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): Daniel Díaz Pareja
Grupo de prácticas: A1
Fecha de entrega: 03/06/2016
Fecha evaluación en clase: 03/06/2016

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i3-2350M CPU @ 2.30GHz*

Sistema operativo utilizado: *Ubuntu 14.04*

Versión de gcc utilizada: 4.8

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

processor : 0

vendor_id : GenuineIntel

cpu family : 6

model : 42

model name : Intel(R) Core(TM) i3-2350M CPU @ 2.30GHz

stepping : 7

microcode : 0x1b

cpu MHz : 884.421

cache size : 3072 KB

physical id : 0

siblings : 4

core id : 0

cpu cores : 2

apicid : 0

initial apicid : 0

fpu : yes

fpu_exception : yes

cpuid level : 13

wp : yes

flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer xsave avx lahf_lm arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid xsaveopt

bugs :
bogomips : 4589.42
clflush size : 64
cache_alignment : 64
address sizes : 36 bits physical, 48 bits virtual
power management:

processor : 1
vendor_id : GenuineIntel
cpu family : 6
model : 42
model name : Intel(R) Core(TM) i3-2350M CPU @ 2.30GHz
stepping : 7
microcode : 0x1b
cpu MHz : 812.097
cache size : 3072 KB
physical id : 0
siblings : 4
core id : 0
cpu cores : 2
apicid : 1
initial apicid : 1
fpu : yes
fpu_exception : yes
cpuid level : 13
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer xsave avx lahf_lm arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid xsaveopt
bugs :
bogomips : 4589.42
clflush size : 64
cache_alignment : 64
address sizes : 36 bits physical, 48 bits virtual
power management:

processor : 2
vendor_id : GenuineIntel
cpu family : 6
model : 42
model name : Intel(R) Core(TM) i3-2350M CPU @ 2.30GHz
stepping : 7
microcode : 0x1b
cpu MHz : 805.718
cache size : 3072 KB
physical id : 0
siblings : 4
core id : 1
cpu cores : 2
apicid : 2
initial apicid : 2
fpu : yes
fpu_exception : yes
cpuid level : 13
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer xsave avx lahf_lm arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid xsaveopt
bugs :
bogomips : 4589.42
clflush size : 64
cache_alignment : 64
address sizes : 36 bits physical, 48 bits virtual
power management:

processor : 3
vendor_id : GenuineIntel
cpu family : 6
model : 42
model name : Intel(R) Core(TM) i3-2350M CPU @ 2.30GHz
stepping : 7
microcode : 0x1b

cpu MHz : 843.812

cache size : 3072 KB

physical id : 0

siblings : 4

core id : 1

cpu cores : 2

apicid : 3

initial apicid : 3

fpu : yes

fpu_exception : yes

cpuid level : 13

wp : yes

flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer xsave avx lahf_lm arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept vpid xsaveopt

bugs :

bogomips : 4589.42

clflush size : 64

cache_alignment : 64

address sizes : 36 bits physical, 48 bits virtual

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)

```
// Compilar con -O2 y -fopenmp
```

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

int main(int argc, char** argv){
		int i, j, f, c;
		double t1, t2, total;
		srand(time(NULL));

		//Leer argumento de entrada (no de componentes del vector)
		if (argc<2){
				printf("Falta tamaño de matriz y vector\n");
				exit(-1);
		}

		unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

		double *v1, *v2, **M;
		v1 = (double*) malloc(N*sizeof(double));// malloc necesita el
tamaño en bytes
		v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio
suficiente malloc devuelve NULL
		M = (double**) malloc(N*sizeof(double *));
		if ( (v1==NULL) || (v2==NULL) || (M==NULL) ){
				printf("Error en la reserva de espacio para los
vectores\n");
				exit(-2);
		}

		for (i=0; i<N; i++){
				M[i] = (double*) malloc(N*sizeof(double));
				if ( M[i]==NULL ){
						printf("Error en la reserva de espacio
para los vectores\n");
						exit(-2);
				}
		}
		//A partir de aqui se pueden acceder las componentes de la
matriz como M[i][j]

		//Inicializar matriz y vectores
		//printf("Vector 1: \n\n[");

		for (i=0; i<N; i++)
		{
				v1[i]=i;
				//printf("%.0lf ",v1[i]);
		}

		//printf("]\n\n");

		//printf("Matriz: \n\n");

		for (f=0; f<N; f++)
		{
				//printf("\n");
				for (c=0; c<N; c++)
				{
						M[f][c] = rand()%(1-10 + 1) + 1;
						//printf("%.0lf ", M[f][c]);
```

```
                        }
                }
                //Medida de tiempo
                t1 = omp_get_wtime();

                //Calcular producto de matriz por vector v2 = M · v1
                for (f=0; f<N; f++)
                        for (c=0; c<N; c++)
                                v2[f] += M[f][c] * v1[c];

                //Medida de tiempo
                t2 = omp_get_wtime();
                total = t2 - t1;

                //Imprimir el resultado y el tiempo de ejecución
                printf("\nTiempo(seg.):%11.9f\t / Tamaño:%u\t/ V2[0]=%8.6f
V2[%d]=%8.6f\n", total,N,v2[0],N-1,v2[N-1]);

                if (N<15)
                {
                        printf("\nv2=[");
                        for (i=0; i<N; i++)
                                printf("%.0lf ",v2[i]);
                        printf("]\n");
                }

                free(v1); // libera el espacio reservado para v1
                free(v2); // libera el espacio reservado para v2
                for (i=0; i<N; i++)
                        free(M[i]);
                free(M);

                return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Se trata de un desenrollado de bucle en el calculo de la matriz multiplicación. Se hacen 4 iteraciones de golpe, por lo tanto solo se permiten matrices cuyo numero de elementos sea múltiplo de 4.

