

2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Javier Ramirez Pulido

Grupo de prácticas y profesor de prácticas: C2 JORGE SANCHEZ GARRIDO

Fecha de entrega: 30/05/2020

Fecha evaluación en clase:

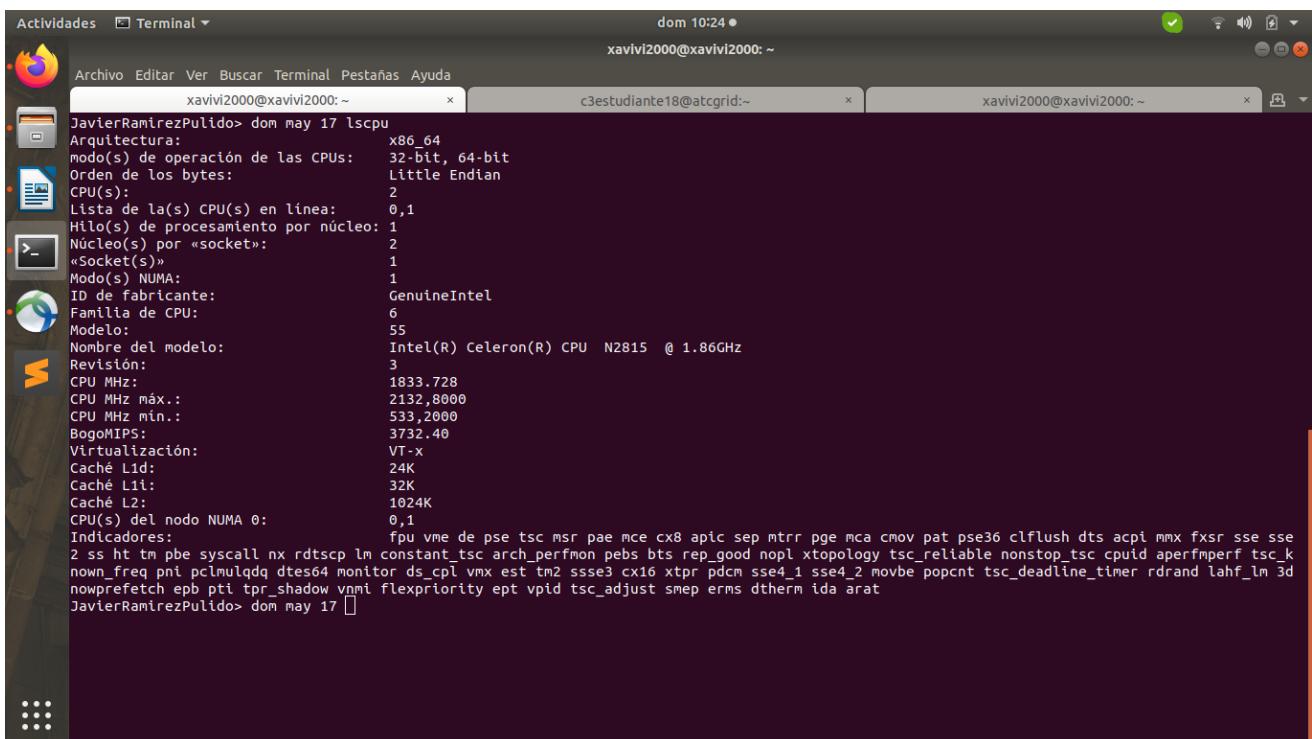
Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Celeron(R) CPU N2815 @ 1.86GHz*

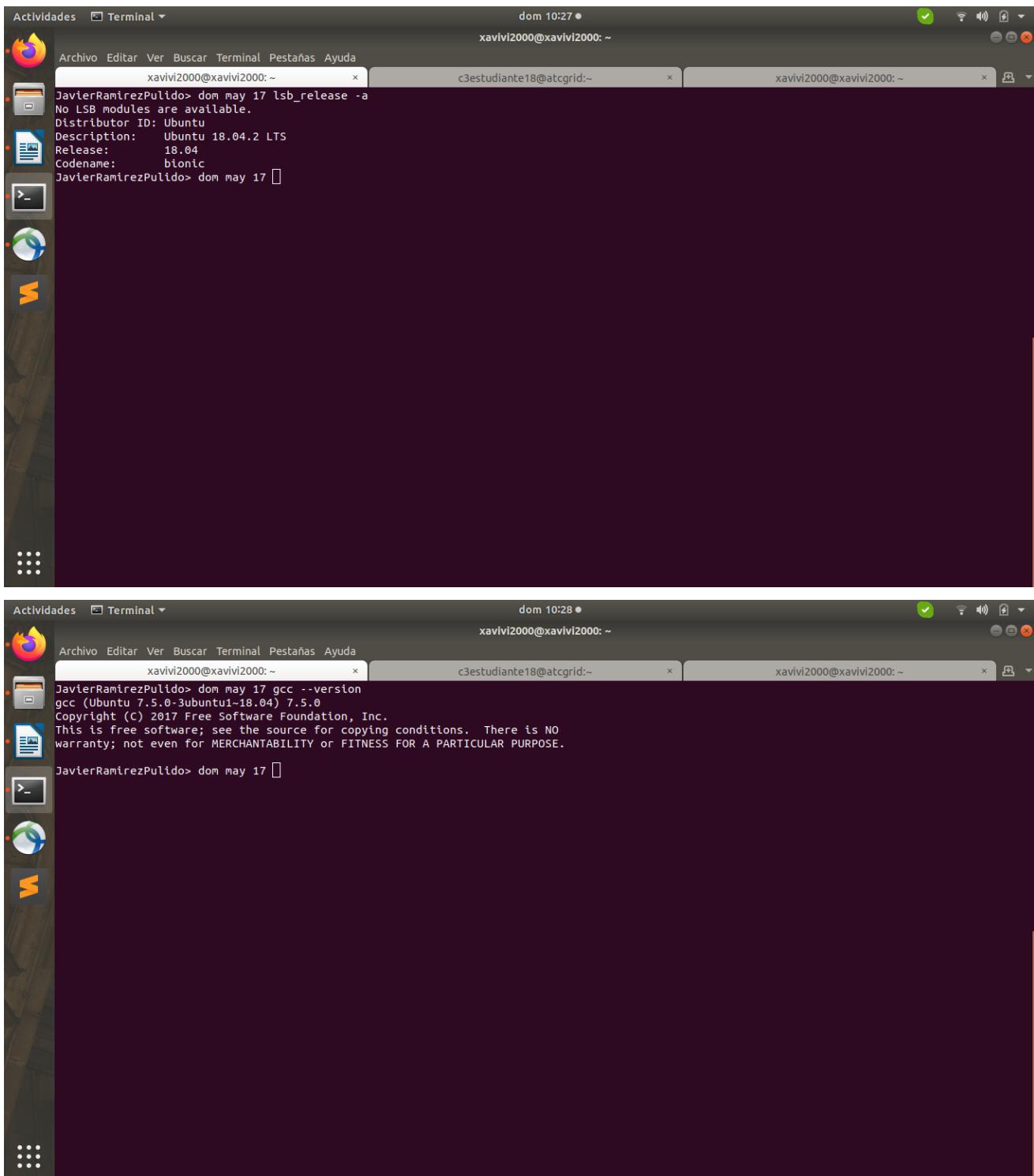
Sistema operativo utilizado: *Ubuntu 18.04.2 LTS*

Versión de gcc utilizada: *gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0*

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



```
JavierRamirezPulido> dom may 17 lscpu
Arquitectura:                                x86_64
modo(s) de operación de las CPUs:          32-bit, 64-bit
Orden de los bytes:                          Little Endian
CPU(s):                                     2
Lista de la(s) CPU(s) en linea:            0,1
Hilo(s) de procesamiento por núcleo:      1
Núcleo(s) por «socket»:                    2
«Socket(s)»:                               1
Modo(s) NUMA:                             1
ID de fabricante:                         GenuineIntel
Familia de CPU:                           6
Modelo:                                    55
Nombre del modelo:                        Intel(R) Celeron(R) CPU N2815 @ 1.86GHz
Revisión:                                  3
CPU MHz:                                   1833.728
CPU MHz máx.:                            2132.8000
CPU MHz mín.:                            533.2000
BogomIPS:                                 3732.40
Virtualización:                          VT-x
Caché L1d:                                24K
Caché L1t:                                32K
Caché L2:                                 1024K
CPU(s) del nodo NUMA 0:                  0,1
Indicadores:                            fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse
2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good noopl xtstopology tsc_reliable nonstop_tsc cpuid aperfmpfperf tsc_k
nmon_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtr pdcm sse4_1 sse4_2 movbe popcnt tsc_deadline_timer rdrand lahf_lm 3d
nowprefetch epb pti tpr_shadow vnmi flexpriority ept vpid tsc_adjust smep erms dtherm ida arat
JavierRamirezPulido> dom may 17
```



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

1.1 Modifique el código C para reducir el tiempo de ejecución (evalúe el tiempo y modifique sólo el trozo que hace la multiplicación y el trozo que se muestra en la Figura 1). 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

