

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): Antonio Miguel Morillo Chica

Grupo de prácticas: D1

Fecha de entrega: 08/06/2016

Fecha evaluación en clase: 02/06/2016

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz

Sistema operativo utilizado: Arch Linux

Versión de gcc utilizada: gcc (GCC) 6.1.1 20160501

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)

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

int main(int argc, char **argv){
    int num_filas=atoi(argv[1]);
    int num_columnas=atoi(argv[1]);
    int chunk=atoi(argv[2]);
    int **matriz1, **matriz2, **resultado;

    matriz1 = (int **)malloc(num_filas*sizeof(int*));
    matriz2 = (int **)malloc(num_filas*sizeof(int*));
    resultado=(int **)malloc(num_filas*sizeof(int*));
    int fila,columna,j;
    struct timespec cgt1,cgt2; double ncgt;

    for (fila=0;fila<num_filas;fila++)
        matriz1[fila] = (int*)malloc(num_filas*sizeof(int));

    for (fila=0;fila<num_filas;fila++)
        matriz2[fila] = (int*)malloc(num_filas*sizeof(int));

    for (fila=0;fila<num_filas;fila++)
        resultado[fila] = (int*)malloc(num_filas*sizeof(int));

    for(fila=0; fila<num_filas; fila++){
        for(columna=0; columna<num_columnas; columna++){
            matriz1[fila][columna]=2;
        }
    }

    printf("Mostramos la matriz1\n");
    for(fila=0; fila<num_filas; fila++){
        for(columna=0; columna<num_columnas; columna++){
            printf(" %d ", matriz1[fila][columna] );
        }
        printf("\n");
    }

    for(fila=0; fila<num_filas; fila++){
        for(columna=0; columna<num_columnas; columna++){
            matriz2[fila][columna]=2;
        }
    }

    printf("Mostramos la matriz2\n");
    for(fila=0; fila<num_filas; fila++){
        for(columna=0; columna<num_columnas; columna++){
            printf(" %d ", matriz2[fila][columna] );
        }
        printf("\n");
    }

    printf("Multiplicamos la matrices\n");
    pmm-secuencial
    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (fila = 0 ; fila < num_filas ; fila++ ){
```

```

        for (columna = 0 ; columna < num_columnas; columna++ ){
            int producto = 0 ;
            for (j = 0 ; j < num_columnas ; j++ ){
                producto += matriz1[fila][j] * matriz2[j][columna];
                resultado[fila][columna] = producto ;
            }
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9));
    printf("Tiempo(seg.):%11.9f\t",ncgt);

    printf("Mostramos el resultado\n");
    for(fila=0; fila<num_filas; fila++){
        for(columna=0; columna<num_columnas; columna++){
            printf(" %d ", resultado[fila][columna] );
        }
        printf("\n");
    }

    for (fila=0;fila<num_filas;fila++)
        free(matriz1[fila]);
    free(matriz1);

    for (fila=0;fila<num_filas;fila++)
        free(matriz2[fila]);
    free(matriz2);

    for (fila=0;fila<num_filas;fila++)
        free(resultado[fila]);
    free(resultado);
}

```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación–: La primera modificación ha sido muy grande, literalmente he cambiado todo el pmm-secuencial.c de la práctica anterior. En primer lugar he cambiado la declaración de las matrices, ya no uso ningún for para reservar memoria ni las matrices son punteros a punteros, sino solo punteros, es decir, lo organizo en un vector, directame. La segunda modificación consiste en rellenar las matrices, antes las rellenaaba con un solo valor y siempre el mismo, ahora se rellenan con valores dependientes a las iteración del bubble que le corresponda. Por último he modificado el algoritmo de la multiplicar matrices, antes guardaba el resultado de la operación en una variable y a continuación la guardaba, ahora lo hago todo en una misma sentencial.

Modificación b) –explicación–: Lo primero que he modificado ha sido modificar el if que comprueba si se ha podido reservar memoria. Como cada && supone un salto lo que he hecho ha sido modificarlo de esta forma (A || B || C) == 0, que realiza menos saltos. También he modificado el algoritmo para multiplicar matrices cuadradas, ahora el último for hace iteraciones que aumentan en cuatro, 4, 8, 12... Así en cada iteración realizo 4 operaciones.

1.1. CÓDIGOS FUENTE MODIFICACIONES**a) pmm-secuencial-modificado_a.c**

```

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

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

    int N, i, j, k, suma=0,c;
    int *A, *B, *C;
    struct timespec cgt1,cgt2;

    if(argc < 2){
        printf("Introduzca El numero de filas/columnas:\n");
        exit(-1);
    }

    N = atoi(argv[1]);

    // reservamos memoria
    A = malloc(sizeof(int) * N * N);
    B = malloc(sizeof(int) * N * N);
    C = malloc(sizeof(int) * N * N);

    if(A == NULL || B == NULL || C == NULL){
        printf("No se ha podido reservar memoria\n");
        exit(-1);
    }

    // inicializamos la matriz A y B
    for(i=0; i<N; ++i){
        for(j=0; j<N; ++j){
            A[i*N+j] = i+j+1;
            B[i*N+j] = i+j*2;
            C[i*N+j] = 0;
        }
    }

    // algoritmo de multiplicacion de matrices cuadradas
    clock_gettime(CLOCK_REALTIME,&cgt1);
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k+=4){
                C[i*N+j] += A[i*N+k] * B[k*N+j];
                C[i*N+j] += A[i*N+(k+1)] * B[(k+1)*N+j];
                C[i*N+j] += A[i*N+(k+2)] * B[(k+2)*N+j];
                C[i*N+j] += A[i*N+(k+3)] * B[(k+3)*N+j];
            }

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

    printf("Resultado:\nC[0][0]=%d, C[%d][%d]=%d\n", C[0], N-1, N-1, C[(N-1)*N+(N-1)]);
    printf("Tiempo transcurrido en segundos: %0.6f\n", ncgt);

    // liberamos memoria
    free(A); free(B); free(C);
}

```

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

```

Codigo : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[mike@looper Codigo]$ ./pmm-secuencial-modificado_a 200
Resultado:
C[0][0]=2666600, C[199][199]=30466900
Tiempo transcurrido en segundos: 0.008574
[mike@looper Codigo]$ ./pmm-secuencial-modificado_b 200
Resultado:
C[0][0]=2666600, C[199][199]=30466900
Tiempo transcurrido en segundos: 0.019194
[mike@looper Codigo]$

```

b) pmm-secuencial-modificado_b.c

```

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

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

    int N, i, j, k, suma=0, c;
    int *A, *B, *C;
    struct timespec cgt1,cgt2;

    if(argc < 2){
        printf("Introduzca El numero de filas/columnas:\n");
        exit(-1);
    }

    N = atoi(argv[1]);

    // reservamos memoria
    A = malloc(sizeof(int) * N * N);
    B = malloc(sizeof(int) * N * N);
    C = malloc(sizeof(int) * N * N);

    if((A || B || C) == 0){
        printf("No se ha podido reservar memoria\n");
        exit(-1);
    }

    // inicializamos la matriz A y B
    for(i=0; i<N; ++i){
        for(j=0; j<N; ++j){
            A[i*N+j] = i+j+1;
            B[i*N+j] = i+j*2;
            C[i*N+j] = 0;
        }
    }

    // algoritmo de multiplicacion de matrices cuadradas
    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(i=0; i<N; ++i)
        for(j=0; j<N; ++j)

```

