2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas. Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): Intel(R) Core(TM) i7-4712MQ CPU @ 2.30GHz

Sistema operativo utilizado: Arch Linux

Versión de gcc utilizada: gcc (GCC) 6.3.1 20170109

Adjunte el contenido del fichero /proc/cpuinfo de la máquina en la que ha tomado las medidas

- 1. Para el núcleo que se muestra en la Figura 1 (ver guion de prácticas), 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

A) MULTIPLICACIÓN DE MATRICES:

CÓDIGO FUENTE: pmm-secuencial.c (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* pmm-secuencial.c
    Producto de dos matrices cuadradas, M y L.
    Para compilar usar (-lrt: real time library)
    gcc -02 pmv-secuencial.c -o pmv -lrt
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()

double **M ;
double **E ;
double **R;
unsigned int f ;
int i;
```

```
int main(int argc, char** argv){
   struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
    //Leer argumento de entrada (nº de filas y columnas)
   if (argc<2){
       printf("Error: Falta el número de filas y columnas.\n");
       exit(-1);
   f = atoi(argv[1]); // Máximo N = 2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
   M = (double **)malloc(f*sizeof(double*));
   for( i = 0 ; i < f ; i++) M[i] = (double*)malloc(f * sizeof(double)) ;</pre>
      = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) L[i] = (double*)malloc(f * sizeof(double)) ;</pre>
    // matriz resultado
   R = (double **)malloc(f*sizeof(double*)) ;
   for( i = 0 ; i < f ; i++) R[i] = (double*)malloc(f * sizeof(double)) ;</pre>
   if ( (L==NULL) || (M==NULL) || (R==NULL)){
       printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
   int j ;
   //Inicializar matrices
    for(int i = 0; i < f; i++){
        for(int j = 0; j < f; j++){
            M[i][j] = 0.5*(i+j);
           L[i][j] = (i+j) ;
           R[i][j] = 0;
       }
   clock_gettime(CLOCK_REALTIME,&cgt1);
    //Calcular multiplicación.
    for(i=0; i<f; i++){</pre>
        for(j=0 ; j<f ; j++){</pre>
           for(k=0 ; k<f ; k++){
               R[i][j] += M[i][k]*L[j][k];
           }
       }
   clock_gettime(CLOCK_REALTIME,&cgt2);
   ncgt+=(double) (cgt2.tv_sec-cgt1.tv_sec)+
       (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
   printf("\nR[0][0]=\%f \ ; \ R[n-1][n-1] \ = \\%f", \ R[0][0], \ R[f-1][f-1]) \ ;
   ncgt = ncgt ;
   printf("\nTiempo(seg.): \$11.9f\t / filas: \$u\t / columnas: \$u\t",
          ncgt,f,f);
   FILE * fp ;
   fp = fopen("salida", "a");
   if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\n%d\t%f",f,ncgt);
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) -explicación-:

He realizado la transposición de la matriz que debería recorrerse por columnas para que pueda recorrerse por filas y evitar así multitud de fallos de caché.

Modificación b) -explicación-:

He realizado la relocalización de las filas de las matrices a multiplicar para que estén alineadas en memoria (estén situadas al inicio de líneas de caché) para minimizar los fallos de caché.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) pmm-secuencial-modificado_a.c (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* pmm-secuencial.c
   Producto de dos matrices cuadradas, M y L.
   Para compilar usar (-lrt: real time library)
   gcc -02 pmv-secuencial.c -o pmv -lrt
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
double **M ;
double **L;
double **R ;
unsigned int f ;
int i;
int main(int argc, char** argv){
   struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
    //Leer argumento de entrada (nº de filas y columnas)
       printf("Error: Falta el número de filas y columnas.\n");
    f = atoi(argv[1]); // \textit{Máximo N} = 2^32 - 1 = 4294967295 \ (size of (unsigned int) = 4 \ B)
   M = (double **)malloc(f*sizeof(double*)) :
   for( i = 0 ; i < f ; i++)</pre>
       M[i] = (double*)malloc(f * sizeof(double));
     = (double **)malloc(f*sizeof(double*));
   for( i = 0 ; i < f ; i++)
       L[i] = (double*)malloc(f * sizeof(double));
    // matriz resultado
    R = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++)</pre>
       R[i] = (double*)malloc(f * sizeof(double)) ;
    if ( (L==NULL) || (M==NULL) || (R==NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
   int j ;
   //Inicializar matrices
    for(int i = 0; i < f; i++){
        for(int j = 0; j < f; j++){
            M[i][j] = 0.5*(i+j);
            L[i][j] = (i+j);
            R[i][j] = 0;
   }
    clock_gettime(CLOCK_REALTIME,&cgt1);
    // transposición del a matriz L para eliminar fallos de caché.
    for(i = 0 ; i < f ; i++)</pre>
        for(j = 0 ; j < f ; j++){}
            if(i != j){
               int aux = L[i][j] ;
                L[i][j] = L[j][i] ;
                L[j][i] = aux ;
            }
        }
    int k;
    //Calcular multiplicación.
    for(i=0; i<f; i++){</pre>
        for(j=0; j<f; j++){
            for(k=0 ; k<f ; k++){</pre>
                R[i][j] += M[i][k]*L[j][k];
```

b) pmm-secuencial-modificado_b.c (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

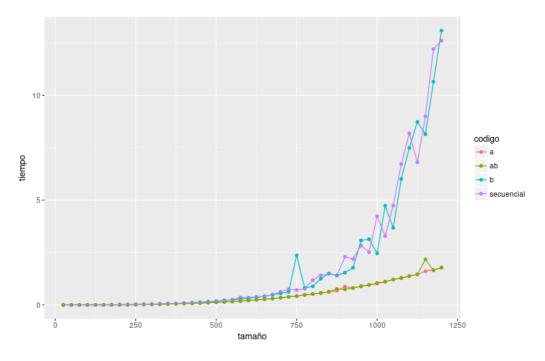
```
/* pmm-secuencial.c
   Producto de dos matrices cuadradas, M y L.
   Para compilar usar (-lrt: real time library)
  gcc -O2 pmv-secuencial.c -o pmv -lrt
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
double **M ;
double **MA ;
double **L ;
double **LA ;
double **R ;
double **RA ;
unsigned int f ;
int i:
const unsigned long BOUND = 64 ;
int main(int argc, char** argv){
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
    //Leer argumento de entrada (nº de filas y columnas)
    if (argc<2){
        printf("Error: Falta el número de filas y columnas.\n");
        exit(-1);
    f = atoi(argv[1]); // Máximo N = 2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
   unsigned int fil_tam = sizeof(double) * f ;
   unsigned int fil_lines = fil_tam / BOUND + 1;
   printf("\n\ntamaño de fila: %d, lineas por fila: %d\n\n", fil_tam, fil_lines) ;
   M = (double **)malloc(f*sizeof(double*));
    MA = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++){</pre>
        // reservo una linea más de las necesarias, menos 1B
       MA[i] = (double *) malloc(BOUND*(fil lines+1)-1) :
       M[i] = (double *) ((((unsigned long) MA[i]) + BOUND-1) & ~(BOUND-1));
    L = (double **)malloc(f*sizeof(double*)) ;
    LA = (double **)malloc(f*sizeof(double*));
    for( i = 0 ; i < f ; i++){</pre>
```