Modificación b) –explicación-: Recorremos las matrices por filas, ya que el compilador las almacena de esta manera.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmm-secuencial-modificado_a.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

double **M1, **M2, **M3;

int main(int argc, char** argv){
                int i, j, f, c, inner;
                double t1, t2, total;
                srand(time(NULL));

                //Leer argumento de entrada (no de componentes del vector)
                if (argc<2){
                        printf("Falta tamaño de matrices\n");
                        exit(-1);
```

```
		}

		unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

		M1 = (double**) malloc(N*sizeof(double *));// malloc necesita el
tamaño en bytes
		M2 = (double**) malloc(N*sizeof(double *)); //si no hay espacio
suficiente malloc devuelve NULL
		M3 = (double**) malloc(N*sizeof(double *));
		if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
			printf("Error en la reserva de espacio para los
vectores\n");
			exit(-2);
		}

		for (i=0; i<N; i++){
			M1[i] = (double*) malloc(N*sizeof(double));
			M2[i] = (double*) malloc(N*sizeof(double));
			M3[i] = (double*) malloc(N*sizeof(double));

			if ( M1[i]==NULL || M2[i]==NULL || M3[i]==NULL ){
				printf("Error en la reserva de espacio
para los vectores\n");
				exit(-2);
			}
		}
		//Inicializar matrices

		for (f=0; f<N; f++)
		{
			for (c=0; c<N; c++)
			{
				M1[f][c] = rand()%(1-10 + 1) + 1;
				M2[f][c] = rand()%(1-10 + 1) + 1;
			}
		}

		printf("Matriz 1: \n");
		for (f=0; f<N; f++)
		{
			printf("\n");
			for (c=0; c<N; c++)
				printf("%.0lf ", M1[f][c]);

		}

		printf("\n\nMatriz 2: \n");
		for (f=0; f<N; f++)
		{
			printf("\n");
			for (c=0; c<N; c++)
				printf("%.0lf ", M2[f][c]);

		}

		//Medida de tiempo
		t1 = omp_get_wtime();

		//Calcular producto de matriz por matriz M1 · M2 = M3
		//int tmp0,tmp1,tmp2,tmp3;
		for (f = 0; f < N; f++)
```

```
              for (c = 0; c < N; c++)
                            {
                                          for (inner = 0; inner < N; inner+=4)
                                          {
                                                        M3[f][c]+=M1[f][inner] *
M2[inner][c];
                                                        M3[f][c]+=M1[f][inner+1]
* M2[inner+1][c];
                                                        M3[f][c]+=M1[f][inner+2]
* M2[inner+2][c];
                                                        M3[f][c]+=M1[f][inner+3]
* M2[inner+3][c];

                                          }

                            }

              //Medida de tiempo
              t2 = omp_get_wtime();
              total = t2 - t1;

        //Imprimir el resultado y el tiempo de ejecución
              printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t", total,N);

              printf("\n\nMatriz 3: \n");
              for (f=0; f<N; f++)
              {
                            printf("\n");
                            for (c=0; c<N; c++)
                                          printf("%.0lf ", M3[f][c]);

              }
              printf("\n");
              for (i=0; i<N; i++)
              {
                            free(M1[i]);
                            free(M2[i]);
                            free(M3[i]);
              }

              free(M1);
              free(M2);
              free(M3);

              return 0;
}
```

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

```
dani@dani-Aspire-5750G:$ ./pmm-secuencial-modificado_a 4
Matriz 1:

1 5 1 6
2 7 4 2
8 6 5 6
7 8 3 1

Matriz 2:

6 5 4 7
3 1 1 2
3 4 7 3
5 4 4 3 Tiempo(seg.):0.000001092          / Tamaño:4

Matriz 3:

54 38 40 38
55 41 51 46
111 90 97 101
80 59 61 77
```

b) `pmm-secuencial-modificado_b.c`

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

double **M1, **M2, **M3;

int main(int argc, char** argv){
            int i, j, f, c, inner;
            double t1, t2, total;
            srand(time(NULL));

            //Leer argumento de entrada (no de componentes del vector)
            if (argc<2){
                        printf("Falta tamaño de matrices\n");
                        exit(-1);
            }

            unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

            M1 = (double**) malloc(N*sizeof(double *));// malloc necesita el
tamaño en bytes
            M2 = (double**) malloc(N*sizeof(double *)); //si no hay espacio
suficiente malloc devuelve NULL
            M3 = (double**) malloc(N*sizeof(double *));
            if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
                        printf("Error en la reserva de espacio para los
vectores\n");
                        exit(-2);
            }
```

```
		for (i=0; i<N; i++){
				M1[i] = (double*) malloc(N*sizeof(double));
				M2[i] = (double*) malloc(N*sizeof(double));
				M3[i] = (double*) malloc(N*sizeof(double));

				if ( M1[i]==NULL || M2[i]==NULL || M3[i]==NULL ){
						printf("Error en la reserva de espacio
para los vectores\n");
						exit(-2);
				}
		}
		//Inicializar matrices

		for (f=0; f<N; f++)
		{
				for (c=0; c<N; c++)
				{
						M1[f][c] = rand()%(1-10 + 1) + 1;
						M2[f][c] = rand()%(1-10 + 1) + 1;
				}
		}

		printf("Matriz 1: \n");
		for (f=0; f<N; f++)
		{
				printf("\n");
				for (c=0; c<N; c++)
						printf("%.0lf ", M1[f][c]);

		}

		printf("\n\nMatriz 2: \n");
		for (f=0; f<N; f++)
		{
				printf("\n");
				for (c=0; c<N; c++)
						printf("%.0lf ", M2[f][c]);

		}

		//Medida de tiempo
		t1 = omp_get_wtime();

		//Calcular producto de matriz por matriz M1 · M2 = M3
		// Recorremos ambas matrices por filas y usamos desenrollado de
bucles.
		for (int f = 0; f < N; f++)
		for (int c= 0; c < N; c++) {
				for (int inner = 0; inner < N; inner+=4) {
						M3[f][inner] += M1[f][c] * M2[c]
[inner];
						M3[f][inner+1] += M1[f][c] * M2[c]
[inner+1];
						M3[f][inner+2] += M1[f][c] * M2[c]
[inner+2];
						M3[f][inner+3] += M1[f][c] * M2[c]
[inner+3];
				}
		}


		//Medida de tiempo
```

```
                t2 = omp_get_wtime();
                total = t2 - t1;

        //Imprimir el resultado y el tiempo de ejecución
                printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t", total,N);

                printf("\n\nMatriz 3: \n");
                for (f=0; f<N; f++)
                {
                        printf("\n");
                        for (c=0; c<N; c++)
                                printf("%.0lf ", M3[f][c]);

                }
                printf("\n");
                for (i=0; i<N; i++)
                {
                        free(M1[i]);
                        free(M2[i]);
                        free(M3[i]);
                }

                free(M1);
                free(M2);
                free(M3);

                return 0;
}
```

```
dani@dani-Aspire-5750G:$ ./pmm-secuencial-modificado_b 4
Matriz 1:

2 5 1 5
2 2 4 4
3 5 2 3
3 3 6 3

Matriz 2:

2 7 3 2
2 3 2 5
3 6 2 5
1 1 6 7 Tiempo(seg.):0.000000817         / Tamaño:4

Matriz 3:

22 40 48 69
24 48 42 62
25 51 41 62
33 69 45 72
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	6.518228108
Modificación a)	6.009105785
Modificación b)	0.677994137

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Como vemos, la modificación b, de desenrrollado del bucle de cálculo no mejora excesivamente los resultados, ya que realmente tampoco ahorramos tantas instrucciones. Reduciríamos el número de saltos en 1000/4 = 250. Ahorraríamos 250 instrucciones de salto, que no parecen suponer tanta mejora. El tamaño del código ha aumentado debido a este desenrollado.

Si vemos muchísima mejora en el caso de acceder a las matrices por filas, ya que en memoria están ordenadas de dicha forma, lo que facilita al procesador el acceso a dichos datos.

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

