2º curso / 2º cuatr. Grado Ing. Inform.	**Arquitectura de Computadores (AC)** **Cuaderno de prácticas.** **Bloque Práctico 5. Optimización de código** Estudiante (nombre y apellidos): Carmen García Romero Grupo de prácticas y profesor de prácticas: Fecha de entrega: 07/06/21 Fecha evaluación en clase: 08/06/21

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):
Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz

Sistema operativo utilizado: Ubuntu 20.04.2 LTS

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:

```
CarmenGarciaRomero c1estudiante11@carmen:~  2021-06-01 martes
$lscpu
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):                              12
Lista de la(s) CPU(s) en línea:      0-11
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:              6
«Socket(s)»                          1
Modo(s) NUMA:                        1
ID de fabricante:                    GenuineIntel
Familia de CPU:                      6
Modelo:                              158
Nombre del modelo:                   Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
Revisión:                            13
CPU MHz:                             800.034
CPU MHz máx.:                        4500,0000
CPU MHz mín.:                        800,0000
BogoMIPS:                            5199.98
Virtualización:                      VT-x
Caché L1d:                           192 KiB
Caché L1i:                           192 KiB
Caché L2:                            1,5 MiB
Caché L3:                            12 MiB
CPU(s) del nodo NUMA 0:              0-11
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 disabl
                                     ed 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 ac
                                     pi mmx fxsr sse sse2 ss ht tm pbe syscall n
                                     x pdpe1gb rdtscp lm constant_tsc art arch_p
                                     erfmon pebs bts rep_good nopl xtopology non
                                     stop_tsc cpuid aperfmperf pni pclmulqdq dte
                                     s64 monitor ds_cpl vmx est tm2 ssse3 sdbg f
                                     ma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
```

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:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer1  2021-06-03 jueves
$gcc -O2 -fopenmp -o pmm-secuencial pmm-secuencial.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer1  2021-06-03 jueves
$./pmm-secuencial 5

m_res[0][0] = 30.000000
m_res[0][1] = 30.000000
m_res[0][2] = 30.000000
m_res[0][3] = 30.000000
m_res[0][4] = 30.000000
m_res[1][0] = 30.000000
m_res[1][1] = 30.000000
m_res[1][2] = 30.000000
m_res[1][3] = 30.000000
m_res[1][4] = 30.000000
m_res[2][0] = 30.000000
m_res[2][1] = 30.000000
m_res[2][2] = 30.000000
m_res[2][3] = 30.000000
m_res[2][4] = 30.000000
m_res[3][0] = 30.000000
m_res[3][1] = 30.000000
m_res[3][2] = 30.000000
m_res[3][3] = 30.000000
m_res[3][4] = 30.000000
m_res[4][0] = 30.000000
m_res[4][1] = 30.000000
m_res[4][2] = 30.000000
m_res[4][3] = 30.000000
m_res[4][4] = 30.000000
Tiempo(seg):0.000001188
```

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp
4/ejer1  2021-06-07 lunes
$./pmm-secuencial 1000

m_res[0][0] = 252.964697
m_res[999][999] = 255.331221
Tiempo(seg):1.165384312
```

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

pmm-secuencial.c

```
5  int main(int argc, char** argv){
6
7    //Leer argumento de entrada (nÂº de componentes del vector)
8    if (argc<2){
9      printf("Faltan tam de las matrices\n");
10     exit(-1);
11   }
12
13   unsigned int N = atoi(argv[1]);
14   double t_ini, t_fin;
15
16   double val;
17   struct drand48_data randBuffer;
18   srand48_r(time(NULL), &randBuffer);
19
20   double **m1, **m2, **m_res;
21   m1 = (double **) malloc(N*sizeof(double *));
22   m2 = (double **) malloc(N*sizeof(double *));
23   m_res = (double **) malloc(N*sizeof(double *));
24
25   if ((m1 == NULL) || (m2 == NULL) || (m_res == NULL)) {
26     printf("No hay suficiente espacio para las matrices\n");
27     exit(-2);
28   }
29
30   for(int i= 0; i<N; ++i){
31     m1[i] = (double *) malloc(N*sizeof(double));
32     m2[i] = (double *) malloc(N*sizeof(double));
33     m_res[i] = (double *) malloc(N*sizeof(double));
34   }
35
```

