2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 5. Optimización de código

Estudiante (nombre y apellidos): Álvaro Vega Romero
Grupo de prácticas y profesor de prácticas: B2 – Niceto Rafael Luque Sola
Fecha de entrega: 27/05/21
Fecha evaluación en clase: 28/05/21

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con las 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) i7-1065G7 CPU @ 1.30GHz

Sistema operativo utilizado: Ubuntu 18.04 LTS

Versión de gcc utilizada: gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

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

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~] 2021-05-17 Monday
$lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              8
On-line CPU(s) list: 0-7
Thread(s) per core:  2
Core(s) per socket:  4
Socket(s):           1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               126
Model name:          Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz
Stepping:            5
CPU MHz:             1498.000
CPU max MHz:         1498.0000
BogoMIPS:            2996.00
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 dts acpi mmx fxsr s
se sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4
_2 movbe popcnt aes xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase tsc_adjust bmi1 avx2 sme
p bmi2 erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt intel_pt avx512cd sha_ni avx512bw avx512vl av
x512vbmi umip avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg avx512_vpopcntdq rdpid ibrs ibpb stibp ssbd
```

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

//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de los ...
//tres defines siguientes puede estar descomentado):
//#define MATRIZ_LOCAL    // descomentar para que los vectores sean variables ...
                          // locales (si se supera el tamaño de la pila se ...
                          // generará el error "Violación de Segmento")
//#define MATRIZ_GLOBAL// descomentar para que los vectores sean variables ...
                          // globales (su longitud no estará limitada por el ...
                          // tamaño de la pila del programa)
#define MATRIZ_DYNAMIC    // descomentar para que los vectores sean variables ...
                          // dinámicas (memoria reutilizable durante la ejecución)
#ifdef MATRIZ_GLOBAL
#define MAX 33554432         //=2^25
double v1[MAX], v2[MAX], v3[MAX];
#endif

int main(int argc, char** argv)
{
  int i;
  struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

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

  unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
  #ifdef MATRIZ_LOCAL
  double m1[N][N], m2[N],[N] m3[N][N];   // Tamaño variable local en tiempo de ejecución ...
                    // disponible en C a partir de actualización C99
  #endif

  #ifdef MATRIZ_GLOBAL
  if (N>MAX) N=MAX;
  #endif

  #ifdef MATRIZ_DYNAMIC
  double **m1, **m2, **m3;
  m1 = (double**) malloc(N*sizeof(double*));// malloc necesita el tamaño en bytes
  m2 = (double**) malloc(N*sizeof(double*)); //si no hay espacio suficiente malloc devuelve NULL
  m3 = (double**) malloc(N*sizeof(double*));

  if ( (m1==NULL) || (m2==NULL) || (m3==NULL) )
  {
    printf("Error en la reserva de espacio para las matrices\n");
    exit(-2);
  }
```

```c
  for(i = 0 ; i < N ; i++)
  {
    m1[i] = (double*) malloc (N*sizeof(double));
    m2[i] = (double*) malloc (N*sizeof(double));
    m3[i] = (double*) malloc (N*sizeof(double));

    if((m1[i]==NULL) || (m2[i]==NULL) || (m3[i]==NULL))
      printf("Error en la reserva de espacio para las matrices\n");
  }

#endif

  //Inicializar matrices
  if (N < 9)
    for (i = 0; i < N; i++)
      for(int j = 0 ; j < N ; j++)
      {
        m1[i][j] = N * 0.1 + i * 0.1;
        m2[i][j] = N * 0.1 + j * 0.1;
        m3[i][j] = 0;
      }
  else
  {
    srand(time(0));
    for (i = 0; i < N; i++)
      for(int j = 0 ; j < N ; j++)
      {
        m1[i][j] = rand() / ((double) rand());
        m2[i][j] = rand() / ((double) rand());
        m3[i][j] = 0;
      }
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);
```

```
  for(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];

  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(seg.):%11.9f\t / Tamaño Matrices:%u\t/ m3[0][0]=(%8.6f) / / m3[N-1][N-1]=(%8.6f\n)", ncgt,N,m3[0][0],m3[N-1][N-1]);
  if(N < 6)
  {
    printf("\nm1: \n");
    for (i = 0; i < N; i++)
    {
      printf("\n");
      for(int j = 0 ; j < N ; j++)
        printf("%8.6f ", m1[i][j]);
    }

    printf("\nm2: \n");
    for (i = 0; i < N; i++)
    {
      printf("\n");
      for(int j = 0 ; j < N ; j++)
        printf("%8.6f ", m2[i][j]);
    }

    printf("\nm3: \n");
    for (i = 0; i < N; i++)
    {
      printf("\n");
      for(int j = 0 ; j < N ; j++)
        printf("%8.6f ", m3[i][j]);
    }
  }

  #ifdef MATRIZ_DYNAMIC

  for(int j = 0 ; j < N ; j++)
  {
    free(m1[i]);
    free(m2[i]);
    free(m3[i]);
  }


  free(m1);
  free(m2);
  free(m3);
  #endif
  return 0;
}
```

