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): Cristina María Garrido López
Grupo de prácticas:A1
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) i7-4510U CPU @ 2.00GHz*

Sistema operativo utilizado: *Ubuntu 15.04*

Versión de gcc utilizada: *(respuesta)*

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)

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

            int **matrizB;
            int **matrizC;
```

```
              int **matrizA;

              if (argc < 2){
                            printf("Falta el número de componentes\n");
                            return(1);
              }

              dimension_matrices = atoi(argv[1]);

              if (dimension_matrices > MAX){
                            printf("Tamaño demasiado grande. No superar
%d\n\n",MAX);
                            return(1);
              }


              matrizB = (int **)malloc(dimension_matrices * sizeof(int*));
              matrizC = (int **)malloc(dimension_matrices * sizeof(int*));
              matrizA = (int **)malloc(dimension_matrices * sizeof(int*));

              for (i=0; i<dimension_matrices; i++){
                            matrizB[i] = (int *)malloc(dimension_matrices *
sizeof(int));
                            matrizC[i] = (int *)malloc(dimension_matrices *
sizeof(int));
                            matrizA[i] = (int *)malloc(dimension_matrices *
sizeof(int));
              }


              for (j=0; j<dimension_matrices; j++){
                            for (i=0; i<dimension_matrices; i++){
                                          matrizB[j][i] = j+i;
                                          matrizC[j][i] = j*i;
                            }
              }


clock_gettime(CLOCK_REALTIME, &cgt1);
              for (i=0; i<dimension_matrices; i++){
                            for(j=0; j<dimension_matrices; j++){
                                          suma = 0;
                                          for (k=0; k<dimension_matrices; k++){
                                                        suma += (matrizB[i]
[k]*matrizC[k][j]);
                                          }
                                          matrizA[i][j]=suma;
                            }
              }
clock_gettime(CLOCK_REALTIME, &cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9));
              printf("Tiempo sin optimizar: %11.9f\t",ncgt);
              printf("\n");

              printf ("Resultado[0][0] = %d\n",matrizA[0][0]);
              printf ("Resultado[N-1,N-1]=%d\n",matrizA[dimension_matrices-1]
[dimension_matrices-1]);

for(int i=0; i<dimension_matrices; i++)
    free(matrizA[i]);
```

```
    free(matrizA);

for(int i=0; i<dimension_matrices; i++)
    free(matrizB[i]);

    free(matrizB);

for(int i=0; i<dimension_matrices; i++)
    free(matrizC[i]);

    free(matrizC);

    return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Utilizar 4 variables para llevar la suma.

Modificación b) –explicación-:

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmm-secuencial-modificado_a.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */
int main(int argc, char **argv){
            int i, j, k;
            int dimension_matrices;
            int suma1,suma2,suma3,suma4;

            struct timespec cgt1,cgt2; double ncgt;

            int **matrizB;
            int **matrizC;
            int **matrizA;

            if (argc < 2){
                        printf("Falta el número de componentes\n");
                        return(1);
            }

            dimension_matrices = atoi(argv[1]);

            if (dimension_matrices > MAX){
                        printf("Tamaño demasiado grande. No superar
%d\n\n",MAX);
                        return(1);
            }


            matrizB = (int **)malloc(dimension_matrices *
sizeof(int*));
            matrizC = (int **)malloc(dimension_matrices *
sizeof(int*));
            matrizA = (int **)malloc(dimension_matrices *
```

```
sizeof(int*));

            for (i=0; i<dimension_matrices; i++){
                        matrizB[i] = (int *)malloc(dimension_matrices
* sizeof(int));
                        matrizC[i] = (int *)malloc(dimension_matrices
* sizeof(int));
                        matrizA[i] = (int *)malloc(dimension_matrices
* sizeof(int));
            }


            for (j=0; j<dimension_matrices; j++){
                        for (i=0; i<dimension_matrices; i++){
                                    matrizB[j][i] = j+i;
                                    matrizC[j][i] = j*i;
                        }
            }