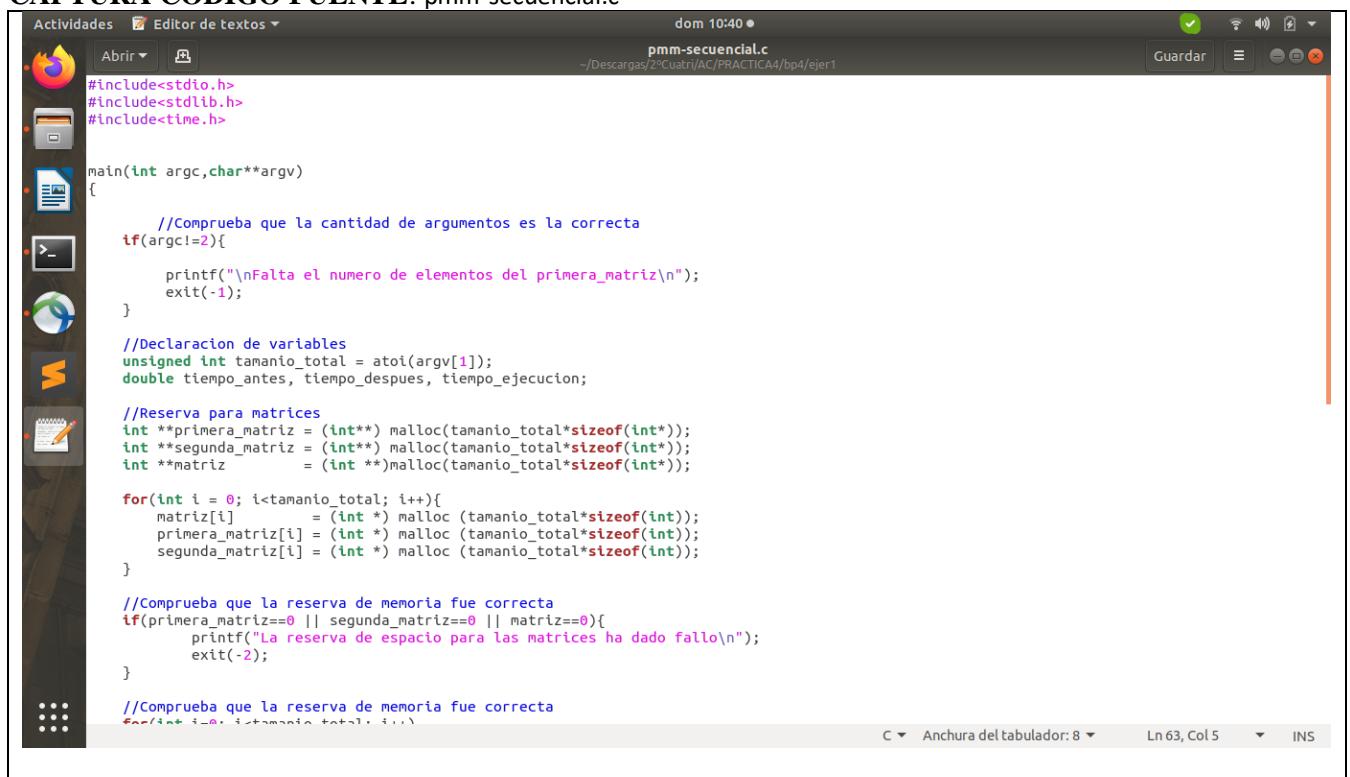
```
struct {
    int a;
    int b;
} s[5000];

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



```
Actividades Editor de textos ▾
Abrir ▾ Guardar
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

main(int argc,char**argv)
{
    //Comprueba que la cantidad de argumentos es la correcta
    if(argc!=2){
        printf("\nFalta el numero de elementos del primera_matriz\n");
        exit(-1);
    }

    //Declaracion de variables
    unsigned int tamano_total = atoi(argv[1]);
    double tiempo_anterior, tiempo_despues, tiempo_ejecucion;

    //Reserva para matrices
    int **primera_matriz = (int**) malloc(tamano_total*sizeof(int*));
    int **segunda_matriz = (int**) malloc(tamano_total*sizeof(int*));
    int **matriz         = (int**) malloc(tamano_total*sizeof(int*));

    for(int i = 0; i<tamano_total; i++){
        matriz[i] = (int *) malloc (tamano_total*sizeof(int));
        primera_matriz[i] = (int *) malloc (tamano_total*sizeof(int));
        segunda_matriz[i] = (int *) malloc (tamano_total*sizeof(int));
    }

    //Comprueba que la reserva de memoria fue correcta
    if(primera_matriz==0 || segunda_matriz==0 || matriz==0){
        printf("La reserva de espacio para las matrices ha dado fallo\n");
        exit(-2);
    }

    //Comprueba que la reserva de memoria fue correcta
    for(int i=0; i<tamano_total; i++)
        for(int j=0; j<tamano_total; j++)
            for(int k=0; k<tamano_total; k++)
                matriz[i][j] = primera_matriz[i][k]*segunda_matriz[k][j];
}

//Comprueba que la multiplicación es correcta
if(strcmp(matriz, "correcto") == 0)
    printf("La multiplicación es correcta\n");
else
    printf("La multiplicación es incorrecta\n");
}

Anchura del tabulador: 8 ▾ Ln 63, Col 5 ▾ INS
```

The screenshot shows a Linux desktop environment with a terminal window and a code editor window.

Code Editor Window:

```

Actividades Editor de textos
dom 10:41 •
pmm-secuencial.c
-/Descargas/2ºCuatri/AC/PRACTICA4/bp4/ejer1
Guarda Guardar
Abrir Guardar
} }

//Comprueba que la reserva de memoria fue correcta
for(int i=0; i<tamano_total; i++){
    if(primer_matriz[i]==0 || segunda_matriz[i]==0 || matriz[i]==0){
        printf("La reserva de espacio ha dado fallo\n");
        exit(-2);
    }
}

//Rellenar la matriz, primera_matriz y primera_matriz de resultados
for(int i=0; i<tamano_total; i++){
    for(int j=0; j<tamano_total; j++){
        matriz[i][j] = 0;
        primera_matriz[i][j] = j*2;
        segunda_matriz[i][j] = j+2;
    }
}

//Realiza las cuentas y se mide el tiempo antes y despues de hacerlo para calcular el tiempo de ejecucion
tiempo_anteriores = omp_get_wtime();

for(int i=0; i<tamano_total; i++)
    for(int j=0; j<tamano_total; j++)
        for(int k=0; k<tamano_total; k++)
            matriz[i][j] += primera_matriz[i][k] * segunda_matriz[k][j];

tiempo_despues = omp_get_wtime();
tiempo_ejecucion=tiempo_despues-tiempo_anteriores;

//Muestra dimension y tiempo final
printf("\nDimension de las matrices: %d Tiempo: %11.9f segundos\n\n", tamano_total, tiempo_ejecucion);
printf("\nPrimer elemento (matriz_resultado[0][0]) = %d, Ultimo elemento (matriz_resultado[%d][%d]) = %d\n\n", matriz[0][0], tamano_total-1, tamano_total-1, matriz[tamano_total-1][tamano_total-1]);

return 0;

```

Terminal Window:

```

Actividades Terminal
dom 10:56 •
xavivl2000@xavivl2000: ~
c3estudiante18@atcggrid: ~
xavivl2000@xavivl2000: ~
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -o pmm-secuencial pmm-secuencial.c
pmm-secuencial.c:14:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(int argc,char**argv)
^~~~~
JavierRamirezPulido> dom may 17 ./pmm-secuencial 12
Dimension de las matrices: 12 Tiempo: 0.000171384 segundos
JavierRamirezPulido> dom may 17 ./pmm-secuencial 12
Primer elemento (matriz_resultado[0][0]) = 264, Ultimo elemento (matriz_resultado[11][11]) = 1716
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -O2 -o pmm-secuencial pmm-secuencial.c
pmm-secuencial.c:14:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(int argc,char**argv)
^~~~~
JavierRamirezPulido> dom may 17 ./pmm-secuencial 12
Dimension de las matrices: 12 Tiempo: 0.000005138 segundos
Primer elemento (matriz_resultado[0][0]) = 264, Ultimo elemento (matriz_resultado[11][11]) = 1716
JavierRamirezPulido> dom may 17

```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación–: Con el objetivo de acercar en memoria la posición de los datos se cambia el nombre de las variables en los dos bucles más internos. En mi caso, el avance se *j* se sustituye por el de *k* y viceversa, ambos cambios en los bucles internos de los 3 que están anidados para realizar las cuentas.

Modificación b) –explicación–: Se realiza una técnica de desenrollado de bucle para aumentar el tamaño del bloque básico. Se desenrolla en 8 iteraciones, lo cual transforma el bucle de partida en un nuevo bucle cuyas iteraciones incluyen las operaciones de varias de las iteraciones del bucle de partida no desenrollado. Se tienen así bloques básicos con mas instrucciones y, al mismo tiempo, disminuye el numero

de instrucciones de salto a ejecutar para controlar el bucle dado que el numero de iteraciones es menor.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura de pmm-secuencial-modificado_a.c

```

Actividades Editor de textos dom 10:46 ●
pmm-secuencial-modificado_a.c
~/Descargas/2ºCuatri/AC/PRACTICA4/bp4/ejer1 Guardar
pmm-secuencial-modificado_a.c x

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

main(int argc,char**argv)
{
    //Comprueba que la cantidad de argumentos es la correcta
    if(argc!=2){
        printf("\nFalta el numero de elementos del primera_matriz\n");
        exit(-1);
    }

    //Declaracion de variables
    unsigned int tamano_total = atoi(argv[1]);
    double tiempo_anterior, tiempo_despues, tiempo_ejecucion;

    //Reserva para matrices
    int **primera_matriz = (int**) malloc(tamano_total*sizeof(int));
    int **segunda_matriz = (int**) malloc(tamano_total*sizeof(int));
    int **matriz         = (int **)malloc(tamano_total*sizeof(int));

    for(int i = 0; i<tamano_total; i++){
        matriz[i]           = (int *) malloc (tamano_total*sizeof(int));
        primera_matriz[i] = (int *) malloc (tamano_total*sizeof(int));
        segunda_matriz[i] = (int *) malloc (tamano_total*sizeof(int));
    }