```
reservo una linea más de las necesarias, menos 1B
        LA[i] = (double *) malloc(BOUND*(fil_lines+1)-1);
        L[i] = (double *) ((((unsigned long) LA[i]) + BOUND-1) & ~(BOUND-1));
   }
   R = (double **)malloc(f*sizeof(double*)) ;
   RA = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++){
       // reservo una linea más de las necesarias, menos 1B
       RA[i] = (double *) malloc(BOUND*(fil_lines+1)-1);
       R[i] = (double *) ((((unsigned long) RA[i]) + BOUND-1) & ~(BOUND-1));
   if ( (L==NULL) || (M==NULL) || (R==NULL)){
       printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
   int j ;
   //Inicializar matrices
    for(int i = 0 ; i < f ; i++){</pre>
        for(int j = 0 ; j < f ; j++){</pre>
           M[i][j] = 0.5*(i+j);
            L[i][j] = (i+j) ;
           R[i][j] = 0;
   clock_gettime(CLOCK_REALTIME,&cgt1);
   int k :
    //Calcular multiplicación.
    for(i=0; i<f; i++){
        for(j=0; j<f; j++){
           for(k=0 ; k<f ; k++){
              R[i][j] += M[i][k]*L[j][k];
   clock_gettime(CLOCK_REALTIME,&cgt2);
   ncgt+=(double) (cgt2.tv_sec-cgt1.tv_sec)+
       (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
   printf("\nR[0][0]=%f; R[n-1][n-1] = %f", R[0][0], R[f-1][f-1]);
   ncgt = ncgt ;
   printf("\nTiempo(seg.): %11.9f\t / filas: %u\t / columnas: %u\t",
           ncgt,f,f) ;
   FILE * fp ;
    fp = fopen("salida","a");
   if(fp==NULL){perror("Error opening file.");}
   fprintf(fp,"\nb\t%d\t%f",f,ncgt);
fclose(fp);
    for(int i = 0 ; i < f ; i++){</pre>
        free(MA[i]);
        free(LA[i]);
        free(RA[i]);
    free(M); free(MA);
    free(L); free(LA);
free(R); free(RA);
}
```

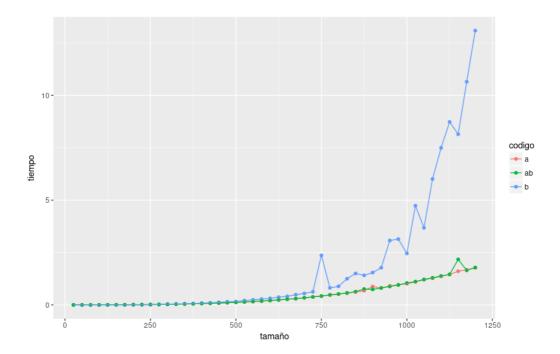
Capturas de pantalla (que muestren que el resultado es correcto):

