

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): César Muñoz Reinoso

Grupo de prácticas: Grupo 2

Fecha de entrega: 30/05

Fecha evaluación en clase:

**Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo):** Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz

**Sistema operativo utilizado:** Ubuntu 18.04

**Versión de gcc utilizada:** gcc 7.3.0

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

```
[César Muñoz Reinoso cesar@cesar-TM1701:~] 2018-05-25 viernes
$ lscpu
Architectura: x86_64
modo(s) de operación de las CPUs: 32-bit, 64-bit
Orden de los bytes: Little Endian
CPU(s): 8
Lista de la(s) CPU(s) en línea: 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: 142
Nombre del modelo: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
Revisión: 10
CPU MHz: 900,066
CPU MHz máx.: 4000,0000
CPU MHz mín.: 400,0000
BogoMIPS: 3984.00
Virtualización: VT-x
Cache L1d: 32K
Cache L1i: 32K
Cache L2: 256K
Cache L3: 8192K
CPU(s) del nodo NUMA 0: 0-7
Indicadores: 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 sy
scall nx pdpeibg rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 moni
tor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefe
tch cpuid_fault epb invpcid_single pti tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx snap clflushop
t intel_pt 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 incorpórellos 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

```

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

#define N 512
int matrix1[N][N],matrix2[N][N],salida[N][N];

int main(int argc, char** argv){
    int suma=0;
    struct timespec cgt1, cgt2; double ncgt;

    //INICIALIZAMOS MATRIZ Y VECTOR
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            matrix1[i][j]=i+j;
            matrix2[i][j]=i+j;
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    //MULTIPLICACIÓN
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            for(int z = 0; z < N; z++){
                salida[i][j] += matrix1[i][z]*matrix2[z][j];
            }
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    printf("\nTiempo(seg.):%11.9f\n Tamaño:%u\n salida[0][0]=%d / / salida[%d][%d]=%d \n",
ncgt,N,salida[0][0],N-1,N-1,salida[N-1][N-1]);

    return 0;
}

```

**1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):****Modificación a) –explicación–:** Desenrollado de bucle**Modificación b) –explicación–:** Transposición de la segunda matriz para acceder a ella de una forma secuencial en memoria.

## 1.1. CÓDIGOS FUENTE MODIFICACIONES

### a) Captura de pmm-secuencial-modificado\_a.c

```
int main(int argc, char** argv){
    int suma=0;
    struct timespec cgt1, cgt2; double ncgt;

    //INICIALIZAMOS MATRIZ Y VECTOR
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            matrix1[i][j]=i+j;
            matrix2[i][j]=i+j;
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    //MULTIPLICACIÓN
    int tmp0=0,tmp1=0,tmp2=0,tmp3=0;

    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            for(int z = 0; z < N; z+=4){
                tmp0 += matrix1[i][z]*matrix2[z][j];
                tmp1 += matrix1[i][z+1]*matrix2[z+1][j];
                tmp2 += matrix1[i][z+2]*matrix2[z+2][j];
                tmp3 += matrix1[i][z+3]*matrix2[z+3][j];
            }
            salida[i][j]=tmp0+tmp1+tmp2+tmp3;
            tmp0=tmp1=tmp2=tmp3=0;
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

```
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$gcc -O2 pmm-secuencial-modificado_a.c -o pmm-secuencial-modificado_a
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$./pmm-secuencial-modificado_a

Tiempo(seg.):0.212987135
Tamaño:512
salida[0][0]=44608256 // salida[511][511]=311996160
```

### b) Captura de pmm-secuencial-modificado\_b.c

```

int main(int argc, char** argv){
    int suma=0;
    struct timespec cgt1, cgt2; double ncgt;

    //INICIALIZAMOS MATRIZ Y VECTOR
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            matrix1[i][j]=i+j;
            matrix2[i][j]=i+j;
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    //TRASPUESTA
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            int aux=matrix2[i][j];
            matrix2[i][j]=matrix2[j][i];
            matrix2[j][i]=aux;
        }
    }

    //MULTIPLICACIÓN
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            for(int z = 0; z < N; z++){
                salida[i][j] += matrix1[i][z]*matrix2[j][z];
            }
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

```

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

```

César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$gcc -O2 pmm-secuencial-modificado_b.c -o pmm-secuencial-modificado_b
César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$./pmm-secuencial-modificado_b

tiempo(seg.):0.070131735
Tamaño:512
salida[0][0]=44608256 / / salida[511][511]=311996160

