2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Carlos Sánchez Páez
Grupo de prácticas: A2
Fecha de entrega: 28/05/2018
Fecha evaluación en clase: 29/05/2018

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

Sistema operativo utilizado: *Ubuntu 18.04 LTS 64bits*

Versión de gcc utilizada: *7.3.0*

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

```
Arquitectura:                        x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:                  Little Endian
CPU(s):                              8
Lista de la(s) CPU(s) en línea:      0-7
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:              4
«Socket(s)»                          1
Modo(s) NUMA:                        1
ID de fabricante:                    GenuineIntel
Familia de CPU:                      6
Modelo:                              94
Nombre del modelo:                   Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
Revisión:                            3
CPU MHz:                             800.020
CPU MHz máx.:                        3500,0000
CPU MHz mín.:                        800,0000
BogoMIPS:                            5184.00
Virtualización:                      VT-x
Caché L1d:                           32K
Caché L1i:                           32K
Caché L2:                            256K
Caché L3:                            6144K
CPU(s) del nodo NUMA 0:              0-7
Indicadores:                         fpu vme de pse tsc msr pae mce cx8 apic sep
 mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe sys
call nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl x
topology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monito
r ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic mov
be popcnt aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb in
vpcid_single pti tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1
hle avx2 smep bmi2 erms invpcid rtm mpx rdseed adx smap clflushopt intel_pt xsav
eopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window
 hwp_epp
```

1. Para el núcleo que se muestra en el Figura 1, y para un programa que implemente la multiplicación de matrices (use variables dinámicas):

   1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

   1.2 Genere los códigos en ensamblador 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

```
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`

```
for (int i = 0; i < TAM; i++) {
    for (int j = 0; j < TAM; j++) {
        temp = 0;
        for (int k = 0; k < TAM; k++)
            temp += m1[i][k] * m2[k][j];
        m3[i][j] = temp;
    }
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: desenrollado de bucle.

Modificación b) –explicación-: declaración de variables óptimas según su aceso (indices).

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura de `pmm-secuencial-modificado_a.c`

```
for (int i = 0; i < TAM; i++) {
  for (int j = 0; j < TAM; j++) {
    temp1 = 0;
    temp2 = 0;
    temp3 = 0;
    temp4 = 0;
    for (int k = 0; k < TAM; k += 4) {
      temp1 += m1[i][k] * m2[k][j];
      temp2 += m1[i][k + 1] * m2[k + 1][j];
      temp3 += m1[i][k + 2] * m2[k + 2][j];
      temp4 += m1[i][k + 3] * m2[k + 3][j];
    }
    m3[i][j] = temp1 + temp2 + temp3 + temp4;
  }
}
```

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

```
[Carlos Sanchez Paez csp98@csp98-PortatilLinux:~/Escritorio/AC/Prácticas/practica4/src] 2018-05-22 martes
$gcc -o pmm-secuencial-opt1 pmm-secuencial-opt1.c
[Carlos Sanchez Paez csp98@csp98-PortatilLinux:~/Escritorio/AC/Prácticas/practica4/src] 2018-05-22 martes
$./pmm-secuencial 12

Tiempo (seg): 0.000026165       Tamaño: 12      m3[0][0]=1.000000       m3[11][11]=1.000000
[Carlos Sanchez Paez csp98@csp98-PortatilLinux:~/Escritorio/AC/Prácticas/practica4/src] 2018-05-22 martes
$./pmm-secuencial-opt1l 12
bash: ./pmm-secuencial-opt1l: No existe el archivo o el directorio
[Carlos Sanchez Paez csp98@csp98-PortatilLinux:~/Escritorio/AC/Prácticas/practica4/src] 2018-05-22 martes
$
```

