2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos):
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz*

Sistema operativo utilizado: *Ubuntu 16.04 LTS*

Versión de gcc utilizada: *gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.9)*

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

```
guillesiesta@guillesiesta:~$ lscpu
Arquitectura:          x86_64
modo(s) de operación de las CPUs:32-bit, 64-bit
Orden de bytes:        Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Hilo(s) de procesamiento por núcleo:2
Núcleo(s) por «socket»:4
Socket(s):             1
Modo(s) NUMA:          1
ID de fabricante:      GenuineIntel
Familia de CPU:        6
Modelo:                94
Model name:            Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
Revisión:              3
CPU MHz:               800.033
CPU max MHz:           3500,0000
CPU min MHz:           800,0000
BogoMIPS:              5184.00
Virtualización:        VT-x
Caché L1d:             32K
Caché L1i:             32K
Caché L2:              256K
Caché L3:              6144K
NUMA node0 CPU(s):     0-7
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm consta
nt_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc
_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm
 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand lahf_lm abm 3dnowprefet
ch cpuid_fault epb invpcid_single pti retpoline intel_pt rsb_ctxsw tpr_shadow vnmi flexpr
iority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx rdseed a
dx smap clflushopt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify
hwp_act_window hwp_epp
```

1. Para el núcleo que se muestra en el Figura 1, y para un programa que implemente la multiplicación de matrices (use variables globales):

   1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos (use –O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

   1.2 Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.

   1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

Figura 1 . Código C++ que suma dos vectores

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

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

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

A) MULTIPLICACIÓN DE MATRICES:
CAPTURA CÓDIGO FUENTE: `pmm-secuencial.c`

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

int main(int argc, char **argv)
{
    unsigned i, j, k;

    if(argc < 2)
    {
        fprintf(stderr, "falta size\n");
        exit(-1);
    }


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

    int **ma, **mb, **mc;
    ma = (int **) malloc(N*sizeof(int*));
    mb = (int **) malloc(N*sizeof(int*));
    mc = (int **) malloc(N*sizeof(int*));

    //reserva de memoria
    for (i=0; i<N; i++){
        ma[i] = (int *) malloc(N*sizeof(int));
        mb[i] = (int *) malloc(N*sizeof(int));
        mc[i] = (int *) malloc(N*sizeof(int));
    }

    // Inicialización
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            ma[i][j] = 0; //aquí es donde almacenaremos el resultado
            mb[i][j] = 6;
            mc[i][j] = 4;
        }
    }

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);
    // Multiplicación
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            for (k=0; k<N; k++){
                ma[i][j] += mb[i][k] * mc[k][j];
            }
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

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

    // Pitamos la primera y la ultima linea de la matriz resultante
    printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,ma[0][0],ma[N-1][N-1]);

    // Liberamos la memoria
    for (i=0; i<N; i++)
    {
        free(ma[i]);
        free(mb[i]);
        free(mc[i]);
    }
    free(ma);
    free(mb);
    free(mc);

    return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-:

He invertido las variables j y k en el bucle, al estar así los datos más cerca en memoria

Modificación b) –explicación-:

He desenrollado el bucle en bloques de 8

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

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

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

int main(int argc, char **argv)
{
    unsigned i, j, k;

    if(argc < 2)
    {
        fprintf(stderr, "falta size\n");
        exit(-1);
    }


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

    int **ma, **mb, **mc;
    ma = (int **) malloc(N*sizeof(int*));
    mb = (int **) malloc(N*sizeof(int*));
    mc = (int **) malloc(N*sizeof(int*));

    //reserva de memoria
    for (i=0; i<N; i++){
        ma[i] = (int *) malloc(N*sizeof(int));
        mb[i] = (int *) malloc(N*sizeof(int));
        mc[i] = (int *) malloc(N*sizeof(int));
    }

    // Inicialización
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            ma[i][j] = 0; //aquí es donde almacenaremos el resultado
            mb[i][j] = 6;
            mc[i][j] = 4;
        }
    }

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);
    // Multiplicación
    for (i=0; i<N; i++){
        for (k=0; k<N; k++){
            for (j=0; j<N; j++){
                ma[i][j] += mb[i][k] * mc[k][j];
            }
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    // Pitamos la primera y la ultima linea de la matriz resultante
    printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,ma[0][0],ma[N-1][N-1]);


    // Pitamos la primera y la ultima linea de la matriz resultante
    printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,ma[0][0],ma[N-1][N-1]);