clock_gettime(CLOCK_REALTIME, &cgt1);
            for (i=0; i<dimension_matrices; i++){
                        for(j=0; j<dimension_matrices; j++){
                                    suma1 = 0;suma2 = 0;suma3 =
0;suma4 = 0;
                                    for (k=0; k<dimension_matrices-4;
k+=4){
                                                suma1 += (matrizB[i]
[k]*matrizC[k][j]);
                                                suma2 += (matrizB[i]
[k+1]*matrizC[k+1][j]);
                                                suma3 += (matrizB[i]
[k+2]*matrizC[k+2][j]);
                                                suma4 += (matrizB[i]
[k+3]*matrizC[k+3][j]);
                                    }
                                    matrizA[i]
[j]=suma1+suma2+suma3+suma4;
                        }
            }
clock_gettime(CLOCK_REALTIME, &cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
            printf("Tiempo optimizado: %11.9f\t",ncgt);
            printf("\n");

            printf ("Resultado[0][0] = %d\n",matrizA[0][0]);
            printf ("Resultado[N-1,N-1]=
%d\n",matrizA[dimension_matrices-1][dimension_matrices-1]);


            for(i=0; i<dimension_matrices; i++)
                        free(matrizA[i]);
    free(matrizA);
```

```
for(i=0; i<dimension_matrices; i++)
                        free(matrizB[i]);
    free(matrizB);
for(i=0; i<dimension_matrices; i++)
                        free(matrizC[i]);
    free(matrizC);

    return 0;
}
```

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

b) ...

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.000000318*
Modificación a)	0.000000285
Modificación b)	
...	

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

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	pmm-secuencial-modificado_b.s	pmm-secuencial-modificado_c.s
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */ .LFB39: .cfi_startproc movl dimension_matrices(%rip), %r11d pushq %r13 .cfi_def_cfa_offset 16 .cfi_offset 13, -16 movq %rsi, %r13	/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */ .LFB39: .cfi_startproc movl dimension_matrices(%rip), %eax testl %eax, %eax jle .L8 pushq	/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */

```
                pushq
                %r12
                .cfi_def_cfa_
offset 24
                .cfi_offset
12, -24
                pushq
                %rbp
                .cfi_def_cfa_
offset 32
                .cfi_offset
6, -32
                xorl
                %ebp, %ebp
                pushq
                %rbx
                .cfi_def_cfa_
offset 40
                .cfi_offset
3, -40
                movq
                %rdi, %r12
                movq
                %rdx, %rsi
                testl
                %r11d, %r11d
                jle
                .L3
.L8:
                movq
(%r12,%rbp,8), %rbx
                movq
0(%r13,%rbp,8), %rdx
                xorl
                %r10d, %r10d
                xorl
                %edi, %edi
                .p2align
4,,10
                .p2align 3
.L7:
                xorl
                %eax, %eax
                xorl
                %r8d, %r8d
                .p2align
4,,10
                .p2align 3
.L4:
                movq
                (%rsi,
%rax,8), %r9
                movl
                (%rdx,
%rax,4), %ecx
                addq
                $1, %rax
                imull
                (%r9,%r10),
%ecx
                addl
                %ecx, %r8d
                cmpl
                %eax, %r11d
                jg
                .L4
                movl
                %r15
                .cfi_def_c
fa_offset 16
                .cfi_offse
t 15, -16
                pushq
                %r14
                .cfi_def_c
fa_offset 24
                .cfi_offse
t 14, -24
                xorl
                %r8d, %r8d
                pushq
                %r13
                .cfi_def_c
fa_offset 32
                .cfi_offse
t 13, -32
                pushq
                %r12
                .cfi_def_c
fa_offset 40
                .cfi_offse
t 12, -40
                xorl
                %r13d,
%r13d
                pushq
                %rbp
                .cfi_def_c
fa_offset 48
                .cfi_offse
t 6, -48
                pushq
                %rbx
                .cfi_def_c
fa_offset 56
                .cfi_offse
t 3, -56
                xorl
                %r14d,
%r14d
                .p2align
4,,10
                .p2align 3
.L3:
                leal
                -1(%rax),
%r12d
                movq
                (%rsi,
%r13), %rcx
                xorl
                %eax, %eax
```

