2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

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

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i7-4712MQ CPU @ 2.30GHz*

Sistema operativo utilizado: *Arch Linux*

Versión de gcc utilizada: *gcc (GCC) 6.3.1 20170109*

Adjunte el contenido del fichero /proc/cpuinfo de la máquina en la que ha tomado las medidas

1. Para el núcleo que se muestra en la Figura 1 (ver guion de prácticas), 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

A) MULTIPLICACIÓN DE MATRICES:
CÓDIGO FUENTE: `pmm-secuencial.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```c
/* pmm-secuencial.c
   Producto de dos matrices cuadradas, M y L.
   Para compilar usar (-lrt: real time library)
   gcc  -O2 pmv-secuencial.c -o pmv -lrt
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>  // biblioteca donde se encuentra la función clock_gettime()

double **M ;
double **L ;
double **R ;

unsigned int f ;
int i;
```

```c
int main(int argc, char** argv){


    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    //Leer argumento de entrada (nº de filas y columnas)
    if (argc<2){
        printf("Error: Falta el número de filas y columnas.\n");
        exit(-1);
    }
    f = atoi(argv[1]);// Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    M = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) M[i] = (double*)malloc(f * sizeof(double)) ;
    L  = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) L[i] = (double*)malloc(f * sizeof(double)) ;
    // matriz resultado
    R = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++) R[i] = (double*)malloc(f * sizeof(double)) ;

    if ( (L==NULL) ||  (M==NULL) || (R==NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }

    int j ;

    //Inicializar matrices

    for(int i = 0 ; i < f ; i++){
        for(int j = 0 ; j < f ; j++){
            M[i][j] = 0.5*(i+j);
            L[i][j] = (i+j) ;
            R[i][j] = 0 ;
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    int k ;
    //Calcular multiplicación.
    for(i=0; i<f; i++){
        for(j=0 ; j<f ; j++){
            for(k=0 ; k<f ; k++){
                R[i][j] += M[i][k]*L[j][k];
            }
        }
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
    ncgt+=(double) (cgt2.tv_sec-cgt1.tv_sec)+
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));



    printf("\nR[0][0]=%f ; R[n-1][n-1] = %f", R[0][0], R[f-1][f-1]) ;

    ncgt = ncgt ;

    printf("\nTiempo(seg.): %11.9f\t / filas: %u\t / columnas: %u\t",
           ncgt,f,f) ;

    FILE * fp ;
    fp = fopen("salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\n%d\t%f",f,ncgt);
    fclose(fp);

}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-:

He realizado la transposición de la matriz que debería recorrerse por columnas para que pueda recorrerse por filas y evitar así multitud de fallos de caché.

Modificación b) –explicación-:

He realizado la relocalización de las filas de las matrices a multiplicar para que estén alineadas en memoria (estén situadas al inicio de líneas de caché) para minimizar los fallos de caché.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmm-secuencial-modificado_a.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```c
/* pmm-secuencial.c
   Producto de dos matrices cuadradas, M y L.
   Para compilar usar (-lrt: real time library)
   gcc  -O2 pmv-secuencial.c -o pmv -lrt
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>  // biblioteca donde se encuentra la función clock_gettime()

double **M ;
double **L ;
double **R ;

unsigned int f ;
int i;

int main(int argc, char** argv){


    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    //Leer argumento de entrada (nº de filas y columnas)
    if (argc<2){
        printf("Error: Falta el número de filas y columnas.\n");
        exit(-1);
    }
    f = atoi(argv[1]);// Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)


    M = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++)
        M[i] = (double*)malloc(f * sizeof(double)) ;
    L  = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++)
        L[i] = (double*)malloc(f * sizeof(double)) ;
    // matriz resultado
    R = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++)
        R[i] = (double*)malloc(f * sizeof(double)) ;

    if ( (L==NULL) ||  (M==NULL) || (R==NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }

    int j ;

    //Inicializar matrices

    for(int i = 0 ; i < f ; i++){
        for(int j = 0 ; j < f ; j++){
            M[i][j] = 0.5*(i+j);
            L[i][j] = (i+j) ;
            R[i][j] = 0 ;
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    // transposición del a matriz L para eliminar fallos de caché.

    for(i = 0 ; i < f ; i++)
        for(j = 0 ; j < f ; j++){
            if(i != j){
                int aux = L[i][j] ;
                L[i][j] = L[j][i] ;
                L[j][i] = aux ;
            }
        }

    int k ;
    //Calcular multiplicación.
    for(i=0; i<f; i++){
        for(j=0 ; j<f ; j++){
            for(k=0 ; k<f ; k++){
                R[i][j] += M[i][k]*L[j][k];
            }
```

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



    printf("\nR[0][0]=%f ; R[n-1][n-1] = %f", R[0][0], R[f-1][f-1]) ;

    ncgt = ncgt ;

    printf("\nTiempo(seg.): %11.9f\t / filas: %u\t / columnas: %u\t",
           ncgt,f,f) ;

    FILE * fp ;
    fp = fopen("salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\na\t%d\t%f",f,ncgt);
    fclose(fp);

}
```

b) `pmm-secuencial-modificado_b.c` (ADJUNTAR CÓDIGO FUENTE AL .ZIP)