```

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

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

    printf("Resultado:\nC[0][0]=%d, C[%d][%d]=%d\n", C[0], N-1, N-1, C[(N-1)*N+(N-1)]);
    printf("Tiempo transcurrido en segundos: %.6f\n", ncgt);

    // liberamos memoria
    free(A); free(B); free(C);
}

```

1.1. TIEMPOS:

Modificación	-O2
Sin modificar	<i>De 2 a 10 seg. aquí</i>
Modificación a)	0.008574
Modificación b)	0.019194

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Al principio modb era más rápido pero a medida que he ido ejecutando unas 10 veces mostraba casi siempre tiempos superiores. Creo que es debido a que en el último bucle hay demasiadas operaciones y es por ello que le cuesta más tiempo.

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	pmm-secuencial-modificado_b.s
<pre> .file "pmm-secuencial-modificado_a.c" .section .rodata ta.str1.8,"aMS",@progbits,1 .align 8 .LC0: .string "Introduzca El numero de filas/columnas:" .align 8 .LC1: .string "No se ha podido reservar memoria" .align 8 .LC3: .string "Resultado:\nC[0][0]=%d, C[%d][%d]=%d\n" .align 8 .LC4: .string "Tiempo transcurrido en segundos: %.6f\n" .section .text .startup,"ax",@progbits .p2align 4,,15 .globl main .type main, @function </pre>	<pre> .file "pmm-secuencial-modificado_a.c" .section .rodata rodata .align 8 .LC0: .string "Introduzca El numero de filas/columnas:" .align 8 .LC1: .string "No se ha podido reservar memoria" .align 8 .LC3: .string "Resultado:\nC[0][0]=%d, C[%d] [%d]=%d\n" .align 8 .LC4: .string "Tiempo transcurrido en segundos: %.6f\n" .text .globl main .type main, </pre>	<pre> .file "pmm-secuencial-modificado-b.c" .section .rodata odata.str1.8,"aMS",@progbits,1 .align 8 .LC0: .string "Introduzca El numero de filas/columnas:" .align 8 .LC2: .string "Resultado:\nC[0][0]=%d, C[%d] [%d]=%d\n" .align 8 .LC3: .string "Tiempo transcurrido en segundos: %.6f\n" .align 8 .LC4: .string "No se ha podido reservar memoria" .section .text ext.startup,"ax",@progbits .p2align 4,,15 .globl main </pre>

main: .LFB21:		main: .LFB2:	main, @function	main: .LFB21:	.type main, @function
	.cfi_startproc				
	pushq %r15		.cfi_startproc		pushq
	.cfi_def_cfa_offset		pushq		.cfi_def_cfa_offs
16			.cfi_def_cfa_off		
	.cfi_offset 15, -16	set 16	.cfi_offset 6,	et 16	.cfi_offset 15,
	pushq %r14	-16		-16	pushq
	.cfi_def_cfa_offset		movq		.cfi_def_cfa_offs
24			%rsp, %rbp		%r14
	.cfi_offset 14, -24		.cfi_def_cfa_reg		
	pushq %r13	ister 6		et 24	.cfi_offset 14,
	.cfi_def_cfa_offset		addq	-24	pushq
32			\$-128, %rsp		%r13
	.cfi_offset 13, -32		movl	et 32	.cfi_def_cfa_offs
	pushq %r12		%edi, -100(%rbp)	-32	.cfi_offset 13,
	.cfi_def_cfa_offset		movq		pushq
40			%rsi, -112(%rbp)	et 40	%r12
	.cfi_offset 12, -40		movl	-40	.cfi_def_cfa_offs
	pushq %rbp		\$0, -16(%rbp)		.cfi_offset 12,
	.cfi_def_cfa_offset		cmpl	et 48	pushq
48			\$1, -100(%rbp)	-48	%rbp
	.cfi_offset 6, -48		jg		.cfi_def_cfa_offs
	pushq %rbx		.L2		
	.cfi_def_cfa_offset		movl	et 56	.cfi_offset 3,
56			\$104,	-56	subq
%rsp			.LC0, %edi	et 112	\$56, %rsp
	.cfi_def_cfa_offset		call		.cfi_def_cfa_offs
160			puts		cmpl
%edi	cmpl \$1,	.L2:	movl		\$1, %edi
	jle .L28		\$-1, %edi		jle
	movq		call		.L30
	8(%rsi), %rdi		exit		movq
	movl \$10,	112(%rbp), %rax			8(%rsi), %rdi
%edx			addq		movl
%esi	xorl %esi,		\$8, %rax		\$10, %edx
	call		movq		xorl
	strtol		(%rax), %rax		%esi, %esi
	movslq %eax,		movq		call
%r13	movq %rax,		%rax, %rdi		strtol
%r14	movq %rax,		atoi		movslq
40(%rsp)	movq %r13,	20(%rbp), %eax	movl		%eax, %rbp
%r12	movl %eax,		%eax, -20(%rbp)		movq
56(%rsp)	imulq %r13,	20(%rbp), %eax	movl		%rax, (%rsp)
%r12	salq \$2,		movslq		movl
%r12	movq %r12,		%eax, %rdx		%eax, %r15d
%rdi	call		movl		%rbp, %rbx
	malloc		cltq		imulq
%rdi	movq %r12,		imulq		%rbp, %rbx
%rbx	movq %rax,		%rdx, %rax		salq
	call		salq		\$2, %rbx
	malloc		\$2, %rax		
%rdi	movq %r12,		movq		
	movq %rax,		%rax, %rdi		
	call		call		
	malloc		malloc		
	movq %r12,		movq		
	movq %rax,		%rax, -32(%rbp)		
	call		movl		
	malloc		movslq		
%rdi	movq %r12,	20(%rbp), %eax	%eax, %rdx		
	movq %rax,		movl		

%rbp	call malloc testq	%rbx,	20(%rbp), %eax	cltq imulq %rdx, %rax salq \$2, %rax movq %rax, %rdi call malloc movq %rax, -40(%rbp) movl -	movq %rbx, %rdi call malloc movq %rbx, %rdi movq %rax, %r14 call malloc movq %rbx, %rdi movq %rax, %r12 call malloc testq %r14, %r14 movq %rax, %r13 setne %dl testq %r12, %r12 setne %al orb %al, %dl jne .L15 testq %r13, %r13 je .L3
%rbx	movq	%rax,	20(%rbp), %eax	movslq %eax, %rdx movl -	movl (%rsp), %eax testl %eax, %eax jle .L31 movl (%rsp), %eax salq \$2, %rbp xorl %edi, %edi xorl %edx, %edx leal (%rax,%rax), %r8d
%rdi	movq	%rax,	20(%rbp), %eax	cltq imulq %rdx, %rax salq \$2, %rax movq %rax, %rdi call malloc movq %rax, -48(%rbp) cmpq \$0, -32(%rbp) je .L3 cmpq \$0, -40(%rbp) je .L3 cmpq \$0, -48(%rbp) jne .L4	leal 1(%rdx), %ebx leal (%rdx,%r8), %esi movq %rdi, %rax movl %ebx, %ecx .p2align 4,,10 .p2align 3
48(%rsp)	sete testq	%dl %rbp,	20(%rbp), %eax	movl \$-1, %edi call exit	movl \$0, -4(%rbp) jmp .L5
%rbp	sete orb	%al %al,	20(%rbp), %eax	movl \$0, -8(%rbp) jmp .L6	addl \$1,
%dl	jne testq	.L3 %rdi,	20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	addq \$4,
%rdi	je testl %r14d, %r14d jle movl 40(%rsp), %edi leaq 0(,	.L3 .L29	20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	cmpl %edx,
%r13,4), %rax	movq 48(%rsp), %r9 movl 56(%rsp), %r10d xorl %edx,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
%edx	movq	%rax,	20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
24(%rsp)	movq	%rax,	20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
%r11	leal (%rdi,%rdi), %r8d xorl %edi,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
%edi .L7:	leal 1(%rdx), %r12d leal (%rdx,%r8), %esi movq %rdi,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