```
                %r8d, (%rbx,
%r10)
                movl
dimension_matrices(%rip),
%r11d
                addl
                $1, %edi
                addq
                $4, %r10
                cmpl
                %edi, %r11d
                jg
                .L7
                leal
                1(%rbp), %eax
                addq
                $1, %rbp
                cmpl
                %r11d, %eax
                jl
                .L8
.L3:
                popq
                %rbx
                .cfi_def_cfa_
offset 32
                popq
                %rbp
                .cfi_def_cfa_
offset 24
                popq
                %r12
                .cfi_def_cfa_
offset 16
                popq
                %r13
                .cfi_def_cfa_
offset 8
                ret
                .cfi_endproc
.LFE39:
                .size
multiplicarMatrices,
.-multiplicarMatrices
                .section
                .text.unlikel
y
.LCOLDE0:
                .text
.LHOTE0:
                .section
                .rodata.str1.
8,"aMS",@progbits,1
                .align 8
.LC1:
                .string
                "Falta el
n\303\272mero de
componentes"
                .align 8
.LC2:
                .string
                "Tama\303\261
o demasiado grande. No
superar %d\n\n"
                .section
                .rodata.str1.
```

```
                xorl
                %ebp, %ebp
                xorl
                %ebx, %ebx
                xorl
                %r11d,
%r11d
                shrl
                $2, %r12d
                xorl
                %r10d,
%r10d
                addq
                $1, %r12
                salq
                $4, %r12
                .p2align
4,,10
                .p2align 3
.L4:
                movq
                (%rdx,
%rax,2), %r15
                movl
                (%rcx,
%rax), %r9d
                imull
(%r15,%r8), %r9d
                movq
                8(%rdx,
%rax,2), %r15
                addl
                %r9d,
%r10d
                movl
                4(%rcx,
%rax), %r9d
                imull
(%r15,%r8), %r9d
                movq
                16(%rdx,
%rax,2), %r15
                addl
                %r9d,
%r11d
                movl
                8(%rcx,
%rax), %r9d
                imull
(%r15,%r8), %r9d
                movq
                24(%rdx,
```

```
1,"aMS",@progbits,1
.LC4:
                .string
                "Tiempo sin
optimizar: %11.9f\t"
.LC5:
                .string
                "Resultado[0]
[0] = %d\n"
.LC6:
                .string
                "Resultado[N-
1,N-1]=%d\n"
                .section
                .text.unlikel
y
.LCOLDB7:
                .section
                .text.startup
,"ax",@progbits
.LHOTB7:
                .p2align
4,,15
                .globl
                main
                .type
                main,
@function
```

```
%rax,2), %r15
                addl
                %r9d, %ebx
                movl
                12(%rcx,
%rax), %r9d
                addq
                $16, %rax
                imull

(%r15,%r8), %r9d
                addl
                %r9d, %ebp
                cmpq
                %r12, %rax
                jne
                .L4
                movq
                (%rdi,
%r13), %rax
                addl
                %r11d,
%r10d
                addl
                $1, %r14d
                addl
                %r10d,
%ebx
                addq
                $8, %r13
                addl
                %ebx, %ebp
                movl
                %ebp,
(%rax,%r8)
                movl

dimension_matrices(%ri
p), %eax
                cmpl
                %r14d,
%eax
                jg
                .L3
                popq
                %rbx
                .cfi_resto
re 3
                .cfi_def_c
fa_offset 48
                popq
                %rbp
                .cfi_resto
re 6
                .cfi_def_c
```