```
pinguino 1N0 UGR ... Practicas 4 code % pmm-secuencial.c -o secuencial -02 pmm-secuencial -03 - o secuencial -04 code % pmm-secuencial -05 pmm-secuencial -05 - o secuencial -05 pmm-secuencial -05 pmm-sec
```

1.1. TIEMPOS y COMENTARIOS SOBRE LOS RESULTADOS:



Los tiempos conseguidos con la mejora b son solo un poco mejores que los tiempos para el código secuencial, dándose casos en los que el código es mejor y casos en los que es peor. Sin embargo, la mejora de la transposición de la matriz genera tiempos mucho mejores que las dos versiones anteriores.



*Nota: la razón por la que en clase me salían los tiempos muy parecidos era que, aunque había eliminado la parte de transposición del código b, seguía accediendo a la segunda matriz como si estuviera transpuesta (se me olvidó darle la vuelta a los índices) y no me dí cuenta de esto porque al inicializar la matriz estaba haciéndola simétrica, y los resultados eran correctos (en realidad, como la matriz era simétrica daba igual que hiciera la transposición o no, por lo que los códigos a y b eran iguales).

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

pmm-secuencial.s calcular_multiplicacion: .LFB21:		pmm-secuencial-modificado_b.s calcular_multiplicacion: .LFB21:			pmm-secuencial- modificado_c.s	
				calcular_mul	tiplicacion:	
. LI DZI.	.cfi_startproc		.cfi_startproc		.cfi_startproc	
	movl		movl		movl	
	f(%rip), %edi		f(%rip), %edi		f(%rip), %ecx	
	testl		testl		movq	
	%edi, %edi		%edi, %edi		L(%rip), %r9	
	je		je		testl	
	.L15		.L15		%ecx, %ecx	
	leal		leal		je	
	-1(%rdi), %eax		-1(%rdi), %eax		.L22	
	pushq		pushq		pushq	
	%rbp		%rbp		%rbx	
et 16	.cfi_def_cfa_offs	et 16	.cfi_def_cfa_offs	et 16	.cfi_def_cfa_off	
et 16	.cfi_offset 6,	60 10	.cfi_offset 6,	60.10	.cfi_offset 3,	
-16	.cri_orrsec o,	-16	.cri_orrsec o,	-16	.cri_orrsec 5,	
-	movq		movq		xorl	
	L(%rip), %r9		L(%rip), %r9		%r8d, %r8d	
	pushq		pushq	.L12:	,	
	%rbx		%rbx		leaq	
	.cfi_def_cfa_offs		.cfi_def_cfa_offs		0(,%r8,8), %r10	
et 24		et 24			movl	
	.cfi_offset 3,		.cfi_offset 3,		%r8d, %edi	
-24		-24			xorl	
	movq		movq		%eax, %eax	
	R(%rip), %rbp		R(%rip), %rbp		.p2align 4,,10	
	xorl		xorl	1.5	.p2align 3	
	%r11d, %r11d		%r11d, %r11d	.L5:	omp]	
	movq M(%rip), %rbx		movq M(%rip), %rbx		cmpl %eax, %edi	
	leag		leaq		jge	
	8(,%rax,8), %r10		8(,%rax,8), %r10		.L4	
.L6:	2(, 2,2,, 2.	.L6:	2(, 2,2,, 2.		movq	
	movq		movq		(%r9,%r8,8), %rd	
	0(%rbp,%r11,8),		0(%rbp,%r11,8),		leaq	
%rsi		%rsi			(%rdx,%rax,8),	
	movq		movq	%rsi		
	(%rbx,%r11,8),		(%rbx,%r11,8),		movq	
%r8		%r8			%r10, %rdx	
	xorl		xorl		addq	
	%edx, %edx		%edx, %edx		(%r9,%rax,8),	
	.p2align 4,,10		.p2align 4,,10	%rdx		
.L7:	.p2align 3	.L7:	.p2align 3		movsd (%rsi) %vmm0	
	movsd		movsd		(%rsi), %xmm0 movsd	
	(%rsi,%rdx),		(%rsi,%rdx),		(%rdx), %xmm1	
%xmm1	(o., ax),	%xmm1	(o., un),		movsd	
	xorl		xorl		%xmm1, (%rsi)	
	%eax, %eax		%eax, %eax		movsd	
	.p2align 4,,10		.p2align 4,,10		%xmm0, (%rdx)	
	.p2align 3		.p2align 3	.L4:		
.L4:		.L4:			addq	
· ·	movq		movq		\$1, %rax	
	(0/rQ 0/ray 0)		(%r9,%rax,8),		cmpl	
	(%r9,%rax,8),	%rcx			%eax, %ecx	
%rcx		701 CX		1	ja	
%rcx	movsd	701 CX	movsd		-	
			<pre>movsd (%rcx,%rdx),</pre>		.L5	
	movsd (%rcx,%rdx),	%xmm0	(%rcx,%rdx),		.L5 addq	
	<pre>movsd (%rcx,%rdx), mulsd</pre>		(%rcx,%rdx),		.L5 addq \$1, %r8	
%×mm0	movsd (%rcx,%rdx),	%xmm0	(%rcx,%rdx),		.L5 addq \$1, %r8 cmpl	
%×mm0	<pre>movsd (%rcx,%rdx), mulsd (%r8,%rax,8),</pre>		(%rcx,%rdx), mulsd (%r8,%rax,8),		.L5 addq \$1, %r8 cmpl %r8d, %ecx	
%×mm0	movsd (%rcx,%rdx), mulsd (%r8,%rax,8), addq	%xmm0	(%rcx,%rdx), mulsd (%r8,%rax,8), addq		.L5 addq \$1, %r8 cmpl %r8d, %ecx ja	
%xmm0	<pre>movsd (%rcx,%rdx), mulsd (%r8,%rax,8), addq \$1, %rax</pre>	%xmm0	(%rcx,%rdx), mulsd (%r8,%rax,8), addq \$1, %rax		.L5 addq \$1, %r8 cmpl %r8d, %ecx ja .L12	
%rcx %xmm0 %xmm0	movsd (%rcx,%rdx), mulsd (%r8,%rax,8), addq	%xmm0	(%rcx,%rdx), mulsd (%r8,%rax,8), addq		.L5 addq \$1, %r8 cmpl %r8d, %ecx ja	

%rdx) j a a \$ c c % j g et 16	%xmm0, %xmm1 movsd %xmm1, (%rsi, ja .L4 addq \$8, %rdx mpq %rdx, %r10 jne .L7 addq \$1, %r11 mpl %r11d, %edi ja .L6 popq %rbx .cfi_restore 3 .cfi_def_cfa_offs	%rdx)	<pre>%xmm0, %xmm1 movsd %xmm1, (%rsi, ja .L4 addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 poppq %rbx</pre>	.L8: %rsi %r8 .L11:	M(%rip), %r11 xorl %r10d, %r10d movq (%rbx,%r10,8), movq (%r11,%r10,8), xorl %edx, %edx .p2align 4,,10 .p2align 3 movq
%rdx) j a a \$ c c % j g et 16	movsd %xmm1, (%rsi, ja .L4 addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$i1, %r11 cmpl %r1ld, %edi ja .L6 copq %rbx .cfi_restore 3 .cfi_def_cfa_offs	%rdx)	movsd %xmm1, (%rsi, ja .L4 addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %rlld, %edi ja .L6 popq	%rsi %r8	<pre>xorl %r10d, %r10d movq (%rbx,%r10,8), movq (%r11,%r10,8), xorl %edx, %edx .p2align 4,,10 .p2align 3</pre>
%rdx) j a a \$ c % j g et 16	Kxmm1, (%rsi, ja L4 addq \$8, %rdx mpq Krdx, %r10 jne L7 addq \$1, %r11 mpl Kr11d, %edi ja L6 bopq Krbx .cfi_restore 3 .cfi_def_cfa_offs	%rdx)	<pre>%xmm1, (%rsi, ja .L4 addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq</pre>	%rsi %r8	<pre>%r10d, %r10d movq (%rbx,%r10,8), movq (%r11,%r10,8), xorl %edx, %edx .p2align 4,,10 .p2align 3</pre>
%rdx) j a a \$ c % j a j et 16	ja .L4 addq \$\$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$\$i, %r11 cmpl %r11d, %edi ja .L6 copq %rbx .cfi_restore 3 .cfi_def_cfa_offs	%rdx)	<pre>ja .L4 addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmp1 cmp1 ja .L6 popq</pre>	%rsi %r8	movq (%rbx,%r10,8), movq (%r11,%r10,8), xorl %edx, %edx .p2align 4,,10 .p2align 3
j	.L4 addq \$8, %rdx cmpq krdx, %r10 jne .L7 addq \$1, %r11 cmpl kr11d, %edi ja .L6 coppq krbx .cfi_restore 3 .cfi_def_cfa_offs	%rdx)	.L4 addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq	%rsi %r8	(%rbx,%r10,8), movq (%r11,%r10,8), xorl %edx, %edx .p2align 4,,10 .p2align 3
	.L4 addq \$8, %rdx cmpq krdx, %r10 jne .L7 addq \$1, %r11 cmpl kr11d, %edi ja .L6 coppq krbx .cfi_restore 3 .cfi_def_cfa_offs		.L4 addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq	%r8	(%rbx,%r10,8), movq (%r11,%r10,8), xorl %edx, %edx .p2align 4,,10 .p2align 3
	.L4 addq \$8, %rdx cmpq krdx, %r10 jne .L7 addq \$1, %r11 cmpl kr11d, %edi ja .L6 coppq krbx .cfi_restore 3 .cfi_def_cfa_offs		.L4 addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq	%r8	(%rbx,%r10,8), movq (%r11,%r10,8), xorl %edx, %edx .p2align 4,,10 .p2align 3
a a \$ c c % % j j	addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 copq %rbx .cfi_restore 3 .cfi_def_cfa_offs		addq \$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq	%r8	movq (%r11,%r10,8), xorl %edx, %edx .p2align 4,,10 .p2align 3
\$ cc % % j j	s8, %rdx cmpq krdx, %r10 jne .L7 addq \$i, %r11 cmpl kr1ld, %edi ja .L6 copq krbx .cfi_restore 3 .cfi_def_cfa_offs		\$8, %rdx cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq	%r8	(%rl1,%rl0,8), xorl %edx, %edx .p2align 4,,10 .p2align 3
et 8	cmpq %rdx, %r10 jne .L7 addq si, %r11 cmpl %r11d, %edi ja .L6 copq %rbx .cfi_restore 3 .cfi_def_cfa_offs		<pre>cmpq %rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq</pre>		(%rl1,%rl0,8), xorl %edx, %edx .p2align 4,,10 .p2align 3
% j . aa s s c % j p et 16	Krdx, %r10 jne LT addq \$1, %r11 mpl Kr11d, %edi ja L6 coppq Krbx .cfi_restore 3 .cfi_def_cfa_offs		<pre>%rdx, %r10 jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq</pre>		xorl %edx, %edx .p2align 4,,10 .p2align 3
et 16	jne .L7 addq \$1, %rll mpl %rlld, %edi ja .L6 copq %rbx .cfi_restore 3 .cfi_def_cfa_offs		<pre>jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq</pre>		<pre>%edx, %edx .p2align 4,,10 .p2align 3</pre>
et 16	jne .L7 addq \$1, %rll mpl %rlld, %edi ja .L6 copq %rbx .cfi_restore 3 .cfi_def_cfa_offs		<pre>jne .L7 addq \$1, %r11 cmpl %r11d, %edi ja .L6 popq</pre>		<pre>%edx, %edx .p2align 4,,10 .p2align 3</pre>
et 16	LT addq \$1, %rll mpl %rlld, %edi ja a.L6 boppq %rbx .cfi_restore 3 .cfi_def_cfa_offs		.L7 addq \$1, %rl1 cmpl %rlld, %edi ja .L6 popq	.L11:	<pre>%edx, %edx .p2align 4,,10 .p2align 3</pre>
a a \$ c c % % j j	addq \$1, %rll cmpl %rlld, %edi ja .L6 oopq %rbx .cfi_restore 3		addq \$1, %rl1 cmpl %rl1d, %edi ja .L6 popq	.tii:	.p2align 4,,10 .p2align 3
\$ cc % % j j	\$1, %r11 cmpl kr11d, %edi ja .L6 copq krbx .cfi_restore 3 .cfi_def_cfa_offs		\$1, %rl1 cmpl %rl1d, %edi ja .L6 popq	.L11:	.p2align 3
et 16 p et 8	empl Krild, %edi ja L6 oopq Krbx .cfi_restore 3 .cfi_def_cfa_offs		cmpl %r11d, %edi ja .L6 popq	.L11:	
% j . p % et 16	%r11d, %edi ja .L6 popq krbx .cfi_restore 3 .cfi_def_cfa_offs		%r11d, %edi ja .L6 popq	.L11:	movq
% j . p % et 16	%r11d, %edi ja .L6 popq krbx .cfi_restore 3 .cfi_def_cfa_offs		%r11d, %edi ja .L6 popq		movq
et 8	ja .L6 popq Krbx .cfi_restore 3 .cfi_def_cfa_offs		ja .L6 popq		movq
et 16 P	LL6 popq Krbx .cfi_restore 3 .cfi_def_cfa_offs		.L6 popq		(0, 0, 0, 1, 0)
et 16 P	oopq %rbx .cfi_restore 3 .cfi_def_cfa_offs		popq		(%r9,%rdx,8),
et 16	<pre>%rbx .cfi_restore 3 .cfi_def_cfa_offs</pre>			%rdi	
et 16	<pre>%rbx .cfi_restore 3 .cfi_def_cfa_offs</pre>				movsd
et 16	.cfi_restore 3 .cfi_def_cfa_offs				(%rsi,%rdx,8),
et 16	.cfi_def_cfa_offs	I		0/vmm1	(o., ax,o),
et 16			.cfi_restore 3	%xmm1	
p % et 8	popq		.cfi_def_cfa_offs		xorl
% • et 8	popq	et 16			%eax, %eax
% • et 8			popq		.p2align 4,,10
et 8	%rbp		%rbp		.p2align 3
et 8				10.	.pzacign 5
et 8	.cfi_restore 6		.cfi_restore 6	.L9:	
	.cfi_def_cfa_offs		.cfi_def_cfa_offs		movsd
r		et 8			(%r8,%rax,8),
	ret		ret	%×mm0	
					mulsd
					(%rdi,%rax,8),
				%×mm0	
					addq
					\$1, %rax
					cmpl
1					
					%eax, %ecx
					addsd
					%xmm0, %xmm1
					movsd
					%xmm1, (%rsi,
					ΔXIIIIII, (ΔΙ 5 Ι ,
				%rdx,8)	
					ja
					.L9
					addq
					\$1, %rdx
					cmpl
					%edx, %ecx
					ja
					.L11
					addq
					\$1, %r10
					cmpl
					%r10d, %ecx
					ja
					.L8
					popq
					%rbx
					.cfi_restore 3
					.cfi_def_cfa_offs
				I .	
				-+ 0	
				et 8	ret

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: figural-original.c (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
int main()
   struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
   int i. ii :
   struct {
        int a:
        int b:
   } s[5000];
   int X1, X2;
   int R[40000];
   // inicializar s ;
    for(i = 0 ; i < 5000 ; i++){</pre>
        s[i].a = i ;
        s[i].b = 4999 - i;
   clock_gettime(CLOCK_REALTIME,&cgt1);
    for (ii=0; ii<40000;ii++) {</pre>
        X1=0; X2=0;
        for(i=0; i<5000;i++) X1+=2*s[i].a+ii;</pre>
        for(i=0; i<5000;i++) X2+=3*s[i].b-ii;</pre>
        if (X1<X2)
            R[ii]=X1 ;
            R[ii]=X2;
   clock_gettime(CLOCK_REALTIME,&cgt2);
   ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
    // imprimir algunos valores de R:
    for(i = 0 ; i < 40000 ; i+=12345){</pre>
       printf("\nR[%d] = %d", i, R[i]);
   printf("\n\ntiempo: %f",ncgt) ;
   FILE * fp ;
    fp = fopen("figura1_salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\noriginal\t%f",ncgt);
    fclose(fp);
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación b) -explicación-: he realizado los cálculos de s[i].a*2 y s[i].b*3 previamente al bucle de ii para luego no tener que calcular dichas multiplicaciones todas las iteraciones. Además (aunque no sé si el compilador ya hará esto de por si), para ahorrarme las multiplicaciones y, dado que se trata de multiplicaciones por números pequeños (2 y 3) he calculado las multiplicaciones como sumas.

Modificación ab) He juntado las dos ideas haciendo el cálculo previo de las multiplicaciones y eliminando un bucle para mejorar el acceso a memoria.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figural-modificado_a.c

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
   int i, ii ;
   struct {
       int a:
        int b;
   } s[5000];
   int X1, X2;
   int R[40000] ;
   // inicializar s ;
   for(i = 0 ; i < 5000 ; i++){</pre>
        s[i].a = i :
        s[i].b = 4999 - i;
   clock_gettime(CLOCK_REALTIME,&cgt1);
    for (ii=0; ii<40000;ii++) {</pre>
        X1=0; X2=0;
        for(i=0; i<5000;i++){</pre>
           X1+=2*s[i].a+ii;
            X2+=3*s[i].b-ii;
        if (X1<X2)
           R[ii]=X1 ;
        else
            R[ii]=X2;
   clock_gettime(CLOCK_REALTIME,&cgt2);
   ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
   // imprimir algunos valores de R:
   for(i = 0 ; i < 40000 ; i+=12345){
    printf("\nR[%d] = %d", i, R[i]) ;</pre>
   printf("\n\ntiempo: %f",ncgt) ;
   FILE * fp ;
    fp = fopen("figura1_salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\nmodificadoA\t%f",ncgt);
    fclose(fp);
```

