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): Paula Ruiz García
Grupo de prácticas: D1
Fecha de entrega: 3 de Junio de 2018
Fecha evaluación en clase: 28 de Mayo de 2018

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz*

Sistema operativo utilizado: *Windows 10 Home (utilizo la Bash de Ubuntu en Windows (16.04))*

Versión de gcc utilizada: *gcc (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609*

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

```
[Paula Ruiz Garcia Paula@Pompitas:~] 2018-06-01 Friday
$lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    2
Socket(s):             1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 142
Model name:            Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
Stepping:              9
CPU MHz:               2701.000
CPU max MHz:           2701.0000
BogoMIPS:              5402.00
Virtualization:        VT-x
Hypervisor vendor:     vertical
Virtualization type:   full
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx f
xsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 fma cx16
 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave osxsave avx f16c rdrand
[Paula Ruiz Garcia Paula@Pompitas:~] 2018-06-01 Friday
$
```

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

   1.1Modifique el código C para reducir el tiempo de ejecución del mismo. 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

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

#define MAX 3355
int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];

int main(int argc, char **argv)
{
 unsigned i, j, k;
 struct timespec cgt1,cgt2; double ncgt;

 if(argc < 2){
  fprintf(stderr, "./pmm-secuencial [TAM]\n");
  exit(-1);
 }

 unsigned int N = atoi(argv[1]);

 if(N>MAX){
  N=MAX;
 }

 // Inicializamos las matrices
 for (i=0; i<N; i++){
  for (j=0; j<N; j++){
   a[i][j] = 0;
   b[i][j] = 2;
   c[i][j] = 2;
  }
 }

 clock_gettime(CLOCK_REALTIME,&cgt1);
```

```
  // Multiplicacion
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      for (k=0; k<N; k++)
        a[i][j] += b[i][k] * c[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));

  if(N<10){
    //Pintamos la matriz
    printf("Matriz b:\n");
    for(i=0; i<N; i++){
      for(j=0; j<N; j++){
        printf("%d ", b[i][j]);
      }
      printf("\n");
    }

    printf("Matriz c:\n");
    for(i=0; i<N; i++){
      for(j=0; j<N; j++){
        printf("%d ", c[i][j]);
      }
      printf("\n");
    }

    //Pintamos la matriz solucion
    printf("Matriz a:\n");
    for(i=0; i<N; i++){
      for(j=0; j<N; j++){
        printf("%d ", a[i][j]);
      }
      printf("\n");
    }
  }

  // Pitamos la primera y la ultima linea de la matriz resultante
  printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,a[0][0],a[N-1][N-1]);

  return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: desenrollado del bucle `k` en 4 partes.

Modificación b) –explicación-: He cambiado el orden de los bucles `j` y `k` ya que así están os datos más próximos en memoria.

…

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura de pmm-secuencial-modificado_a.c

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

#define MAX 3355
int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];
```

```
int main(int argc, char **argv)
{
  unsigned i, j, k;
  int S0, S1, S2, S3, total, h;
  struct timespec cgt1,cgt2; double ncgt;

  if(argc < 2){
    fprintf(stderr, "./pmm-secuencial [TAM]\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);
  int iteraciones = N/4;

  if(N>MAX){
    N=MAX;
  }

  // Inicializamos las matrices
  for (i=0; i<N; i++){
    for (j=0; j<N; j++){
      a[i][j] = 0;
      b[i][j] = 2;
      c[i][j] = 2;
    }
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);

  // Multiplicacion
  for (i=0; i<N; i++)
    for (j=0; j<N; j++){
      S0 = S1 = S2 = S3 = 0;
      for (k=0, h=0; h<iteraciones; ++h, k+=4){
        S0 += b[i][k] * c[k][j];
        S1 += b[i][k+1] * c[j][k+1];
        S2 += b[i][k+2] * c[j][k+2];
        S3 += b[i][k+3] * c[j][k+3];
      }
      total = S0 + S1 + S2 + S3;
      a[i][j] = total;
    }

  clock_gettime(CLOCK_REALTIME,&cgt2);