```
fa_offset 40
            popq
            %r12
            .cfi_resto
re 12
            .cfi_def_c
fa_offset 32
            popq
            %r13
            .cfi_resto
re 13
            .cfi_def_c
fa_offset 24
            popq
            %r14
            .cfi_resto
re 14
            .cfi_def_c
fa_offset 16
            popq
            %r15
            .cfi_resto
re 15
            .cfi_def_c
fa_offset 8
.L8:
            rep ret
            .cfi_endpr
oc
.LFE39:
            .size

multiplicarMatrices, .
-multiplicarMatrices
            .section
            .text.unli
kely
.LCOLDE0:
            .text
.LHOTE0:
            .section
            .rodata.st
r1.8,"aMS",@progbits,1
            .align 8
.LC1:
            .string
            "Falta el
n\303\272mero de
componentes"
            .align 8
.LC2:
            .string
            "Tama\303\
261o demasiado grande.
No superar %d\n\n"
```

```
            .section
            .rodata.st
r1.1,"aMS",@progbits,1
.LC4:
            .string
            "Tiempo
optimizado: %11.9f\t"
.LC5:
            .string
            "Resultado
[0][0] = %d\n"
.LC6:
            .string
            "Resultado
[N-1,N-1]=%d\n"
            .section
            .text.unli
kely
.LCOLDB7:
            .section
            .text.star
tup,"ax",@progbits
.LHOTB7:
            .p2align
4,,15
            .globl
            main
            .type
            main,
@function
```

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-original.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

main(){
            int X1=0, X2=0, R[39999], ii, i;
            struct timespec cgt1,cgt2; double ncgt;

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

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

```
                        }

                        if (X1<X2)
                                    R[ii]=X1;
                        else
                                    R[ii]=X2;
            }
            clock_gettime(CLOCK_REALTIME, &cgt2);
            ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
            printf("Tiempo sin optimizar: %11.9f\t",ncgt);
            printf("\n");
            printf("Resultado[0] %d\n",R[0]);
            printf("Resultado[39999] %d\n",R[39999]);

}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Fusionar dos for con la misma cabecera

Modificación b) –explicación-: Utilizar 4 variables para llevar X1 y X2

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figura1**`-modificado_a.c`**

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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

main(){
            int X1=0, X2=0, X1a, X1b, X1c, X1d, X2a, X2b, X2c, X2d,
R[39999], ii, i;
            struct timespec cgt1,cgt2; double ncgt;
            for(i=0; i<5000; i++){
                        s[i].a=0;
                        s[i].b=0;
            }
            clock_gettime(CLOCK_REALTIME, &cgt1);
            for (ii=0; ii<40000;ii++) {
                        X1a=0;X1b=0;X1c=0;X1d=0;
X2a=0;X2b=0;X2c=0;X2d=0;
                        for(i=0; i<5000;i+=4){
                                    X1a+=2*s[i].a+ii;
                                    X1b+=2*s[i+1].a+ii;
                                    X1c+=2*s[i+2].a+ii;
                                    X1d+=2*s[i+3].a+ii;
                        }
                        for(i=0; i<5000;i+=4){
                                    X2a+=3*s[i].b-ii;
                                    X2b+=3*s[i+1].b-ii;
                                    X2c+=3*s[i+2].b-ii;
                                    X2d+=3*s[i+3].b-ii;
```

```
                        }
                        X1=X1a+X1b+X1c+X1d;
                        X2=X2a+X2b+X2c+X2d;
                        if (X1<X2)
                                R[ii]=X1;
                        else
                                R[ii]=X2;
            }
            clock_gettime(CLOCK_REALTIME, &cgt2);
            ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double)
((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
            printf("Tiempo optimizado: %11.9f\t",ncgt);
            printf("\n");
            printf("Resultado[0] %d\n",R[0]);
            printf("Resultado[39999] %d\n",R[39999]);

