2° curso / 2° 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 1scpu en la máquina en la que ha tomado las medidas

```
selejposele:- S PSI="[JoseluisPedrazaRoman | Ug|h: |w] \[ 0[%F %A]\nS" oseluisPedrazaRoman joselegijosele:-] 2018-05-23 miércoles sepu sepu juitectura: x86_64 do(s) de operación de las CPUs:32-bit, 64-bit den de bytes: Little Endian (Line CPU(s) list: 0-3 do(s) de procesamiento por núcleo:1 (lock) por socket:-4 (ket(s): 1 do(s)) NUMA: 1 (lock) por socket:-4 (ket(s): 1 do(s)) NUMA: 1 (lock) por socket:-4 (
```

- 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;
        }
        ...</pre>
```

7

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]);</pre>
          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]);</pre>
          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]);</pre>
          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>
  #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]);</pre>
          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]);</pre>
          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);
}</pre>
```

Capturas de pantalla:

b) Captura de pmm-secuencial-modificado_b.c

```
#include <stdio.h>
#include <stdib.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;</pre>
```

```
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++)</pre>
    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]);</pre>
          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]);</pre>
          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]);</pre>
          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:

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		pmm-secue modificae		pmm-secuencial- modificado_b.s	
call	clock_gettime xorl	call	clock_gettime movq	call	clock_gettime xorl
	%r11d, %r11d		\$0, 8(%rsp)		%r11d, %r11d
.L16:		.L18:	, , , , ,	.L18:	
	movq		movq		movq
	(%r12,%r11,8),		16(%rsp), %rdi		0(%rbp,%r11,8)
6r10			movq	%rdx	
	movq		8(%rsp), %rax		movq
	0(%rbp,%r11,8),		xorl		(%r12,%r11,8),
Krdi			%r13d, %r13d	%r10	
	xorl %r9d, %r9d		movq (%rdi %rov 8)		xorl %r9d, %r9d
	.p2align 4,,10	%r15	(%rdi,%rax,8),		.p2align 4,,10
	.p2align 3	701 13	movq		.p2align 3
L14:	·pzarrgii o		24(%rsp), %rdi	.L17:	.pzarryn o
	leag		movq		movl
	0(,%r9,4), %rsi		(%rdi,%rax,8),		(%r10,%r9,4),
	xorl	%r14		%edi	. , , , , , , , , , , , , , , , , , , ,
	%eax, %eax		.p2align 4,,10		movq
	xorl		.p2align 3		(%r14,%r9,8),
	%ecx, %ecx	.L17:		%rsi	
	.p2align 4,,10		leaq		xorl
	.p2align 3		0(,%r13,4),		%eax, %eax
L13:		%rcx			.p2align 4,,10
	movq		movq		.p2align 3
/.a.al	(%r14,%rax,8),		%r15, %rdx	.L14:	
Krdx	mou.l		movd		movl
	movl		%rbx, %rax xorl	%ecx	(%rsi,%rax,4),
edx .	(%rdx,%rsi),		%r12d, %r12d	70ECX	imull
icux	imull		xorl		%edi, %ecx
	(%rdi,%rax,4),		%r11d, %r11d		addl
edx	(10. 0=7.0. 0, 177		xorl		%ecx, (%rdx,
	addq		%r10d, %r10d	%rax,4)	, , , ,
	\$1, %rax		xorl	, ,	addq
	addl		%r9d, %r9d		\$1, %rax
	%edx, %ecx		xorl		cmpl
	cmpl		%r8d, %r8d		%eax, %r13d
	%eax, %r13d		xorl		jg
	jg		%edi, %edi		. L14
	.L13		.p2align 4,,10		addq
	movl	111.	.p2align 3		\$1, %r9
(0/r10 0/r0 1)	%ecx,	.L14:	mova		cmpl
(%r10,%r9,4)	addq		movq (%rax), %rsi		%r9d, %r13d
	\$1, %r9		addl		jg . L17
	cmpl		\$5, %edi		addq
	%r9d, %r13d		addq		\$1, %r11
	jg		\$20, %rdx		cmpl
	.L14		addq		%r11d, %r13d
	addq		\$40, %rax		jg
	\$1, %r11		movl		.L18
	cmpl		(%rsi,%rcx),	.L16:	
	%r11d, %r13d	%esi			leaq
	jg		imull		32(%rsp), %rsi
	.L16		-20(%rdx), %esi		xorl