```
.L25:
                movsd   (%r8,%rsi), %xmm1
                xorl    %ecx, %ecx
                .p2align 4,,10
                .p2align 3
.L23:
                movq    (%rdx, %rcx,8), %rdi
                movsd   (%rax, %rcx,8), %xmm0
                addq    $1, %rcx
                cmpl    %ecx, %r12d
                mulsd   (%rdi,%rsi), %xmm0
                addsd   %xmm0, %xmm1
                movsd   %xmm1, (%r8,%rsi)
                ja      .L23
                addq    $8, %rsi
                cmpq    %rbx, %rsi
                jne     .L25
                addq    $1, %r9
                cmpl
```

pmm-secuencial-modificado_b.s

```
.L25:
                movsd   (%rdi), %xmm0
                xorl    %ecx, %ecx
                xorl    %edx, %edx
                .p2align 4,,10
                .p2align 3
.L23:
                movq    (%rax,%rcx), %r9
                movsd   (%r8,%rcx), %xmm1
                addl    $4, %edx
                mulsd   (%r9,%rsi), %xmm1
                movq    8(%rax,%rcx), %r9
                addsd   %xmm0, %xmm1
                movsd   %xmm1, (%rdi)
                movsd   8(%r8,%rcx), %xmm0
                mulsd   (%r9,%rsi), %xmm0
                movq    16(%rax, %rcx), %r9
                addsd
```

pmm-secuencial-modificado_c.s

```
.L25:
                movq    (%r9,%rdi,8), %rsi
                movq    %rdx, %rcx
                xorl    %eax, %eax
                .p2align 4,,10
                .p2align 3
.L23:
                movsd   (%r8,%rdi,8), %xmm0
                addl    $4, %eax
                addq    $32, %rcx
                addq    $32, %rsi
                mulsd   -32(%rsi), %xmm0
                addsd   -32(%rcx), %xmm0
                movsd   %xmm0, -32(%rcx)
                movsd   (%r8,%rdi,8), %xmm0
                mulsd   -24(%rsi), %xmm0
                addsd   -24(%rcx), %xmm0
```

```
                %r9d, %r12d
                ja
                .L20
                call
                omp_get_wtime
                subsd
                8(%rsp),
%xmm0
                movl
                %ebp, %edx
                movl
                $.LC5, %esi
                movl
                $1, %edi
                movl
                $1, %eax
                xorl
                %r12d, %r12d
                call
                __printf_chk
                movl
                $.LC6, %edi
                call
                puts
```

```
                %xmm1, %xmm0
                movsd
                %xmm0, (%rdi)
                movsd
                16(%r8,%rcx),
%xmm1
                mulsd
                (%r9,%rsi),
%xmm1
                movq
                24(%rax,
%rcx), %r9
                addsd
                %xmm0, %xmm1
                movsd
                %xmm1, (%rdi)
                movsd
                24(%r8,%rcx),
%xmm0
                addq
                $32, %rcx
                cmpl
                %edx, %r12d
                mulsd
                (%r9,%rsi),
%xmm0
                addsd
                %xmm1, %xmm0
                movsd
                %xmm0, (%rdi)
                ja
                .L23
                addq
                $8, %rsi
                addq
                $8, %rdi
                cmpq
                %rbx, %rsi
                jne
                .L25
                addq
                $1, %r10
                cmpl
                %r10d, %r12d
                ja
                .L20
                call
                omp_get_wtime
                subsd
                8(%rsp),
%xmm0
                movl
                %ebp, %edx
                movl
                $.LC5, %esi
                movl
                $1, %edi
                movl
                $1, %eax
                xorl
                %r12d, %r12d
                call
                __printf_chk
                movl
                $.LC6, %edi
                call
                puts
```

```
                movsd
                %xmm0,
-24(%rcx)
                movsd
                (%r8,%rdi,8),
%xmm0
                mulsd
                -16(%rsi),
%xmm0
                addsd
                -16(%rcx),
%xmm0
                movsd
                %xmm0,
-16(%rcx)
                movsd
                (%r8,%rdi,8),
%xmm0
                mulsd
                -8(%rsi),
%xmm0
                addsd
                -8(%rcx),
%xmm0
                movsd
                %xmm0,
-8(%rcx)
                cmpl
                %eax, %r12d
                ja
                .L23
                addq
                $1, %rdi
                cmpl
                %edi, %r12d
                ja
                .L25
                addq
                $1, %r10
                cmpl
                %r10d, %r12d
                ja
                .L20
                call
                omp_get_wtime
                subsd
                8(%rsp),
%xmm0
                movl
                %ebp, %edx
                movl
                $.LC5, %esi
                movl
                $1, %edi
                movl
                $1, %eax
                xorl
                %r12d, %r12d
                call
                __printf_chk
                movl
                $.LC6, %edi
                call
                puts
```

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

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

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

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

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

    for (ii=0; ii<=40000;ii++) {
        for(i=0; i<5000;i++) X1=2*s[i].a+ii;
        for(i=0; i<5000;i++) X2=3*s[i].b-ii;
        if (X1<X2)  R[ii]=X1;
        else  R[ii]=X2;
    }

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

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

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Desenrollamos los bucles en 5 partes usando variables temporales.

Modificación b) –explicación-: Podemos ver que el segundo bucle más interno hace exactamente lo mismo que el primero, por lo que podemos eliminarlo.

1.1. CÓDIGOS FUENTE MODIFICACIONES
a) figura1**`-modificado_a.c`**
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// Modificacion A): Desenrollado de bucles

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

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

main()
{
```