}
```

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

b) figura1-modificado_b.c

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

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

main(){
        int X1=0, X2=0, R[39999], ii, i;
        struct timespec cgt1,cgt2; double ncgt;

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

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

                if (X1<X2)
                        R[ii]=X1;
                else
                        R[ii]=X2;
        }
        clock_gettime(CLOCK_REALTIME, &cgt2);
        ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
```

```
        printf("Tiempo sin optimizar: %11.9f\t",ncgt);
        printf("\n");
        printf("Resultado[0] %d\n",R[0]);
        printf("Resultado[39999] %d\n",R[39999]);

}
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.260233061*
Modificación a)	0.174488123
Modificación b)	0.188325503
...	

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

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	pmm-secuencial-modificado_b.s	pmm-secuencial-modificado_c.s
call clock_gettime xorl %r8d, %r8d .p2align 4,,10 .p2align 3 .L3: movl %r8d, %edi movl $s, %eax xorl %esi, %esi .p2align 4,,10 .p2align 3 .L4: movl (%rax), %edx addq $8, %rax leal	call clock_gettime .p2align 4,,10 .p2align 3 .L3: movl %r12d, %eax movl $s, %edx xorl %ebp, %ebp xorl %r11d, %r11d xorl %r10d, %r10d xorl %esi, %esi .p2align 4,,10 .p2align 3	call clock_gettime xorl %r8d, %r8d .p2align 4,,10 .p2align 3 .L3: movl %r8d, %edi movl $s, %eax xorl %ecx, %ecx xorl %esi, %esi .p2align 4,,10 .p2align 3 .L4: movl (%rax), %edx addq

```
            (%rdi,
%rdx,2), %edx
            addl
            %edx, %esi
            cmpq
            %rax, %rbx
            jne
            .L4
            movl
            $s+4, %eax
            xorl
            %ecx, %ecx
            .p2align
4,,10
            .p2align 3
.L5:
            movl
            (%rax),
%edx
            addq
            $8, %rax
            leal
            (%rdx,
%rdx,2), %edx
            subl
            %edi, %edx
            addl
            %edx, %ecx
            cmpq
            $s+40004,
%rax
            jne
            .L5
            cmpl
            %esi, %ecx
            cmovg
            %esi, %ecx
            movl
            %ecx,
32(%rsp,%r8,4)
            addq
            $1, %r8
            cmpq
            $40000,
%r8
            jne
            .L3
            leaq
            16(%rsp),
%rsi
            xorl
            %edi, %edi
            call

clock_gettime
```

```
.L4:
            movl
            (%rdx),
%ecx
            addq
            $32, %rdx
            leal
            (%rax,
%rcx,2), %ecx
            addl
            %ecx, %esi
            movl
            -24(%rdx),
%ecx
            leal
            (%rax,
%rcx,2), %ecx
            addl
            %ecx,
%r10d
            movl
            -16(%rdx),
%ecx
            leal
            (%rax,
%rcx,2), %ecx
            addl
            %ecx,
%r11d
            movl
            -8(%rdx),
%ecx
            leal
            (%rax,
%rcx,2), %ecx
            addl
            %ecx, %ebp
            cmpq
            %rdx, %rbx
            jne
            .L4
            movl
            $s+4, %edx
            xorl
            %edi, %edi
            xorl
            %r8d, %r8d
            xorl
            %r13d,
%r13d
            xorl
            %r9d, %r9d
            .p2align
4,,10
            .p2align 3
```

```
            $8, %rax
            leal
            (%rdi,
%rdx,2), %edx
            addl
            %edx, %esi
            movl
            -4(%rax),
%edx
            leal
            (%rdx,
%rdx,2), %edx
            subl
            %edi, %edx
            addl
            %edx, %ecx
            cmpq
            %rax, %rbx
            jne
            .L4
            cmpl
            %esi, %ecx
            cmovg
            %esi, %ecx
            movl
            %ecx,
32(%rsp,%r8,4)
            addq
            $1, %r8
            cmpq
            $40000,
%r8
            jne
            .L3
            leaq
            16(%rsp),
%rsi
            xorl
            %edi, %edi
            call