    // Liberamos la memoria
    for (i=0; i<N; i++)
    {
        free(ma[i]);
        free(mb[i]);
        free(mc[i]);
    }
    free(ma);
    free(mb);
    free(mc);

    return 0;
}
```

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

```
guillesiesta@guillesiesta:~$ gcc -O2 pmm-secuencial.c -o pm
guillesiesta@guillesiesta:~$ ./pm 2000
Tiempo = 48.565257990     Primera = 48000         Ultima=48000
guillesiesta@guillesiesta:~$ gcc -O2 pmm-secuencial-modA.c -o pmA
guillesiesta@guillesiesta:~$ ./pmA 2000
Tiempo = 5.317406115      Primera = 48000         Ultima=48000
guillesiesta@guillesiesta:~$
```

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

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

int main(int argc, char **argv)
{
    unsigned i, j, k;
    int total = 0;
    int h;
    int s1,s2,s3,s4,s5,s6,s7,s8;
    s1=s2=s3=s4=s5=s6=s7=s8=0;


    if(argc < 2)
    {
        fprintf(stderr, "falta size\n");
        exit(-1);
    }


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

    int **ma, **mb, **mc;
    ma = (int **) malloc(N*sizeof(int*));
    mb = (int **) malloc(N*sizeof(int*));
    mc = (int **) malloc(N*sizeof(int*));

    //reserva de memoria
    for (i=0; i<N; i++){
        ma[i] = (int *) malloc(N*sizeof(int));
        mb[i] = (int *) malloc(N*sizeof(int));
        mc[i] = (int *) malloc(N*sizeof(int));
    }

    // Inicialización
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            ma[i][j] = 0; //aquí es donde almacenaremos el resultado
            mb[i][j] = 6;
            mc[i][j] = 4;
        }
    }

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    int iterations = N/8;

```

```
    // Multiplicación
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
        {
            s1=s2=s3=s4=s5=s6=s7=s8=0;
            for (h=0, k=0;h < iterations; ++h, k+=8)
            {
                    s1 += (mb[i][k]  *mc[k][j]  );
                    s2 += (mb[i][k+1]*mc[j][k+1]);
                    s3 += (mb[i][k+2]*mc[j][k+2]);
                    s4 += (mb[i][k+3]*mc[j][k+3]);
                    s5 += (mb[i][k+4]*mc[j][k+4]);
                    s6 += (mb[i][k+5]*mc[j][k+5]);
                    s7 += (mb[i][k+6]*mc[j][k+6]);
                    s8 += (mb[i][k+7]*mc[j][k+7]);
            }

            total = s1 + s2 + s3 + s4 + s5 + s6 + s7 + s8;
            ma[i][j]=total;

            for(k=iterations*8; k<N; ++k)
                total += (mb[i][k]*mc[j][k]);

            ma[i][j]=total;
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    // Pitamos la primera y la ultima linea de la matriz resultante
    printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,ma[0][0],ma[N-1][N-1]);

    // Liberamos la memoria
    for (i=0; i<N; i++)
    {
        free(ma[i]);
        free(mb[i]);
        free(mc[i]);
    }
    free(ma);
    free(mb);
    free(mc);

    return 0;
}
```

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

```
guillesiesta@guillesiesta:~$ gcc -O2 pmm-secuencial.c -o pm
guillesiesta@guillesiesta:~$ ./pm 2000
Tiempo = 48.565257990    Primera = 48000         Ultima=48000
guillesiesta@guillesiesta:~$ gcc -O2 pmm-secuencial-modA.c -o pmA
guillesiesta@guillesiesta:~$ ./pmA 2000
Tiempo = 5.317406115     Primera = 48000         Ultima=48000
guillesiesta@guillesiesta:~$ gcc -O2 pmm-secuencial-modB.c -o pmB
guillesiesta@guillesiesta:~$ ./pmB 2000
Tiempo = 4.630081184     Primera = 48000         Ultima=48000
guillesiesta@guillesiesta:~$
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*48.565257990*
Modificación a)	5.317406115
Modificación b)	4.630081184
...	

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Es increíble como con una modificación tan sencilla en el bucle se puede llegar a tener algo casi unas 10 veces más rápido. Hay que tener en cuenta la arquitectura del computador que ejecuta el programa para poder realizar estas modificaciones, sin embargo, merece la pena pues se aumenta bastante el rendimiento.

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

pmm-secuencial.s

