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) i5-6300HQ CPU @ 2.30GHz*

Sistema operativo utilizado: *Ubuntu 16.04 LTS*

Versión de gcc utilizada: *gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.9)*

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

```
juanma@juanma-X550VX: ~/Escritorio/AC/BP4_Rubio_Rodriguez_JuanManuel
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP4_Rubio_Rodrigu
ez_JuanManuel] 2018-06-02 sábado
$lscpu
Arquitectura:          x86_64
modo(s) de operación de las CPUs:32-bit, 64-bit
Orden de bytes:        Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Hilo(s) de procesamiento por núcleo:1
Núcleo(s) por «socket»:4
Socket(s):             1
Modo(s) NUMA:          1
ID de fabricante:      GenuineIntel
Familia de CPU:        6
Modelo:                94
Model name:            Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz
Revisión:              3
CPU MHz:               800.028
CPU max MHz:           3200,0000
CPU min MHz:           800,0000
BogoMIPS:              4608.00
Virtualización:        VT-x
Caché L1d:             32K
Caché L1i:             32K
Caché L2:              256K
Caché L3:              6144K
NUMA node0 CPU(s):     0-3
```

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

   1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos (use –O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

   1.2 Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.

1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

Figura 1 . Código C++ que suma dos vectores

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

main()
{
   …
   for (ii=0; ii<40000;ii++) {
      X1=0; X2=0;
      for(i=0; i<5000;i++)  X1+=2*s[i].a+ii;
      for(i=0; i<5000;i++)  X2+=3*s[i].b-ii;

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

A) MULTIPLICACIÓN DE MATRICES:

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

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



int main(int argc, char** argv)
{
        int i, j, k;
        double t1, t2, total;
        double ncgt;
               struct timespec cgt1, cgt2;

if (argc<2){
                printf("Falta tamaño de matriz y vector\n");
                exit(-1);
        }

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

               //Reservo memoria para los vectores y la matriz

        double **M1, **M2, **M3;
        M1 = (double**) malloc(N*sizeof(double *));
        M2 = (double**) malloc(N*sizeof(double *));
        M3 = (double**) malloc(N*sizeof(double *));

        for (i=0; i<N; i++){
                M1[i] = (double*) malloc(N*sizeof(double));
                M2[i] = (double*) malloc(N*sizeof(double));
                M3[i] = (double*) malloc(N*sizeof(double));
        }

        //Inicializo matriz y vectores
        for (i=0; i<N; i++)
                for(j=0; j<N; j++){
```

```
                                        M1[i][j] = i;
                                        M2[i][j] = i;
                                        M3[i][j] = 0;
                                }

        clock_gettime(CLOCK_REALTIME, &cgt1);

        for (i=0; i<N;i++)
                for (j=0;j<N; j++)
                                        for (k=0; k<N; k++)
                                        M3[i][j] += M1[i][k] * M2[k][j];

        clock_gettime(CLOCK_REALTIME, &cgt2);

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

        //Imprimir el resultado y el tiempo de ejecución
        printf("\nTiempo(seg.):%11.9f\t / Tamaño:%u\t/ M3[0]=%8.6f M3[%d]=%8.6f\n",
ncgt,N,M3[0][0],N-1,M3[N-1][N-1]);

        if (N<5)
                for (i=0; i<N; i++)
                                        for (j=0; j<N; j++)
                                        printf(" M3[%d][%d]=%5.2f\n", i, j, M3[i]
[j]);

               //Libero memoria

        for (i=0; i<N; i++){
               free(M1[i]);
               free(M2[i]);
               free(M3[i]);
               }

        free(M1);
        free(M2);
        free(M3);

        return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: se ha hecho un desenrollado del bucle, con la restricción de que la matriz debe tener un tamaño múltiplo de 4.

Modificación b) –explicación-: se ha calculado la traspuesta de una de las matrices para realizar el cálculo por filas en lugar de por columnas.

