

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): Javier Gómez Luzón

Grupo de prácticas: C1

Fecha de entrega:

Fecha evaluación en clase:

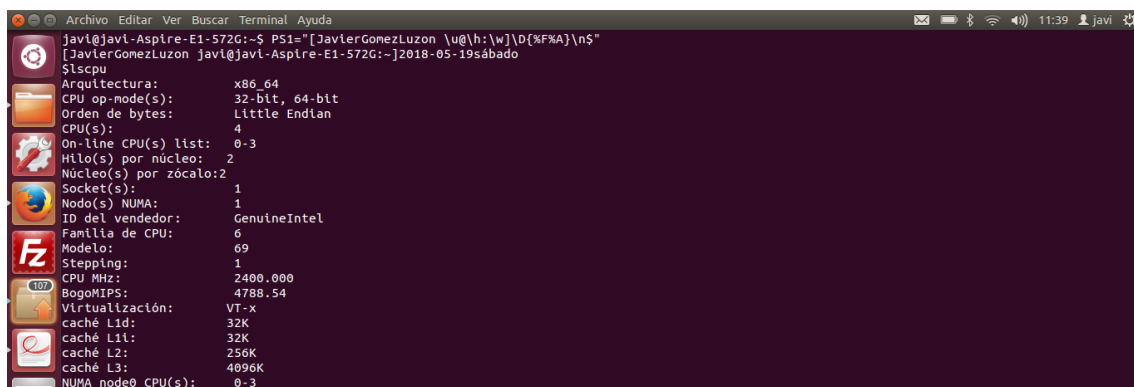
Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo):  
Intel® Core™ i7-4500U CPU @ 1.80GHz

Sistema operativo utilizado: Ubuntu 12.04.5 LTS

Versión de gcc utilizada: 4.6.3

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

Imagen 1: Ejecución de lscpu en la maquina local.



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;

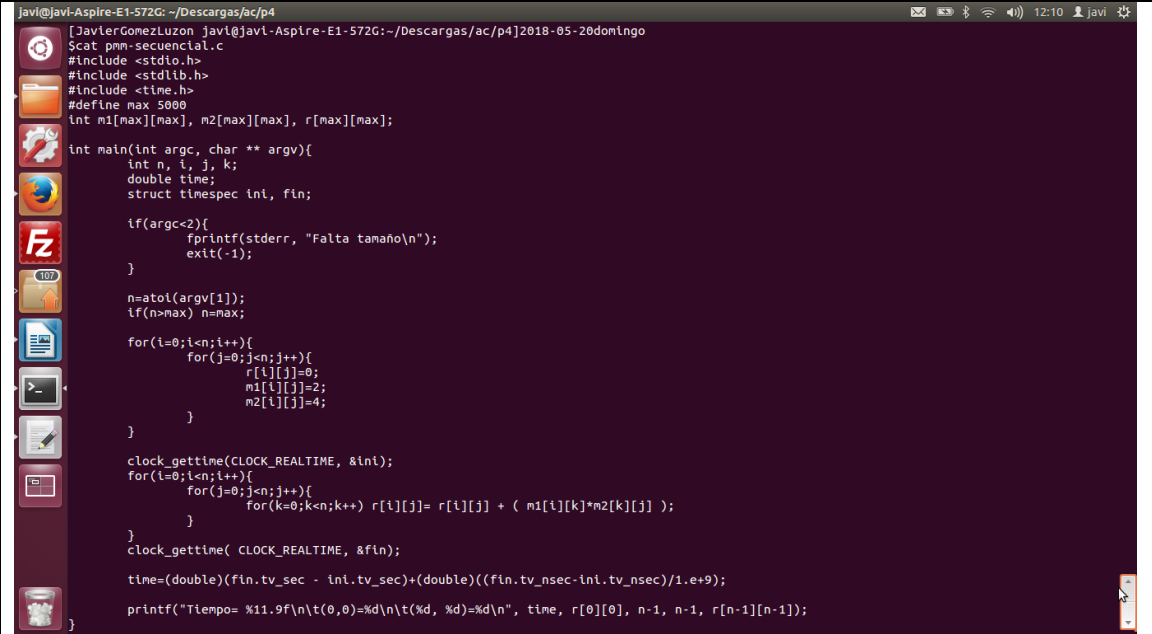
        if (X1<X2) R[ii]=X1 else R[ii]=X2;
    }
    ...
}

```

## A) MULTIPLICACIÓN DE MATRICES:

**CAPTURA CÓDIGO FUENTE:** pmm-secuencial.c

Imagen 2: Cat pmm-secuencial.c.