pmm-secuencial.c

```
35
36    //Inicialización
37    for (int i = 0; i < N; i++){
38      for(int j=0; j<N; j++){
39        if(N>8){
40          drand48_r(&randBuffer,&val);
41          m1[i][j] = val;
42          drand48_r(&randBuffer,&val);
43          m2[i][j] = val;
44          m_res[i][j]= 0;
45        }
46        else{
47          m1[i][j] = 3.00;
48          m2[i][j] = 2.00;
49          m_res[i][j]= 0;
50        }
51       }
52      }
53
54    t_ini = omp_get_wtime();
55
56    //Calcular multiplicacion de matrices
57    for(int i=0; i<N; i++)
58      for(int j=0; j<N; j++)
59        for(int k=0; k<N; k++)
60          m_res[i][j] += m1[i][k] * m2[k][j];
61
62
63    t_fin = omp_get_wtime() - t_ini;
64
65  //Mostramos resultados
66    for(int i=0; i<N; i++)
67      for(int j=0; j<N; j++)
68        printf("\nm_res[%d][%d] = %f", i,j,m_res[i][j]);
69
70    printf("\nTiempo(seg):%11.9f\t\n" , t_fin);
71
72    //Liberamos espacio
73    for(int i= 0; i<N; ++i){
74      free(m_res[i]);
75      free(m1[i]);
76      free(m2[i]);
77    }
78
79    free(m_res);
80    free(m1);
```

(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-: Separación de las iteraciones en sumas parciales de 4 en 4

Modificación B) –explicación-: Intercambio de los valores j y k, para que los datos estén más cerca en memoria aprovechando la localidad.

A) Captura de pmm-secuencial-modificado_A.c

```
36
37      //Inicialización
38      for (int i = 0; i < N; i++){
39        for(int j=0; j<N; j++){
40          if(N>8){
41            drand48_r(&randBuffer,&val);
42            m1[i][j] = val;
43            drand48_r(&randBuffer,&val);
44            m2[i][j] = val;
45            m_res[i][j]= 0;
46          }
47          else{
48            m1[i][j] = j+1;
49            m2[i][j] = j+1;
50            m_res[i][j]= 0;
51          }
52        }
53      }
54
55      t_ini = omp_get_wtime();
56
57      //Calcular multiplicacion de matrices
58      for(int i=0; i<N; i++){
59        for(int j=0; j<N; j++){
60          s1=s2=s3=s4=0;
61          for(int k=0; k<N/4;k+=4 ){
62            s1 += m1[i][k] * m2[k][j];
63            s2 += m1[i][k+1] * m2[j][k+1];
64            s3 += m1[i][k+2] * m2[j][k+2];
65            s4 += m1[i][k+3] * m2[j][k+3];
66           }
67           total= s1+s2+s3+s4;
68           m_res[i][j]= total;
69          }
70        }
71
```

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

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp
4/ejer1  2021-06-07 lunes
$gcc -O2 -fopenmp -o pmm-secuencial-modificado-a pmm-secuencial-modificado-a.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp
4/ejer1  2021-06-07 lunes
$./pmm-secuencial-modificado-a 10

m_res[0][0] = 0.549348
m_res[9][9] = 0.967757
Tiempo(seg):0.000001704
```

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp
4/ejer1  2021-06-07 lunes
$./pmm-secuencial-modificado-a 1000

m_res[0][0] = 58.399616
m_res[999][999] = 61.690848
Tiempo(seg):0.108559044
```

B) Captura de pmm-secuencial-modificado_b.c

```
56
57      //Calcular multiplicacion de matrices
58      for(int i=0; i<N; i++)
59        for(int k=0; k<N; k++)
60          for(int j=0; j<N;j++ )
61            m_res[i][j] += m1[i][k] * m2[k][j];
62
```

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp
4/ejer1  2021-06-07 lunes
$gcc -O2 -fopenmp -o pmm-secuencial-modificado-b pmm-secuencial-modificado-b.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp
4/ejer1  2021-06-07 lunes
$./pmm-secuencial-modificado-b 1000

m_res[0][0] = 264.960039
m_res[999][999] = 257.390246
Tiempo(seg):0.489421916
```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	*Secuencial*	
Modificación A)	Separación de las iteraciones en sumas parciales de 4 en 4	0.108559044

Modificación B)	Intercambio de los valores j y k	0.489421916
...		

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

Con ambas modificaciones obtenemos mejores resultados que con la implementación secuencial, esto se debe a que con la primera modificación se ha reducido en 4 el número de instrucciones de salto a realizar, y en la segunda se han optimizado los accesos a memoria para aprovechar la localidad.

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

figura1-original.c

```c
struct {
        int a;
        int b;
}
s[MAX];

int main(int argc, char** argv){

  //Leer argumento de entrada
  if (argc<3){
    printf("Faltan parametros M y N\n");
    exit(-1);
  }

  unsigned int M = atoi(argv[1]);
  unsigned int N = atoi(argv[2]);
  double t_ini, t_fin;

  int X1, X2;
  int R[M];

  double val;
  struct drand48_data randBuffer;
  srand48_r(time(NULL), &randBuffer);

