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): Jose Luis Pedraza Román
Grupo de prácticas: A3
Fecha de entrega: 24/6/2018
Fecha evaluación en clase: 25/5/2018

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *(respuesta)*

Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz

Sistema operativo utilizado: *(respuesta)*

ejecuto: lsb_release -a

Distributor ID: Ubuntu

Description: Ubuntu 16.04.4 LTS

Release: 16.04

Codename: xenial

Versión de gcc utilizada: *(respuesta)*

Ejecuto:
```
gcc --version
```

gcc (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609

Copyright (C) 2015 Free Software Foundation, Inc.

This is free software; see the source for copying conditions. There is NO

warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

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

```
josele@josele:~$ PS1="[JoseLuisPedrazaRoman \u@\h:\w] \D{%F %A}\n$"
[JoseLuisPedrazaRoman josele@josele:~] 2018-05-23 miércoles
$lscpu
Arquitectura:          x86_64
modo(s) de operación de las CPUs:32-bit, 64-bit
Orden de bytes:        Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Hilo(s) de procesamiento por núcleo:1
Núcleo(s) por «socket»:4
Socket(s):             1
Modo(s) NUMA:          1
ID de fabricante:      GenuineIntel
Familia de CPU:        6
Modelo:                60
Model name:            Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz
Revisión:              3
CPU MHz:               1099.156
CPU max MHz:           3800,0000
CPU min MHz:           800,0000
BogoMIPS:              6784.63
Virtualización:        VT-x
Caché L1d:             32K
Caché L1i:             32K
Caché L2:              256K
Caché L3:              6144K
NUMA node0 CPU(s):     0-3
Flags:                 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 syscall nx pdpe1gb rdtscp
lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pd
cm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm epb invpcid_single spec_ctrl retpoline kaiser tpr_shadow vnmi flexpriority ep
t vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts
[JoseLuisPedrazaRoman josele@josele:~] 2018-05-23 miércoles
$
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif

//#define TIMES
//#define PRINTF_ALL

int main(int argc, char **argv) {
  // 1. Lectura valores de entrada
  if(argc < 2) {
    fprintf(stderr,"Falta num\n");
    exit(-1);
  }
  int n = atoi(argv[1]);
  int i,j,k;
  struct timespec ini,fin; double transcurrido;

  // 2. Creación e inicialización de vector y matriz
  // 2.1. Creación
  int **A, **B, **C;
  A = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    A[i] = (int*)malloc(n*sizeof(int));

  B = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    B[i] = (int*)malloc(n*sizeof(int));

  C = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    C[i] = (int*)malloc(n*sizeof(int));

  // 2.2. Inicialización
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      B[i][j]=n*i+j;
      C[i][j]=n*i+j;
    }
  }

  // 3. Impresión de vector y matriz
  #ifndef TIMES
    #ifdef PRINTF_ALL
```

```
      printf("Matriz inicial B:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(B[i][j]<10) printf(" %d ",B[i][j]);
          else printf("%d ",B[i][j]);
        }
        printf("\n");
      }
      printf("Matriz inicial C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(C[i][j]<10) printf(" %d ",C[i][j]);
          else printf("%d ",C[i][j]);
        }
        printf("\n");
      }
    #endif
  #endif

  // 4. Cálculo resultado
  clock_gettime(CLOCK_REALTIME,&ini);

  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      A[i][j]=0;
      for (k=0; k<n; k++) {
        A[i][j]+=B[i][k]*C[k][j];
      }
    }
  }

  clock_gettime(CLOCK_REALTIME,&fin);
  transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double)
((fin.tv_nsec-ini.tv_nsec)/(1.e+9));

  // 5. Impresión de vector resultado
  #ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
  #else
    #ifdef PRINTF_ALL
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("Matriz resultado A=B*C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(A[i][j]<10) printf(" %d ",A[i][j]);
          else printf("%d ",A[i][j]);
        }
        printf("\n");
      }
    #else
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("A[0][0]: %d, A[n-1][n-1]: %d\n",A[0][0],A[n-1][n-1]);
    #endif
  #endif

  // 6. Eliminar de memoria
```

```
  free(A);
  free(B);
  free(C);
return(0);
}
```

1.1. MODIFICACIONES REALIZADAS:

Modificación a) –explicación-: DESENRROLLADO DE BUCLES
EN VEZ DE REALIZAR UN SALTO EN CADA ITERACIÓN, SE REALIZARÁ UN DEL BUCLE CADA 5 ITERACIONES REDUCIENDO ASÍ EL NUMERO DE INSTRUCCIONES.

Modificación b) –explicación-: CAMBIA EL ORDEN DE EJECUCIÓN DE LOS BUCLES:
i,k,j EN VEZ DE i,j,k DE ESTA MANERA CONSEGUIMOS ACCEDER A ELEMENTOS CONSECUTIVOS CUANDO SE LLAMA A M[k][j].

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
#endif

//#define TIMES
//#define PRINTF_ALL

int main(int argc, char **argv) {
  // 1. Lectura valores de entrada
  if(argc < 2) {
    fprintf(stderr,"Falta num\n");
    exit(-1);
  }
  int n = atoi(argv[1]);
  if(n%5!=0) {
     fprintf(stderr,"num debe ser divisible entre 5\n");
     exit(-1);
  }
  int i,j,k;
  struct timespec ini,fin; double transcurrido;

  // 2. Creación e inicialización de vector y matriz
  // 2.1. Creación
  int **A, **B, **C;
  A = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    A[i] = (int*)malloc(n*sizeof(int));

  B = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    B[i] = (int*)malloc(n*sizeof(int));

  C = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    C[i] = (int*)malloc(n*sizeof(int));

  // 2.2. Inicialización
```

```
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      B[i][j]=n*i+j;
      C[i][j]=n*i+j;
      A[i][j]=0;
    }
  }

  // 3. Impresión de vector y matriz
  #ifndef TIMES
    #ifdef PRINTF_ALL
      printf("Matriz inicial B:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(B[i][j]<10) printf(" %d ",B[i][j]);
          else printf("%d ",B[i][j]);
        }
        printf("\n");
      }
      printf("Matriz inicial C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(C[i][j]<10) printf(" %d ",C[i][j]);
          else printf("%d ",C[i][j]);
        }
        printf("\n");
      }
    #endif
  #endif