```

### 1.1. TIEMPOS:

Modificación	-O2
Sin modificar	0.236325213
Modificación a)	0.212987135
Modificación b)	0.070131735

### 1.1. COMENTARIOS SOBRE LOS RESULTADOS:

En cuanto a la ejecución de los diferentes programas, podemos observar que la primera modificación de desenrollado de bucles tan solo reduce el tiempo inicial en 0,02 segundos. En cambio, la segunda modificación de transposición de la segunda matriz es mucho más efectiva, reduciendo en 0,16 segundos la ejecución.

## 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	pmm-secuencial-modificado_a.s	pmm-secuencial-modificado_b.s
<pre> .L3:     leal (%rcx,%rax), %edx     movl %edx, (%rsi,%rax,4)     movl %edx, (%rdi,%rax,4)     addq \$1, %rax     cmpq \$512, %rax     jne .L3     addl \$1, %ecx     addq \$2048, %rsi     addq \$2048, %rdi     cmpl \$512, %ecx     jne .L2     movq %rsp, %rsi     xorl %edi, %edi     call clock_gettime@PLT     leaq salida(%rip), %r8     leaq 1048576*salida(%rip), %r10     leaq 1048576(%rbp), %r11     leaq 1050624(%rbp), %r9     subq %rbp, %r8     subq %rbp, %r10  .L5:     movq %r11, %rdi     .p2align 4,,10     .p2align 3  .L9:     movl -1048576(%r8,%rdi), %esi     leaq -1048576(%rdi), %rax     movq %rbx, %rcx     .p2align 4,,10     .p2align 3  .L6:     movl (%rcx), %edx     addq \$2048, %rax     addq \$4, %rcx     imull -2048(%rax), %edx     addl %edx, %esi     cmpq %rax, %rdi     jne .L6     movl %esi, -1048576(%r8,%rdi)     addq \$4, %rdi     cmpq %r9, %rdi     jne .L9     addq \$2048, %r8     addq \$2048, %rbx     cmpq %r10, %r8     jne .L5     leaq 16(%rsp), %rsi     xorl %edi, %edi     call clock_gettime@PLT </pre>	<pre> .L3:     leal (%rcx,%rax), %edx     movl %edx, (%rsi,%rax,4)     movl %edx, (%rdi,%rax,4)     addq \$1, %rax     cmpq \$512, %rax     jne .L3     addl \$1, %ecx     addq \$2048, %rsi     addq \$2048, %rdi     cmpl \$512, %ecx     jne .L2     movq %rsp, %rsi     xorl %edi, %edi     leaq 1048576(%rbp), %r12     call clock_gettime@PLT     leaq 1048576*salida(%rip), %rax     leaq salida(%rip), %r11     subq %rbp, %rax     subq %rbp, %r11     addq \$1050624, %rbp     movq %rax, %r13  .L5:     movq %r12, %r10     .p2align 4,,10     .p2align 3  .L9:     leaq -1048576(%r10), %rax     movq %rbx, %rdx     xorl %r9d, %r9d     xorl %r8d, %r8d     xorl %edi, %edi     xorl %esi, %esi     .p2align 4,,10     .p2align 3  .L6:     movl (%rdx), %ecx     addq \$8192, %rax     addq \$16, %rdx     imull -8192(%rax), %ecx     addl %ecx, %esi     movl 12(%rdx), %ecx     imull -6144(%rax), %ecx     addl %ecx, %edi     movl -8(%rdx), %ecx     imull -4096(%rax), %ecx     addl %ecx, %r8d     movl -4(%rdx), %ecx     imull -2048(%rax), %ecx     addl %ecx, %r9d     cmpq %rax, %r10     jne .L6     addl %edi, %esi     addl %esi, %r8d     addl %r8d, %r9d     movl %r9d, -1048576(%r11,%r10)     addq \$4, %r10     cmpq %r10, %rbp     jne .L9     addq \$2048, %r11     addq \$2048, %rbx     cmpq %r11, %r13     jne .L5     leaq 16(%rsp), %rsi     xorl %edi, %edi     call clock_gettime@PLT </pre>	<pre> .L3:     leal (%rcx,%rax), %edx     movl %edx, (%rsi,%rax,4)     movl %edx, (%rdi,%rax,4)     addq \$1, %rax     cmpq \$512, %rax     jne .L3     addl \$1, %ecx     addq \$2048, %rsi     addq \$2048, %rdi     cmpl \$512, %ecx     jne .L2     movq %rsp, %rsi     xorl %edi, %edi     call clock_gettime@PLT     leaq 2048*matrix2(%rip), %r11     leaq 1048576(%rbp), %r8     leaq -2048(%r11), %r9     movq %r9, %r10  .L5:     leaq 1048576(%r9), %rdi     movq %r9, %rax     movq %r10, %rdx     .p2align 4,,10     .p2align 3  .L6:     movl (%rdx), %ecx     movl %rax, %esi     addq \$2048, %rax     addq \$4, %rdx     movl %esi, -4(%rdx)     movl %ecx, -2048(%rax)     cmpq %rdi, %rax     jne .L6     addq \$4, %r9     addq \$2048, %r10     cmpq %r11, %r9     jne .L5     leaq salida(%rip), %r9     leaq 1048576*matrix1(%rip), %r10  .L7:     movq %rbp, %rs     movq %r9, %rdi     .p2align 4,,10     .p2align 3  .L11:     movl (%rdi), %ecx     xorl %eax, %eax     .p2align 4,,10     .p2align 3  .L8:     movl (%rbx,%rax), %edx     imull (%rsi,%rax), %edx     addq \$4, %rax     addl %edx, %ecx     cmpq \$2048, %rax     jne .L8     addq \$2048, %rsi     movl %ecx, (%rdi)     addq \$4, %rdi     cmpq %r8, %rsi     jne .L11     addq \$2048, %rbx     addq \$2048, %r9     cmpq %r10, %rbx     jne .L7     leaq 16(%rsp), %rsi     xorl %edi, %edi     call clock_gettime@PLT </pre>