```
.L16:
                movq
                0(%rbp,%rbx,8),
%r11
                movq
                (%r12,%rbx,8),
%rdi
                xorl
                %r10d, %r10d
                .p2align 4,,10
                .p2align 3
.L12:
                movl
                (%r11,%r10,4),
%ecx
                leaq
                0(,%r10,4),
%rsi
                xorl
                %eax, %eax
                .p2align 4,,10
                .p2align 3
.L10:
                movq
                (%r14,%rax,8),
%rdx
                movl
                (%rdx,%rsi),
%edx
                imull
                (%rdi,%rax,4),
%edx
                addq
                $1, %rax
                addl
                %edx, %ecx
                cmpl
                %eax, %r15d
                ja
                .L10
                movl
                %ecx,
(%r11,%r10,4)
                addq
                $1, %r10
                cmpl
                %r10d, %r15d
                ja
                .L12
                addq
                $1, %rbx
                cmpl
                %ebx, %r15d
                ja
                .L16
.L9:
                leaq
```

pmm-secuencial-modificado_b.s

```
call            clock_gettime
.L16:
                movq
                0(%rbp,%rbx,8),
%rdx
                movq
                (%r12,%rbx,8),
%r11
                xorl
                %r10d, %r10d
                .p2align 4,,10
                .p2align 3
.L12:
                movl
                (%r11,%r10,4),
%edi
                movq
                (%r14,%r10,8),
%rsi
                xorl
                %eax, %eax
                .p2align 4,,10
                .p2align 3
.L10:
                movl
                (%rsi,%rax,4),
%ecx
                imull
                %edi, %ecx
                addl
                %ecx, (%rdx,
%rax,4)
                addq
                $1, %rax
                cmpl
                %eax, %r15d
                ja
                .L10
                addq
                $1, %r10
                cmpl
                %r10d, %r15d
                ja
                .L12
                addq
                $1, %rbx
                cmpl
                %ebx, %r15d
                ja
                .L16
.L9:
                leaq
                48(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