  // 4. Cálculo resultado

  int tmp0,tmp1,tmp2,tmp3,tmp4;

  clock_gettime(CLOCK_REALTIME,&ini);

  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      tmp0=tmp1=tmp2=tmp3=tmp4=0;
      for (k=0; k<n; k=k+5) {
        tmp0=tmp0+B[i][k]*C[k][j];
        tmp1=tmp1+B[i][k+1]*C[k+1][j];
        tmp2=tmp2+B[i][k+2]*C[k+2][j];
        tmp3=tmp3+B[i][k+3]*C[k+3][j];
        tmp4=tmp4+B[i][k+4]*C[k+4][j];
      }
      A[i][j]=tmp0+tmp1+tmp2+tmp3+tmp4;
    }
  }

  clock_gettime(CLOCK_REALTIME,&fin);
  transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

  // 5. Impresión de vector resultado
  #ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
  #else
    #ifdef PRINTF_ALL
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("Matriz resultado A=B*C:\n");
      for (i=0; i<n; i++) {
```

```
        for (j=0; j<n; j++) {
          if(A[i][j]<10) printf(" %d ",A[i][j]);
          else printf("%d ",A[i][j]);
        }
        printf("\n");
      }
    #else
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("A[0][0]: %d, A[n-1][n-1]: %d\n",A[0][0],A[n-1][n-1]);
    #endif
  #endif

  // 6. Eliminar de memoria
  free(A);
  free(B);
  free(C);
return(0);
}
```

Capturas de pantalla :

```
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$gcc -O2 -o pmm-secuencial_mod_a pmm-secuencial-modificado_a.c -lrt
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$./pmm-secuencial 100
Tiempo: 0.002669736
A[0][0]: 32835000, A[n-1][n-1]: 736867754
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$./pmm-secuencial_mod_a 100
Tiempo: 0.002051131
A[0][0]: 32835000, A[n-1][n-1]: 736867754
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$./pmm-secuencial_mod_a 100
Tiempo: 0.001011276
A[0][0]: 32835000, A[n-1][n-1]: 736867754
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$
```

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

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

//#define TIMES
//#define PRINTF_ALL

int main(int argc, char **argv) {
  // 1. Lectura valores de entrada
  if(argc < 2) {
    fprintf(stderr,"Falta num\n");
    exit(-1);
  }
  int n = atoi(argv[1]);
  if(n%5!=0) {
    fprintf(stderr,"num debe ser divisible entre 5\n");
    exit(-1);
  }
  int i,j,k;
```

```
  struct timespec ini,fin; double transcurrido;

  // 2. Creación e inicialización de vector y matriz
  // 2.1. Creación
  int **A, **B, **C;
  A = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    A[i] = (int*)malloc(n*sizeof(int));

  B = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    B[i] = (int*)malloc(n*sizeof(int));

  C = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    C[i] = (int*)malloc(n*sizeof(int));

  // 2.2. Inicialización
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      B[i][j]=n*i+j;
      C[i][j]=n*i+j;
      A[i][j]=0;
    }
  }

  // 3. Impresión de vector y matriz
  #ifndef TIMES
    #ifdef PRINTF_ALL
      printf("Matriz inicial B:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(B[i][j]<10) printf(" %d ",B[i][j]);
          else printf("%d ",B[i][j]);
        }
        printf("\n");
      }
      printf("Matriz inicial C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(C[i][j]<10) printf(" %d ",C[i][j]);
          else printf("%d ",C[i][j]);
        }
        printf("\n");
      }
    #endif
  #endif

  // 4. Cálculo resultado
  clock_gettime(CLOCK_REALTIME,&ini);

  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,&fin);
  transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));
```

```
  // 5. Impresión de vector resultado
  #ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
  #else
    #ifdef PRINTF_ALL
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("Matriz resultado A=B*C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(A[i][j]<10) printf(" %d ",A[i][j]);
          else printf("%d ",A[i][j]);
        }
        printf("\n");
      }
    #else
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("A[0][0]: %d, A[n-1][n-1]: %d\n",A[0][0],A[n-1][n-1]);
    #endif
  #endif

  // 6. Eliminar de memoria
  free(A);
  free(B);
  free(C);
return(0);
}
```

Capturas de pantalla:

```
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$gcc -O2 -o pmm-secuencial_mod_b pmm-secuencial-modificado_b.c -lrt
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$./pmm-secuencial_mod_b 100
Tiempo: 0.000705620
A[0][0]: 32835000, A[n-1][n-1]: 736867754
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$./pmm-secuencial 100
Tiempo: 0.002325162
A[0][0]: 32835000, A[n-1][n-1]: 736867754
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$
```

1.1. TIEMPOS: tam 100

Modificación	**-O2**
Sin modificar	*0,02325162*
Modificación a)	0,001011276
Modificación b)	0,000705620

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES :

pmm-secuencial.s

```
call            clock_gettime
                xorl
                %r11d, %r11d
.L16:
                movq
                (%r12,%r11,8),
%r10
                movq
                0(%rbp,%r11,8),
%rdi
                xorl
                %r9d, %r9d
                .p2align 4,,10
                .p2align 3
.L14:
                leaq
                0(,%r9,4), %rsi
                xorl
                %eax, %eax
                xorl
                %ecx, %ecx
                .p2align 4,,10
                .p2align 3
.L13:
                movq
                (%r14,%rax,8),
%rdx
                movl
                (%rdx,%rsi),
%edx
                imull
                (%rdi,%rax,4),
%edx
                addq
                $1, %rax
                addl
                %edx, %ecx
                cmpl
                %eax, %r13d
                jg
                .L13
                movl
                %ecx,
(%r10,%r9,4)
                addq
                $1, %r9
                cmpl
                %r9d, %r13d
                jg
                .L14
                addq
                $1, %r11
                cmpl
                %r11d, %r13d
                jg
                .L16
.L15:
                leaq
                32(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
                movq
                40(%rsp), %rax
                subq
                24(%rsp), %rax
                movl
                $.LC2, %esi
                pxor
                %xmm0, %xmm0
                movl
```