  //Inicialización
  for (int i = 0; i < N; i++){
    if(N>8){
      drand48_r(&randBuffer,&val);
      s[i].a = val;
      drand48_r(&randBuffer,&val);
      s[i].b = val;
    }
    else{
      s[i].a = i+1;
      s[i].b = i+2;
    }
   }
```

```
48      t_ini = omp_get_wtime();
49     //Calculo
50     for (int ii=0; ii<M; ii++) {
51       X1=0;
52       X2=0;
53       for(int i=0; i<N; i++)
54         X1+=2*s[i].a+ii;
55
56       for(int i=0; i<N; i++)
57         X2+=3*s[i].b-ii;
58
59        if(X1<X2)
60           R[ii]=X1;
61        else
62           R[ii]=X2;
63       }
64
65       t_fin = omp_get_wtime() - t_ini;
66       printf("Tiempo: %11.9f\n", t_fin);
67       return 0;
68   }
```

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-: He unido los dos bucles for internos en uno solo, ya que eran el mismo bucle, y además he sustituido el if/else por el operador ? , lo que hace una comprobación y asignación más rápida.

Modificación B) –explicación-: Manteniendo las modificaciones anteriores, además he dividido las iteraciones del segundo bucle de 4 en 4.

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c

```
43
44      t_ini = omp_get_wtime();
45     //Calculo
46     for (int ii=0; ii<M; ii++) {
47       X1=0;
48       X2=0;
49       for(int i=0; i<N; i++){
50         X1+=2*s[i].a+ii;
51         X2+=3*s[i].b-ii;
52       }
53
54       R[ii] = (X1 < X2) ? X1: X2;
55       }
56
```

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

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas
/bp4/ejer2  2021-06-03 jueves
$gcc -O2 -fopenmp -o figura1-modificadoA figura1-modificadoA.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas
/bp4/ejer2  2021-06-03 jueves
$./figura1-original 1500 1200
Tiempo: 0.005201391
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas
/bp4/ejer2  2021-06-03 jueves
$./figura1-modificadoA 1500 1200
Tiempo: 0.004275053
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas
/bp4/ejer2  2021-06-03 jueves
$./figura1-original 2000 2000
Tiempo: 0.007530611
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas
/bp4/ejer2  2021-06-03 jueves
$./figura1-modificadoA 2000 2000
Tiempo: 0.006397291
```

B) Captura figura1-modificado_B.c

```
46    //Calculo
47    for (int ii=0; ii<M; ii++) {
48      X1=0;
49      X2=0;
50      for(int i=0; i<N; i+=4){
51        X1+=2*s[i].a+ii;
52        X2+=3*s[i].b-ii;
53        X1+=2*s[i+1].a+ii;
54        X2+=3*s[i+1].b-ii;
55        X1+=2*s[i+2].a+ii;
56        X2+=3*s[i+2].b-ii;
57        X1+=2*s[i+3].a+ii;
58        X2+=3*s[i+3].b-ii;
59      }
60
61      R[ii] = (X1 < X2) ? X1: X2;
62      }
63
```

Capturas de pantalla

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer2  2021-06-03 jueves
$gcc -O2 -fopenmp -o figura1-modificadoB figura1-modificadoB.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer2  2021-06-03 jueves
$./figura1-original 1500 1200
Tiempo: 0.004767869
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer2  2021-06-03 jueves
$./figura1-modificadoB 1500 1200
Tiempo: 0.003918851
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer2  2021-06-03 jueves
$./figura1-original 2000 2000
Tiempo: 0.007009070
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer2  2021-06-03 jueves
$./figura1-modificadoB 2000 2000
Tiempo: 0.004555345
```

Capturas de pantalla

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer2  2021-06-03 jueves
$./figura1-original 2000 2000
Tiempo: 0.007648770
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer2  2021-06-03 jueves
$./figura1-modificadoA 2000 2000
Tiempo: 0.004854034
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp4/ejer2  2021-06-03 jueves
$./figura1-modificadoB 2000 2000
Tiempo: 0.004368174
```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		*0.007648770*
Modificación A)	Unión de bucles for y uso del operador ?	0.004854034
Modificación B)	División de las iteraciones del bucle for interno de 4 en 4	0.004368174
...		

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

La segunda modificación da lugar a una mejora más considerable en el tiempo, ya que se reducen bastantes más instrucciones de salto, que con la eliminación de un bucle, en la primera modificación.