b) figural-modificado_b.c (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <xmmintrin.h>
int main()
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
    struct {
        int a;
        int b;
    } s[5000];
    int X1, X2;
    struct{
         int x1;
         int x2;
    } x[5000];
    int R[40000];
    // inicializar s ;
    for(i = 0 ; i < 5000 ; i++){
         s[i].a = i :
         s[i].b = 4999 - i;
    clock_gettime(CLOCK_REALTIME,&cgt1);
    for(i = 0 ; i < 5000 ; i++){
        x[i].x1 = 0 ; x[i].x2 = 0 ;
         x[i].x1 = s[i].a+s[i].a; //x2
         x[i].x2 = s[i].b+s[i].b+s[i].b; //x3
    for (ii=0; ii<40000;ii++) {</pre>
        X1=0; X2=0;
for(i=0; i<5000;i++){
            X1+=x[i].x1+ii;
        for(i=0; i<5000;i++){
            X2+=x[i].x2-ii;
         if (X1<X2)
            R[ii]=X1 ;
             R[ii]=X2;
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
         (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
    // imprimir algunos valores de R:
    for(i = 0 ; i < 40000 ; i+=12345){</pre>
        printf("\nR[%d] = %d", i, R[i]);
    printf("\n\ntiempo: %f",ncgt) ;
    FILE * fp ;
    fp = fopen("figural_salida","a");
if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\nmodificadoB\t%f",ncgt);
    fclose(fp);
```

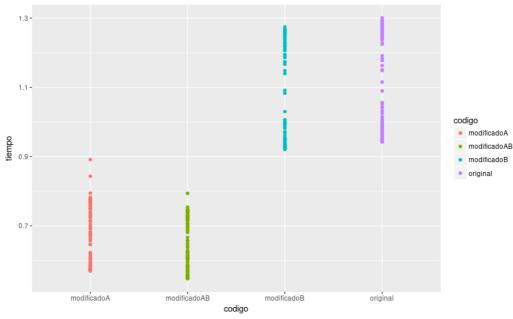
c) figural-modificado_ab.c (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <xmmintrin.h>
int main()
     struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
    int i, ii ;
    struct {
         int a;
         int b:
    } s[5000];
    int X1, X2;
    struct{
         int x1;
         int x2;
    } x[5000];
    int R[40000] ;
    // inicializar s ;
    for(i = 0 ; i < 5000 ; i++){</pre>
         s[i].a = i ;
         s[i].b = 4999 - i;
    clock_gettime(CLOCK_REALTIME,&cgt1);
    for(i = 0; i < 5000; i++){
    x[i].x1 = 0; x[i].x2 = 0;</pre>
         x[i].x1 = s[i].a+s[i].a; //x2
         x[i].x2 = s[i].b+s[i].b+s[i].b; //x3
     for (ii=0; ii<40000;ii++) {</pre>
         X1=0; X2=0;
         for(i=0; i<5000;i++){
            X1+=x[i].x1+ii;
             X2+=x[i].x2-ii;
         if (X1<X2)
             R[ii]=X1 ;
         else
              R[ii]=X2;
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
         (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
    // imprimir algunos valores de R:
     for(i = 0 ; i < 40000 ; i+=12345){</pre>
         printf("\nR[%d] = %d", i, R[i]);
    printf("\n\ntiempo: %f",ncgt) ;
    FILE * fp;
fp = fopen("figural_salida","a");
if(fp==NULL){perror("Error opening file.");}
fprintf(fp,"\nmodificadoAB\t%f",ncgt);
     fclose(fp);
```