clock_gettime
```

```
.L5:
            movl
            (%rdx),
%ecx
            addq
            $32, %rdx
            leal
            (%rcx,
%rcx,2), %ecx
            subl
            %eax, %ecx
            addl
            %ecx, %r9d
            movl
            -24(%rdx),
%ecx
            leal
            (%rcx,
%rcx,2), %ecx
            subl
            %eax, %ecx
            addl
            %ecx,
%r13d
            movl
            -16(%rdx),
%ecx
            leal
            (%rcx,
%rcx,2), %ecx
            subl
            %eax, %ecx
            addl
            %ecx, %r8d
            movl
            -8(%rdx),
%ecx
            leal
            (%rcx,
%rcx,2), %ecx
            subl
            %eax, %ecx
            addl
            %ecx, %edi
            cmpq
            $s+40004,
%rdx
            jne
            .L5
            addl
            %r10d,
%esi
            addl
            %r13d,
%r9d
```

	addl %esi, %r11d
	addl %r9d, %r8d
	addl %r11d, %ebp
	addl %r8d, %edi
	cmpl %edi, %ebp
	cmovl %ebp, %edi
	movl %edi, 32(%rsp,%r12,4)
	addq $1, %r12
	cmpq $40000, %r12
	jne .L3
	leaq 16(%rsp), %rsi
	xorl %edi, %edi
	call clock_gettime	

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)

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 y 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

void daxpy(double *y, double *x, double a, unsigned n){
            unsigned i;
            for(i=0; i<n; i++)
                        y[i]+=a*x[i];
}

int main(int argc, char * argv[]){
            unsigned n=10;
            double x[10]={0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
            double y[10]={-1.0,-2.0,-3.0,-4.0,-5.0,-6.0,-7.0,-8.0,-9.0,-
10.0};
            const double a=1.5;
            int j;
            struct timespec cgt1,cgt2; double ncgt;

            clock_gettime(CLOCK_REALTIME, &cgt1);
            daxpy(y,x,a,n);
            clock_gettime(CLOCK_REALTIME, &cgt2);

            for (int j=0; j<10; j++)
                        printf("%d  X[]=%g  Y[]=%g\n",j,x[j],y[j]);

            ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));
            printf("Tiempo: %11.9f\t",ncgt);
            printf("\n");

}
```

	-O0	**-O2**	**-O3**
Tiempos ejec.	*0.000000240*	0.000000202	0.000000126

CAPTURAS DE PANTALLA:

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

Con optimización O0 se utilizan direcciones de la pila y con O2 registros, lo cual reduce bastante el tamaño del código para optimización O2 ya que no se opera con direcciones de la pila. Con optimización O3 el compilador realiza un desenrollado de bucle de 4 iteraciones, ya que suma 16 a la variable contadora, es decir, el tamaño de cuatro enteros.

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)

daxpyO0.s

```
.LFB2:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -24(%rbp)
        movq    %rsi, -32(%rbp)
        movsd   %xmm0, -40(%rbp)
        movl    %edx, -44(%rbp)
        movl    $0, -4(%rbp)
        jmp     .L2
.L3:
        movl    -4(%rbp), %eax
        leaq    0(,%rax,8), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movl    -4(%rbp), %edx
        leaq    0(,%rdx,8), %rcx
        movq    -24(%rbp), %rdx
        addq    %rcx, %rdx
        movsd   (%rdx), %xmm1
        movl    -4(%rbp), %edx
        leaq    0(,%rdx,8), %rcx
```

daxpyO2.s

```
.LFB41:
        .cfi_startproc
        xorl    %eax, %eax
        testl   %edx, %edx
        je      .L1
        .p2align 4,,10
        .p2align 3
.L5:
        movsd   (%rsi,%rax,8), %xmm1
        mulsd   %xmm0, %xmm1
        addsd   (%rdi,%rax,8), %xmm1
        movsd   %xmm1, (%rdi,%rax,8)
        addq    $1, %rax
        cmpl    %eax, %edx
        ja      .L5
.L1:
        rep ret
        .cfi_endproc
.LFE41:
        .size   daxpy, .-daxpy
        .section        .text.unlikely
.LCOLDE0:
        .text
.LHOTE0:
        .section        .rodata.str1.1,"aMS",@progbits,1
```