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

daxpy.c

```c
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() 
#include <stdio.h>  // biblioteca donde se encuentra la funciÃ³n 
#include <time.h> // biblioteca donde se encuentra la funciÃ³n cl
#include <omp.h>


int main(int argc, char** argv){

  //Leer argumento de entrada
  if (argc<3){
    printf("Faltan los parametros N y, un numero, a\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);
  unsigned int a = atoi(argv[2]);
  double *x, *y;
  double t_ini, t_final;

  x = (double *) malloc(N*sizeof(double));
  y = (double *) malloc(N*sizeof(double));

  //Inicialización
  for(int i = 0; i<N; ++i){
    x[i]= i+1;
    y[i]=i+2;
  }

  //Calculo
  t_ini = omp_get_wtime();

  for(int i= 0; i< N; i++)
    y[i] = a*x[i] + y[i];


  t_final = omp_get_wtime() - t_ini;
  printf("Tiempo: %11.9f\n", t_final);
  return 0;
}
```

Tiempos ejec. Longitud vectores=1000000	**-O0**	**-Os**	**-O2**	**-O3**
	0.003822727	0.002911871	0.000813784	0.000637443

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

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC
/practicas/bp4/ejer3  2021-06-03 jueves
$gcc -O0 -fopenmp -o daxpy daxpy.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC
/practicas/bp4/ejer3  2021-06-03 jueves
$./daxpy 1000000 1.75
Tiempo: 0.003822727
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC
/practicas/bp4/ejer3  2021-06-03 jueves
$gcc -Os -fopenmp -o daxpy daxpy.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC
/practicas/bp4/ejer3  2021-06-03 jueves
$./daxpy 1000000 1.75
Tiempo: 0.002911871
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC
/practicas/bp4/ejer3  2021-06-03 jueves
$gcc -O2 -fopenmp -o daxpy daxpy.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC
/practicas/bp4/ejer3  2021-06-03 jueves
$./daxpy 1000000 1.75
Tiempo: 0.000813784
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC
/practicas/bp4/ejer3  2021-06-03 jueves
$gcc -O3 -fopenmp -o daxpy daxpy.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC
/practicas/bp4/ejer3  2021-06-03 jueves
$./daxpy 1000000 1.75
Tiempo: 0.000637443
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

- **-O0** es la optimización por defecto que ejecuta gcc, la cual no tiene ningún nivel de optimización. En ella se usan direcciones relativas a la pila.
- **-Os** es el encargado de compilar un código ensamblador lo mas reducido posible, para ello activa todas las opciones que también tiene -O2, sin incrementar el tamaño del código a generar.
- **-O2** es el nivel recomendado de optimización. Con -O2 el compilador intentará aumentar el rendimiento del código sin aumentar demasiado el tamaño del mismo, y el tiempo de compilación. Utiliza registros de la arquitectura para almacenar la información, y así ahorrar muchas operaciones móviles que no son necesarias.
- **-O3** es el mayor nivel de optimización. En él se activan optimizaciones que aumentan el tam del código a la vez que mejoran su velocidad, lo que conlleva que ocupe mayor cantidad de memoria.

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)

daxpyO0.s

```
        call    omp_get_wtime@PLT
        movq    %xmm0, %rax
        movq    %rax, -16(%rbp)
        movl    $0, -44(%rbp)
        jmp     .L5
.L8:
        movl    -36(%rbp), %eax
        testq   %rax, %rax
        js      .L6
        cvtsi2sdq   %rax, %xmm0
        jmp     .L7
.L6:
        movq    %rax, %rdx
        shrq    %rdx
        andl    $1, %eax
        orq     %rax, %rdx
        cvtsi2sdq   %rdx, %xmm0
        addsd   %xmm0, %xmm0
.L7:
        movl    -44(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        movq    -32(%rbp), %rax
        addq    %rdx, %rax
        movsd   (%rax), %xmm1
        mulsd   %xmm0, %xmm1
        movl    -44(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movsd   (%rax), %xmm0
        movl    -44(%rbp), %eax
```

daxpyOs.s

```
.L11:
        call    omp_get_wtime@PLT
        xorl    %edx, %edx
        movsd   %xmm0, 8(%rsp)
.L5:
        cmpl    %edx, %ebx
        jbe     .L12
        cvtsi2sdq   %r12, %xmm0
        mulsd   0(%r13,%rdx,8), %xmm0
        addsd   0(%rbp,%rdx,8), %xmm0
        movsd   %xmm0, 0(%rbp,%rdx,8)
        incq    %rdx
        jmp     .L5
.L12:
        call    omp_get_wtime@PLT
        subsd   8(%rsp), %xmm0
```