L15:	1		addl		%edi, %edi
	leaq		%esi, %r8d		call
	32(%rsp), %rsi		movq		clock_gettime
	xorl %edi, %edi		-32(%rax), %rsi movl		movq 40(%rsn) %ray
	call		(%rsi,%rcx),		40(%rsp), %rax subq
	clock_gettime	%esi	(/01 31, /01 UX),		24(%rsp), %ra>
	movq	,3001	imull		movl
	40(%rsp), %rax		-16(%rdx), %esi		\$.LC3, %esi
	subq		addl		pxor
	24(%rsp), %rax		%esi, %r9d		%xmm0, %xmm0
	movl		movq		movl
	\$.LC2, %esi		-24(%rax), %rsi		\$1, %edi
	pxor		movl		cvtsi2sdq
	%xmm0, %xmm0		(%rsi,%rcx),		%rax, %xmm0
	movl	%esi		l	movq

	\$1, %edi		imull		32(%rsp), %rax
	cvtsi2sdq		-12(%rdx), %esi		subg
	%rax, %xmm0		addl		16(%rsp), %rax
	movq		%esi, %r10d		movapd
	·				•
	32(%rsp), %rax		movq		%xmm0, %xmm1
	subq		-16(%rax), %rsi		pxor
	16(%rsp), %rax		movl		%xmm0, %xmm0
	movapd		(%rsi,%rcx),		divsd
	%xmm0, %xmm1	%esi			.LC2(%rip),
	pxor		imull	%×mm1	
	%xmm0, %xmm0		-8(%rdx), %esi		cvtsi2sdq
	divsd		addl		%rax, %xmm0
	.LC1(%rip),		%esi, %r11d		movl
%xmm1	. 202(% 15),		movq		\$1, %eax
70/1111111	outsizeda		•		
	cvtsi2sdq		-8(%rax), %rsi		addsd
	%rax, %xmm0		movl		%xmm1, %xmm0
	movl		(%rsi,%rcx),		call
	\$1, %eax	%esi			printf_chk
	addsd		imull		movq
	%xmm1, %xmm0		-4(%rdx), %esi		(%rsp), %rax
	call		addl		movl
	printf_chk		%esi, %r12d		\$.LC4, %esi
	movq		cmpl		movl
	(%rsp), %rax		%ebp, %edi		\$1, %edi
	movl		jl		movq
			-		
	\$.LC3, %esi		.L14	0/	-8(%rbp,%rax),
	movl		addl	%rax	_
	\$1, %edi		%r9d, %r8d		movl
	movq		addl		-4(%rax,%rbx),
	-8(%r12,%rax),		%r8d, %r10d	%ecx	
%rax			addl		movq
	movl		%r10d, %r11d		0(%rbp), %rax
	-4(%rax,%rbx),		addl		movl
%ecx	. (70. 0.7.7.0. 27.77		%r11d, %r12d		(%rax), %edx
70007	movq		movl		xorl
	(%r12), %rax	(0/:=4.4.0/:=4.04.)	%r12d,		%eax, %eax
	mov1	(%r14,%r13,4)			call
	(%rax), %edx		addq		printf_chk
	xorl		\$1, %r13		movq
	%eax, %eax		cmpl		%rbp, %rdi
	call		%r13d, %ebp		call
	printf_chk		jg		free
	movq		.L17		movq
	%r12, %rdi		addq		%r12, %rdi
	call		\$1, 8(%rsp)		call
	free				free
			movq		
	movq		8(%rsp), %rax		movq
	%rbp, %rdi		cmpl		%r14, %rdi
	call		%eax, %ebp		call
	free		jg		free
	movq		.L18		xorl
	%r14, %rdi	.L16:			%eax, %eax
	call		leaq		movq
	free		64(%rsp), %rsi		56(%rsp), %rbx
	xorl		xorl		xorq
	%eax, %eax		%edi, %edi		%fs:40, %rbx
	movq		call		ine
	56(%rsp), %rbx		clock_gettime		.L40
	` ' ' '				
	xorq		movq		addq
	%fs:40, %rbx		72(%rsp), %rax		\$72, %rsp
	jne		subq	l	.cfi_remember_s
			56(%rsp), %rax	tate	
	.L38		1	I	.cfi_def_cfa_of
	. L38 addq		movl		.cri_acr_cra_or
			\$.LC3, %esi	fset 56	.cri_dcr_crd_or
	addq			fset 56	popq
tate	addq \$72, %rsp		\$.LC3, %esi	fset 56	
tate	addq \$72, %rsp .cfi_remember_s		<pre>\$.LC3, %esi pxor %xmm0, %xmm0</pre>	fset 56	popq %rbx
	addq \$72, %rsp		<pre>\$.LC3, %esi pxor %xmm0, %xmm0 mov1</pre>		popq
tate fset 56	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of		\$.LC3, %esi pxor %xmm0, %xmm0 mov1 \$1, %edi	fset 56	popq %rbx .cfi_def_cfa_of
	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq		\$.LC3, %esi pxor %xmm0, %xmm0 movl \$1, %edi cvtsi2sdq		popq %rbx .cfi_def_cfa_of popq
	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq %rbx		\$.LC3, %esi pxor %xmm0, %xmm0 movl \$1, %edi cvtsi2sdq %rax, %xmm0		popq %rbx .cfi_def_cfa_of popq %rbp
fset 56	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq		\$.LC3, %esi pxor %xmm0, %xmm0 movl \$1, %edi cvtsi2sdq %rax, %xmm0 movq	fset 48	popq %rbx .cfi_def_cfa_of popq
	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq %rbx		\$.LC3, %esi pxor %xmm0, %xmm0 movl \$1, %edi cvtsi2sdq %rax, %xmm0 movq 64(%rsp), %rax		popq %rbx .cfi_def_cfa_of popq %rbp