```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p4
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
Scat pmm-secuencial.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define max 5000
int m1[max][max], m2[max][max], r[max][max];

int main(int argc, char ** argv){
    int n, i, j, k;
    double time;
    struct timespec ini, fin;

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

    n=atoi(argv[1]);
    if(n>max) n=max;

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            r[i][j]=0;
            m1[i][j]=2;
            m2[i][j]=4;
        }
    }

    clock_gettime(CLOCK_REALTIME, &ini);
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            for(k=0;k<n;k++){
                r[i][j]= r[i][j] + ( m1[i][k]*m2[k][j] );
            }
        }
    }
    clock_gettime( CLOCK_REALTIME, &fin);

    time=(double)(fin.tv_sec - ini.tv_sec)+(double)((fin.tv_nsec-ini.tv_nsec)/1.e+9);
    printf("Tiempo= %11.9f\n\t(0,0)=%d\n\t(%d, %d)=%d\n", time, r[0][0], n-1, n-1, r[n-1][n-1]);
}

```

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

**Modificación a) –explicación–:** Calculo de la matriz traspuesta antes de realizar la multiplicación.

**Modificación b) –explicación–:** Reducir el número de iteraciones del bucle más interno.

### 1.1. CÓDIGOS FUENTE MODIFICACIONES

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

Imagen 3 y 4: Cat pmm-secuencial-modificado\_a.c.

```
javi@javi-Aspire-E1-572G: ~
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-21lunes
Scat pmm-secuencial-modificado_a.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define max 5000
int m1[max][max], m2[max][max], r[max][max], m2T[max][max];

int main(int argc, char ** argv){
    int n, i, j, k;
    double time;
    struct timespec tnt, fin;

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

    n=atoi(argv[1]);
    if(n>max) n=max;

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            r[i][j]=0;
            m1[i][j]=2;
            m2[i][j]=4;
        }
    }

    clock_gettime(CLOCK_REALTIME, &tnt);
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            m2T[i][j]=m2[j][i];
        }
    }

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            for(k=0;k<n;k++){ r[i][j]= r[i][j] + ( m1[i][k]*m2T[i][k] );
        }
    }

    clock_gettime( CLOCK_REALTIME, &fin);

    time=(double)(fin.tv_sec - tnt.tv_sec)+(double)((fin.tv_nsec-tnt.tv_nsec)/1.e+9);
    printf("Tiempo= %11.9f\n\t(0,0)=%d\n\t(%d, %d)=%d\n", time, r[0][0], n-1, n-1, r[n-1][n-1]);
}
```

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

Imagen 5: Compilacion de pmm-secuencial-modificado\_a.c.

```
javi@javi-Aspire-E1-572G: ~
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-21lunes
$gcc -O2 -o pmm-secuencial-modificado_a pmm-secuencial-modificado_a.c -lrt
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-21lunes
$gcc -O2 -S pmm-secuencial-modificado_a.c -lrt
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-21lunes
```

Las capturas de pantalla de la ejecución se mostrarán más adelante con las ejecuciones de las demás versiones de este programa (en el apartado de tiempos).

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

Imagen 6 y 7: Cat pmm-secuencial-modificado\_a.c.

```
javi@javi-Aspire-E1-572G: ~
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-21lunes
Scat pmm-secuencial-modificado_b.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define max 5000
int m1[max][max], m2[max][max], r[max][max], m2T[max][max];

int main(int argc, char ** argv){
    int n, i, j, k, total;
    double time;
    struct timespec tnt, fin;
    int r1, r2, r3, r4;
    if(argc<2){
        fprintf(stderr, "Falta tamaño\n");
        exit(-1);
    }