pmm-secuencial-modificado_a.s

```
call            clock_gettime
                movq
                $0, 8(%rsp)
.L18:
                movq
                16(%rsp), %rdi
                movq
                8(%rsp), %rax
                xorl
                %r13d, %r13d
                movq
                (%rdi,%rax,8),
%r15
                movq
                24(%rsp), %rdi
                movq
                (%rdi,%rax,8),
%r14
                .p2align 4,,10
                .p2align 3
.L17:
                leaq
                0(,%r13,4),
%rcx
                movq
                %r15, %rdx
                movq
                %rbx, %rax
                xorl
                %r12d, %r12d
                xorl
                %r11d, %r11d
                xorl
                %r10d, %r10d
                xorl
                %r9d, %r9d
                xorl
                %r8d, %r8d
                xorl
                %edi, %edi
                .p2align 4,,10
                .p2align 3
.L14:
                movq
                (%rax), %rsi
                addl
                $5, %edi
                addq
                $20, %rdx
                addq
                $40, %rax
                movl
                (%rsi,%rcx),
%esi
                imull
                -20(%rdx), %esi
                addl
                %esi, %r8d
                movq
                -32(%rax), %rsi
                movl
                (%rsi,%rcx),
%esi
                imull
                -16(%rdx), %esi
                addl
                %esi, %r9d
                movq
                -24(%rax), %rsi
                movl
                (%rsi,%rcx),
%esi
```

pmm-secuencial-modificado_b.s

```
call            clock_gettime
                xorl
                %r11d, %r11d
.L18:
                movq
                0(%rbp,%r11,8),
%rdx
                movq
                (%r12,%r11,8),
%r10
                xorl
                %r9d, %r9d
                .p2align 4,,10
                .p2align 3
.L17:
                movl
                (%r10,%r9,4),
%edi
                movq
                (%r14,%r9,8),
%rsi
                xorl
                %eax, %eax
                .p2align 4,,10
                .p2align 3
.L14:
                movl
                (%rsi,%rax,4),
%ecx
                imull
                %edi, %ecx
                addl
                %ecx, (%rdx,
%rax,4)
                addq
                $1, %rax
                cmpl
                %eax, %r13d
                jg
                .L14
                addq
                $1, %r9
                cmpl
                %r9d, %r13d
                jg
                .L17
                addq
                $1, %r11
                cmpl
                %r11d, %r13d
                jg
                .L18
.L16:
                leaq
                32(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
                movq
                40(%rsp), %rax
                subq
                24(%rsp), %rax
                movl
                $.LC3, %esi
                pxor
                %xmm0, %xmm0
                movl
                $1, %edi
                cvtsi2sdq
                %rax, %xmm0
                movq
```

```
                $1, %edi
                cvtsi2sdq
                %rax, %xmm0
                movq
                32(%rsp), %rax
                subq
                16(%rsp), %rax
                movapd
                %xmm0, %xmm1
                pxor
                %xmm0, %xmm0
                divsd
                .LC1(%rip),
%xmm1
                cvtsi2sdq
                %rax, %xmm0
                movl
                $1, %eax
                addsd
                %xmm1, %xmm0
                call
                __printf_chk
                movq
                (%rsp), %rax
                movl
                $.LC3, %esi
                movl
                $1, %edi
                movq
                -8(%r12,%rax),
%rax
                movl
                -4(%rax,%rbx),
%ecx
                movq
                (%r12), %rax
                movl
                (%rax), %edx
                xorl
                %eax, %eax
                call
                __printf_chk
                movq
                %r12, %rdi
                call
                free
                movq
                %rbp, %rdi
                call
                free
                movq
                %r14, %rdi
                call
                free
                xorl
                %eax, %eax
                movq
                56(%rsp), %rbx
                xorq
                %fs:40, %rbx
                jne
                .L38
                addq
                $72, %rsp
                .cfi_remember_s
tate
                .cfi_def_cfa_of
fset 56
                popq
                %rbx
                .cfi_def_cfa_of
fset 48
                popq
                %rbp
                .cfi_def_cfa_of
fset 40
                popq
                %r12
                .cfi_def_cfa_of
                imull
                -12(%rdx), %esi
                addl
                %esi, %r10d
                movq
                -16(%rax), %rsi
                movl
                (%rsi,%rcx),
%esi
                imull
                -8(%rdx), %esi
                addl
                %esi, %r11d
                movq
                -8(%rax), %rsi
                movl
                (%rsi,%rcx),
%esi
                imull
                -4(%rdx), %esi
                addl
                %esi, %r12d
                cmpl
                %ebp, %edi
                jl
                .L14
                addl
                %r9d, %r8d
                addl
                %r8d, %r10d
                addl
                %r10d, %r11d
                addl
                %r11d, %r12d
                movl
                %r12d,
(%r14,%r13,4)
                addq
                $1, %r13
                cmpl
                %r13d, %ebp
                jg
                .L17
                addq
                $1, 8(%rsp)
                movq
                8(%rsp), %rax
                cmpl
                %eax, %ebp
                jg
                .L18
.L16:
                leaq
                64(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
                movq
                72(%rsp), %rax
                subq
                56(%rsp), %rax
                movl
                $.LC3, %esi
                pxor
                %xmm0, %xmm0
                movl
                $1, %edi
                cvtsi2sdq
                %rax, %xmm0
                movq
                64(%rsp), %rax
                subq
                48(%rsp), %rax
                movapd
                %xmm0, %xmm1
                pxor
                %xmm0, %xmm0
                divsd
                32(%rsp), %rax
                subq
                16(%rsp), %rax
                movapd
                %xmm0, %xmm1
                pxor
                %xmm0, %xmm0
                divsd
                .LC2(%rip),
%xmm1
                cvtsi2sdq
                %rax, %xmm0
                movl
                $1, %eax
                addsd
                %xmm1, %xmm0
                call
                __printf_chk
                movq
                (%rsp), %rax
                movl
                $.LC4, %esi
                movl
                $1, %edi
                movq
                -8(%rbp,%rax),
%rax
                movl
                -4(%rax,%rbx),
%ecx
                movq
                0(%rbp), %rax
                movl
                (%rax), %edx
                xorl
                %eax, %eax
                call
                __printf_chk
                movq
                %rbp, %rdi
                call
                free
                movq
                %r12, %rdi
                call
                free
                movq
                %r14, %rdi
                call
                free
                xorl
                %eax, %eax
                movq
                56(%rsp), %rbx
                xorq
                %fs:40, %rbx
                jne
                .L40
                addq
                $72, %rsp
                .cfi_remember_s
tate
                .cfi_def_cfa_of
fset 56
                popq
                %rbx
                .cfi_def_cfa_of
fset 48
                popq
                %rbp
                .cfi_def_cfa_of
fset 40
                popq
                %r12
                .cfi_def_cfa_of
fset 32
                popq
                %r13
                .cfi_def_cfa_of
```

```
fset 32
                popq
                %r13
                .cfi_def_cfa_of
fset 24
                popq
                %r14
                .cfi_def_cfa_of
fset 16
                popq
                %r15
                .cfi_def_cfa_of
fset 8
                ret
.L3:
                .cfi_restore_st
ate
                movq
                (%rsp), %r15
                salq
                $2, %rbx
                movq
                %r15, %rdi
                call
                malloc
                movq
                %r15, %rdi
                movq
                %rax, %rbp
                call
                malloc
                leaq
                16(%rsp), %rsi
                xorl
                %edi, %edi
                movq
                %rax, %r14
                call
                clock_gettime
                .LC2(%rip),
%xmm1
                cvtsi2sdq
                %rax, %xmm0
                movl
                $1, %eax
                addsd
                %xmm1, %xmm0
                call
                __printf_chk
                movq
                24(%rsp), %r15
                movq
                32(%rsp), %rdi
                movl
                $.LC4, %esi
                movq
                -8(%r15,%rdi),
%rax
                movq
                40(%rsp), %rdi
                movl
                -4(%rax,%rdi),
%ecx
                movq
                (%r15), %rax
                movl
                $1, %edi
                movl
                (%rax), %edx
                xorl
                %eax, %eax
                call
                __printf_chk
                movq
                %r15, %rdi
                call
                free
                movq
                16(%rsp), %rdi
                call
                free
                movq
                %rbx, %rdi
                call
                free
                xorl
                %eax, %eax
                movq
                88(%rsp), %rbx
                xorq
                %fs:40, %rbx
                jne
                .L41
                addq
                $104, %rsp
                .cfi_remember_s
tate
                .cfi_def_cfa_of
fset 56
                popq
                %rbx
                .cfi_def_cfa_of
fset 48
                popq
                %rbp
                .cfi_def_cfa_of
fset 40
                popq
                %r12
                .cfi_def_cfa_of
fset 32
                popq
                %r13
                .cfi_def_cfa_of
fset 24
                popq
                %r14
                .cfi_def_cfa_of
fset 24
                popq
                %r14
                .cfi_def_cfa_of
fset 16
                popq
                %r15
                .cfi_def_cfa_of
fset 8
                ret
.L4:
                .cfi_restore_st
ate
                movq
                (%rsp), %r15
                salq
                $2, %rbx
                movq
                %r15, %rdi
                call
                malloc
                movq
                %r15, %rdi
                movq
                %rax, %r12
                call
                malloc
                leaq
                16(%rsp), %rsi
                xorl
                %edi, %edi
                movq
                %rax, %r14
                call
                clock_gettime
```

