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):
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz

Sistema operativo utilizado: *Linux version 4.4.0-43-Microsoft*

Versión de gcc utilizada: *gcc version 5.4.0*

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

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

   1.1 Modifique 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.

   1.2 Genere 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

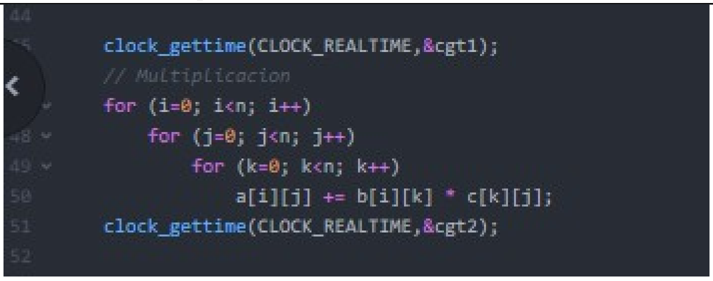
Figura 1 . Código C++ que suma dos vectores

```
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`



```
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);
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

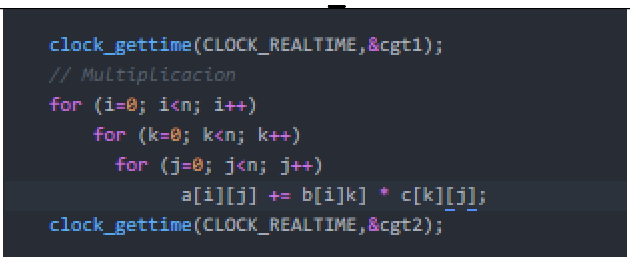
Modificación a) –explicación-: Cambiar los bucles j y k para mejorar accesos a memoria

Modificación b) –explicación-: Cambiamos las matrices por vectores

...

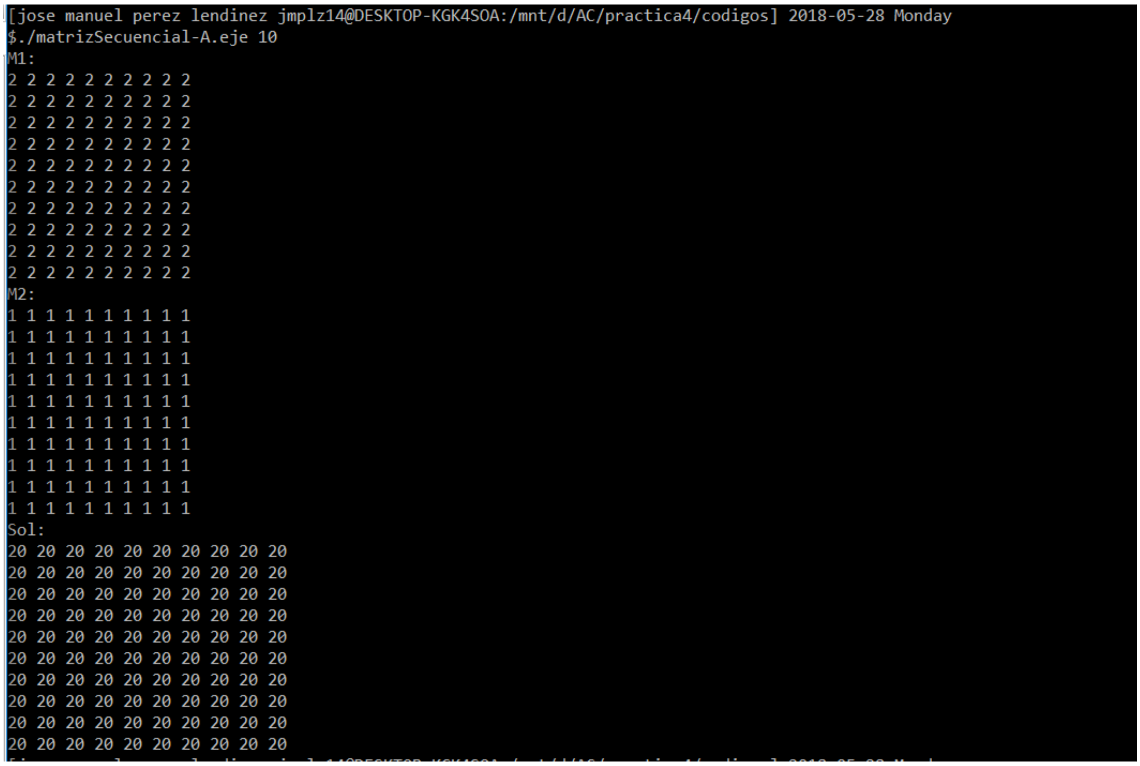
1.1. CÓDIGOS FUENTE MODIFICACIONES

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



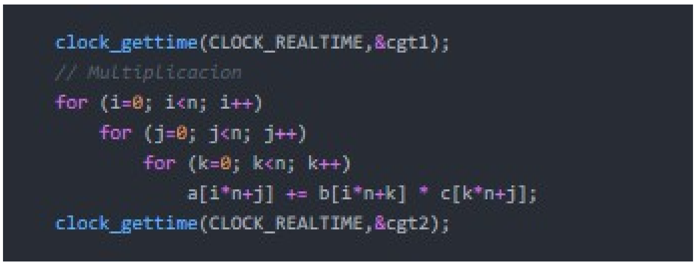
```
clock_gettime(CLOCK_REALTIME,&cgt1);
// Multiplicacion
for (i=0; i<n; i++)
    for (k=0; k<n; k++)
      for (j=0; j<n; j++)
            a[i][j] += b[i]k] * c[k][j];
clock_gettime(CLOCK_REALTIME,&cgt2);
```