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

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

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

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

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

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

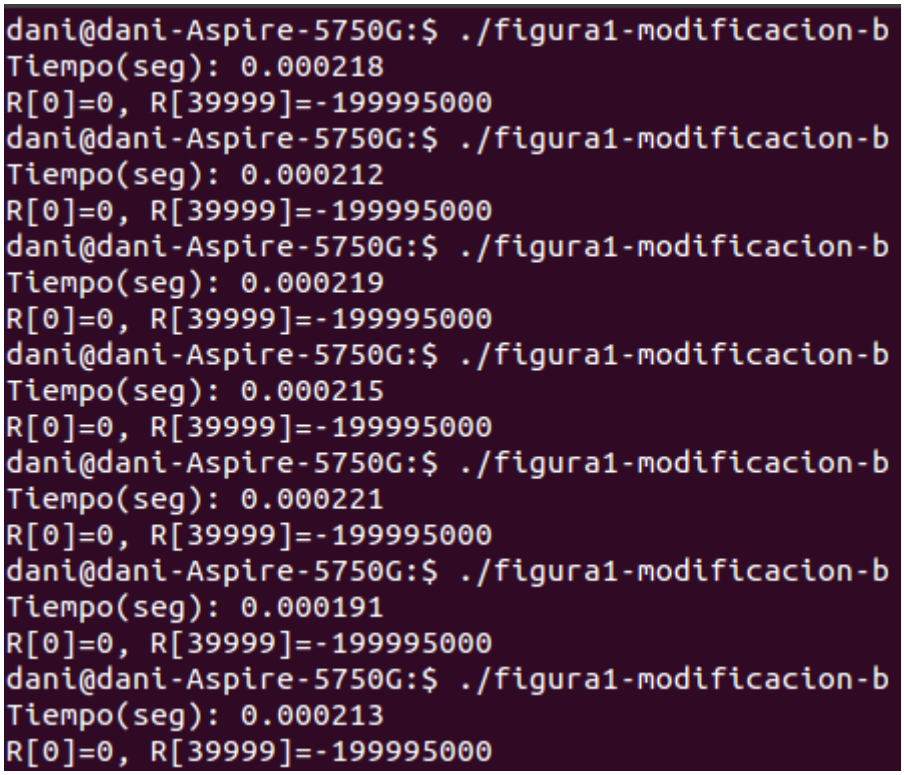
```
dani@dani-Aspire-5750G:$ ./figura1-modificacion-a
Tiempo(seg): 0.000223
R[0]=0, R[39999]=-199995000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-a
Tiempo(seg): 0.000187
R[0]=0, R[39999]=-199995000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-a
Tiempo(seg): 0.000215
R[0]=0, R[39999]=-199995000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-a
Tiempo(seg): 0.000219
R[0]=0, R[39999]=-199995000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-a
Tiempo(seg): 0.000213
R[0]=0, R[39999]=-199995000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-a
Tiempo(seg): 0.000274
R[0]=0, R[39999]=-199995000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-a
Tiempo(seg): 0.000170
R[0]=0, R[39999]=-199995000
```

b) figura1**-modificado_b.c**

```
// Modificacion B): Eliminacion de un bucle innecesario

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

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

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

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

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

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

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

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

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



c) figura1**-modificado_c.c**

```
// Modificacion C): Sacar multiplicacion del interior de un bucle

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

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

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

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

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

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

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

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

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

```
dani@dani-Aspire-5750G:$ ./figura1-modificacion-c
Tiempo(seg): 0.000218
R[0]=0, R[39999]=-599985000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-c
Tiempo(seg): 0.000217
R[0]=0, R[39999]=-599985000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-c
Tiempo(seg): 0.000194
R[0]=0, R[39999]=-599985000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-c
Tiempo(seg): 0.000221
R[0]=0, R[39999]=-599985000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-c
Tiempo(seg): 0.000192
R[0]=0, R[39999]=-599985000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-c
Tiempo(seg): 0.000215
R[0]=0, R[39999]=-599985000
dani@dani-Aspire-5750G:$ ./figura1-modificacion-c
Tiempo(seg): 0.000213
R[0]=0, R[39999]=-599985000
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.000219*
Modificación a)	*0.000170*
Modificación b)	*0.000191*
Modificación c)	*0.000192*

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Como vemos, el tamaño del código aumenta debido a desenrollado de bucles en el caso a. En el caso b se reduce un poco debido al ahorro de un bucle. En cuanto a tiempos, en el mejor de los casos se ha obtenido la mejor diferencia al desenrollar los bucles.

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)

figura1.s	figura1-modificado_a.s	figura1-modificado_b.s	figura1-modificado_c.s
.LFB45: .cfi_startproc subq $160040, %rsp .cfi_def_cfa_offset 160048 xorl %edi, %edi movq	.LFB45: .cfi_startproc subq $160040, %rsp .cfi_def_cfa_offset 160048 xorl %edi, %edi movq	.LFB45: .cfi_startproc subq $160040, %rsp .cfi_def_cfa_offset 160048 xorl %edi, %edi movq	.LFB45: .cfi_startproc subq $160040, %rsp .cfi_def_cfa_offset 160048 xorl %edi, %edi movq

```
                %rsp,
%rsi
                call

clock_gettime
                xorl
                %eax,
%eax
                .p2ali
gn 4,,10
                .p2ali
gn 3
```

```
                %rsp,
%rsi
                call

clock_gettime
                leaq

36(%rsp), %rdx
                movl
                $-
5000, %eax
                .p2ali
gn 4,,10
                .p2ali
gn 3
```

```
                %rsp,
%rsi
                call

clock_gettime
                leaq

36(%rsp), %rdx
                movl
                $-
5000, %eax
                .p2ali
gn 4,,10
                .p2ali
gn 3
```