	fset 16
	popq %r15
	.cfi_def_cfa_of
	fset 8
	ret
	.L4:
	.cfi_restore_st
	ate
	movq 32(%rsp), %rbx
	movq %rbx, %rdi
	call malloc
	movq %rbx, %rdi
	movq %rax, 16(%rsp)
	call malloc
	leaq 48(%rsp), %rsi
	xorl %edi, %edi
	movq %rax, %rbx
	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];

main()
{
    int ii,i,X1,X2;
    int R[40000];

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

    for (ii=1; ii<=40000;ii++) {
        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,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

    printf("Tiempo(seg): %f\nR[0]=%d, R[39999]=%d
\n",transcurrido,R[0],R[39999]);
}
```

1.1. MODIFICACIONES REALIZADAS:

Modificación a) –explicación-: DESENRROLLADO DE BUCLE. EN VEZ DE REALIZAR UN SALTO EN CADA ITERACIÓN. SE REALIZARÁ UN SALTO CADA 4 ITERACIONES REDUCIENDO ASÍ EL NÚMERO DE INSTRUCCIONES

Modificación b) –explicación-: ADEMÁS DE INCLUIR EL DESENRROLLADO, UNIMOS DOS BUCLES PARA EVITAR RECORRER DOS VECES NUESTRA ESTRUCTURA Y HACERLO SOLO UNA VEZ HACIENDO DOS OPERACIONES POR INDICE.

Modificación b) –explicación-: INCLUYENDO LAS DOS MODIFICACIONES ANTERIORES, HACEMOS UNA MULTIPLICACIÓN QUE SE HACÍA EN EL INTERIOR DEL BUCLE FUERA, PARA NO TENER QUE REALIZAR N OPERACIONES Y HACERLA UNA SOLA VEZ.

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];

main()
{
    int ii,i,X1,X2;
    int tmpX1_0,tmpX1_1,tmpX1_2,tmpX1_3,tmpX1_4;
    int tmpX2_0,tmpX2_1,tmpX2_2,tmpX2_3,tmpX2_4;
    int R[40000];

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

    for (ii=1; ii<=40000;ii++) {
        // iniciar temporales
        tmpX1_0=tmpX1_1=tmpX1_2=tmpX1_3=tmpX1_4=0.0;
        tmpX2_0=tmpX2_1=tmpX2_2=tmpX2_3=tmpX2_4=0.0;
        // calcular en temporales
        for(i=0; i<5000;i+=5) {
            tmpX1_0+=2*s[i].a+ii;
            tmpX1_1+=2*s[i+1].a+ii;
            tmpX1_2+=2*s[i+2].a+ii;
            tmpX1_3+=2*s[i+3].a+ii;
            tmpX1_4+=2*s[i+4].a+ii;
              }
        for(i=0; i<5000;i+=5) {
            tmpX2_0+=3*s[i].b-ii;
            tmpX2_1+=3*s[i+1].b-ii;
            tmpX2_2+=3*s[i+2].b-ii;
            tmpX2_3+=3*s[i+3].b-ii;
            tmpX2_4+=3*s[i+4].b-ii;
        }
        // sumar temporales
        X1=tmpX1_0+tmpX1_1+tmpX1_2+tmpX1_3+tmpX1_4;
        X2=tmpX2_0+tmpX2_1+tmpX2_2+tmpX2_3+tmpX2_4;
```