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



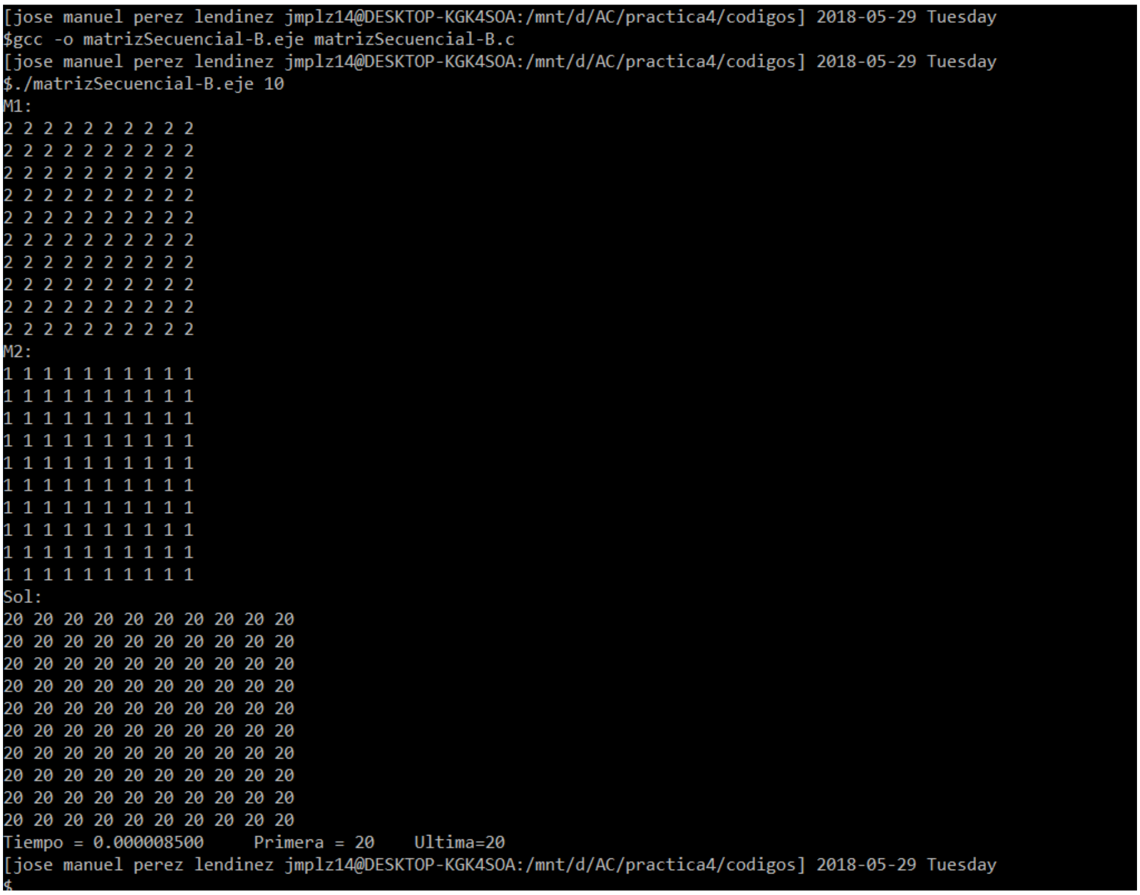
```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-28 Monday
$./matrizSecuencial-A.eje 10
M1:
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
M2:
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
Sol:
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
```

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



```
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*n+j] += b[i*n+k] * c[k*n+j];
clock_gettime(CLOCK_REALTIME,&cgt2);
```

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



```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$gcc -o matrizSecuencial-B.eje matrizSecuencial-B.c
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$./matrizSecuencial-B.eje 10
M1:
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
M2:
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
Sol:
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20
Tiempo = 0.000008500     Primera = 20    Ultima=20
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$
```

1.1. TIEMPOS:(ejecuto con tamaño 1000)

Fichero	-o0	-o1	-o2	-o3	-oS
Sin modificar	10,65	3,42	2,00	2,02	3,41
Modificacion A	6,03	1,97	1,18	1,47	1,89
Modificacion B	5,97	1,17	1,17	1,21	1,23

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

La mejora del apartado B es superiormente mejor que la A aunque con muy poca diferencia.

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`

```
	call	clock_gettime
	movl	$0, -112(%rbp)
	jmp	.L14
.L19:
	movl	$0, -108(%rbp)
	jmp	.L15
```

`pmm-secuencial-modificado_a.s`

```
	call	clock_gettime
	movl	$0, -112(%rbp)
	jmp	.L14
.L19:
	movl	$0, -104(%rbp)
	jmp	.L15
```

`pmm-secuencial-modificado_b.s`

```
	call	clock_gettime
	movl	$0, -96(%rbp)
	jmp	.L10
.L15:
	movl	$0, -88(%rbp)
	jmp	.L11
```

```
.L18:
        movl    $0, -104(%rbp)
        jmp     .L16
.L17:
        movl    -112(%rbp), %eax
        leaq    0(,%rax,8), %rdx
        movq    -96(%rbp), %rax
        addq    %rdx, %rax
        movq    (%rax), %rax
        movl    -108(%rbp), %edx
        salq    $2, %rdx
        addq    %rdx, %rax
        movl    -112(%rbp), %edx
        leaq    0(,%rdx,8), %rcx
        movq    -96(%rbp), %rdx
        addq    %rcx, %rdx
        movq    (%rdx), %rdx
        movl    -108(%rbp), %ecx
        salq    $2, %rcx
        addq    %rcx, %rdx
        movl    (%rdx), %ecx
        movl    -112(%rbp), %edx
        leaq    0(,%rdx,8), %rsi
        movq    -88(%rbp), %rdx
        addq    %rsi, %rdx
        movq    (%rdx), %rdx
        movl    -104(%rbp), %esi
        salq    $2, %rsi
        addq    %rsi, %rdx
        movl    (%rdx), %esi
        movl    -104(%rbp), %edx
        leaq    0(,%rdx,8), %rdi
        movq    -80(%rbp), %rdx
        addq    %rdi, %rdx
        movq    (%rdx), %rdx
        movl    -108(%rbp), %edi
        salq    $2, %rdi
        addq    %rdi, %rdx
        movl    (%rdx), %edx
        imull   %esi, %edx
        addl
```

```
.L18:
        movl    $0, -108(%rbp)
        jmp     .L16
.L17:
        movl    -112(%rbp), %eax
        leaq    0(,%rax,8), %rdx
        movq    -96(%rbp), %rax
        addq    %rdx, %rax
        movq    (%rax), %rax
        movl    -108(%rbp), %edx
        salq    $2, %rdx
        addq    %rdx, %rax
        movl    -112(%rbp), %edx
        leaq    0(,%rdx,8), %rcx
        movq    -96(%rbp), %rdx
        addq    %rcx, %rdx
        movq    (%rdx), %rdx
        movl    -108(%rbp), %ecx
        salq    $2, %rcx
        addq    %rcx, %rdx
        movl    (%rdx), %ecx
        movl    -112(%rbp), %edx
        leaq    0(,%rdx,8), %rsi
        movq    -88(%rbp), %rdx
        addq    %rsi, %rdx
        movq    (%rdx), %rdx
        movl    -104(%rbp), %esi
        salq    $2, %rsi
        addq    %rsi, %rdx
        movl    (%rdx), %esi
        movl    -104(%rbp), %edx
        leaq    0(,%rdx,8), %rdi
        movq    -80(%rbp), %rdx
        addq    %rdi, %rdx
        movq    (%rdx), %rdx
```

```
.L14:
        movl    $0, -92(%rbp)
        jmp     .L12
.L13:
        movl    -96(%rbp), %eax
        imull   -84(%rbp), %eax
        movl    %eax, %edx
        movl    -92(%rbp), %eax
        addl    %edx, %eax
        movl    %eax, %eax
        leaq    0(,%rax,4), %rdx
        movq    -64(%rbp), %rax
        addq    %rax, %rdx
        movl    -96(%rbp), %eax
        imull   -84(%rbp), %eax
        movl    %eax, %ecx
        movl    -92(%rbp), %eax
        addl    %ecx, %eax
        movl    %eax, %eax
        leaq    0(,%rax,4), %rcx
        movq    -64(%rbp), %rax
        addq    %rcx, %rax
        movl    (%rax), %ecx
        movl    -96(%rbp), %eax
        imull   -84(%rbp), %eax
        movl    %eax, %esi
        movl    -88(%rbp), %eax
        addl    %esi, %eax
        movl    %eax, %eax
        leaq    0(,%rax,4), %rsi
        movq    -72(%rbp), %rax
        addq    %rsi, %rax
        movl    (%rax), %esi
```

