2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Miguel Ángel Campos Cubillas
Grupo de prácticas y profesor de prácticas: B3
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): AMD Ryzen 5 2600 Six-Core Processor

Sistema operativo utilizado: *Ubuntu 20.04*

Versión de gcc utilizada: *gcc (Ubuntu 9.3.0-10ubuntu2) 9.3.0*

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

```
leonbjj@DESKTOP-5CE60HG:~$ lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
Address sizes:           36 bits physical, 48 bits virtual
CPU(s):                  12
On-line CPU(s) list:     0-11
Thread(s) per core:      2
Core(s) per socket:      6
Socket(s):               1
Vendor ID:               AuthenticAMD
CPU family:              23
Model:                   8
Model name:              AMD Ryzen 5 2600 Six-Core Processor
Stepping:                2
CPU MHz:                 3400.000
CPU max MHz:             3400.0000
BogoMIPS:                6800.00
Hypervisor vendor:       Windows Subsystem for Linux
Virtualization type:     container
Flags:                   fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 h
                         t syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm pni pclmulqdq ssse3 fma cx16 sse4_1 sse4_2 movbe pop
                         cnt aes xsave osxsave avx f16c rdrand hypervisor lahf_lm cmp_legacy cr8_legacy abm sse4a misalignss
                         e 3dnowprefetch osvw wdt topoext fsgsbase bmi1 avx2 smep bmi2 rdseed adx smap clflushopt sha_ni
```

1. Para el núcleo que se muestra en el Figura 1, y para un programa que implemente la multiplicación de matrices con datos flotantes en doble precisión (use variables globales):

   1.1Modifique el código C para reducir el tiempo de ejecución (evalúe el tiempo y modifique sólo el trozo que hace la multiplicación y el trozo que se muestra en la Figura 1). Justifique los tiempos obtenidos (use –O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

   1.2Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.

   1.3(Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

Figura 1 . Código C++ que suma dos vectores

```
struct {
```

```
    int a;
    int b;
} s[5000];

main()
{
  …
  for (ii=0; ii<40000;ii++) {
    X1=0; X2=0;
    for(i=0; i<5000;i++)  X1+=2*s[i].a+ii;
    for(i=0; i<5000;i++)  X2+=3*s[i].b-ii;

    if (X1<X2)  R[ii]=X1  else  R[ii]=X2;
  }
  …
}
```

A) MULTIPLICACIÓN DE MATRICES:

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

```c
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h>  // biblioteca que incluye func. printf()
#include <time.h>   // biblioteca que incluye funcionalidad para controlar tiempo de ejecución

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

    int i,j,k;
    struct timespec cgt1, cgt2; double ncgt; // Tiempo de ejecución

    // Leer argumento de entrada (nº de componentes del vector)
    if (argc < 2){
        printf("Faltan nº componentes del vector \n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N = 2^32-1 = 4294967295 (sizeof(unsigned int) = 4B)

    int **mr, **m1, **m2;
    mr = (int **) malloc(N*sizeof(int *));
    m1 = (int **) malloc(N*sizeof(int *));
    m2 = (int **) malloc(N*sizeof(int *));

    // Comprobamos que no se haya cometido algún error en la reserva de espacio
    if (mr == NULL || m1 == NULL || m2 == NULL){
        printf("Error en la reserva de espacio para las matrices\n");
        exit(-2);
    }

    for(i=0; i<N; i++){
        mr[i] = (int *) malloc(N*sizeof(int));
        m1[i] = (int *) malloc(N*sizeof(int));
        m2[i] = (int *) malloc(N*sizeof(int));
    }

    // inicializamos las matrices
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            mr[i][j] = 0;
            m1[i][j] = 28+i;
            m2[i][j] = 2+i;
        }
    }

    // tiempo inicial
    clock_gettime(CLOCK_REALTIME,&cgt1); // tiempo ini