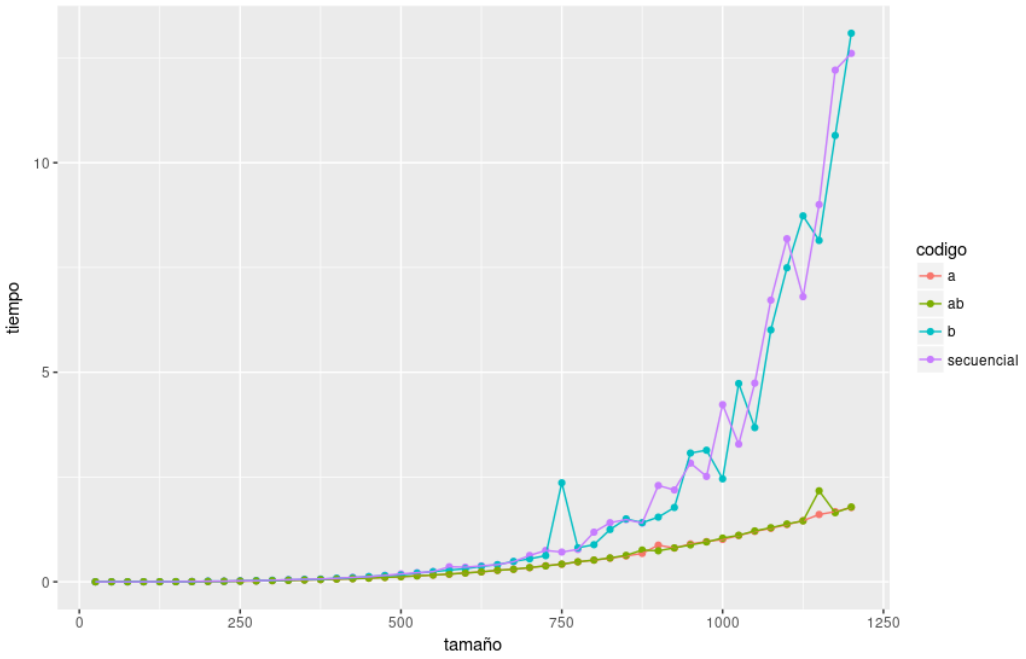
```
/* pmm-secuencial.c
   Producto de dos matrices cuadradas, M y L.
   Para compilar usar (-lrt: real time library)
   gcc  -O2 pmv-secuencial.c -o pmv -lrt
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>  // biblioteca donde se encuentra la función clock_gettime()

double **M ;
double **MA ;
double **L ;
double **LA ;
double **R ;
double **RA ;

unsigned int f ;
int i;

const unsigned long BOUND = 64 ;

int main(int argc, char** argv){


    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    //Leer argumento de entrada (nº de filas y columnas)
    if (argc<2){
        printf("Error: Falta el número de filas y columnas.\n");
        exit(-1);
    }
    f = atoi(argv[1]);// Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

    unsigned int fil_tam = sizeof(double) * f ;

    unsigned int fil_lines = fil_tam / BOUND + 1;

    printf("\n\ntamaño de fila: %d, lineas por fila: %d\n\n", fil_tam, fil_lines) ;


    M = (double **)malloc(f*sizeof(double*)) ;
    MA = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++){
        // reservo una linea más de las necesarias, menos 1B
        MA[i]  =  (double *) malloc(BOUND*(fil_lines+1)-1) ;
        M[i] = (double *) ((((unsigned long) MA[i]) + BOUND-1) & ~(BOUND-1)) ;
    }

    L = (double **)malloc(f*sizeof(double*)) ;
    LA = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++){
```

```
        // reservo una linea más de las necesarias, menos 1B
        LA[i]  =  (double *) malloc(BOUND*(fil_lines+1)-1) ;
        L[i] = (double *) ((((unsigned long) LA[i]) + BOUND-1) & ~(BOUND-1)) ;
    }

    R = (double **)malloc(f*sizeof(double*)) ;
    RA = (double **)malloc(f*sizeof(double*)) ;
    for( i = 0 ; i < f ; i++){
        // reservo una linea más de las necesarias, menos 1B
        RA[i]  =  (double *) malloc(BOUND*(fil_lines+1)-1) ;
        R[i] = (double *) ((((unsigned long) RA[i]) + BOUND-1) & ~(BOUND-1)) ;
    }


    if ( (L==NULL) ||  (M==NULL) || (R==NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
    }

    int j ;

    //Inicializar matrices

    for(int i = 0 ; i < f ; i++){
        for(int j = 0 ; j < f ; j++){
            M[i][j] = 0.5*(i+j);
            L[i][j] = (i+j) ;
            R[i][j] = 0 ;
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    int k ;
    //Calcular multiplicación.
    for(i=0; i<f; i++){
        for(j=0 ; j<f ; j++){
            for(k=0 ; k<f ; k++){
                R[i][j] += M[i][k]*L[j][k];
            }
        }
    }

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



    printf("\nR[0][0]=%f ; R[n-1][n-1] = %f", R[0][0], R[f-1][f-1]) ;

    ncgt = ncgt ;

    printf("\nTiempo(seg.): %11.9f\t / filas: %u\t / columnas: %u\t",
           ncgt,f,f) ;

    FILE * fp ;
    fp = fopen("salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\nb\t%d\t%f",f,ncgt);
    fclose(fp);

    for(int i = 0 ; i < f ; i++){
        free(MA[i]);
        free(LA[i]);
        free(RA[i]);
    }
    free(M) ; free(MA) ;
    free(L) ; free(LA) ;
    free(R) ; free(RA) ;
}
```

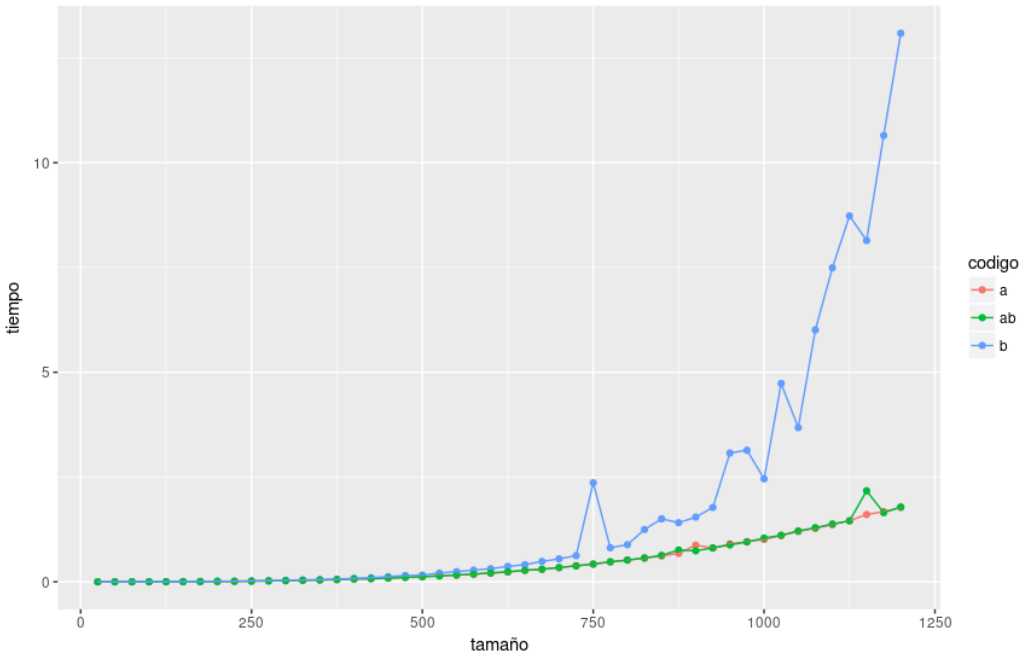
Capturas de pantalla (que muestren que el resultado es correcto):

```
pinguino 1N0 ~ UGR … Practicas 4 code % gcc pmm-secuencial.c -o secuencial -O2
pinguino 1N0 ~ UGR … Practicas 4 code % gcc pmm-secuencial-modificado_a.c -o secuencialmA -O2
pinguino 1N0 ~ UGR … Practicas 4 code % gcc pmm-secuencial-modificado_b.c -o secuencialmB -O2
pinguino 1N0 ~ UGR … Practicas 4 code % gcc pmm-secuencial-modificado_ab.c -o secuencialmAB -O2
pinguino 1N0 ~ UGR … Practicas 4 code % ./secuencial 100

R[0][0]=328350.000000 ; R[n-1][n-1] = 8904225.000000
Tiempo(seg.): 0.001443847        / filas: 100    / columnas: 100        %
pinguino 1N0 ~ UGR … Practicas 4 code % ./secuencialmA 100

R[0][0]=328350.000000 ; R[n-1][n-1] = 8904225.000000
Tiempo(seg.): 0.003027683        / filas: 100    / columnas: 100        %
pinguino 1N0 ~ UGR … Practicas 4 code % ./secuencialmB 100


tamaño de fila: 800, lineas por fila: 13


R[0][0]=328350.000000 ; R[n-1][n-1] = 8904225.000000
Tiempo(seg.): 0.003202516        / filas: 100    / columnas: 100        %
pinguino 1N0 ~ UGR … Practicas 4 code % ./secuencialmAB 100


tamaño de fila: 800, lineas por fila: 13


R[0][0]=328350.000000 ; R[n-1][n-1] = 8889548.250000
Tiempo(seg.): 0.001095807        / filas: 100    / columnas: 100        %
pinguino 1N0 ~ UGR … Practicas 4 code %
```

1.1. TIEMPOS y COMENTARIOS SOBRE LOS RESULTADOS:



Los tiempos conseguidos con la mejora b son solo un poco mejores que los tiempos para el código secuencial, dándose casos en los que el código es mejor y casos en los que es peor. Sin embargo, la mejora de la transposición de la matriz genera tiempos mucho mejores que las dos versiones anteriores.