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

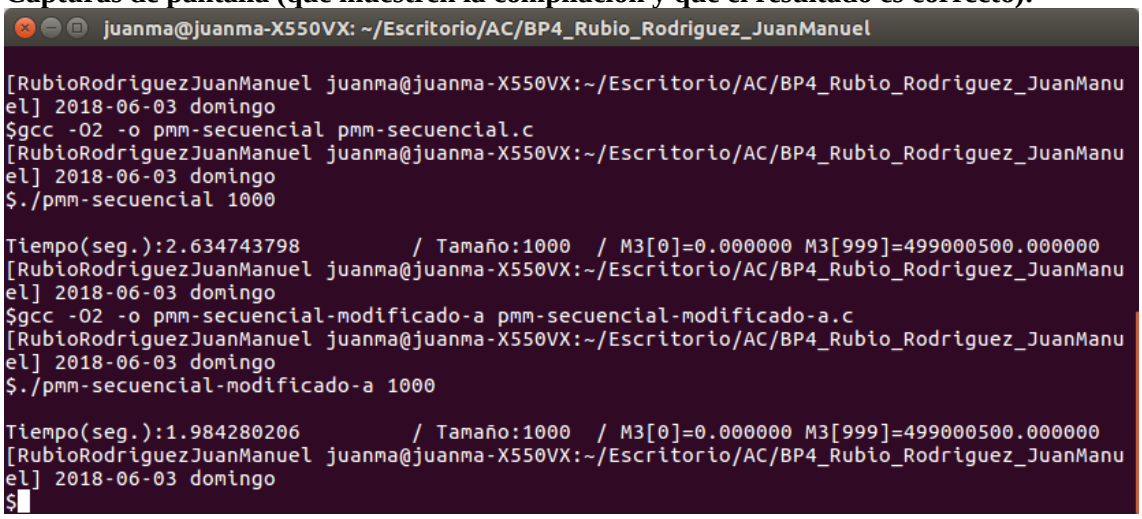
a) Captura de `pmm-secuencial-modificado_a.c`

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



int main(int argc, char** argv)
{
        int i, j, k;
               double coord1, coord2, coord3, coord4;
        double t1, t2, total;
        double ncgt;
               struct timespec cgt1, cgt2;

if (argc<2){
                printf("Falta tamaño de matriz y vector\n");
```

```
                exit(-1);
        }

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

               //Reservo memoria para los vectores y la matriz

        double **M1, **M2, **M3;
        M1 = (double**) malloc(N*sizeof(double *));
        M2 = (double**) malloc(N*sizeof(double *));
        M3 = (double**) malloc(N*sizeof(double *));

        for (i=0; i<N; i++){
                M1[i] = (double*) malloc(N*sizeof(double));
                M2[i] = (double*) malloc(N*sizeof(double));
                M3[i] = (double*) malloc(N*sizeof(double));
        }

        //Inicializo matriz y vectores
        for (i=0; i<N; i++)
                for(j=0; j<N; j++){
                                        M1[i][j] = i;
                                        M2[i][j] = i;
                                        M3[i][j] = 0;
                        }

        clock_gettime(CLOCK_REALTIME, &cgt1);

        for (i=0; i<N;i++){
                for (j=0;j<N; j+=4){
                        coord1 = coord2 = coord3 = coord4 = 0.0;
                                        for (k=0; k<N; k++){
                                        coord1 += M1[i][k] * M2[k][j];
                                        coord2 += M1[i][k] * M2[k][j+1];
                                        coord3 += M1[i][k] * M2[k][j+2];
                                        coord4 += M1[i][k] * M2[k][j+3];
                                        }
                        M3[i][j] = coord1;
                        M3[i][j+1] = coord2;
                        M3[i][j+2] = coord3;
                        M3[i][j+3] = coord4;
                        }
               }

        clock_gettime(CLOCK_REALTIME, &cgt2);