```

```
47      // Calculamos la multiplicación de la matriz por el vector
48      for (i=0; i<N; i++){
49          for (j=0; j<N; j++){
50              for(k=0; k<N; k++){
51                  mr[i][j] += m1[i][k] * m2[k][j];
52              }
53          }
54      }
55
56      clock_gettime(CLOCK_REALTIME,&cgt2); // tiempo fin
57
58      // Guardamos en la variable ncgt el tiempo que ha tardado en calcular la multiplicación
59      ncgt = (double)(cgt2.tv_sec-cgt1.tv_sec)+(double)((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
60
61      // Visualizamos la primera y la última línea de la matriz mr
62      printf("Tamaño= %d\t Tiempo= %11.9f\t Primera línea= %d\t Última línea= %d\n",N,ncgt,mr[0][0],mr[N-1][N-1]);
63
64      for (i=0; i<N; i++){
65          free(mr[i]);
66          free(m1[i]);
67          free(m2[i]);
68      }
69
70          free(mr);
71          free(m1);
72          free(m2);
73
74          return 0;
75  }
```

Ejecución normal:

```
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ gcc -lrt -O2 -o pmm-secuencial pmm-secuencial.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./pmm-secuencial 15
Tamaño= 15       Tiempo= 0.000002500     Primera línea= 3780     Última línea= 5670
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-:

He modificado el bucle en el que calculamos la multiplicación. El cambio reside en el segundo for, pues este presenta la iteración j a cambio de la iteración determinada por k, en el tercero, la k se cambia por la j. Este cambio se debe a que los datos de esta manera son fieles a la proximidad que existe en memoria.

Modificación b) –explicación-:

En la última modificación, desenrollamos el bucle k en 8-iteraciones, pues es la manera más eficiente de evitar saltos en el código y no repercutir de manera masiva en el incremento de instrucciones máquina como contrapartida.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura de pmm-secuencial-modificado_a.c

```
47      // Calculamos la multiplicación de la matriz por el vector
48      for (i=0; i<N; i++){
49          for (k=0; k<N; k++){
50              for(j=0; j<N; j++){
51                  mr[i][j] += m1[i][k] * m2[k][j];
52              }
53          }
54      }
```

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

```
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ gcc -lrt -O2 -o pmm-secuencial-modificado_a pmm-secuencial-modificado_a.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./pmm-secuencial-modificado_a 15
Tamaño= 15       Tiempo= 0.000002600     Primera línea= 3780     Última línea= 5670
```

b) Captura de pmm-secuencial-modificado_b.c

```
50      // Calculamos la multiplicación de la matriz por el vector
51      for(i=0; i<N; i++){
52          for(j=0; j<N; j++){
53              v1 = v2 = v3 = v4 = v5 = v6 = v7 = v8 = 0;
54              for(k=0, n=0; k<iter; k++, n+=8){
55                  v1 += (m1[i][n]*m2[j][n]);
56                  v2 += (m1[i][n+1]*m2[j][n+1]);
57                  v3 += (m1[i][n+2]*m2[j][n+2]);
58                  v4 += (m1[i][n+3]*m2[j][n+3]);
59                  v5 += (m1[i][n+4]*m2[j][n+4]);
60                  v6 += (m1[i][n+5]*m2[j][n+5]);
61                  v7 += (m1[i][n+6]*m2[j][n+6]);
62                  v8 += (m1[i][n+7]*m2[j][n+7]);
63              }
64              v0 = v1 + v2 + v3 + v4 + v5 + v6 + v7 + v8;
65              mr[i][j] = v0;
66
67              for(n = iter*8; n<N; n++){
68                  v0 += (m1[i][n]*m2[j][n]);
69              }
70              mr[i][j] = v0;
71          }
72      }
```

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

```
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ gcc -lrt -O2 -o pmm-secuencial-modificado_b pmm-secuencial-modificado_b.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./pmm-secuencial-modificado_b 15
Tamaño= 15      Tiempo= 0.000002600     Primera línea= 840      Última línea= 10080
```

1.1. TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar	SIN CAMBIOS	9.2236
Modificación a)	Cambio en los bucles que calculan la multiplicación. El segundo bucle anidado se recorre con la k y el tercer bucle anidado con la j.	1.6956
Modificación b)	Desenrollado del bucle k en 8 iteraciones	1.5226