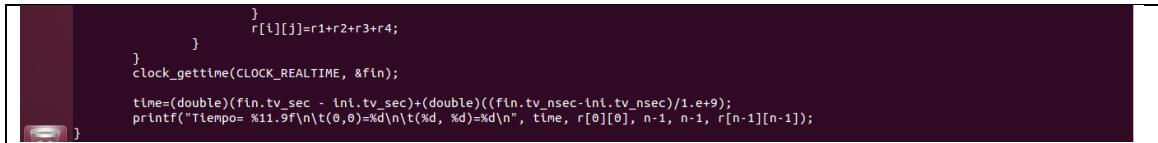
    n=atoi(argv[1]);
    if(n>max) n=max;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            r[i][j]=0;
            m1[i][j]=2;
            m2[i][j]=4;
        }
    }

    clock_gettime(CLOCK_REALTIME, &tnt);
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            m2T[i][j]=m2[j][i];
        }
    }

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            r1=0; r2=0; r3=0; r4=0;
            for(k=0;k<n;k++){
                r1+=m1[i][k]*m2T[i][k];
                r2+=m1[i][k+1]*m2T[i][k+1];
                r3+=m1[i][k+2]*m2T[i][k+2];
                r4+=m1[i][k+3]*m2T[i][k+3];
            }
        }
    }

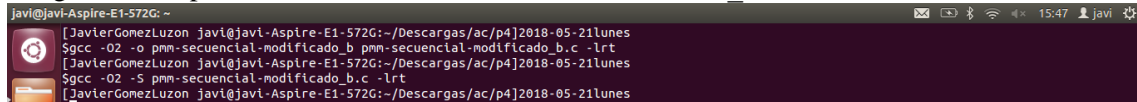
    clock_gettime( CLOCK_REALTIME, &fin);

    time=(double)(fin.tv_sec - tnt.tv_sec)+(double)((fin.tv_nsec-tnt.tv_nsec)/1.e+9);
    printf("Tiempo= %11.9f\n\t(0,0)=%d\n\t(%d, %d)=%d\n", time, r1, r2, r3, r4);
}
```



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

Imagen 8: Compilación de `pmm-secuencial-modificado_b.c`.

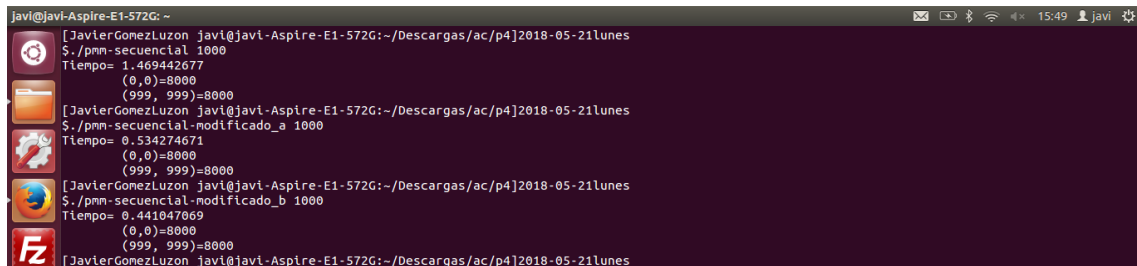


### 1.1. TIEMPOS:

Modificación	-O2
Sin modificar	1.336 s
Modificación a)	0.566 s
Modificación b)	0.439s

Estos tiempos son para ejecuciones donde el parámetro de entrada es 1000.

Imagen 9, 10 y 11: Ejecuciones de `pmm-secuencial.c`, `pmm-secuencial-modificado_a.c` y `pmm-secuencial-modificado_b.c` con el mismo parámetro.



Como podemos ver lo que más ha mejorado el tiempo de ejecución ha sido la optimización de

calcular la matriz traspuesta y multiplicar sobre esa. La segunda optimización no ha reducido tanto el tiempo. En la segunda optimización reducíamos las iteraciones del bucle en 4 (íbamos sumando de 4 en 4), tal vez si lo hubiéramos hecho por ejemplo entre 10 los tiempos se habría reducido mucho más.

## 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> .L10:     movl r(%rdi,%r8,4), %esi     movslq %r8d, %rax     leaq m2(,%rax,4), %rcx     xorl %eax, %eax     .p2align 4,,10     .p2align 3 .L8:     movl m1(%rdi,%rax), %edx     addq \$4, %rax     imull (%rcx), %edx     addq \$20000, %rcx     addl %edx, %esi     cmpq %rbx, %rax     jne .L8     movl %esi, r(%rdi,%r8,4)     addq \$1, %r8     cmpq %rbp, %r8     jne .L10     addq \$20000, %rdi     cmpq %r13, %rdi </pre>	<p><b>Calculo de traspuesta</b></p> <pre> .L7:     movslq %edi, %rax     leaq m2(,%rax,4), %rdx     xorl %eax, %eax     .p2align 4,,10     .p2align 3 .L8:     mov (%rdx), %ecx     addq \$20000, %rdx     movl %ecx, m2T(%rsi,%rax)     addq \$4, %rax     cmpq %rbx, %rax     jne .L8     addl \$1, %edi     addq \$20000, %rsi     cmpl %ebp, %edi     jne .L7 </pre> <p><b>Multiplicación de matrices</b></p> <pre> .L10:     movl m1(%rsi,%rax), %edx     imull     m2T(%rsi,%rax), %edx     addq \$4, %rax     addl %edx, %ecx     cmpq %rbx, %rax     jne .L10     movl %ecx, (%rdi)     addq \$4, %rdi     cmpq %r8, %rdi     jne .L12     addl \$1, %r9d     addq \$20000, %rsi     cmpl %ebp, %r9d </pre> <p>Como podemos ver, con este código, los saltos de 20000 posiciones solo los hacemos al calcular la traspuesta. Y en la multiplicación realizamos solo saltos de 4 posiciones, que nos valen para las dos matrices.</p>	<p><b>Multiplicación de matrices en subterminos:</b></p> <pre> .L10:     movl (%rdx), %esi     imull     m2T(%rcx,%rax), %esi     addl %esi, %edi     movl 4(%rdx), %esi     imull     m2T+4(%rcx,%rax), %esi     addl %esi, %r8d     movl 8(%rdx), %esi     imull     m2T+8(%rcx,%rax), %esi     addl %esi, %r9d     movl 12(%rdx), %esi     addq \$16, %rdx     imull     m2T+12(%rcx,%rax), %esi </pre> <p>Con este algoritmo, en vez de estar haciendo saltos y comparaciones en la cabecera del for en cada iteración, nos ahorramos algunas iteraciones, ya que como vemos esos cálculos los hace seguidos dentro del bucle.</p>

### B) CÓDIGO FIGURA 1:

**CAPTURA CÓDIGO FUENTE:** figura1-original.c

Imagen 12: Cat figura1-original.c.