fset 56	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq %rbx		\$.LC3, %esi pxor %xmm0, %xmm0 movl \$1, %edi cvtsi2sdq %rax, %xmm0 movq	fset 48	popq %rbx .cfi_def_cfa_of popq %rbp
fset 56	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq %rbx .cfi_def_cfa_of		\$.LC3, %esi pxor %xmm0, %xmm0 movl \$1, %edi cvtsi2sdq %rax, %xmm0 movq 64(%rsp), %rax	fset 48	popq %rbx .cfi_def_cfa_of popq %rbp .cfi_def_cfa_of
fset 56	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq %rbx .cfi_def_cfa_of popq		\$.LC3, %esi pxor %xmm0, %xmm0 mov1 \$1, %edi cvtsi2sdq %rax, %xmm0 movq 64(%rsp), %rax subq	fset 48	popq %rbx .cfi_def_cfa_of popq %rbp .cfi_def_cfa_of popq
fset 56	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq %rbx .cfi_def_cfa_of popq %rbp		\$.LC3, %esi pxor %xmm0, %xmm0 mov1 \$1, %edi cvtsi2sdq %rax, %xmm0 movq 64(%rsp), %rax subq 48(%rsp), %rax movapd	fset 48	popq %rbx .cfi_def_cfa_of popq %rbp .cfi_def_cfa_of popq %r12
fset 56 fset 48	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq %rbx .cfi_def_cfa_of popq %rbp .cfi_def_cfa_of		\$.LC3, %esi pxor %xmm0, %xmm0 mov1 \$1, %edi cvtsi2sdq %rax, %xmm0 movq 64(%rsp), %rax subq 48(%rsp), %rax movapd %xmm0, %xmm1	fset 48	popq %rbx .cfi_def_cfa_of popq %rbp .cfi_def_cfa_of popq %r12 .cfi_def_cfa_of
fset 56 fset 48	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq %rbx .cfi_def_cfa_of popq %rbp .cfi_def_cfa_of		\$.LC3, %esi pxor %xmm0, %xmm0 movl \$1, %edi cvtsi2sdq %rax, %xmm0 movq 64(%rsp), %rax subq 48(%rsp), %rax movapd %xmm0, %xmm1 pxor	fset 48	popq %rbx .cfi_def_cfa_of popq %rbp .cfi_def_cfa_of popq %r12 .cfi_def_cfa_of popq
fset 56 fset 48	addq \$72, %rsp .cfi_remember_s .cfi_def_cfa_of popq %rbx .cfi_def_cfa_of popq %rbp .cfi_def_cfa_of		\$.LC3, %esi pxor %xmm0, %xmm0 mov1 \$1, %edi cvtsi2sdq %rax, %xmm0 movq 64(%rsp), %rax subq 48(%rsp), %rax movapd %xmm0, %xmm1	fset 48	popq %rbx .cfi_def_cfa_of popq %rbp .cfi_def_cfa_of popq %r12 .cfi_def_cfa_of

Test 32						
Serial S	fset 32			.LC2(%rip),	fset 24	
Set 24 S			%xmm1			
First 24				•		
popq		.cfi_def_cfa_of				.cfi_def_cfa_of
Serial	fset 24				fset 16	
Set 16 Set 24 Set 25 Set 24 Set 24 Set 26 S						
fset 16						
ponq	5	.cfi_def_cfa_of		,	5	.cfi_def_cfa_of
Mari5	TSET 16	2000			rset 8	wa+
Coff_def_cfa_of ret ret movq 32(%rsp), %r15 movq 32(%rsp), %r15 salq movq (%rsp), %r15 salq movq (%rsp), %r15 salq movq movq (%rsp), %r15 salq movq mov						ret
Test S				•	. L4 i	ofi rootoro ot
.3: .efi_restore_st ate	foot 0	.cii_dei_cia_oi			ata	.cii_restore_st
1.3:	I SEL O	rot			ale	mova
S.LC4, Wesi	12:	rec				
movq	.13.	cfi restore st				
movq	ato	.011_1651016_51				
(%rsp), %r15 salq \$2, %rbx movq wr15, %rd1 call malloc movq %r15, %rd1 mov1 movq %r15, %rd1 mov1 movq %r15, %rd1 mov1 movq %r15, %rd1 mov1 mov1 sl, %ed1 mov1 sl, %ed1 mov1 slowrax, %rbp call mov1 slowrax, %rbp call mov1 slowrax, %edx soli slowray, %edx soli slowray, %edx movq %rax, %r14 call movq %rax, %r14 call call clock_gettime movq slowray, %rd1 call free movq wrbx, %rd1 call free movq slowray, %rd1 call free corl movq slowray, %rd2 call slowray, %rd2	ale	mova				
Salq movq call malloc movq fis, %rdi movq %ris, %rdi movq %ris, %rdi movq mo		·	%ray	-0(%113,%1d1),		