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

        //Imprimir el resultado y el tiempo de ejecución
        printf("\nTiempo(seg.):%11.9f\t / Tamaño:%u\t/ M3[0]=%8.6f M3[%d]=%8.6f\n",
ncgt,N,M3[0][0],N-1,M3[N-1][N-1]);

        if (N<5)
                for (i=0; i<N; i++)
                                        for (j=0; j<N; j++)
                                        printf(" M3[%d][%d]=%5.2f\n", i, j, M3[i]
[j]);

               //Libero memoria

        for (i=0; i<N; i++){
               free(M1[i]);
               free(M2[i]);
               free(M3[i]);
               }

        free(M1);
        free(M2);
        free(M3);
```

```
        return 0;
}
```

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



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

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



int main(int argc, char** argv)
{
        int i, j, k;
               double coord1, coord2, coord3, coord4;
        double t1, t2, total;
        double ncgt;
               struct timespec cgt1, cgt2;

if (argc<2){
                printf("Falta tamaño de matriz y vector\n");
                exit(-1);
        }

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

               //Reservo memoria para los vectores y la matriz

        double **M1, **M2, **M3, **M4;
        M1 = (double**) malloc(N*sizeof(double *));
        M2 = (double**) malloc(N*sizeof(double *));
        M3 = (double**) malloc(N*sizeof(double *));
               M4 = (double**) malloc(N*sizeof(double *));

        for (i=0; i<N; i++){
                M1[i] = (double*) malloc(N*sizeof(double));
                M2[i] = (double*) malloc(N*sizeof(double));
                M3[i] = (double*) malloc(N*sizeof(double));
                M4[i] = (double*) malloc(N*sizeof(double));
        }

        //Inicializo matriz y vectores
        for (i=0; i<N; i++)
                for(j=0; j<N; j++){
                                                M1[i][j] = i;
```

```
                                        M2[i][j] = i;
                                        M3[i][j] = 0;
                                                M4[i][j] = 0;
                        }

        clock_gettime(CLOCK_REALTIME, &cgt1);

                //Calculamos la traspuesta para trabajar con las filas de la matriz

                for (i=0; i<N; i++)
                        for (j=0; j<N; j++)
                                        M4[i][j] = M2[j][i];

        for (i=0; i<N;i++){
                 for (j=0;j<N; j+=4){
                        coord1 = coord2 = coord3 = coord4 = 0.0;
                                        for (k=0; k<N; k++){
                                        coord1 += M1[i][k] * M4[j][k];
                                        coord2 += M1[i][k] * M4[j+1][k];
                                        coord3 += M1[i][k] * M4[j+2][k];
                                        coord4 += M1[i][k] * M4[j+3][k];
                                        }
                        M3[i][j] = coord1;
                        M3[i][j+1] = coord2;
                        M3[i][j+2] = coord3;
                        M3[i][j+3] = coord4;
                        }
                }

        clock_gettime(CLOCK_REALTIME, &cgt2);

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

        //Imprimir el resultado y el tiempo de ejecución
        printf("\nTiempo(seg.):%11.9f\t / Tamaño:%u\t/ M3[0]=%8.6f M3[%d]=%8.6f\n",
ncgt,N,M3[0][0],N-1,M3[N-1][N-1]);

        if (N<5)
                 for (i=0; i<N; i++)
                                        for (j=0; j<N; j++)
                                        printf(" M3[%d][%d]=%5.2f\n", i, j, M3[i]
[j]);

                //Libero memoria

        for (i=0; i<N; i++){
                free(M1[i]);
                free(M2[i]);
                free(M3[i]);
                        free(M4[i]);
                }

        free(M1);
        free(M2);
        free(M3);
                free(M4);

        return 0;
}
```

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

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	2.651668919
Modificación a)	1.984280206
Modificación b)	1.225180202
...	