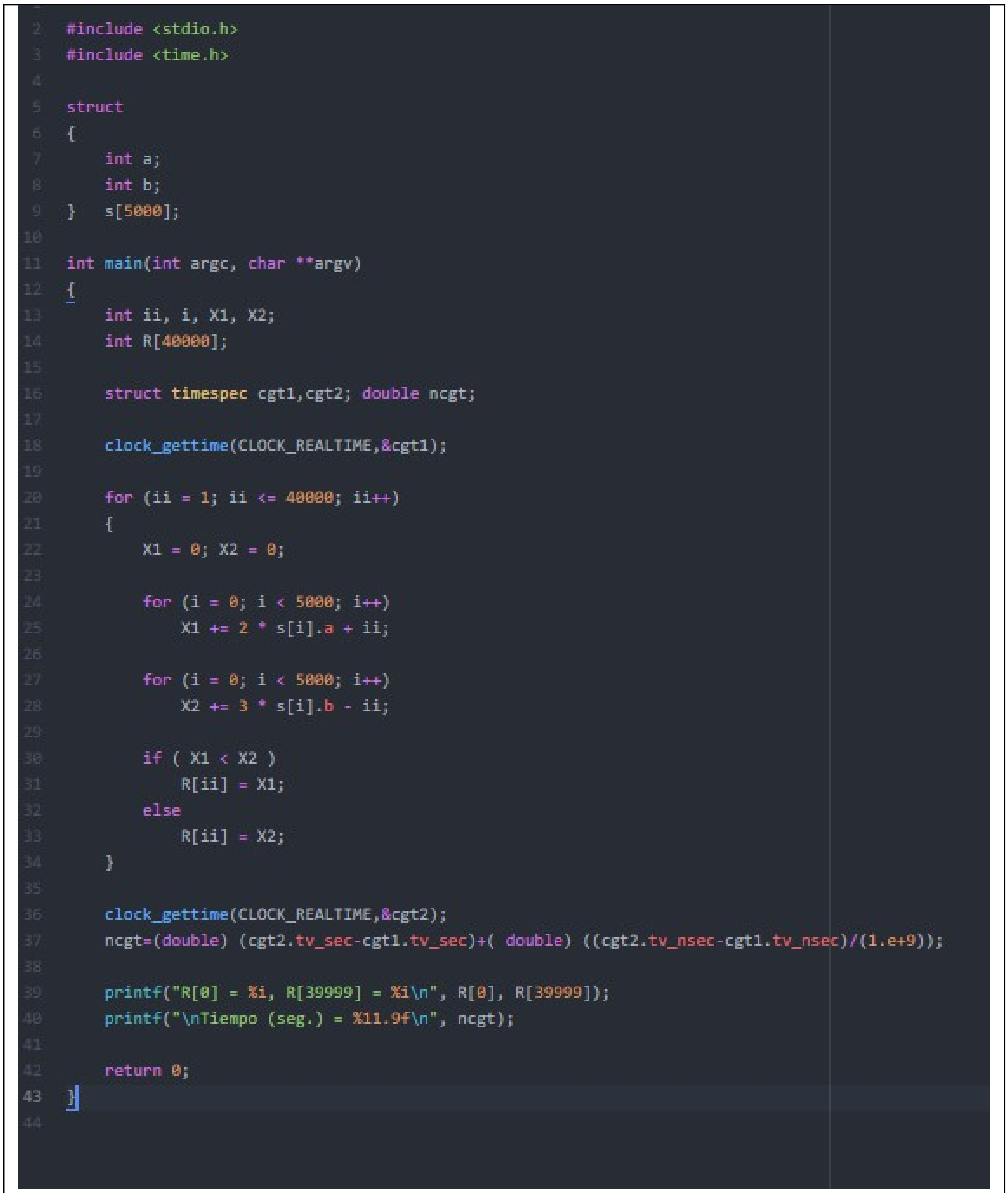
```
                 %ecx, %edx
                 movl
                 %edx, (%rax)
                 addl
                 $1, -104(%rbp)
.L16:
                 movl
                 -104(%rbp), %eax
                 cmpl
                 -100(%rbp), %eax
                 jb
                 .L17
                 addl
                 $1, -108(%rbp)
.L15:
                 movl
                 -108(%rbp), %eax
                 cmpl
                 -100(%rbp), %eax
                 jb
                 .L18
                 addl
                 $1, -112(%rbp)
.L14:
                 movl
                 -112(%rbp), %eax
                 cmpl
                 -100(%rbp), %eax
                 jb
                 .L19
                 leaq
                 -48(%rbp), %rax
                 movq
                 %rax, %rsi
                 movl
                 $0, %edi
                 call
                 clock_gettime
```

```
                 movl
                 -108(%rbp),
%edi
                 salq
                 $2, %rdi
                 addq
                 %rdi, %rdx
                 movl
                 (%rdx), %edx
                 imull
                 %esi, %edx
                 addl
                 %ecx, %edx
                 movl
                 %edx, (%rax)
                 addl
                 $1, -108(%rbp)
.L16:
                 movl
                 -108(%rbp),
%eax
                 cmpl
                 -100(%rbp),
%eax
                 jb
                 .L17
                 addl
                 $1, -104(%rbp)
.L15:
                 movl
                 -104(%rbp),
%eax
                 cmpl
                 -100(%rbp),
%eax
                 jb
                 .L18
                 addl
                 $1, -112(%rbp)
.L14:
                 movl
                 -112(%rbp),
%eax
                 cmpl
                 -100(%rbp),
%eax
                 jb
                 .L19
                 leaq
                 -48(%rbp), %rax
                 movq
                 %rax, %rsi
                 movl
                 $0, %edi
                 call
                 clock_gettime
```

```
                 movl
                 -88(%rbp),
%eax
                 imull
                 -84(%rbp),
%eax
                 movl
                 %eax, %edi
                 movl
                 -92(%rbp),
%eax
                 addl
                 %edi, %eax
                 movl
                 %eax, %eax
                 leaq
                 0(,%rax,4),
%rdi
                 movq
                 -80(%rbp),
%rax
                 addq
                 %rdi, %rax
                 movl
                 (%rax), %eax
                 imull
                 %esi, %eax
                 addl
                 %ecx, %eax
                 movl
                 %eax, (%rdx)
                 addl
                 $1, -92(%rbp)
.L12:
                 movl
                 -92(%rbp),
%eax
                 cmpl
                 -84(%rbp),
%eax
                 jb
                 .L13
                 addl
                 $1, -88(%rbp)
.L11:
                 movl
                 -88(%rbp),
%eax
                 cmpl
                 -84(%rbp),
%eax
                 jb
                 .L14
                 addl
                 $1, -96(%rbp)
.L10:
                 movl
                 -96(%rbp),
%eax
                 cmpl
                 -84(%rbp),
%eax
                 jb
                 .L15
                 leaq
                 -32(%rbp),
%rax
                 movq
                 %rax, %rsi
                 movl
                 $0, %edi
```