\$2, %rbx movq %ri5, %rdi call malloc movq %ri5, %rdi malloc movq %ri5, %rdi movq %rax, %rbp call malloc leaq leaq leaq leaq leaq; sort word %rax, %ri4 call movq %rax, %ri4 call movq %rax, %ri4 call movq %rax, %ri4 call free movq %ri5, %rdi call free movq %ri5, %rdi call free movq %ri5, %rdi call free movq %ri6, %rdi call free movq lo(%rsp), %rbi call free movq %rbx, %rdi call free xorl %eax, %eax movq %rbx, %rdi call free free movq lo(%rsp), %rbi call free free movq lo(%rsp), %rbi call free free movq lof(%rsp), %rbi call free free free free free free movq %rby, %rdi call free xorl %rb, %rbi jne lu41 addq silo4, %rsp .cfi_renember_s cfi_renember_s cfi_def_cfa_of fset 48 popq %rbx .cfi_def_cfa_of fset 48 popq %rby .cfi_def_cfa_of fset 32 popq %ri2 .cfi_def_cfa_of fset 24 popq %ri3 .cfi_def_cfa_of fset 24			701 UX	mova		
movq %ris, %rdi call %ecx movq (%ris), %rax movq %rax, %rdi), andloc movq %rax, %rdi movl movl movl movl movl movl movl movl		·				
Wr15, Wrdi Call Mecx movq m						
Call						
malloc movq wris, %rdi movq %rax, %rdp call malloc leaq %rax, %rbp call malloc leaq is(%rsy), %rsi worl wein, %edi movq %rax, %rdi call malloc leaq is(%rsy), %rsi worl wein, %edi movq seax, %eax call movq %rax, %rax wein movq is(%rsy), %rsi worl movq %rbx, %rdi call free movq %rbx, %rdi call free movq %rbx, %rdi call free wovq %rbx, %rdi call free xorl %eax, %eax movq %rbx, %rdi call free call free free movq %rbx, %rdi call free call free free movq %rbx, %rdi call free call free free movq %rbx, %rdi call free free free free free free free fr			%ecx	+(/01 dx,/01 d1),		
movq			,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	mova		
Wris, Wedi movq Si, Wedi leaq						
movq movl movl 16(%rsp), %rsi xorl xorl movl xorl movl xorl movl xorl movl movl movl xorl movl						
Wrax, %rbp call (%rax), %edx xorl		'				
Call (%rax), %edx xorl %edi, %edi %edi, %edi %edi, %edi movq fokrsp), %rsi call printf_chk movq %rax, %r14 call free movq 16(%rsp), %rdi call free movq %rbx, %rdi call free xorl %eax, %eax movq %fs:40, %rbx jne .141 addq %104, %rsp .cfi_def_cfa_of fset 48 popq %rbx .cfi_def_cfa_of fset 48 popq %rbx .cfi_def_cfa_of fset 32 fset 24 popq %r13 .cfi_def_cfa_of fset 24 popq %r14 popq popq %r14 popq po		·				
malloc leaq						
leaq						
16(%rsp), %rsi xorl						
Xedi, %edi		•				
%edi, %edi movq clock_gettime wrax, %r14 call free clock_gettime movq 16(%rsp), %rdi call free movq %rbx, %rdi call free movq 86(%rsp), %rbx xorq %fs:40, %rbx jne .141 addq %104, %rsp .cfi_def_cfa_of fset 56 popq %rbx .cfi_def_cfa_of fset 48 popq %rbp .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14 %r14						
movq %r15, %rdi call free clock_gettime movq l6(%rsp), %rdi call free movq wrbx, %rdi call free xorl weax, %eax movq movq 88(%rsp), %rbx xorq %fs:40, %rbx jne .L41 addq \$104, %rsp .cfi_remember_s .cfi_remember_s tate .cfi_def_cfa_of fset 56 popq %rbx .cfi_def_cfa_of fset 48 popq %rbx .cfi_def_cfa_of fset 40 popq %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14 %r14						
### Call call free movq 16(%rsp), %rdi call free movq %rbx, %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 %rby .cfi_def_cfa_of fset 40 popq %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14				•		9
call clock_gettime 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 %rby .cfi_def_cfa_of fset 40 popq %rby .cfi_def_cfa_of fset 32 popq %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14		·				
Clock_gettime						
16(%rsp), %rdi call free movq %rbx, %rdi call free xor1 %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 %rty .cfi_def_cfa_of fset 32 popq %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r13 .cfi_def_cfa_of						