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

    clock_gettime(CLOCK_REALTIME,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

    printf("Tiempo(seg): %f\nR[0]=%d, R[39999]=%d
\n",transcurrido,R[0],R[39999]);
}
```

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

```
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$gcc -O2 -o figura1-mod-a figura1-modificado_a.c -lrt
figura1-modificado_a.c:12:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$./figura1-mod-a
Tiempo(seg): 0.164926
R[0]=0, R[39999]=-199995000
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$
```

b) Captura figura1**`-modificado_b.c`**

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

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

main()
{
    int ii,i,X1,X2;
    int tmpX1_0,tmpX1_1,tmpX1_2,tmpX1_3,tmpX1_4;
    int tmpX2_0,tmpX2_1,tmpX2_2,tmpX2_3,tmpX2_4;
    int R[40000];

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

    for (ii=1; ii<=40000;ii++) {
        // iniciar temporales
        tmpX1_0=tmpX1_1=tmpX1_2=tmpX1_3=tmpX1_4=0.0;
        tmpX2_0=tmpX2_1=tmpX2_2=tmpX2_3=tmpX2_4=0.0;
        // calcular en temporales
        for(i=0; i<5000;i+=5) {
            tmpX1_0+=2*s[i].a+ii;
            tmpX1_1+=2*s[i+1].a+ii;
            tmpX1_2+=2*s[i+2].a+ii;
```

```
            tmpX1_3+=2*s[i+3].a+ii;
            tmpX1_4+=2*s[i+4].a+ii;

            tmpX2_0+=3*s[i].b-ii;
            tmpX2_1+=3*s[i+1].b-ii;
            tmpX2_2+=3*s[i+2].b-ii;
            tmpX2_3+=3*s[i+3].b-ii;
            tmpX2_4+=3*s[i+4].b-ii;
        }
        // sumar temporales
        X1=tmpX1_0+tmpX1_1+tmpX1_2+tmpX1_3+tmpX1_4;
        X2=tmpX2_0+tmpX2_1+tmpX2_2+tmpX2_3+tmpX2_4;
        // comprobacion
        if (X1<X2)  R[ii]=X1;
        else  R[ii]=X2;
    }

    clock_gettime(CLOCK_REALTIME,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

    printf("Tiempo(seg): %f\nR[0]=%d, R[39999]=%d
\n",transcurrido,R[0],R[39999]);
}
```

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

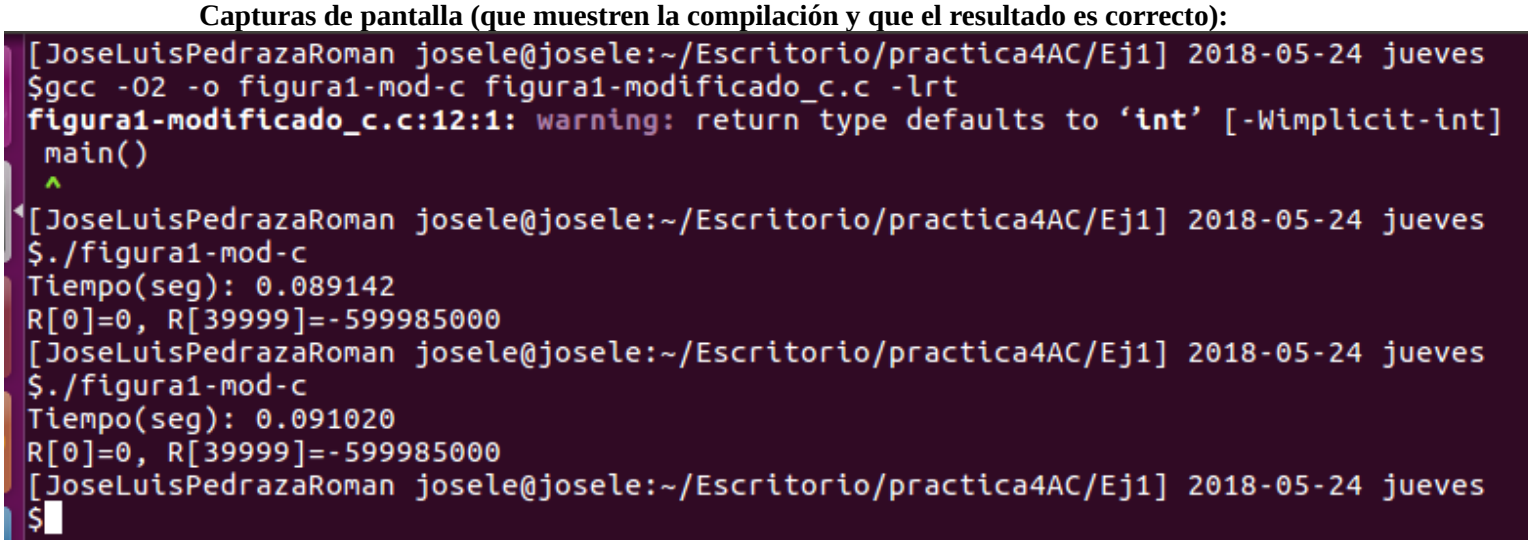
```
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$gcc -O2 -o figura1-mod-b figura1-modificado_b.c -lrt
figura1-modificado_b.c:12:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$./figura1-mod-b
Tiempo(seg): 0.173917
R[0]=0, R[39999]=-199995000
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$./figura1-mod-b
Tiempo(seg): 0.208957
R[0]=0, R[39999]=-199995000
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves
$
```

c) Captura figura1**`-modificado_c.c`**

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

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

main()
{
    int ii,i,X1,X2;
```

```
    int tmpX1_0,tmpX1_1,tmpX1_2,tmpX1_3,tmpX1_4;
    int tmpX2_0,tmpX2_1,tmpX2_2,tmpX2_3,tmpX2_4;
    int R[40000];