```
                %rsp,
%rsi
                call

clock_gettime
                leaq

36(%rsp), %rdx
                movl
                $-
15000, %eax
                .p2ali
gn 4,,10
                .p2ali
gn 3
```

2. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina denominada `DAXPY` (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

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

2.1. Genere los programas en ensamblador para cada una de las 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)

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

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

    struct timespec ini,fin;
    double transcurrido;

    // 2. Creación e inicialización de vector y matriz
```

```
    // 2.1. Creación
    int i,a=47;
    int x[N], y[N];

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

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

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

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

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

Tiempos ejec.	**-O0**	**-O2**	**-O3**
	0,010130	*0,004448*	*0,001872*

CAPTURAS DE PANTALLA:

-O0:

```
dani@dani-Aspire-5750G:$ ./daxpyO0 1000000
Tiempo(seg): 0.010130
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO0 1000000
Tiempo(seg): 0.010345
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO0 1000000
Tiempo(seg): 0.009764
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO0 1000000
Tiempo(seg): 0.009470
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO0 1000000
Tiempo(seg): 0.010595
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO0 1000000
Tiempo(seg): 0.014334
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO0 1000000
Tiempo(seg): 0.010820
y[0]=0, y[N-1]=47999952
```

-O2:

```
dani@dani-Aspire-5750G:$ ./daxpy 1000000
Tiempo(seg): 0.004448
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpy 1000000
Tiempo(seg): 0.004249
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpy 1000000
Tiempo(seg): 0.004430
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpy 1000000
Tiempo(seg): 0.003235
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpy 1000000
Tiempo(seg): 0.004506
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpy 1000000
Tiempo(seg): 0.004065
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpy 1000000
Tiempo(seg): 0.004001
y[0]=0, y[N-1]=47999952
```

-O3:

```
dani@dani-Aspire-5750G:$ ./daxpyO3 1000000
Tiempo(seg): 0.001877
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO3 1000000
Tiempo(seg): 0.001872
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO3 1000000
Tiempo(seg): 0.001951
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO3 1000000
Tiempo(seg): 0.001996
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO3 1000000
Tiempo(seg): 0.001932
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO3 1000000
Tiempo(seg): 0.001950
y[0]=0, y[N-1]=47999952
dani@dani-Aspire-5750G:$ ./daxpyO3 1000000
Tiempo(seg): 0.001932
y[0]=0, y[N-1]=47999952
```

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

Extraido del manual de GCC:

Con -O0, el compilador trata de reducir el tamaño del código y tiempo de ejecución sin emplear optimizaciones que requieran un gran tiempo de tiempo de compilación.

-O2 optimiza aún más. GCC realiza casi todas las optimizaciones, menos aquellas que hagan que disminuir el tamaño del código suponga un decremento en la velocidad de ejecución del mismo. Esta opción incrementa el tiempo de compilación y mejora el tiempo de ejecución.

-O3 optimiza aún más, incluyendo las opciones -finline-functions, -funswitch-loops, -fpredictive-commoning, -fgcse-after-reload, -ftree-loop-vectorize, -ftree-loop-distribute-patterns, -fsplit-paths -ftree-slp-vectorize, -fvect-cost-model, -ftree-partial-pre, -fpeel-loops y -fipa-cp-clone.

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)

daxpyO0.s	daxpyO2.s	daxpyO3.s
.file "daxpy.c" .section .rodata .LC0: .string "Falta num\n" .align 8 .LC2: .string "Tiempo(seg): %f\ny[0]=%d, y[N-1]=%d \n" .text .globl main .type main, @function main: .LFB2: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 pushq %r15 pushq %r14 pushq %r13 pushq %r12 pushq	.file "daxpy.c" .section .rodata.str1.1,"aMS",@progbits,1 .LC0: .string "Falta num\n" .section .rodata.str1.8,"aMS",@progbits,1 .align 8 .LC2: .string "Tiempo(seg): %f\ny[0]=%d, y[N-1]=%d \n" .section .text.startup,"ax",@progbits .p2align 4,,15 .globl main .type main, @function main: .LFB45: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 pushq	.file "daxpy.c" .section .rodata.str1.1,"aMS",@progbits,1 .LC0: .string "Falta num\n" .section .rodata.str1.8,"aMS",@progbits,1 .align 8 .LC3: .string "Tiempo(seg): %f\ny[0]=%d, y[N-1]=%d \n" .section .text.startup,"ax",@progbits .p2align 4,,15 .globl main .type main, @function main: .LFB45: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 pushq

```
            %rbx                    %r14                    %r15
            subq                    pushq                   pushq
            $152, %rsp              %r13                    %r14
            .cfi_offset             pushq                   pushq
15, -24                             %r12                    %r13
            .cfi_offset             pushq                   pushq
14, -32                             %rbx                    %r12
            .cfi_offset             subq                    pushq
13, -40                             $32, %rsp               %rbx
            .cfi_offset             .cfi_offset             subq
12, -48                 14, -24                             $56, %rsp
            .cfi_offset             .cfi_offset             .cfi_offset
3, -56                  13, -32                 15, -24
            movl                    .cfi_offset             .cfi_offset
            %edi,       12, -40                 14, -32
-148(%rbp)                          .cfi_offset             .cfi_offset
            movq        3, -48                  13, -40
            %rsi,                   cmpl                    .cfi_offset
-160(%rbp)                          $1, %edi    12, -48
            movq                    jle                     .cfi_offset
            %rsp, %rax              .L12        3, -56
            movq                    movq                    cmpl
            %rax, %rbx              8(%rsi),                $1, %edi
            cmpl        %rdi                                jle
            $1,                     movl                    .L48
-148(%rbp)                          $10, %edx               movq
            jg                      xorl                    8(%rsi),
            .L2                     %esi, %esi  %rdi
            movq                    call                    movl
                                    strtol                  $10, %edx
stderr(%rip), %rax                  movq                    xorl
            movq                    %rax, %r14              %esi, %esi
            %rax, %rcx              movl                    call
            movl                    %eax, %esi              strtol
            $10, %edx               cltq                    movq
            movl                    leaq                    %rax, %rbx
            $1, %esi                18(,                    cltq
            movl        %rax,4), %rdx                       leaq
            $.LC0, %edi             andq                    18(,
            call                    $-16, %rdx  %rax,4), %rax
            fwrite                  subq                    andq
            movl                    %rdx, %rsp              $-16, %rax
            $-1, %edi               movq                    subq
            call                    %rsp, %r12              %rax, %rsp
            exit                    subq                    leaq
.L2:                                %rdx, %rsp              3(%rsp),
            movq                    leaq        %r13
            -160(%rbp),             3(%rsp),                subq
%rax                    %r13                                %rax, %rsp
            addq                    shrq                    leaq
            $8, %rax                $2, %r13                3(%rsp),
            movq                    testl       %r14
            (%rax),                 %r14d,                  shrq
%rax                    %r14d                               $2, %r13
            movq                    leaq                    shrq
            %rax, %rdi              0(,%r13,4),             $2, %r14
            call        %rbx                                testl
            atoi                    jle                     %ebx, %ebx
            movl                    .L3                     leaq
            %eax,                   xorl                    0(,%r13,4),
-128(%rbp)                          %ecx, %ecx  %r15
            movl                    movl                    leaq
```