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

  if(N<10){
    //Pintamos la matriz
    printf("Matriz b:\n");
    for(i=0; i<N; i++){
      for(j=0; j<N; j++){
        printf("%d ", b[i][j]);
      }
      printf("\n");
    }
```

```
  printf("Matriz c:\n");
  for(i=0; i<N; i++){
   for(j=0; j<N; j++){
    printf("%d ", c[i][j]);
   }
   printf("\n");
  }

  //Pintamos la matriz solucion
  printf("Matriz a:\n");
  for(i=0; i<N; i++){
   for(j=0; j<N; j++){
    printf("%d ", a[i][j]);
   }
   printf("\n");
  }
 }

 // Pitamos la primera y la ultima linea de la matriz resultante
 printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,a[0][0],a[N-1][N-1]);

 return 0;
}
```

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

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./pmm-secuencial 1000
Tiempo = 2.608114900     Primera = 4000  Ultima=4000
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$
```

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -O2 pmm-secuencial-modificado_a.c -o pmm-secuencial_a
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./pmm-secuencial_a 1000
Tiempo = 1.137043200     Primera = 4000  Ultima=4000
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$
```

b) Captura de pmm-secuencial-modificado_b.c

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

#define MAX 3355
int a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];

int main(int argc, char **argv)
{
 unsigned i, j, k;
 struct timespec cgt1,cgt2; double ncgt;
```

```
if(argc < 2){
  fprintf(stderr, "./pmm-secuencial [TAM]\n");
  exit(-1);
}

unsigned int N = atoi(argv[1]);

if(N>MAX){
  N=MAX;
}

// Inicializamos las matrices
for (i=0; i<N; i++){
  for (j=0; j<N; j++){
    a[i][j] = 0;
    b[i][j] = 2;
    c[i][j] = 2;
  }
}

clock_gettime(CLOCK_REALTIME,&cgt1);

// Multiplicacion
for (i=0; i<N; i++)
  for (k=0; k<N; k++)
    for (j=0; j<N; j++)
      a[i][j] += b[i][k] * c[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));

if(N<10){
  //Pintamos la matriz
  printf("Matriz b:\n");
  for(i=0; i<N; i++){
    for(j=0; j<N; j++){
      printf("%d ", b[i][j]);
    }
    printf("\n");
  }

  printf("Matriz c:\n");
  for(i=0; i<N; i++){
    for(j=0; j<N; j++){
      printf("%d ", c[i][j]);
    }
    printf("\n");
  }

  //Pintamos la matriz solucion
  printf("Matriz a:\n");
  for(i=0; i<N; i++){
    for(j=0; j<N; j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
```

```
   }
  }

  // Pitamos la primera y la ultima linea de la matriz resultante
  printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,a[0][0],a[N-1][N-1]);

  return 0;
}
```

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

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./pmm-secuencial 1000
Tiempo = 2.608114900      Primera = 4000  Ultima=4000
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$
```

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -O2 pmm-secuencial-modificado_b.c -o pmm-secuencial_b
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./pmm-secuencial_b 1000
Tiempo = 1.266426400      Primera = 4000  Ultima=4000
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*2.608114900*
Modificación a)	*1.137043200*
Modificación b)	*1.266426400*

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Aunque siempre debemos tener en cuenta la arquitectura que estamos usando para optimizar el código, con estas pruebas también podemos comprobar que con solo modificar el orden de dos bucles podemos optimizar bastante tanto el tiempo como el tamaño del código en ensamblador, aunque con el desenrollado de bucles nos de una mayor eficiencia de tiempo, pues consigue más velocidad de procesamiento, pero siempre aumentando también el tamaño del código.

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_a.s	pmm-secuencial-modificado_b.s

call clock_gettime
movq 40(%rsp), %rax
subq 24(%rsp), %rax
pxor %xmm0, %xmm0
pxor %xmm1, %xmm1
cvtsi2sdq %rax, %xmm0
movq 32(%rsp), %rax
subq 16(%rsp), %rax
cmpl $9, %ebx
cvtsi2sdq %rax, %xmm1
	call clock_gettime
	movq 56(%rsp), %rax
	subq 40(%rsp), %rax
	pxor %xmm0, %xmm0
	pxor %xmm1, %xmm1
	cvtsi2sdq %rax, %xmm0
	movq 48(%rsp), %rax
	subq 32(%rsp), %rax
	cmpl $9, %r15d
	cvtsi2sdq %rax, %xmm1
		call clock_gettime
		movq 40(%rsp), %rax
		subq 24(%rsp), %rax
		pxor %xmm0, %xmm0
		pxor %xmm1, %xmm1
		cvtsi2sdq %rax, %xmm0
		movq 32(%rsp), %rax
		subq 16(%rsp), %rax
		cmpl $9, %ebp
		cvtsi2sdq %rax, %xmm1