***Nota:** la razón por la que en clase me salían los tiempos muy parecidos era que, aunque había eliminado la parte de transposición del código b, seguía accediendo a la segunda matriz como si estuviera transpuesta (se me olvidó darle la vuelta a los índices) y no me dí cuenta de esto porque al inicializar la matriz estaba haciéndola simétrica, y los resultados eran correctos (en realidad, como la matriz era simétrica daba igual que hiciera la transposición o no, por lo que los códigos a y b eran iguales).

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

pmm-secuencial.s

```
calcular_multiplicacion:
.LFB21:
                    .cfi_startproc
                    movl
                    f(%rip), %edi
                    testl
                    %edi, %edi
                    je
                    .L15
                    leal
                    -1(%rdi), %eax
                    pushq
                    %rbp
                    .cfi_def_cfa_offs
et 16
                    .cfi_offset 6,
-16
                    movq
                    L(%rip), %r9
                    pushq
                    %rbx
                    .cfi_def_cfa_offs
et 24
                    .cfi_offset 3,
-24
                    movq
                    R(%rip), %rbp
                    xorl
                    %r11d, %r11d
                    movq
                    M(%rip), %rbx
                    leaq
                    8(,%rax,8), %r10
.L6:
                    movq
                    0(%rbp,%r11,8),
%rsi
                    movq
                    (%rbx,%r11,8),
%r8
                    xorl
                    %edx, %edx
                    .p2align 4,,10
                    .p2align 3
.L7:
                    movsd
                    (%rsi,%rdx),
%xmm1
                    xorl
                    %eax, %eax
                    .p2align 4,,10
                    .p2align 3
.L4:
                    movq
                    (%r9,%rax,8),
%rcx
                    movsd
                    (%rcx,%rdx),
%xmm0
                    mulsd
                    (%r8,%rax,8),
%xmm0
                    addq
                    $1, %rax
                    cmpl
                    %eax, %edi
                    addsd
```

pmm-secuencial-modificado_b.s

```
calcular_multiplicacion:
.LFB21:
                    .cfi_startproc
                    movl
                    f(%rip), %edi
                    testl
                    %edi, %edi
                    je
                    .L15
                    leal
                    -1(%rdi), %eax
                    pushq
                    %rbp
                    .cfi_def_cfa_offs
et 16
                    .cfi_offset 6,
-16
                    movq
                    L(%rip), %r9
                    pushq
                    %rbx
                    .cfi_def_cfa_offs
et 24
                    .cfi_offset 3,
-24
                    movq
                    R(%rip), %rbp
                    xorl
                    %r11d, %r11d
                    movq
                    M(%rip), %rbx
                    leaq
                    8(,%rax,8), %r10
.L6:
                    movq
                    0(%rbp,%r11,8),
%rsi
                    movq
                    (%rbx,%r11,8),
%r8
                    xorl
                    %edx, %edx
                    .p2align 4,,10
                    .p2align 3
.L7:
                    movsd
                    (%rsi,%rdx),
%xmm1
                    xorl
                    %eax, %eax
                    .p2align 4,,10
                    .p2align 3
.L4:
                    movq
                    (%r9,%rax,8),
%rcx
                    movsd
                    (%rcx,%rdx),
%xmm0
                    mulsd
                    (%r8,%rax,8),
%xmm0
                    addq
                    $1, %rax
                    cmpl
                    %eax, %edi
                    addsd
```

pmm-secuencial-modificado_c.s

```
calcular_multiplicacion:
.LFB21:
                    .cfi_startproc
                    movl
                    f(%rip), %ecx
                    movq
                    L(%rip), %r9
                    testl
                    %ecx, %ecx
                    je
                    .L22
                    pushq
                    %rbx
                    .cfi_def_cfa_offs
et 16
                    .cfi_offset 3,
-16
                    xorl
                    %r8d, %r8d
.L12:
                    leaq
                    0(,%r8,8), %r10
                    movl
                    %r8d, %edi
                    xorl
                    %eax, %eax
                    .p2align 4,,10
                    .p2align 3
.L5:
                    cmpl
                    %eax, %edi
                    jge
                    .L4
                    movq
                    (%r9,%r8,8), %rdx
                    leaq
                    (%rdx,%rax,8),
%rsi
                    movq
                    %r10, %rdx
                    addq
                    (%r9,%rax,8),
%rdx
                    movsd
                    (%rsi), %xmm0
                    movsd
                    (%rdx), %xmm1
                    movsd
                    %xmm1, (%rsi)
                    movsd
                    %xmm0, (%rdx)
.L4:
                    addq
                    $1, %rax
                    cmpl
                    %eax, %ecx
                    ja
                    .L5
                    addq
                    $1, %r8
                    cmpl
                    %r8d, %ecx
                    ja
                    .L12
                    movq
                    R(%rip), %rbx
                    movq
```

```
                %xmm0, %xmm1
                movsd
                %xmm1, (%rsi,
%rdx)
                ja
                .L4
                addq
                $8, %rdx
                cmpq
                %rdx, %r10
                jne
                .L7
                addq
                $1, %r11
                cmpl
                %r11d, %edi
                ja
                .L6
                popq
                %rbx
                .cfi_restore 3
                .cfi_def_cfa_offs
et 16
                popq
                %rbp
                .cfi_restore 6
                .cfi_def_cfa_offs
et 8
                ret
```

```
                %xmm0, %xmm1
                movsd
                %xmm1, (%rsi,
%rdx)
                ja
                .L4
                addq
                $8, %rdx
                cmpq
                %rdx, %r10
                jne
                .L7
                addq
                $1, %r11
                cmpl
                %r11d, %edi
                ja
                .L6
                popq
                %rbx
                .cfi_restore 3
                .cfi_def_cfa_offs
et 16
                popq
                %rbp
                .cfi_restore 6
                .cfi_def_cfa_offs
et 8
                ret
```

```
                M(%rip), %r11
                xorl
                %r10d, %r10d
.L8:
                movq
                (%rbx,%r10,8),
%rsi
                movq
                (%r11,%r10,8),
%r8
                xorl
                %edx, %edx
                .p2align 4,,10
                .p2align 3
.L11:
                movq
                (%r9,%rdx,8),
%rdi
                movsd
                (%rsi,%rdx,8),
%xmm1
                xorl
                %eax, %eax
                .p2align 4,,10
                .p2align 3
.L9:
                movsd
                (%r8,%rax,8),
%xmm0
                mulsd
                (%rdi,%rax,8),
%xmm0
                addq
                $1, %rax
                cmpl
                %eax, %ecx
                addsd
                %xmm0, %xmm1
                movsd
                %xmm1, (%rsi,
%rdx,8)
                ja
                .L9
                addq
                $1, %rdx
                cmpl
                %edx, %ecx
                ja
                .L11
                addq
                $1, %r10
                cmpl
                %r10d, %ecx
                ja
                .L8
                popq
                %rbx
                .cfi_restore 3
                .cfi_def_cfa_offs
et 8
                ret
```

