2º curso / 2º cuatr. Grado Ing. Inform.	**Arquitectura de Computadores (AC)** **Cuaderno de prácticas.** **Bloque Práctico 5. Optimización de código** Estudiante (nombre y apellidos): Grupo de prácticas y profesor de prácticas: 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):
Intel® Core™ i7-3540M CPU @ 3.00GHz × 4

Sistema operativo utilizado: *UBUNTU 18.04.4*

Versión de gcc utilizada: gcc version 7.5.0

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

```
amador@MacBookPro:~/Escritorio$ lscpu
Arquitectura:                        x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:                  Little Endian
CPU(s):                              4
Lista de la(s) CPU(s) en línea:      0-3
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:              2
«Socket(s)»                          1
Modo(s) NUMA:                        1
ID de fabricante:                    GenuineIntel
Familia de CPU:                      6
Modelo:                              58
Nombre del modelo:                   Intel(R) Core(TM) i7-3540M CPU @
 3.00GHz
Revisión:                            9
CPU MHz:                             3392.091
CPU MHz máx.:                        3700,0000
CPU MHz mín.:                        1200,0000
BogoMIPS:                            5986.13
Virtualización:                      VT-x
Caché L1d:                           32K
Caché L1i:                           32K
Caché L2:                            256K
Caché L3:                            4096K
CPU(s) del nodo NUMA 0:              0-3
Indicadores:                         fpu vme de pse tsc msr pae mce c
x8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse
 sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon peb
s bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmul
qdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid s
se4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdra
nd lahf_lm cpuid_fault pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexp
riority ept vpid fsgsbase smep erms xsaveopt dtherm ida arat pln pts
md_clear flush_l1d
amador@MacBookPro:~/Escritorio$ cat /proc/version
Linux version 5.4.0-74-generic (buildd@lcy01-amd64-023) (gcc version
7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #83~18.04.1-Ubuntu SMP Tue May 1
1 16:01:00 UTC 2021
```

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 <omp.h>
#include <stdio.h>
#include <time.h>

#define TAM 1000
double m1[TAM][TAM],m2[TAM][TAM],m3[TAM][TAM];
//funciones que voy a utilizar
void secuencial(double m1[][TAM],double m2[][TAM],double m3[][TAM], int N);
void modificadoA(double m1[][TAM],double m2[][TAM],double m3[][TAM], int N);
void modificadoB(double m1[][TAM],double m2[][TAM],double m3[][TAM], int N);
void borrarMatriz(double m3[][TAM],int N);

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

//comprobamos argumentos
    if(argc<2){
        printf("Falta el tamaño de las matrices");
        exit(-1);
    }
//almacenar el tamaño de la matriz
    unsigned int N=atoi(argv[1]);
    if(N>TAM){
        N=TAM;
    }
//creacion de las variables que miden el tiempo
    struct timespec ctgsec1,ctgsec2,ctga1,ctga2,ctgb1,ctgb2;
    double nctgsec,nctga,nctgb;
//Rellenamos las matrices
    for(int i=0;i<N;i++){
        for(int j=0; j<N; j++){
            m1[i][j]=drand48();
            m2[i][j]=drand48();
        }
    }
//programa secuencial
    borrarMatriz(m3,N);
    clock_gettime(CLOCK_REALTIME,&ctgsec1);
    secuencial(m1,m2,m3,N);
    clock_gettime(CLOCK_REALTIME,&ctgsec2);
    nctgsec=(double)(ctgsec2.tv_sec-ctgsec1.tv_sec)+(double)((ctgsec2.tv_nsec-ctgsec1.tv_nsec)/(1.e+9));
    printf("\nTiempo secuencial: %11.9f | Tamaño: %d | m3[0][0]=%f | m3[%d][%d]=%f ",nctgsec,N,m3[0][0],N-1,N-1,m3[N-1][N-1]);
//programa modificadoA
    borrarMatriz(m3,N);
    clock_gettime(CLOCK_REALTIME,&ctga1);
    modificadoA(m1,m2,m3,N);
    clock_gettime(CLOCK_REALTIME,&ctga2);
    nctga=(double)(ctga2.tv_sec-ctga1.tv_sec)+(double)((ctga2.tv_nsec-ctga1.tv_nsec)/(1.e+9));
    printf("\nTiempo modificadoA: %11.9f | Tamaño: %d | m3[0][0]=%f | m3[%d][%d]=%f",nctga,N,m3[0][0],N-1,N-1,m3[N-1][N-1]);