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`



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

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

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 = 1; 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("R[0] = %i, R[39999] = %i\n", R[0], R[39999]);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    return 0;
}
```

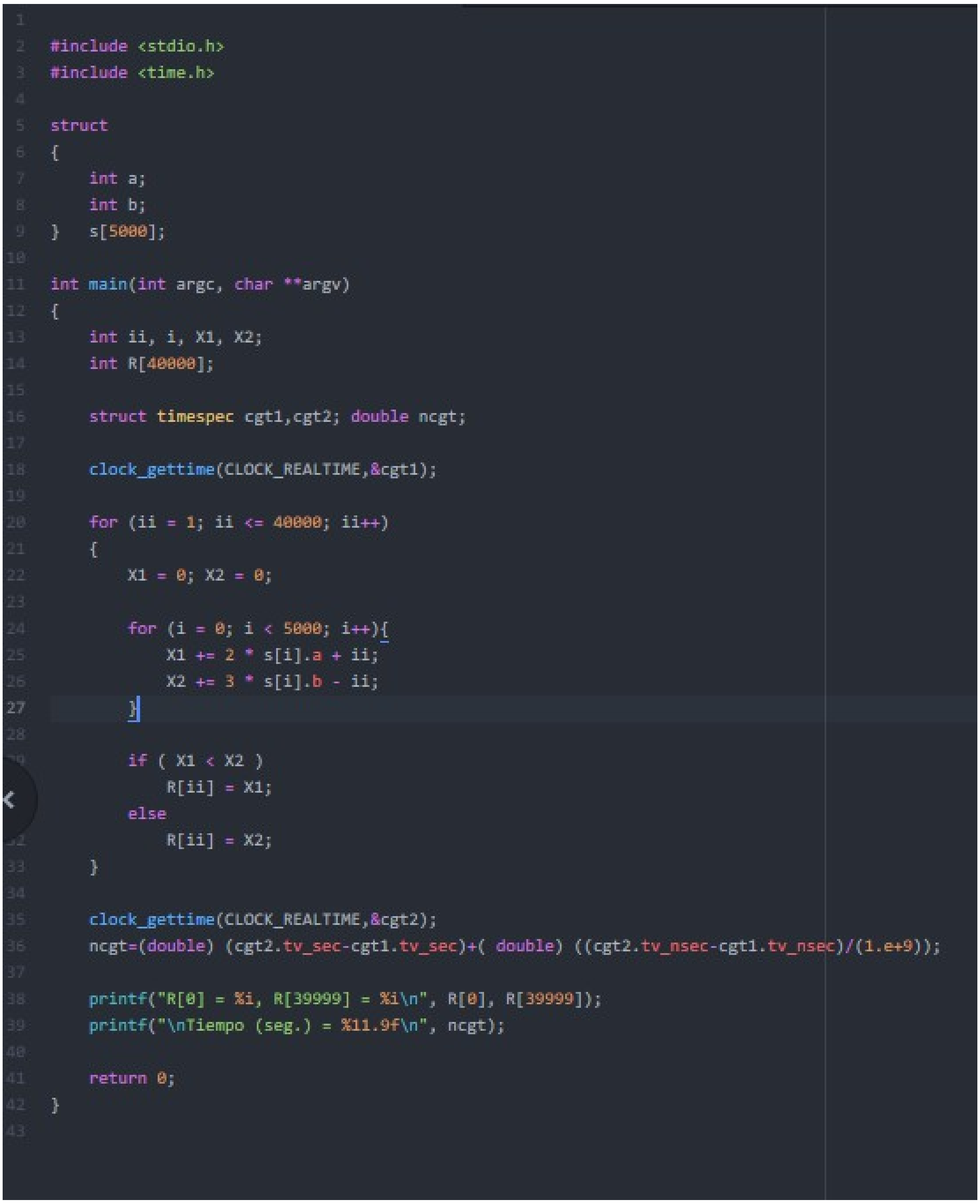
1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Se elimina un for de los dos.

Modificación b) –explicación-: Se realizan 4 operaciones por cada pasada del for en vez de 1.

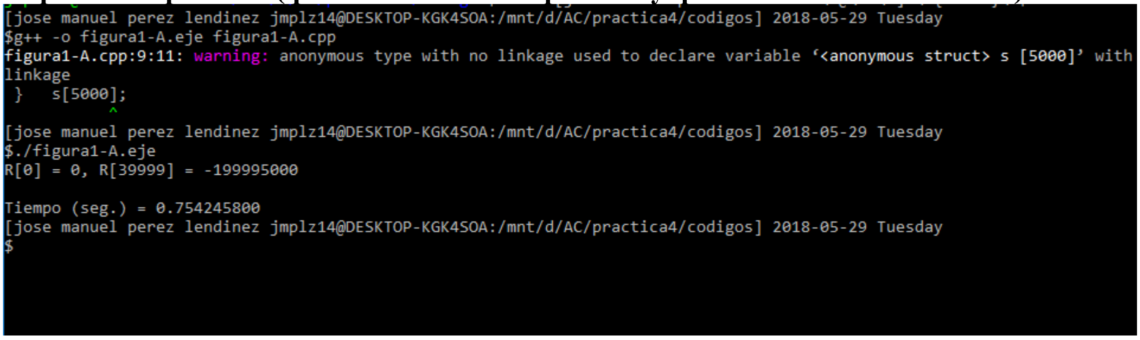
1.1. CÓDIGOS FUENTE MODIFICACIONES

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



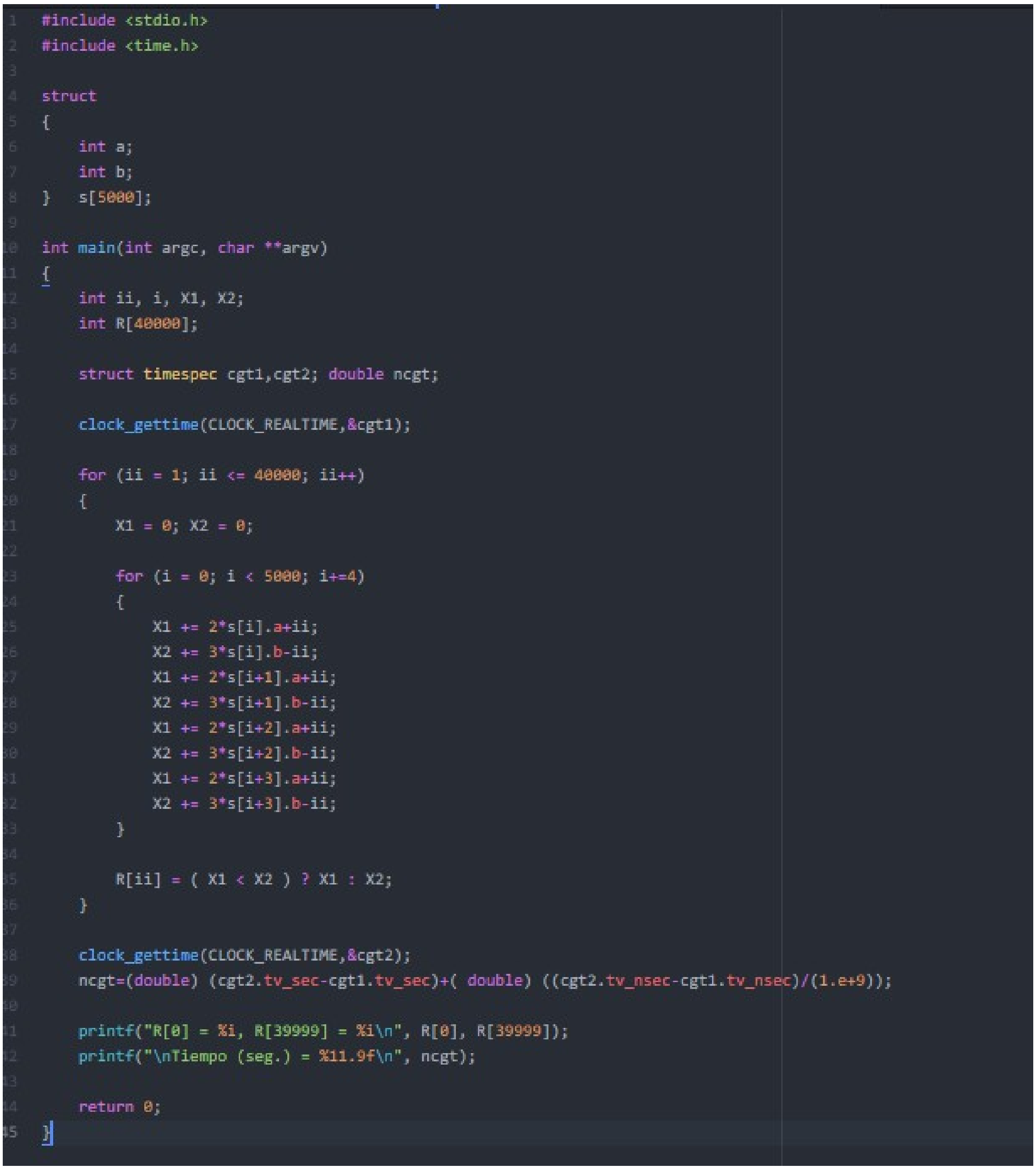
```c