(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-: Cambiamos j por k para aprovechar la localidad espacial.

Modificación B) –explicación-: Desenrollaremos el bucle de j en 4 iteraciones.

…

CÓDIGOS FUENTE MODIFICACIONES

A) Captura de pmm-secuencial-modificado_A.c

```
for(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):

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer1] 2021-05-17 Monday
$./pmm-secuencial-modificado_A 4
Tiempo(seg.):0.000001900         / Tamaño Matrices:4    / m3[0][0]=(0.640000) / / m3[N-1][N-1]=(1.960000
)
m1:

0.400000 0.400000 0.400000 0.400000
0.500000 0.500000 0.500000 0.500000
0.600000 0.600000 0.600000 0.600000
0.700000 0.700000 0.700000 0.700000
m2:

0.400000 0.500000 0.600000 0.700000
0.400000 0.500000 0.600000 0.700000
0.400000 0.500000 0.600000 0.700000
0.400000 0.500000 0.600000 0.700000
m3:

0.640000 0.800000 0.960000 1.120000
0.800000 1.000000 1.200000 1.400000
0.960000 1.200000 1.440000 1.680000
1.120000 1.400000 1.680000 1.960000
```

B)

```
float tmp1, tmp2, tmp3, tmp4;
int aux;

for(i=0; i<N; i++)
{
  for(int j = 0 ; j < N ; j++)
  {
    tmp1 = 0;
    tmp2 = 0;
    tmp3 = 0;
    tmp4 = 0;

    for(int k = 0, aux = 0 ; aux < N/4 ; k+=4, aux++)
    {
      tmp1+=m1[i][k]*m2[k][j];
      tmp2+=m1[i][k+1]*m2[k+1][j];
      tmp3+=m1[i][k+2]*m2[k+2][j];
      tmp4+=m1[i][k+3]*m2[k+3][j];
    }

    m3[i][j]=tmp1+tmp2+tmp3+tmp4;
  }
}
```

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer1] 2021-05-17 Monday
$gcc -O2 -lrt -o pmm-secuencial-modificado_B pmm-secuencial-modificado_B.c
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer1] 2021-05-17 Monday
$./pmm-secuencial-modificado_B 4
Tiempo(seg.):0.000001400         / Tamaño Matrices:4    / m3[0][0]=(0.640000) / / m3[N-1][N-1]=(1.960000
)
m1:

0.400000 0.400000 0.400000 0.400000
0.500000 0.500000 0.500000 0.500000
0.600000 0.600000 0.600000 0.600000
0.700000 0.700000 0.700000 0.700000
m2:

0.400000 0.500000 0.600000 0.700000
0.400000 0.500000 0.600000 0.700000
0.400000 0.500000 0.600000 0.700000
0.400000 0.500000 0.600000 0.700000
m3:

0.640000 0.800000 0.960000 1.120000
0.800000 1.000000 1.200000 1.400000
0.960000 1.200000 1.440000 1.680000
1.120000 1.400000 1.680000 1.960000
```

TIEMPOS: (N=10000)

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	-	1.5645
Modificación A)	Cambio de índices entre j y k	0.6216
Modificación B)	Desenrollado de bucle k en 4 iteraciones	1.6712

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer1] 2021-05-17 Monday
$./pmm-secuencial 1000
Tiempo(seg.):1.564528000         / Tamaño Matrices:1000 / m3[0][0]=(10233.195015) / / m3[N-1][N-1]=(8804.898990
)[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer1] 2021-05-17 Monday
$./pmm-secuencial-modificado_A 1000
Tiempo(seg.):0.621638900         / Tamaño Matrices:1000 / m3[0][0]=(10554.297259) / / m3[N-1][N-1]=(26062.342575
)[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer1] 2021-05-17 Monday
$./pmm-secuencial-modificado_B 1000
Tiempo(seg.):1.671282400         / Tamaño Matrices:1000 / m3[0][0]=(14962.193359) / / m3[N-1][N-1]=(11262.910156
)[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer1] 2021-05-17 Monday
```

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

Se puede ver a vista de los resultados, que merece la pena optimizar el código. Con un simple cambio de cambiar j por k en un índice, (para aprovechar la localidad espacial) podemos obtener una gran mejora. También (aunque en mi caso probablemente por un error acumulado no) se debería ver una mejora al desenrollar el bucle en 4 iteraciones. Por lo tanto, merece mucho la pena optimizar el código para optimizar el rendimiento

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

struct aux{
        int a;
        int b;
};


int main(int argc, char** argv)
{
  //Leer argumento de entrada (nº de componentes del vector)
  if (argc<3)
  {
    printf("Faltan M y/o N\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[2]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
  unsigned int M = atoi(argv[1]);
  int i, ii, X1, X2;
  struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
  int R[M];
  struct aux s[N];

  //Inicializar vectores
  if (N < 9)
    for (i = 0; i < N; i++)
    {
      s[i].a = N * 0.1 + i * 0.1;
      s[i].b = N * 0.1 - i * 0.1;
    }
  else
  {
    srand(time(0));
    for (i = 0; i < N; i++)
    {
      s[i].a = rand()/ ((double) rand());
      s[i].b = rand()/ ((double) rand()); //printf("%d:%f,%f/",i,v1[i],v2[i]);
    }
  }
```

```
clock_gettime(CLOCK_REALTIME,&cgt1);

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;
}

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(seg.):%11.9f\t / Tamaño Vectores:N: %u, M: %u\n / R[0] = %u / R[N-1] = %u \n",ncgt,N,M, R[0], R[N-1]);


return 0;
```

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-: Juntar los dos for anidados en un for donde i vaya de 2 en 2

Modificación B) –explicación-: Juntar los dos for anidados en un for donde desenrolemos de 2 en 2 u otro número

Otras posibles mejoras que podríamos hacer pero no he implementado son:

Desplazar 1 bit a la izq cuando multiplicamos por dos

En vez de usar if else, usar el operador ?

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c

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

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

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

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer2] 2021-05-17 Monday
$./figura1-modificado_A 1000 1000
Tiempo(seg.):0.001948300         / Tamaño Vectores:N: 1000, M: 1000
 / R[0] = 6480 / R[N-1] = 4293982297
```

B)

```
for (ii=0; ii<M;ii++)
{
   X1=0; X2=0;
   for(i=0; i<N;i+=2)
   {
     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;
   }

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

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer2] 2021-05-17 Monday
$./figura1-modificado_B 1000 1000
Tiempo(seg.):0.001433900         / Tamaño Vectores:N: 1000, M: 1000
 / R[0] = 9352 / R[N-1] = 4293983044
```

TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar	-	0.119748600
Modificación A)	Fusionar bucles	0.078078000
Modificación B)	Fusionar bucles y desenrolar de 2 en 2	0.077411500

COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:
Volvemos a observar como podemos mejorar los resultados de los programas que hagamos mediante pequeños cambios. Estas mejoras en estos casos se producen al no usar recursos en hacer dos bucles idénticos y en desenrolar ese bucle funcionado de 2 en 2 iteraciones.