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-original.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```c
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>  // biblioteca donde se encuentra la función clock_gettime()

int main()
{
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    int i, ii ;

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

    int X1, X2 ;

    int R[40000] ;

    // inicializar s ;

    for(i = 0 ; i < 5000 ; i++){
        s[i].a = i ;
        s[i].b = 4999 - i ;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for (ii=0; ii<40000;ii++) {
        X1=0; X2=0;
        for(i=0; i<5000;i++) X1+=2*s[i].a+ii;
        for(i=0; i<5000;i++) X2+=3*s[i].b-ii;
        if (X1<X2)
            R[ii]=X1 ;
        else
            R[ii]=X2;
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    // imprimir algunos valores de R:

    for(i = 0 ; i < 40000 ; i+=12345){
        printf("\nR[%d] = %d", i, R[i]) ;
    }

    printf("\n\ntiempo: %f",ncgt) ;

    FILE * fp ;
    fp = fopen("figura1_salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\noriginal\t%f",ncgt);
    fclose(fp);
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: he eliminado uno de los bucles for y he hecho las operaciones sobre los parámetros a y b del struct en una misma iteración para aprovechar mejor la localidad espacial de estos ya que estarán intercalados en el array (es decir estarán situados de la forma: abababababababababababababab….)

Modificación b) –explicación-: he realizado los cálculos de s[i].a*2 y s[i].b*3 previamente al bucle de ii para luego no tener que calcular dichas multiplicaciones todas las iteraciones. Además (aunque no sé si el compilador ya hará esto de por si), para ahorrarme las multiplicaciones y, dado que se trata de multiplicaciones por números pequeños (2 y 3) he calculado las multiplicaciones como sumas.

Modificación ab) He juntado las dos ideas haciendo el cálculo previo de las multiplicaciones y eliminando un bucle para mejorar el acceso a memoria.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figura1**-modificado_a.c**

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```c
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>  // biblioteca donde se encuentra la función clock_gettime()

int main()
{
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    int i, ii ;

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

    int X1, X2 ;

    int R[40000] ;

    // inicializar s ;

    for(i = 0 ; i < 5000 ; i++){
        s[i].a = i ;
        s[i].b = 4999 - i ;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for (ii=0; ii<40000;ii++) {
        X1=0; X2=0;
        for(i=0; i<5000;i++){
            X1+=2*s[i].a+ii;
            X2+=3*s[i].b-ii;
        }
        if (X1<X2)
            R[ii]=X1 ;
        else
            R[ii]=X2;
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    // imprimir algunos valores de R:

    for(i = 0 ; i < 40000 ; i+=12345){
        printf("\nR[%d] = %d", i, R[i]) ;
    }

    printf("\n\ntiempo: %f",ncgt) ;

    FILE * fp ;
    fp = fopen("figura1_salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\nmodificadoA\t%f",ncgt);
    fclose(fp);
}
```

b) figura1**-modificado_b.c**
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```c
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>  // biblioteca donde se encuentra la función clock_gettime()
#include <xmmintrin.h>


int main()
{
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    int i, ii ;

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

    int X1, X2 ;

    struct{
        int x1;
        int x2;
    } x[5000] ;

    int R[40000] ;

    // inicializar s ;

    for(i = 0 ; i < 5000 ; i++){
        s[i].a = i ;
        s[i].b = 4999 - i ;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for(i = 0 ; i < 5000 ; i++){
        x[i].x1 = 0 ; x[i].x2 = 0 ;
        x[i].x1 = s[i].a+s[i].a; //x2
        x[i].x2 = s[i].b+s[i].b+s[i].b ; //x3
    }
    for (ii=0; ii<40000;ii++) {
        X1=0; X2=0;
        for(i=0; i<5000;i++){
            X1+=x[i].x1+ii;
        }
        for(i=0; i<5000;i++){
            X2+=x[i].x2-ii;
        }
        if (X1<X2)
            R[ii]=X1 ;
        else
            R[ii]=X2;
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    // imprimir algunos valores de R:

    for(i = 0 ; i < 40000 ; i+=12345){
        printf("\nR[%d] = %d", i, R[i]) ;
    }

    printf("\n\ntiempo: %f",ncgt) ;

    FILE * fp ;
    fp = fopen("figura1_salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\nmodificadoB\t%f",ncgt);
    fclose(fp);
}
```

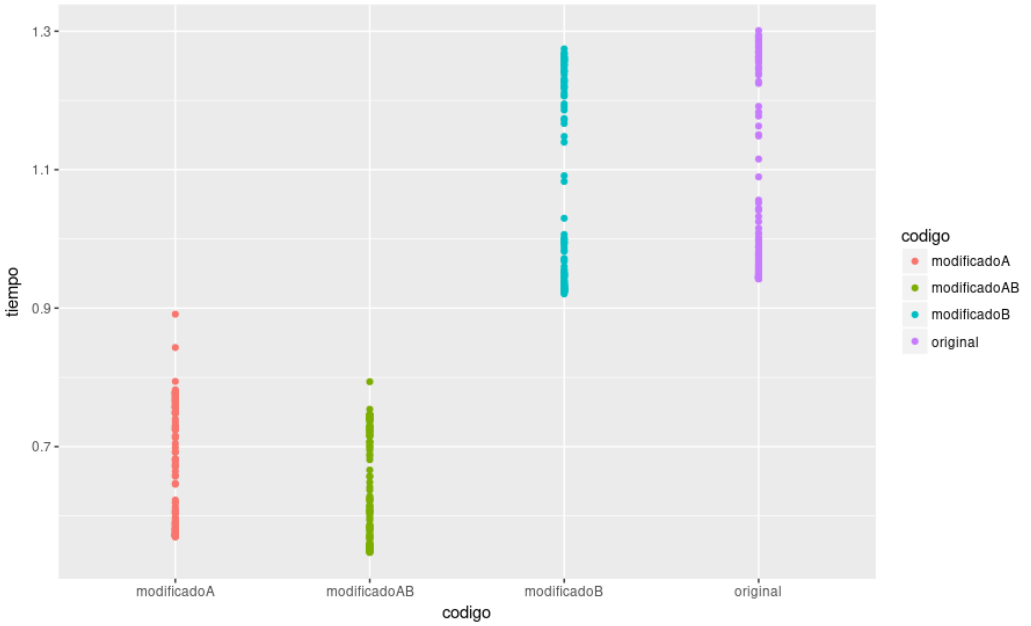
c) figura1**-modificado_ab.c**
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```c
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>  // biblioteca donde se encuentra la función clock_gettime()
#include <xmmintrin.h>


int main()
{
    struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

    int i, ii ;

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

    int X1, X2 ;

    struct{
        int x1;
        int x2;
    } x[5000] ;

    int R[40000] ;

    // inicializar s ;

    for(i = 0 ; i < 5000 ; i++){
        s[i].a = i ;
        s[i].b = 4999 - i ;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for(i = 0 ; i < 5000 ; i++){
        x[i].x1 = 0 ; x[i].x2 = 0 ;
        x[i].x1 = s[i].a+s[i].a; //x2
        x[i].x2 = s[i].b+s[i].b+s[i].b ; //x3
    }
    for (ii=0; ii<40000;ii++) {
        X1=0; X2=0;
        for(i=0; i<5000;i++){
            X1+=x[i].x1+ii;
            X2+=x[i].x2-ii;
        }
        if (X1<X2)
            R[ii]=X1 ;
        else
            R[ii]=X2;
    }
    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    // imprimir algunos valores de R:

    for(i = 0 ; i < 40000 ; i+=12345){
        printf("\nR[%d] = %d", i, R[i]) ;
    }

    printf("\n\ntiempo: %f",ncgt) ;