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

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

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 = 1; 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("R[0] = %i, R[39999] = %i\n", R[0], R[39999]);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    return 0;
}

```

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



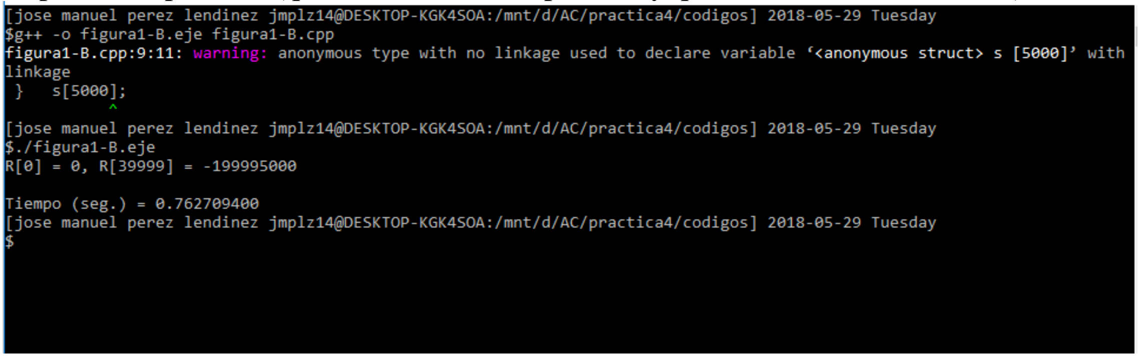
```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$g++ -o figura1-A.eje figura1-A.cpp
figura1-A.cpp:9:11: warning: anonymous type with no linkage used to declare variable ‘<anonymous struct> s [5000]’ with
linkage
 }   s[5000];
           ^

[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$./figura1-A.eje
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.754245800
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$
```

b) Captura figura1-modificado_b.c



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

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

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 = 1; 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;
        }

        R[ii] = ( X1 < X2 ) ? X1 : 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("R[0] = %i, R[39999] = %i\n", R[0], R[39999]);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    return 0;
}
```

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



```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$g++ -o figura1-B.eje figura1-B.cpp
figura1-B.cpp:9:11: warning: anonymous type with no linkage used to declare variable ‘<anonymous struct> s [5000]’ with
linkage
 }   s[5000];
           ^

[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$./figura1-B.eje
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.762709400
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$
```

1.1. TIEMPOS:

Fichero	-o0	-o1	-o2	-o3
Sin modificar	1,20	0,18	0,00017	0,00018
Modificacion A	0,75	0,092	0,00018	0,00019
Modificacion B	0,75	0,093	0,00017	0,00019

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Como se ve apartir de -O2 no cambia mucho

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

```
                 call
                 clock_gettime
                 movl
                 $1, -160072(%rbp)
.L9:
                 cmpl
                 $40000,
-160072(%rbp)
                 jg
                 .L2
                 movl
                 $0, -160064(%rbp)
                 movl
                 $0, -160060(%rbp)
                 movl
                 $0, -160068(%rbp)
.L4:
                 cmpl
                 $4999,
-160068(%rbp)
                 jg
                 .L3
                 movl              -
160068(%rbp), %eax
                 cltq
                 movl
                 s(,%rax,8), %eax
                 leal
                 (%rax,%rax), %edx
                 movl              -
160072(%rbp), %eax
                 addl
                 %edx, %eax
                 addl
                 %eax,
-160064(%rbp)
                 addl
                 $1, -160068(%rbp)
                 jmp
                 .L4
.L3:
                 movl
                 $0, -160068(%rbp)
.L6:
                 cmpl
                 $4999,
-160068(%rbp)
                 jg
                 .L5
```