%rax	movl %r12d, %ecx .p2align 4,,10 .p2align 3		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
.L6:	movl %edx,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
0(%rbp,%rax)	addl \$2,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
%edx	movl %ecx,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
(%rbx,%rax)	movl \$0,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
(%r9,%rax)	addl \$1,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
%ecx	addq \$4,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
%rax	cmpl %edx,		20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
%esi	jne addq	.L6 %r11,	20(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,
%rdi			8(%rbp), %eax	movl \$0, -4(%rbp) jmp .L5	jne .L6 addq \$4,

<div>%edi</div> <div>%r13</div> <div>%r12,4), %r10d</div> <div>(%rsp)</div> <div>%r11</div> <div>%eax</div> <div>%r10</div> <div>%r9</div> <div>60(%rsp)</div> <div>%eax</div> <div>8(%rsp)</div> <div>%rax,4), %rax</div> <div>16(%rsp)</div> <div>32(%rsp)</div> <div>.L12:</div> <div>%rax</div> <div>%ebx</div> <div>.L10:</div>	<pre> cmpl %r10d, %r12d movl %r12d, %edx jne .L7 leaq 64(%rsp), %rsi xorl %edi, movq %rbx, xorl %r14d, %r14d call clock_gettime movl 40(%rsp), %eax leal (%r12,%r12), %edx leal 0(, movq 48(%rsp), %r15 movslq %r12d, %rdi movl \$0, movslq %r10d, %r10 movslq %edx, addl %r12d, %edx subl \$1, salq \$2, movslq %edx, movl %eax, shr1 \$2, movq %rbx, leaq 4(, movq %rbp, movq %rax, movq 32(%rsp), %rax movq 8(%rsp), %rbx movq 16(%rsp), %rbp addq %r14, leaq (%rbx,%rax,4), %r8 xorl %ebx, .p2align 4,,10 .p2align 3 movl (%r15,%rbx,4), %eax </pre>	<pre> addl %edx, %eax cltq leaq 0(,%rax,4), %rdx movq - addq %rdx, %rax movl - movl - addl %ecx, %edx addl \$1, %edx movl %edx, (%rax) movl - imull - movl %eax, %edx movl - addl %edx, %eax cltq leaq 0(,%rax,4), %rdx movq - addq %rdx, %rax movl - leal (%rdx,%rdx), movl - addl %ecx, %edx movl %edx, (%rax) movl - imull - movl %eax, %edx movl - addl %edx, %eax cltq leaq 0(,%rax,4), %rdx movq - addq %rdx, %rax movl \$0, (%rax) addl \$1, -8(%rbp) </pre>	<pre> movl \$0, 0(%r13,%rax) addl \$1, %ecx addq \$4, %rax cmpl %edx, %esi jne .L7 addq %rbp, %rdi cmpl %r15d, %ebx movl %ebx, %edx jne .L8 leaq 16(%rsp), %rsi xorl %edi, %edi call clock_gettime movq %r13, %r10 movq %r14, %rdi xorl %r11d, %r11d .L13: movq %r12, %r9 xorl %r8d, %r8d .p2align 4,,10 .p2align 3 .L11: movl (%r10,%r8,4), %esi movq %r9, %rcx xorl %eax, %eax .p2align 4,,10 .p2align 3 .L10: movl (%rdi,%rax,4), %edx addq \$1, %rax imull (%rcx), %edx addq %rbp, %rcx addl %edx, %esi cmpl %eax, %ebx jg .L10 movl %esi, (%r10,%r8,4) addq \$1, %r8 addq </pre>
--	---	---	---

%rcx	movq	%rbp,	8(%rbp), %eax	movl	-	\$4, %r9
%rdx	movq	%r13,	20(%rbp), %eax	cmpl	-	cmpl
.L9:	.p2align 4,,10			jl		%r8d, %ebx
	.p2align 3			.L7		addl
	movl	(%rdx), %esi	.L5:	addl		\$1, %r11d
%rdx	addq	\$16,	4(%rbp), %eax	movl	-	addq
	imull	(%rcx), %esi	20(%rbp), %eax	cmpl	-	%rbp, %r10
%eax	addl	%esi,		jl		addq
12(%rdx), %esi	movl	-	80(%rbp), %rax	.L8		cmpl
	imull	(%rcx,%rdi,4), %esi		leaq	-	%r15d, %r11d
%esi	addl	%eax,				jne
8(%rdx), %eax	movl	-		.L14:		.L13
	imull	(%rcx,%r11,4), %eax		leaq		32(%rsp), %rsi
%eax	addl	%esi,		movq		xorl
4(%rdx), %esi	movl	-		%rax, %rsi		%edi, %edi
	imull	(%rcx,%r9,4), %esi		movl		call
%rcx	addq	%r10,		\$0, %edi		clock_gettime
%eax	addl	%esi,		call		movq
4(%rdx), %esi	movl	-	.L14:	clock_gettime		40(%rsp), %rax
	imull	(%rcx,%r9,4), %esi		movl		subq
%rcx	addq	%r10,		\$0, -4(%rbp)		24(%rsp), %rax
%eax	addl	%esi,		jmp		movl
4(%rdx), %esi	movl	-		.L9		\$.LC2, %edi
	imull	(%rcx,%r9,4), %esi				pxor
%rcx	addq	%r10,		movl		%xmm0, %xmm0
%eax	addl	%esi,	.L13:	\$0, -8(%rbp)		movl
%r8	cmpq	%rdx,		jmp		0(%r13), %esi
(%r15,%rbx,4)	jne	.L9		.L10		pxor
%rbx	movl	%eax,	.L12:	movl		%xmm1, %xmm1
%rbp	addq	\$1,	4(%rbp), %eax	\$0, -12(%rbp)		cvtsi2sdq
%r12d	addq	\$4,	20(%rbp), %eax	jmp		%rax, %xmm0
	cmpl	%ebx,		.L11		movq
(%rsp)	jg	.L10				32(%rsp), %rax
%r14	movq	24(%rsp), %rbx		movl	-	subq
	addl	\$1,		imull	-	16(%rsp), %rax
	addq	%rdi,		movl		cvtsi2sdq
%r15	addq	%rbx,		movl		%rax, %xmm1
%r13	cmpl	56(%rsp), %eax		%eax, %edx		movq
	jne	.L12		movl	-	(%rsp), %rax
	movq	8(%rsp), %rbx				leal
.L13:	movq	16(%rsp), %rbp				-
	leaq	80(%rsp), %rsi				addl
						\$1, %eax
						imull
						%edx, %eax
						movl
						%edx, %ecx
						divsd
						.LC1(%rip), %xmm0
						cltq
						movl
						0(%r13,%rax,4),
						%r8d
						xorl
						%eax, %eax
						addsd
						%xmm1, %xmm0
						movsd
						%xmm0, 8(%rsp)
						call
						printf
						movsd
						8(%rsp), %xmm0
						movl

%edi	xorl	%edi,	48(%rbp), %rax			\$.LC3, %edi
	call		addq			movl
	clock_gettime		%rcx, %rax			\$1, %eax
	movq		movl			call
	88(%rsp), %rax		(%rax), %ecx			printf
	subq		movl	-		movq
	72(%rsp), %rax	4(%rbp), %eax				%r14, %rdi
	pxor		imull	-		call
	%xmm0, %xmm0	20(%rbp), %eax				free
	movl		movl			movq
	60(%rsp), %edi		%eax, %esi			%r12, %rdi
	movq		movl	-		call
	48(%rsp), %r14	12(%rbp), %eax				free
	pxor		addl			movq
	%xmm1, %xmm1		%esi, %eax			%r13, %rdi
	cvtsi2sdq	%rax,	cltq			call
%xmm0	movq		leaq			free
	80(%rsp), %rax		0(,%rax,4), %rsi			addq
	subq		movq	-		\$56, %rsp
	64(%rsp), %rax	32(%rbp), %rax				.cfi_remember_sta
	movl		addq		te	
	(%r14), %esi		%rsi, %rax			.cfi_def_cfa_offs
	movl		movl		et 56	
%ecx	movl	%edi,	(%rax), %esi			xorl
			movl	-		%eax, %eax
%edx	movl	%edi,				popq
	cvtsi2sdq	%rax,	imull	-		%rbx
%xmm1	movl					.cfi_def_cfa_offs
	40(%rsp), %eax	20(%rbp), %eax	movl		et 48	popq
	addl		%eax, %edi			%rbp
%eax	imull		movl	-		.cfi_def_cfa_offs
	\$.LC3, %edi	8(%rbp), %eax			et 40	popq
	divsd		addl			%r12
	.LC2(%rip), %xmm0		%edi, %eax			.cfi_def_cfa_offs
	cltq		cltq			popq
	movl		leaq			%r13
	(%r14,%rax,4), %r8d	40(%rbp), %rax	0(,%rax,4), %rdi		et 32	.cfi_def_cfa_offs
	xorl		movq	-		popq
%eax	addsd				et 24	%r14
	%xmm1, %xmm0		addq			.cfi_def_cfa_offs