//programa modificadoB
    borrarMatriz(m3,N);
    clock_gettime(CLOCK_REALTIME,&ctgb1);
    modificadoB(m1,m2,m3,N);
    clock_gettime(CLOCK_REALTIME,&ctgb2);
    nctgb=(double)(ctgb2.tv_sec-ctgb1.tv_sec)+(double)((ctgb2.tv_nsec-ctgb1.tv_nsec)/(1.e+9));
    printf("\nTiempo modificadoB: %11.9f | Tamaño: %d | m3[0][0]=%f | m3[%d][%d]=%f",nctgb,N,m3[0][0],N-1,N-1,m3[N-1][N-1]);b
}
```

```c
void secuencial(double m1[][TAM],double m2[][TAM],double m3[][TAM], int N){
    for(int i=0; i<N; i++){
        for(int j=0;j<N; j++){
            for(int k=0;k<N; k++){
                m3[i][j]+=m1[i][k]*m2[k][j];
            }
        }
    }
}
```

(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, he utilizado el desenrollado de bucles para romper secuencias de instrucciones dependientes intercalando otras instrucciones. Con ello, se consigue:

-Reducir el número de saltos, y asi aumentar la oportunidad de encontrar instrucciones independientes.

-Facilitar la posibilidad de insertar instrucciones para ocultar las latencias.

Sin embargo, hemos aumentado el tamaño del código. Se ha desenrollado el bucle interno para que contenga 10 operaciones, iterando de 10 en 10. El resultado de cada operación se almacena en una variable temporal. Cuando termina el bucle interno, la suma de dichas variables se asigna a la posición i,j correspondiente de la matriz.

Modificación B) –explicación-: INTERCAMBIO DE BUCLES, he utilizado el intercambio de bucles para aprovechar la localidad cambiando así la forma de acceder a los datos según los almacena el compilador. Conseguimos acceder a posiciones más cercanas un número mayor de veces.

CÓDIGOS FUENTE MODIFICACIONES

A) Captura de pmm-secuencial-modificado_A.c

```
void modificadoA(double m1[][TAM],double m2[][TAM],double m3[][TAM], int N){
    double t0,t1,t2,t3,t4,t5,t6,t7,t8,t9;
    for(int i=0; i<N; i++){
        for(int j=0;j<N; j++){
            t0=0;t1=0;t2=0;t3=0;t4=0;t5=0;t6=0;t7=0;t8=0;t9=0;
            for(int k=0;k<N; k+=10){
                t0+=m1[i][k]*m2[k][j];
                t1+=m1[i][k+1]*m2[k+1][j];
                t2+=m1[i][k+2]*m2[k+2][j];
                t3+=m1[i][k+3]*m2[k+3][j];
                t4+=m1[i][k+4]*m2[k+4][j];
                t5+=m1[i][k+5]*m2[k+5][j];
                t6+=m1[i][k+6]*m2[k+6][j];
                t7+=m1[i][k+7]*m2[k+7][j];
                t8+=m1[i][k+8]*m2[k+8][j];
                t9+=m1[i][k+9]*m2[k+9][j];
            }
            m3[i][j]=t0+t1+t2+t3+t4+t5+t6+t7+t8+t9;
        }
    }
}
```

B) …