pmm-secuencial-modificado_b.s

```
Call clock_gettime
                 movl
                 $1,
-160072(%rbp)
.L7:
                 cmpl
                 $40000,
-160072(%rbp)
                 jg
                 .L2
                 movl
                 $0,
-160064(%rbp)
                 movl
                 $0,
-160060(%rbp)
                 movl
                 $0,
-160068(%rbp)
.L4:
                 cmpl
                 $4999,
-160068(%rbp)
                 jg
                 .L3
                 movl
                 -
160068(%rbp), %eax
                 cltq
                 movl
                 s(,%rax,8),
%eax
                 leal
                 (%rax,%rax),
%edx
                 movl
                 -
160072(%rbp), %eax
                 addl
                 %edx, %eax
                 addl
                 %eax,
-160064(%rbp)
                 movl
                 -
160068(%rbp), %eax
                 cltq
                 movl
                 s+4(,
```

pmm-secuencial-modificado_c.s

```
call             clock_gettime
                 movl
                 $1,
-160072(%rbp)
.L7:
                 cmpl
                 $40000,
-160072(%rbp)
                 jg
                 .L2
                 movl
                 $0,
-160064(%rbp)
                 movl
                 $0,
-160060(%rbp)
                 movl
                 $0,
-160068(%rbp)
.L4:
                 cmpl
                 $4999,
-160068(%rbp)
                 jg
                 .L3
                 movl
                 -160068(%rbp),
%eax
                 cltq
                 movl
                 s(,%rax,8),
%eax
                 leal
                 (%rax,%rax),
%edx
                 movl
                 -160072(%rbp),
%eax
                 addl
                 %edx, %eax
                 addl
                 %eax,
-160064(%rbp)
                 movl
                 -160068(%rbp),
%eax
                 cltq
                 movl
                 s+4(,%rax,8),
```

```
                movl            -
160068(%rbp), %eax
                cltq
                movl
                s+4(,%rax,8), %edx
                movl
                %edx, %eax
                addl
                %eax, %eax
                addl
                %edx, %eax
                subl            -
160072(%rbp), %eax
                addl
                %eax,
-160060(%rbp)
                addl
                $1, -160068(%rbp)
                jmp
                .L6
.L5:
                movl            -
160064(%rbp), %eax
                cmpl            -
160060(%rbp), %eax
                jge
                .L7
                movl            -
160072(%rbp), %eax
                cltq
                movl            -
160064(%rbp), %edx
                movl
                %edx,
-160016(%rbp,%rax,4)
                jmp
                .L8
.L7:
                movl            -
160072(%rbp), %eax
                cltq
                movl            -
160060(%rbp), %edx
                movl
                %edx,
-160016(%rbp,%rax,4)
.L8:
                addl
                $1, -160072(%rbp)
                jmp
                .L9
.L2:
                leaq            -
160032(%rbp), %rax
                movq
                %rax, %rsi
                movl
                $0, %edi
                call
                clock_gettime
```

```
%rax,8), %edx
                movl
                %edx, %eax
                addl
                %eax, %eax
                addl
                %edx, %eax
                subl
                -
160072(%rbp), %eax
                addl
                %eax,
-160060(%rbp)
                addl
                $1,
-160068(%rbp)
                jmp
                .L4
.L3:
                movl
                -
160064(%rbp), %eax
                cmpl
                -
160060(%rbp), %eax
                jge
                .L5
                movl
                -
160072(%rbp), %eax
                cltq
                movl
                -
160064(%rbp), %edx
                movl
                %edx,
-160016(%rbp,%rax,4)
                jmp
                .L6
.L5:
                movl
                -
160072(%rbp), %eax
                cltq
                movl
                -
160060(%rbp), %edx
                movl
                %edx,
-160016(%rbp,%rax,4)
.L6:
                addl
                $1,
-160072(%rbp)
                jmp
                .L7
.L2:
                leaq
                -
160032(%rbp), %rax
                movq
                %rax, %rsi
                movl
                $0, %edi
                call

clock_gettime
```

```
%edx
                movl
                %edx, %eax
                addl
                %eax, %eax
                addl
                %edx, %eax
                subl
                -160072(%rbp),
%eax
                addl
                %eax,
-160060(%rbp)
                addl
                $1,
-160068(%rbp)
                jmp
                .L4
.L3:
                movl
                -160064(%rbp),
%eax
                cmpl
                -160060(%rbp),
%eax
                jge
                .L5
                movl
                -160072(%rbp),
%eax
                cltq
                movl
                -160064(%rbp),
%edx
                movl
                %edx,
-160016(%rbp,%rax,4)
                jmp
                .L6
.L5:
                movl
                -160072(%rbp),
%eax
                cltq
                movl
                -160060(%rbp),
%edx
                movl
                %edx,
-160016(%rbp,%rax,4)
.L6:
                addl
                $1,
-160072(%rbp)
                jmp
                .L7
.L2:
                leaq
                -160032(%rbp),
%rax
                movq
                %rax, %rsi
                movl
                $0, %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

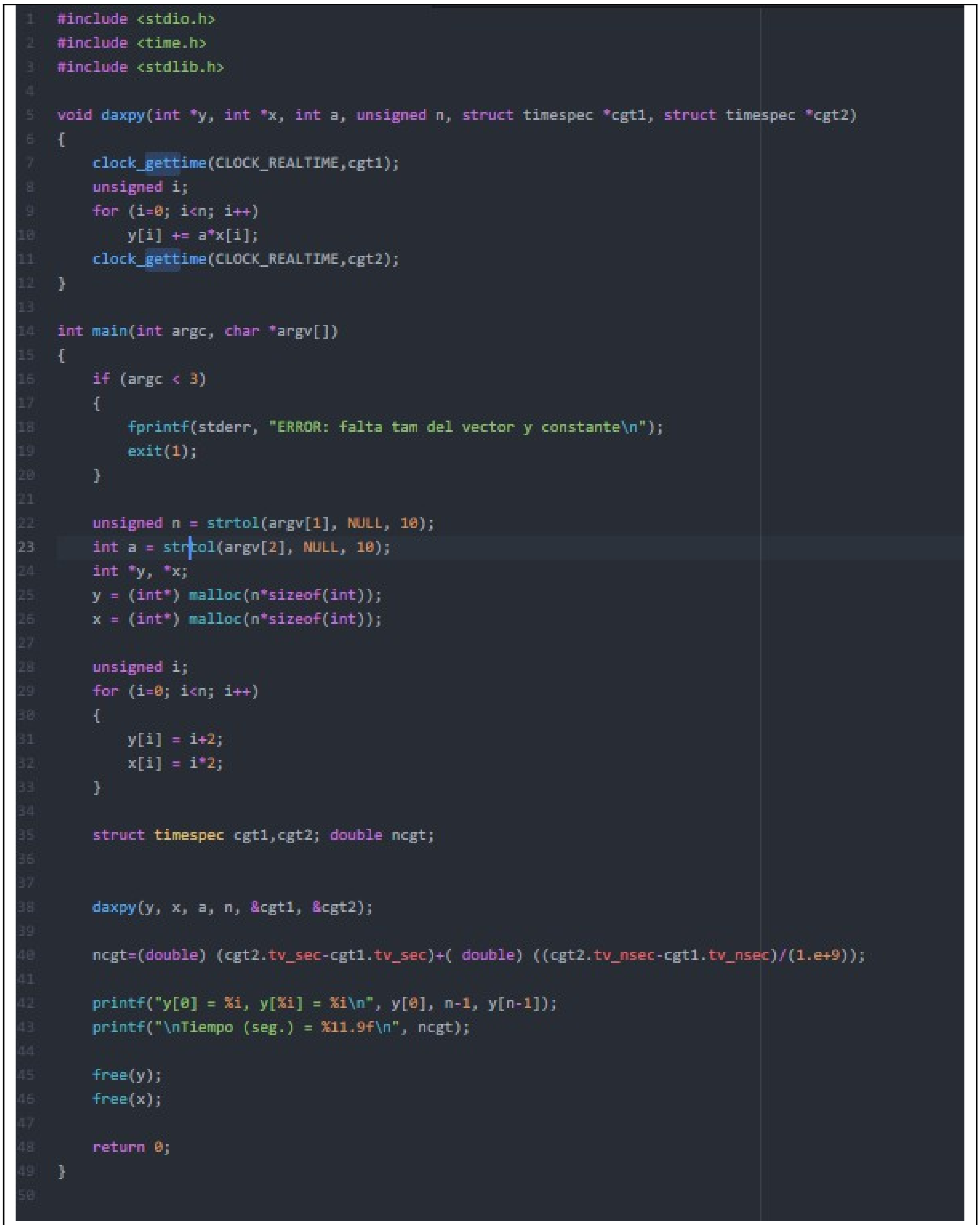
programa es una rutina denominada `DAXPY` (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