```

javi@javi-Aspire-E1-572G: ~
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
Scat figural-original.c
#include <stdio.h>
#include <time.h>

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

int main(){
    int ii, X1, X2, i;
    int R[40000];
    struct timespec tni, fin;
    double time;

    clock_gettime(CLOCK_REALTIME, &tni);
    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, &fin);

    time=(double)(fin.tv_sec-tni.tv_sec)+(double)((fin.tv_nsec-tni.tv_nsec)/1.e+9);

    printf("R[0]=%d\tR[39999]=%d\n",R[0], R[39999]);
    printf("Time: %11.9f\n", time);
}

```

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

**Modificación a) –explicación–:** Realizar los dos bucles distintos en uno solo.

**Modificación b) –explicación–:** Reducir el número de iteraciones del bucle más interno.

### 1.1. CÓDIGOS FUENTE MODIFICACIONES

#### a) Captura figural-modificado\_a.c

Imagen 13: Cat figural-modificado\_a.c.

```

javi@javi-Aspire-E1-572G: ~
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
Scat figural-modificado_a.c
#include <stdio.h>
#include <time.h>

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

int main(){
    int ii, X1, X2, i;
    int R[40000];
    struct timespec tni, fin;
    double time;

    clock_gettime(CLOCK_REALTIME, &tni);
    for(ii=0;ii<40000;ii++){
        X1=0; X2=0;
        for(i=0;i<500;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, &fin);

    time=(double)(fin.tv_sec-tni.tv_sec)+(double)((fin.tv_nsec-tni.tv_nsec)/1.e+9);

    printf("R[0]=%d\tR[39999]=%d\n",R[0], R[39999]);
    printf("Time: %11.9f\n", time);
}

```

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

Imagen 14: Compilacion de figural-modificado\_a.c.

