

2º curso / 2º cuatr.
Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Miguel Ángel Fernández Gutiérrez

Grupo de prácticas y profesor de prácticas: GIM2, Francisco Barranco

Fecha de entrega: 31 de mayo, 2019

Fecha evaluación en clase: 30 de mayo, 2019

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) Core(TM) i7-7700HQ CPU @ 2.80GHz

Sistema operativo utilizado: Ubuntu 18.04.1 LTS “bionic”

Versión de gcc utilizada: gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0

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

```
mianfg : fish — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
mianfg ) lscpu
Arquitectura: 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: 158
Nombre del modelo: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Revisión: 9
CPU MHz: 800.183
CPU MHz máx.: 3800,0000
CPU MHz mín.: 800,0000
BogoMIPS: 5616.00
Virtualización: VT-x
Caché L1d: 32K
Caché L1i: 32K
Caché L2: 256K
Caché L3: 6144K
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 syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid
aperfperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic mo
vbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid_single pti ssbd ibrs ibpb s
tibrp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_
pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp md_clear flush_l1d
mianfg )
```

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 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

Nota: he tenido problemas con el software capturas de pantalla, así que voy a pegar el código directamente de aquí en adelante.

```
/**
 * @file pmm-secuencial.c
 * @brief Cálculo del producto de dos matrices - secuencial, var. globales
 * @author Miguel Ángel Fernández Gutiérrez <mianfg@correo.ugr.es>
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Vectores globales
#define MAX 10000
double A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];

int main(int argc, char ** argv){
    struct timespec cgt1,cgt2; double t;                // para tiempos
```

```

    if ( argc < 2 ) {
        printf("[ERROR]-Debe insertar tamaño matriz\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    if ( N > MAX )
        N = MAX;

    int i, j, k;
    double suma;

    // Inicialización matrices B, C
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < N; j++ ) {
            B[i][j]=j+1;
            C[i][j]=j+1;
        }

    // Tiempo
    clock_gettime(CLOCK_REALTIME,&cgt1);

    // Cálculo de A=B*C
    for ( i = 0; i < N ; i++ )
        for ( j = 0; j < N; j++ ) {
            A[i][j] = 0;
            for ( k = 0; k < N; k++ )
                A[i][j] = A[i][j]+B[i][k]*C[k]
        }

    // Tiempo
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    // Impresión de tiempo de ejecución
    printf("Tiempo (seg): %11.9f\n", t);

    // Impresión de resultados
    printf("\n_____ \nResultados:\n");

    printf("\nMatriz B: \n");
    if ( N < 10 )
        for ( i = 0; i < N; i++ ) {
            for ( j = 0; j < N; j++ )
                printf("%f ", B[i][j]);
            printf("\n");
        }
    else
        printf("B[0][0]=%f B[%d][%d]=%f\n", B[0][0], N-1, N-1,
B[N-1][N-1]);

```

```

        printf("\nMatriz C: \n");
        if ( N < 10 )
            for ( i = 0; i < N; i++ ) {
                for ( j = 0; j < N; j++ )
                    printf("%f ", C[i][j]);
                printf("\n");
            }
        else
            printf("C[0][0]=%f C[%d][%d]=%f\n", C[0][0], N-1, N-1,
C[N-1][N-1]);

        printf("\nMatriz A=B*C: \n");
        if ( N < 10 )
            for ( i = 0; i < N; i++ ) {
                for ( j = 0; j < N; j++ )
                    printf("%f ", A[i][j]);
                printf("\n");
            }
        else
            printf("A[0][0]=%f A[%d][%d]=%f\n", A[0][0], N-1, N-1,
A[N-1][N-1]);

        printf("\n");

        return 0;
}

```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación–: desenrollado de bucles, con 4 cálculos por iteración, teniendo en cuenta que el tamaño N no tiene por qué ser múltiplo de 4 (desenrollamos el for hasta $N-N\%4$ y luego calculamos de forma independiente, fuera del for, desde $N\%4$ hasta N).

Modificación b) –explicación–: cambio en el orden de ejecución: i,k,j en lugar de i,j,k, para disminuir las penalizaciones por caché.