	movsd		%rdi, %rax			popq
	%xmm0, (%rsp)		movl			%r15
	call		(%rax), %eax			.cfi_def_cfa_offs
	printf		imull		et 16	popq
	movsd		%esi, %eax			%r15
	(%rsp), %xmm0		addl			.cfi_def_cfa_offs
	movl		%ecx, %eax			ret
	\$.LC4, %edi		movl		et 8	
	movl		%eax, (%rdx)			.cfi_restore_stat
%eax	call		movl	-	.L3:	
	printf		imull	-		movl
	movq				e	\$.LC4, %edi
	%rbx,		movl			call
%rdi	call		%eax, %edx			puts
	movq		movl	-		orl
	free					\$-
	%rbp,		addl		1, %edi	call
	call		%edx, %eax			exit
	movq		cltq			
%rdi	call		leaq		.L31:	leaq
	free		0(,%rax,4), %rdx			16(%rsp), %rsi
	%r14,		movq	-		xorl
	call					%edi, %edi
	addq		addq			call
	\$104,		%rax, %rdx			clock_gettime
			movl	-		jmp
			imull	-		

%rsp		20(%rbp), %eax		.L14	
	.cfi_remember_state	movl		.L30:	
	.cfi_def_cfa_offset	%eax, %ecx		movl	
56		movl	-	\$.LC0, %edi	
%eax	xorl %eax,	8(%rbp), %eax		call	
	popq %rbx	addl		puts	
	.cfi_def_cfa_offset	%ecx, %eax		orl	\$-
48		cltq		1, %edi	
	popq %rbp	leaq		call	
	.cfi_def_cfa_offset	0(,%rax,4), %rcx		exit	
40		movq	-	.cfi_endproc	
	popq %r12	48(%rbp), %rax		.LFE21:	
	.cfi_def_cfa_offset	addq		.size	
32		%rcx, %rax		main, -.main	
	popq %r13	movl		.section	.r
	.cfi_def_cfa_offset	(%rax), %ecx		odata.cst8,"aM",@progbits,8	
24		movl	-	.align 8	
	popq %r14	4(%rbp), %eax		.LC1:	
	.cfi_def_cfa_offset	imull	-	.long	0
16		20(%rbp), %eax		.long	
	popq %r15	movl	-	1104006501	
	.cfi_def_cfa_offset	12(%rbp), %esi		.ident	"G
8		addl		CC: (GNU) 6.1.1 20160501"	
	ret	\$1, %esi		.section	.n
		addl		ote.GNU-stack,"",@progbits	
.L29:	.cfi_restore_state	%esi, %eax			
	leaq	cltq			
	64(%rsp), %rsi	leaq			
%edi	xorl %edi,	0(,%rax,4), %rsi			
	call	movq	-		
	clock_gettime	32(%rbp), %rax			
	movq	addq			
	40(%rsp), %rax	%rsi, %rax			
%eax	subl \$1,	movl			
	movl %eax,	(%rax), %esi			
60(%rsp)	jmp .L13	movl	-		
.L3:		addl			
	movl	\$1, %eax			
	\$.LC1, %edi	imull	-		
	call puts	20(%rbp), %eax			
%edi	orl \$-1,	movl			
	call exit	%eax, %edi			
.L28:		movl	-		
	movl	8(%rbp), %eax			
	\$.LC0, %edi	addl			
	call puts	%edi, %eax			
%edi	orl \$-1,	cltq			
	call exit	leaq			
.LFE21:	.cfi_endproc	0(,%rax,4), %rdi			
		movq	-		
	.size main,	40(%rbp), %rax			
.-main		addq			
	.section .roda	%rdi, %rax			
	.align 8	movl			
.LC2:	.long 0	(%rax), %eax			
	.long	imull	-		
	1104006501	%esi, %eax			
	.ident "GCC:	addl			
(GNU) 6.1.1 20160501"		%ecx, %eax			
	.section .note	movl			
.GNU-stack,"",@progbits		%eax, (%rdx)			
		movl	-		
		4(%rbp), %eax			
		imull	-		
		20(%rbp), %eax			
		movl			
		%eax, %edx			
		movl	-		

8(%rbp), %eax	addl %edx, %eax cltq leaq 0(,%rax,4), %rdx movq -
48(%rbp), %rax	addq %rax, %rdx movl -
4(%rbp), %eax	imull -
20(%rbp), %eax	movl %eax, %ecx movl -
8(%rbp), %eax	addl %ecx, %eax cltq leaq 0(,%rax,4), %rcx movq -
48(%rbp), %rax	addq %rcx, %rax movl (%rax), %ecx movl -
4(%rbp), %eax	imull -
20(%rbp), %eax	movl -
12(%rbp), %esi	addl \$2, %esi addl %esi, %eax cltq leaq 0(,%rax,4), %rsi movq -
32(%rbp), %rax	addq %rsi, %rax movl (%rax), %esi movl -
12(%rbp), %eax	addl \$2, %eax imull -
20(%rbp), %eax	movl %eax, %edi movl -
8(%rbp), %eax	addl %edi, %eax cltq leaq 0(,%rax,4), %rdi movq -
40(%rbp), %rax	addq %rdi, %rax movl (%rax), %eax

	imull	
	%esi, %eax	
	addl	
	%ecx, %eax	
	movl	
	%eax, (%rdx)	
	movl	-
4(%rbp), %eax		
	imull	-
20(%rbp), %eax		
	movl	
	%eax, %edx	
	movl	-
8(%rbp), %eax		
	addl	
	%edx, %eax	
	cltq	
	leaq	
	0(,%rax,4), %rdx	
	movq	-
48(%rbp), %rax		
	addq	
	%rax, %rdx	
	movl	-
4(%rbp), %eax		
	imull	-
20(%rbp), %eax		
	movl	
	%eax, %ecx	
	movl	-
8(%rbp), %eax		
	addl	
	%ecx, %eax	
	cltq	
	leaq	
	0(,%rax,4), %rcx	
	movq	-
48(%rbp), %rax		
	addq	
	%rcx, %rax	
	movl	
	(%rax), %ecx	
	movl	-
4(%rbp), %eax		
	imull	-
20(%rbp), %eax		
	movl	-
12(%rbp), %esi		
	addl	
	\$3, %esi	
	addl	
	%esi, %eax	
	cltq	
	leaq	
	0(,%rax,4), %rsi	
	movq	-
32(%rbp), %rax		
	addq	
	%rsi, %rax	
	movl	
	(%rax), %esi	
	movl	-
12(%rbp), %eax		
	addl	
	\$3, %eax	
	imull	-
20(%rbp), %eax		
	movl	
	%eax, %edi	

	movl	-	
8(%rbp), %eax	addl		
	%edi, %eax		
	cltq		
	leaq		
	0(%rax,4), %rdi		
	movq	-	
40(%rbp), %rax	addq		
	%rdi, %rax		
	movl		
	(%rax), %eax		
	imull		
	%esi, %eax		
	addl		
	%ecx, %eax		
	movl		
	%eax, (%rdx)		
	addl		
	\$4, -12(%rbp)		
.L11:	movl	-	
12(%rbp), %eax	cmpl	-	
20(%rbp), %eax	j1		
	.L12		
	addl		
	\$1, -8(%rbp)		
.L10:	movl	-	
8(%rbp), %eax	cmpl	-	
20(%rbp), %eax	j1		
	.L13		
	addl		
	\$1, -4(%rbp)		
.L9:	movl	-	
4(%rbp), %eax	cmpl	-	
20(%rbp), %eax	j1		
	.L14		
	leaq	-	
96(%rbp), %rax	movq		
	%rax, %rsi		
	movl		
	\$0, %edi		
	call		
	clock_gettime		
	movq	-	
96(%rbp), %rdx	movq	-	
80(%rbp), %rax	subq		
	%rax, %rdx		
	movq		
	%rdx, %rax		
	pxor		
	%xmm1, %xmm1		
	cvtsi2sdq		
	%rax, %xmm1		
	movq	-	
88(%rbp), %rdx	movq	-	

	<pre> 72(%rbp), %rax subq %rax, %rdx movq %rdx, %rax pxor %xmm0, %xmm0 cvtsi2sdq %rax, %xmm0 movsd .LC2(%rip), %xmm2 divsd %xmm2, %xmm0 addsd %xmm1, %xmm0 movsd %xmm0, -56(%rbp) movl 20(%rbp), %eax leal 1(%rax), %edx movl 20(%rbp), %eax subl \$1, %eax imull %edx, %eax cltq leaq 0(,%rax,4), %rdx movq 48(%rbp), %rax addq %rdx, %rax movl (%rax), %esi movl 20(%rbp), %eax leal 1(%rax), %ecx movl 20(%rbp), %eax leal 1(%rax), %edx movq 48(%rbp), %rax movl (%rax), %eax movl %esi, %r8d movl %eax, %esi movl \$.LC3, %edi movl \$0, %eax call printf movq 56(%rbp), %rax movq %rax, -120(%rbp) movsd 120(%rbp), %xmm0 movl \$.LC4, %edi movl \$1, %eax </pre>	
--	---	--

	call printf movq 32(%rbp), %rax	-
	movq %rax, %rdi call free movq 40(%rbp), %rax	-
	movq %rax, %rdi call free movq 48(%rbp), %rax	-
8	movq %rax, %rdi call free movl \$0, %eax leave .cfi_def_cfa 7,	
	ret .cfi_endproc	
.LFE2:	.size main, .-main .section	.
rodata	.align 8	
.LC2:	.long .long 1104006501 .ident	0 "
GCC: (GNU) 6.1.1 20160501"	.section	.
	note.GNU-stack,"",@progbits	

B) CÓDIGO FIGURA 1:**CÓDIGO FUENTE:** figura1-original.c**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