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



1.1. TIEMPOS y COMENTARIOS SOBRE LOS RESULTADOS:

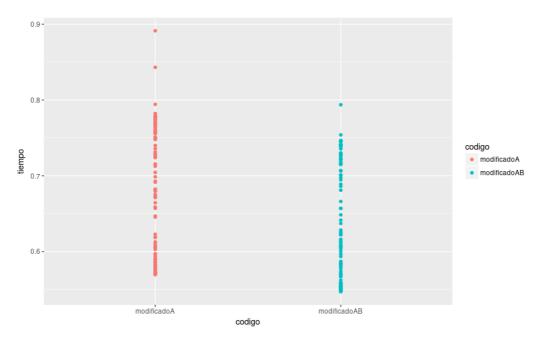


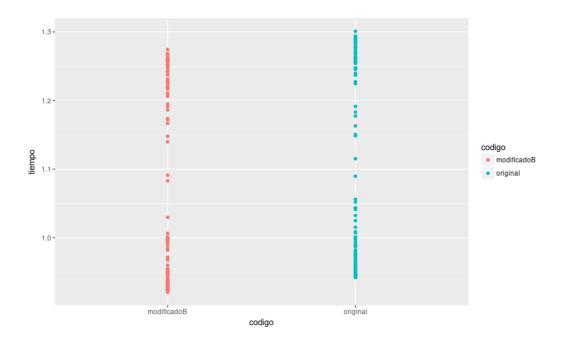
(No son tiempos en función del tamaño sino nubes de puntos con ejecuciones de los distintos códigos con un mismo tamaño para ver los intervalos de tiempos que se obtienen con cada uno)

Como se aprecia en la gráfica, los tiempos peores son los del código original (como es lógico) y los mejores, el resultado de combinar ambas mejoras (en verde), aunque la diferencia no es muy grande con respecto al código con la primera modificación (rojo) que es bastante mejor que el de la segunda mejora (azul) aunque esta tambien mejora un poco los tiempos del original. La conclusión que se puede extraer de aquí es que es

mucho más importante reducir los fallos de caché/accesos a memoria (mejora 1) que reducir el número de operaciones simples (como pueden ser las multiplicaciones). Si se tratara de divisiones en lugar de multiplicaciones probablemente la mejora B daría resultados bastante mejores con respecto al original pero aún así, lo más probable es que la mejora A siguiera siendo mejor porque está ahorrando accesos a memoria, que incluso utilizando cachés son extremádamente lentos en comparación con la velocidad a la que ejecuta instrucciones el procesador.

Para apreciar mejor las diferencias entre los códigos original y mejoraB y entre los códigos mejoraA y mejoraAB he hecho estas otras dos gráficas:





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

pmm-secuencial.s		pmm-secuencial- modificado_a.s		<pre>pmm-secuencial- modificado_ab.s</pre>	
	.file		.file		.file
	"figural-		"figura1-		"figura1-
original.c"		modificado_a.c"		modificado_ab.c"	
	.section		.section		.section
	.rodata		.rodata		.rodata
.LC1:		.LC1:		.LC1:	
	.string		.string		.string
	"\nR[%d] = %d"		"\nR[%d] = %d"		"\nR[%d] = %d"
.LC2:		.LC2:		.LC2:	
	.string		.string		.string
1.00	"\n\ntiempo: %f"	1.60	"\n\ntiempo: %f"	1.00	"\n\ntiempo: %f"
.LC3:		.LC3:	-4	.LC3:	
	.string "a"		.string "a"		.string "a"
.LC4:	"a"	1.64	"a"	1.64	"a"
. LC4:	string	.LC4:	ctring	.LC4:	string
	.string "figura1_salida"		.string "figura1_salida"		.string "figura1_salida"
.LC5:	i igui a±_satiua"	.LC5:	i igui a±_satiua"	.LC5:	i igui a±_satida"
	.string	.203.	.string		.string
	"Error opening		"Error opening		"Error opening
file."	27101 Opening	file."	2.101 opening	file."	ziroi opening
.LC6:		.LC6:		.LC6:	
	.string		.string		.string
	"\noriginal\t%f"		"\nmodificadoA\t		"\nmodificadoAB\
	.text	%f"	, , . , . , . , . , . , . , . , . ,	%f"	,
	.globl		.text		.text
	main		.globl		.globl
	.type		main		main
	main, @function		.type		.type
main:			main, @function		main, @function
.LFB0:		main:		main:	
	.cfi_startproc	.LFB0:		.LFB499:	
	pushq		.cfi_startproc		.cfi_startproc
	%rbp		pushq		pushq
	.cfi_def_cfa_offs		%rbp		%rbp
et 16			.cfi_def_cfa_offs		.cfi_def_cfa_off
	.cfi_offset 6,	et 16		et 16	
-16			.cfi_offset 6,		.cfi_offset 6,
	movq	-16		-16	
	%rsp, %rbp		movq		movq
	.cfi_def_cfa_regi		%rsp, %rbp		%rsp, %rbp
ster 6			.cfi_def_cfa_regi		.cfi_def_cfa_reg
	subq	ster 6		ster 6	
	\$200080, %rsp		subq		subq
	movl		\$200080, %rsp		\$240080, %rsp
	\$0, -4(%rbp)		movl \$0, -4(%rbp)		movl \$0, -4(%rbp)
	jmp .L2				
.L3:	. L.2		jmp .L2		jmp .L2
	movl	.L3:	•	.L3:	
	-4(%rbp), %eax		movl		movl
	cltq		-4(%rbp), %eax		-4(%rbp), %eax
	movl		cltq		cltq
	-4(%rbp), %edx		movl		movl
	movl		-4(%rbp), %edx		-4(%rbp), %edx
	%edx,		movl		movl
-40064(%rbp,%rax,8)	•		%edx,		%edx,
, , , , ,	movl	-40064(%rbp,%rax,8)	•	-40064(%rbp,%rax,8)	•
	\$4999, %eax		movl		movl
	subl		\$4999, %eax		\$4999, %eax
	-4(%rbp), %eax		subl		subl
	movl		-4(%rbp), %eax		-4(%rbp), %eax
	%eax, %edx		movl		movl
	movl		%eax, %edx		%eax, %edx
	-4(%rbp), %eax		movl		movl
	cltq		-4(%rbp), %eax		-4(%rbp), %eax
	movl	1	cltq	I	cltq