    FILE * fp ;
    fp = fopen("figura1_salida","a");
    if(fp==NULL){perror("Error opening file.");}
    fprintf(fp,"\nmodificadoAB\t%f",ncgt);
    fclose(fp);
}
```

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

```
pinguino 1N0 ~ UGR … Practicas 4 code % ./figura10

R[0] = 24995000
R[12345] = -24232500
R[24690] = -85957500
R[37035] = -147682500

tiempo: 1.377744%
pinguino 1N0 ~ UGR … Practicas 4 code % ./figura1mA

R[0] = 24995000
R[12345] = -24232500
R[24690] = -85957500
R[37035] = -147682500

tiempo: 0.800745%
pinguino 1N0 ~ UGR … Practicas 4 code % ./figura1mB

R[0] = 24995000
R[12345] = -24232500
R[24690] = -85957500
R[37035] = -147682500

tiempo: 1.284184%
pinguino 1N0 ~ UGR … Practicas 4 code % ./figura1mAB

R[0] = 24995000
R[12345] = -24232500
R[24690] = -85957500
R[37035] = -147682500

tiempo: 0.759292%
pinguino 1N0 ~ UGR … Practicas 4 code %
```

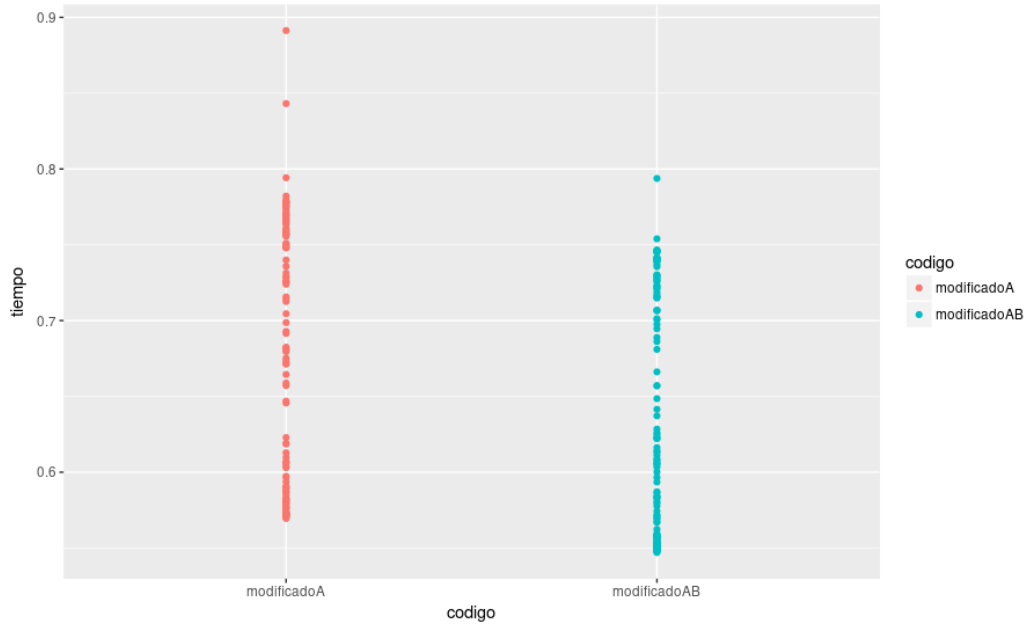
1.1. TIEMPOS y COMENTARIOS SOBRE LOS RESULTADOS:

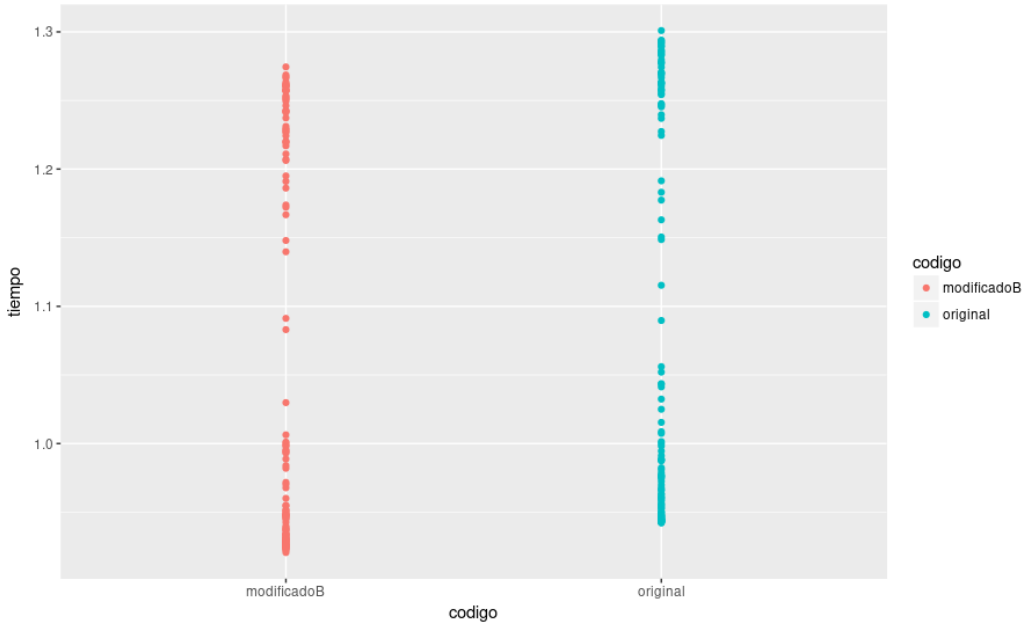


(No son tiempos en función del tamaño sino nubes de puntos con ejecuciones de los distintos códigos con un mismo tamaño para ver los intervalos de tiempos que se obtienen con cada uno)

Como se aprecia en la gráfica, los tiempos peores son los del código original (como es lógico) y los mejores, el resultado de combinar ambas mejoras (en verde), aunque la diferencia no es muy grande con respecto al código con la primera modificación (rojo) que es bastante mejor que el de la segunda mejora (azul) aunque esta tambien mejora un poco los tiempos del original. La conclusión que se puede extraer de aquí es que es

mucho más importante reducir los fallos de caché/accesos a memoria (mejora 1) que reducir el número de operaciones simples (como pueden ser las multiplicaciones). Si se tratara de divisiones en lugar de multiplicaciones probablemente la mejora B daría resultados bastante mejores con respecto al original pero aún así, lo más probable es que la mejora A siguiera siendo mejor porque está ahorrando accesos a memoria, que incluso utilizando cachés son extremádamente lentos en comparación con la velocidad a la que ejecuta instrucciones el procesador.
Para apreciar mejor las diferencias entre los códigos original y mejoraB y entre los códigos mejoraA y mejoraAB he hecho estas otras dos gráficas:





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

pmm-secuencial.s