pmm-secuencial-modificado_c.s

```
call            clock_gettime
                movl
                80(%rsp), %eax
                movq
                $0, 16(%rsp)
                shrl
                $3, %eax
                leal
                0(,%rax,8),
%ebx
                movl
                %eax, 60(%rsp)
                subl
                $1, %eax
                addq
                $1, %rax
                salq
                $5, %rax
                movl
                %ebx, 88(%rsp)
                movq
                %rax, 64(%rsp)
.L19:
                movq
                72(%rsp), %rax
                movq
                16(%rsp), %rbx
                xorl
                %r13d, %r13d
                movq
                $0, 8(%rsp)
                movq
                (%rax,%rbx,8),
%rax
                movq
                %rax, 48(%rsp)
                .p2align 4,,10
                .p2align 3
.L15:
                movl
                60(%rsp), %edx
                testl
                %edx, %edx
                je
                .L21
                movq
                16(%rsp), %rbx
                movq
                24(%rsp), %rax
                xorl
                %r12d, %r12d
                movq
                8(%rsp), %rcx
                xorl
                %ebp, %ebp
                xorl
                %r11d, %r11d
```

```
                48(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
                movl
                28(%rsp), %ebx
                pxor
                %xmm0, %xmm0
                movq
                0(%rbp), %rdx
                movl
                $.LC2, %esi
                movl
                $1, %edi
                movq
                0(%rbp,%rbx,8),
%rax
                movl
                (%rdx), %edx
                leaq
                8(,%rbx,8),
%r13
                movl
                (%rax,%rbx,4),
%ecx
                movq
                56(%rsp), %rax
                xorl
                %ebx, %ebx
                subq
                40(%rsp), %rax
                cvtsi2sdq
                %rax, %xmm0
                movq
                48(%rsp), %rax
                subq
                32(%rsp), %rax
                movapd
                %xmm0, %xmm1
                pxor
                %xmm0, %xmm0
                divsd
                .LC1(%rip),
%xmm1
                cvtsi2sdq
                %rax, %xmm0
                movl
                $1, %eax
                addsd
                %xmm1, %xmm0
                call
                __printf_chk
                movl
                16(%rsp), %eax
                testl
                %eax, %eax
                je
                .L15
.L18:
                movq
                0(%rbp,%rbx),
%rdi
                call
                free
                movq
                (%r12,%rbx),
%rdi
                call
                free
                movq
                (%r14,%rbx),
%rdi
                addq
                $8, %rbx
                call
                free
                cmpq
                %r13, %rbx
```

```
                xorl
                %r10d, %r10d
                xorl
                %r9d, %r9d
                xorl
                %r8d, %r8d
                movq
                (%rax,%rbx,8),
%rax
                movq
                32(%rsp), %rbx
                movq
                %rbx, %rdi
                movq
                (%rbx,%rcx,8),
%rbx
                xorl
                %ecx, %ecx
                movq
                %rbx, 40(%rsp)
                leaq
                4(%rbx), %rdx
                movq
                64(%rsp), %rbx
                leaq
                (%rax,%rbx),
%r14
                xorl
                %ebx, %ebx
                .p2align 4,,10
                .p2align 3
.L10:
                movq
                (%rdi), %rsi
                addq
                $32, %rax
                addq
                $64, %rdi
                addq
                $32, %rdx
                movl
                (%rsi,%r13),
%esi
                imull
                -32(%rax), %esi
                addl
                %esi, %ecx
                movl
                -28(%rax), %esi
                imull
                -32(%rdx), %esi
                addl
                %esi, %r8d
                movl
                -24(%rax), %esi
                imull
                -28(%rdx), %esi
                addl
                %esi, %r9d
                movl
                -20(%rax), %esi