b)

```
double temp;
clock_gettime(CLOCK_REALTIME, &cgt1);
int i,j,k;
for (i = 0; i < TAM; i++) {
  for (j = 0; j < TAM; j++) {
    temp = 0;
    for (k = 0; k < TAM; k++)
      temp += m1[i][k] * m2[k][j];
    m3[i][j] = temp;
  }
}
```

```
[Carlos Sanchez Paez csp98@csp98-PortatilLinux:~/Escritorio/AC/Prácticas/practica4/src] 2018-05-22 martes
$gcc -o pmm-secuencial-opt2 pmm-secuencial-opt2.c
[Carlos Sanchez Paez csp98@csp98-PortatilLinux:~/Escritorio/AC/Prácticas/practica4/src] 2018-05-22 martes
$./pmm-secuencial-opt2 12

Tiempo (seg): 0.000026093        Tamaño: 12     m3[0][0]=1.000000       m3[11][11]=1.000000
[Carlos Sanchez Paez csp98@csp98-PortatilLinux:~/Escritorio/AC/Prácticas/practica4/src] 2018-05-22 martes
$./pmm-secuencial 12

Tiempo (seg): 0.000040929        Tamaño: 12     m3[0][0]=1.000000       m3[11][11]=1.000000
[Carlos Sanchez Paez csp98@csp98-PortatilLinux:~/Escritorio/AC/Prácticas/practica4/src] 2018-05-22 martes
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	0.000005899
Modificación a)	0.000004407
Modificación b)	0.000005776

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES : (PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

pmm-secuencial.s

```
.L14:
 movl  -124(%rbp), %eax
 cltq
 leaq  0(,%rax,8), %rdx
 movq  -96(%rbp), %rax
 addq  %rdx, %rax
 movq  (%rax), %rax
 movl  -116(%rbp), %edx
 movslq  %edx, %rdx
 salq  $3, %rdx
 addq  %rdx, %rax
 movsd (%rax), %xmm1
 movl  -116(%rbp), %eax
 cltq
 leaq  0(,%rax,8), %rdx
 movq  -88(%rbp), %rax
 addq  %rdx, %rax
 movq  (%rax), %rax
 movl  -120(%rbp), %edx
 movslq  %edx, %rdx
 salq  $3, %rdx
 addq  %rdx, %rax
 movsd (%rax), %xmm0
 mulsd %xmm1, %xmm0
 movsd -104(%rbp), %xmm1
 addsd %xmm1, %xmm0
 movsd %xmm0, -104(%rbp)
 addl  $1, -116(%rbp)
```

pmm-secuencial-modificado_b.s

```
.L14:
 movl  -148(%rbp), %eax
 cltq
 leaq  0(,%rax,8), %rdx
 movq  -96(%rbp), %rax
 addq  %rdx, %rax
 movq  (%rax), %rax
 movl  -140(%rbp), %edx
 movslq  %edx, %rdx
 salq  $3, %rdx
 addq  %rdx, %rax
 movsd (%rax), %xmm1
 movl  -140(%rbp), %eax
 cltq
 leaq  0(,%rax,8), %rdx
 movq  -88(%rbp), %rax
 addq  %rdx, %rax
 movq  (%rax), %rax
 movl  -144(%rbp), %edx
 movslq  %edx, %rdx
 salq  $3, %rdx
 addq  %rdx, %rax
 movsd (%rax), %xmm0
 mulsd %xmm1, %xmm0
 movsd -128(%rbp), %xmm1
 addsd %xmm1, %xmm0
 movsd %xmm0, -128(%rbp)
 movl  -148(%rbp), %eax
 cltq
 leaq  0(,%rax,8), %rdx
 movq  -96(%rbp), %rax
 addq  %rdx, %rax
 movq  (%rax), %rax
 movl  -140(%rbp), %edx
 movslq  %edx, %rdx
 addq  $1, %rdx
 salq  $3, %rdx
 addq  %rdx, %rax
 movsd (%rax), %xmm1
 movl  -140(%rbp), %eax
 cltq
 addq  $1, %rax
 leaq  0(,%rax,8), %rdx
 movq  -88(%rbp), %rax
 addq  %rdx, %rax
 movq  (%rax), %rax
 movl  -144(%rbp), %edx
 movslq  %edx, %rdx
 salq  $3, %rdx
 addq  %rdx, %rax
 movsd (%rax), %xmm0
 mulsd %xmm1, %xmm0
 movsd -120(%rbp), %xmm1
 addsd %xmm1, %xmm0
 movsd %xmm0, -120(%rbp)
 movl  -148(%rbp), %eax
 cltq
 leaq  0(,%rax,8), %rdx
 movq  -96(%rbp), %rax
 addq  %rdx, %rax
 movq  (%rax), %rax
 movl  -140(%rbp), %edx
 movslq  %edx, %rdx
```

pmm-secuencial-modificado_c.s