```
	.file	"figura1-original.c"
	.section	.rodata
.LC1:
	.string	"\nR[%d] = %d"
.LC2:
	.string	"\n\ntiempo: %f"
.LC3:
	.string	"a"
.LC4:
	.string	"figura1_salida"
.LC5:
	.string	"Error opening file."
.LC6:
	.string	"\noriginal\t%f"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset 6, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register 6
	subq	$200080, %rsp
	movl	$0, -4(%rbp)
	jmp	.L2
.L3:
	movl	-4(%rbp), %eax
	cltq
	movl	-4(%rbp), %edx
	movl	%edx, -40064(%rbp,%rax,8)
	movl	$4999, %eax
	subl	-4(%rbp), %eax
	movl	%eax, %edx
	movl	-4(%rbp), %eax
	cltq
	movl
```

pmm-secuencial-modificado_a.s

```
	.file	"figura1-modificado_a.c"
	.section	.rodata
.LC1:
	.string	"\nR[%d] = %d"
.LC2:
	.string	"\n\ntiempo: %f"
.LC3:
	.string	"a"
.LC4:
	.string	"figura1_salida"
.LC5:
	.string	"Error opening file."
.LC6:
	.string	"\nmodificadoA\t%f"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset 6, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register 6
	subq	$200080, %rsp
	movl	$0, -4(%rbp)
	jmp	.L2
.L3:
	movl	-4(%rbp), %eax
	cltq
	movl	-4(%rbp), %edx
	movl	%edx, -40064(%rbp,%rax,8)
	movl	$4999, %eax
	subl	-4(%rbp), %eax
	movl	%eax, %edx
	movl	-4(%rbp), %eax
	cltq
```

pmm-secuencial-modificado_ab.s

```
	.file	"figura1-modificado_ab.c"
	.section	.rodata
.LC1:
	.string	"\nR[%d] = %d"
.LC2:
	.string	"\n\ntiempo: %f"
.LC3:
	.string	"a"
.LC4:
	.string	"figura1_salida"
.LC5:
	.string	"Error opening file."
.LC6:
	.string	"\nmodificadoAB\t%f"
	.text
	.globl	main
	.type	main, @function
main:
.LFB499:
	.cfi_startproc
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset 6, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register 6
	subq	$240080, %rsp
	movl	$0, -4(%rbp)
	jmp	.L2
.L3:
	movl	-4(%rbp), %eax
	cltq
	movl	-4(%rbp), %edx
	movl	%edx, -40064(%rbp,%rax,8)
	movl	$4999, %eax
	subl	-4(%rbp), %eax
	movl	%eax, %edx
	movl	-4(%rbp), %eax
	cltq
```

```
                    %edx,
-40060(%rbp,%rax,8)
                    addl
                    $1, -4(%rbp)
.L2:
                    cmpl
                    $4999, -4(%rbp)
                    jle
                    .L3
                    leaq
                    -48(%rbp), %rax
                    movq
                    %rax, %rsi
                    movl
                    $0, %edi
                    call
                    clock_gettime
                    movl
                    $0, -8(%rbp)
                    jmp
                    .L4
.L11:
                    movl
                    $0, -12(%rbp)
                    movl
                    $0, -16(%rbp)
                    movl
                    $0, -4(%rbp)
                    jmp
                    .L5
.L6:
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -40064(%rbp,
%rax,8), %eax
                    leal
                    (%rax,%rax), %edx
                    movl
                    -8(%rbp), %eax
                    addl
                    %edx, %eax
                    addl
                    %eax, -12(%rbp)
                    addl
                    $1, -4(%rbp)
.L5:
                    cmpl
                    $4999, -4(%rbp)
                    jle
                    .L6
                    movl
                    $0, -4(%rbp)
                    jmp
                    .L7
.L8:
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -40060(%rbp,
%rax,8), %edx
                    movl
                    %edx, %eax
                    addl
                    %eax, %eax
                    addl
                    %edx, %eax
                    subl
                    -8(%rbp), %eax
                    addl
                    %eax, -16(%rbp)
                    addl
                    $1, -4(%rbp)
.L7:
                    cmpl
                    $4999, -4(%rbp)
                    jle
                    .L8
```

```
                    movl
                    %edx,
-40060(%rbp,%rax,8)
                    addl
                    $1, -4(%rbp)
.L2:
                    cmpl
                    $4999, -4(%rbp)
                    jle
                    .L3
                    leaq
                    -48(%rbp), %rax
                    movq
                    %rax, %rsi
                    movl
                    $0, %edi
                    call
                    clock_gettime
                    movl
                    $0, -8(%rbp)
                    jmp
                    .L4
.L9:
                    movl
                    $0, -12(%rbp)
                    movl
                    $0, -16(%rbp)
                    movl
                    $0, -4(%rbp)
                    jmp
                    .L5
.L6:
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -40064(%rbp,
%rax,8), %eax
                    leal
                    (%rax,%rax), %edx
                    movl
                    -8(%rbp), %eax
                    addl
                    %edx, %eax
                    addl
                    %eax, -12(%rbp)
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -40060(%rbp,
%rax,8), %edx
                    movl
                    %edx, %eax
                    addl
                    %eax, %eax
                    addl
                    %edx, %eax
                    subl
                    -8(%rbp), %eax
                    addl
                    %eax, -16(%rbp)
                    addl
                    $1, -4(%rbp)
.L5:
                    cmpl
                    $4999, -4(%rbp)
                    jle
                    .L6
                    movl
                    -12(%rbp), %eax
                    cmpl
                    -16(%rbp), %eax
                    jge
                    .L7
                    movl
                    -8(%rbp), %eax
                    cltq
                    movl
                    -12(%rbp), %edx
```

```
                    movl
                    %edx,
-40060(%rbp,%rax,8)
                    addl
                    $1, -4(%rbp)
.L2:
                    cmpl
                    $4999, -4(%rbp)
                    jle
                    .L3
                    leaq
                    -48(%rbp), %rax
                    movq
                    %rax, %rsi
                    movl
                    $0, %edi
                    call
                    clock_gettime
                    movl
                    $0, -4(%rbp)
                    jmp
                    .L4
.L5:
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    $0, -80064(%rbp,
%rax,8)
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    $0, -80060(%rbp,
%rax,8)
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -40064(%rbp,
%rax,8), %edx
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -40064(%rbp,
%rax,8), %eax
                    addl
                    %eax, %edx
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    %edx,
-80064(%rbp,%rax,8)
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -40060(%rbp,
%rax,8), %edx
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -40060(%rbp,
%rax,8), %eax
                    addl
                    %eax, %edx
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -40060(%rbp,
%rax,8), %eax
                    addl
                    %eax, %edx
                    movl
                    -4(%rbp), %eax
                    cltq
```