                imull
                -24(%rdx), %esi
                addl
                %esi, %r10d
                movl
                -16(%rax), %esi
                imull
                -20(%rdx), %esi
                addl
                %esi, %r11d
                movl
                -12(%rax), %esi
                imull
                -16(%rdx), %esi
                addl
                %esi, %ebx
                movl
                -8(%rax), %esi
```

```
                jne
                .L18
.L15:
                movq
                %rbp, %rdi
                call
                free
                movq
                %r12, %rdi
                call
                free
                movq
                %r14, %rdi
                call
                free
                xorl
                %eax, %eax
                movq
                72(%rsp), %rbx
                xorq
                %fs:40, %rbx
                jne
                .L30
                addq
                $88, %rsp
                .cfi_remember_s
tate
                .cfi_def_cfa_of
fset 56
                popq
                %rbx
                .cfi_def_cfa_of
fset 48
                popq
                %rbp
                .cfi_def_cfa_of
fset 40
                popq
                %r12
                .cfi_def_cfa_of
fset 32
                popq
                %r13
                .cfi_def_cfa_of
fset 24
                popq
                %r14
                .cfi_def_cfa_of
fset 16
                popq
                %r15
                .cfi_def_cfa_of
fset 8
                ret
.L3:
                .cfi_restore_st
ate
                leaq
                32(%rsp), %rsi
                xorl
                %edi, %edi
```

```
                imull
                -12(%rdx), %esi
                addl
                %esi, %ebp
                movl
                -4(%rax), %esi
                imull
                -8(%rdx), %esi
                addl
                %esi, %r12d
                cmpq
                %r14, %rax
                jne
                .L10
                addl
                %r8d, %ecx
                addl
                %r9d, %ecx
                addl
                %r10d, %ecx
                addl
                %ecx, %r11d
                addl
                %r11d, %ebx
                addl
                %ebx, %ebp
                addl
                %ebp, %r12d
.L14:
                movl
                88(%rsp), %eax
                cmpl
                %eax, %r15d
                jbe
                .L11
                movq
                24(%rsp), %rbx
                movq
                16(%rsp), %rcx
                movq
                (%rbx,%rcx,8),
%rsi
                movq
                32(%rsp), %rbx
                movq
                8(%rsp), %rcx
                movq
                (%rbx,%rcx,8),
%rcx
                .p2align 4,,10
                .p2align 3
.L12:
                movl
                %eax, %edi
                addl
                $1, %eax
                movl
                (%rsi,%rdi,4),
%edx
                imull
                (%rcx,%rdi,4),
%edx
                addl
                %edx, %r12d
                cmpl
                %eax, %r15d
                jne
                .L12
.L11:
                movq
                48(%rsp), %rax
                addq
                $1, 8(%rsp)
                movl
                %r12d, (%rax,
%r13)
                addq
                $4, %r13
                movq
```

			8(%rsp), %rax cmpl %eax, %r15d ja .L15 addq $1, 16(%rsp) movq 16(%rsp), %rax cmpl %eax, %r15d ja .L19
		.L9:	leaq 112(%rsp), %rsi xorl %edi, %edi call clock_gettime

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

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

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

int main(int argc, char **argv)
{
    int ii, i, X1, X2;
    int R[40000];

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (ii = 1; 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;
    }

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

    printf("R[0] = %i, R[39999] = %i\n", R[0], R[39999]);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-:

Se han agrupado los dos for en uno solo, pues hacen los mismo. Y se ha usado el operador ternario en lugar del if/else.

Modificación b) –explicación-:

También he agrupado los dos for en uno, y he sustituido el if/else por el operador ternario. A esta modificación le añado un desenrrollado de bucle de 5.