```
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ gcc -lrt -O2 -o pmm-secuencial pmm-secuencial.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ gcc -lrt -O2 -o pmm-secuencial-modificado_a pmm-secuencial-modificado_a.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ gcc -lrt -O2 -o pmm-secuencial-modificado_b pmm-secuencial-modificado_b.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./pmm-secuencial 1500
Tamaño= 1500    Tiempo= 9.223566100     Primera línea= 31563000         Última línea= 1721310750
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./pmm-secuencial-modificado_a 1500
Tamaño= 1500    Tiempo= 1.695990900     Primera línea= 31563000         Última línea= 1721310750
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./pmm-secuencial-modificado_b 1500
Tamaño= 1500    Tiempo= 1.522624300     Primera línea= 84000    Última línea= -856926796
```

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

Según los tiempos obtenidos, podemos determinar que tras las modificaciones mejoramos considerablemente el tiempo de ejecución consumido para dar el resultado de la multiplicación de matrices. La modificación a) es 5 y ½ veces más veloz para calcular el resultado. La modificación b) es prácticamente 1/3 más veloz que la modificación a. Estas aproximaciones se deben a ese cálculo, sin embargo podrían variar en sucesivas ejecuciones en función de procesos en segundo plano y tamaño N de componentes. Sin embargo, podemos determinar que el desenrollado de bucles resulta bastante más veloz en este caso, es más óptimo que el secuencial.

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: figura1-original.c

```c
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>

struct{
    int a;
    int b;
} s[500];

int main (int argc, char **argv){
    int ii, i, X1, X2;
    int R[40000];

    struct timespec cgt1, cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(ii=0; ii<40000; ii++){
        X1 = 0;
        X2 = 0;
        for(i=0; i<5000; i++) X1+=2*s[i].a+ii;
        for(i=0; i<5000; i++) X2+=3*s[i].b-ii;

        if(X1<X2) R[ii]=X1;else R[ii]=X2;
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    printf("Tiempo= %11.9f\t R[0] = %i\t R[39999]= %i\n",ncgt,R[0],R[39999]);

    return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Los dos for que se encuentran en el for principal, los he reemplazado por un solo for.

Modificación b) –explicación-: He desenrollado el bucle de la modificación anterior.

Modificación c) –explicación-: He desenrollado ambos bucles independientes de la figura1-original en 4 iteraciones cada uno.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1-modificado_a.c

```c
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>

struct{
    int a;
    int b;
} s[500];

int main (int argc, char **argv){
    int ii, i, X1, X2;
    int R[40000];

    struct timespec cgt1, cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(ii=0; ii<40000; ii++){
        X1 = 0;
        X2 = 0;
        for(i=0; i<5000; i++){
            X1+=2*s[i].a+ii;
            X2+=3*s[i].b-ii;
        }

        if(X1<X2) R[ii]=X1;else R[ii]=X2;
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    printf("Tiempo= %11.9f\t R[0] = %i\t R[39999]= %i\n",ncgt,R[0],R[39999]);

    return 0;
}
```

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

b) Captura figura1-modificado_b.c

```c
struct{
    int a;
    int b;
} s[500];

int main (int argc, char **argv){
    int ii, i, X1, X2;
    int R[40000];

    struct timespec cgt1, cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(ii=0; ii<40000; ii++){
        X1 = 0;
        X2 = 0;
        for(i=0; i<5000; i+=8){
            X1 += 2*s[i].a+ii;
            X2 += 3*s[i].b-ii;
            X1 += 2*s[i+1].a+ii;
            X2 += 3*s[i+1].b-ii;
            X1 += 2*s[i+2].a+ii;
            X2 += 3*s[i+2].b-ii;
            X1 += 2*s[i+3].a+ii;
            X2 += 3*s[i+3].b-ii;
            X1 += 2*s[i+4].a+ii;
            X2 += 3*s[i+4].b-ii;
            X1 += 2*s[i+5].a+ii;
            X2 += 3*s[i+5].b-ii;
            X1 += 2*s[i+6].a+ii;
            X2 += 3*s[i+6].b-ii;
            X1 += 2*s[i+7].a+ii;
            X2 += 3*s[i+7].b-ii;
        }

        if(X1<X2) R[ii]=X1;else R[ii]=X2;
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    printf("Tiempo= %11.9f\t R[0] = %i\t R[39999]= %i\n",ncgt,R[0],R[39999]);

    return 0;
}
```