```
                    movl
                    -12(%rbp), %eax
                    cmpl
                    -16(%rbp), %eax
                    jge
                    .L9
                    movl
                    -8(%rbp), %eax
                    cltq
                    movl
                    -12(%rbp), %edx
                    movl
                    %edx,
-200064(%rbp,%rax,4)
                    jmp
                    .L10
.L9:
                    movl
                    -8(%rbp), %eax
                    cltq
                    movl
                    -16(%rbp), %edx
                    movl
                    %edx,
-200064(%rbp,%rax,4)
.L10:
                    addl
                    $1, -8(%rbp)
.L4:
                    cmpl
                    $39999, -8(%rbp)
                    jle
                    .L11
                    leaq
                    -64(%rbp), %rax
                    movq
                    %rax, %rsi
                    movl
                    $0, %edi
                    call
                    clock_gettime
                    movq
                    -64(%rbp), %rdx
                    movq
                    -48(%rbp), %rax
                    subq
                    %rax, %rdx
                    movq
                    %rdx, %rax
                    pxor
                    %xmm1, %xmm1
                    cvtsi2sdq
                    %rax, %xmm1
                    movq
                    -56(%rbp), %rdx
                    movq
                    -40(%rbp), %rax
                    subq
                    %rax, %rdx
                    movq
                    %rdx, %rax
                    pxor
                    %xmm0, %xmm0
                    cvtsi2sdq
                    %rax, %xmm0
                    movsd
                    .LC0(%rip), %xmm2
                    divsd
                    %xmm2, %xmm0
                    addsd
                    %xmm1, %xmm0
                    movsd
                    %xmm0, -24(%rbp)
                    movl
                    $0, -4(%rbp)
                    jmp
                    .L12
.L13:
                    movl
                    -4(%rbp), %eax
                    movl
                    %edx,
-200064(%rbp,%rax,4)
                    jmp
                    .L8
.L7:
                    movl
                    -8(%rbp), %eax
                    cltq
                    movl
                    -16(%rbp), %edx
                    movl
                    %edx,
-200064(%rbp,%rax,4)
.L8:
                    addl
                    $1, -8(%rbp)
.L4:
                    cmpl
                    $39999, -8(%rbp)
                    jle
                    .L9
                    leaq
                    -64(%rbp), %rax
                    movq
                    %rax, %rsi
                    movl
                    $0, %edi
                    call
                    clock_gettime
                    movq
                    -64(%rbp), %rdx
                    movq
                    -48(%rbp), %rax
                    subq
                    %rax, %rdx
                    movq
                    %rdx, %rax
                    pxor
                    %xmm1, %xmm1
                    cvtsi2sdq
                    %rax, %xmm1
                    movq
                    -56(%rbp), %rdx
                    movq
                    -40(%rbp), %rax
                    subq
                    %rax, %rdx
                    movq
                    %rdx, %rax
                    pxor
                    %xmm0, %xmm0
                    cvtsi2sdq
                    %rax, %xmm0
                    movsd
                    .LC0(%rip), %xmm2
                    divsd
                    %xmm2, %xmm0
                    addsd
                    %xmm1, %xmm0
                    movsd
                    %xmm0, -24(%rbp)
                    movl
                    $0, -4(%rbp)
                    jmp
                    .L10
.L11:
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -200064(%rbp,
%rax,4), %edx
                    movl
                    -4(%rbp), %eax
                    movl
                    %eax, %esi
                    movl
                    $.LC1, %edi
                    movl
                    %edx,
-80060(%rbp,%rax,8)
                    addl
                    $1, -4(%rbp)
.L4:
                    cmpl
                    $4999, -4(%rbp)
                    jle
                    .L5
                    movl
                    $0, -8(%rbp)
                    jmp
                    .L6
.L11:
                    movl
                    $0, -12(%rbp)
                    movl
                    $0, -16(%rbp)
                    movl
                    $0, -4(%rbp)
                    jmp
                    .L7
.L8:
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -80064(%rbp,
%rax,8), %edx
                    movl
                    -8(%rbp), %eax
                    addl
                    %edx, %eax
                    addl
                    %eax, -12(%rbp)
                    movl
                    -4(%rbp), %eax
                    cltq
                    movl
                    -80060(%rbp,
%rax,8), %eax
                    subl
                    -8(%rbp), %eax
                    addl
                    %eax, -16(%rbp)
                    addl
                    $1, -4(%rbp)
.L7:
                    cmpl
                    $4999, -4(%rbp)
                    jle
                    .L8
                    movl
                    -12(%rbp), %eax
                    cmpl
                    -16(%rbp), %eax
                    jge
                    .L9
                    movl
                    -8(%rbp), %eax
                    cltq
                    movl
                    -12(%rbp), %edx
                    movl
                    %edx,
-240064(%rbp,%rax,4)
                    jmp
                    .L10
.L9:
                    movl
                    -8(%rbp), %eax
                    cltq
                    movl
                    -16(%rbp), %edx
                    movl
                    %edx,
-240064(%rbp,%rax,4)
.L10:
                    addl
```

```
        cltq
        movl
        -200064(%rbp,
%rax,4), %edx
        movl
        -4(%rbp), %eax
        movl
        %eax, %esi
        movl
        $.LC1, %edi
        movl
        $0, %eax
        call
        printf
        addl
        $12345, -4(%rbp)
.L12:
        cmpl
        $39999, -4(%rbp)
        jle
        .L13
        movq
        -24(%rbp), %rax
        movq
        %rax,
-200072(%rbp)
        movsd
        -200072(%rbp),
%xmm0
        movl
        $.LC2, %edi
        movl
        $1, %eax
        call
        printf
        movl
        $.LC3, %esi
        movl
        $.LC4, %edi
        call
        fopen
        movq
        %rax, -32(%rbp)
        cmpq
        $0, -32(%rbp)
        jne
        .L14
        movl
        $.LC5, %edi
        call
        perror
.L14:
        movq
        -24(%rbp), %rdx
        movq
        -32(%rbp), %rax
        movq
        %rdx,
-200072(%rbp)
        movsd
        -200072(%rbp),
%xmm0
        movl
        $.LC6, %esi
        movq
        %rax, %rdi
        movl
        $1, %eax
        call
        fprintf
        movq
        -32(%rbp), %rax
        movq
        %rax, %rdi
        call
        fclose
        movl
        $0, %eax
        leave
        .cfi_def_cfa 7, 8
```

```
        $0, %eax
        call
        printf
        addl
        $12345, -4(%rbp)
.L10:
        cmpl
        $39999, -4(%rbp)
        jle
        .L11
        movq
        -24(%rbp), %rax
        movq
        %rax,
-200072(%rbp)
        movsd
        -200072(%rbp),
%xmm0
        movl
        $.LC2, %edi
        movl
        $1, %eax
        call
        printf
        movl
        $.LC3, %esi
        movl
        $.LC4, %edi
        call
        fopen
        movq
        %rax, -32(%rbp)
        cmpq
        $0, -32(%rbp)
        jne
        .L12
        movl
        $.LC5, %edi
        call
        perror
.L12:
        movq
        -24(%rbp), %rdx
        movq
        -32(%rbp), %rax
        movq
        %rdx,
-200072(%rbp)
        movsd
        -200072(%rbp),
%xmm0
        movl
        $.LC6, %esi
        movq
        %rax, %rdi
        movl
        $1, %eax
        call
        fprintf
        movq
        -32(%rbp), %rax
        movq
        %rax, %rdi
        call
        fclose
        movl
        $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size
        main, .-main
        .section
        .rodata
        .align 8
.LC0:
        .long
        0
```