    //comprueba que la reserva de memoria fue correcta
    if(primera_matriz==0 || segunda_matriz==0 || matriz==0){
        printf("La reserva de espacio para las matrices ha dado fallo\n");
        exit(-2);
    }
}

Actividades Editor de textos dom 10:47 ●
*pmm-secuencial-modificado_a.c
~/Descargas/2ºCuatri/AC/PRACTICA4/bp4/ejer1 Guardar
pmm-secuencial-modificado_a.c x

//Comprueba que la reserva de memoria fue correcta
for(int i=0; i<tamano_total; i++)
    if(primera_matriz[i]==0 || segunda_matriz[i]==0 || matriz[i]==0){
        printf("La reserva de espacio ha dado fallo\n");
        exit(-2);
    }

    //Rellenar la matriz, primera_matriz y primera_matriz de resultados
    for(int i=0; i<tamano_total; i++){
        for(int j=0; j<tamano_total; j++){
            matriz[i][j]       = 0 ;
            primera_matriz[i][j] = j*2;
            segunda_matriz[i][j] = j+2;
        }
    }

    //Realiza las cuentas y se mide el tiempo antes y despues de hacerlo para calcular el tiempo de ejecucion
    tiempo_anterior = omp_get_wtime();

    for(int i=0; i<tamano_total; i++)
        for(int k=0; k<tamano_total; k++)
            for(int j=0; j<tamano_total; j++)
                matriz[i][j] += primera_matriz[i][k] * segunda_matriz[k][j];

    tiempo_despues = omp_get_wtime();
    tiempo_ejecucion=tiempo_despues-tiempo_anterior;

    //Muestra dimension y tiempo final
    printf("\nDimension de las matrices: %d    Tiempo: %11.9f segundos\n", tamano_total, tiempo_ejecucion);
    printf("\nPrimer elemento (matriz_resultado[0][0]) = %d, Ultimo elemento (matriz_resultado[%d][%d]) = %d\n", matriz[0][0], tamano_total-1, tamano_total-1, matriz[tamano_total-1][tamano_total-1]);

    return 0;
}

```

```

Actividades Terminal dom 11:45 ●
xavivi2000@xavivi2000: ~ xavivi2000@xavivi2000: ~ xavivi2000@xavivi2000: ~
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
xavivi2000@xavivi2000: ~ x3estudiante18@atcgrid:~ xavivi2000@xavivi2000: ~
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -O2 -o pmm-secuencial-modificado_a pmm-secuencial-modificado_a.c
pmm-secuencial-modificado_a.c:14:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(int argc,char**argv)
^~~~
JavierRamirezPulido> dom may 17 ./pmm-secuencial-modificado_a 12
Dimension de las matrices: 12 Tiempo: 0.000004365 segundos
JavierRamirezPulido> dom may 17 ./pmm-secuencial-modificado_a 12
Primer elemento (matriz_resultado[0][0]) = 264, Ultimo elemento (matriz_resultado[11][11]) = 1716
JavierRamirezPulido> dom may 17 ./pmm-secuencial-modificado_a 12
Dimension de las matrices: 12 Tiempo: 0.000214254 segundos
Primer elemento (matriz_resultado[0][0]) = 264, Ultimo elemento (matriz_resultado[11][11]) = 1716
JavierRamirezPulido> dom may 17

```

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

b) Captura de pmm-secuencial-modificado_b.c

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

```

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#ifndef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#define omp_get_num_threads() 1
#define omp_set_num_threads(int)
#define omp_in_parallel() 0
#define omp_set_dynamic(int)
#endif

main(int argc,char**argv)
{
    //Comprueba que la cantidad de argumentos es la correcta
    if(argc!=2){
        printf("\nFalta el numero de elementos del primera_matriz\n");
        exit(-1);
    }

    //Declaracion de variables
    unsigned int tamano_total = atoi(argv[1]);
    double tiempo_antes, tiempo_despues, tiempo_ejecucion;

    //Declaro 8 vectores y uno para resultado
    int vector_resultado, primerv, segundov, tercerv, cuartov, quintov, sextov, septimov, octavov, repeticiones = tamano_total/8;
    primerv = segundov = tercerv = cuartov = quintov = sextov = septimov = octavov = 0;

    //Reserva para matrices
    int **primera_matriz = (int**) malloc(tamano_total*sizeof(int__));
    int **segunda_matriz = (int**) malloc(tamano_total*sizeof(int__));
    int **matriz         = (int **)malloc(tamano_total*sizeof(int));

    for(int i = 0; i<tamano_total; i++){
        matriz[i]      = (int *) malloc (tamano_total*sizeof(int));
        primera_matriz[i] = (int *) malloc (tamano_total*sizeof(int));
    }
}

```

```

    segunda_matriz[i] = (int *) malloc (tamano_total*sizeof(int));
}

//Comprueba que la reserva de memoria fue correcta
if(primer_matriz==0 || segunda_matriz==0 || matriz==0){
    printf("La reserva de espacio para las matrices ha dado fallo\n");
    exit(-2);
}

//Comprueba que la reserva de memoria fue correcta
for(int i=0; i<tamano_total; i++){
    if(primer_matriz[i]==0 || segunda_matriz[i]==0 || matriz[i]==0){
        printf("La reserva de espacio ha dado fallo\n");
        exit(-2);
    }
}

//Rellenar la matriz, primera_matriz y primera_matriz de resultados
for(int i=0; i<tamano_total; i++){
    for(int j=0; j<tamano_total; j++){
        matriz[i][j] = 0;
        primera_matriz[i][j] = j*2;
        segunda_matriz[i][j] = j+2;
    }
}

//Realiza las cuentas y se mide el tiempo antes y despues de hacerlo para calcular el tiempo de ejecucion
tiempo_anter = omp_get_wtime();
int aux;
for(int i=0; i<tamano_total; i++){
for(int j=0; j<tamano_total; j++){
    aux = 0;
    for(int k=0; k<repeticiones; k++){

        primerv += (primer_matriz[i][aux+0]*segunda_matriz[j][aux+0]);
        segundo += (primer_matriz[i][aux+1]*segunda_matriz[j][aux+1]);
        tercerv += (primer_matriz[i][aux+2]*segunda_matriz[j][aux+2]);
        cuartov += (primer_matriz[i][aux+3]*segunda_matriz[j][aux+3]);
        quintov += (primer_matriz[i][aux+4]*segunda_matriz[j][aux+4]);
        sextov += (primer_matriz[i][aux+5]*segunda_matriz[j][aux+5]);
        septimov += (primer_matriz[i][aux+6]*segunda_matriz[j][aux+6]);
        octavov += (primer_matriz[i][aux+7]*segunda_matriz[j][aux+7]);
        aux += 8;

    }

    vector_resultado = primerv + segundo + tercerv + cuartov + quintov + sextov + septimov + octavov;
    matriz[i][j] = vector_resultado;

    for(aux = repeticiones*8 ; aux<tamano_total; aux++){
        vector_resultado = vector_resultado + (segunda_matriz[j][aux]*primer_matriz[i][aux]);
    }

    matriz[i][j] = vector_resultado;
}
}

tiempo_despues = omp_get_wtime();
tiempo_ejecucion=tiempo_despues-tiempo_anter;

//Muestra dimension y tiempo final
printf("\nDimension de las matrices: %d Tiempo: %11.9f segundos\n\n", tamano_total, tiempo_ejecucion);
printf("\nPrimer elemento (matriz_resultado[0][0]) = %d, Ultimo elemento (matriz_resultado[%d][%d]) = %d\n\n",
matriz[0][0],tamano_total-1,tamano_total-1,matriz[tamano_total-1][tamano_total-1]);

return 0;
}

```

```

Actividades Terminal dom 11:44 ●
xavivi2000@xavivi2000: ~
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
xavivi2000@xavivi2000: ~ x c3estudiante18@atcgrid:~ x xavivi2000@xavivi2000: ~
JavierRamirezPulido> dom may 17 gcc -lrlt -fopenmp -O2 -o pmm-secuencial-modificado_b pmm-secuencial-modificado_b.c
pmm-secuencial-modificado_b.c:14:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(int argc,char**argv)
^~~~~
JavierRamirezPulido> dom may 17 ./pmm-secuencial-modificado_b 12
Dimension de las matrices: 12 Tiempo: 0.000007875 segundos
>_
Primer elemento (matriz_resultado[0][0]) = 1276, Ultimo elemento (matriz_resultado[11][11]) = 57332
JavierRamirezPulido> dom may 17 gcc -lrlt -fopenmp -O2 -o pmm-secuencial-modificado_b pmm-secuencial-modificado_b.c
pmm-secuencial-modificado_b.c:14:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(int argc,char**argv)
^~~~~
JavierRamirezPulido> dom may 17 ./pmm-secuencial-modificado_b 12
Dimension de las matrices: 12 Tiempo: 0.000032520 segundos
>_
Primer elemento (matriz_resultado[0][0]) = 1276, Ultimo elemento (matriz_resultado[11][11]) = 57332
JavierRamirezPulido> dom may 17

```

1.1. TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	-	25.495236753
Modificación a)	Intercambiar dos de los bucles para acercar las posiciones en memoria	3.330784344
Modificación b)	Dividir las ejecuciones del bucle en 8 iteraciones desenrolladas	2.659169333