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

    for (ii=1; ii<=40000;ii++) {
        // iniciar temporales
        tmpX1_0=tmpX1_1=tmpX1_2=tmpX1_3=tmpX1_4=0.0;
        tmpX2_0=tmpX2_1=tmpX2_2=tmpX2_3=tmpX2_4=0.0;
        // calcular en temporales
        for(i=0; i<5000;i+=5) {
            tmpX1_0+=s[i].a+ii;
            tmpX1_1+=s[i+1].a+ii;
            tmpX1_2+=s[i+2].a+ii;
            tmpX1_3+=s[i+3].a+ii;
            tmpX1_4+=s[i+4].a+ii;

            tmpX2_0+=s[i].b-ii;
            tmpX2_1+=s[i+1].b-ii;
            tmpX2_2+=s[i+2].b-ii;
            tmpX2_3+=s[i+3].b-ii;
            tmpX2_4+=s[i+4].b-ii;
        }
        // sumar temporales
        X1=(tmpX1_0+tmpX1_1+tmpX1_2+tmpX1_3+tmpX1_4)*2;
        X2=(tmpX2_0+tmpX2_1+tmpX2_2+tmpX2_3+tmpX2_4)*3;
        // comprobacion
        if (X1<X2)  R[ii]=X1;
        else  R[ii]=X2;
    }

    clock_gettime(CLOCK_REALTIME,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

    printf("Tiempo(seg): %f\nR[0]=%d, R[39999]=%d
\n",transcurrido,R[0],R[39999]);
}
```

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



1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0,240674*
Modificación a)	0,164926
Modificación b)	0,173917
Modificación c)	0,091020

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES:

figura1.s

```
call            clock_gettime
                movl
s+39992(%rip), %eax
                leaq
                36(%rsp),
%rdx
                leal
                (%rax,%rax),
%edi
                movl
s+39996(%rip), %eax
                leal
                (%rax,
%rax,2), %r8d
                movl
                $1, %eax
                jmp
                .L2
                .p2align
4,,10
                .p2align 3
.L5:
                addl
                $1, %eax
                movl
                %esi, (%rdx)
                addq
                $4, %rdx
```

figura1-modificado_a.s

```
call            clock_gettime
                leaq
                52(%rsp), %rax
                movq
                %rax, 8(%rsp)
                movl
                $1, %eax
                .p2align 4,,10
                .p2align 3
.L2:
                movl
                $s, %edx
                xorl
                %r12d, %r12d
                xorl
                %ebp, %ebp
                xorl
                %ebx, %ebx
                xorl
                %r11d, %r11d
                xorl
                %esi, %esi
                .p2align 4,,10
                .p2align 3
.L3:
                movl
                (%rdx), %ecx
                addq
                $40, %rdx
                leal
                (%rax,%rcx,2),
%ecx
                addl
                %ecx, %esi
                movl
```

figura1-modificado_b.s

```
call            clock_gettime
                movl
                $1, %edx
                .p2align 4,,10
                .p2align 3
.L2:
                movl
                $s, %eax
                xorl
                %r13d, %r13d
                xorl
                %r12d, %r12d
                xorl
                %ebp, %ebp
                xorl
                %ebx, %ebx
                xorl
                %edi, %edi
                xorl
                %r11d, %r11d
                xorl
                %r10d, %r10d
                xorl
                %r9d, %r9d
                xorl
                %r8d, %r8d
                xorl
                %esi, %esi
                .p2align 4,,10
                .p2align 3
.L3:
                movl
                (%rax), %ecx
                addq
                $40, %rax
```

figura1-modificado_c.s

```
call            clock_gettime
                movl
                $1, %edx
                .p2align 4,,10
                .p2align 3
.L2:
                movl
                $s, %eax
                xorl
                %r9d, %r9d
                xorl
                %r8d, %r8d
                xorl
                %edi, %edi
                xorl
                %esi, %esi
                xorl
                %ecx, %ecx
                xorl
                %r12d, %r12d
                xorl
                %ebp, %ebp
                xorl
                %ebx, %ebx
                xorl
                %r11d, %r11d
                xorl
                %r10d, %r10d
                .p2align 4,,10
                .p2align 3
.L3:
                movl
                (%rax), %r15d
                subl
                %edx, %ecx
```