daxpyO2.s

```
        jne     .L4
        call    omp_get_wtime@PLT
        movl    %r12d, %r12d
        pxor    %xmm1, %xmm1
        xorl    %edx, %edx
        cvtsi2sdq   %r12, %xmm1
        movsd   %xmm0, 8(%rsp)
        .p2align 4,,10
        .p2align 3
.L6:
        movsd   0(%rbp,%rdx,8), %xmm0
        movq    %rdx, %rax
        mulsd   %xmm1, %xmm0
        addsd   (%rbx,%rdx,8), %xmm0
        movsd   %xmm0, (%rbx,%rdx,8)
        addq    $1, %rdx
        cmpq    %rax, %r13
        jne     .L6
.L9:
        call    omp_get_wtime@PLT
        subsd   8(%rsp), %xmm0
```

```
        cltq
        leaq        0(,%rax,8), %rdx
        movq        -24(%rbp), %rax
        addq        %rdx, %rax
        addsd       %xmm1, %xmm0
        movsd       %xmm0, (%rax)
        addl        $1, -44(%rbp)
.L5:
        movl        -44(%rbp), %eax
        cmpl        %eax, -40(%rbp)
        ja          .L8
        call
omp_get_wtime@PLT
```

daxpyO3.s

```
call    omp_get_wtime@PLT
        movl    %r13d, %eax
        pxor    %xmm1, %xmm1
        cvtsi2sdq       %rax, %xmm1
        movsd   %xmm0, 8(%rsp)
.L17:
        shrl    %ebp
        unpcklpd        %xmm1, %xmm1
        xorl    %edx, %edx
        salq    $4, %rbp
        .p2align 4,,10
        .p2align 3
.L9:
        movupd(%r12,%rdx), %xmm0
        movupd(%rbx,%rdx), %xmm7
        mulpd   %xmm1, %xmm0
        addpd   %xmm7, %xmm0
        movups%xmm0, (%rbx,%rdx)
        addq    $16, %rdx
        cmpq    %rbp, %rdx
        jne     .L9
.L14:
        call    omp_get_wtime@PLT
        subsd   8(%rsp), %xmm0
        movl    $1, %edi
        leaq    .LC5(%rip), %rsi
        movl    $1, %eax
        call    __printf_chk@PLT
        addq    $16, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 48
        xorl    %eax, %eax
        popq    %rbx
        .cfi_def_cfa_offset 40
        popq    %rbp
        .cfi_def_cfa_offset 32
        popq    %r12
        .cfi_def_cfa_offset 24
        popq    %r13
```

```
        .cfi_def_cfa_offset 16
        popq    %r14
        .cfi_def_cfa_offset 8
        ret
.L3:
        .cfi_restore_state
        call    omp_get_wtime@PLT
        movsd   %xmm0, 8(%rsp)
```

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:

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp4
/ejer4  2021-06-07 lunes
$gcc -O2 -fopenmp -o pmm-paralelo pmm-paralelo.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp4
/ejer4  2021-06-07 lunes
$./pmm-paralelo 1000

m_res[0][0] = 250.069466
m_res[999][999] = 240.903697
Tiempo(seg):0.094798214
```

CAPTURA CÓDIGO FUENTE: pmm-paralelo.c

pmm-paralelo.c

```
37
38    //Inicialización
39    for (int i = 0; i < N; i++){
40      for(int j=0; j<N; j++){
41        if(N>8){
42          drand48_r(&randBuffer,&val);
43          m1[i][j] = val;
44          drand48_r(&randBuffer,&val);
45          m2[i][j] = val;
46          m_res[i][j]= 0;
47        }
48        else{
49          m1[i][j] = j+1;
50          m2[i][j] = j+1;
51          m_res[i][j]= 0;
52        }
53       }
54      }
55
56    t_ini = omp_get_wtime();
57
58    //Calcular multiplicacion de matrices
59    #pragma omp parallel for
60      for(int i=0; i<N; i++)
61        for(int k=0; k<N; k++)
62          for(int j=0; j<N;j++ )
63            m_res[i][j] += m1[i][k] * m2[k][j];
64
65
66
67    t_fin = omp_get_wtime() - t_ini;
68
69    //Mostramos resultados
70    if(N<8){
71      for(int i=0; i<N; i++)
72         for(int j=0; j<N; j++)
73            printf("\nm_res[%d][%d] = %f", i,j,m_res[i][j]);
74    }
75    else{
76      printf("\nm_res[%d][%d] = %f", 0,0,m_res[0][0]);
77      printf("\nm_res[%d][%d] = %f", N-1,N-1,m_res[N-1][N-1]);
78    }
```

(b) RESPUESTA