```
.L14:
 movl  -124(%rbp), %eax
 cltq
 leaq  0(,%rax,8), %rdx
 movq  -96(%rbp), %rax
 addq  %rdx, %rax
 movq  (%rax), %rax
 movl  -116(%rbp), %edx
 movslq  %edx, %rdx
 salq  $3, %rdx
 addq  %rdx, %rax
 movsd (%rax), %xmm1
 movl  -116(%rbp), %eax
 cltq
 leaq  0(,%rax,8), %rdx
 movq  -88(%rbp), %rax
 addq  %rdx, %rax
 movq  (%rax), %rax
 movl  -120(%rbp), %edx
 movslq  %edx, %rdx
 salq  $3, %rdx
 addq  %rdx, %rax
 movsd (%rax), %xmm0
 mulsd %xmm1, %xmm0
 movsd -104(%rbp), %xmm1
 addsd %xmm1, %xmm0
 movsd %xmm0, -104(%rbp)
 addl  $1, -116(%rbp)
```

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

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

struct {
  int a;
  int b;
} s[5000];

int main() {
  int x1, x2, i, j;
  int *R;
  const int TAM = 40000;
  const int TAM2 = 5000;
  struct timespec cgt1, cgt2;
  double ncgt;

  clock_gettime(CLOCK_REALTIME, &cgt1);

  R = (int *)malloc(TAM * sizeof(int));
  for (i = 0; i < TAM; i++) {
    x1 = 0;
    x2 = 0;
    for (j = 0; j < TAM2; j++)
      x1 += 2 * s[i].a + i;
    for (j = 0; j < TAM2; j++)
      x2 += 3 * s[i].b - i;
    if (x1 < x2)
      R[i] = x1;
    else
      R[i] = 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("\nTiempo (seg): %11.9f\n", ncgt);

  free(R);
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: fusionar los dos bucles for.

Modificación b) –explicación-: desenrollado de bucle.

...

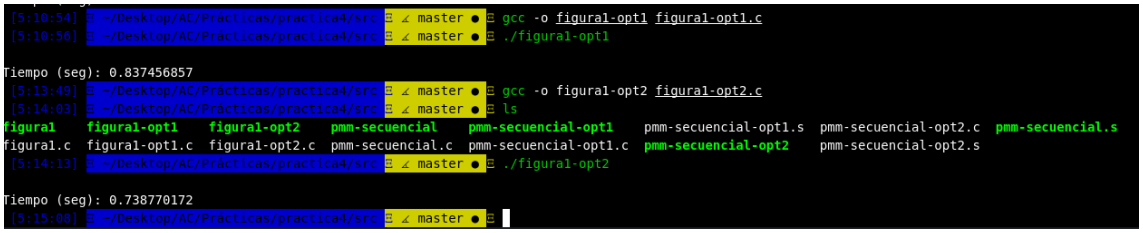
1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1**`-modificado_a.c`**

```
for (ii = 0; ii < TAM; ii++) {
  X1 = 0;
  X2 = 0;
  for (i = 0; i < TAM2; i++) {
    X1 += 2 * s[i].a + ii;
    X2 += 3 * s[i].b - ii;
  }
  if (X1 < X2)
    R[ii] = X1;
  else
    R[ii] = X2;
}
```