```

Actividades Terminal dom 11:41 ●
xavivi2000@xavivi2000: ~
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
xavivi2000@xavivi2000: ~ x c3estudiante18@atcgrid:~ x xavivi2000@xavivi2000: ~
JavierRamirezPulido> dom may 17 ./pmm-secuencial 1200
Dimension de las matrices: 1200 Tiempo: 25.495236753 segundos
>_
Primer elemento (matriz_resultado[0][0]) = 2877600, Ultimo elemento (matriz_resultado[1199][1199]) = 1727998800
JavierRamirezPulido> dom may 17 ./pmm-secuencial-modificado_a 1200
Dimension de las matrices: 1200 Tiempo: 3.330784344 segundos
>_
Primer elemento (matriz_resultado[0][0]) = 2877600, Ultimo elemento (matriz_resultado[1199][1199]) = 1727998800
JavierRamirezPulido> dom may 17 ./pmm-secuencial-modificado_b 1200
Dimension de las matrices: 1200 Tiempo: 2.659169333 segundos
>_
Primer elemento (matriz_resultado[0][0]) = 1153438000, Ultimo elemento (matriz_resultado[1199][1199]) = 967290880
JavierRamirezPulido> dom may 17

```

1.1. COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:

Que la primera modificación reduzca el tiempo en 22 segundos es la prueba de que aproximar datos en memoria reduce el tiempo de acceso a ellos. Ya con estos tamaños la diferencia es altamente notable teniendo en cuenta que son ejecuciones en un pc, que deberían rondar las milésimas de

segundos es las ejecuciones, pero cuanto más grande sea el número de operaciones que hace, y por tanto, el acceso a memoria a leerlos y escribirlos, más diferencia se notará. Ya, la siguiente modificación no solo reduce el tiempo, sino que lo hace aun en una mayor cantidad de segundos. Para este caso concreto, son 23 los segundo ahorrados. El único problema del desenrollado de bucles es el elevado tamaño de la memoria ocupada en comparación con los otros dos, pero aumentar el numero de operaciones independientes en el bloque básico se disminuye el numero de instrucciones de salto a ejecutar.

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.(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

Pmm-secuencial.s	pmm-secuencial-modificado_a.s	pmm-secuencial-modificado_b.s
movl \$0, -72(%rbp)	xorl %ecx, %ecx	leaq 0(%rax,8), %rdx
leaq 4(%rax, 4), %rbp	.L18:	movq -64(%rbp), %rax
jmp .L18:	movl -80(%rbp), %eax	addq %rdx, %rax
movl -80(%rbp), %eax	cltq	movq (%rax), %rax
cltq	leaq 0(%rax,8), %rdx	movslq %edx, %rdx
leaq 0(%rax,8), %rdx	addq %rdx, %rax	addq \$7, %rdx
movq -48(%rbp), %rax	movq -48(%rbp), %rax	salq \$2, %rdx
addq %rdx, %rax	movl -72(%rbp), %edx	addq %rdx, %rax
movq (%rbx,%rax,8), %rdx	movslq %edx, %rdx	movl (%rax), %edx
movl (%rdx, %rdi), %edx	salq \$2, %rdx	movl -80(%rbp), %eax
movslq %edx, %rdx	addq %rdx, %rax	cltq 0(%rax,8), %rcx
salq \$2, %rdx	movl (%rax), %ecx	leaq -56(%rbp), %rax
addq %rdx, %rax	movl -80(%rbp), %eax	addq %rcx, %rax
movl (%rax), %ecx	cltq	movq (%rax), %rax
movl -80(%rbp), %eax	leaq 0(%rax,8), %rdx	movl -88(%rbp), %ecx
cltq	movq -64(%rbp), %rax	movslq %ecx, %rcx
leaq 0(%rax,8), %rdx	addq %rdx, %rax	addq \$7, %rcx
movq -64(%rbp), %rax	movl -76(%rbp), %edx	salq \$2, %rcx
addq %rdx, %rax	movslq %edx, %rdx	addq %rcx, %rax
movq (%rax), %rax	salq \$2, %rdx	movl (%rax), %eax
movl -72(%rbp), %edx	addq %rdx, %rax	imull %edx, %eax
movslq %edx, %rdx	movl (%rax), %edx	addl %eax, -108(%rbp)
salq \$2, %rdx	movl -76(%rbp), %eax	addl \$8, -88(%rbp)
addq %rdx, %rax	cltq	addl \$1, -76(%rbp)
movl (%rax), %edx	leaq 0(%rax,8), %rsi	.L17:
movl -72(%rbp), %eax	movq -56(%rbp), %rax	movl -76(%rbp), %eax
cltq	addq %rsi, %rax	cmpl -68(%rbp), %eax
leaq 0(%rax,8), %rsi	movq (%rax), %rax	jl .L18
movq -56(%rbp), %rax	movl -72(%rbp), %esi	movl -136(%rbp), %edx
addq %rsi, %rax	movslq %esi, %rsi	movl -132(%rbp), %eax
movq (%rax), %rax	salq \$2, %rsi	addl %eax, %edx
movl -76(%rbp), %esi	addq %rsi, %rax	movl -128(%rbp), %eax
movslq %esi, %rsi	movl (%rax), %eax	addl %eax, %edx
salq \$2, %rsi	imull %eax, %edx	movl -124(%rbp), %eax
addq %rsi, %rax	movl -80(%rbp), %eax	addl %eax, %edx
movl (%rax), %eax	cltq	movl -120(%rbp), %eax
imull %eax, %edx	leaq 0(%rax,8), %rsi	addl %eax, %edx
movl -80(%rbp), %eax		

cltq		movq	-48(%rbp), %rax	movl	-116(%rbp), %eax
leaq	0(%rax,8), %rsi	addq	%rsi, %rax	addl	%eax, %edx
movq	-48(%rbp), %rax	movq	(%rax), %rax	movl	-112(%rbp), %eax
addq	%rsi, %rax	movl	-72(%rbp), %esi	addl	%eax, %edx
movq	(%rax), %rax	movslq	%esi, %rsi	movl	-108(%rbp), %eax
movl	-76(%rbp), %esi	salq	\$2, %rsi	addl	%edx, %eax
movslq	%esi, %rsi	addq	%rsi, %rax	movl	%eax, -140(%rbp)
salq	\$2, %rsi	addl	%ecx, %edx	movl	-84(%rbp), %eax
addq	%rsi, %rax	movl	%edx, (%rax)	cltq	
addl	%ecx, %edx	addl	\$1, -72(%rbp)	leaq	0(%rax,8), %rdx
movl	%edx, (%rax)	.L17:		movq	-48(%rbp), %rax
addl	\$1, -72(%rbp)	movl	-72(%rbp), %eax	addq	%rdx, %rax
.L17:		movq	(%rbx, %r11, 8), %rdi	movq	(%rax), %rax
movl	-72(%rbp), %eax	cmpl	%eax, -68(%rbp)	movl	-80(%rbp), %edx
cmpl	%eax, -68(%rbp)	ja	.L18	movslq	%edx, %rdx
ja	.L18	addl	\$1, -76(%rbp)	salq	\$2, %rdx
addl	\$1, -76(%rbp)	.L16:		addq	%rax, %rdx
.L16:		movl	-76(%rbp), %eax	movl	-140(%rbp), %eax
movl	-76(%rbp), %eax	cmpl	%eax, -68(%rbp)	movl	%eax, (%rdx)
cmpl	%eax, -68(%rbp)	ja	.L19	movl	-68(%rbp), %eax
xorl	%ebp, %ebp	addl	\$1, -80(%rbp)	sall	\$3, %eax
ja	.L19			movl	%eax, -88(%rbp)
addl	\$1, -80(%rbp)			jmp	.L19
				.L20:	
				movl	-80(%rbp), %eax
				cltq	
				leaq	0(%rax,8), %rdx
				movq	-56(%rbp), %rax
				addq	%rdx, %rax
				movq	(%rax), %rax
				movl	-88(%rbp), %edx
				movslq	%edx, %rdx
				salq	\$2, %rdx
				addq	%rdx, %rax
				movl	(%rax), %edx
				movl	-84(%rbp), %eax
				cltq	
				leaq	0(%rax,8), %rcx
				movq	-64(%rbp), %rax
				addq	%rcx, %rax
				movq	(%rax), %rax
				movl	-88(%rbp), %ecx
				movslq	%ecx, %rcx
				salq	\$2, %rcx
				addq	%rcx, %rax
				movl	(%rax), %eax
				imull	%edx, %eax
				addl	%eax, -140(%rbp)
				addl	\$1, -88(%rbp)

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: figura1-original.c

```

Actividades Editor de textos dom 13:00 ● figura1.c figura1-modificado_b.c figura1-modificado_a.c figura1-modificado_c.c
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<time.h>
4 #include <omp.h>
5
6
7 struct{
8     int a;
9     int b;
10}s[5000];
11
12 int main(int argc,char**argv)
13{
14
15    double tiempo_antes, tiempo_despues, tiempo_ejecucion;
16    int ii, i, X1, X2;
17    int R[40000];
18
19
20
21    //Realiza las cuentas y se mide el tiempo antes y despues de hacerlo para calcular el tiempo de ejecucion
22    tiempo_antes = omp_get_wtime();
23
24
25    for (ii=0; ii<40000;ii++) {
26        X1=0; X2=0;
27        for(i=0; i<5000;i++)
28            X1+=2*s[i].a+ii;
29        for(i=0; i<5000;i++)
30            X2+=3*s[i].b-ii;
31
32        if(X1<X2)  R[ii]=X1;  else  R[ii]=X2;
33    }
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48 }
49

```