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1**`-modificado_a.c`**

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

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

int main(int argc, char **argv)
{
    int ii, i, X1, X2;
    int R[40000];

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (ii = 1; 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;


        R[ii] = ( X1 < X2 ) ? X1 : X2;
    }

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

    printf("R[0] = %i, R[39999] = %i\n", R[0], R[39999]);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    return 0;
}
```

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

```
guillesiesta@guillesiesta:~$ gcc -O2 figura1-original.c -o f
guillesiesta@guillesiesta:~$ ./f
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.205815694
guillesiesta@guillesiesta:~$ gcc -O2 figura1-modA.c -o fA -w
guillesiesta@guillesiesta:~$ ./fA
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.157116599
guillesiesta@guillesiesta:~$
```

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

pmm-secuencial-modB.c × pmm-secuencial.s × figura1-original.c × figura1-modA.c × figura1-mod

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

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

int main(int argc, char **argv)
{
    int ii, i, X1, X2;
    int R[40000];

    struct timespec cgt1,cgt2; double ncgt;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (ii = 1; ii <= 40000; ii++)
    {
        X1 = 0; X2 = 0;

         for (i = 0; i < 5000; i+=4)
        {
            X1 += 2*s[i].a+ii;
            X2 += 3*s[i].b-ii;
            X1 += 2*s[i+1].a+ii;
            X2 += 3*s[i+1].b-ii;
            X1 += 2*s[i+2].a+ii;
            X2 += 3*s[i+2].b-ii;
            X1 += 2*s[i+3].a+ii;
            X2 += 3*s[i+3].b-ii;
        }

         R[ii] = ( X1 < X2 ) ? X1 : X2;
    }

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

    printf("R[0] = %i, R[39999] = %i\n", R[0], R[39999]);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    return 0;
}
```

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

```
guillesiesta@guillesiesta:~$ gcc -O2 figura1-original.c -o f
guillesiesta@guillesiesta:~$ ./f
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.205815694
guillesiesta@guillesiesta:~$ gcc -O2 figura1-modA.c -o fA -w
guillesiesta@guillesiesta:~$ ./fA
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.157116599
guillesiesta@guillesiesta:~$ gcc -O2 figura1-modB.c -o fB -w
guillesiesta@guillesiesta:~$ ./fB
R[0] = 0, R[39999] = -199995000

Tiempo (seg.) = 0.125431234
guillesiesta@guillesiesta:~$
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.205815694*
Modificación a)	0.157116599
Modificación b)	0.125431234
...	

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Reducir el número de bucles se nota de gran manera, es evidente, pues se reduce a la mitad la ejecución. El desenrollado de bucle aún funciona mejor.

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

Figura1-original.s

```
call            clock_gettime
                leaq
                52(%rsp), %r9
                movl
                $1, %ecx
                movl
                $s+40004, %r8d
                .p2align 4,,10
                .p2align 3
.L2:
                movl
                $s, %eax
                xorl
                %esi, %esi
                .p2align 4,,10
                .p2align 3
.L3:
                movl
                (%rax), %edx
                addq
                $8, %rax
                leal
                (%rcx,%rdx,2),
%edx
                addl
                %edx, %esi
                cmpq
                $s+40000, %rax
                jne
                .L3
                movl
                $s+4, %eax
                xorl
                %edi, %edi
                .p2align 4,,10
                .p2align 3
.L4:
                movl
                (%rax), %edx
                addq
                $8, %rax
                leal
                (%rdx,%rdx,2),
%edx
                subl
                %ecx, %edx
                addl
                %edx, %edi
                cmpq
                %rax, %r8
                jne
                .L4
                cmpl
                %edi, %esi
```

figura1-modificado_a.s

```
call            clock_gettime
                leaq
                52(%rsp), %r9
                movl
                $1, %esi
                movl
                $s+40000, %r8d
                .p2align 4,,10
                .p2align 3
.L2:
                movl
                $s, %eax
                xorl
                %edi, %edi
                xorl
                %ecx, %ecx
                .p2align 4,,10
                .p2align 3
.L3:
                movl
                (%rax), %edx
                addq
                $8, %rax
                leal
                (%rsi,%rdx,2),
%edx
                addl
                %edx, %ecx
                movl
                -4(%rax), %edx
                leal
                (%rdx,%rdx,2),
%edx
                subl
                %esi, %edx
                addl
                %edx, %edi
                cmpq
                %rax, %r8
                jne
                .L3
                cmpl
                %edi, %ecx
                cmovg
                %edi, %ecx
                addl
                $1, %esi
                addq
                $4, %r9
                movl
                %ecx, -4(%r9)
                cmpl
                $40001, %esi
                jne
```

figura1-modificado_b.s