```
        divsd  .LC3(%rip), %xmm0
        addsd %xmm1, %xmm0
        movsd          %xmm0, 8(%rsp)
        jbe    .L50
.L20:
        movl   %ebp, %eax
        movl   a(%rip), %edx
        movl   $.LC2, %esi
        imulq $13424, %rax, %rax
        movsd          8(%rsp), %xmm0
        movl   $1, %edi
        movl   a(%rax), %ecx
        movl   $1, %eax
        call   __printf_chk
        xorl   %eax, %eax
        movq 56(%rsp), %rbx
        xorq   %fs:40, %rbx
        jne    .L51
        addq   $72, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 56
        popq   %rbx
        .cfi_def_cfa_offset 48
        popq   %rbp
        .cfi_def_cfa_offset 40
        popq   %r12
        .cfi_def_cfa_offset 32
        popq   %r13
        .cfi_def_cfa_offset 24
        popq   %r14
        .cfi_def_cfa_offset 16
        popq   %r15
        .cfi_def_cfa_offset 8
        ret
.L3:
        .cfi_restore_state
        leaq   16(%rsp), %rsi
        xorl   %edi, %edi
        movl   $-1, %ebp
        call   clock_gettime
```

```
        divsd  .LC3(%rip), %xmm0
        addsd %xmm1, %xmm0
        movsd          %xmm0, (%rsp)
        ja     .L21
        movl   $.LC4, %edi
        xorl   %r12d, %r12d
        xorl   %ebp, %ebp
        call   puts
.L14:
        xorl   %ebx, %ebx
.L13:
        movl   b(%r12,%rbx,4), %edx
        xorl   %eax, %eax
        movl   $.LC1, %esi
        movl   $1, %edi
        addq   $1, %rbx
        call   __printf_chk
        cmpl   %ebx, %r15d
        ja     .L13
        movl   $10, %edi
        addl   $1, %ebp
        addq   $13420, %r12
        call   putchar
        cmpl   %r15d, %ebp
        jb     .L14
        movl   $.LC5, %edi
        xorl   %r12d, %r12d
        xorl   %ebp, %ebp
        call   puts
.L17:
        xorl   %ebx, %ebx
.L16:
        movl   c(%r12,%rbx,4), %edx
        xorl   %eax, %eax
        movl   $.LC1, %esi
        movl   $1, %edi
        addq   $1, %rbx
        call   __printf_chk
        cmpl   %ebx, %r15d
        ja     .L16
        movl   $10, %edi
        addl   $1, %ebp
        addq   $13420, %r12
        call   putchar
        cmpl   %r15d, %ebp
        jb     .L17
        movl   $.LC6, %edi
        xorl   %r12d, %r12d
        xorl   %ebp, %ebp
        call   puts
.L20:
        xorl   %ebx, %ebx
.L19:
        movl   a(%r12,%rbx,4), %edx
        xorl   %eax, %eax
        movl   $.LC1, %esi
        movl   $1, %edi
        addq   $1, %rbx
        call   __printf_chk
        cmpl   %ebx, %r15d
        ja     .L19
        movl   $10, %edi
        addl   $1, %ebp
        addq   $13420, %r12
```