**B) CÓDIGO FIGURA 1:****CAPTURA CÓDIGO FUENTE:** figura1-original.c

```

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

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

int R[40000];

int main(int argc, char ** argv){
    int ii,i,X1,X2;
    struct timespec cgt1, cgt2; double ncgt;

    for(int i=0;i<5000;i++){
        s[i].a=i;
        s[i].b=i+1;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    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;
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

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

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

    return 0;
}

```

**1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):**

**Modificación a) –explicación–:** Unimos los dos bucles ya que los datos se almacenan secuencialmente, teniendo en cada iteración un valor para a y otro para b unidos en memoria.

**Modificación b) –explicación–:** Desenrollamos los bucles para romper la dependencia de datos en cada iteración y así poder unificar el procesamiento de datos, reduciendo el número de iteraciones. También va incluida la modificación a.

**1.1. CÓDIGOS FUENTE MODIFICACIONES****a) Captura figura1-modificado\_a.c**

```

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

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

int R[40000];

int main(int argc, char ** argv){
    int ii,i,X1,X2;
    struct timespec cgt1, cgt2; double ncgt;

    for(int i=0;i<5000;i++){
        s[i].a=i;
        s[i].b=i+1;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(ii=0;ii<40000;ii++){

        X1=0;X2=0;

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

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

    clock_gettime(CLOCK_REALTIME,&cgt2);

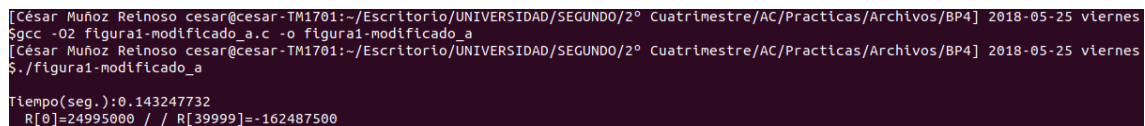
    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

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

    return 0;
}

```

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



```

[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$gcc -O2 figura1-modificado_a.c -o figura1-modificado_a
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$./figura1-modificado_a

Tiempo(seg.):0.143247732
  R[0]=24995000 / /  R[39999]=-162487500