daxpyO3.s

```
.LFB41:
        .cfi_startproc
        testl   %edx, %edx
        je      .L22
        leaq    16(%rsi), %rax
        cmpq    %rax, %rdi
        leaq    16(%rdi), %rax
        setnb   %cl
        cmpq    %rax, %rsi
        setnb   %al
        orb     %al, %cl
        je      .L3
        cmpl    $6, %edx
        jbe     .L3
        movq    %rdi, %rax
        pushq   %rbx
        .cfi_def_cfa_offset 16
        .cfi_offset 3, -16
        salq    $60, %rax
        shrq    $63, %rax
        cmpl    %edx, %eax
        cmova   %edx, %eax
        xorl    %r9d, %r9d
        testl
```

```
                movq    -32(%rbp), %rdx
                addq    %rcx, %rdx
                movsd   (%rdx), %xmm0
                mulsd   -40(%rbp), %xmm0
                addsd   %xmm1, %xmm0
                movsd   %xmm0, (%rax)
                addl    $1, -4(%rbp)
.L2:
                movl    -4(%rbp), %eax
                cmpl    -44(%rbp), %eax
                jb      .L3
                nop
                popq    %rbp
                .cfi_def_cfa 7, 8
                ret
                .cfi_endproc
.LFE2:
                .size   daxpy, .-daxpy
                .section        .rodata
.LC21:
                .string "%d  X[]=%g Y[]=%g\n"
.LC23:
                .string "Tiempo: %11.9f\t"
                .text
                .globl  main
                .type   main, @function
.LC22:
                .string "%d  X[]= %g  Y[]=%g\n"
.LC24:
                .string "Tiempo: %11.9f\t"
                .section        .text.unlikely
.LCOLDB25:
                .section        .text.startup,"ax",@progbits
.LHOTB25:
                .p2align 4,,15
                .globl  main
                .type   main, @function
                %eax, %eax
                je      .L4
                movsd   (%rsi), %xmm1
                movl    $1, %r9d
                mulsd   %xmm0, %xmm1
                addsd   (%rdi), %xmm1
                movsd   %xmm1, (%rdi)
.L4:
                subl    %eax, %edx
                movapd  %xmm0, %xmm2
                leal    -2(%rdx), %r10d
                movl    %eax, %eax
                xorl    %ecx, %ecx
                unpcklpd        %xmm2, %xmm2
                salq    $3, %rax
                shrl    %r10d
                leaq    (%rdi, %rax), %r11
                xorl    %r8d, %r8d
                addl    $1, %r10d
                addq    %rsi, %rax
                leal    (%r10,%r10), %ebx
.L5:
                movupd  (%rax, %rcx), %xmm1
                addl    $1, %r8d
```

		mulpd %xmm2, %xmm1
		addpd (%r11,%rcx), %xmm1
		movaps %xmm1, (%r11,%rcx)
		addq $16, %rcx
		cmpl %r10d, %r8d
		jb .L5
		cmpl %ebx, %edx
		leal (%r9,%rbx), %eax
		je .L1
		mulsd (%rsi, %rax,8), %xmm0
		leaq (%rdi, %rax,8), %rdx
		addsd (%rdx), %xmm0
		movsd %xmm0, (%rdx)
		.L1:
		popq %rbx
		.cfi_resto re 3
		.cfi_def_c fa_offset 8
		.L22:
		rep ret
		.p2align 4,,10
		.p2align 3
		.L3:
		xorl %eax, %eax
		.p2align 4,,10
		.p2align 3
		.L8:
		movsd

		(%rsi, %rax,8), %xmm1
		mulsd %xmm0, %xmm1
		addsd (%rdi, %rax,8), %xmm1
		movsd %xmm1, (%rdi,%rax,8)
		addq $1, %rax
		cmpl %eax, %edx
		ja .L8
		rep ret
		.cfi_endproc
		.LFE41:
		.size daxpy, .-daxpy
		.section .text.unlikely
		.LCOLDE0:
		.text
		.LHOTE0:
		.section .rodata.str1.1,"aMS",@progbits,1
		.LC11:
		.string "%d X[]= %g Y[]=%g\n"
		.LC13:
		.string "Tiempo: %11.9f\t"
		.section .text.unlikely
		.LCOLDB14:
		.section .text.startup,"ax",@progbits
		.LHOTB14:
		.p2align 4,,15
		.globl main
		.type main,

		`@function`