```
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`

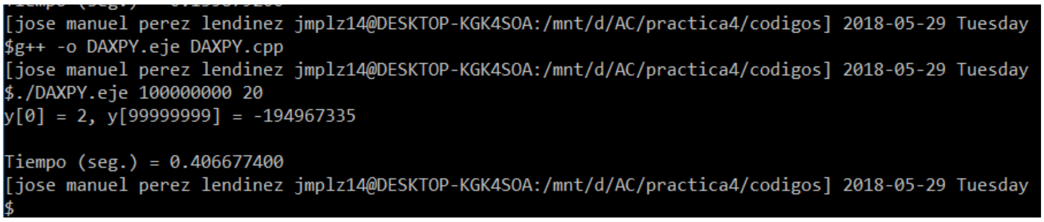


```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

void daxpy(int *y, int *x, int a, unsigned n, struct timespec *cgt1, struct timespec *cgt2)
{
    clock_gettime(CLOCK_REALTIME,cgt1);
    unsigned i;
    for (i=0; i<n; i++)
        y[i] += a*x[i];
    clock_gettime(CLOCK_REALTIME,cgt2);
}

int main(int argc, char *argv[])
{
    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);
    int *y, *x;
    y = (int*) malloc(n*sizeof(int));
    x = (int*) malloc(n*sizeof(int));

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

    struct timespec cgt1,cgt2; double ncgt;


    daxpy(y, x, a, n, &cgt1, &cgt2);

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

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

    free(y);
    free(x);

    return 0;
}
```

Tiempos ejec.	**-O0**	**-Os**	**-O2**	**-O3**
	0,3619	0,1358	0,1460	0,1653

N = 100000000

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

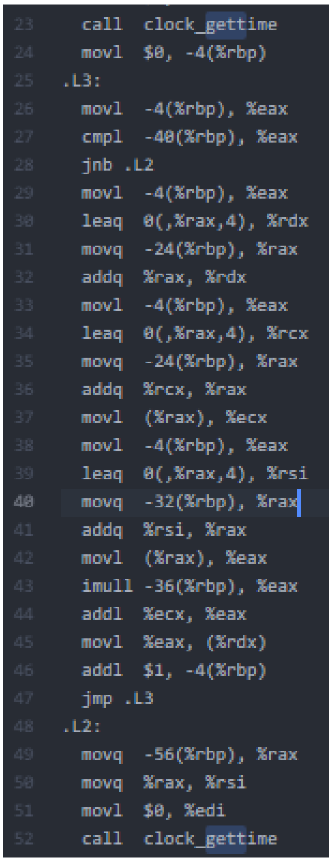


```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$g++ -o DAXPY.eje DAXPY.cpp
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$./DAXPY.eje 100000000 20
y[0] = 2, y[99999999] = -194967335

Tiempo (seg.) = 0.406677400
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica4/codigos] 2018-05-29 Tuesday
$
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:
En O0 se usan direcciones dke pila y con O2 registros. O3 desarrolla el bucle alargando el código ensamblador.

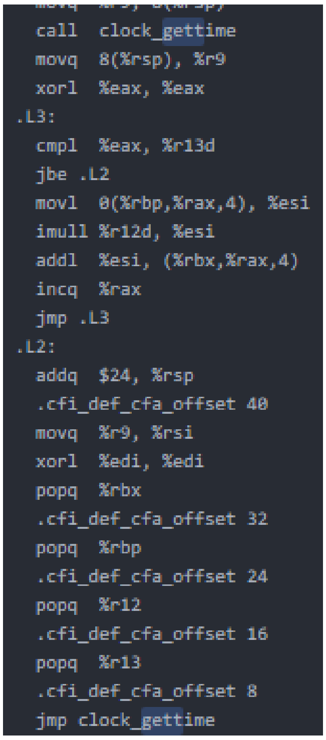
CÓDIGO EN ENSAMBLADOR (no es necesario introducir aquí el código como captura de pantalla, ajustar el tamaño de la letra para que una instrucción no ocupe más de un renglón)**:**
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO)

daxpyO0.s



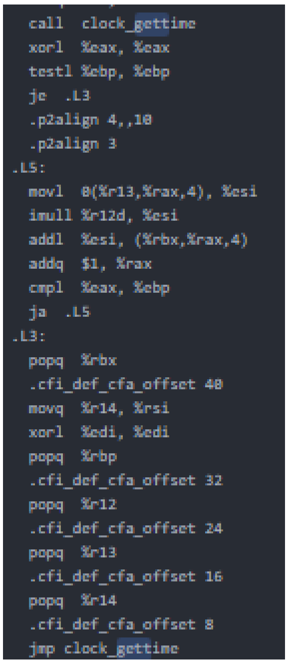
```
  call  clock_gettime
  movl  $0, -4(%rbp)
.L3:
  movl  -4(%rbp), %eax
  cmpl  -40(%rbp), %eax
  jnb .L2
  movl  -4(%rbp), %eax
  leaq  0(,%rax,4), %rdx
  movq  -24(%rbp), %rax
  addq  %rax, %rdx
  movl  -4(%rbp), %eax
  leaq  0(,%rax,4), %rcx
  movq  -24(%rbp), %rax
  addq  %rcx, %rax
  movl  (%rax), %ecx
  movl  -4(%rbp), %eax
  leaq  0(,%rax,4), %rsi
  movq  -32(%rbp), %rax
  addq  %rsi, %rax
  movl  (%rax), %eax
  imull -36(%rbp), %eax
  addl  %ecx, %eax
  movl  %eax, (%rdx)
  addl  $1, -4(%rbp)
  jmp .L3
.L2:
  movq  -56(%rbp), %rax
  movq  %rax, %rsi
  movl  $0, %edi
  call  clock_gettime
```

daxpyOs.s



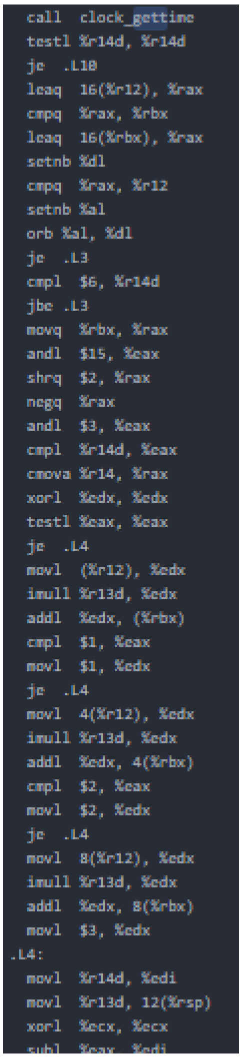
```
  call  clock_gettime
  movq  8(%rsp), %r9
  xorl  %eax, %eax
.L3:
  cmpl  %eax, %r13d
  jbe .L2
  movl  0(%rbp,%rax,4), %esi
  imull %r12d, %esi
  addl  %esi, (%rbx,%rax,4)
  incq  %rax
  jmp .L3
.L2:
  addq  $24, %rsp
  .cfi_def_cfa_offset 40
  movq  %r9, %rsi
  xorl  %edi, %edi
  popq  %rbx
  .cfi_def_cfa_offset 32
  popq  %rbp
  .cfi_def_cfa_offset 24
  popq  %r12
  .cfi_def_cfa_offset 16
  popq  %r13
  .cfi_def_cfa_offset 8
  jmp clock_gettime
```

daxpyO2.s

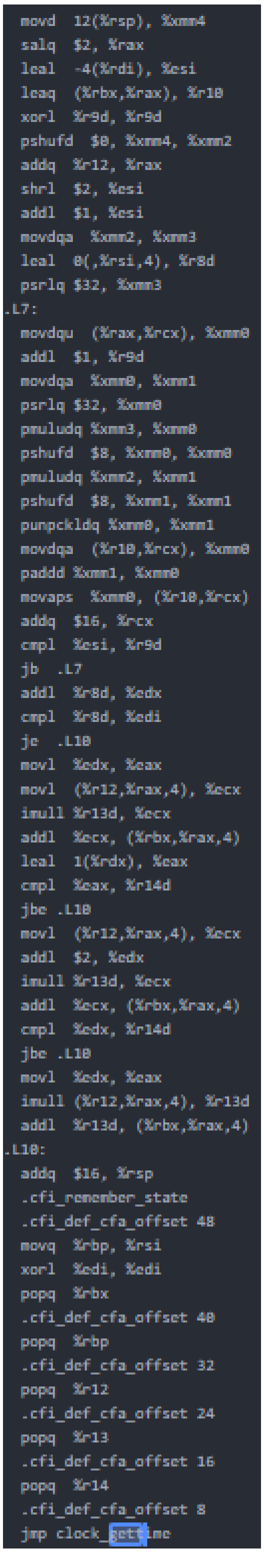