	%edx,		movl		movl
-40060(%rbp,%rax,8)	weux,		%edx,		%edx,
	addl	-40060(%rbp,%rax,8)		-40060(%rbp,%rax,8)	
	\$1, -4(%rbp)		addl		addl
.L2:	1		\$1, -4(%rbp)		\$1, -4(%rbp)
	cmpl \$4999, -4(%rbp)	.L2:	cmpl	.L2:	cmpl
	jle		\$4999, -4(%rbp)		\$4999, -4(%rbp)
	.L3		jle		jle
	leaq		.L3		.L3
	-48(%rbp), %rax		leaq		leaq
	movq %rax, %rsi		-48(%rbp), %rax		-48(%rbp), %rax
	movl		movq %rax, %rsi		movq %rax, %rsi
	\$0, %edi		movl		movl
	call		\$0, %edi		\$0, %edi
	clock_gettime		call		call
	movl		clock_gettime movl		clock_gettime movl
	\$0, -8(%rbp) jmp		\$0, -8(%rbp)		\$0, -4(%rbp)
	.L4		jmp		jmp
.L11:			.L4		.L4
	movl	.L9:		.L5:	
	\$0, -12(%rbp)		movl		movl
	movl \$0, -16(%rbp)		\$0, -12(%rbp) movl		-4(%rbp), %eax
	movl		\$0, -16(%rbp)		movl
	\$0, -4(%rbp)		movl		\$0, -80064(%rbp,
	jmp		\$0, -4(%rbp)	%rax,8)	. ,
l . <u>.</u>	.L5		jmp		movl
.L6:	1	16.	.L5		-4(%rbp), %eax
	movl -4(%rbp), %eax	.L6:	movl		cltq movl
	cltq		-4(%rbp), %eax		\$0, -80060(%rbp,
	movl		cltq	%rax,8)	, , , , , ,
	-40064(%rbp,		movl		movl
%rax,8), %eax			-40064(%rbp,		-4(%rbp), %eax
	leal (%rax,%rax), %edx	%rax,8), %eax	leal		cltq movl
	movl		(%rax,%rax), %edx		-40064(%rbp,
	-8(%rbp), %eax		movl	%rax,8), %edx	` ',
	addl		-8(%rbp), %eax		movl
	%edx, %eax		addl		-4(%rbp), %eax
	addl %eax, -12(%rbp)		%edx, %eax addl		cltq movl
	addl		%eax, -12(%rbp)		-40064(%rbp,
	\$1, -4(%rbp)		movl	%rax,8), %eax	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
.L5:			-4(%rbp), %eax		addl
	cmpl		cltq		%eax, %edx
	\$4999, -4(%rbp)		movl -40060(%rbp		movl
	jle .L6	%rax,8), %edx	-40060(%rbp,		-4(%rbp), %eax cltq
	movl	,,,,	movl		movl
	\$0, -4(%rbp)		%edx, %eax		%edx,
	jmp		addl	-80064(%rbp,%rax,8)	
.L8:	.L7		%eax, %eax addl		movl -4(%rbp), %eax
. 20.	movl		%edx, %eax		cltq
	-4(%rbp), %eax		subl		movl
	cltq		-8(%rbp), %eax		-40060(%rbp,
	movl		addl	%rax,8), %edx	,
eray 8) ead	-40060(%rbp,		%eax, -16(%rbp) addl		movl
%rax,8), %edx	movl		\$1, -4(%rbp)		-4(%rbp), %eax cltq
	%edx, %eax	.L5:	,-, (,		movl
	addl		cmpl		-40060(%rbp,
	%eax, %eax		\$4999, -4(%rbp)	%rax,8), %eax	
	addl %edx, %eax		jle .L6		addl %eax, %edx
	subl		movl		movl
	-8(%rbp), %eax		-12(%rbp), %eax		-4(%rbp), %eax
	addl		cmpl		cltq
	%eax, -16(%rbp)		-16(%rbp), %eax		movl
	addl		jge	0(524 0) 0(25::	-40060(%rbp,
.L7:	\$1, -4(%rbp)		.L7 movl	%rax,8), %eax	addl
	cmpl		-8(%rbp), %eax		%eax, %edx
	\$4999, -4(%rbp)		cltq		movl
	, , , , , , , , , , , , , , , , , , , ,				
	jle .L8		movl -12(%rbp), %edx		-4(%rbp), %eax cltq

				I	
	movl		movl		movl
	-12(%rbp), %eax		%edx,		%edx,
	cmpl	-200064(%rbp,%rax,4))	-80060(%rbp,%rax,8)	
	-16(%rbp), %eax		jmp		addl
	jge		.L8		\$1, -4(%rbp)
	.L9	.L7:		.L4:	, , , , , , , , ,
	movl		movl		cmpl
	-8(%rbp), %eax		-8(%rbp), %eax		\$4999, -4(%rbp)
	cltq		cltq		jle
	movl		movl		.L5
	-12(%rbp), %edx		-16(%rbp), %edx		movl
	movl		movl		\$0, -8(%rbp)
	%edx,		%edx,		jmp
-200064(%rbp,%rax,4	1)	-200064(%rbp,%rax,4))		.L6
	jmp	.L8:		.L11:	
	.L10		addl		movl
.L9:			\$1, -8(%rbp)		\$0, -12(%rbp)
	movl	.L4:			movl
	-8(%rbp), %eax		cmpl		\$0, -16(%rbp)
	cltq		\$39999, -8(%rbp)		movl
	movl		jle		\$0, -4(%rbp)
	-16(%rbp), %edx		.L9		jmp
	movl		leaq		.L7
	%edx,		-64(%rbp), %rax	.L8:	
-200064(%rbp,%rax,4			movq		movl
.L10:	,		%rax, %rsi		-4(%rbp), %eax
	addl		movl		cltq
	\$1, -8(%rbp)		\$0, %edi		movl
.L4:	4τ, ο(ωι nh)		call		
	cmnl			0/ray 0) 0/adv	-80064(%rbp,
	cmpl		clock_gettime	%rax,8), %edx	may.1
	\$39999, -8(%rbp)		movq		movl
	jle		-64(%rbp), %rdx		-8(%rbp), %eax
	.L11		movq		addl
	leaq		-48(%rbp), %rax		%edx, %eax
	-64(%rbp), %rax		subq		addl
	movq		%rax, %rdx		%eax, -12(%rbp)
	%rax, %rsi		movq		movl
	movl		%rdx, %rax		-4(%rbp), %eax
	\$0, %edi		pxor		cltq
	call		%xmm1, %xmm1		movl
	clock_gettime		cvtsi2sdq		-80060(%rbp,
	movq		%rax, %xmm1	%rax,8), %eax	
	-64(%rbp), %rdx		movq		subl
	movq		-56(%rbp), %rdx		-8(%rbp), %eax
	-48(%rbp), %rax		movq		addl
	subq		-40(%rbp), %rax		%eax, -16(%rbp)
	%rax, %rdx		subq		addl
	movq		%rax, %rdx		\$1, -4(%rbp)
	%rdx, %rax		movq	.L7:	71, ((OLDP)
	pxor		%rdx, %rax	1.2	cmpl
	%xmm1, %xmm1		pxor		\$4999, -4(%rbp)
	•		•		
	cvtsi2sdq		%xmm0, %xmm0		jle
	%rax, %xmm1		cvtsi2sdq		.L8
	movq		%rax, %xmm0		movl
	-56(%rbp), %rdx		movsd		-12(%rbp), %eax
	movq		.LC0(%rip), %xmm2		cmpl
	-40(%rbp), %rax		divsd		-16(%rbp), %eax
	subq		%xmm2, %xmm0		jge
	%rax, %rdx		addsd		.L9
	movq		%xmm1, %xmm0		movl
	%rdx, %rax		movsd		-8(%rbp), %eax
	pxor		%xmm0, -24(%rbp)		cltq
	%xmm0, %xmm0		movl		movl
	cvtsi2sdq		\$0, -4(%rbp)		-12(%rbp), %edx
	%rax, %xmm0		jmp		movl
	movsd		.L10		%edx,
	.LC0(%rip), %xmm2	.L11:		-240064(%rbp,%rax,4))
	divsd		movl		jmp
	%xmm2, %xmm0		-4(%rbp), %eax		.L10
		1	cltq	.L9:	
	addsd		Cttq		
	addsd		movl		movl
	addsd %xmm1, %xmm0		movl		
	addsd %xmm1, %xmm0 movsd	%rax,4), %edx	•		-8(%rbp), %eax
	addsd %xmm1, %xmm0 movsd %xmm0, -24(%rbp)	%rax,4), %edx	movl -200064(%rbp,		-8(%rbp), %eax cltq
	addsd %xmm1, %xmm0 movsd %xmm0, -24(%rbp) movl	%rax,4), %edx	movl -200064(%rbp,		-8(%rbp), %eax cltq movl
	addsd %xmm1, %xmm0 movsd %xmm0, -24(%rbp) movl \$0, -4(%rbp)	%rax,4), %edx	movl -200064(%rbp, movl -4(%rbp), %eax		-8(%rbp), %eax cltq movl -16(%rbp), %edx
	addsd %xmm1, %xmm0 movsd %xmm0, -24(%rbp) movl \$0, -4(%rbp) jmp	%rax,4), %edx	movl -200064(%rbp, movl -4(%rbp), %eax movl		-8(%rbp), %eax cltq movl -16(%rbp), %edx movl
113.	addsd %xmm1, %xmm0 movsd %xmm0, -24(%rbp) movl \$0, -4(%rbp)	%rax,4), %edx	movl -200064(%rbp, movl -4(%rbp), %eax movl %eax, %esi	-2/0064/9*hp 9*-2** 4*	-8(%rbp), %eax cltq movl -16(%rbp), %edx movl %edx,
.L13:	addsd %xmm1, %xmm0 movsd %xmm0, -24(%rbp) movl \$0, -4(%rbp) jmp .L12	%rax,4), %edx	movl -200064(%rbp, movl -4(%rbp), %eax movl %eax, %esi movl	-240064(%rbp,%rax,4)	-8(%rbp), %eax cltq movl -16(%rbp), %edx movl %edx,
.L13:	addsd %xmm1, %xmm0 movsd %xmm0, -24(%rbp) movl \$0, -4(%rbp) jmp	%rax,4), %edx	movl -200064(%rbp, movl -4(%rbp), %eax movl %eax, %esi	-240064(%rbp,%rax,4)	-8(%rbp), %eax cltq movl -16(%rbp), %edx movl %edx,