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Conforme vamos acumulando mejoras parciales sobre el código vemos la correspondiente mejora en el tiempo de ejecución. Se producen mejoras similares al utilizar un desenrollado de uno de los bucles for como de hacer uso de la localidad espacial al realizar el producto por filas, una mejora notable teniendo en cuenta que dentro del tiempo total se incluye el cálculo de la matriz traspuesta.

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

pmm-secuencial.s	pmm-secuencial-modificado_b.s	pmm-secuencial-modificado_c.s

.L9:
movq (%r12,%r11,8), %rdi
movq 0(%r13,%r11,8), %rsi
xorl %ecx, %ecx
.p2align 4,,10
.p2align 3
.L13:
movsd (%rdi,%rcx), %xmm1
xorl %eax, %eax
.p2align 4,,10
.p2align 3
.L10:
movq (%r15,%rax,8), %rdx
movsd (%rdx,%rcx), %xmm0
mulsd (%rsi,%rax,8), %xmm0
addq $1, %rax
cmpl %eax, %ebx
addsd %xmm0, %xmm1
ja.L10
movsd %xmm1, (%rdi,%rcx)
addq $8, %rcx
	.L9:
	movq 0(%r13,%r15,8), %r9
	movq 0(%rbp,%r15,8), %r10
	xorl %ecx, %ecx
	xorl %r11d, %r11d
	.p2align 4,,10
	.p2align 3
	.L13:
	leaq 8(%rcx), %r8
	leaq 16(%rcx), %rdi
	leaq 24(%rcx), %rsi
	movapd %xmm6, %xmm4
	xorl %eax, %eax
	movapd %xmm6, %xmm3
	movapd %xmm6, %xmm2
	movapd %xmm6, %xmm1
	.p2align 4,,10
	.p2align 3
	.L10:
	movq (%r12,%rax,8), %rdx
	movsd (%r9,%rax,8), %xmm0
		.L9:
		movq 0(%r13,%rcx), %rsi
		xorl %eax, %eax
		.p2align 4,,10
		.p2align 3
		.L10:
		movq (%r14,%rax,8), %rdx
		movsd (%rdx,%rcx), %xmm0
		movsd %xmm0, (%rsi,%rax,8)
		addq $1, %rax
		cmpl %eax, %r15d
		ja .L10
		addq $8, %rcx
		cmpq %rcx, %rdi
		jne .L9
		leaq 8(%r13), %r8
		movq %r13, 8(%rsp)
		movq 16(%rsp), %r13
		pxor %xmm6, %xmm6
		xorl %ebp, %ebp
		.L14:

```
cmpq %rcx, %r10
jne .L13
addq $1, %r11
cmpl %r11d, %ebx
ja .L9
.L12:
leaq 48(%rsp), %rsi
xorl %edi, %edi
call clock_gettime
```

```
addq $1, %rax
cmpl %eax, %ebx
movsd (%rdx,%rcx), %xmm5
mulsd %xmm0, %xmm5
addsd %xmm5, %xmm1
movsd (%rdx,%r8), %xmm5
mulsd %xmm0, %xmm5
addsd %xmm5, %xmm2
movsd (%rdx,%rdi), %xmm5
mulsd %xmm0, %xmm5
mulsd (%rdx,%rsi), %xmm0
addsd %xmm5, %xmm3
addsd %xmm0, %xmm4
ja .L10
addl $4, %r11d
movsd %xmm1, (%r10,%rcx)
movsd %xmm2, 8(%r10,%rcx)
movsd %xmm3, 16(%r10,%rcx)
movsd %xmm4, 24(%r10,%rcx)
addq $32, %rcx
cmpl %r11d, %ebx
ja .L13
addq $1, %r15
cmpl %r15d, %ebx
ja .L9
.L12:
leaq 48(%rsp), %rsi
xorl %edi, %edi
call clock_gettime
```

```
movq (%r12,%rbp,8), %rax
movq 0(%r13,%rbp,8), %r9
movq %r8, %r11
xorl %ebx, %ebx
leaq 8(%rax), %r10
.p2align 4,,10
.p2align 3
.L16:
movq -8(%r11), %rdi
movq (%r11), %rsi
xorl %eax, %eax
movq 8(%r11), %rcx
movq 16(%r11), %rdx
movapd %xmm6, %xmm4
movapd %xmm6, %xmm3
movapd %xmm6, %xmm2
movapd %xmm6, %xmm1
.p2align 4,,10
.p2align 3
.L12:
movsd (%r9,%rax,8), %xmm0
movsd (%rdi,%rax,8), %xmm5
mulsd %xmm0, %xmm5
addsd %xmm5, %xmm1
movsd (%rsi,%rax,8), %xmm5
mulsd %xmm0, %xmm5
addsd %xmm5, %xmm2
movsd (%rcx,%rax,8), %xmm5
mulsd %xmm0, %xmm5
mulsd (%rdx,%rax,8), %xmm0
addq $1, %rax
cmpl %eax, %r15d
addsd %xmm5, %xmm3
addsd %xmm0, %xmm4
ja .L12
addl $4, %ebx
movsd %xmm1, -8(%r10)
addq $32, %r11
movsd %xmm2, (%r10)
addq $32, %r10
movsd %xmm3, -24(%r10)
movsd %xmm4, -16(%r10)
cmpl %ebx, %r15d
ja .L16
addq $1, %rbp
cmpl %ebp, %r15d
ja .L14
movq 8(%rsp), %r13
.L15:
leaq 64(%rsp), %rsi
xorl %edi, %edi
call clock_gettime
```