```
                cmpl    $40001, %eax
                je      .L4
.L2:
                leal    (%rax,%rdi),
%esi
                movl    %r8d, %ecx
                subl    %eax, %ecx
                cmpl    %ecx, %esi
                jl      .L5
                addl    $1, %eax
                movl    %ecx, (%rdx)
                addq    $4, %rdx
                cmpl    $40001, %eax
                jne     .L2
.L4:
                leaq    16(%rsp),
%rsi
                xorl    %edi, %edi
                call    clock_gettime
                        -32(%rdx), %ecx
                        leal
                        (%rax,%rcx,2),
%ecx
                        addl
                        %ecx, %r11d
                        movl
                        -24(%rdx), %ecx
                        leal
                        (%rax,%rcx,2),
%ecx
                        addl
                        %ecx, %ebx
                        movl
                        -16(%rdx), %ecx
                        leal
                        (%rax,%rcx,2),
%ecx
                        addl
                        %ecx, %ebp
                        movl
                        -8(%rdx), %ecx
                        leal
                        (%rax,%rcx,2),
%ecx
                        addl
                        %ecx, %r12d
                        cmpq
                        %rdx, %r15
                        jne
                        .L3
                        movl
                        $s+4, %edx
                        xorl
                        %edi, %edi
                        xorl
                        %r8d, %r8d
                        xorl
                        %r9d, %r9d
                        xorl
                        %r10d, %r10d
                        xorl
                        %r13d, %r13d
                        .p2align 4,,10
                        .p2align 3
.L4:
                        movl
                        (%rdx), %ecx
                        addq
                        $40, %rdx
                        leal
                        (%rcx,%rcx,2),
%ecx
                        subl
                        %eax, %ecx
                        addl
                        %ecx, %r13d
                        movl
                        -32(%rdx), %ecx
                        leal
                        (%rcx,%rcx,2),
%ecx
                        subl
                        %eax, %ecx
                        addl
                        %ecx, %r10d
                        movl
                        -24(%rdx), %ecx
                        leal
                        (%rcx,%rcx,2),
%ecx
                        subl
                        %eax, %ecx
                        addl
                        %ecx, %r9d
                        movl
                        -16(%rdx), %ecx
                        leal
                        (%rcx,%rcx,2),
                        leal
                        (%rdx,%rcx,2),
%ecx
                        addl
                        %ecx, %esi
                        movl
                        -32(%rax),
%ecx
                        leal
                        (%rdx,%rcx,2),
%ecx
                        addl
                        %ecx, %r8d
                        movl
                        -24(%rax),
%ecx
                        leal
                        (%rdx,%rcx,2),
%ecx
                        addl
                        %ecx, %r9d
                        movl
                        -16(%rax),
%ecx
                        leal
                        (%rdx,%rcx,2),
%ecx
                        addl
                        %ecx, %r10d
                        movl
                        -8(%rax), %ecx
                        leal
                        (%rdx,%rcx,2),
%ecx
                        addl
                        %ecx, %r11d
                        movl
                        -36(%rax),
%ecx
                        leal
                        (%rcx,%rcx,2),
%ecx
                        subl
                        %edx, %ecx
                        addl
                        %ecx, %edi
                        movl
                        -28(%rax),
%ecx
                        leal
                        (%rcx,%rcx,2),
%ecx
                        subl
                        %edx, %ecx
                        addl
                        %ecx, %ebx
                        movl
                        -20(%rax),
%ecx
                        leal
                        (%rcx,%rcx,2),
%ecx
                        subl
                        %edx, %ecx
                        addl
                        %ecx, %ebp
                        movl
                        -12(%rax),
%ecx
                        leal
                        (%rcx,%rcx,2),
%ecx
                        subl
                        %edx, %ecx
                        addl
                        %ecx, %r12d
                        movl
                        -4(%rax), %ecx
                        leal
                        subl
                        %edx, %esi
                        addl
                        4(%rax), %ecx
                        addl
                        12(%rax), %esi
                        subl
                        %edx, %edi
                        subl
                        %edx, %r8d
                        addl
                        20(%rax), %edi
                        addl
                        28(%rax), %r8d
                        addl
                        %edx, %r15d
                        subl
                        %edx, %r9d
                        addq
                        $40, %rax
                        addl
                        %r15d, %r10d
                        movl            -
32(%rax), %r15d
                        addl            -
4(%rax), %r9d
                        addl
                        %edx, %r15d
                        addl
                        %r15d, %r11d
                        movl            -
24(%rax), %r15d
                        addl
                        %edx, %r15d
                        addl
                        %r15d, %ebx
                        movl            -
16(%rax), %r15d
                        addl
                        %edx, %r15d
                        addl
                        %r15d, %ebp
                        movl            -
8(%rax), %r15d
                        addl
                        %edx, %r15d
                        addl
                        %r15d, %r12d
                        cmpq
                        %rax, %r13
                        jne
                        .L3
                        addl
                        %esi, %ecx
                        leal
                        (%r10,%r11), %eax
                        addl
                        %ecx, %edi
                        addl
                        %ebx, %eax
                        addl
                        %edi, %r8d
                        addl
                        %ebp, %eax
                        addl
                        %r8d, %r9d
                        addl
                        %r12d, %eax
                        leal
                        (%r9,%r9,2), %ecx
                        addl
                        %eax, %eax
                        cmpl
                        %ecx, %eax
                        jge
                        .L4
                        movl
                        %eax, (%r14)
.L5:
```

```
%ecx
                subl
                %eax, %ecx
                addl
                %ecx, %r8d
                movl
                -8(%rdx), %ecx
                leal
                (%rcx,%rcx,2),
%ecx
                subl
                %eax, %ecx
                addl
                %ecx, %edi
                cmpq
                %rdx, %r14
                jne
                .L4
                addl
                %r11d, %esi
                addl
                %r13d, %r10d
                addl
                %esi, %ebx
                addl
                %r10d, %r9d
                addl
                %ebx, %ebp
                addl
                %r9d, %r8d
                addl
                %ebp, %r12d
                addl
                %r8d, %edi
                cmpl
                %edi, %r12d
                jge
                .L5
                movq
                8(%rsp), %rdi
                movl
                %r12d, (%rdi)
.L6:
                addl
                $1, %eax
                addq
                $4, 8(%rsp)
                cmpl
                $40001, %eax
                jne
                .L2
                leaq
                32(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

```
                (%rcx,%rcx,2),
%ecx
                subl
                %edx, %ecx
                addl
                %ecx, %r13d
                cmpq
                %rax, %r14
                jne
                .L3
                addl
                %r8d, %esi
                addl
                %ebx, %edi
                addl
                %esi, %r9d
                addl
                %edi, %ebp
                addl
                %r9d, %r10d
                addl
                %ebp, %r12d
                addl
                %r10d, %r11d
                addl
                %r12d, %r13d
                cmpl
                %r13d, %r11d
                jge
                .L4
                movl
                %r11d, (%r15)
.L5:
                addl
                $1, %edx
                addq
                $4, %r15
                cmpl
                $40001, %edx
                jne
                .L2
                leaq
                16(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

```
                addl
                $1, %edx
                addq
                $4, %r14
                cmpl
                $40001, %edx
                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 <stdlib.h>
#include <time.h>

main(int argc, char **argv)
{
    // 1. Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr,"Falta num\n");
        exit(-1);
    }
    int N = atoi(argv[1]);

    struct timespec ini,fin;
    double transcurrido;

    // 2. Creación e inicialización de vector y matriz
    // 2.1. Creación
    int i,a=47;
    int x[N], y[N];