```
        divsd  .LC3(%rip), %xmm0
        addsd %xmm1, %xmm0
        movsd          %xmm0, 8(%rsp)
        jbe    .L50
.L20:
        movl   %r12d, %eax
        movl   a(%rip), %edx
        movl   $1, %edi
        imulq $13424, %rax, %rax
        movsd          8(%rsp), %xmm0
        movl   $.LC2, %esi
        movl   a(%rax), %ecx
        movl   $1, %eax
        call   __printf_chk
        xorl   %eax, %eax
        movq 56(%rsp), %rdi
        xorq   %fs:40, %rdi
        jne    .L51
        addq   $64, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 48
        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
        leaq   16(%rsp), %rsi
        xorl   %edi, %edi
        movl   $-1, %r12d
        call   clock_gettime
```

```
        call    putchar
        cmpl    %r15d, %ebp
        jb      .L20
        jmp     .L21
.L3:
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime
```

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: figura1-original.c

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

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

int R[40000];

int main(){
 struct timespec cgt1,cgt2;
  double ncgt;
 int ii, i, X1, X2;

 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;
 }
 //Calculo del tiempo
  clock_gettime(CLOCK_REALTIME,&cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

  //Imprimir resultado de la suma y el tiempo de ejecución
  printf("Tiempo(seg.):%11.9f\t",ncgt);
 printf("R[0] = %i, R[3999] = %i\n", R[0], R[3999]);

 return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: He hecho una agrupación de los dos `for` de `i` en uno ya que hacen lo mismo y así reducimos a la mitad de elementos a recorrer. Además, esto

también ayuda a aprovechar la localidad espacial de los valores `a` y `b` del `struct`, ya que estarán intercalados en el `array`.

Modificación b) –explicación-: En el siguiente caso, además de agrupar los dos `for` en un único `for`, lo he desenrollado en operaciones de 4.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1-modificado_a.c

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

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

int R[40000];

int main(){
 struct timespec cgt1,cgt2;
  double ncgt;
 int ii, i, X1, X2;

 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;
 }
 //Calculo del tiempo
  clock_gettime(CLOCK_REALTIME,&cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

  //Imprimir resultado de la suma y el tiempo de ejecución
  printf("Tiempo(seg.):%11.9f\t",ncgt);
 printf("R[0] = %i, R[3999] = %i\n", R[0], R[3999]);

 return 0;
}
```

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

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -O2 figura1-original.c -o figura1-original
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./figura1-original
Tiempo(seg.):0.431344500        R[0] = 0, R[3999] = -19995000
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$
```

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -O2 figura1-modificado_a.c -o figura1_a
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./figura1_a
Tiempo(seg.):0.279945400        R[0] = 0, R[3999] = -19995000
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$
```

b) Captura figura1-modificado_b.c

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

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

int R[40000];

int main(){
 struct timespec cgt1,cgt2;
  double ncgt;
 int ii, i, X1, X2;

 clock_gettime(CLOCK_REALTIME,&cgt1);
 for (ii = 0; ii < 40000; ++ii)
 {
      X1=0, X2=0;
      for (i = 0; i < 5000; 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;
 }
 //Calculo del tiempo
  clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

  //Imprimir resultado de la suma y el tiempo de ejecución
  printf("Tiempo(seg.):%11.9f\t",ncgt);
 printf("R[0] = %i, R[3999] = %i\n", R[0], R[3999]);

 return 0;
}
```

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

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -O2 figura1-original.c -o figura1-original
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./figura1-original
Tiempo(seg.):0.431344500        R[0] = 0, R[3999] = -19995000
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$
```

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -O2 figura1-modificado_b.c -o figura1_b
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./figura1_b
Tiempo(seg.):0.211713100        R[0] = 0, R[3999] = -19995000
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.431344500*
Modificación a)	*0.279945400*
Modificación b)	*0.211713100*

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Con los resultados obtenidos con este ejercicio, podemos comprobar que la reducción del número de bucles a la mitad reduce, más o menos, a la mitad la ejecución, y nos proporciona un código en ensamblador de tamaño más pequeño que el original. Mientras que, igual que en el ejercicio anterior, podemos ver que el desenrollado de código nos hace ganar mayor velocidad, pero también nos proporciona un código en ensamblador mayor. Y esto dice que según la arquitectura de nuestro ordenador y el tipo de optimización que busquemos, nos va a servir mejor una opción u otra.