```

**b) Captura figura1-modificado\_b.c**

```

int main(int argc, char ** argv){
    int ii,i,X1,X2,tmp0,tmp1,tmp2,tmp3;
    struct timespec cgt1, cgt2; double ncgt;

    for(int i=0;i<5000;i++){
        s[i].a=i;
        s[i].b=i+1;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(ii=0;ii<40000;ii++){

        X1=0;X2=0,tmp0=0,tmp1=0,tmp2=0,tmp3=0;

        for(i=0;i<5000;i+=2){
            tmp0+=2*s[i].a+ii;
            tmp1+=2*s[i+1].a+ii;
            tmp2+=3*s[i].b-ii;
            tmp3+=3*s[i+1].b-ii;
        }

        X1=tmp0+tmp1;
        X2=tmp2+tmp3;

        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));

```

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

```

[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$gcc -O2 figura1-modificado_b.c -o figura1-modificado_b
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$./figura1-modificado_b

Tiempo(seg.):0.122902374
R[0]=24995000 / / R[39999]=-162487500

```

### 1.1. TIEMPOS:

Modificación	-O2
Sin modificar	0.213954227
Modificación a)	0.143247732
Modificación b)	0.122902374

### 1.1. COMENTARIOS SOBRE LOS RESULTADOS:

En este caso tenemos un tiempo de referencia un poco menor. Con la primera modificación, unión de dos simples bucles, conseguimos reducir el tiempo de ejecución 0,7 segundos. Con la segunda modificación, desenrollando los bucles, conseguimos reducir el tiempo 0,9 segundos desde el inicial.

### 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	figura1-modificado_a.s	figura1-modificado_b.s
<pre> .L2:     movl %eax, (%rdx)     addl \$1, %eax     addq \$8, %rdx     movl %eax, -4(%rdx)     cmpl \$5000, %eax     jne .L2     movq %rsp, %rsi     xorl %edi, %edi     call clock_gettime@PLT     leaq 40004(%rbx), %r9     leaq R(%rip), %r11     leaq 40000(%rbx), %r8     xorl %r10d, %r10d </pre>	<pre> .L2:     movl %eax, (%rdx)     addl \$1, %eax     addq \$8, %rdx     movl %eax, -4(%rdx)     cmpl \$5000, %eax     jne .L2     movq %rsp, %rsi     xorl %edi, %edi     call clock_gettime@PLT     leaq R(%rip), %r10     leaq 40000(%rbx), %r8     xorl %r9d, %r9d     .p2align 4,,10 </pre>	<pre> .L2:     movl %eax, (%rdx)     addl \$1, %eax     addq \$8, %rdx     movl %eax, -4(%rdx)     cmpl \$5000, %eax     jne .L2     movq %rsp, %rsi     xorl %edi, %edi     leaq R(%rip), %rbp     call clock_gettime@PLT     leaq 40000(%rbx), %r10     xorl %r11d, %r11d     .p2align 4,,10 </pre>



<pre> .L3:      .p2align 4,,10           .p2align 3           movl %r10d, %edi           movq %rbx, %rax           xorl %ecx, %ecx           .p2align 4,,10           .p2align 3  .L4:      movl (%rax), %edx           addq \$8, %rax           leal (%rdi,%rdx,2), %edx           addl %edx, %ecx           cmpq %r8, %rax           jne .L4           leaq 4+*(%rip), %rax           xorl %esi, %esi           .p2align 4,,10           .p2align 3  .L5:      movl (%rax), %edx           addq \$8, %rax           leal (%rdx,%rdx,2), %edx           subl %edi, %edx           addl %edi, %esi           cmpq %r9, %rax           jne .L5           cmpl %ecx, %esi           jle .L6           movl %ecx, (%r11,%r10,4)  .L7:      addq \$1, %r10           cmpq \$40000, %r10           jne .L3           leaq 16(%rsp), %rsi           xorl %edi, %edi           call clock_gettime@PLT           movq 24(%rsp), %rax           ...  .L6:      .cfi_restore_state           movl %esi, (%r11,%r10,4)           jmp .L7 </pre>	<pre> .L3:      .p2align 3           movl %r9d, %edi           movq %rbx, %rax           xorl %esi, %esi           xorl %ecx, %ecx           .p2align 4,,10           .p2align 3  .L4:      movl (%rax), %edx           addq \$8, %rax           leal (%rdi,%rdx,2), %edx           addl %edx, %ecx           movl -4(%rax), %edx           leal (%rdx,%rdx,2), %edx           subl %edi, %edx           addl %edx, %esi           cmpq %rax, %r8           jne .L4           cmpl %esi, %ecx           jge .L5           movl %ecx, (%r10,%r9,4)  .L6:      addq \$1, %r9           cmpq \$40000, %r9           jne .L3           leaq 16(%rsp), %rsi           xorl %edi, %edi           call clock_gettime@PLT           ...  .L5:      .cfi_restore_state           movl %esi, (%r10,%r9,4)           jmp .L6 </pre>	<pre> .L3:      .p2align 3           movl %r11d, %edx           movq %rbx, %rax           xorl %r9d, %r9d           xorl %edi, %edi           xorl %r8d, %r8d           .p2align 4,,10           .p2align 3  .L4:      movl (%rax), %ecx           addq \$16, %rax           leal (%rdx,%rcx,2), %ecx           addl %ecx, %esi           movl -8(%rax), %ecx           leal (%rdx,%rcx,2), %ecx           addl %ecx, %r8d           movl -12(%rax), %ecx           leal (%rcx,%rcx,2), %ecx           subl %edx, %ecx           addl %ecx, %edi           movl -4(%rax), %ecx           leal (%rcx,%rcx,2), %ecx           subl %edx, %ecx           addl %ecx, %r8d           cmpq %rax, %r10           jne .L4           addl %r8d, %esi           addl %r9d, %edi           cmpl %edi, %esi           jge .L5           movl %esi, 0(%rbp,%r11,4)           ...  .L6:      addq \$1, %r11           cmpq \$40000, %r11           jne .L3           leaq 16(%rsp), %rsi           xorl %edi, %edi           call clock_gettime@PLT           ... </pre>
--	--	--