    // 2.2. Inicialización
    for (i=1; i<=N;i++) {
        x[i]=i;
        y[i]=i;
    }

    // 3. Cálculo resultado
    clock_gettime(CLOCK_REALTIME,&ini);

    for (i=1; i<=N;i++) {
        y[i]=a*x[i]+y[i];
    }

    clock_gettime(CLOCK_REALTIME,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double)
((fin.tv_nsec-ini.tv_nsec)/(1.e+9));

    // 4. Impresión de vector resultado
    printf("Tiempo(seg): %f\ny[0]=%d, y[N-1]=%d
\n",transcurrido,y[0],y[N-1]);
}
```

Tiempos ejec. PARA N= 1000000	**-O0**	**-Os**	**-O2**	**-O3**
	0,004346	0,000859	0,000737	0,000603

CAPTURAS DE PANTALLA

```
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-O0 1000000
Tiempo(seg): 0.004340
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-O0 1000000
Tiempo(seg): 0.004346
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-Os 1000000
Tiempo(seg): 0.002309
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-Os 1000000
Tiempo(seg): 0.000785
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-Os 1000000
Tiempo(seg): 0.000859
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-O2 1000000
Tiempo(seg): 0.000737
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-O2 1000000
Tiempo(seg): 0.000674
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-O3 1000000
Tiempo(seg): 0.001393
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-O3 1000000
Tiempo(seg): 0.000736
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-O3 1000000
Tiempo(seg): 0.000603
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-O3 1000000
Tiempo(seg): 0.001232
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

-O0 – EN ESTE NIVEL SE DESHABILITA LA OPTIMIZACIÓN, POR TANTO, EL COMPILADOR NO OPTIMIZARÁ NINGUNA PARTE DEL CÓDIGO.

-Os – EL TIEMPO AL COMPILAR ES ALGO MAYOR (ESTO ES INAPRECIABLE). LA PRINCIPAL DIFERENCIA SE ENCUENTRA EN EL CÑODIGO ENSAMBLADOR, DONDE SE REDUCE EL NUMERO DE LÍNEAS NOTORIAMENTE AUNQUE SE GENEREN MÁS SUBRUTINAS, ESTAS TIENEN UN NUMERO MENOR DE INSTRUCCIONES QUE SIN LA OPTIMIZACIÓN.

-O2 – ES EL NIVEL DE OPTIMIZACIÓN RECOMENDADO- EL NUMERO DE INSTRUCCIONES SIGUE SIENDO SIMILAR AL QUE HAY EN EL ANTERIOR CASO, PERO EXISTE MAYOR MEJORA USANDO DIFERENTES INSTRUCCIONES.

-O3 – NIVEL MÁXIMO DE OPTIMIZACIÓN, ACTIVA OPCIONES QUE SON MUY COSTOSAS EN TIEMPO DE COMPILACION Y USO DE MEMORIA, Y AUNQUE ESTE NO ES EL CASO, HAY OCASIONES EN LAS QUE ESTO NO PROPORCIONA UNA MEJORA DE RENDIMIENTO. LA PARTE DE LA SUMA DE LOS VECTORES EN ESTE NIVEL ES BASTANTE MÁS ILEGIBLE QUE EL RESTO. CREA MUCHAS SUBRUTINAS A LAS QUE LLAMA, PERO VIENDO LOS RESULTADOS, EN ESTE CASO LOS TIEMPOS MEJORAN PERO NO DE FORMA NOTORIA (YA QUE SE ASEMEJA A LOS TIEMPOS DEL NIVEL ANTERIOR)

CÓDIGO EN ENSAMBLADOR

daxpyO0.s

```
call          clock_gettime
              movl
              $1, -148(%rbp)
              jmp
              .L5
.L6:
              movq          -
128(%rbp), %rax
              movl          -
148(%rbp), %edx
              movslq
              %edx, %rdx
              movl
              (%rax,%rdx,4),
%eax
              imull         -
140(%rbp), %eax
              movl
              %eax, %ecx
              movq          -
112(%rbp), %rax
              movl          -
148(%rbp), %edx
              movslq
              %edx, %rdx
```

daxpyOs.s

```
call          clock_gettime
              movl
              $1, %eax
.L5:
              cmpl
              %ebx, %eax
              jg
              .L11
              movslq
              %eax, %rdx
              incl
              %eax
              imull
              $47,
(%r14,%rdx,4), %ecx
              addl
              %ecx,
0(%r13,%rdx,4)
              jmp
              .L5
.L11:
              leaq
              -56(%rbp),
%rsi
              xorl
```

daxpyO2.s

```
call          clock_gettime
              movl
              %r14d, %esi
              xorl
              %edx, %edx
              addq          $1,
%rsi
              .p2align 4,,10
              .p2align 3
.L6:
              imull
              $47,
4(%r13,%rdx,4), %ecx
              addl
              %ecx, 4(%rbx,
%rdx,4)
              addq          $1,
%rdx
              cmpq
              %rdx, %rsi
              jne           .L6
.L7:
              leaq          -
64(%rbp), %rsi
              xorl
```

daxpyO3.s

```
call
clock_gettime
              leaq
4(%r12), %rcx
              movq          -
104(%rbp), %rdx
              movq
              %rcx,
%rax
              andl
              $15,
%eax
              shrq
              $2,
%rax
              negq
              %rax
              andl
              $3,
%eax
              cmpl
              %ebx,
```

movl (%rax,%rdx,4), %eax
addl %eax, %ecx
movq -112(%rbp), %rax
movl -148(%rbp), %edx
movslq %edx, %rdx
movl %ecx, (%rax, %rdx,4)
addl $1, -148(%rbp)
.L5:
movl -148(%rbp), %eax
cmpl -144(%rbp), %eax
jle .L6
leaq -80(%rbp), %rax
movq %rax, %rsi
movl $0, %edi
call clock_gettime
	%edi, %edi
	decl %ebx
	movslq %ebx, %rbx
	call clock_gettime
		%edi, %edi
		movslq %r14d, %r14
		call clock_gettime
			%eax
			cmova %ebx, %eax