		1			
	cltq		\$0, %eax		\$1, -8(%rbp)
	movl		call	.L6:	1
0/ 4) 0/	-200064(%rbp,		printf		cmpl
%rax,4), %edx	movl		addl \$12345, -4(%rbp)		\$39999, -8(%rbp) jle
	-4(%rbp), %eax	.L10:	712545; 4(MIDP)		.L11
	movl		cmpl		leaq
	%eax, %esi		\$39999, -4(%rbp)		-64(%rbp), %rax
	movl		jle		movq
	\$.LC1, %edi		.L11		%rax, %rsi
	movl \$0, %eax		movq		movl \$0, %edi
	call		-24(%rbp), %rax movq		call
	printf		%rax,		clock_gettime
	addl	-200072(%rbp)			movq
	\$12345, -4(%rbp)		movsd		-64(%rbp), %rdx
.L12:	,		-200072(%rbp),		movq
	cmpl \$39999, -4(%rbp)	%xmm0	movl		-48(%rbp), %rax subq
	jle		\$.LC2, %edi		%rax, %rdx
	.L13		movl		movq
	movq		\$1, %eax		%rdx, %rax
	-24(%rbp), %rax		call		pxor
	movq		printf		%xmm1, %xmm1
200072 (2: 1)	%rax,		movl		cvtsi2sdq
-200072(%rbp)	movsd		\$.LC3, %esi movl		%rax, %xmm1
	-200072(%rbp),		\$.LC4, %edi		movq -56(%rbp), %rdx
%xmm0	- (> p/)		call		movq
	movl		fopen		-40(%rbp), %rax
	\$.LC2, %edi		movq		subq
	movl		%rax, -32(%rbp)		%rax, %rdx
	\$1, %eax		cmpq		movq
	call printf		\$0, -32(%rbp) jne		%rdx, %rax pxor
	movl		.L12		%xmm0, %xmm0
	\$.LC3, %esi		movl		cvtsi2sdq
	movl		\$.LC5, %edi		%rax, %xmm0
	\$.LC4, %edi		call		movsd
	call	110	perror		.LC0(%rip), %xmm2
	fopen movq	.L12:	movq		divsd %xmm2, %xmm0
	%rax, -32(%rbp)		-24(%rbp), %rdx		addsd
	cmpq		movq		%xmm1, %xmm0
	\$0, -32(%rbp)		-32(%rbp), %rax		movsd
	jne		movq		%xmm0, -24(%rbp)
	.L14		%rdx,		movl
	movl	-200072(%rbp)	movsd		\$0, -4(%rbp)
	<pre>\$.LC5, %edi call</pre>		-200072(%rbp),		jmp .L12
	perror	%×mm0	200012(30156);	.L13:	
.L14:	P		movl		movl
	movq		\$.LC6, %esi		-4(%rbp), %eax
	-24(%rbp), %rdx		movq		cltq
	movq		%rax, %rdi		movl
	-32(%rbp), %rax movq		movl \$1, %eax	%rax,4), %edx	-240064(%rbp,
	%rdx,		call	, ., ., ., ., ., ., ., ., ., ., .	movl
-200072(%rbp)	,		fprintf		-4(%rbp), %eax
	movsd		movq		movl
	-200072(%rbp),		-32(%rbp), %rax		%eax, %esi
%xmm0	1		movq		movl
	movl		%rax, %rdi		\$.LC1, %edi
	\$.LC6, %esi movq		call fclose		movl \$0, %eax
	%rax, %rdi		movl		call
	movl		\$0, %eax		printf
	\$1, %eax		leave		addl
	call		.cfi_def_cfa 7, 8		\$12345, -4(%rbp)
	fprintf		ret offi ondores	.L12:	cmnl
	movq -32(%rbp), %rax	.LFE0:	.cfi_endproc		cmpl \$39999, -4(%rbp)
	movq		.size		jle
	%rax, %rdi		main,main		.L13
	call		.section		movq
	fclose		.rodata		-24(%rbp), %rax
	movl		.align 8		movq
	\$0, %eax	.LC0:	long	240072(0/	%rax,
	leave .cfi_def_cfa 7, 8		.long 0	-240072(%rbp)	movsd
	.cii_uei_cia i, o				mo v su