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)

figura1-original.s	figura1-modificado_a.s	figura1-modificado_b.s
call clock_gettime xorl %r9d, %r9d movl $S+40004, %r8d .p2align 4,,10 .p2align 3	call clock_gettime xorl %r9d, %r9d movl $S+40000, %r8d .p2align 4,,10 .p2align 3	call clock_gettime xorl %r10d, %r10d movl $S+40000, %r9d .p2align 4,,10 .p2align 3

```
.L2:
        movl    %r9d, %edi
        movl    $S, %eax
        xorl    %esi, %esi
        .p2align 4,,10
        .p2align 3
.L3:
        movl    (%rax), %edx
        addq    $8, %rax
        leal    (%rdi,%rdx,2), %edx
        addl    %edx, %esi
        cmpq    $S+40000, %rax
        jne     .L3
        movl    $S+4, %eax
        xorl    %ecx, %ecx
        .p2align 4,,10
        .p2align 3
.L4:
        movl    (%rax), %edx
        addq    $8, %rax
        leal    (%rdx,%rdx,2), %edx
        subl    %edi, %edx
        addl    %edx, %ecx
        cmpq    %rax, %r8
        jne     .L4
        cmpl    %ecx, %esi
        cmovl   %esi, %ecx
        movl    %ecx, R(,%r9,4)
        addq    $1, %r9
        cmpq    $40000, %r9
        jne     .L2
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime
```

```
.L2:
        movl    %r9d, %edi
        movl    $S, %eax
        xorl    %ecx, %ecx
        xorl    %esi, %esi
        .p2align 4,,10
        .p2align 3
.L3:
        movl    (%rax), %edx
        addq    $8, %rax
        leal    (%rdi,%rdx,2), %edx
        addl    %edx, %esi
        movl    -4(%rax), %edx
        leal    (%rdx,%rdx,2), %edx
        subl    %edi, %edx
        addl    %edx, %ecx
        cmpq    %rax, %r8
        jne     .L3
        cmpl    %ecx, %esi
        cmovl   %esi, %ecx
        movl    %ecx, R(,%r9,4)
        addq    $1, %r9
        cmpq    $40000, %r9
        jne     .L2
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime
```

```
.L2:
        movl    %r10d, %r8d
        movl    $S, %eax
        xorl    %edx, %edx
        xorl    %esi, %esi
        .p2align 4,,10
        .p2align 3
.L3:
        movl    (%rax), %ecx
        addq    $32, %rax
        leal    (%r8,%rcx,2), %edi
        movl    -28(%rax), %ecx
        addl    %esi, %edi
        leal    (%rcx,%rcx,2), %esi
        movl    -24(%rax), %ecx
        subl    %r8d, %esi
        addl    %esi, %edx
        leal    (%r8,%rcx,2), %esi
        movl    -20(%rax), %ecx
        addl    %esi, %edi
        leal    (%rcx,%rcx,2), %ecx
        subl    %r8d, %ecx
        leal    (%rcx,%rdx), %esi
        movl    -16(%rax), %edx
        leal    (%r8,%rdx,2), %edx
        addl    %edx, %edi
        movl    -12(%rax), %edx
        leal    (%rdx,%rdx,2), %edx
        subl    %r8d, %edx
        leal    (%rdx,%rsi), %ecx
        movl    -8(%rax), %edx
        leal    (%r8,%rdx,2), %edx
        leal    (%rdx,%rdi), %esi
        movl    -4(%rax), %edx
        leal    (%rdx,%rdx,2), %edx
        subl    %r8d, %edx
        addl    %ecx, %edx
        cmpq    %rax, %r9
        jne     .L3
        cmpl    %edx, %esi
        cmovl   %esi, %edx
        movl    %edx, R(,%r10,4)
        addq    $1, %r10
        cmpq    $40000, %r10
        jne     .L2
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime
```