```
  call  clock_gettime
  xorl  %eax, %eax
  testl %ebp, %ebp
  je  .L3
  .p2align 4,,10
  .p2align 3
.L5:
  movl  0(%r13,%rax,4), %esi
  imull %r12d, %esi
  addl  %esi, (%rbx,%rax,4)
  addq  $1, %rax
  cmpl  %eax, %ebp
  ja  .L5
.L3:
  popq  %rbx
  .cfi_def_cfa_offset 40
  movq  %r14, %rsi
  xorl  %edi, %edi
  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
  jmp clock_gettime
```

daxpyO3.s



```
  call  clock_gettime
  testl %r14d, %r14d
  je  .L10
  leaq  16(%r12), %rax
  cmpq  %rax, %rbx
  leaq  16(%rbx), %rax
  setnb %dl
  cmpq  %rax, %r12
  setnb %al
  orb %al, %dl
  je  .L3
  cmpl  $6, %r14d
  jbe .L3
  movq  %rbx, %rax
  andl  $15, %eax
  shrq  $2, %rax
  negq  %rax
  andl  $3, %eax
  cmpl  %r14d, %eax
  cmova %r14, %rax
  xorl  %edx, %edx
  testl %eax, %eax
  je  .L4
  movl  (%r12), %edx
  imull %r13d, %edx
  addl  %edx, (%rbx)
  cmpl  $1, %eax
  movl  $1, %edx
  je  .L4
  movl  4(%r12), %edx
  imull %r13d, %edx
  addl  %edx, 4(%rbx)
  cmpl  $2, %eax
  movl  $2, %edx
  je  .L4
  movl  8(%r12), %edx
  imull %r13d, %edx
  addl  %edx, 8(%rbx)
  movl  $3, %edx
.L4:
  movl  %r14d, %edi
  movl  %r13d, 12(%rsp)
  xorl  %ecx, %ecx
  subl  %eax, %edi
```

```
  movd  12(%rsp), %xmm4
  salq  $2, %rax
  leal  -4(%rdi), %esi
  leaq  (%rbx,%rax), %r10
  xorl  %r9d, %r9d
  pshufd  $0, %xmm4, %xmm2
  addq  %r12, %rax
  shrl  $2, %esi
  addl  $1, %esi
  movdqa  %xmm2, %xmm3
  leal  0(,%rsi,4), %r8d
  psrlq $32, %xmm3
.L7:
  movdqu  (%rax,%rcx), %xmm0
  addl  $1, %r9d
  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,%rcx), %xmm0
  paddd %xmm1, %xmm0
  movaps  %xmm0, (%r10,%rcx)
  addq  $16, %rcx
  cmpl  %esi, %r9d
  jb  .L7
  addl  %r8d, %edx
  cmpl  %r8d, %edi
  je  .L10
  movl  %edx, %eax
  movl  (%r12,%rax,4), %ecx
  imull %r13d, %ecx
  addl  %ecx, (%rbx,%rax,4)
  leal  1(%rdx), %eax
  cmpl  %eax, %r14d
  jbe .L10
  movl  (%r12,%rax,4), %ecx
  addl  $2, %edx
  imull %r13d, %ecx
  addl  %ecx, (%rbx,%rax,4)
  cmpl  %edx, %r14d
  jbe .L10
  movl  %edx, %eax
  imull (%r12,%rax,4), %r13d
  addl  %r13d, (%rbx,%rax,4)
.L10:
  addq  $16, %rsp
  .cfi_remember_state
  .cfi_def_cfa_offset 48
  movq  %rbp, %rsi
  xorl  %edi, %edi
  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
  jmp clock_gettime
```