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 %r13 .cfi_def_cfa_of fset 24 popq %r14		_3		•		
free movq %rbx, %rdi call free xorl %eax, %eax movq 88(%rsp), %rbx xorq %fs:40, %rbx jne .L41 addq \$104, %rsp .cfi_def_cfa_of fset 56 popq %rbx .cfi_def_cfa_of fset 48 popq %rbp .cfi_def_cfa_of fset 40 popq %rbp .cfi_def_cfa_of fset 32 popq %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 44 popq %r14						
movq %rbx, %rdi call free xorl						
#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 %r13 .cfi_def_cfa_of						
call free xor1 %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 %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of						
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 %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14						
## ## ## ## ## ## ## ## ## ## ## ## ##				free		
movq				xorl		
### ### ### ### ### ### ### ### ### ##				%eax, %eax		
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				movq		
#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				88(%rsp), %rbx		
jne				xorq		
.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						
\$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 %r13 .cfi_def_cfa_of				.L41		
.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				addq		
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 %r13 .cfi_def_cfa_of				.cfi_remember_s		
fset 56			tate			
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		
%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 %r13 .cfi_def_cfa_of			fset 56			
.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 %r13 .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			. .	.cfi_def_cfa_of		
%rbp .cfi_def_cfa_of fset 40 popq %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24			fset 48			
.cfi_def_cfa_of fset 40 popq %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14						
fset 40				•		
popq %r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14				.cti_def_cfa_of		
%r12 .cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14			rset 40			
.cfi_def_cfa_of fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14						
fset 32 popq %r13 .cfi_def_cfa_of fset 24 popq %r14						
popq %r13 .cfi_def_cfa_of fset 24 popq %r14			5 00	.cti_det_cfa_of		
%r13 .cfi_def_cfa_of fset 24 popq %r14			rset 32			
.cfi_def_cfa_of fset 24 popq %r14						
fset 24 popq %r14						
popq %r14				.cti_def_cfa_of		
%r14			rset 24	200		
cri_der_cra_or						
				.cri_det_cta_of		

```
fset 16
                 popq
                 %r15
                 .cfi_def_cfa_of
fset 8
.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]);
}</pre>
```

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

```
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves $gcc -02 -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):

c) Captura figura1-modificado_c.c

```
#include <stdio.h>
#include <stdib.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):

```
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves $gcc -02 -o figura1-mod-c figura1-modificado_c.c -lrt figura1-modificado_c.c:12:1: warning: return type defaults to 'int' [-Wimplicit-int] main()  

*[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves $./figura1-mod-c  
Tiempo(seg): 0.089142  
R[0]=0, R[39999]=-599985000  
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves $./figura1-mod-c  
Tiempo(seg): 0.091020  
R[0]=0, R[39999]=-599985000  
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej1] 2018-05-24 jueves $.