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura de pmm-secuencial-modificado_a.c

```

/**
 * @file pmm-secuencial.c
 * @brief Cálculo del producto de dos matrices - secuencial, var. globales
 * @author Miguel Ángel Fernández Gutiérrez <mianfg@correo.ugr.es>
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Vectores globales
#define MAX 10000
double A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];

int main(int argc, char ** argv){
    struct timespec cgt1,cgt2; double t;                // para tiempos

    if ( argc < 2 ) {
        printf("[ERROR]-Debe insertar tamaño matriz\n");
        exit(-1);
    }
}

```

```

    }

    unsigned int N = atoi(argv[1]);
    if ( N > MAX )
        N = MAX;

    int i, j, k;
    double suma;

    // Inicialización matrices B, C
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < N; j++ ) {
            B[i][j]=j+1;
            C[i][j]=j+1;
        }

    int mod = N%4;

    // Tiempo
    clock_gettime(CLOCK_REALTIME,&cgt1);

    // Cálculo de A=B*C
    for ( i = 0; i < N ; i++ )
        for ( j = 0; j < N; j++ ) {
            A[i][j] = 0;
            for ( k = 0; k < N-mod; k+=4 ) {
                A[i][j] = A[i][j]+B[i][k]*C[k]
           ];
            A[i][j] = A[i][j]+B[i]
            [k+1]*C[k+1][j];
            A[i][j] = A[i][j]+B[i]
            [k+2]*C[k+2][j];
            A[i][j] = A[i][j]+B[i]
            [k+3]*C[k+3][j];
        }

        if ( N-mod+2 < N ) {
            A[i][j] = A[i][j]+B[i][N-
mod]*C[N-mod][j];
            A[i][j] = A[i][j]+B[i][N-
mod+1]*C[N-mod+1][j];
            A[i][j] = A[i][j]+B[i][N-
mod+2]*C[N-mod+2][j];
        } else if ( N-mod+1 < N ) {
            A[i][j] = A[i][j]+B[i][N-
mod]*C[N-mod][j];
            A[i][j] = A[i][j]+B[i][N-
mod+1]*C[N-mod+1][j];
        } else if ( N-mod < N ) {
            A[i][j] = A[i][j]+B[i][N-
mod]*C[N-mod][j];
        }

        }

    // Tiempo
    clock_gettime(CLOCK_REALTIME,&cgt2);

```

```

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

// Impresión de tiempo de ejecución
printf("Tiempo (seg): %11.9f\n", t);

// Impresión de resultados
printf("\n_____ \nResultados:\n");

printf("\nMatriz B: \n");
if ( N < 10 )
    for ( i = 0; i < N; i++ ) {
        for ( j = 0; j < N; j++ )
            printf("%f ", B[i][j]);
        printf("\n");
    }
else
    printf("B[0][0]=%f B[%d][%d]=%f\n", B[0][0], N-1, N-1,
B[N-1][N-1]);

printf("\nMatriz C: \n");
if ( N < 10 )
    for ( i = 0; i < N; i++ ) {
        for ( j = 0; j < N; j++ )
            printf("%f ", C[i][j]);
        printf("\n");
    }
else
    printf("C[0][0]=%f C[%d][%d]=%f\n", C[0][0], N-1, N-1,
C[N-1][N-1]);

printf("\nMatriz A=B*C: \n");
if ( N < 10 )
    for ( i = 0; i < N; i++ ) {
        for ( j = 0; j < N; j++ )
            printf("%f ", A[i][j]);
        printf("\n");
    }
else
    printf("A[0][0]=%f A[%d][%d]=%f\n", A[0][0], N-1, N-1,
A[N-1][N-1]);

    printf("\n");

    return 0;
}

```

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

```

ejer1 ) ./pmm-secuencial 100
Tiempo (seg): 0.003124022

```

Resultados:

Matriz B:

B[0][0]=1.000000 B[99][99]=100.000000

Matriz C:

C[0][0]=1.000000 C[99][99]=100.000000

Matriz A=B*C:

A[0][0]=5050.000000 A[99][99]=505000.000000

ejer1) ./pmm-secuencial-modificado_a 100

Tiempo (seg): 0.003990918

Resultados:

Matriz B:

B[0][0]=1.000000 B[99][99]=100.000000

Matriz C:

C[0][0]=1.000000 C[99][99]=100.000000

Matriz A=B*C:

A[0][0]=5050.000000 A[99][99]=505000.000000

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

```
/**
 * @file pmm-secuencial.c
 * @brief Cálculo del producto de dos matrices - secuencial, var. globales
 * @author Miguel Ángel Fernández Gutiérrez <mianfg@correo.ugr.es>
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <emmintrin.h>

// Vectores globales
#define MAX 10000
double A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];

int main(int argc, char ** argv){
    struct timespec cgt1,cgt2; double t;                // para tiempos

    if ( argc < 2 ) {
        printf("[ERROR]-Debe insertar tamaño matriz\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    if ( N > MAX )
        N = MAX;

    int i, j, k;
    double suma;
```

```

// Inicialización matrices B, C
for ( i = 0; i < N; i++ )
    for ( j = 0; j < N; j++ ) {
        B[i][j]=j+1;
        C[i][j]=j+1;
    }

// Tiempo
clock_gettime(CLOCK_REALTIME,&cgt1);

// Cálculo de A=B*C
for ( i = 0; i < N ; i++ )
    for ( k = 0; k < N; j++ ) {
        A[i][j] = 0;
        for ( j = 0; j < N; k++ )
            A[i][j] = A[i][j]+B[i][k]*C[k]
[j];
    }

// Tiempo
clock_gettime(CLOCK_REALTIME,&cgt2);

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

// Impresión de tiempo de ejecución
printf("Tiempo (seg): %11.9f\n", t);

// Impresión de resultados
printf("\n_____ \nResultados:\n");

printf("\nMatriz B: \n");
if ( N < 10 )
    for ( i = 0; i < N; i++ ) {
        for ( j = 0; j < N; j++ )
            printf("%f ", B[i][j]);
        printf("\n");
    }
else
    printf("B[0][0]=%f B[%d][%d]=%f\n", B[0][0], N-1, N-1,
B[N-1][N-1]);

printf("\nMatriz C: \n");
if ( N < 10 )
    for ( i = 0; i < N; i++ ) {
        for ( j = 0; j < N; j++ )
            printf("%f ", C[i][j]);
        printf("\n");
    }
else
    printf("C[0][0]=%f C[%d][%d]=%f\n", C[0][0], N-1, N-1,
C[N-1][N-1]);

printf("\nMatriz A=B*C: \n");

```



```

        if ( N < 10 )
            for ( i = 0; i < N; i++ ) {
                for ( j = 0; j < N; j++ )
                    printf("%f ", A[i][j]);
                printf("\n");
            }
        else
            printf("A[0][0]=%f A[%d][%d]=%f\n", A[0][0], N-1, N-1,
A[N-1][N-1]);

        printf("\n");

        return 0;
}

```

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

```

ejer1 ) ./pmm-secuencial-modificado_b 100
Tiempo (seg): 0.002049872

```

Resultados:

Matriz B:
B[0][0]=1.000000 B[99][99]=100.000000

Matriz C:
C[0][0]=1.000000 C[99][99]=100.000000

Matriz A=B*C:
A[0][0]=5050.000000 A[99][99]=505000.000000

1.1. TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		0.003124022
Modificación a)	Desenrollado de bucle	0.003990918
Modificación b)	Cambio de iteradores en el for	0.002049872
...		

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

Es claro que las modificaciones mejoran el tiempo. De entre ellas, la que más mejora el tiempo es la modificación 2, lo cual tiene sentido por la penalización que se aprecia en caché.

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: figura1-original.c

```

/**
 * @file figura1-original.c
 * @brief Cálculo de la suma de dos vectores - secuencial, no optimizado
 * @author Miguel Ángel Fernández Gutiérrez <mianfg@correo.ugr.es>

```

```

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

// Vectores globales
struct {
    int a;
    int b;
} s[5000];

int R[40000];

int main(){
    struct timespec cgt1,cgt2; double t;           // para tiempos

    int i, ii, X1, X2;

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

    // Tiempo
    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;
    }

    // Tiempo
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    // Impresión de tiempo de ejecución
    printf("Tiempo (seg): %f\n", t);

    // Impresión de resultados
    printf("\n_____ \nResultados:\n");

    printf("\nVector R: \n");
    printf("R[0]=%d R[39999]=%d\n", R[0], R[40000-1]);

    return 0;
}

```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación–: recorremos los vectores a y b en el mismo bucle. Por como están colocados en el struct, tenemos en memoria $\{a_0, b_0, a_1, b_1, \dots, a_n, b_n\}$. Con el código original estamos trayendo de caché todos los bloques del struct dos veces, para recorrer los vectores a y b, respectivamente. Sin embargo, con esta modificación sólo traemos el struct completo de caché una sola vez, porque vamos recorriendo la memoria de forma contigua, sin saltos.