```
void modificadoB(double m1[][TAM],double m2[][TAM],double m3[][TAM], int N){
    for(int i=0; i<N; i++){
        for(int k=0;k<N; k++){
            for(int j=0;j<N; j++){
                m3[i][j]+=m1[i][k]*m2[k][j];
            }
        }
    }
}
```

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

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer1] 2021-06-13 domingo
$gcc -o pmm pmm-secuencial.c
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer1] 2021-06-13 domingo
$./pmm 5

Tiempo secuencial: 0.000001218 | Tamaño: 5 | m3[0][0]=0.242619 | m3[4][4]=1.319711
Tiempo modificadoA: 0.000005641 | Tamaño: 5 | m3[0][0]=0.242619 | m3[4][4]=1.319711
Tiempo modificadoB: 0.000000748 | Tamaño: 5 | m3[0][0]=0.242619 | m3[4][4]=1.319711[
AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer1] 2021-06-13 domingo
$./pmm 1000

Tiempo secuencial: 12.665535052 | Tamaño: 1000 | m3[0][0]=261.400093 | m3[999][999]=
256.234227
Tiempo modificadoA: 14.252077324 | Tamaño: 1000 | m3[0][0]=261.400093 | m3[999][999]
=256.234227
Tiempo modificadoB: 7.240382047 | Tamaño: 1000 | m3[0][0]=261.400093 | m3[999][999]=
256.234227[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer1] 2021-06-
13 domingo
$./pmm 1000

Tiempo secuencial: 20.386934157 | Tamaño: 1000 | m3[0][0]=261.400093 | m3[999][999]=
256.234227
Tiempo modificadoA: 13.844276926 | Tamaño: 1000 | m3[0][0]=261.400093 | m3[999][999]
=256.234227
Tiempo modificadoB: 7.603478297 | Tamaño: 1000 | m3[0][0]=261.400093 | m3[999][999]=
256.234227[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer1] 2021-06-
13 domingo
$
```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		*20.38s*
Modificación A)	Desenrrollado de bucles	13.84s
Modificación B)	Intercambio de bucles	7.60s
...		

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

En la primera ejecución de N=1000 el tiempo de ejecución de la modificación A es mayor que la secuencial seguramente debido a que el sistema estaría más cargado durante su ejecución, por ello he decidido ejecutarlo una segunda vez y ahí observamos que mejora tanto en la modificacionA como en la modificacionB al igual que en la ejecución de N=5.

En el desenrollado de bucle se mejora el tiempo obtenido con respecto al algoritmo sin modificar. Esto es debido a que se rompen secuencias de instrucciones dependientes. Sin embargo, el tiempo del desenrollado del bucle es mayor que el del intercambio de bucles, casi 3 veces mayor. La modificación del intercambio de bucles nos permite optimizar mucho el algoritmo porque se realizan muchos accesos a las matrices, que es lo que logramos hacer más rápido aprovechando 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

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-: Reducción de los saltos por el condicional,hemos realizado un cambio en el condicional para intentar reducir el número de saltos obteniendo, en su lugar, una instrucción con predicado.

Modificación B) –explicación-: Desenrollado del bucle, hemos utilizado el desenrollado de bucles para romper secuencias de instrucciones dependientes intercalando otras instrucciones como hemos utilizado en el ejercicio anterior.

...

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c

```
125  void modificadoA(int R[]){
126      int X1, X2;
127      for(int ii=0; ii<TAM; ii++){
128          X1=0;
129          X2=0;
130          for(int i=0;i<TAM1; i++){
131              X1+=2*s.a[i]+ii;
132          }
133          for(int i=0;i<TAM1; i++){
134              X2+=3*s.b[i]+ii;
135          }
136          R[i]=(X1<X2) ? X1:X2;
137      }
138  }
```

```
91  void modificadoB(int R[]){
92      int X1, X2;
93      for(int ii=0; ii<TAM; ii++){
94          X1=0;
95          X2=0;
96          for(int ii=0;ii<TAM1; ii+=10){
97              X1+=2*s.a[i]+ii;
98              X2+=3*s.b[i]+ii;
99              X1+=2*s.a[i+1]+ii;
100             X2+=3*s.b[i+1]+ii;
101             X1+=2*s.a[i+2]+ii;
102             X2+=3*s.b[i+2]+ii;
103             X1+=2*s.a[i+3]+ii;
104             X2+=3*s.b[i+3]+ii;
105             X1+=2*s.a[i+4]+ii;
106             X2+=3*s.b[i+4]+ii;
107             X1+=2*s.a[i+5]+ii;
108             X2+=3*s.b[i+5]+ii;
109             X1+=2*s.a[i+6]+ii;
110             X2+=3*s.b[i+6]+ii;
111             X1+=2*s.a[i+7]+ii;
112             X2+=3*s.b[i+7]+ii;
113             X1+=2*s.a[i+8]+ii;
114             X2+=3*s.b[i+8]+ii;
115             X1+=2*s.a[i+9]+ii;
116             X2+=3*s.b[i+9]+ii;
117         }
118         if(X1<X2){
119             R[i]=X1;
120         }else{
121             R[i]=X2;
122         }
123     }
124 }
```