```

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		figura1-modificad	o_a.s	figura1-	modificado_b.s	figura1-mod	ificado_c.s
call	<pre>clock_gettime movl</pre>	call clock_g leaq 52(%rsp		call	clock_gettime movl \$1, %edx	call	<pre>clock_gettime movl \$1, %edx</pre>
s+39992(%rip),	leaq	movq %rax, 8 movl	(%rsp)	.L2:	.p2align 4,,10 .p2align 3	.L2:	.p2align 4,,10 .p2align 3
%rdx	36(%rsp), leal	\$1, %ea .p2alig .p2alig	n 4,,10		movl \$s, %eax xorl		movl \$s, %eax xorl
%edi	(%rax,%rax),	.L2: movl \$s, %ed	x		%r13d, %r13d xorl %r12d, %r12d		%r9d, %r9d xorl %r8d, %r8d
s+39996(%rip),	movl %eax	xorl %r12d, xorl			xorl %ebp, %ebp xorl		xorl %edi, %edi xorl
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	leal (%rax,	%ebp, % xorl %ebx, %	·		%ebx, %ebx xorl %edi, %edi		%esi, %esi xorl %ecx, %ecx
%rax,2), %r8d	movl \$1, %eax jmp	xorl %r11d, xorl %esi, %	%r11d		xorl %r11d, %r11d xorl %r10d, %r10d		xorl %r12d, %r12d xorl %ebp, %ebp
4,,10	.L2 .p2align	.p2alig .p2alig .L3:			xorl %r9d, %r9d xorl		xorl %ebx, %ebx xorl
.L5:	.p2align 3	movl (%rdx), addq \$40, %r			%r8d, %r8d xorl %esi, %esi .p2align 4,,10		%r11d, %r11d xorl %r10d, %r10d .p2align 4,,10
	<pre>\$1, %eax movl %esi, (%rdx)</pre>	leal (%rax,%		.L3:	.p2align 3 movl	.L3:	.p2align 3 movl
	addq \$4, %rdx	addl %ecx, % movl	esi		(%rax), %ecx addq \$40, %rax		(%rax), %r15d subl %edx, %ecx

		T					
	cmpl		-32(%rdx), %ecx		leal		subl
	\$40001, %eax		leal		(%rdx,%rcx,2),		%edx, %esi
	je ,		(%rax,%rcx,2),	%ecx			addl
	.L4	%ecx			addl		4(%rax), %ecx
	4		addl		%ecx, %esi		addl
.L2:			%ecx, %r11d		movl		12(%rax), %esi
	leal		movl		-32(%rax),		subl
	(%rax,%rdi),		-24(%rdx), %ecx	%ecx			%edx, %edi
%esi			leal		leal		subl
	movl		(%rax,%rcx,2),		(%rdx,%rcx,2),		%edx, %r8d
	%r8d, %ecx	%ecx		%ecx			addl
	•		addl		addl		20(%rax), %edi
	subl		%ecx, %ebx		%ecx, %r8d		addl
	%eax, %ecx		movl		mov1		28(%rax), %r8d
	cmpl		-16(%rdx), %ecx		-24(%rax),		addl
	%ecx, %esi		leal	%ecx	* **		%edx, %r15d
	jl		(%rax,%rcx,2),		leal		subl
	.L5	%ecx	, , , , , , , , , , , , , , , , , , , ,		(%rdx,%rcx,2),		%edx, %r9d
	addl		addl	%ecx	, , , , , , , , , , , , , , , , , , , ,		addq
			%ecx, %ebp		addl		\$40, %rax
	\$1, %eax		movl		%ecx, %r9d		add1
	movl		-8(%rdx), %ecx		movl		%r15d, %r10d
	%ecx, (%rdx)		leal		-16(%rax),		movl -
	addq		(%rax,%rcx,2),	%ecx	v · · · //	32(%rax), %r15d	
	\$4, %rdx	%ecx	, , , , , , , , , , , , , , , , , , , ,		leal	, , , ,	addl -
			addl		(%rdx,%rcx,2),	4(%rax), %r9d	•
	cmpl		%ecx, %r12d	%ecx	(, o., =) j	,,	addl
	\$40001, %eax		cmpq		addl		%edx, %r15d
	jne		%rdx, %r15		%ecx, %r10d		addl
	.L2		jne		mov1		%r15d, %r11d
.L4:			.L3		-8(%rax), %ecx		movl -
	leaq		movl		leal	24(%rax), %r15d	