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

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

2.1. Genere los programas en ensamblador para cada una de las 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 acarrearán. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

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

**CAPTURA CÓDIGO FUENTE:** daxpy.c

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

int main(int argc, char ** argv){
    int N=520000, a=1000;
    struct timespec cgt1, cgt2; double ncgt;
    double y[N];
    double x[N];
    for(int i=0; i<N;i++){
        y[i]=i+1;
        x[i]=i+2;
    }
    clock_gettime(CLOCK_REALTIME,&cgt1);
    for (int i=0;i<N;i++) y[i]= a*x[i] + y[i];
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    printf("\nTamano:%d \nTiempo(seg.):%11.9f\n  y[0]=%f / / y[%d]=%f \n", N,ncgt,y[0],N,y[N-1]);

    return 0;
}
```

	-O0	-Os	-O2	-O3
<b>Tiempos ejec.</b>	0.002784 702	0.00057 0547	0.000706 274	0.000633 818

Tamaño: 520000

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

```
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$gcc -O0 daxpy.c -o daxpy
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$./daxpy

Tamano:520000
Tiempo(seg.):0.002784702
  y[0]=2001.000000 / / y[520000]=520521000.000000
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$gcc -Os daxpy.c -o daxpy
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$./daxpy

Tamano:520000
Tiempo(seg.):0.000570547
  y[0]=2001.000000 / / y[520000]=520521000.000000
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$gcc -O2 daxpy.c -o daxpy
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$./daxpy

Tamano:520000
Tiempo(seg.):0.000706274
  y[0]=2001.000000 / / y[520000]=520521000.000000
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$gcc -O3 daxpy.c -o daxpy
[César Muñoz Reinoso cesar@cesar-TM1701:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP4] 2018-05-25 viernes
$./daxpy