```

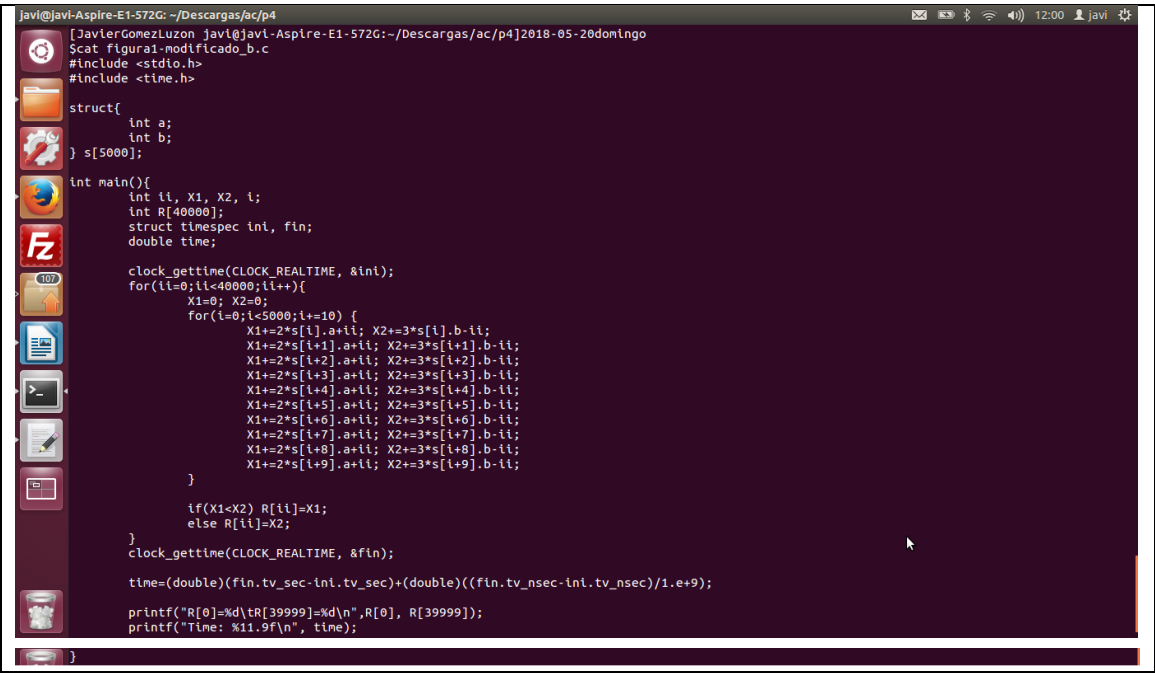
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p4
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$gcc -O2 -o figural-modificado_a figural-modificado_a.c -lrt
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$gcc -O2 -S figural-modificado_a.c -lrt
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo

```

Las capturas de pantalla de la ejecución se mostrarán más adelante con las ejecuciones de las demás versiones de este programa (en el apartado de tiempos).

#### b) Captura figural-modificado\_b.c

Imagen 15 y 16: Cat figural-modificado\_b.c.



```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p4
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
Scat figural-modificado_b.c
#include <stdio.h>
#include <time.h>

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

int main(){
    int i1, X1, X2, i;
    int R[40000];
    struct timespec t1, t2;
    double time;

    clock_gettime(CLOCK_REALTIME, &t1);
    for(i1=0; i1<40000; i1++){
        X1=0; X2=0;
        for(i=0; i<5000; i+=10){
            X1+=2*s[i].a+i1; X2+=3*s[i].b-i1;
            X1+=2*s[i+1].a+i1; X2+=3*s[i+1].b-i1;
            X1+=2*s[i+2].a+i1; X2+=3*s[i+2].b-i1;
            X1+=2*s[i+3].a+i1; X2+=3*s[i+3].b-i1;
            X1+=2*s[i+4].a+i1; X2+=3*s[i+4].b-i1;
            X1+=2*s[i+5].a+i1; X2+=3*s[i+5].b-i1;
            X1+=2*s[i+6].a+i1; X2+=3*s[i+6].b-i1;
            X1+=2*s[i+7].a+i1; X2+=3*s[i+7].b-i1;
            X1+=2*s[i+8].a+i1; X2+=3*s[i+8].b-i1;
            X1+=2*s[i+9].a+i1; X2+=3*s[i+9].b-i1;
        }