```

Actividades Editor de textos dom 13:00 ● figura1.c figura1-modificado_b.c figura1-modificado_a.c figura1-modificado_c.c
14 int main(int argc,char**argv)
15 {
16
17    double tiempo_antes, tiempo_despues, tiempo_ejecucion;
18    int ii, i, X1, X2;
19    int R[40000];
20
21
22    //Realiza las cuentas y se mide el tiempo antes y despues de hacerlo para calcular el tiempo de ejecucion
23    tiempo_antes = omp_get_wtime();
24
25    for (ii=0; ii<40000;ii++) {
26        X1=0; X2=0;
27        for(i=0; i<5000;i++)
28            X1+=2*s[i].a+ii;
29        for(i=0; i<5000;i++)
30            X2+=3*s[i].b-ii;
31
32        if(X1<X2)  R[ii]=X1;  else  R[ii]=X2;
33    }
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48 }
49

```

```

Actividades Terminal dom 12:27 ● xavivl2000@xavivl2000: ~
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
xavivl2000@xavivl2000: ~ c3estudiante18@atcgrid:~
JavierRamirezPulido> dom may 17 gcc -fopenmp -O2 -o figura1 figura1.c
JavierRamirezPulido> dom may 17 ./figura1
R[0]: 0
R[39999]: -199995000
Tiempo: 0.613808637 segundos
JavierRamirezPulido> dom may 17

```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación–: Repito el desenrollado de bucle del código original en dos bucles

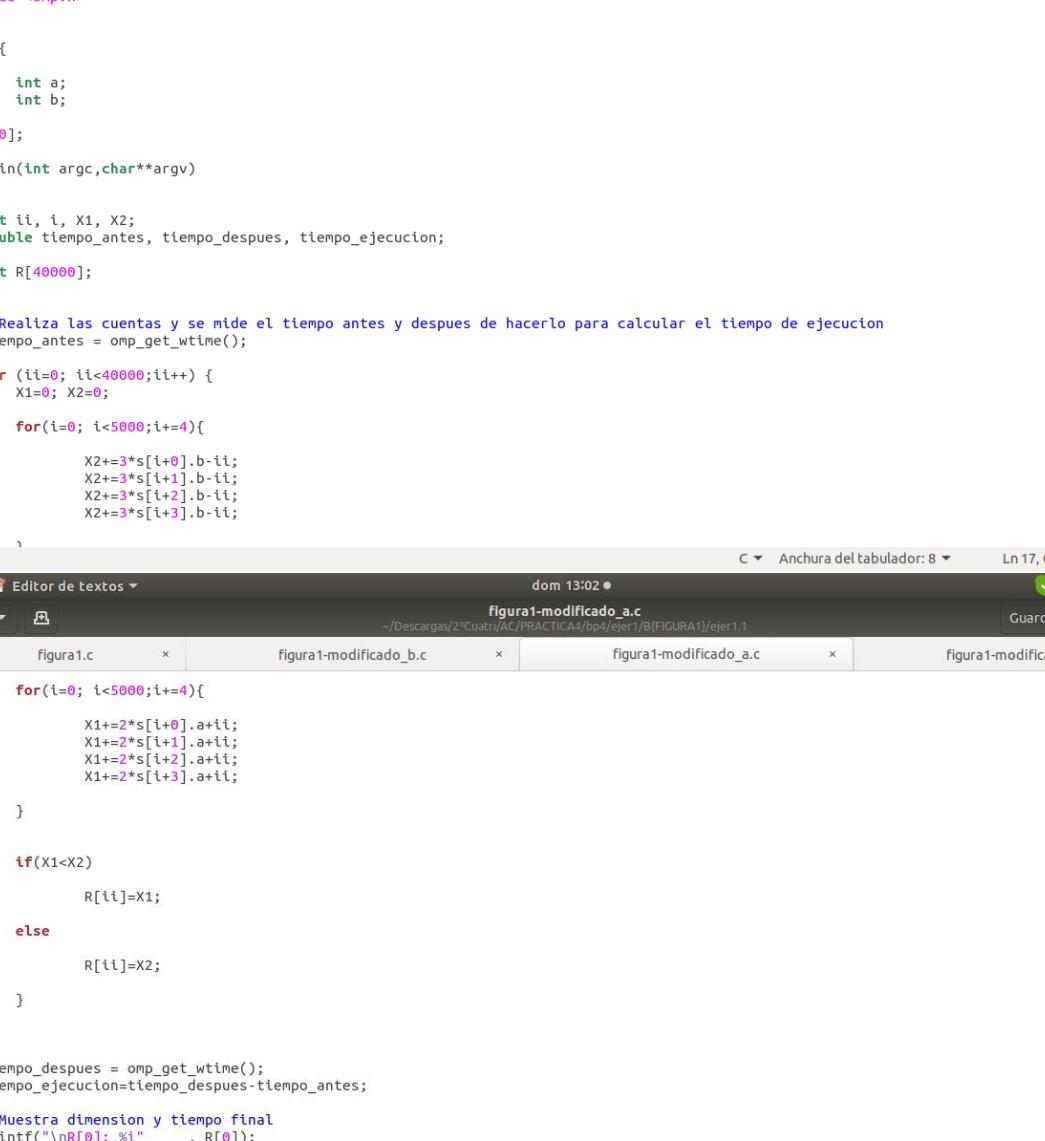
independientes entre si

Modificación b) –explicación-: Nuevo desenrollado pero en un único bucle de 8 iteraciones como en el ejercicio anterior.

Modificación c) -explicación-: Supresión de uno de los `for` de 40000 iteraciones uniéndolo con el otro.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1-modificado_a.c



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

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

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

    //Realiza las cuentas y se mide el tiempo antes y despues de hacerlo para calcular el tiempo de ejecucion
    tiempo_anteriores = omp_get_wtime();

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

        for(i=0; i<5000;i+=4){

            X2+=3*s[i+0].b-ii;
            X2+=3*s[i+1].b-ii;
            X2+=3*s[i+2].b-ii;
            X2+=3*s[i+3].b-ii;
        }

        if(X1<X2)

            R[ii]=X1;

        else

            R[ii]=X2;
    }

    tiempo_despues = omp_get_wtime();
    tiempo_ejecucion=tiempo_despues-tiempo_anteriores;

    //Muestra dimension y tiempo final
    printf("\nR[0]: %i", R[0]);
    printf("\nR[39999]: %i", R[39999]);
    printf("\nTiempo: %.11f segundos\n", tiempo_ejecucion);

    return 0;
}
```

```

Actividades Terminal
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
xavivi2000@xavivi2000: ~ x c3estud
JavierRamirezPulido> dom may 17 ./figura1-modificado_a
R[0]: 0
R[39999]: -199995000
Tiempo: 0.426226166 segundos
JavierRamirezPulido> dom may 17 ./figura1-modificado_a
R[0]: 0
R[39999]: -199995000
Tiempo: 0.426610896 segundos
JavierRamirezPulido> dom may 17

```

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

b) Captura figura1-modificado_b.c

```

Actividades Editor de textos
dom 13:03 ● figura1-modificado_b.c -/Descargas/2ºCuatr/A/C/PRACTICA4/bp4/ejer1/B(FIGURA1)/ejer1.i Guardar
figura1.c figura1-modificado_b.c figura1-modificado_a.c figura1-modificado_c.c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include <omp.h>

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

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

    //Realiza las cuentas y se mide el tiempo antes y despues de hacerlo para calcular el tiempo de ejecucion
    tiempo_antes = omp_get_wtime();

    for (ii=0; ii<40000;ii++) {
        X1=0; X2=0;
        for(i=0; i<5000; i+=10){

            X1+=2*s[i+0].a+ii; X2+=3*s[i+0].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;
            X1+=2*s[i+4].a+ii; X2+=3*s[i+4].b-ii;
            X1+=2*s[i+5].a+ii; X2+=3*s[i+5].b-ii;
            X1+=2*s[i+6].a+ii; X2+=3*s[i+6].b-ii;
            X1+=2*s[i+7].a+ii; X2+=3*s[i+7].b-ii;
            X1+=2*s[i+8].a+ii; X2+=3*s[i+8].b-ii;
            X1+=2*s[i+9].a+ii; X2+=3*s[i+9].b-ii;

        }

        if(X1>=X2)

            R[ii]=X2;

        else

            R[ii]=X1;

    }

    tiempo_despues = omp_get_wtime();
    tiempo_ejecucion=tiempo_despues-tiempo_antes;

    //Muestra dimension y tiempo final
    printf("\nR[0]: %i", R[0]);
    printf("\nR[39999]: %i", R[39999]);
    printf("\nTiempo: %11.9f segundos\n\n", tiempo_ejecucion);

    return 0;
}

```

```

Actividades Terminal
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
xavivi2000@xavivi2000: ~
JavierRamirezPulido> dom may 17 ./figura1-modificado_b
R[0]: 0
R[39999]: -199995000
Tiempo: 0.356780956 segundos
JavierRamirezPulido> dom may 17 ./figura1-modificado_b
R[0]: 0
R[39999]: -199995000
Tiempo: 0.356479538 segundos
JavierRamirezPulido> dom may 17

```

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

c) Captura figura1-modificado_c.c

```

Actividades Editor de textos
dom 13:04 ● figura1-modificado_c.c
~/Descargas/2º Cuatri/A/C/PRACTICA4/bp4/ejer1/B(FIGURA1)/ejer1.1
Guarda
figura1.c figura1-modificado_b.c figura1-modificado_a.c figura1-modificado_c.c

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

struct{
    int a;
    int b;
}s[5000];
int main(int argc,char**argv)
{
    double tiempo_antes, tiempo_despues, tiempo_ejecucion;
    int ii, i, X1, X2;
    int R[40000];

    //Realiza las cuentas y se mide el tiempo antes y despues de hacerlo para calcular el tiempo de ejecucion
    tiempo_antes = omp_get_wtime();

    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;
    }

    tiempo_despues = omp_get_wtime();
    tiempo_ejecucion=tiempo_despues-tiempo_antes;

    //Muestra dimension y tiempo final
    printf("\nR[0]: %i ", R[0]);
    printf("\nR[39999]: %i ", R[39999]);
    printf("\nTiempo: %.11f segundos\n", tiempo_ejecucion);

    return 0;
}