Tamano:520000
Tiempo(seg.):0.000633818
  y[0]=2001.000000 / / y[520000]=520521000.000000
```

**COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:**

Vemos que la mayor eficiencia del código no se obtiene con la optimización -O3 como sería lo normal, si no que se obtiene con la optimización -Os, ya que se obtiene un código ensamblador con menos instrucciones y por tanto de menor duración.

En daxpyOs.s vemos que el código es mucho menor que el original, donde tenemos un código anidado a una función, en el optimizado vemos que se resume en apenas 10 instrucciones.

Por otro lado vemos que el código daxpyO2.s es prácticamente igual que la optimización Os por eso los tiempos son tan similares.

Finalmente podemos ver que la optimización O3 es la menos eficiente, ya que no está definitivamente preparado para alcanzar ese nivel de optimización.

**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)**

daxpy00.s	daxpy0s.s
<pre> .L2:     movl -140(%rbp), %eax     cmpl -148(%rbp), %eax     jl .L3     leaq -96(%rbp), %rax     movq %rax, %rsi     movl \$0, %edi     call clock_gettime@PLT     movl \$0, -144(%rbp)     jmp .L4  .L5:     cvtsi2sd -152(%rbp), %xmm0     movq -112(%rbp), %rax     movl -144(%rbp), %edx     movslq %edx, %rdx     movsd (%rax,%rdx,8), %xmm1     mulsd %xmm1, %xmm0     movq -128(%rbp), %rax     movl -144(%rbp), %edx     movslq %edx, %rdx     movsd (%rax,%rdx,8), %xmm1     addsd %xmm1, %xmm0     movq -128(%rbp), %rax     movl -144(%rbp), %edx     movslq %edx, %rdx     movsd %xmm0, (%rax,%rdx,8)     addl \$1, -144(%rbp)  .L4:     movl -144(%rbp), %eax     cmpl -148(%rbp), %eax     jl .L5     leaq -80(%rbp), %rax     movq %rax, %rsi     movl \$0, %edi     call clock_gettime@PLT     movq -80(%rbp), %rdx     movq -96(%rbp), %rax     subq %rax, %rdx     movq %rdx, %rax     cvtsi2sd %rax, %xmm1     movq -72(%rbp), %rdx     movq -88(%rbp), %rax     subq %rax, %rdx     movq %rdx, %rax     cvtsi2sd %rax, %xmm0     movsd .LC0(%rip), %xmm2     divsd %xmm2, %xmm0     addsd %xmm1, %xmm0     movsd %xmm0, -194(%rbp)     movl -148(%rbp), %eax     leal -1(%rax), %edx     movq -128(%rbp), %rax     movslq %edx, %rdx     movsd (%rax,%rdx,8), %xmm1     movq -128(%rbp), %rax     movsd (%rax), %xmm0     movl -148(%rbp), %edx     movq -104(%rbp), %rcx     movl -148(%rbp), %eax     movapd %xmm1, %xmm2     movapd %xmm0, %xmm1     movq %rcx, -184(%rbp)     movsd -184(%rbp), %xmm0     movl %eax, %esi     leaq .LC1(%rip), %rdi     movl \$3, %eax     call printf@PLT     movl \$0, %eax     movq %rbx, %rsp     movq -56(%rbp), %rcx     xorq %fs:40, %rcx     je .L7     call __stack_chk_fail@PLT </pre>	<pre> .L2:     cvtsi2sd %eax, %xmm0     leal 1(%rax), %edx     movsd %xmm0, -8(%r12,%rax,8)     cvtsi2sd %edx, %xmm0     movsd %xmm0, -8(%r13,%rax,8)     incq %rax     cmpq \$520001, %rax     jne .L2     leaq -72(%rbp), %rsi     xorl %edi, %edi     call clock_gettime@PLT     movsd .LC0(%rip), %xmm1     xorl %eax, %eax  .L3:     movsd 0(%r13,%rax), %xmm0     mulsd %xmm1, %xmm0     addsd (%r12,%rax), %xmm0     movsd %xmm0, (%r12,%rax)     addq \$8, %rax     cmpq \$4160000, %rax     jne .L3     leaq -96(%rbp), %rsi     xorl %edi, %edi     call clock_gettime@PLT     movq -48(%rbp), %rax     subq -64(%rbp), %rax     leaq .LC2(%rip), %rsi     movsd 4159992(%rbx,8), %xmm2     movl \$520000, %ecx     movl \$520000, %edx     movl \$1, %edi     cvtsi2sdq %rax, %xmm0     movq -56(%rbp), %rax     subq -72(%rbp), %rax     cvtsi2sdq %rax, %xmm1     movb \$3, %al     divsd .LC1(%rip), %xmm0     addsd %xmm1, %xmm0     movsd 0(%rbx,8), %xmm1     call __printf_chk@PLT     xorl %eax, %eax     movq -40(%rbp), %rcx     xorq %fs:40, %rcx     je .L4     call __stack_chk_fail@PLT </pre>
daxpy02.s	daxpy03.s
<pre> .L2:     pxor %xmm0, %xmm0     leal 1(%rax), %edx     cvtsi2sd %eax, %xmm0     movsd %xmm0, (%rbx,%rax,8)     pxor %xmm0, %xmm0     cvtsi2sd %edx, %xmm0     movsd %xmm0, -8(%r13,%rax,8)     addq \$1, %rax     cmpq \$520001, %rax     jne .L2     leaq -80(%rbp), %rsi     xorl %edi, %edi     call clock_gettime@PLT     movsd .LC0(%rip), %xmm1     xorl %eax, %eax     repzalln 4, 10     repzalln 8  .L3:     movsd 0(%r13,%rax), %xmm0     mulsd %xmm1, %xmm0     addsd (%rbx,%rax), %xmm0 </pre>	<pre> .L4:     leaq -96(%rbp), %rsi     xorl %edi, %edi     movq %r8, -112(%rbp)     movl %r9d, -100(%rbp)     movq %rcx, -120(%rbp)     call clock_gettime@PLT     movl -100(%rbp), %r9d     xorl %eax, %eax     movq -112(%rbp), %r8     testl %r9d, %r9d     je .L5     movq -120(%rbp), %rcx     movsd .LC0(%rip), %xmm0     movl \$1, %eax     mulsd 0(%rcx,8), %xmm0     addsd 0(%r12,8), %xmm0     movsd %xmm0, 0(%r12,8)  .L5:     movl %r15d, %esi     movapd .LC7(%rip), %xmm1     shril %esi </pre>