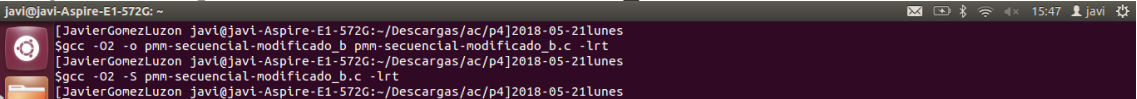
        if(X1<X2) R[i1]=X1;
        else R[i1]=X2;
    }
    clock_gettime(CLOCK_REALTIME, &t2);

    time=(double)(t2.tv_sec-t1.tv_sec)+(double)((t2.tv_nsec-t1.tv_nsec)/1.e+9);

    printf("R[0]=%d\tR[39999]=%d\n", R[0], R[39999]);
    printf("Time: %11.9f\n", time);
}
```

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

Imagen 17: Compilacion de figural-modificado\_b.c.



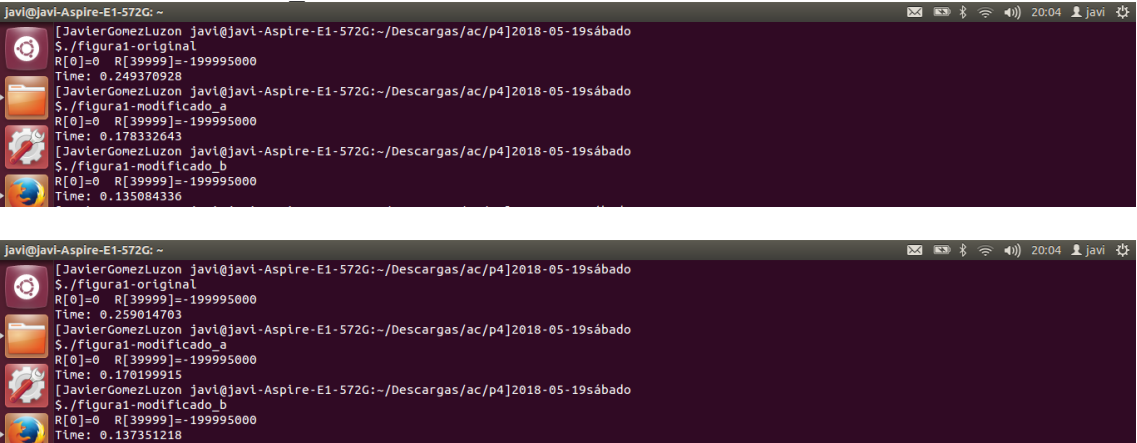
```
javi@javi-Aspire-E1-572G: ~
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-21lunes
$gcc -O2 -o pmm-secuencial-modificado_b pmm-secuencial-modificado_b.c -lrt
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-21lunes
$gcc -O2 -S pmm-secuencial-modificado_b.c -lrt
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-21lunes
```

Las capturas de pantalla de la ejecución se mostrarán más adelante con las ejecuciones de las demás versiones de este programa (en el apartado de tiempos).

1.1. TIEMPOS:

Modificación	-O2
Sin modificar	0.253s
Modificación a)	0.173s
Modificación b)	0.134s

Imagen 18, 19 y 20: Ejecución de figural-original.c, figural-modificado\_a.c y figural-modificado\_b.c.



```
javi@javi-Aspire-E1-572G: ~
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
$./figural-original
R[0]=0 R[39999]=-199995000
Time: 0.249370928
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
$./figural-modificado_a
R[0]=0 R[39999]=-199995000
Time: 0.178332643
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
$./figural-modificado_b
R[0]=0 R[39999]=-199995000
Time: 0.135084336

javi@javi-Aspire-E1-572G: ~
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
$./figural-original
R[0]=0 R[39999]=-199995000
Time: 0.259014703
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
$./figural-modificado_a
R[0]=0 R[39999]=-199995000
Time: 0.17019915
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
$./figural-modificado_b
R[0]=0 R[39999]=-199995000
Time: 0.137351218
```