```
call            clock_gettime
                leaq
                52(%rsp), %r10
                movl
                $1, %ecx
                movl
                $s+40000, %r9d
                .p2align 4,,10
                .p2align 3
.L2:
                movl
                $s, %eax
                xorl
                %esi, %esi
                xorl
                %edx, %edx
                .p2align 4,,10
                .p2align 3
.L3:
                movl
                (%rax), %edi
                addq
                $32, %rax
                leal
                (%rcx,%rdi,2),
%edi
                addl
                %edi, %edx
                movl
                -28(%rax), %edi
                leal
                (%rdi,%rdi,2),
%r8d
                movl
                -24(%rax), %edi
                subl
                %ecx, %r8d
                leal
                (%rcx,%rdi,2),
%edi
                addl
                %r8d, %esi
                addl
                %edi, %edx
                movl
                -20(%rax), %edi
                leal
                (%rdi,%rdi,2),
%edi
                subl
                %ecx, %edi
                leal
                (%rdi,%rsi),
%r8d
```

```
                jge
                .L5
                movl
                %esi, (%r9)
.L6:
                addl
                $1, %ecx
                addq
                $4, %r9
                cmpl
                $40001, %ecx
                jne
                .L2
                leaq
                32(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

```
                .L2
                leaq
                32(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

```
                movl
                -16(%rax), %esi
                leal
                (%rcx,%rsi,2),
%esi
                addl
                %esi, %edx
                movl
                -12(%rax), %esi
                leal
                (%rsi,%rsi,2),
%esi
                subl
                %ecx, %esi
                leal
                (%rsi,%r8),
%edi
                movl
                -8(%rax), %esi
                leal
                (%rcx,%rsi,2),
%esi
                addl
                %esi, %edx
                movl
                -4(%rax), %esi
                leal
                (%rsi,%rsi,2),
%esi
                subl
                %ecx, %esi
                addl
                %edi, %esi
                cmpq
                %rax, %r9
                jne
                .L3
                cmpl
                %esi, %edx
                cmovg
                %esi, %edx
                addl
                $1, %ecx
                addq
                $4, %r10
                movl
                %edx, -4(%r10)
                cmpl
                $40001, %ecx
                jne
                .L2
                leaq
                32(%rsp), %rsi
                xorl
                %edi, %edi
                call
                clock_gettime
```

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

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

2.1. Genere los programas en ensamblador para cada una de las siguientes opciones de optimización del compilador: -O0, -Os, -O2, -O3. Explique las diferencias que se observan

en el código justificando al mismo tiempo las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

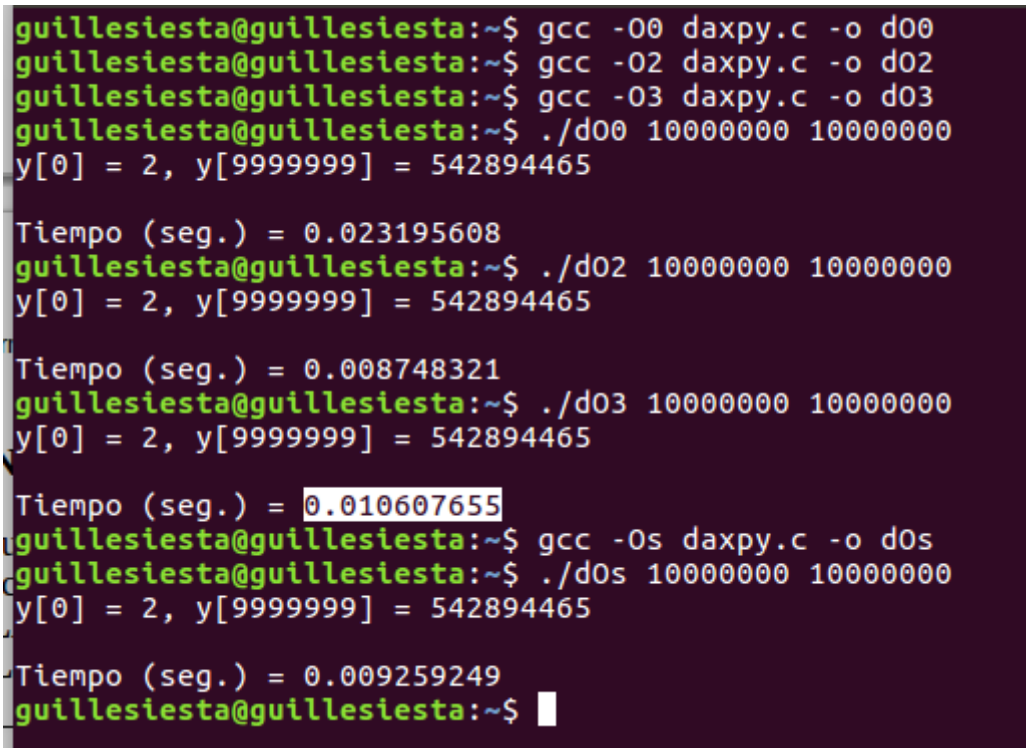
2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo del número de operaciones en coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CAPTURA CÓDIGO FUENTE: `daxpy.c`

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

void daxpy(int *y, int *x, int a, unsigned n, struct timespec *cgt1, struct timespec *cgt2)
{
    clock_gettime(CLOCK_REALTIME,cgt1);
    unsigned i;
    for (i=0; i<n; i++)
        y[i] += a*x[i];
    clock_gettime(CLOCK_REALTIME,cgt2);
}

int main(int argc, char *argv[])
{
    if (argc < 3)
    {
        fprintf(stderr, "ERROR: falta tam del vector y constante\n");
        exit(1);
    }

    unsigned n = strtol(argv[1], NULL, 10);
    int a = strtol(argv[2], NULL, 10);
    int *y, *x;
    y = (int*) malloc(n*sizeof(int));
    x = (int*) malloc(n*sizeof(int));

    unsigned i;
    for (i=0; i<n; i++)
    {
        y[i] = i+2;
        x[i] = i*2;
    }

    struct timespec cgt1,cgt2; double ncgt;


    daxpy(y, x, a, n, &cgt1, &cgt2);

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

    printf("y[0] = %i, y[%i] = %i\n", y[0], n-1, y[n-1]);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    free(y);
    free(x);

    return 0;
}
```