```
            $47,
-124(%rbp)
            movl
            -128(%rbp),
%eax
            cltq
            subq
            $1, %rax
            movq
            %rax,
-120(%rbp)
            movq
            %rax, %rdx
            addq
            $1, %rdx
            movq
            %rdx,
-176(%rbp)
            movq
            $0,
-168(%rbp)
            movq
            %rax, %rdx
            addq
            $1, %rdx
            movq
            %rdx,
-192(%rbp)
            movq
            $0,
-184(%rbp)
            addq
            $1, %rax
            salq
            $2, %rax
            leaq
            3(%rax),
%rdx
            movl
            $16, %eax
            subq
            $1, %rax
            addq
            %rdx, %rax
            movl
            $16, %esi
            movl
            $0, %edx
            divq
            %rsi
            imulq
            $16, %rax,
%rax
            subq
            %rax, %rsp
            movq
            %rsp, %rax
            addq
            $3, %rax
            shrq
            $2, %rax
            movl
            $1, %edx
            .p2align
4,,10
            .p2align 3
.L5:
            movl
            %edx,
            movl
            %edx,
            addl
            $1, %edx
            addq
            $4, %rcx
            cmpl
            %esi, %edx
            jle
            leaq
            -64(%rbp),
            xorl
            %edi, %edi
            subl
            $1, %r14d
            call
clock_gettime
            movl
            %r14d, %eax
            xorl
            %edx, %edx
            leaq
            4(,%rax,4),
            .p2align
4,,10
            .p2align 3
.L7:
            movl
4(%r12,%rdx), %ecx
            movl
            $47, %eax
            imull
            %eax, %ecx
            addl
            %ecx,
            addq
            $4, %rdx
            cmpq
            %rsi, %rdx
            jne
.L8:
            leaq
            -48(%rbp),
            xorl
            %edi, %edi
            movslq
            %r14d, %r14
            call
clock_gettime
            movq
            -48(%rbp),
            0(,%r14,4),
%r12
            jle
            .L3
            leaq
            4(%r15),
%rax
            andl
            $15, %eax
            shrq
            $2, %rax
            negq
            %rax
            andl
            $3, %eax
            cmpl
            %ebx, %eax
            cmova
            %ebx, %eax
            cmpl
            $6, %ebx
            cmovbe
            %ebx, %eax
            testl
            %eax, %eax
            je
            .L26
            cmpl
            $1, %eax
            movl
            $1, 4(,
%r13,4)
            movl
            $1, 4(,
%r14,4)
            jbe
            .L27
            cmpl
            $2, %eax
            movl
            $2, 8(,
%r13,4)
            movl
            $2, 8(,
%r14,4)
            jbe
            .L28
            cmpl
            $3, %eax
            movl
            $3, 12(,
%r13,4)
            movl
            $3, 12(,
%r14,4)
            jbe
            .L29
            cmpl
            $4, %eax
            movl
            $4, 16(,
```

```
        salq
        $2, %rax
        movq
        %rax,
-112(%rbp)
        movl
        -128(%rbp),
%eax
        cltq
        subq
        $1, %rax
        movq
        %rax,
-104(%rbp)
        movq
        %rax, %rdx
        addq
        $1, %rdx
        movq
        %rdx, %r14
        movl
        $0, %r15d
        movq
        %rax, %rdx
        addq
        $1, %rdx
        movq
        %rdx, %r12
        movl
        $0, %r13d
        addq
        $1, %rax
        salq
        $2, %rax
        leaq
        3(%rax),
%rdx
        movl
        $16, %eax
        subq
        $1, %rax
        addq
        %rdx, %rax
        movl
        $16, %esi
        movl
        $0, %edx
        divq
        %rsi
        imulq
        $16, %rax,
%rax
        subq
        %rax, %rsp
        movq
        %rsp, %rax
        addq
        $3, %rax
        shrq
        $2, %rax
        salq
        $2, %rax
        subq
        -64(%rbp),
%rax
        movl
        $.LC2, %esi
        movl
        (%rbx,
%r14,4), %ecx
        movl
        0(,%r13,4),
%edx
        movl
        $1, %edi
        cvtsi2sdq
        %rax, %xmm0
        movq
        -40(%rbp),
%rax
        subq
        -56(%rbp),
%rax
        cvtsi2sdq
        %rax, %xmm1
        movl
        $1, %eax
        divsd
        .LC1(%rip),
%xmm1
        addsd
        %xmm1,
%xmm0
        call

__printf_chk
        leaq
        -32(%rbp),
%rsp
        xorl
        %eax, %eax
        popq
        %rbx
        popq
        %r12
        popq
        %r13
        popq
        %r14
        popq
        %rbp
        .cfi_rememb
er_state
        .cfi_def_cf
a 7, 8
        ret
.L3:
        .cfi_restor
e_state
        leaq
        -64(%rbp),
%rsi
        xorl
        %edi, %edi
%r13,4)
        movl
        $4, 16(,
%r14,4)
        jbe
        .L30
        cmpl
        $5, %eax
        movl
        $5, 20(,
%r13,4)
        movl
        $5, 20(,
%r14,4)
        jbe
        .L31
        movl
        $6, 24(,
%r13,4)
        movl
        $6, 24(,
%r14,4)
        movl
        $7, %edx
.L6:
        cmpl
        %eax, %ebx
        je
        .L7
.L5:
        movl
        %ebx, %r8d
        movl
        %eax, %ecx
        subl
        %eax, %r8d
        movl
        %r8d, %esi
        shrl
        $2, %esi
        leal
        0(,%rsi,4),
%edi
        testl
        %edi, %edi
        je
        .L8
        leal
        1(%rdx),
%eax
        leaq
        4(,%rcx,4),
%r9
        xorl
        %ecx, %ecx
        movdqa
        .LC1(%rip),
%xmm1
        movl
        %eax,
-84(%rbp)
        leal
```