```

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

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

main(){
    int i, ii;
    int X1, X2, *R;
    double t1, t2, tiempo;
    R = (int*)malloc(40000*sizeof(int));

    // Inicialización R,S
    for (i=0 ; i<40000 ; i++)
        R[i] = 0;
    for (i=0 ; i<5000 ; i++){

```

```

        s[i].a = 0;
        s[i].b = 0;
    }
    t1 = omp_get_wtime();
    // Cálculo del algoritmo
    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;
    }
    t2 = omp_get_wtime();
    tiempo = t2-t1;
    printf("R[0] = %d\t", R[0]);
    printf("R[39999] = %d", R[39999]);
    printf("\nTiempo de proceso: %f segundos.\n", tiempo);
    free(R);
}

```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación–: El fichero original utiliza dos bucles para calcular independiente X1 y X2, en la primera modificación reduzco uno de los for pero dejo el mismo número de iteraciones.

Modificación b) –explicación–: La segunda modificación es dividir los calculos en dos for para hacer menos operaciones. Por un lado a y por otro b.

Modificación c) –explicación–: La última es reorganizar, he generado dos for nuevos pese a haberlo eliminado en el apartado a. De esta manera me ahorro iteraciones y realizo más calculos por iteración.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figura1-modificado_a.c

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```

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

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

main(){
    int i, ii;
    int X1, X2, *R;
    double t1, t2, tiempo;
    R = (int*)malloc(40000*sizeof(int));

    // Inicialización R,S
    for (i=0 ; i<40000 ; i++)
        R[i] = 0;
    for (i=0 ; i<5000 ; i++){
        s[i].a = 0;
        s[i].b = 0;
    }
    t1 = omp_get_wtime();
    // Cálculo del algoritmo
    for (ii=0 ; ii<40000 ; ii++){
        X1=0; X2=0;

```

```

        for (i=0 ; i<5000 ; i++){
            X1 += 2*s[i].a + ii;
            X2 += 3*s[i].b - ii;
        }
        if (X1<X2)
            R[ii] = X1;
        else R[ii] = X2;
    }
    t2 = omp_get_wtime();
    tiempo = t2-t1;
    printf("R[0] = %d\t", R[0]);
    printf("R[39999] = %d", R[39999]);
    printf("\nTiempo de proceso: %f segundos.\n", tiempo);
    free(R);
}

```

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

```

Codigo : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[mike@looper Codigo]$ ./figura1
R[0] = 0          R[39999] = -199995000
Tiempo de proceso: 0.251584 segundos.
[mike@looper Codigo]$ ./figura1_a
R[0] = 0          R[39999] = -199995000
Tiempo de proceso: 0.190677 segundos.
[mike@looper Codigo]$ ./figura1_b
R[0] = 0          R[39999] = -199995000
Tiempo de proceso: 0.190426 segundos.
[mike@looper Codigo]$ ./figura1_c
R[0] = 0          R[39999] = -199995000
Tiempo de proceso: 0.153190 segundos.
[mike@looper Codigo]$