```

```

Actividades Terminal
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
xavivi2000@xavivi2000: ~ x c3estudian
JavierRamirezPulido> dom may 17 ./figura1-modificado_c
R[0]: 0
R[39999]: -199995000
Tiempo: 0.473541314 segundos

JavierRamirezPulido> dom may 17 ./figura1-modificado_c
R[0]: 0
R[39999]: -199995000
Tiempo: 0.474453878 segundos

JavierRamirezPulido> dom may 17

```

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

1.1. TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	-	0.615334632
Modificación a)		0.427277567
Modificación b)		0.357324516
Modificacion c)		0.477465568

```

Actividades Terminal
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
xavivi2000@xavivi2000: ~ x c3estudian
JavierRamirezPulido> dom may 17 ./figura1
R[0]: 0
R[39999]: -199995000
Tiempo: 0.615334632 segundos

JavierRamirezPulido> dom may 17 ./figura1-modificado_a
R[0]: 0
R[39999]: -199995000
Tiempo: 0.427277567 segundos

JavierRamirezPulido> dom may 17 ./figura1-modificado_b
R[0]: 0
R[39999]: -199995000
Tiempo: 0.357324516 segundos

JavierRamirezPulido> dom may 17 ./figura1-modificado_c
R[0]: 0
R[39999]: -199995000
Tiempo: 0.477465568 segundos

JavierRamirezPulido> dom may 17

```

1.1. COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:

Con todas y cada una de las modificaciones conseguimos reducir el tiempo de ejecución. La mayor diferencia se da al hacer el desenrollado en el mismo número de iteraciones que en el ejercicio anterior. Por otra parte, con las modificaciones, en un principio realmente diferentes, de unir bucles y desenrollarlo en dos, se obtienen tiempos similares pero igualmente menores que los originales. Se puede interpretar la modificación b como una mezcla de las otras por el desarrollo de las 8 iteraciones pero con el for unido.

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.(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

figura1.c	figura1-modificado_a.c	figura1-modificado_b.c
.L4:	.L4:	.L4:
movl -160052(%rbp), %eax	movl -160052(%rbp), %eax	movl -160052(%rbp), %eax
cltq	cltq	cltq
leaq 0(%rax,8), %rdx	leaq 0(%rax,8), %rdx	leaq 0(%rax,8), %rdx
leaq s(%rip), %rax	leaq 4+s(%rip), %rax	leaq s(%rip), %rax
movl (%rdx,%rax), %eax	movl (%rdx,%rax), %edx	movl (%rdx,%rax), %eax
leal (%rax,%rax), %edx	movl %edx, %eax	leal (%rax,%rax), %edx
movl -160056(%rbp), %eax	movl %eax, %eax	movl -160056(%rbp), %eax
addl %edx, %eax	addl %eax, %eax	addl %edx, %eax
addl %eax, -160048(%rbp)	addl %edx, %eax	addl %eax, -160048(%rbp)
addl \$1, -160052(%rbp)	subl -160056(%rbp), %eax	movl -160052(%rbp), %eax
.L3:	.L3:	.L3:
Cmpl \$4999, -160052(%rbp)	movl -160052(%rbp), %eax	cltq
jle .L4	addl \$1, %eax	leaq 0(%rax,8), %rdx
movl \$0, -160052(%rbp)	cltq	leaq 4+s(%rip), %rax
jmp .L5	leaq 0(%rax,8), %rdx	movl (%rdx,%rax), %edx
.L6:	leaq 4+s(%rip), %rax	addl %eax, %eax
movl -160052(%rbp), %eax	movl (%rdx,%rax), %edx	subl -160056(%rbp), %eax
cltq	movl %edx, %eax	addl %eax, -160044(%rbp)
leaq 0(%rax,8), %rdx	addl %eax, %eax	movl -160052(%rbp), %eax
leaq 4+s(%rip), %rax	subl -160056(%rbp), %eax	addl \$1, %eax
movl (%rdx,%rax), %edx	addl %eax, -160044(%rbp)	cltq
movl %edx, %eax	movl -160052(%rbp), %eax	leaq 0(%rax,8), %rdx
addl %eax, %eax	addl \$2, %eax	leaq s(%rip), %rax
addl %edx, %eax	cltq	movl (%rdx,%rax), %eax
cmpq \$s+40000, %rax	leaq 4+s(%rip), %rax	addl (%rax,%rax), %edx
subl -160056(%rbp), %eax	movl (%rdx,%rax), %edx	movl -160056(%rbp), %eax
addl %eax, -160044(%rbp)	movl %edx, %eax	addl %eax, -160044(%rbp)
addl \$1, -160052(%rbp)	addl %eax, -160044(%rbp)	movl -160052(%rbp), %eax
.L5:	.L5:	.L5:
cmpl \$4999, -160052(%rbp)	leaq 0(%rax,8), %rdx	cltq
jle .L6	leaq 4+s(%rip), %rax	leaq 0(%rax,8), %rdx
movl -160048(%rbp), %eax	movl (%rdx,%rax), %edx	leaq s(%rip), %rax
xorl %ecx, %ecx	movl %edx, %eax	movl (%rdx,%rax), %eax
.p2align 4, ,10	addl %eax, %eax	addl (%rax,%rax), %edx
.p2align 3	addl %edx, %eax	movl -160056(%rbp), %eax
jge .L7	subl -160056(%rbp), %eax	addl %eax, -160044(%rbp)
movl -160056(%rbp), %eax	addl %eax, -160044(%rbp)	movl -160052(%rbp), %eax
addq \$8, %rax	movl -160052(%rbp), %eax	addl \$1, %eax
leal (%rdx, %rdx, 2), %edx	addl \$3, %eax	cltq
subl %edi, %edx	cltq	leaq 0(%rax,8), %rdx
cmpq %rax, %r8	leaq 4+s(%rip), %rax	leaq 4+s(%rip), %rax
jne .L5	movl (%rdx,%rax), %edx	movl (%rdx,%rax), %edx
cltq	leaq 0(%rax,8), %rdx	addl %eax, %eax
movl -160048(%rbp), %edx	leaq 4+s(%rip), %rax	addl %edx, %eax
movl %edx, -	movl (%rdx,%rax), %edx	addl %eax, %eax
160016(%rbp,%rax,4)	movl %edx, %eax	subl -160056(%rbp), %eax
jmp .L8	addl %eax, %eax	addl %eax, -160044(%rbp)
.L7:	subl -160056(%rbp), %eax	movl -160052(%rbp), %eax
movl -160056(%rbp), %eax	addl \$4, -160052(%rbp)	addl \$2, %eax
cltq	cltq	cltq
movl -160044(%rbp), %edx	leaq 0(%rax,8), %rdx	leaq 0(%rax,8), %rdx
movl %edx, -	leaq s(%rip), %rax	leaq s(%rip), %rax
160016(%rbp,%rax,4)	cmpl \$4999, -160052(%rbp)	movl (%rdx,%rax), %eax
.L8:	jle .L4	leal (%rax,%rax), %edx
	movl \$0, -160052(%rbp)	movl -160056(%rbp), %eax

cltq	
leaq	0(%rax,8), %rdx
leaq	s(%rip), %rax
movl	(%rdx,%rax), %eax
leal	(%rax,%rax), %edx
movl	-160056(%rbp), %eax
addl	%edx, %eax
addl	%eax, -160048(%rbp)
movl	-160052(%rbp), %eax
addl	\$5, %eax
cltq	
leaq	0(%rax,8), %rdx
leaq	4+s(%rip), %rax
movl	(%rdx,%rax), %edx
movl	%edx, %eax
addl	%eax, %eax
addl	%edx, %eax
subl	-160056(%rbp), %eax
addl	%eax, -160044(%rbp)
movl	-160052(%rbp), %eax
addl	\$6, %eax
cltq	
leaq	0(%rax,8), %rdx
leaq	s(%rip), %rax
movl	(%rdx,%rax), %eax
leal	(%rax,%rax), %edx
movl	-160056(%rbp), %eax
addl	%edx, %eax
addl	%eax, -160048(%rbp)
movl	-160052(%rbp), %eax
addl	\$6, %eax
cltq	
leaq	0(%rax,8), %rdx
leaq	4+s(%rip), %rax
movl	(%rdx,%rax), %edx
movl	%edx, %eax
addl	%eax, %eax
addl	%edx, %eax
subl	-160056(%rbp), %eax
addl	%eax, -160044(%rbp)
movl	-160052(%rbp), %eax
addl	\$7, %eax
cltq	
leaq	0(%rax,8), %rdx
leaq	s(%rip), %rax
movl	(%rdx,%rax), %eax
leal	(%rax,%rax), %edx
movl	-160056(%rbp), %eax
addl	%edx, %eax
addl	%eax, -160048(%rbp)
movl	-160052(%rbp), %eax
addl	\$7, %eax
cltq	
leaq	0(%rax,8), %rdx
leaq	4+s(%rip), %rax
movl	(%rdx,%rax), %edx
movl	%edx, %eax