```

javi@javi-Aspire-E1-572G: ~
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
$./figura1-origlnal
R[0]=0 R[39999]=199995000
Time: 0.251313611
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
$./figura1-modificado_a
R[0]=0 R[39999]=199995000
Time: 0.172971655
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-19sábado
$./figura1-modificado_b
R[0]=0 R[39999]=199995000
Time: 0.132137412

```

### 1.1. COMENTARIOS SOBRE LOS RESULTADOS:

En la estructura los datos están seguidos unos de otros (a1, b1, a2, b2, a3, b3...). Por lo que en si lo ejecutamos seguido (el calculo con la variable 'a' y después el calculo con la variable 'b') vemos bastante mejora en el tiempo. La segunda mejora también ha sido bastante efectiva ya que nos hemos quitado 5000/10 comparaciones y saltos del bucle for con respecto a la ejecución anterior.

### 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_b.s	pmm-secuencial-modificado_c.s
<pre> .L3:     movl (%rax), %edx     addq \$8, %rax     leal (%rdi,%rdx,2), %edx     addl %edx, %ecx     cmpq \$s+40000, %rax     jne .L3     movl \$s+4, %eax     xorl %esi, %esi     .p2align 4,,10     .p2align 3 .L4:     movl (%rax), %edx     addq \$8, %rax     leal (%rdx,%rdx,2), %edx     subl %edi, %edx     addl %edx, %esi     cmpq \$s+40004, %rax     jne .L4 </pre>	<pre> Calculo: .L3:     movl (%rax), %edx     leal (%rdi,%rdx,2), %edx     addl %edx, %ecx     movl 4(%rax), %edx     addq \$8, %rax     leal (%rdx,%rdx,2), %edx     subl %edi, %edx     addl %edx, %esi     cmpq \$s+40000, %rax     jne .L3 </pre> <p>Como vemos, aquí a parte de realizar muchos menos saltos, estos son más pequeños. En la versión original realiza el salto de 8 posiciones 5000*2 veces, mientras que aquí lo realizamos la mitad de veces, además de evitarnos los saltos de ejecutar dos bucles.</p>	<pre> Calculo:     movl 8(%rax), %r10d     addl %esi, %ecx     movl 12(%rax), %esi     leal (%rdx,%r10,2), %r10d     leal (%rsi,%rsi,2), %esi     addl %r10d, %r9d     movl 16(%rax), %r10d     subl %edx, %esi     addl %esi, %ecx     ...     movl 68(%rax), %ecx     leal (%rcx,%rcx,2), %ecx     subl %edx, %ecx     addl %ecx, %esi     movl 76(%rax), %ecx     addq \$80, %rax     leal (%rcx,%rcx,2), %ecx     subl %edx, %ecx     addl %esi, %ecx     cmpl \$5000, %edi     jne .L3 </pre> <p>Con este algoritmo, en vez de estar haciendo saltos y comparaciones en la cabecera del for en cada iteración, nos ahorramos algunas iteraciones, ya que como vemos esos cálculos los hace seguidos dentro del bucle.</p>

- 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 acarreen. 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

Imagen 21: Cat daxpy.c.

```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p4
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$ cat daxpy.c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define max 33554432

int x[max], y[max];

int main(int argc, char *argv[]){
    int n, constante, i;
    struct timespec ini, fin;
    double time;

    if(argc<3){
        fprintf(stderr, "Falta el tamaño del vector y la constante");
        exit(-1);
    }

    n=atoi(argv[1]);
    constante=atoi(argv[2]);
    if(n>max) n=max;

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

    clock_gettime(CLOCK_REALTIME, &ini);
    for(i=0;i<n;i++) y[i]= constante*x[i]+y[i];
    clock_gettime(CLOCK_REALTIME, &fin);

    time=(double)(fin.tv_sec-ini.tv_sec)+(double)((fin.tv_nsec-ini.tv_nsec)/1.e+9);

    printf("Time=%11.9f\ty[0]=%d, y[%d]=%d\n", time, y[0], n-1, y[n-1]);
}
```

Tiempos ejec.	-O0	-Os	-O2	-O3
	0.000007627s	0.000002671s	0.000001473s	0.000000759s

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

Imagen 22: Compilación de daxpy.c con las opciones -O0, -Os, -O2 y -O3.

```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p4
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$ gcc -O0 -o daxpy0 daxpy.c -lrt
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$ gcc -O0 -S daxpy.c -lrt && cat daxpy.s > daxpy0.s
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$ gcc -Os -o daxpyS daxpy.c -lrt
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$ gcc -Os -S daxpy.c -lrt && cat daxpy.s > daxpyS.s
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$ gcc -O2 -o daxpy2 daxpy.c -lrt
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$ gcc -O2 -S daxpy.c -lrt && cat daxpy.s > daxpy2.s
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$ gcc -O3 -o daxpy3 daxpy.c -lrt
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
$ gcc -O3 -S daxpy.c -lrt && cat daxpy.s > daxpy3.s
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
```

Imagen 23, 24 y 25: Ejecución de daxpy.c con las opciones -O0, -Os, -O2 y -O3.