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

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

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

int main (){

                int x1, x2, i, j;
                int R[40000];

                for (i = 0; i < 5000; i++){
                                s[i].a = 1;
                                s[i].b = 1;
                }

        double ncgt;
                struct timespec cgt1, cgt2;

        clock_gettime(CLOCK_REALTIME, &cgt1);

                for (i = 0; i < 40000; i++){
                                x1 = 0;
                                x2 = 0;
```

```
                                for (j = 0; j < 5000; j++)
                                        x1 += 2*s[j].a + i;
                                for (j = 0; j < 5000; j++)
                                        x2 += 3*s[j].b - i;

                if (x1 < x2)
                                R[i] = x1;
                else
                                R[i] = 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("R[0] = %d R[39999] = %d\n", R[0], R[39999]);
                printf("Tiempo total empleado en el cálculo: %11.9f \n" , ncgt);

}
```

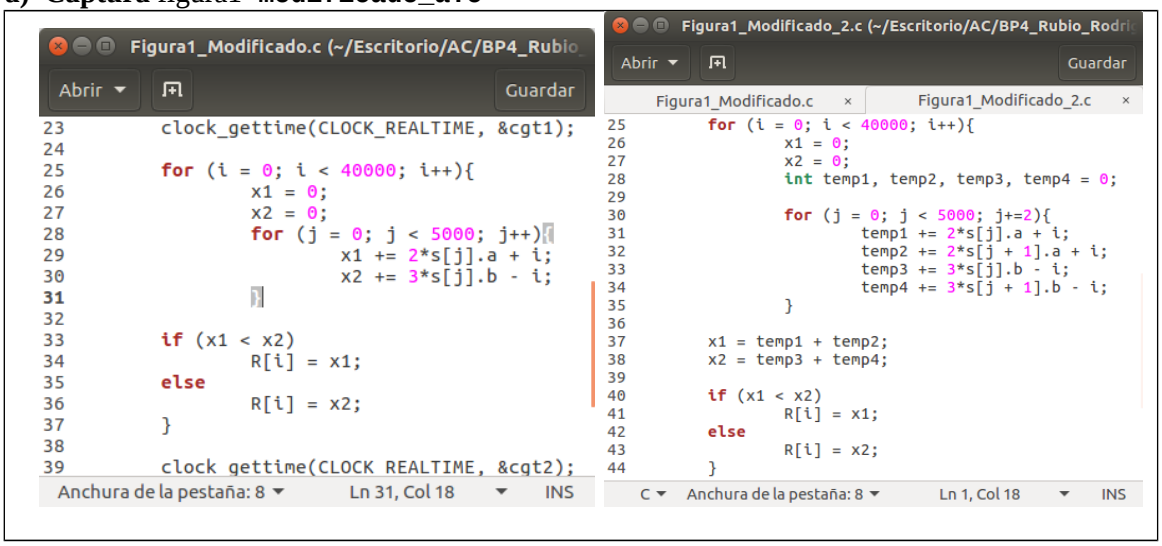
1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: La primera modificación consiste en eliminar el segundo de los bucles anidados y realizar las dos operaciones en el primero. Con ello nos ahorramos un bucle con las consecuentes comparaciones y saltos.

Modificación b) –explicación-: La segunda modificación consiste en desenrollar el bucle anidado, reduciendo las comparaciones y consiguiendo instrucciones independientes para su cómputo.

1.1. CÓDIGOS FUENTE MODIFICACIONES

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



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