addl	%eax, %eax
addl	%edx, %eax
subl	-160056(%rbp), %eax
addl	%eax, -160044(%rbp)
movl	-160052(%rbp), %eax
addl	\$8, %eax
cltq	
leaq	0(%rax,8), %rdx
leaq	s(%rip), %rax
movl	(%rdx,%rax), %eax
leal	(%rax,%rax), %edx
movl	-160056(%rbp), %eax
addl	%edx, %eax
addl	%eax, -160048(%rbp)
movl	-160052(%rbp), %eax
addl	\$8, %eax
cltq	
leaq	0(%rax,8), %rdx
leaq	4+s(%rip), %rax
movl	(%rdx,%rax), %edx
movl	%edx, %eax
addl	%eax, %eax
addl	%edx, %eax
subl	-160056(%rbp), %eax
addl	%eax, -160044(%rbp)
movl	-160052(%rbp), %eax
addl	\$9, %eax
cltq	
leaq	0(%rax,8), %rdx
leaq	s(%rip), %rax
movl	(%rdx,%rax), %eax
leal	(%rax,%rax), %edx
movl	-160056(%rbp), %eax
addl	%edx, %eax
addl	%eax, -160048(%rbp)
movl	-160052(%rbp), %eax
addl	\$9, %eax
cltq	
leaq	0(%rax,8), %rdx
leaq	4+s(%rip), %rax
movl	(%rdx,%rax), %edx
movl	%edx, %eax
addl	%eax, %eax
addl	%edx, %eax
subl	-160056(%rbp), %eax
addl	%eax, -160044(%rbp)
addl	\$10, -160052(%rbp)
jne	.L4
cmpl	%edx, %esi
cmovl	%esi, %edx
addq	\$1, %r11
.L3:	
cmpl	\$4999, -160052(%rbp)
jle	.L4
movl	-160048(%rbp), %eax
cmpl	-160044(%rbp), %eax
jl	.L5

movl	-160056(%rbp), %eax
cltq	
movl	-160044(%rbp), %edx
movl	%edx, -160016(%rbp,%rax,4)
jmp	.L6
.L5:	
movl	-160056(%rbp), %eax
cltq	
movl	-160048(%rbp), %edx
movl	%edx, -160016(%rbp,%rax,4)
.L6:	
addl	\$1, -160056(%rbp)

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 que opera con flotantes de doble precisión 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 acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos. Sólo se debe evaluar el tiempo del núcleo DAXPY

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 y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CAPTURA CÓDIGO FUENTE: daxpy.c

The screenshot shows a Linux desktop environment with a dark theme. In the top window, titled "Actividades Editor de textos", a C program named "daxpy.c" is displayed. The code implements a simple matrix-vector multiplication using OpenMP for parallelization. It includes checks for argument correctness, variable declarations, vector filling, and a function to perform the computation. The bottom window, also titled "Actividades Terminal", shows the command-line interface where the program is compiled with "gcc -fopenmp" and run with dimensions of 200 and 5. The terminal output displays the dimension, initial values of arrays x and y, and the execution time.

```

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

main(int argc,char**argv)
{
    //Comprueba que la cantidad de argumentos es la correcta
    if(argc!=3){
        printf("\nFalta el numero de elementos del primera_matriz\n");
        exit(-1);
    }

    //Declaracion de variables
    double tiempo_antes, tiempo_despues, tiempo_ejecucion;

    /*
     * Nucleo original
     */
    double *x, *y, a= atoi(argv[2]);
    unsigned int N = atoi(argv[1]) ;

    //Reserva para vectores
    x = (double*) malloc(N*sizeof(double));
    y = (double*) malloc(N*sizeof(double));

    //Rellenar vectores
    for(int i=0; i<N; i++){
        x[i] = i*4 ;
        y[i] = i+3;
    }

    /* Función a optimizar */
    inline void daxpy (){
        int i;

        // hacemos la multiplicación
        for (i=0; i<N; i++){
            y[i] = a*x[i] + y[i];
        }
    }

    //Realiza las cuentas y se mide el tiempo antes y despues de hacerlo para calcular el tiempo de ejecución
    tiempo_antes = omp_get_wtime();

    daxpy();

    tiempo_despues = omp_get_wtime();
    tiempo_ejecucion=tiempo_despues-tiempo_antes;

    //Muestra dimension y tiempo final
    printf("\nDimension: %d", N);
    printf("\ny[0]: %f ", y[0]);
    printf("\ny[%d]: %f ", N-1, y[N-1]);
    printf("\nTiempo: %11.9f segundos\n\n", tiempo_ejecucion);

    return 0;
}

```

```

xavier2000@xavier2000: ~ xavivl2000@xavivl2000: ~
JavierRamirezPulido> dom may 17 gcc -fopenmp -o daxpy daxpy.c
JavierRamirezPulido> dom may 17 ./daxpy 200 5
Dimension: 200
y[0]: 3.000000
y[199]: 4182.000000
Tiempo: 0.000013297 segundos
JavierRamirezPulido> dom may 17 gcc -fopenmp -O2 -o daxpy daxpy.c
JavierRamirezPulido> dom may 17 ./daxpy 200 5
Dimension: 200
y[0]: 3.000000
y[199]: 4182.000000
Tiempo: 0.000001672 segundos
JavierRamirezPulido> dom may 17 []

```

```

Actividades Terminal dom 17:5
xavivi2000@xavivi2000: ~ x c3estudiante18@atcgri: ~
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -O0 -o daxpy0 daxpy.c
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -Os -o daxpy0 daxpy.c
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -O2 -o daxpy2 daxpy.c
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -O3 -o daxpy3 daxpy.c
JavierRamirezPulido> dom may 17 ./daxpy0 200 5
Dimension: 200
y[0]: 3.000000
y[199]: 4182.000000
Tiempo: 0.000007778 segundos
JavierRamirezPulido> dom may 17 ./daxpy0 200 5
Dimension: 200
y[0]: 3.000000
y[199]: 4182.000000
Tiempo: 0.000007778 segundos
JavierRamirezPulido> dom may 17 ./daxpy0 200 5
Dimension: 200
y[0]: 3.000000
y[199]: 4182.000000
Tiempo: 0.000001395 segundos
JavierRamirezPulido> dom may 17 ./daxpy2 200 5
Dimension: 200
y[0]: 3.000000
y[199]: 4182.000000
Tiempo: 0.000001762 segundos
JavierRamirezPulido> dom may 17 ./daxpy3 200 5
Dimension: 200
y[0]: 3.000000
y[199]: 4182.000000
Tiempo: 0.000002303 segundos
JavierRamirezPulido> dom may 17

```

	-O0	-Os	-O2	-O3
Tiempos ejec.	0.000007778	0.000001395	0.000001762	0.000002303

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

```

xavivi2000@xavivi2000: ~ x c3estudiante18@atcgri: ~
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -O0 -o daxpy0 daxpy.c
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -Os -o daxpy0 daxpy.c
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -O2 -o daxpy2 daxpy.c
JavierRamirezPulido> dom may 17 gcc -lrt -fopenmp -O3 -o daxpy3 daxpy.c
JavierRamirezPulido> dom may 17

```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Las principales diferencias en la compilación con una optimización u otra es el uso de las direcciones de datos. Estas pueden ser relativas a los registros cuando compilamos con la optimización O2, a diferencia de la opción O0 que usa las relativas a pila. Lo que permite compilar con O2 es ahorrar muchas operaciones innutiles. La opción -O3 desenrolla el bucle , que provoca un ensamblador notablemente mas largo y por tanto mas lento que la opción 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)

daxpy00.s	daxpy0s.s	daxpy02.s	daxpy03.s
-----------	-----------	-----------	-----------

movl	\$0, -4(%rbp)	.L9:		movsd	(%rsp), %xmm1	movsd	%xmm0,
jmp	.L2	call		xorl	%ebx, %ebx	0(%r13,%rsi,8)	
.L3:			omp_get_wtim	movq	%rdx, %r12	pxor	
movl	-4(%rbp), %edx	e@PLT		addq	\$1, %rdx	0	%xmm0, %xmm
movslq	%edx, %rdx	movsd	%xmm0,	.p2align 4,,10		cvttsi2sd	%edx, %xmm0
leaq	0(%rdx,8), %rcx	8(%rsp)		.p2align 3		movsd	%xmm0,
movq	16(%rax), %rdx	xorl	%eax, %eax	.L4:		jbe	.L8
addq	%rcx, %rdx	.L5:		pxor	%xmm0, %xmm0	pxor	
movsd	(%rdx), %xmm0	cmpl	%eax, %ebx	leal	0(%rbx,4), %eax	0(%rbp,%rsi,8)	
movsd		jbe	.L10	cvttsi2sd	%eax, %xmm0	pxor	
	8(%rax), %xmm	movsd	(%rsp), %xmm0	leal	3(%rbx), %eax	0	%xmm0, %xmm
1		mulsd		movsd	%xmm0,	movslq	%ecx, %rdx
mulsd			(%r12,%rax,8), %xmm0	pxor	%xmm0, %xmm0	sall	\$2, %ecx
0	%xmm1, %xmm	addsd	0(%rbp,%rax,8)	cvttsi2sd	%eax, %xmm0	addl	\$6, %eax
movl	-4(%rbp), %edx	, %xmm0		movsd	%xmm0,	cvttsi2sd	%ecx, %xmm0
movslq	%edx, %rdx	movsd	%xmm0,	0(%r13,%rbx,8)		movsd	%xmm0,
leaq	0(%rdx,8), %rcx	0(%rbp,%rax,8)		addq	\$1, %rbx	0(%r13,%rdx,8)	
movq	(%rax), %rdx	incq	%rax	cmpq	%rdx, %rbx	pxor	
addq	%rcx, %rdx	jmp	.L5	jne	.L4	0	%xmm0, %xmm
movsd	(%rdx), %xmm1	.L10:		movsd	%xmm1, (%rsp)	movsd	
movl	-4(%rbp), %edx	call		salq	\$3, %rbx	cvttsi2sd	%eax, %xmm0
movslq	%edx, %rdx		omp_get_wtim	call		movsd	%xmm0,
leaq	0(%rdx,8), %rcx	e@PLT		omp_get_wtime		0(%rbp,%rdx,8)	
movq	(%rax), %rdx	leaq	.LC1(%rip), %rsi	@PLT		.L8:	
addq	%rcx, %rdx	movl	%ebx, %edx	movsd	(%rsp), %xmm1	call	
addsd		movl	\$1, %edi	xorl	%eax, %eax	omp_get_wtime	
0	%xmm1, %xmm	xorl	%eax, %eax	movsd	%xmm0, 8(%rsp)	@PLT	
movsd		movsd	%xmm0, (%rsp)	.p2align 4,,10		movq	%rbp, %rcx
addl	%xmm0, (%rdx)			.p2align 3		movsd	%xmm0,
.L2:	\$1, -4(%rbp)			.L5:		8(%rsp)	
movl	-4(%rbp), %edx			movsd		shrq	\$3, %rcx
movl	24(%rax), %e				(%r14,%rax), %xm	andl	\$1, %ecx
				m0		cmpl	\$1, %ebx
				mulsd	%xmm1, %xmm0	jbe	.L17
				addsd		.L14:	
					0(%r13,%rax), %x	xorl	%esi, %esi
						testl	%ecx, %ecx
				mm0	%xmm0,	je	.L10
				movsd	0(%r13,%rax)	movsd	(%rsp), %xmm0
				addq	\$8, %rax	movl	\$1, %esi
				cmpq	%rax, %rbx	mulsd	
				jne	.L5	0	0(%r13), %xmm
				call		addsd	
					omp_get_wtime	0	0(%rbp), %xmm
					@PLT	0	%xmm0,
					leaq	.LC1(%rip), %rsi	movsd
					movl	%ebp, %edx	0(%rbp)
					movl	\$1, %edi	.L10:
					xorl	%eax, %eax	movsd
							(%rsp), %xmm1
							%r12d, %r9d
							%eax, %eax
							%ecx, %r9d
							subl
							\$3, %rcx
							salq
							xorl
							unpcklpd
							%xmm1, %xmm
						1	
						movl	%r9d, %r8d
						leaq	0(%r13,%rcx), %
						rdi	
						shr1	%r8d