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

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

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

 int *y, *x;
 int i;
 struct timespec cgt1,cgt2;
 double ncgt;

 if (argc < 3) {
  fprintf(stderr, "ERROR: falta tam del vector y constante\n");
  exit(1);
 }

 unsigned n = strtol(argv[1], NULL, 10);
 int a = strtol(argv[2], NULL, 10);

 y = (int*) malloc(n*sizeof(int));
 x = (int*) malloc(n*sizeof(int));

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

 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("\nTiempo (seg.) = %11.9f\t", ncgt);
 printf("y[0] = %i, y[%i] = %i\n", y[0], n-1, y[n-1]);

 free(y);
 free(x);

 return 0;
}
```

Tiempos ejec.	**-O0**	**-Os**	**-O2**	**-O3**
	0.001668200	*0.000496500*	*0.000666400*	*0.000455700*

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

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -O0 daxpy.c -o daxpy_O0
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./daxpy_O0 400000 222

Tiempo (seg.) = 0.001668200     y[0] = 2, y[399999] = 177999557
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -Os daxpy.c -o daxpy_Os
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./daxpy_Os 400000 222

Tiempo (seg.) = 0.000496500     y[0] = 2, y[399999] = 177999557
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -O2 daxpy.c -o daxpy_O2
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./daxpy_O2 400000 222

Tiempo (seg.) = 0.000666400     y[0] = 2, y[399999] = 177999557
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$gcc -O3 daxpy.c -o daxpy_O3
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$./daxpy_O3 400000 222

Tiempo (seg.) = 0.000455700     y[0] = 2, y[399999] = 177999557
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P4/Codigo] 2018-06-02 Saturday
$
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

`-O0` es la optimización por defecto que ejecuta `gcc`, en la cual se desconecta por completo la optimización. En ella se usan direcciones relativas a la pila.

`-Os` es el encargado de compilar un código ensamblador lo mas pequeño posible, para ello activa todas las opciones que también tiene `-O2` que no incrementen el tamaño del código a generar. Es el que se usa en sistemas limitados, como los sistemas empotrados, maquinas con capacidad limitada de disco o con CPUs que tienen poca caché.

`-O2` Es el nivel recomendado de optimización. Con `-O2` el compilador intentara aumentar el rendimiento del código sin aumentar demasiado el tamaño del mismo a la vez que intenta no aumentar tampoco el tiempo de compilación. Utiliza registros de la arquitectura para almacenar la información, y así ahorrar muchas operaciones `move` que no nos son necesarias.

`-O3` es el mayor nivel de optimización. En el se activan optimizaciones que aumentan el tamaño del código a la vez que mejoran su velocidad, lo que conlleva a que ocupe mayor cantidad de memoria. En nuestro caso, para la optimización del código ha realizado un desenrollado de bucle, aumentando así su velocidad, pero también su tamaño.

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    clock_gettime
        movl    $0, -84(%rbp)
        jmp     .L5
.L6:
        movl    -84(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rdx
        movq    -72(%rbp), %rax
        addq    %rax, %rdx
        movl    -84(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rcx
        movq    -72(%rbp), %rax
        addq    %rcx, %rax
        movl    (%rax), %ecx
        movl    -84(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rsi
        movq    -64(%rbp), %rax
        addq    %rsi, %rax
        movl    (%rax), %eax
        imull   -76(%rbp), %eax
        addl    %ecx, %eax
        movl    %eax, (%rdx)
        addl    $1, -84(%rbp)
.L5:
        movl    -84(%rbp), %eax
        cmpl    -80(%rbp), %eax
        jb      .L6
        leaq    -32(%rbp), %rax
        movq    %rax, %rsi
        movl    $0, %edi
        call    clock_gettime
```