B)

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

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer2] 2021-06-13 domingo
$./figura1
Tiempo secuencial:      0.260054830      R[0] = 0        R[39999] = -199995000
Tiempo modificado a:    0.238694068      R[0] = 0        R[39999] = -199995000
Tiempo modificado b:    0.135919683      R[0] = 0        R[39999] = -199995000
```

TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar		*0,26*
Modificación A)	Desenrollado de bucles	0,23
Modificación B)	Ahorro de saltos por la condición	0,13
...		

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

Observamos que el desenrollado del bucle mejora el tiempo obtenido con respecto al algoritmo sin modificar. Es casi el doble de rápido y el más rápido de todos. Esto es debido a que se rompen secuencias de instrucciones dependientes. Con respecto a la modificación del if sustituido, como se ha mencionado anteriormente, vemos que no se produce ninguna diferencia (diferencia mínima porque son tiempos muy pequeños y pueden variar ligeramente de una ejecución a otra).

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*** *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 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

```
11
12  int main( int argc, char * argv[]){
13
14  //comprobamos argumentos
15      if(argc<2){
16          printf("Falta el tamaño del vector");
17          exit(-1);
18      }
19  //almacenar el tamaño de la matriz
20      unsigned int N=atoi(argv[1]);
21      if(N>TAM){
22          N=TAM;
23      }
24  //creacion de las variables que miden el tiempo
25      struct timespec ctgsec1,ctgsec2,ctga1,ctga2,ctgb1,ctgb2;
26      double nctgsec,nctga,nctgb;
27  //asignamos valores
28      const double a = 1.5;
29      for (int i = 0; i < TAM; i++) {
30          x[i] = (rand() % 15) * 2;
31          y[i] = (rand() % 23) * 3;
32      }
33  //programa secuencial
34      //borrarMatriz(m3,N);
35      clock_gettime(CLOCK_REALTIME,&ctgsec1);
36      daxpy(x,y,a,N);
37      clock_gettime(CLOCK_REALTIME,&ctgsec2);
38      nctgsec=(double)(ctgsec2.tv_sec-ctgsec1.tv_sec)+(double)((ctgsec2.tv_nsec-ctgsec1.tv_nsec)/(1.e+9));
39      printf("\nTiempo secuencial: %11.9f | Tamaño: %d | y[%d] = %f\t | y[%d] = %f\n", nctgsec, 0, y[0], N-1, y[N-1]);
40  /*//programa modificadoA
41      borrarMatriz(m3,N);
42      clock_gettime(CLOCK_REALTIME,&ctga1);
43      modificadoA(m1,m2,m3,N);
44      clock_gettime(CLOCK_REALTIME,&ctga2);
45      nctga=(double)(ctga2.tv_sec-ctga1.tv_sec)+(double)((ctga2.tv_nsec-ctga1.tv_nsec)/(1.e+9));
46      printf("\nTiempo modificadoA: %11.9f | Tamaño: %d | m3[0][0]=%f | m3[%d][%d]=%f",nctga,N,m3[0][0],N-1,N-1,m3[N-1][N-1]);
47
48  //programa modificadoB
49      borrarMatriz(m3,N);
50      clock_gettime(CLOCK_REALTIME,&ctgb1);
51      modificadoB(m1,m2,m3,N);
52      clock_gettime(CLOCK_REALTIME,&ctgb2);
53      nctgb=(double)(ctgb2.tv_sec-ctgb1.tv_sec)+(double)((ctgb2.tv_nsec-ctgb1.tv_nsec)/(1.e+9));
54      printf("\nTiempo modificadoB: %11.9f | Tamaño: %d | m3[0][0]=%f | m3[%d][%d]=%f",nctgb,N,m3[0][0],N-1,N-1,m3[N-1][N-1]);**/
55
56  }
57
58
59
60
61  void daxpy (double x[], double y[], const double a, unsigned int N) {
62      for (unsigned int i = 0; i < N; i++)
63          y[i] = a*x[i] + y[i];
64  }
65
```

Tiempos ejec. Longitud vectores=XXXX	-O0	-Os	-O2	-O3
	0.00000291 3	0.0000021 74	0.0000105 01	0.0000017 36

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

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$./daxpy 1000

Tiempo secuencial: 0.000363384 | Tamaño: 0 y[999] = 39.000000    y[24] = 33.000000
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$
```