```

movsd %xmm0, (%rbx,%rax)
addq $8, %rax
cmpq $4159992, %rax
jne .L9
leaq -64(%rbp), %rsi
xorl %edi, %edi
call clock_gettime@PLT
movq -56(%rbp), %rax
subq -72(%rbp), %rax
leaq .LC2(%rip), %rsi
pxor %xmm0, %xmm0
movl $520000, %ecx
pxor %xmm1, %xmm1
movl $520000, %edx
movsd 4159992(, %r12, 8), %xmm2
movl $1, %edi
cvtis2sdq %rax, %xmm0
movq -64(%rbp), %rax
subq -80(%rbp), %rax
cvtis2sdq %rax, %xmm1
movl $3, %eax
divsd .LC1(%rip), %xmm0
addsd %xmm1, %xmm0
movsd 0(, %r12, 8), %xmm1
call printf_chk@PLT
xorl %eax, %eax
movq -40(%rbp), %rcx
xorq %fs:40, %rcx
jne .L9
leaq -24(%rbp), %rsp
popq %rbx
popq %r12
popq %r13
popq %rbp
.cfi_remember_state
.cfi_def_cfa 7, 8
ret

```

.L6:

```

xorl %edx, %edx
xorl %ecx, %ecx
p2align 4,,10
p2align 3
movupd (%r8,%rdx), %xmm0
addl $1, %ecx
mulpd %xmm1, %xmm0
addpd (%rbx,%rdx), %xmm0
movaps %xmm0, (%rbx,%rdx)
addq $16, %rdx
cmpl %esi, %ecx
jb .L6
movl %r15d, %edx
andl $-2, %edx
addl %edx, %eax
cmpl %r15d, %edx
je .L7
movsd .LC6(%rip), %xmm0
cvt
leaq (%r14,%rax,8), %rdx
mulsd 0(%r13,%rax,8), %xmm0
addsd (%rdx), %xmm0
movsd %xmm0, (%rdx)

```

.L7

```

leaq -80(%rbp), %rsi
xorl %edi, %edi
call clock_gettime@PLT
movq -72(%rbp), %rax
subq -80(%rbp), %rax
leaq .LC9(%rip), %rsi
pxor %xmm0, %xmm0
movl $1, %edi
pxor %xmm1, %xmm1
movl $520000, %ecx
movsd 4159992(, %r12, 8), %xmm2
movl $520000, %edx
cvtis2sdq %rax, %xmm0
movq -80(%rbp), %rax
subq -96(%rbp), %rax
cvtis2sdq %rax, %xmm1
movl $3, %eax
divsd .LC8(%rip), %xmm0
addsd %xmm1, %xmm0
movsd 0(, %r12, 8), %xmm1
call printf_chk@PLT
xorl %eax, %eax
movq -56(%rbp), %rdi
xorq %fs:40, %rdi
jne .L22
leaq -40(%rbp), %rsp
popq %rbx
popq %r12
popq %r13
popq %r14
popq %r15
popq %rbp
.cfi_remember_state
.cfi_def_cfa 7, 8
ret

```