	•		\$s+4, %edx		(%rdx,%rcx,2),	24(701 dx), 701 13d	addl
0/10 0 3	16(%rsp),		xorl	%ecx	(%1 0x, %1 0x, 2),		%edx, %r15d
%rsi			%edi, %edi	70CCX	addl		addl
	xorl		xorl				%r15d, %ebx
	%edi, %edi				%ecx, %r11d movl		movl -
	call		%r8d, %r8d			16(0/rox) 0/r1Ed	IIIOAT -
	clock_gettime		xorl	9/000	-36(%rax),	16(%rax), %r15d	addl
	orook_gottime		%r9d, %r9d	%ecx	1001		addl
			xorl		leal		%edx, %r15d
			%r10d, %r10d	0/004	(%rcx,%rcx,2),		addl
			xorl	%ecx	oub1		%r15d, %ebp
			%r13d, %r13d		subl	0/0/max) 0/m1Ed	movl -
			.p2align 4,,10		%edx, %ecx	8(%rax), %r15d	odd1
			.p2align 3		addl		addl
		.L4:	may/l		%ecx, %edi		%edx, %r15d
			mov1		mov1		addl
			(%rdx), %ecx	0/004	-28(%rax),		%r15d, %r12d
			addq \$40 %rdy	%ecx	1001		cmpq %rov %r13
			\$40, %rdx		leal		%rax, %r13
			leal	9/004	(%rcx,%rcx,2),		jne
		9/00/	(%rcx,%rcx,2),	%ecx	cub1		.L3
		%ecx	cub1		subl		addl
			subl		%edx, %ecx		%esi, %ecx
			%eax, %ecx		addl		leal
			addl %acy %r12d		%ecx, %ebx		(%r10,%r11), %eax
			%ecx, %r13d		movl		addl
			movl	9/004	-20(%rax),		%ecx, %edi
			-32(%rdx), %ecx	%ecx	lool		addl
			leal		leal		%ebx, %eax
		9/00/	(%rcx,%rcx,2),	9/004	(%rcx,%rcx,2),		addl %odi %red
		%ecx	cub1	%ecx	cub1		%edi, %r8d
			subl		subl		addl
			%eax, %ecx		%edx, %ecx		%ebp, %eax
			addl		addl		addl
			%ecx, %r10d		%ecx, %ebp		%r8d, %r9d
			movl		movl		addl
			-24(%rdx), %ecx	9/004	-12(%rax),		%r12d, %eax
			leal	%ecx	lool		leal
		0/0.04	(%rcx,%rcx,2),		leal		(%r9,%r9,2), %ecx
		%ecx	b.1	0/	(%rcx,%rcx,2),		addl
			subl	%ecx	oub1		%eax, %eax
			%eax, %ecx		subl		cmpl
			addl		%edx, %ecx		%ecx, %eax
			%ecx, %r9d		addl		jge
			movl		%ecx, %r12d		. L4
			-16(%rdx), %ecx		movl		movl
			leal		-4(%rax), %ecx	15.	%eax, (%r14)
I			(%rcx,%rcx,2),		leal	.L5:	

0/			(0/22 0/22 0)	- 4 - 1
%ecx			(%rcx,%rcx,2),	addl
	subl	%ecx		\$1, %edx
	%eax, %ecx		subl	addq
	addl		%edx, %ecx	\$4, %r14
	%ecx, %r8d		addl	cmpl
	movl		%ecx, %r13d	\$40001, %edx
	-8(%rdx), %ecx		cmpq	jne
	leal		%rax, %r14	.L2
	(%rcx,%rcx,2),		jne	leaq
%ecx			.L3	16(%rsp), %rsi
	subl		addl	xorl
	%eax, %ecx		%r8d, %esi	%edi, %edi
	addl		addl	call
	%ecx, %edi		%ebx, %edi	clock_gettime
	cmpq		addl	
	%rdx, %r14		%esi, %r9d	
	jne		addl	
	.L4		%edi, %ebp	
	addl		addl	
	%r11d, %esi		%r9d, %r10d	
	addl		addl	
	%r13d, %r10d		%ebp, %r12d	
	addl		addl	
	%esi, %ebx		%r10d, %r11d	
	addl		addl	
	%r10d, %r9d		%r12d, %r13d	
	addl		cmp1	
	%ebx, %ebp		%r13d, %r11d	
	addl		jge	
	%r9d, %r8d		.L4	
	addl		movl	
	%ebp, %r12d		%r11d, (%r15)	
	addl	.L5:	701 110, (701 13)	
	%r8d, %edi		addl	
	cmpl		\$1, %edx	
			addq	
	%edi, %r12d		\$4, %r15	
	jge			
	.L5 movq		cmpl \$40001, %edx	
	8(%rsp), %rdi		jne	
	movl		.L2	
