 ea
      shr    %edx
1390: 31 c0
      xor    %eax,%eax
1392: 48 c1 e2 04
      shl    $0x4,%rdx
1396: 66 0f 14 c9
      unpcklpd %xmm1,%xmm1
139a: 66 0f 1f 44 00 00
      nopw   0x0(%rax,%rax,1)
13a0: 66 41 0f 10 04 06
    movupd (%r14,%rax,1),%xmm0
13a6: 66 41 0f 10 14 04
    movupd (%r12,%rax,1),%xmm2
13ac: 66 0f 59 c1
      mulpd  %xmm1,%xmm0
13b0: 66 0f 58 c2
      addpd  %xmm2,%xmm0
13b4: 41 0f 11 04 04
      movups %xmm0,(%r12,%rax,1)
13b9: 48 83 c0 10
      add    $0x10,%rax
13bd: 48 39 d0
      cmp    %rdx,%rax
13c0: 75 de
      jne    13a0 <main+0x240>
```

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:**~~

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

**~~(b) RESPUESTA~~**