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

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

int main(int argc, char** argv)
{
  //Leer argumento de entrada (nº de componentes del vector)
  if (argc<3)
  {
    printf("Falta N\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);
  const int a = 23 ; // cte
  int i;
  struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
  double *x = (double *) malloc(N*sizeof(double));
  double *y = (double *) malloc(N*sizeof(double));

  //Inicializar vectores

    srand(time(0));
    for (i = 0; i < N; i++)
    {
      x[i] = rand()/ ((double) rand());
      y[i] = rand()/ ((double) rand());
    }

  clock_gettime(CLOCK_REALTIME,&cgt1);

   for (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("Tiempo(seg.):%11.9f\t / Tamaño Vectores:N: %d, / y[0] = %f / y[N-1] = %f \n",ncgt,N, y[0], y[N-1]);


  return 0;
}
```

Tiempos ejec. Longitud vectores=23Millones(23000000)	-O0	-Os	-O2	-O3
	0.08400790	0.02256300	0.022479700	0.0238578

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

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer3] 2021-05-17 Monday
$gcc -O2 -lrt -o daxpyO2 daxpy.c
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer3] 2021-05-17 Monday
$gcc -O1 -lrt -o daxpyO1 daxpy.c
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer3] 2021-05-17 Monday
$gcc -Os -lrt -o daxpyOs daxpy.c
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer3] 2021-05-17 Monday
$gcc -O0 -lrt -o daxpyO0 daxpy.c
```

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer3] 2021-05-18 Tuesday
$./daxpyO3 23000000
Tiempo(seg.):0.023857800         / Tamaño Vectores:N: 23000000, / y[0] = 11.112781 / y[N-1] = 20.202529
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer3] 2021-05-18 Tuesday
$./daxpyO2 23000000
Tiempo(seg.):0.022479700         / Tamaño Vectores:N: 23000000, / y[0] = 31.475705 / y[N-1] = 5.322294
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer3] 2021-05-18 Tuesday
$./daxpyOs 23000000
Tiempo(seg.):0.022563000         / Tamaño Vectores:N: 23000000, / y[0] = 2208.752252 / y[N-1] = 24.489891
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp4/ejer3] 2021-05-18 Tuesday
$./daxpyO0 23000000
Tiempo(seg.):0.084007900         / Tamaño Vectores:N: 23000000, / y[0] = 226.896647 / y[N-1] = 9.961027
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

- O0: El compilador no realiza ninguna optimización del Código, por tanto, suele tener el peor resultado en las ejecuciones aunque mejor tiempo de compilación.
- Os: Optimiza el que más el Código, ocupando menos instrucciones y ejecutándose en un muy buen tiempo.
- O2: Es la opción que hemos usado durante todo el curso, y como bien podemos suponer, la mejor en términos generales. Intenta aumentar el rendimiento del Código sin comprometer el tamaño ni gastar ,ucho tiempo de compilación.
- O3: Obtenemos la mayor optimización posible del Código, aunque estas optimizaciones sean caras a nivel de recursos, provocando que realmente no optimice mucho a nivel a cantidad de instrucciones. Por lo general, el tiempo de ejecución suele ser el mejor de todos.