Teff						
1.FE9:		ret		.long		-240072(%rbp),
### ### ### ### ### ### ### ### ### ##		.cfi_endproc		1104006501	%×mm0	
main, .main .section .secti	.LFE0:			.ident		movl
main, .main .section .secti		.size		"GCC: (GNU) 6.3.1		\$.LC2, %edi
.section .rodata .align 8 .al			20170109"			
rodata .align 8long				.section		
1.1ce						
LCO:			stack "" Anroghits	1110 00 10110		
Liong Lion	1.00.	· acigii o	scack, ,eprogores			
O	.100.	lana				
.long 1104096591 .ident "CCC: (GNU) 6.3.1 20170109" .section .note.GNU- stack,"",@progbits .li4: .novq .li4: .novq						
1104006591						
. ident "GCC: (ONU) 6.3.1 . ident "GCC: (ONU) 6.3.1 . section . note.GNU- stack,"",@progbits . L14: movq -24(krbp), %rax movq -24(krbp), %rax movq -24(krbp), %rax movq %rax, %rdi movd S.LG, %esi movq -22(krbp), %rax movq %rax, %rdi movl S.LG, %esi movq %rax, %rdi call fprintf movq -22(krbp), %rax movq %rax, %rdi call fprintf movq -22(krbp), %rax movq %rax, %rdi call fprintf movq -24(krbp), %rax movq %rax, %rdi call fprintf movq -24(krbp), %rax movq %rax, %rdi call fclose movl S.LG, %esi movq . section . long . lo						
"GCC: (GNU) 6.3.1 20170109" .section .note.GNU- Stack,"",@progbits .section .note.GNU- Stack,"",@progbits .li4: .section .s						
20170109" .section .note.GNU- stack,"",@progbits .section .note.GNU- stack,"",@progbits .section .note.GNU- stack,"",@progbits .li4: movq -24(mrbp), %rak movq -240072(%rbp) movad -240072(%rbp) movad -240072(%rbp), %rax movq %rax, %rdi mo						
.section .note.ONU- stack,"",@progbits .L14: .L14: L14:		"GCC: (GNU) 6.3.1				
.note.GNU- stack,"",@progbits .li4 .li4 .li4 .movq	20170109"					%rax, -32(%rbp)
Jine Lild movl S.LC5, %edi call perror lild movl S.LC5, %edi call perror lild movq -24(%rbp), %rdx movq -24(%rbp), %rdx movq rds r						cmpq
.1.14 mov1 s.LC5, %ed; call perror		.note.GNU-				\$0, -32(%rbp)
S.L.C5, %edicall perror	stack,"",@progbits					jne
S.LCS, %edicall perror						.L14
L14:						movl
L14:						\$.LC5. %edi
L14:						
1.14: movq						
movq					114.	perror
-24(%rbp), %rdx movq -32(%rbp), %rax mova %rdx, -240072(%rbp) movsd -240072(%rbp), %nov solution movl \$1.06, %esi movq %rax, %rdi movq \$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$6, %eax eave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .lC0: .long					1.514.	mova
movq -32(%rbp), %rax movq %rdx, -240072(%rbp) movsd -240072(%rbp), %xmm0 movl \$.LC6, %esi movq %rax, %rdi movl \$1, %eax call fprintf movq %rax, %rdi call fprintf movq %rax, %rdi call fclose movl \$8, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 0 .long 0 0 .long 0 0 .long 1lo4006501 .ident "GCC: (GNU) 6.3						
-32(%rbp), %rax movq %rdx, -240072(%rbp) movsd -240072(%rbp), movl \$.LC6, %esi movq %rax, %rdi movl \$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$6, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long						
movd %rdx -240072(%rbp) movsd -240072(%rbp),						
### ##################################						
-240072(%rbp) movsd -240072(%rbp), %xmm0 movl \$.LC6, %esi movq %rax, %rdi movl \$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$6, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.6NU-						
movad						%rdx,
-240072(%rbp), %xmm0 mov1 \$.LC6, %esi movq %rax, %rdi mov1 \$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$5, %eax leave .cfi_def_cfa 7, ret .cff_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-					-240072(%rbp)	
%xmm0						movsd
movl \$.LC6, %esi movq %rax, %rdi movl \$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						-240072(%rbp),
\$.LC6, %esi movq %rax, %rdi movl \$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-					%×mm0	
movq %rax, %rdi movl \$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$6, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC (GNU) 6.3 20170109" .section .note.GNU-						movl
movq %rax, %rdi movl \$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$6, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC (GNU) 6.3 20170109" .section .note.GNU-						\$.LC6, %esi
### ##################################						
movl \$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .IFE499: .size main,main .section .rodata .align 8 .LCO: .long 0 .long 1164005501 .ident "GCC: (GNU) 6.3 29170109" .section .note.GNU-						
\$1, %eax call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006551 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
call fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
fprintf movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: long 0 .long 104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
movq -32(%rbp), %rax movq %rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104090501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
-32(%rbp), %rax movq %rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
movq %rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
#rax, %rdi call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
call fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .IFE499: .size main,main .section .rodata .align 8 .LCO: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
fclose movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
movl \$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
\$0, %eax leave .cfi_def_cfa 7, ret .cfi_endproc .FE499: .size main,main .section .rodata .align 8 .LCO: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						fclose
leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						movl
leave .cfi_def_cfa 7, ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						\$0, %eax
.cfi_def_cfa 7,						leave
ret .cfi_endproc .LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						.cfi_def_cfa 7, 8
.cfi_endproc .LFE499: .size						
.LFE499: .size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
.size main,main .section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-					. I FF499 •	. cchapi oc
main,main .section .rodata .align 8 .LCO: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						sizo
.section .rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
.rodata .align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						•
.align 8 .LC0: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
.LCO: .long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
.long 0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-					1	.align 8
0 .long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-					.LC0:	
.long 1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						
1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						0
1104006501 .ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						.long
.ident "GCC: (GNU) 6.3 20170109" .section .note.GNU-						1104006501
"GCC: (GNU) 6.3 20170109" .section .note.GNU-						
20170109" .section .note.GNU-						
.section .note.GNU-					20170109"	100. (0.10) 0.011
.note.GNU-					231,0103	section
stack,"",@progbits						.note.GNU-
					stack,"",@progbits	

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):

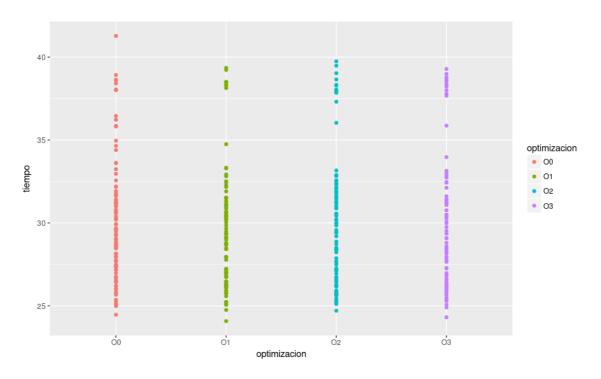
```
for (i=1; i \le N, i++) y[i] = a*x[i] + y[i];
```

- 2.1. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O0, -O2, -O3) y explique las diferencias que se observan en el código justificando 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.

CÓDIGO FUENTE: daxpy.c (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Ejemplo de Programa de Prueba */
#include <stdio.h>
#include <time.h>
int suma_prod(int a, int b, int n)
        return a * b + n;
int main()
        int i,j,a,b,n,c;
        clock_t start,stop;
        start = clock();
        n=6000;a=1;b=2;
        for (j=1;j<=10000;j++)
                printf("a=%d b=%d n=%d\n",a,b,n);
                c=suma prod(a,b,n):
                printf("resultado= %d\n",c);
        stop = clock();
        printf("Tiempo= %f ms\n",difftime(stop,start) / (CLOCKS_PER_SEC / 1000.0));
        return 0;
```

Tiempos:



CAPTURAS DE PANTALLA:

```
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
esultado= 6002
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
Tiempo= 32.524000 ms
```

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

las optimizaciones con -O2 y con -O3 dan como resultado el mismo código para la función de suma y multiplica. Estas opciones utilizan tan solo una instrucción imul y una leal mientras que la opción con -O1 necesita hacer uso de otras muchas instrucciones para trabajar con los argumentos que se le pasan a la función y utiliza un imul y un add+mov en lugar de un leal. Las mejoras no son muy relevantes pero se deben principalmente al uso de menos instrucciones en la función y al uso de leal para evitar hacer la suma y un mov.

CÓDIGO EN ENSAMBLADOR (ADJUNTAR AL .ZIP): (PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpy00.s		daxpy	daxpy02.s		daxpy03.s	
suma_prod: .LFB0:		suma_prod: .LFB14:		suma_prod: .LFB14:		
	.cfi_startproc		.cfi_startproc		.cfi_startproc	
	pushq		imull		imull	
	%rbp		%esi, %edi		%esi, %edi	
	<pre>.cfi_def_cfa_offs</pre>		leal		leal	
et 16			(%rdi,%rdx), %eax		(%rdi,%rdx), %eax	
	.cfi_offset 6,		ret		ret	
-16						
	movq					
	%rsp, %rbp					
	.cfi_def_cfa_regi					
ster 6						
	movl					
	%edi, −4(%rbp)					
	movl					
	%esi, −8(%rbp)					
	movl					
	%edx, −12(%rbp)					
	movl					
	-4(%rbp), %eax					
	imull					
	-8(%rbp), %eax					
	movl					
	%eax, %edx					
	movl					
	-12(%rbp), %eax					
	addl					
	%edx, %eax					
	popq					
	%rbp					
	.cfi_def_cfa 7, 8					
	ret					