```
            movq
            %rax,
-96(%rbp)
            movl
            $1,
-132(%rbp)
            jmp
            .L3
.L4:
            movq
            -112(%rbp),
%rax
            movl
            -132(%rbp),
%edx
            movslq
            %edx, %rdx
            movl
            -132(%rbp),
%ecx
            movl
            %ecx,
(%rax,%rdx,4)
            movq
            -96(%rbp),
%rax
            movl
            -132(%rbp),
%edx
            movslq
            %edx, %rdx
            movl
            -132(%rbp),
%ecx
            movl
            %ecx,
(%rax,%rdx,4)
            addl
            $1,
-132(%rbp)
.L3:
            movl
            -132(%rbp),
%eax
            cmpl
            -128(%rbp),
%eax
            jle
            .L4
            leaq
            -80(%rbp),
%rax
            movq
            %rax, %rsi
            movl
            $0, %edi
            call

clock_gettime
            movl
            $1,
```

```
            subl
            $1, %r14d
            call

clock_gettime
            jmp
            .L8
.L12:
            movq

stderr(%rip), %rcx
            movl
            $.LC0, %edi
            movl
            $10, %edx
            movl
            $1, %esi
            call
            fwrite
            orl
            $-1, %edi
            call
            exit
            .cfi_endpro
c
.LFE45:
            .size
            main,
.-main
            .section
            .rodata.cst
8,"aM",@progbits,8
            .align 8
.LC1:
            .long
            0
            .long
            1104006501
            .ident
            "GCC:
(Ubuntu 4.8.4-
2ubuntu1~14.04.3) 4.8.4"
            .section
            .note.GNU-
stack,"",@progbits
```

```
            2(%rdx),
%eax
            leaq
            (%r15,%r9),
%r10
            movd
            -84(%rbp),
%xmm4
            addq
            %r12, %r9
            movl
            %eax,
-88(%rbp)
            leal
            3(%rdx),
%eax
            movd
            -88(%rbp),
%xmm2
            movl
            %edx,
-88(%rbp)
            movl
            %eax,
-92(%rbp)
            movd
            -88(%rbp),
%xmm0
            xorl
            %eax, %eax
            movd
            -92(%rbp),
%xmm3
            punpckldq
            %xmm4,
%xmm0
            punpckldq
            %xmm3,
%xmm2
            punpcklqdq
            %xmm2,
%xmm0
.L13:
            movdqa
            %xmm0,
%xmm2
            addl
            $1, %ecx
            movdqa
            %xmm0,
(%r10,%rax)
            paddd
            %xmm1,
%xmm2
            movdqu
            %xmm0,
(%r9,%rax)
            addq
            $16, %rax
            cmpl
            %esi, %ecx
```

```
-132(%rbp)
            jmp
            .L5
.L6:
            movq
            -112(%rbp),
%rax
            movl
            -132(%rbp),
%edx
            movslq
            %edx, %rdx
            movl
            (%rax,
%rdx,4), %eax
            imull
            -124(%rbp),
%eax
            movl
            %eax, %ecx
            movq
            -96(%rbp),
%rax
            movl
            -132(%rbp),
%edx
            movslq
            %edx, %rdx
            movl
            (%rax,
%rdx,4), %eax
            addl
            %eax, %ecx
            movq
            -96(%rbp),
%rax
            movl
            -132(%rbp),
%edx
            movslq
            %edx, %rdx
            movl
            %ecx,
(%rax,%rdx,4)
            addl
            $1,
-132(%rbp)
.L5:
            movl
            -132(%rbp),
%eax
            cmpl
            -128(%rbp),
%eax
            jle
            .L6
            leaq
            -64(%rbp),
%rax
            movq
            %rax, %rsi
            movl
```

```
            jnb
            .L49
            movdqa
            %xmm2,
%xmm0
            jmp
            .L13
.L49:
            addl
            %edi, %edx
            cmpl
            %edi, %r8d
            je
            .L7
.L8:
            movslq
            %edx, %rax
            movl
            %edx,
(%r15,%rax,4)
            movl
            %edx,
(%r12,%rax,4)
            leal
            1(%rdx),
%eax
            cmpl
            %eax, %ebx
            jl
            .L7
            addl
            $2, %edx
            movslq
            %eax, %rcx
            cmpl
            %edx, %ebx
            movl
            %eax,
(%r15,%rcx,4)
            movl
            %eax,
(%r12,%rcx,4)
            jl
            .L7
            movslq
            %edx, %rax
            movl
            %edx,
(%r15,%rax,4)
            movl
            %edx,
(%r12,%rax,4)
.L7:
            leaq
            -80(%rbp),
%rsi
            xorl
            %edi, %edi
            call

clock_gettime
            leaq
```

```
            $0, %edi
            call
clock_gettime
            movq
            -64(%rbp),
            movq
            -80(%rbp),
            subq
            %rax, %rdx
            movq
            %rdx, %rax
            cvtsi2sdq
            %rax, %xmm1
            movq
            -56(%rbp),
            movq
            -72(%rbp),
            subq
            %rax, %rdx
            movq
            %rdx, %rax
            cvtsi2sdq
            %rax, %xmm0
            movsd
            .LC1(%rip),
%xmm2
            divsd
            %xmm2,
%xmm0
            addsd
            %xmm1,
%xmm0
            movsd
            %xmm0,
-88(%rbp)
            movl
            -128(%rbp),
            leal
            -1(%rax),
            movq
            -96(%rbp),
            movslq
            %edx, %rdx
            movl
            (%rax,
%rdx,4), %edx
            movq
            -96(%rbp),
            movl
            (%rax),
%ecx
            movq
            -88(%rbp),
            movl
            %ecx, %esi
            movq
            %rax,
            movsd
            -176(%rbp),
            movl
```