c) Captura figura1-modificado_c.c

```c
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>

struct{
    int a;
    int b;
} s[500];

int main (int argc, char **argv){
    int ii, i, X1, X2;
    int R[40000];

    struct timespec cgt1, cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(ii=0; ii<40000; ii++){
        X1 = 0;
        X2 = 0;
        for(i=0; i<5000; i++){
            X1+=2*s[i].a+ii;
            X1+=2*s[i+1].a+ii;
            X1+=2*s[i+2].a+ii;
            X1+=2*s[i+3].a+ii;
        }
        for(i=0; i<5000; i++){
            X2+=3*s[i].b-ii;
            X2+=3*s[i+1].b-ii;
            X2+=3*s[i+2].b-ii;
            X2+=3*s[i+3].b-ii;
        }

        if(X1<X2) R[ii]=X1;else R[ii]=X2;
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    printf("Tiempo= %11.9f\t R[0] = %i\t R[39999]= %i\n",ncgt,R[0],R[39999]);

    return 0;
}
```

1.1. TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar	SIN MODIFICACIÓN	0.2031
Modificación a)	He reemplazado los for internos al for principal por un simple for	0.1505
Modificación b)	He desenrollado el bucle for obtenido en la modificación a) en 8 operaciones	0.1219
Modificación c)	He desenrollado los dos bucles internos de la versión original.	0.1930

```
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./figura1-original
Tiempo= 0.203135000      R[0] = 0        R[39999]= -199995000
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./figura1-modificado_a
Tiempo= 0.150499800      R[0] = 0        R[39999]= -199995000
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./figura1-modificado_b
Tiempo= 0.118501800      R[0] = 0        R[39999]= -199995000
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer1$ ./figura1-modificado_c
Tiempo= 0.563048300      R[0] = 0        R[39999]= -799980000
```

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

Según las ejecuciones podemos comprobar que la versión más rápida resulta ser la de desenrollar el bucle interno obtenido bajo la modificación a). De esta manera limitamos la secuencialidad, eso sí, añadimos más saltos al código en ensamblador. Sin embargo el código en ensamblador realiza menos instrucciones de salto durante la ejecución.

2. 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=1;i<=N,i++) y[i]= a*x[i] + y[i];
```

2.1. Genere 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. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos. Sólo se debe evaluar el tiempo del núcleo DAXPY

2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo del número de operaciones en coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CAPTURA CÓDIGO FUENTE: daxpy.c

```c
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>

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

    struct timespec cgt1, cgt2; double ncgt;

    if(argc < 3){
        printf("Faltan nº componentes del vector y la constante\n");
        exit(-1);
    }

    unsigned n = atoi(argv[1]);
    int a = atoi(argv[2]);

    double *x, *y;
    x = (double*)malloc(n*sizeof(double));
    y = (double*)malloc(n*sizeof(double));

    for(int i=0; i<n; i++){
        x[i] = i + 9;
        y[i] = i + 2;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

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

    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    printf("Tamaño= %d\t Tiempo= %11.9.f\t y[0]= %f\t y[%d]= %f\n",n,ncgt,y[0],n-1,y[n-1]);

    free(x);
    free(y);
    return 0;
}
```

Tiempos ejec.	-O0	-Os	-O2	-O3
	0.5357	0.5146	0.49041	0.4717

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

```
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer2$ gcc -o daxpy daxpy.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer2$ ./daxpy 100000000 9
Tamaño= 100000000        Tiempo= 0.535666500     y[0]= 83.000000         y[99999999]= 1000000073.000000
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer2$ gcc -Os daxpy daxpy.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer2$ ./daxpy 100000000 9
Tamaño= 100000000        Tiempo= 0.514605900     y[0]= 83.000000         y[99999999]= 1000000073.000000
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer2$ gcc -O2 daxpy daxpy.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer2$ ./daxpy 100000000 9
Tamaño= 100000000        Tiempo= 0.490405800     y[0]= 83.000000         y[99999999]= 1000000073.000000
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer2$ gcc -O3 daxpy daxpy.c
leonbjj@DESKTOP-5CE60HG:/mnt/d/MigueCc99/Escritorio/Universidad/AC/bp4/ejer2$ ./daxpy 100000000 9
Tamaño= 100000000        Tiempo= 0.471654200     y[0]= 83.000000         y[99999999]= 1000000073.000000
```

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
En carpeta /ejer2/daxpyO0s	En carpeta /ejer2/daxpyOss	En carpeta /ejer2/daxpyO2s	En carpeta /ejer2/daxpyO3s