Modificación b) –explicación–: desenrollamos ambos for, para ir de cinco en cinco (divisor de 5000, que es el tamaño fijo del vector s).

1.1. CÓDIGOS FUENTE MODIFICACIONES**a) Captura figura1-modificado_a.c**

```
/**
 * @file figura1-modificado_a.c
 * @brief Cálculo de la suma de dos vectores - secuencial, modificación A
 * @author Miguel Ángel Fernández Gutiérrez <mianfg@correo.ugr.es>
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Vectores globales
struct {
    int a;
    int b;
} s[5000];

int R[40000];

int main(){
    struct timespec cgt1,cgt2; double t;                // para tiempos

    int i, ii, X1, X2;

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

    // Tiempo
    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;
    }

    // Tiempo
    clock_gettime(CLOCK_REALTIME,&cgt2);
```

```

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

    // Impresión de tiempo de ejecución
    printf("Tiempo (seg): %f\n", t);

    // Impresión de resultados
    printf("\n_____ \nResultados:\n");

    printf("\nVector R: \n");
    printf("R[0]=%d R[39999]=%d\n", R[0], R[40000-1]);

    return 0;
}

```

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

```

ejer1 ) ./figura1-original
Tiempo (seg): 0.194163

```

Resultados:

```

Vector R:
R[0]=24995000 R[39999]=-162502500

```

```

ejer1 ) ./figura1-modificado_a
Tiempo (seg): 0.169247

```

Resultados:

```

Vector R:
R[0]=24995000 R[39999]=-162502500

```

b) Captura figura1-modificado_b.c

```

/**
 * @file figura1-modificado_b.c
 * @brief Cálculo de la suma de dos vectores - secuencial, modificación B
 * @author Miguel Ángel Fernández Gutiérrez <mianfg@correo.ugr.es>
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Vectores globales
struct {
    int a;
    int b;
} s[5000];

int R[40000];

```

```

int main(){
    struct timespec cgt1,cgt2; double t;           // para tiempos

    int i, ii, X1, X2;

    // Tiempo
    clock_gettime(CLOCK_REALTIME,&cgt1);

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

        for ( i = 0; i < 5000; i+=5 ) {
            X1 += 2*s[i].a+ii;
            X1 += 2*s[i+1].a+ii;
            X1 += 2*s[i+2].a+ii;
            X1 += 2*s[i+3].a+ii;
        }
        for ( i = 0; i < 5000; i+=5 ) {
            X2 += 3*s[i].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
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    // Impresión de tiempo de ejecución
    printf("Tiempo (seg): %f\n", t);

    // Impresión de resultados
    printf("\n_____ \nResultados:\n");

    printf("\nVector R: \n");
    printf("R[0]=%d R[39999]=%d\n", R[0], R[40000-1]);

    return 0;
}

```

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

```

ejer1 ) ./figura1-modificado_b
Tiempo (seg): 0.098717

```

Resultados:

```

Vector R:
R[0]=0 R[39999]=-159996000

```

1.1. TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		0.194163
Modificación a)	Se recorren los vectores a y b en el mismo bucle (caché)	0.169247
Modificación b)	Desenrollado de bucles	0.098717
...		

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

En este caso, el desenrollado de bucles es la mejor opción.