	-O0	-Os	-O2	-O3
Tiempos ejec.	*0.023195 608*	0.00925 9249	0.008748 321	0.010607 655

CAPTURAS DE PANTALLA (que muestren la compilación y que el resultado es correcto):



COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Con O0 usamos direcciones relativas a la pila y con O2, registros de la arquitectura. Así ahorramos muchas operaciones move innecesarias y como se puede ver abajo obtenemos un código mucho más reducido para O2 que para O0, donde estamos moviendo a registros de la arquitectura direcciones relativas de la pila y operando con estas dimensiones

Por último, en el O3, el compilador ha hecho un desenrollado del bucle, dándonos un código más largo, y en este caso más lento que O2.

CÓDIGO EN ENSAMBLADOR (no es necesario introducir aquí el código como captura de pantalla, ajustar el tamaño de la letra para que una instrucción no ocupe más de un renglón)**:**
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpyO0.s	daxpyOs.s	daxpyO2.s	daxpyO3.s

movl $0, -4(%rbp)
jmp .L2
.L3:
movl -4(%rbp), %eax
leaq 0(, %rax,4), %rdx
movq -24(%rbp), %rax
addq %rax,
	xorl %eax, %eax
	.L2:
	cmpl %eax, %ebp
	jbe .L6
	movl (%r12,%rax,4), %edx
	imull %r13d, %edx
	addl %edx,
		xorl %eax, %eax
		testl %ebp, %ebp
		je .L4
		.p2align 4,,10
		.p2align 3
		.L5:
		movl 0(%r13,%rax,4), %esi
		imull %r12d, %esi
		addl
			call clock_gettime
			testl %r14d, %r14d
			je .L10
			leaq 16(%r12), %rax
			cmpq %rax, %rbx
			leaq 16(%rbx),

```
%rdx
        movl    -4(%rbp),
%eax
        leaq    0(,
%rax,4), %rcx
        movq    -
24(%rbp), %rax
        addq    %rcx,
%rax
        movl    (%rax),
%ecx
        movl    -4(%rbp),
%eax
        leaq    0(,
%rax,4), %rsi
        movq    -
32(%rbp), %rax
        addq    %rsi,
%rax
        movl    (%rax),
%eax
        imull   -
36(%rbp), %eax
        addl    %ecx,
%eax
        movl    %eax,
(%rdx)
        addl    $1,
-4(%rbp)
.L2:
        movl    -4(%rbp),
%eax
        cmpl    -
40(%rbp), %eax
        jb      .L3
        movq    -
56(%rbp), %rax
        movq    %rax,
%rsi
        movl    $0, %edi
```

```
(%rbx,%rax,4)
        incq    %rax
        jmp     .L2
.L6:
        addq    $24,
%rsp
        .cfi_def
_cfa_offset 40
        movq    %r9,
%rsi
        xorl    %edi,
%edi
        popq    %rbx
        .cfi_def
_cfa_offset 32
        popq    %rbp
        .cfi_def
_cfa_offset 24
        popq    %r12
        .cfi_def
_cfa_offset 16
        popq    %r13
        .cfi_def
_cfa_offset 8
```

```
        %esi,
(%rbx,%rax,4)
        addq    $1, %rax
        cmpl    %eax, %ebp
        ja      .L5
.L4:
        popq    %rbx
        .cfi_def_c
fa_offset 40
        movq    %r14, %rsi
        xorl    %edi, %edi
        popq    %rbp
        .cfi_def_c
fa_offset 32
        popq    %r12
        .cfi_def_c
fa_offset 24
        popq    %r13
        .cfi_def_c
fa_offset 16
        popq    %r14
        .cfi_def_c
fa_offset 8
        jmp

clock_gettime
```

```
%rax
        setnb   %dl
        cmpq    %rax,
%r12
        setnb   %al
        orb     %al, %dl
        je      .L3
        cmpl    $6, %r14d
        jbe     .L3
        movq    %rbx,
%rax
        andl    $15, %eax
        shrq    $2, %rax
        negq    %rax
        andl    $3, %eax
        cmpl    %r14d,
%eax
        cmova   %r14,
%rax
        xorl    %edx,
%edx
        testl   %eax,
%eax
        je      .L4
        movl    (%r12),
%edx
        imull   %r13d,
%edx
        addl    %edx,
(%rbx)
        cmpl    $1, %eax
        movl    $1, %edx
        je      .L4
        movl    4(%r12),
%edx
        imull   %r13d,
%edx
```

			addl %edx, 4(%rbx)
			cmpl $3, %eax
			movl $2, %edx
			jne .L4
			movl 8(%r12), %edx
			imull %r13d, %edx
			addl %edx, 8(%rbx)
			movl $3, %edx
			.L4:
			movl %r14d, %edi
			movl %r13d, 12(%rsp)
			xorl %ecx, %ecx
			subl %eax, %edi
			movd 12(%rsp), %xmm4
			salq $2, %rax
			leal -4(%rdi), %esi
			leaq (%rbx, %rax), %r10
			xorl %r9d, %r9d
			pshufd $0, %xmm4, %xmm2
			addq %r12, %rax
			shrl $2, %esi
			addl $1, %esi
			movdqa %xmm2, %xmm3
			leal 0(,

			%rsi,4), %r8d psrlq $32, %xmm3 .L6: movdqu (%rax, %rcx), %xmm0 addl $1, %r9d movdqa %xmm0, %xmm1 psrlq $32, %xmm0 pmuludq %xmm3, %xmm0 pshufd $8, %xmm0, %xmm0 pmuludq %xmm2, %xmm1 pshufd $8, %xmm1, %xmm1 punpckldq %xmm0, %xmm1 movdqa (%r10,%rcx), %xmm0 paddd %xmm1, %xmm0 movaps %xmm0, (%r10,%rcx) addq $16, %rcx cmpl %esi, %r9d jb .L6 addl %r8d, %edx cmpl %r8d, %edi je .L10 movl %edx, %eax movl (%r12,%rax,4), %ecx imull

			%r13d, %ecx
			addl %ecx, (%rbx,%rax,4)
			leal 1(%rdx), %eax
			cmpl %eax, %r14d
			jbe .L10
			movl (%r12,%rax,4), %ecx
			addl $2, %edx
			imull %r13d, %ecx
			addl %ecx, (%rbx,%rax,4)
			cmpl %edx, %r14d
			jbe .L10
			movl %edx, %eax
			imull (%r12,%rax,4), %r13d
			addl %r13d, (%rbx,%rax,4)
			.L10:
			addq $16, %rsp
			.cfi_remember_state
			.cfi_def_cfa_offset 48
			movq %rbp, %rsi
			xorl %edi, %edi
			popq %rbx
			.cfi_def_cfa_offset 40
			popq %rbp
			.cfi_def_cfa_offset 32
			popq %r12
			.cfi_def_

			cfa_offset 24 popq %r13 .cfi_def_ cfa_offset 16 popq %r14 .cfi_def_ cfa_offset 8 jmp clock_gettime