```

b) figura1-modificado_b.c

```

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

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

main(){
    int i, ii;
    int X1, X2, *R;
    double t1, t2, tiempo;
    R = (int*)malloc(40000*sizeof(int));

    // Inicialización R,S
    for (i=0 ; i<40000 ; i++)
        R[i] = 0;
    for (i=0 ; i<5000 ; i++){
        s[i].a = 0;
        s[i].b = 0;
    }
}

```

```

    }
    t1 = omp_get_wtime();
    // Cálculo del algoritmo
    for (ii=0 ; ii<40000 ; ii++){
        X1=0; X2=0;
        for (i=0 ; i<5000 ; i+=2){
            X1 += 2*s[i].a + ii;
            X1 += 2*s[i+1].a + ii;
        }
        for (i=0 ; i<5000 ; i+=2){
            X2 += 3*s[i].b - ii;
            X2 += 3*s[i+1].b - ii;
        }
        if (X1<X2)
            R[ii] = X1;
        else R[ii] = X2;
    }
    t2 = omp_get_wtime();
    tiempo = t2-t1;
    printf("R[0] = %d\t", R[0]);
    printf("R[39999] = %d", R[39999]);
    printf("\nTiempo de proceso: %f segundos.\n", tiempo);
    free(R);
}

```

c) figura1-modificado_c.c

```

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

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

main(){
    int i, ii;
    int X1, X2, *R;
    double t1, t2, tiempo;
    R = (int*)malloc(40000*sizeof(int));

    // Inicialización R,S
    for (i=0 ; i<40000 ; i++)
        R[i] = 0;
    for (i=0 ; i<5000 ; i++){
        s[i].a = 0;
        s[i].b = 0;
    }
    t1 = omp_get_wtime();
    // Cálculo del algoritmo
    for (ii=0 ; ii<40000 ; ii++){
        X1=0; X2=0;
        for (i=0 ; i<5000 ; i+=2){
            X1 += 2*s[i].a + ii;
            X1 += 2*s[i+1].a + ii;
            X2 += 3*s[i].b - ii;
            X2 += 3*s[i+1].b - ii;
        }
        if (X1<X2)
            R[ii] = X1;
        else R[ii] = X2;
    }

    t2 = omp_get_wtime();
    tiempo = t2-t1;
    printf("R[0] = %d\t", R[0]);
    printf("R[39999] = %d", R[39999]);
}

```

```
printf("\nTiempo de proceso: %f segundos.\n", tiempo);
free(R);
}
```

1.1. TIEMPOS:

Modificación	-O2
Sin modificar	0,251584
Modificación a)	0.190677
Modificación b)	0.190426
Modificación c)	0.153190

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Como vemos en los resultados las optimizaciones han resultado efectivas.

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_b.s	pmm-secuencial-modificado_c.s
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */	/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */	/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */

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<=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 acarrearán. 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)**

```

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    struct timespec cgt1, cgt2;
    int N, a=3, i;
    int *y, *x;

    if(argc < 2){
        printf("Introduzca El tamaño N:\n");
        exit(-1);
    }

    N=atoi(argv[1]);

    y = malloc(sizeof(int) * N+1);
    x = malloc(sizeof(int) * N+1);

    if(y == NULL || x == NULL){
        printf("No se ha podido reservar memoria\n");
        exit(-1);
    }

    // inicializacion
    for(i=1; i<=N; ++i){
        x[i]=i+1;
        y[i]=i+2;
    }

    // daxpy
    clock_gettime(CLOCK_REALTIME,&cgt1);
    for (i=1; i<=N; i++)
        y[i]+= a*x[i];
    clock_gettime(CLOCK_REALTIME,&cgt2);
    double ncgt=(double)(cgt2.tv_sec-cgt1.tv_sec) + (double)
((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    printf("Tiempo transcurrido en segundos: %0.4f\n", ncgt);

    return 0;
}

```

Tiempos ejec.	-O0	-O2	-O3
	0,0014	0,0012	0,0003

CAPTURAS DE PANTALLA:

```
Codigo : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[mike@looper Codigo]$ ./daxpy00 500000
Tiempo transcurrido en segundos: 0.0014
[mike@looper Codigo]$ ./daxpy02 500000
Tiempo transcurrido en segundos: 0.0012
[mike@looper Codigo]$ ./daxpy03 500000
Tiempo transcurrido en segundos: 0.0003
[mike@looper Codigo]$
```

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR: Ninguno.

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	daxpy02.s	daxpy03.s
<pre>.file "daxpy.c" .section .rodata .LC0: .string "Introduzca El tama\303\261o N:" .align 8 .LC1: .string "No se ha podido reservar memoria" .align 8 .LC3: .string "Tiempo transcurrido en segundos: %0.4f\n" .text .globl main .type main, @function main: .LFB21: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp</pre>	<pre>.file "daxpy.c" .section .rodata .a.str1.1, "aMS", @progbits, 1 .LC0: .string "Intro duzca El tama\303\261o N:" .section .rodata .a.str1.8, "aMS", @progbits, 1 .align 8 .LC1: .string "No se ha podido reservar memoria" .align 8 .LC3: .string "Tiempo o transcurrido en segundos: %0.4f\n" .section .text startup, "ax", @progbits .p2align 4,,15 .globl main .type main, @function main: .LFB21: .cfi_startproc pushq %r13 .cfi_def_cfa_offset 16 .cfi_offset 13, -16 pushq %r12 .cfi_def_cfa_offset</pre>	<pre>.file "daxpy.c" .section .rodata odata.str1.1, "aMS", @progbits, 1 .LC0: .string "Introduzca El tama\303\261o N:" .section .rodata odata.str1.8, "aMS", @progbits, 1 .align 8 .LC1: .string "No se ha podido reservar memoria" .align 8 .LC7: .string "Tiempo transcurrido en segundos: %0.4f\n" .section .text ext.startup, "ax", @progbits .p2align 4,,15 .globl main .type main, @function main: .LFB21: .cfi_startproc pushq %r12 .cfi_def_cfa_offset</pre>