.L6:	%r12d, (%rdi)		leaq	
	addl		16(%rsp), %rsi xorl	
	\$1, %eax		%edi, %edi	
	addq		call	
	\$4, 8(%rsp)		clock_gettime	
	cmpl			
	\$40001, %eax			
	jne			
	.L2			
	leaq			
	32(%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 (*Double precision- real Alpha X Plus Y*) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

for
$$(i=1; i \le N, i++)$$
 $y[i] = a*x[i] + y[i];$

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.	-O0	-Os	-02	-O3
PARA N=	0,004346	0,00085	0,000737	0,000603
1000000		9		

CAPTURAS DE PANTALLA

```
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-00 1000000
Tiempo(seg): 0.004340
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-00 1000000
Tiempo(seg): 0.004346
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-0s 1000000
Tiempo(seg): 0.002309
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-0s 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-02 1000000
Tiempo(seg): 0.000737
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-02 1000000
Tiempo(seg): 0.000674
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-03 1000000
Tiempo(seg): 0.001393
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-03 1000000
Tiempo(seg): 0.000736
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-03 1000000
Tiempo(seg): 0.000603
y[0]=0, y[N-1]=47999952
[JoseLuisPedrazaRoman josele@josele:~/Escritorio/practica4AC/Ej2] 2018-05-24 jueves
$./daxpy-03 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 ENSAMBLADOR, DONDE SE REDUCE ELNUMERO 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

daxpy00.s	daxpy0s.s	daxpy02.s	daxpy03.s
call clock_gettime	call clock_gettime	call clock_gettime	call
movl	movl	movl	
\$1, -148(%rbp)	\$1, %eax	%r14d, %esi	clock_gettime
jmp	.L5:	xorl	leaq
. L5	cmpl	%edx, %edx	
.L6:	%ebx, %eax	addq \$1,	4(%r12), %rcx
movq -	jg	%rsi	movq
128(%rbp), %rax	.L11	.p2align 4,,10	-
movl -	movslq	.p2align 3	104(%rbp), %rdx
148(%rbp), %edx	%eax, %rdx	.L6:	movq
movslq	incl	imull	%rcx,
%edx, %rdx	%eax	\$47,	%rax
movl	imull	4(%r13,%rdx,4), %ecx	andl
(%rax,%rdx,4),	\$47,	addl	\$15,
%eax	(%r14,%rdx,4), %ecx	%ecx, 4(%rbx,	%eax
imull -	addl	%rdx,4)	shrq
140(%rbp), %eax	%ecx,	addq \$1,	\$2,
movl	0(%r13,%rdx,4)	%rdx	%rax
%eax, %ecx	jmp	cmpq	negq
movq -	. L5	%rdx, %rsi	%rax
112(%rbp), %rax	.L11:	jne .L6	andl
movl -	leaq	.L7:	\$3,
148(%rbp), %edx	-56(%rbp),	leaq -	%eax
movslq	%rsi	64(%rbp), %rsi	cmpl
%edx, %rdx	xorl	xorl	%ebx,

movl	%edi, %edi	%edi, %edi	%eax
(%rax,%rdx,4),	decl	movslq	cmova
%eax	%ebx	%r14d, %r14	%ebx,
addl	movslq	call	%eax
%eax, %ecx	%ebx, %rbx	clock_gettime	
movq -	call		
112(%rbp), %rax	clock_gettime		
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			