```
        $1, -8(%rbp)
.L6:
        cmpl
        $39999, -8(%rbp)
        jle
        .L11
        leaq
        -64(%rbp), %rax
        movq
        %rax, %rsi
        movl
        $0, %edi
        call
        clock_gettime
        movq
        -64(%rbp), %rdx
        movq
        -48(%rbp), %rax
        subq
        %rax, %rdx
        movq
        %rdx, %rax
        pxor
        %xmm1, %xmm1
        cvtsi2sdq
        %rax, %xmm1
        movq
        -56(%rbp), %rdx
        movq
        -40(%rbp), %rax
        subq
        %rax, %rdx
        movq
        %rdx, %rax
        pxor
        %xmm0, %xmm0
        cvtsi2sdq
        %rax, %xmm0
        movsd
        .LC0(%rip), %xmm2
        divsd
        %xmm2, %xmm0
        addsd
        %xmm1, %xmm0
        movsd
        %xmm0, -24(%rbp)
        movl
        $0, -4(%rbp)
        jmp
        .L12
.L13:
        movl
        -4(%rbp), %eax
        cltq
        movl
        -240064(%rbp,
%rax,4), %edx
        movl
        -4(%rbp), %eax
        movl
        %eax, %esi
        movl
        $.LC1, %edi
        movl
        $0, %eax
        call
        printf
        addl
        $12345, -4(%rbp)
.L12:
        cmpl
        $39999, -4(%rbp)
        jle
        .L13
        movq
        -24(%rbp), %rax
        movq
        %rax,
-240072(%rbp)
        movsd
```

```
                ret
                .cfi_endproc
.LFE0:
                .size
                main, .-main
                .section
                .rodata
                .align 8
.LC0:
                .long
                0
                .long
                1104006501
                .ident
                "GCC: (GNU) 6.3.1
20170109"
                .section
                .note.GNU-
stack,"",@progbits
```

```
                .long
                1104006501
                .ident
                "GCC: (GNU) 6.3.1
20170109"
                .section
                .note.GNU-
stack,"",@progbits
```

```
                -240072(%rbp),
%xmm0
                movl
                $.LC2, %edi
                movl
                $1, %eax
                call
                printf
                movl
                $.LC3, %esi
                movl
                $.LC4, %edi
                call
                fopen
                movq
                %rax, -32(%rbp)
                cmpq
                $0, -32(%rbp)
                jne
                .L14
                movl
                $.LC5, %edi
                call
                perror
.L14:
                movq
                -24(%rbp), %rdx
                movq
                -32(%rbp), %rax
                movq
                %rdx,
-240072(%rbp)
                movsd
                -240072(%rbp),
%xmm0
                movl
                $.LC6, %esi
                movq
                %rax, %rdi
                movl
                $1, %eax
                call
                fprintf
                movq
                -32(%rbp), %rax
                movq
                %rax, %rdi
                call
                fclose
                movl
                $0, %eax
                leave
                .cfi_def_cfa 7, 8
                ret
                .cfi_endproc
.LFE499:
                .size
                main, .-main
                .section
                .rodata
                .align 8
.LC0:
                .long
                0
                .long
                1104006501
                .ident
                "GCC: (GNU) 6.3.1
20170109"
                .section
                .note.GNU-
stack,"",@progbits
```

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

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

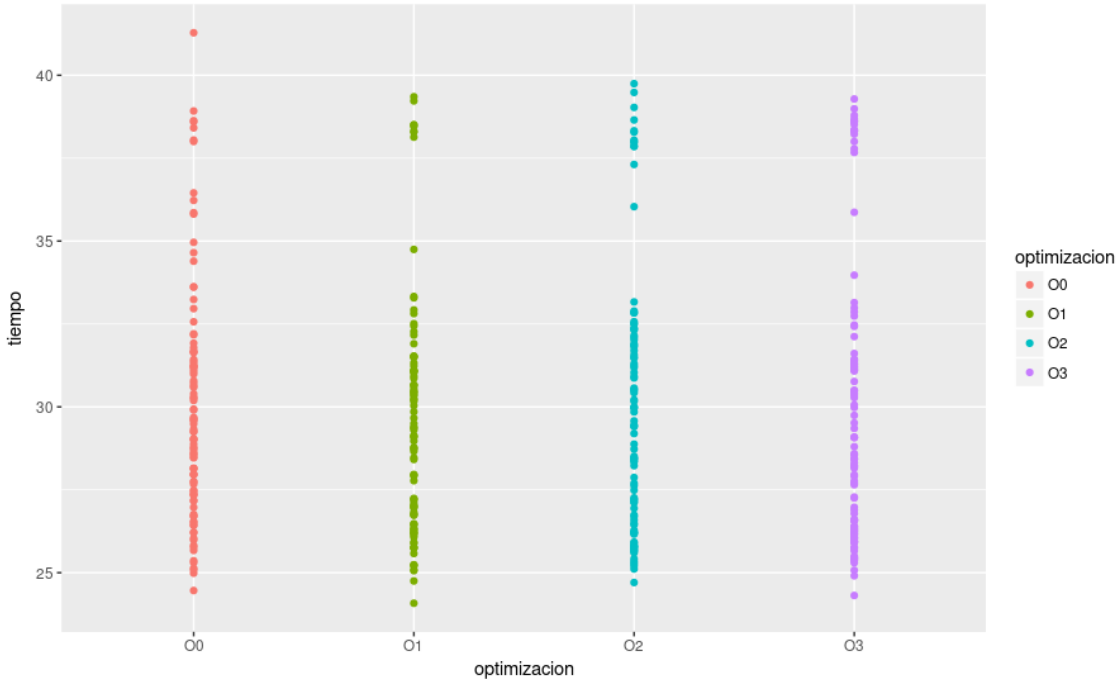
2.1. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O0, -O2, -O3) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

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

CÓDIGO FUENTE: `daxpy.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```c
/* Ejemplo de Programa de Prueba */

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

int suma_prod(int a, int b, int n)
{
        return a * b + n;
}

int main()
{
        int i,j,a,b,n,c;

        clock_t start,stop;

        start = clock();

        n=6000;a=1;b=2;

        for (j=1;j<=10000;j++)
        {
                printf("a=%d b=%d n=%d\n",a,b,n);
                c=suma_prod(a,b,n);
                printf("resultado= %d\n",c);
        }

        stop = clock();

        printf("Tiempo= %f ms\n",difftime(stop,start) / (CLOCKS_PER_SEC / 1000.0));

        return 0;
}
```

Tiempos:



CAPTURAS DE PANTALLA:

```
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
a=1 b=2 n=6000
resultado= 6002
Tiempo= 32.524000 ms
```

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

las optimizaciones con -O2 y con -O3 dan como resultado el mismo código para la función de suma y multiplica. Estas opciones utilizan tan solo una instrucción imul y una leal mientras que la opción con -O1 necesita hacer uso de otras muchas instrucciones para trabajar con los argumentos que se le pasan a la función y utiliza un imul y un add+mov en lugar de un leal. Las mejoras no son muy relevantes pero se deben principalmente al uso de menos instrucciones en la función y al uso de leal para evitar hacer la suma y un mov.

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

daxpy00.s

```
suma_prod:
.LFB0:
                .cfi_startproc
                pushq   %rbp
                .cfi_def_cfa_offset 16
                .cfi_offset 6, -16
                movq    %rsp, %rbp
                .cfi_def_cfa_register 6
                movl    %edi, -4(%rbp)
                movl    %esi, -8(%rbp)
                movl    %edx, -12(%rbp)
                movl    -4(%rbp), %eax
                imull   -8(%rbp), %eax
                movl    %eax, %edx
                movl    -12(%rbp), %eax
                addl    %edx, %eax
                popq    %rbp
                .cfi_def_cfa 7, 8
                ret
```

daxpy02.s

```
suma_prod:
.LFB14:
                .cfi_startproc
                imull   %esi, %edi
                leal    (%rdi,%rdx), %eax
                ret
```

daxpy03.s

```
suma_prod:
.LFB14:
                .cfi_startproc
                imull   %esi, %edi
                leal    (%rdi,%rdx), %eax
                ret
```