ster 6	.cfi_def_cfa_regi	24	.cfi_offset 12, -24	-16	.cfi_offset 12,
	subq		pushq %rbp		pushq
	\$112, %rsp		.cfi_def_cfa_offset		%rbp
	movl	32	.cfi_offset 6, -32	et 24	.cfi_def_cfa_offs
	%edi, -84(%rbp)		pushq %rbx		.cfi_offset 6,
	movq	40	.cfi_def_cfa_offset	-24	pushq
	%rsi, -96(%rbp)		.cfi_offset 3, -40		%rbx
	movl	%rsp	subq \$40,	et 32	.cfi_def_cfa_offs
	\$3, -8(%rbp)		.cfi_def_cfa_offset	-32	.cfi_offset 3,
	cmpl	80	cmpl \$1,		subq
	\$1, -84(%rbp)	%edi	jle .L20	et 80	\$48, %rsp
	jg		movq 8(%rsi), %rdi		.cfi_def_cfa_offs
	.L2		movl \$10,		cmpl
	movl	%edx	xorl %esi,		\$1, %edi
	\$.LC0, %edi	%esi	call strtol		jle
	call		movq %rax,		.L46
	puts		movl %eax,		movq
	movl \$-	%r13	cltq		8(%rsi), %rdi
1, %edi	call	%r12d	leaq 1(,		movl
.L2:	exit	%rax,4), %rbp	movq %rbp,		\$10, %edx
	movq -	%rdi	call malloc		xorl
96(%rbp), %rax	addq \$8, %rax	%r13d	movq %rbp,		%esi, %esi
	movq (%rax), %rax		call malloc		call
	(%rax), %rax		testq %rbx,		strtol
	movq %rax, %rdi		movq %rax,		movq
	call atoi		je .L3		%rax, %rbx
	movl %eax, -12(%rbp)		testq %rax,		cltq
12(%rbp), %eax	movl -	%rdi	je .L3		leaq
	cltq	%rbx	testl %r13d,		1(,%rax,4), %r12
	salq \$2, %rax	%rbp	leal 3(%r13), %ecx		movq
	addq \$1, %rax	%rax	movl \$3,		%r12, %rdi
	movq %rax, %rdi	%r13d	movl \$2,		call
	call malloc		jle .L21		malloc
	movq %rax, -24(%rbp)		.p2align 4,10		movq
12(%rbp), %eax	movl -		.p2align 3		%r12, %rdi
	cltq		movl %edx,		call
	salq \$2, %rax	%edx	addl \$1,		malloc
	addq \$1, %rax	%eax	movl %eax,		movq
	movq %rax, %rdi	-4(%rbp,%rax,4)	addq \$1,		%r12, %rdi
	call malloc	%rax	cmpl %edx,		%rax, %rbp
	movq %rax, -32(%rbp)	%ecx			call
	cmpq \$0, -24(%rbp)				malloc
	je .L3				testq
	cmpq \$0, -32(%rbp)				%rbp, %rbp
	jne .L4				movq
					%rax, %r12
					je
					.L3
					testq
					%rax, %rax
					je
					.L3
					testl
					%ebx, %ebx
					jle
					.L47
					leaq
					4(%rax), %rax
					shrq
					\$2, %rax
					negq
					%rax
					andl
					\$3, %eax
					cmpl
					%ebx, %eax

<pre> .L3: movl \$.LC1, %edi call puts movl \$-1, %edi call exit .L4: movl \$1, -4(%rbp) jmp .L5 .L6: movl 4(%rbp), %eax cltq leaq 0(%rax,4), %rdx movq 32(%rbp), %rax addq %rdx, %rax movl 4(%rbp), %edx addl \$1, %edx movl %edx, (%rax) movl 4(%rbp), %eax cltq leaq 0(%rax,4), %rdx movq 24(%rbp), %rax addq %rdx, %rax movl 4(%rbp), %edx addl \$2, %edx movl %edx, (%rax) addl \$1, -4(%rbp) .L5: movl 4(%rbp), %eax cmpl 12(%rbp), %eax jle .L6 leaq 64(%rbp), %rax movq %rax, %rsi movl \$0, %edi call clock_gettime movl \$1, -4(%rbp) jmp .L7 .L8: movl 4(%rbp), %eax </pre>	<pre> jne .L10 movq %rsp, %rsi xorl %edi, %edi call clock_gettime movl \$1, %eax .p2align 4,,10 .p2align 3 .L8: movl 0(%rbp,%rax,4), %edx leal (%rdx,%rdx,2), %edx addl %edx, (%rbx,%rax,4) addq \$1, %rax cmpl %eax, %r12d jge .L8 .L9: leaq 16(%rsp), %rsi xorl %edi, %edi call clock_gettime movq 24(%rsp), %rax subq 8(%rsp), %rax movl \$.LC3, %edi pxor %xmm0, %xmm0 pxor %xmm1, %xmm1 cvtsi2sdq %rax, %xmm0 movq 16(%rsp), %rax subq (%rsp), %rax cvtsi2sdq %rax, %xmm1 movl \$1, %eax divsd .LC2(%rip), %xmm0 addsd %xmm1, %xmm0 call printf addq \$40, %rsp .cfi_remember_state .cfi_def_cfa_offset 40 xorl %eax, %eax popq %rbx .cfi_def_cfa_offset 32 popq %rbp .cfi_def_cfa_offset 24 popq %r12 </pre>	<pre> cmova %ebx, %eax cmpl \$6, %ebx cmovbe %ebx, %eax testl %eax, %eax je .L24 cmpl \$1, %eax movl \$2, 4(%r12) movl \$3, 4(%rbp) je .L25 cmpl \$2, %eax movl \$3, 8(%r12) movl \$4, 8(%rbp) je .L26 cmpl \$3, %eax movl \$4, 12(%r12) movl \$5, 12(%rbp) je .L27 cmpl \$4, %eax movl \$5, 16(%r12) movl \$6, 16(%rbp) je .L28 cmpl \$6, %eax movl \$6, 20(%r12) movl \$7, 20(%rbp) jne .L29 movl \$7, 24(%r12) movl \$8, 24(%rbp) movl \$7, %ecx .L8: cmpl %eax, %ebx je .L48 .L7: movl %ebx, %r8d leal -1(%rbx), %esi movl %eax, %edi subl </pre>
--	---	---

	cltq		.cfi_def_cfa_offset		%eax, %r8d
	leaq	16			leal -
	0(,%rax,4), %rdx		popq %r13	4(%r8), %edx	
24(%rbp), %rax	movq -		.cfi_def_cfa_offset 8		subl
	addq	.L21:	ret		%eax, %esi
4(%rbp), %eax	%rax, %rdx		.cfi_restore_state		shrl
	movl -	%rsi	movq %rsp,		\$2, %edx
	cltq	%edi	xorl %edi,		addl
	leaq				\$1, %edx
	0(,%rax,4), %rcx		call		cmpl
24(%rbp), %rax	movq -		clock_gettime		\$2, %esi
	addq	.L20:	jmp .L9		leal
	%rcx, %rax				0(,%rdx,4), %r9d
	movl	%edi	movl \$.LC0,		jbe
4(%rbp), %eax	(%rax), %ecx		call puts		.L10
	movl -	%edi	orl \$-1,		movl
	cltq	%edi	call exit		%ecx, 12(%rsp)
	leaq	.L3:	movl \$.LC1,		leaq
	0(,%rax,4), %rsi	%edi	call puts		4(,%rdi,4), %rdi
32(%rbp), %rax	movq -	%edi	orl \$-1,		xorl
	addq	%edi	call exit		%eax, %eax
	%rsi, %rax		.cfi_endproc		movd
	movl	.LFE21:			12(%rsp), %xmm6
8(%rbp), %eax	(%rax), %eax		.size		xorl
	imull -		main, .-main		%esi, %esi
	addl		.section	.rodat	movdqa
	%ecx, %eax	a.cst8,"aM",@progbits,8	.align 8		.LC3(%rip), %xmm4
	addl	.LC2:	.long 0		leaq
.L7:	\$1, -4(%rbp)		.long 1104006501		(%r12,%rdi), %r10
	movl -		.ident	"GCC: .L11:	addq
4(%rbp), %eax	cmpl -	(GNU) 6.1.1 20160501"	.section	.note.	%rbp, %rdi
12(%rbp), %eax	jle	GNU-stack,"",@progbits			pshufd
	.L8				\$0, %xmm6, %xmm0
80(%rbp), %rax	leaq -				movdqa
	movq				.LC4(%rip), %xmm3
	%rax, %rsi				movdqa
	movl				.LC5(%rip), %xmm2
	\$0, %edi				padd
	call				.LC2(%rip), %xmm0
	clock_gettime				movdqa
80(%rbp), %rdx	movq -				%xmm0, %xmm1
	movq -				movdqa
64(%rbp), %rax	subq				%xmm0, %xmm5
	%rax, %rdx				movaps
	movq				%xmm1,
	%rdx, %rax			(%r10,%rax)	addq
	pxor				\$16, %rax
	%xmm1, %xmm1				cmpl
	cvtsi2sdq				%esi, %edx
	%rax, %xmm1				jbe
72(%rbp), %rdx	movq -				.L49
	movq -				movdqa
56(%rbp), %rax	subq				%xmm5, %xmm0
					jmp
				.L49:	.L11