```
for (ii = 0; ii < TAM; ii++) {
  X1 = 0;
  X2 = 0;
  for (i = 0; i < TAM2; i+=4) {
    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;
  }
  if (X1 < X2)
    R[ii] = X1;
  else
    R[ii] = X2;
}
```

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



1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*1.205832117*
Modificación a)	0.838946310
Modificación b)	0.738770172
...	

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

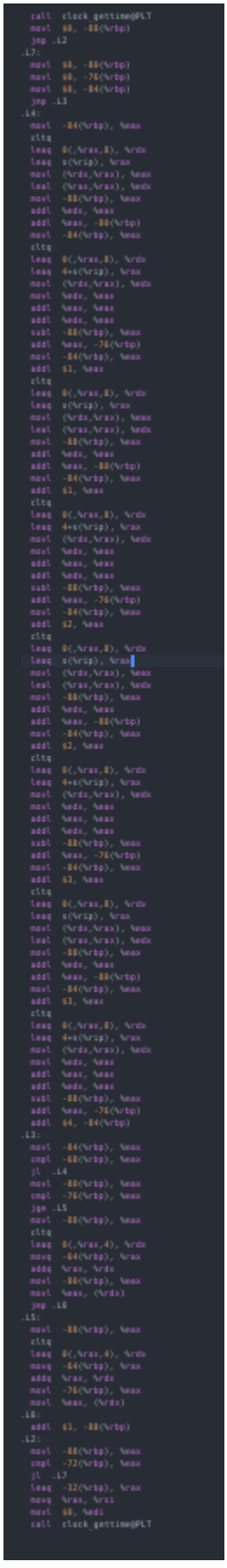
Vemos que el desenrollado de bucle vuelve a optimizar mucho la ejecución. Si además fusionamos ambos bucles, el tiempo se reduce a casi la mitad.

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES: (PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

`pmm-secuencial.s`	`pmm-secuencial-modificado_b.s`	`pmm-secuencial-modificado_c.s`

```
  call  clock_gettime@PLT
  movl  $0, -88(%rbp)
  jmp .L2
.L9:
  movl  $0, -80(%rbp)
  movl  $0, -76(%rbp)
  movl  $0, -84(%rbp)
  jmp .L3
.L4:
  movl  -84(%rbp), %eax
  cltq
  leaq  0(,%rax,8), %rdx
  leaq  s(%rip), %rax
  movl  (%rdx,%rax), %eax
  leal  (%rax,%rax), %edx
  movl  -88(%rbp), %eax
  addl  %edx, %eax
  addl  %eax, -80(%rbp)
  addl  $1, -84(%rbp)
.L3:
  movl  -84(%rbp), %eax
  cmpl  -68(%rbp), %eax
  jl  .L4
  movl  $0, -84(%rbp)
  jmp .L5
.L6:
  movl  -84(%rbp), %eax
  cltq
  leaq  0(,%rax,8), %rdx
  leaq  4+s(%rip), %rax
  movl  (%rdx,%rax), %edx
  movl  %edx, %eax
  addl  %eax, %eax
  addl  %edx, %eax
  subl  -88(%rbp), %eax
  addl  %eax, -76(%rbp)
  addl  $1, -84(%rbp)
.L5:
  movl  -84(%rbp), %eax
  cmpl  -68(%rbp), %eax
  jl  .L6
  movl  -80(%rbp), %eax
  cmpl  -76(%rbp), %eax
  jge .L7
  movl  -88(%rbp), %eax
  cltq
  leaq  0(,%rax,4), %rdx
  movq  -64(%rbp), %rax
  addq  %rax, %rdx
  movl  -80(%rbp), %eax
  movl  %eax, (%rdx)
  jmp .L8
.L7:
  movl  -88(%rbp), %eax
  cltq
  leaq  0(,%rax,4), %rdx
  movq  -64(%rbp), %rax
  addq  %rax, %rdx
  movl  -76(%rbp), %eax
  movl  %eax, (%rdx)
.L8:
  addl  $1, -88(%rbp)
.L2:
  movl  -88(%rbp), %eax
  cmpl  -72(%rbp), %eax
  jl  .L9
  leaq  -32(%rbp), %rax
  movq  %rax, %rsi
  movl  $0, %edi
  call  clock_gettime@PLT
```