daxpyOs.s

```
        call    clock_gettime
        xorl    %eax, %eax
.L5:
        cmpl    %eax, %r14d
        jbe     .L11
        movl
        (%r12,%rax,4), %edx
        imull   %r13d, %edx
        addl    %edx,
(%rbx,%rax,4)
        incq    %rax
        jmp     .L5
.L11:
        leaq    24(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime
```

daxpyO2.s

```
        call    clock_gettime
        xorl    %edx, %edx
        .p2align 4,,10
        .p2align 3
.L5:
        movl
        (%r12,%rdx,4), %ecx
        imull   %r13d, %ecx
        addl    %ecx,
(%rbx,%rdx,4)
        addq    $1, %rdx
        cmpl    %edx, %r14d
        ja      .L5
.L6:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime
```

daxpyO3.s

```
        call    clock_gettime
        cmpl    $4, %r12d
        ja      .L54
        movl    %r12d, %ebp
.L22:
        movl    0(%r13), %eax
        imull   %r15d, %eax
        addl    %eax, (%rbx)
        cmpl    $1, %ebp
        je      .L33
        movl    4(%r13), %eax
        imull   %r15d, %eax
        addl    %eax, 4(%rbx)
        cmpl    $2, %ebp
        je      .L34
        movl    8(%r13), %eax
        imull   %r15d, %eax
        addl    %eax, 8(%rbx)
        cmpl    $4, %ebp
        jne     .L35
        movl    12(%r13), %eax
        imull   %r15d, %eax
        addl    %eax, 12(%rbx)
        movl    $4, %eax
.L14:
        cmpl    %ebp, %r12d
        je      .L21
.L13:
        movl    %r12d, %r8d
        movl    %r14d, %edx
        movl    %ebp, %esi
        subl    %ebp, %r8d
        subl    %ebp, %edx
        leal    -4(%r8), %ecx
        shrl    $2, %ecx
        addl    $1, %ecx
        cmpl    $2, %edx
        leal    0(,%rcx,4), %r9d
        jbe     .L16
        movd    8(%rsp), %xmm6
        salq    $2, %rsi
        xorl    %edx, %edx
        leaq    (%rbx,%rsi), %r10
        xorl    %edi, %edi
        addq    %r13, %rsi
        pshufd $0, %xmm6, %xmm2
        movdqa %xmm2, %xmm3
        psrlq   $32, %xmm3
.L17:
        movdqu          (%rsi,%rdx), %xmm0
        addl    $1, %edi
        movdqa          %xmm0, %xmm1
        psrlq   $32, %xmm0
        pmuludq         %xmm3, %xmm0
        pshufd
        $8, %xmm0, %xmm0
        pmuludq         %xmm2, %xmm1
        pshufd
        $8, %xmm1, %xmm1
        punpckldq       %xmm0, %xmm1
        movdqa
        (%r10,%rdx), %xmm0
```

```
        paddd %xmm1, %xmm0
        movaps %xmm0, (%r10,%rdx)
        addq  $16, %rdx
        cmpl  %edi, %ecx
        ja    .L17
        addl  %r9d, %eax
        cmpl  %r9d, %r8d
        je    .L21
.L16:
        movslq        %eax, %rdx
        movl  0(%r13,%rdx,4), %ecx
        imull %r15d, %ecx
        addl  %ecx, (%rbx,%rdx,4)
        leal  1(%rax), %edx
        cmpl  %edx, %r12d
        jbe   .L21
        movslq        %edx, %rdx
        addl  $2, %eax
        movl  0(%r13,%rdx,4), %ecx
        imull %r15d, %ecx
        addl  %ecx, (%rbx,%rdx,4)
        cmpl  %eax, %r12d
        jbe   .L21
        movslq        %eax, %rdx
        imull 0(%r13,%rdx,4), %r15d
        addl  %r15d, (%rbx,%rdx,4)
.L21:
        leaq  32(%rsp), %rsi
        xorl  %edi, %edi
        call  clock_gettime
```