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 acarreen. 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

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

int main(int argc, char **argv) {
    if ( argc < 2 ) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }

    int N = atoi(argv[1]);

    struct timespec ini, fin;
    double tiempo;

    // Creación de vector y matriz
    int i,a=47;
    int x[N], y[N];
```

```

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

// Cálculo del resultado
clock_gettime(CLOCK_REALTIME,&ini);

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

clock_gettime(CLOCK_REALTIME,&fin);

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

// Resultados
printf("Tiempo(seg): %f\n y[0]=%d, y[N-1]=%d \n", tiempo, y[0], y[N-1]);
}

```

Tiempos ejec.	-O0	-O1	-Os	-O2	-O3
	0.002355	0.000812	0.002267	0.002272	0.000523

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

```

ejer2 ) ./daxpy-00 1000000
Tiempo(seg): 0.002355
y[0]=0, y[N-1]=47999952
ejer2 ) ./daxpy-01 1000000
Tiempo(seg): 0.000812
y[0]=0, y[N-1]=47999952
ejer2 ) ./daxpy-02 1000000
Tiempo(seg): 0.002272
y[0]=0, y[N-1]=47999952
ejer2 ) ./daxpy-03 1000000
Tiempo(seg): 0.000523
y[0]=0, y[N-1]=47999952
ejer2 ) ./daxpy-0s 1000000
Tiempo(seg): 0.002267
y[0]=0, y[N-1]=47999952

```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

-O0: no tenemos optimización alguna.

-O1: el número de líneas se reduce, se generan más llamadas a subrutinas con un número de instrucciones mucho menor, especialmente en la suma de vectores.

-O2: vemos que el número de instrucciones es parecido, pero observamos cambios considerables en las instrucciones utilizadas, que serán más eficientes.

-O3: el código ensamblador es mucho más complejo y difícil de entender, aumenta también el número de

subrutinas llamadas.

-Os: disminuimos el tamaño del ejecutable, el código se parece mucho al de -O1 y -O2, pero podemos ver en el tamaño de archivo que este es menor.

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

```

    jmp    .L5
.L6:
    movq   -128(%rbp), %rax
    movl   -144(%rbp), %edx
    movslq    %edx, %rdx
    movl   (%rax,%rdx,4), %eax
    imull   -140(%rbp), %eax
    movl   %eax, %ecx
    movq   -112(%rbp), %rax
    movl   -144(%rbp), %edx
    movslq    %edx, %rdx
    movl   (%rax,%rdx,4), %eax
    addl   %eax, %ecx
    movq   -112(%rbp), %rax
    movl   -144(%rbp), %edx
    movslq    %edx, %rdx
    movl   %ecx, (%rax,%rdx,4)
    addl   $1, -144(%rbp)
.L5:
    movl   -144(%rbp), %eax
    cmpl   -148(%rbp), %eax
    jle    .L6

```

daxpy01.s

```

.L4:
    movslq    %eax, %rdx
    movl   %eax, (%r12,%rdx,4)
    movl   %eax, (%rbx,%rdx,4)
    addl   $1, %eax
    cmpl   %ecx, %eax
    jle    .L4

```

daxpy02.s

```

.L4:
    movl   %ebx, (%r15,%rbx,4)
    movl   %ebx, (%r14,%rbx,4)

```



```
addq $1, %rbx
cmpq %rax, %rbx
jne .L4
```

daxpy03.s

```
.L15:
    leaq -80(%rbp), %rsi
    xorl %edi, %edi
    movslq %r13d, %r13
    call clock_gettime@PLT
    movq -72(%rbp), %rax
    pxor %xmm0, %xmm0
    subq -88(%rbp), %rax
    pxor %xmm1, %xmm1
    movl (%rbx,%r13,4), %ecx
    movl 0(,%r12,4), %edx
    leaq .LC4(%rip), %rsi
    movl $1, %edi
    cvtsi2sdq %rax, %xmm0
    movq -80(%rbp), %rax
    subq -96(%rbp), %rax
    cvtsi2sdq %rax, %xmm1
    movl $1, %eax
    divsd .LC3(%rip), %xmm0
    addsd %xmm1, %xmm0
    call __printf_chk@PLT
    xorl %eax, %eax
    movq -56(%rbp), %rbx
    xorq %fs:40, %rbx
    jne .L34
    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
.L19:
    .cfi_restore_state
    movl $2, -104(%rbp)
    jmp .L5
.L23:
    movl $2, %esi
    jmp .L13
.L3:
    leaq -96(%rbp), %rsi
```

```
xorl %edi, %edi  
leal -1(%r14), %r13d  
call clock_gettime@PLT  
jmp  .L15
```

daxpy0s.s

```
.L3:  
    cmpl %eax, %ebx  
    jl   .L10  
    movl %eax, (%r14,%rax,4)  
    movl %eax, 0(%r13,%rax,4)  
    incq %rax  
    jmp  .L3
```