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

```
        movl$0, -84(%rbp)
        jmp     .L5
.L6:
        movl    -84(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        movq    -64(%rbp), %rax
        addq    %rdx, %rax
        movsd   (%rax), %xmm1
        cvtsi2sd -76(%rbp), %xmm0
        movl    -84(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        movq    -72(%rbp), %rax
        addq    %rdx, %rax
        movsd   (%rax), %xmm2
        mulsd   %xmm2, %xmm0
        movl    -84(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        movq    -64(%rbp), %rax
        addq    %rdx, %rax
        addsd   %xmm1, %xmm0
        movsd   %xmm0, (%rax)
        addl    $1, -84(%rbp)
.L5:
        movl    -84(%rbp), %eax
        cmpl    %eax, -80(%rbp)
        ja      .L6
        leaq    -32(%rbp), %rax
        movq    %rax, %rsi
        movl    $0, %edi
```

daxpyOs.s

```
    movsd   .LC1(%rip), %xmm1
        xorl    %eax, %eax
.L5:
        cmpl    %eax, %ebp
        jbe     .L11
        movsd
        (%r14,%rax,8), %xmm0
        mulsd   %xmm1, %xmm0
        addsd
        (%rbx,%rax,8), %xmm0
        movsd   %xmm0,
(%rbx,%rax,8)
        incq    %rax
        jmp     .L5
.L11:
        leaq    24(%rsp), %rsi
        xorl    %edi, %edi
```

daxpyO2.s

```
        movsd   .LC1(%rip), %xmm1
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L5:
        movsd
        (%r12,%rax), %xmm0
        mulsd   %xmm1, %xmm0
        addsd
        0(%r13,%rax), %xmm0
        movsd%xmm0, 0(%r13,%rax)
        addq    $8, %rax
        cmpq    %rax, %rbx
        jne     .L5
.L6:
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
```

daxpyO3.s

```
movq    %rbp, %rcx
        shrq    $3, %rcx
        andl    $1, %ecx
        cmpl    $1, 12(%rsp)
        jbe     .L11
        xorl    %edi, %edi
        testl   %ecx, %ecx
        je      .L6
        movsd
        .LC1(%rip), %xmm0
        movl    $1, %edi
        mulsd   (%r12), %xmm0
        addsd   0(%rbp), %xmm0
        movsd   %xmm0, 0(%rbp)
.L6:
        movl    %r14d, %r9d
        movapd
        .LC2(%rip), %xmm1
        subl    %ecx, %r9d
        salq    $3, %rcx
        xorl    %eax, %eax
        movl    %r9d, %r8d
        leaq    0(%rbp,%rcx), %rsi
        xorl    %edx, %edx
        shrl    %r8d
        addq    %r12, %rcx
        .p2align 4,,10
        .p2align 3
.L7:
        movupd
        (%rcx,%rax), %xmm0
        addl    $1, %edx
        mulpd   %xmm1, %xmm0
        addpd(%rsi,%rax), %xmm0
        movaps%xmm0, (%rsi,%rax)
        addq    $16, %rax
        cmpl    %r8d, %edx
        jb      .L7
        movl    %r9d, %edx
        andl    $-2, %edx
        cmpl    %edx, %r9d
        leal    (%rdx,%rdi), %eax
        je      .L9
.L5:
        movslq  %eax, %rcx
        movsd
        .LC1(%rip), %xmm0
        movsd
        (%r12,%rcx,8), %xmm1
        leaq
        0(%rbp,%rcx,8), %rdx
        addl    $1, %eax
        mulsd   %xmm0, %xmm1
        cmpl    %eax, %r14d
```

			addsd (%rdx), %xmm1
			movsd %xmm1, (%rdx)
			jbe .L9
			cltq
			mulsd (%r12,%rax,8), %xmm0
			leaq 0(%rbp,%rax,8), %rdx
			addsd (%rdx), %xmm0
			movsd %xmm0, (%rdx)
			.L9:
			leaq 32(%rsp), %rsi
			xorl %edi, %edi