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$gcc -O3 -o daxpy_3 daxpy.c
daxpy.c: In function ‘main’:
daxpy.c:39:57: warning: format ‘%d’ expects argument of type ‘int’, but argument 4 has type ‘doubl
```

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$gcc -O2 -o daxpy_2 daxpy.c
daxpy.c: In function ‘main’:
daxpy.c:39:57: warning: format ‘%d’ expects argument of type ‘int’, but argument 4 has type ‘doubl
```

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$gcc -Os -o daxpy_s daxpy.c
daxpy.c: In function ‘main’:
daxpy.c:39:57: warning: format ‘%d’ expects argument of type ‘int’, but argument 4 has type ‘doubl
e’ [-Wformat=]
```

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$gcc -O0 -o daxpy_0 daxpy.c
daxpy.c: In function ‘main’:
daxpy.c:39:57: warning: format ‘%d’ expects argument of type ‘int’, but argument 4 has type ‘doubl
e’ [-Wformat=]
  printf("\nTiempo secuencial: %11.9f | Tamaño: %d | y[%d] = %f\t | y[%d] = %f\n", nctgsec, 0, y[0
```

```
daxpy.c:39:78: warning: format ‘%f’ expects a matching ‘double’ argument [-Wformat=]
  printf("\nTiempo secuencial: %11.9f | Tamaño: %d | y[%d] = %f\t | y[%d] = %f\n", nctgsec, 0, y[0
], N-1, y[N-1]);
                                                                              ~^
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$./daxpy_0 1000

Tiempo secuencial: 0.000002913 | Tamaño: 0 | y[999] = 39.000000  | y[24] = 33.000000
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$./daxpy_s 1000

Tiempo secuencial: 0.000002174 | Tamaño: 0 | y[999] = 39.000000  | y[1623620205] = 33.000000
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$./daxpy_2 1000

Tiempo secuencial: 0.000010501 | Tamaño: 0 | y[999] = 39.000000  | y[1623620209] = 33.000000
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$./daxpy_3 1000

Tiempo secuencial: 0.000001736 | Tamaño: 0 | y[999] = 39.000000  | y[1623620213] = 33.000000
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP4/ejer3] 2021-06-13 domingo
$
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

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	daxpyOs.s	daxpyO2.s	daxpyO3.s

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

```
114    void paralelo(double m1[][TAM],double m2[][TAM],double m3[][TAM], int N){
115        //#pragma omp parallel for
116        for(int i=0; i<N; i++){
117            #pragma omp parallel for
118            for(int k=0;k<N; k++){
119                for(int j=0;j<N; j++){
120                    m3[i][j]+=m1[i][k]*m2[k][j];
121                }
122            }
123        }
124    }
```

(b) RESPUESTA

```
srun: error: atcgrid4: task 0: Terminated
[b1estudiante4@atcgrid ejer4]$ srun -p ac4 -A ac ./pmm 2000

Tiempo secuencial: 13.925240066 | Tamaño: 2000 | m3[0][0]=492.399304 | m3[1999][
1999]=512.803629
Tiempo modificadoA: 9.871322192 | Tamaño: 2000 | m3[0][0]=492.399304 | m3[1999][
1999]=512.803629
Tiempo modificadoB: 6.543354380 | Tamaño: 2000 | m3[0][0]=492.399304 | m3[1999][
1999]=512.803629
Tiempo paralelo: 6.537335307 | Tamaño: 2000 | m3[0][0]=492.399304 | m3[1999][199
9]=512.803629[b1estudiante4@atcgrid ejer4]$
```

Tengo el mismo error que en la practica anterior y cuando lo ejecuto ya que me sale el mismo tiempo que el modificado B supongo que no se utilizan todos los cores del atc para ejecutar y no puedo cambiar el numthread.