```
  call  clock_gettime@PLT
  movl  $0, -88(%rbp)
  jmp .L2
.L7:
  movl  $0, -80(%rbp)
  movl  $0, -76(%rbp)
  movl  $0, -84(%rbp)
  jmp .L3
.L4:
  movl  -84(%rbp), %eax
  cltq
  leaq  0(,%rax,8), %rdx
  leaq  s(%rip), %rax
  movl  (%rdx,%rax), %eax
  leal  (%rax,%rax), %edx
  movl  -88(%rbp), %eax
  addl  %edx, %eax
  addl  %eax, -80(%rbp)
  movl  -84(%rbp), %eax
  cltq
  leaq  0(,%rax,8), %rdx
  leaq  4+s(%rip), %rax
  movl  (%rdx,%rax), %edx
  movl  %edx, %eax
  addl  %eax, %eax
  addl  %edx, %eax
  subl  -88(%rbp), %eax
  addl  %eax, -76(%rbp)
  addl  $1, -84(%rbp)
.L3:
  movl  -84(%rbp), %eax
  cmpl  -68(%rbp), %eax
  jl  .L4
  movl  -80(%rbp), %eax
  cmpl  -76(%rbp), %eax
  jge .L5
  movl  -88(%rbp), %eax
  cltq
  leaq  0(,%rax,4), %rdx
  movq  -64(%rbp), %rax
  addq  %rax, %rdx
  movl  -80(%rbp), %eax
  movl  %eax, (%rdx)
  jmp .L6
.L5:
  movl  -88(%rbp), %eax
  cltq
  leaq  0(,%rax,4), %rdx
  movq  -64(%rbp), %rax
  addq  %rax, %rdx
  movl  -76(%rbp), %eax
  movl  %eax, (%rdx)
.L6:
  addl  $1, -88(%rbp)
.L2:
  movl  -88(%rbp), %eax
  cmpl  -72(%rbp), %eax
  jl  .L7
  leaq  -32(%rbp), %rax
  movq  %rax, %rsi
  movl  $0, %edi
  call  clock_gettime@PLT
```



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 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.

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 (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CAPTURA CÓDIGO FUENTE: `daxpy.c`

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

  if (argc != 2) {
    printf("Falta tamanio\n");
    exit(-1);
  }

  const int N = atoi(argv[1])>INT_MAX ? INT_MAX : atoi(argv[1]);
  const float a = 5.15445455445;
  int i;
  float *x, *y;
  x = (float *)malloc(N * sizeof(float));
  y = (float *)malloc(N * sizeof(float));

  for (i = 0; i < N; i++) {
    x[i] = 23.1313455445513;
    y[i] = 54.54455544554;
  }

  struct timespec cgt1, cgt2;
  double ncgt;

  clock_gettime(CLOCK_REALTIME, &cgt1);

  for (i = 0; i < N; i++)
    y[i] = a * x[i] + y[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("\nTiempo (seg): %11.9f\n", ncgt);

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

El tamaño usado es 999999999.

	-O0	**-Os**	**-O2**	**-O3**
Tiempos ejec.	*3.601008 055*	3.59974 6155	1.139416 070	1.067987 405

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

```
[5:50:17] ~/Desktop/AC/Prácticas/practica4/src master ● gcc -o daxpy03 -03  daxpy.c ; gcc -o daxpy02 -02 daxpy.c ; gcc -o daxpy daxpy.c
[5:50:42] ~/Desktop/AC/Prácticas/practica4/src master ● ./daxpy03 999999999 ; ./daxpy02 999999999 ; ./daxpy 999999999

Tiempo (seg): 1.067987405

Tiempo (seg): 1.139416070

Tiempo (seg): 3.601008055
[5:51:06] ~/Desktop/AC/Prácticas/practica4/src master ●
```

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)**

daxpy00.s	daxpy0s.s	daxpy02.s	daxpy03.s