		addq	%rbp, %rcx
		.p2align 4,,10	
		.p2align 3	
		.L11:	
		movupd	(%rdi,%rax), %x
		mm0	
		addl	\$1, %edx
		mulpd	%xmm1, %xmm
		0	
		addpd	(%rcx,%rax), %x
		mm0	
		movaps	%xmm0,
		(%rcx,%rax)	
		addq	\$16, %rax
		cmpl	%r8d, %edx
		jb	.L11
		movl	%r9d, %edx
		andl	\$-2, %edx
		cmpl	%edx, %r9d
		leal	(%rdx,%rsi), %ea
		x	
		je	.L13
		.L9:	
		movslq	%eax, %rcx
		movsd	(%rsp), %xmm5
		movsd	0(%r13,%rcx,8),
		%xmm0	
		leaq	0(%rbp,%rcx,8),
		%rdx	
		addl	\$1, %eax
		mulsd	%xmm5, %xmm
		0	
		cmpl	%r12d, %eax
		addsd	(%rdx), %xmm0
		movsd	%xmm0, (%rdx)
		jnb	.L13
		cltq	
		mulsd	0(%r13,%rax,8),
		%xmm5	
		leaq	0(%rbp,%rax,8),
		%rdx	
		movsd	(%rdx), %xmm0
		addsd	%xmm5, %xmm
		0	
		movsd	%xmm0, (%rdx)
		.L13:	
		call	omp_get_wtime
		@PLT	
		leaq	.LC6(%rip), %rsi
		movl	%r12d, %edx
		movl	\$1, %edi
		xorl	%eax, %eax

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 y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

```

xavivi2000@xavivi2000:~ xavivi2000@xavivi2000:~ xavivi2000@xavivi2000:~
daxpy.c daxpy daxpy_tiempos daxpy_tiempos.c script.sh
JavierRamirezPulido> dom may 17 ./script.sh
Dimension: 100      y[0]: 3.000000      y[99]: 39702.000000      Tiempo: 0.000001702 segundos
Dimension: 80100     y[0]: 3.000000      y[80099]: 25663799702.000000      Tiempo: 0.000484632 segundos
Dimension: 160100    y[0]: 3.000000      y[160099]: 102527559702.000000      Tiempo: 0.001172595 segundos
Dimension: 240100    y[0]: 3.000000      y[240099]: 230591319702.000000      Tiempo: 0.001297491 segundos
Dimension: 320100    y[0]: 3.000000      y[320099]: 409855079702.000000      Tiempo: 0.001937882 segundos
Dimension: 400100    y[0]: 3.000000      y[400099]: 640318839702.000000      Tiempo: 0.002292728 segundos
Dimension: 480100    y[0]: 3.000000      y[480099]: 921982599702.000000      Tiempo: 0.002983375 segundos
Dimension: 560100    y[0]: 3.000000      y[560099]: 1254846359702.000000      Tiempo: 0.004693056 segundos
Dimension: 640100    y[0]: 3.000000      y[640099]: 1638910119702.000000      Tiempo: 0.004368452 segundos
Dimension: 720100    y[0]: 3.000000      y[720099]: 2074173879702.000000      Tiempo: 0.005258771 segundos
Dimension: 800100    y[0]: 3.000000      y[800099]: 2560637639702.000000      Tiempo: 0.005122181 segundos
Dimension: 880100    y[0]: 3.000000      y[880099]: 3098301399702.000000      Tiempo: 0.004795139 segundos
Dimension: 960100    y[0]: 3.000000      y[960099]: 3687165159702.000000      Tiempo: 0.005559541 segundos
Dimension: 1040100   y[0]: 3.000000      y[1040099]: 4327228919702.000000      Tiempo: 0.007735075 segundos
Dimension: 1120100   y[0]: 3.000000      y[1120099]: 5018492679702.000000      Tiempo: 0.005860245 segundos
Dimension: 1200100   y[0]: 3.000000      y[1200099]: 5760956439702.000000      Tiempo: 0.006474130 segundos
JavierRamirezPulido> dom may 17 clear
JavierRamirezPulido> dom may 17

```

Hay 1 instrucción en coma flotante fuera del bucle y 4 instrucciones dentro del bucle en daxpyOs

$$\text{Para } N = 100 : R = (1 + 4 * 100) / 0.000001005 = 399003958.996080$$

$$\text{Para } N = 80100 : R = (1 + 4 * 80100) / 0.000500850 = 639714488.594528$$

$$\text{Para } N = 160100 : R = (1 + 4 * 160100) / 0.000816060 = 784747444.133746$$

$$\text{Para } N = 240100 : R = (1 + 4 * 240100) / 0.001285793 = 746932825.829579$$

$$\text{Para } N = 320100 : R = (1 + 4 * 320100) / 0.002182208 = 586745626.014210$$

$$\text{Para } N = 400100 : R = (1 + 4 * 400100) / 0.002092245 = 764920456.807019$$

$$\text{Para } N = 480100 : R = (1 + 4 * 480100) / 0.002579603 = 744456026.116557$$

$$\text{Para } N = 560100 : R = (1 + 4 * 560100) / 0.002985713 = 750373863.957809$$

$$\text{Para } N = 640100 : R = (1 + 4 * 640100) / 0.003383220 = 756794119.481302$$

$$\text{Para } N = 720100 : R = (1 + 4 * 720100) / 0.004363043 = 660181667.197252$$

$$\text{Para } N = 800100 : R = (1 + 4 * 800100) / 0.004771095 = 670789619.282663$$

$$\text{Para } N = 880100 : R = (1 + 4 * 880100) / 0.005238024 = 672085694.744001$$

$$\text{Para } N = 960100 : R = (1 + 4 * 960100) / 0.005963925 = 643938513.713801$$

Para N = 1040100 : R = $(1 + 4 * 1040100) / 0.005393648 = 771351968.109847$

Para N = 1120100 : R = $(1 + 4 * 1120100) / 0.005782793 = 774781494.057904$

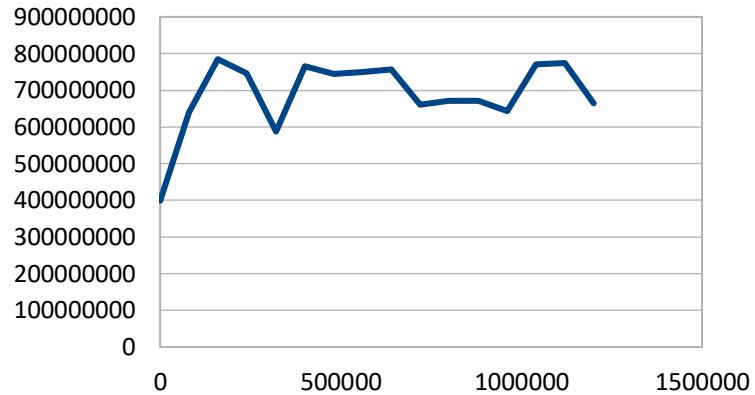
Para N = 1200100 : R = $(1 + 4 * 1200100) / 0.007230848 = 663878012.871889$

$$R_{\max} = 784747444.133746$$

$$N_{\max} = 160100$$

$$R_{1/2} = 392373722.1$$

$$N_{1/2} = 160100$$



Tamaño	R
100	399003959
80100	639714488,6
160100	784747444,1
240100	746932825,8
320100	586745626
400100	764920456,8
480100	744456026,1
560100	750373864
640100	756794119,5
720100	660181667,2
800100	670789619,3
880100	672085694,7
960100	643938513,7
1040100	771351968,1
1120100	774781494,1
1200100	663878012,9