```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p4
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
S./daxpy0 1000 5
Time=0.000004891 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
S./daxpy5 1000 5
Time=0.000002085 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
S./daxpy2 1000 5
Time=0.000001157 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
S./daxpy3 1000 5
Time=0.000000831 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
```

```
Archivo Editar Ver Buscar Terminal Ayuda
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
S./daxpy0 1000 5
Time=0.000007037 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
S./daxpy5 1000 5
Time=0.000002081 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
S./daxpy2 1000 5
Time=0.000000817 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
S./daxpy3 1000 5
Time=0.000000648 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-20domingo
```

```
javi@javi-Aspire-E1-572G: ~
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-24jueves
S./daxpy0 1000 5
Time=0.000010955 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-24jueves
S./daxpy5 1000 5
Time=0.000003047 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-24jueves
S./daxpy2 1000 5
Time=0.000002446 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-24jueves
S./daxpy3 1000 5
Time=0.000000859 y[0]=0, y[999]=1002996
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p4]2018-05-24jueves
```

## COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Como vemos, para las opciones Os y O2 el código ensamblador es mucho mas corto que para O0 y O3.

O0: No hay optimización con respecto al código original, es la opción por defecto.

Os: Se optimiza usando las opciones de O2 que no aumenten el tamaño del código.

O2: Se optimiza el número de instrucciones en comparación a O0, pero aquí si usa las opciones de optimización que aumentan el tamaño del código, aunque el aumento de tamaño es insignificante.

O3: Activa las opciones que hacen que el tiempo de compilación y el uso de memoria aumente.

**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
<pre>movl \$0, -8(%rbp) jmp .L6 .L7: movl -8(%rbp), %eax cltq movl x(,%rax,4), %eax movl %eax, %edx imull -4(%rbp), %edx movl -8(%rbp), %eax cltq movl y(,%rax,4), %eax addl %eax, %edx movl -8(%rbp), %eax cltq movl %edx, y(,%rax,4) addl \$1, -8(%rbp) .L6: movl -8(%rbp), %eax</pre>	<pre>xorl %eax, %eax jmp .L5 .L6: movl x(,%rax,4), %edx imull %ebp, %edx addl %edx, y(,%rax,4) incq %rax .L5: cmpl %eax, %ebx jg .L6 leaq 16(%rsp), %rsi xorl %edi, %edi</pre>	<pre>xorl %eax, %eax .p2align 4,,10 .p2align 3 .L5: movl x(%rax), %edx imull %ebp, %edx addl %edx, y(%rax) addq \$4, %rax cmpq %rbx, %rax jne .L5 .L6: leaq 16(%rsp), %rsi xorl %edi, %edi</pre>	<pre>testb %r14b, %r14b jne .L16 .L13: movl %r12d, 12(%rsp) xorl %eax, %eax xorl %edx, %edx movd 12(%rsp), %xmm0 pshufd \$0, %xmm0, %xmm2 .p2align 4,,10 .p2align 3 .L9: movdqa x(%rax), %xmm1 addl \$1, %edx movdqa %xmm2, %xmm0 movdqa %xmm2, %xmm3 pmuludq %xmm1, %xmm0 psrldq \$4, %xmm1</pre>

<pre> cmpl -12(%rbp), %eax jl .L7 leaq -48(%rbp), %rax movq %rax, %rsi movl \$0, %edi </pre>			<pre> pshufd \$8, %xmm0, %xmm0 psrldq \$4, %xmm3 pmuludq %xmm3, %xmm1 pshufd \$8, %xmm1, %xmm1 punpckldq %xmm1, %xmm0 padd y(%rax), %xmm0 movdqa %xmm0, y(%rax) addq \$16, %rax cmpl %ebx, %edx jb .L9 cmpl %r13d, %ebp je .L12 .p2align 4,,10 .p2align 3 .L17: movslq %r13d, %rax addl \$1, %r13d movl x(,%rax,4), %edx imull %r12d, %edx addl %edx, y(,%rax,4) cmpl %r13d, %ebp jg .L17 .L12: leaq 32(%rsp), %rsi xorl %edi, %edi </pre>
--	--	--	--