<pre> %rax, %rdx movq %rdx, %rax pxor %xmm0, %xmm0 cvtsi2sdq %rax, %xmm0 movsd .LC2(%rip), %xmm2 divsd %xmm2, %xmm0 addsd %xmm1, %xmm0 movsd %xmm0, -40(%rbp) movq 40(%rbp), %rax movq %rax, -104(%rbp) movsd 104(%rbp), %xmm0 movl \$.LC3, %edi movl \$1, %eax call printf movl \$0, %eax leave .cfi_def_cfa 7, 8 ret .cfi_endproc .LFE2: .size main, .-main .section odata .align 8 .LC2: .long 0 .long 1104006501 .ident "G CC: (GNU) 6.1.1 20160501" .section ote.GNU-stack,"",@progbits </pre>		<pre> addl %r9d, %ecx cmpl %r9d, %r8d je .L13 .L10: leal 1(%rcx), %edx leal 2(%rcx), %eax movslq %ecx, %rsi cmpl %edx, %ebx movl %edx, (%r12,%rsi,4) movl %eax, 0(%rbp, %rsi,4) j1 .L13 leal 3(%rcx), %esi movslq %edx, %rdx cmpl %eax, %ebx movl %eax, (%r12,%rdx,4) movl %esi, 0(%rbp, %rdx,4) j1 .L13 cltq addl \$4, %ecx movl %esi, (%r12,%rax,4) movl %ecx, 0(%rbp, %rax,4) .L13: leaq 16(%rsp), %rsi xorl %edi, %edi call clock_gettime leaq 4(%rbp), %rdx movq %rdx, %rax shrq \$2, %rax negq %rax andl \$3, %eax cmpl %ebx, %eax cmova %ebx, %eax cmpl \$4, %ebx </pre>
---	--	---

		<pre> cmovbe %ebx, %eax .L17: testl %eax, %eax je .L50 .L23: imull \$3, 4(%r12), %ecx addl %ecx, (%rdx) cmpl \$1, %eax je .L32 imull \$3, 8(%r12), %edx addl %edx, 8(%rbp) cmpl \$2, %eax je .L33 imull \$3, 12(%r12), %edx addl %edx, 12(%rbp) cmpl \$4, %eax jne .L34 imull \$3, 16(%r12), %edx movl \$5, %ecx addl %edx, 16(%rbp) .L19: cmpl %eax, %ebx je .L22 .L18: movl %ebx, %r8d leal 1(%rbx), %esi movl %eax, %edi subl %eax, %r8d leal 4(%r8), %edx subl %eax, %esi shr1 \$2, %edx addl \$1, %edx cmpl \$2, %esi leal 0(,%rdx,4), %r10d jbe .L21 leaq </pre>
--	--	--

		<pre> 4(,%rdi,4), %rdi xorl %eax, %eax xorl %esi, %esi leaq 0(%rbp,%rdi), %r9 addq %r12, %rdi .L15: movdqu (%rdi,%rax), %xmm1 addl \$1, %esi movdqa %xmm1, %xmm0 pslld \$1, %xmm0 padd %xmm1, %xmm0 padd (%r9,%rax), %xmm0 movaps %xmm0, (%r9,%rax) addq \$16, %rax cmpl %edx, %esi jb .L15 addl %r10d, %ecx cmpl %r10d, %r8d je .L22 .L21: movslq %ecx, %rax addl \$1, %ecx imull \$3, (%r12,%rax,4), %edx addl %edx, 0(%rbp, %rax,4) cmpl %ecx, %ebx jl .L22 movslq %ecx, %rcx imull \$3, (%r12,%rcx,4), %eax addl %eax, 0(%rbp, %rcx,4) .L22: leaq 32(%rsp), %rsi xorl %edi, %edi call clock_gettime movq 40(%rsp), %rax </pre>
--	--	---

		<pre> pxor %xmm0, %xmm0 subq 24(%rsp), %rax pxor %xmm1, %xmm1 movl \$.LC7, %edi cvtsi2sdq %rax, %xmm0 movq 32(%rsp), %rax subq 16(%rsp), %rax cvtsi2sdq %rax, %xmm1 movl \$1, %eax divsd .LC6(%rip), %xmm0 addsd %xmm1, %xmm0 call printf addq \$48, %rsp .cfi_remember_state te .cfi_def_cfa_offset 32 xorl %eax, %eax popq %rbx .cfi_def_cfa_offset 24 popq %rbp .cfi_def_cfa_offset 16 popq %r12 .cfi_def_cfa_offset 8 ret .L50: .cfi_restore_state movl \$1, %ecx jmp .L18 .L48: leaq 16(%rsp), %rsi xorl %edi, %edi call clock_gettime leaq 4(%rbp), %rdx movq %rdx, %rax shrq \$2, %rax negq %rax andl \$3, %eax </pre>
--	--	---

		<pre> cmpl %ebx, %eax cmova %ebx, %eax cmpl \$4, %ebx ja .L17 movl %ebx, %eax jmp .L23 .L24: movl \$1, %ecx jmp .L7 .L33: movl \$3, %ecx jmp .L19 .L34: movl \$4, %ecx jmp .L19 .L32: movl \$2, %ecx jmp .L19 .L28: movl \$5, %ecx jmp .L8 .L29: movl \$6, %ecx jmp .L8 .L25: movl \$2, %ecx jmp .L8 .L26: movl \$3, %ecx jmp .L8 .L27: movl \$4, %ecx jmp .L8 .L47: leaq 16(%rsp), %rsi xorl %edi, %edi call clock_gettime jmp .L22 .L46: movl \$.LC0, %edi </pre>
--	--	---

		<pre> call puts orl \$- 1, %edi call exit .L3: movl \$.LC1, %edi call puts orl \$- 1, %edi call exit .cfi_endproc .LFE21: .size main, .-main .section odata.cst16,"aM",@progbits,16 .align 16 .LC2: .long 0 .long 1 .long 2 .long 3 .align 16 .LC3: .long 4 .long 4 .long 4 .long 4 .align 16 .LC4: .long 1 .long 1 .long 1 .long 1 .align 16 .LC5: .long 2 .long 2 .long 2 .long 2 .section odata.cst8,"aM",@progbits,8 .align 8 .LC6: .long 0 .long 1104006501 .ident CC: (GNU) 6.1.1 20160501" .section ote.GNU-stack,"",@progbits </pre>	<pre> \$- \$- .r .r 0 1 2 3 4 4 4 4 1 1 1 1 2 2 2 2 .r 0 "G .n </pre>
--	--	--	---