```
            4(%r12),
%rax
            andl
            $15, %eax
            shrq
            $2, %rax
            negq
            %rax
            andl
            $3, %eax
            cmpl
            %ebx, %eax
            cmova
            %ebx, %eax
            cmpl
            $4, %ebx
            cmovbe
            %ebx, %eax
            testl
            %eax, %eax
            je
            .L33
            imull
            $47, 4(,
%r13,4), %edx
            addl
            %edx,
4(%r12)
            cmpl
            $1, %eax
            jbe
            .L34
            imull
            $47, 8(,
%r13,4), %edx
            addl
            %edx,
8(%r12)
            cmpl
            $2, %eax
            jbe
            .L35
            imull
            $47, 12(,
%r13,4), %edx
            addl
            %edx,
12(%r12)
            cmpl
            $3, %eax
            jbe
            .L36
            imull
            $47, 16(,
%r13,4), %edx
            addl
            %edx,
16(%r12)
            movl
            $5, %edx
.L20:
```

```
        $.LC2, %edi
        movl
        $1, %eax
        call
        printf
        movq
        %rbx, %rsp
        movl
        $0, %eax
        leaq
        -40(%rbp),
%rsp
        popq
        %rbx
        popq
        %r12
        popq
        %r13
        popq
        %r14
        popq
        %r15
        popq
        %rbp
        .cfi_def_cf
a 7, 8
        ret
        .cfi_endpro
c
.LFE2:
        .size
        main,
.-main
        .section
        .rodata
        .align 8
.LC1:
        .long
        0
        .long
        1104006501
        .ident
        "GCC:
(Ubuntu 4.8.4-
2ubuntu1~14.04.3) 4.8.4"
        .section
        .note.GNU-
stack,"",@progbits
```

```
        cmpl
        %ebx, %eax
        je
        .L24
.L19:
        movl
        %ebx, %r9d
        movl
        %eax, %ecx
        subl
        %eax, %r9d
        movl
        %r9d, %edi
        shrl
        $2, %edi
        leal
        0(,%rdi,4),
%r8d
        testl
        %r8d, %r8d
        je
        .L22
        leaq
        4(,%rcx,4),
%rsi
        xorl
        %eax, %eax
        xorl
        %ecx, %ecx
        leaq

(%r15,%rsi), %r10
        addq
        %r12, %rsi
.L23:
        movdqu

(%r10,%rax), %xmm1
        addl
        $1, %ecx
        movdqa
        %xmm1,
%xmm2
        pslld
        $4, %xmm2
        movdqa
        %xmm2,
%xmm0
        pslld
        $2, %xmm0
        psubd
        %xmm2,
%xmm0
        psubd
        %xmm1,
%xmm0
        paddd
        (%rsi,
%rax), %xmm0
        movdqa
        %xmm0,
(%rsi,%rax)
```

		addq $16, %rax
		cmpl %edi, %ecx
		jb .L23
		addl %r8d, %edx
		cmpl %r8d, %r9d
		je .L24
		.L22:
		movslq %edx, %rax
		imull $47, (%r15,%rax,4), %ecx
		addl %ecx, (%r12,%rax,4)
		leal 1(%rdx), %eax
		cmpl %eax, %ebx
		jl .L24
		cltq
		addl $2, %edx
		imull $47, (%r15,%rax,4), %ecx
		addl %ecx, (%r12,%rax,4)
		cmpl %ebx, %edx
		jg .L24
		movslq %edx, %rdx
		imull $47, (%r15,%rdx,4), %eax
		addl %eax, (%r12,%rdx,4)
		.L24:
		leaq -64(%rbp), %rsi
		xorl %edi, %edi
		subl $1, %ebx
		movslq %ebx, %rbx
		call

```
clock_gettime
                movq
                -64(%rbp),
%rax
                movl
                $.LC3, %esi
                movl
                $1, %edi
                subq
                -80(%rbp),
%rax
                movl

(%r12,%rbx,4), %ecx
                movl
                0(,%r14,4),
%edx
                cvtsi2sdq
                %rax, %xmm0
                movq
                -56(%rbp),
%rax
                subq
                -72(%rbp),
%rax
                cvtsi2sdq
                %rax, %xmm1
                movl
                $1, %eax
                divsd
                .LC2(%rip),
%xmm1
                addsd
                %xmm1,
%xmm0
                call

__printf_chk
                leaq
                -40(%rbp),
%rsp
                xorl
                %eax, %eax
                popq
                %rbx
                popq
                %r12
                popq
                %r13
                popq
                %r14
                popq
                %r15
                popq
                %rbp
                .cfi_rememb
er_state
                .cfi_def_cf
a 7, 8
                ret
.L33:
                .cfi_restor
```

```
e_state
                movl
                $1, %edx
                jmp
                .L19
.L26:
                movl
                $1, %edx
                jmp
                .L5
.L36:
                movl
                $4, %edx
                jmp
                .L20
.L34:
                movl
                $2, %edx
                jmp
                .L20
.L35:
                movl
                $3, %edx
                jmp
                .L20
.L31:
                movl
                $6, %edx
                jmp
                .L6
.L27:
                movl
                $2, %edx
                jmp
                .L6
.L28:
                movl
                $3, %edx
                jmp
                .L6
.L29:
                movl
                $4, %edx
                jmp
                .L6
.L30:
                movl
                $5, %edx
                jmp
                .L6
.L3:
                leaq
                -80(%rbp),
%rsi
                xorl
                %edi, %edi
                call

clock_gettime
                jmp
                .L24
```

		.L48: movq stderr(%rip), %rcx movl $.LC0, %edi movl $10, %edx movl $1, %esi call fwrite orl $-1, %edi call exit .cfi_endpro c .LFE45: .size main, .-main .section .rodata.cst 16,"aM",@progbits,16 .align 16 .LC1: .long 4 .long 4 .long 4 .long 4 .section .rodata.cst 8,"aM",@progbits,8 .align 8 .LC2: .long 0 .long 1104006501 .ident "GCC: (Ubuntu 4.8.4- 2ubuntu1~14.04.3) 4.8.4" .section .note.GNU- stack,"",@progbits