```
juanma@juanma-X550VX: ~/Escritorio/AC/BP4_Rubio_Rodriguez_JuanManuel
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP4_Rubio_Rodrigu
ez_JuanManuel] 2018-06-02 sábado
$gcc -O2 -o Figura1 Figura1.c
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP4_Rubio_Rodrigu
ez_JuanManuel] 2018-06-02 sábado
$./Figura1
R[0] = 10000 R[39999] = -199980000
Tiempo total empleado en el cálculo: 0.242546398
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP4_Rubio_Rodrigu
ez_JuanManuel] 2018-06-02 sábado
$gcc -O2 -o Figura1_Modificado Figura1_Modificado.c
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP4_Rubio_Rodrigu
ez_JuanManuel] 2018-06-02 sábado
$./Figura1_Modificado
R[0] = 10000 R[39999] = -199980000
Tiempo total empleado en el cálculo: 0.168469874
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP4_Rubio_Rodrigu
ez_JuanManuel] 2018-06-02 sábado
$gcc -O2 -o Figura1_Modificado_2 Figura1_Modificado_2.c
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP4_Rubio_Rodrigu
ez_JuanManuel] 2018-06-02 sábado
$./Figura1_Modificado_2
R[0] = 4206240 R[39999] = 1689643664
Tiempo total empleado en el cálculo: 0.139463795
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP4_Rubio_Rodrigu
ez_JuanManuel] 2018-06-02 sábado
$
```

b)

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.242546398*
Modificación a)	0.168469874
Modificación b)	0.139463795
...	

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Observamos una mejora progresiva en tiempo conforme vamos implementando mejoras sobre el código. Quizá la más significativa haya sido eliminar el segundo bucle for anidado que realmente era una sobrecarga computacional superflua.

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

Figura1.s	Figura1_modificado.s	Figura1_Modificado_2.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 (%rdi,%rdx,2), %edx
addl %edx, %esi
cmpq $s+40000, %rax
jne .L4
movl $s+4, %eax
	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 $8, %rax
	leal (%rdi,%rdx,2), %edx
	addl %edx, %esi
	movl -4(%rax), %edx
	leal (%rdx,%rdx,2), %edx
		Call clock_gettime
		xorl %edi, %edi
		.p2align 4,,10
		.p2align 3
		.L3:
		movl %edi, %edx
		movl $s, %eax
		xorl %esi, %esi
		.p2align 4,,10
		.p2align 3
		.L4:
		movl (%rax), %ecx
		addq $16, %rax
		leal (%rdx,%rcx,2), %ecx
		addl %ecx, %ebx
		movl -8(%rax), %ecx
		leal (%rdx,%rcx,2), %ecx
		addl %ecx, %ebp

```
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 %ecx, %esi
cmovl %esi, %ecx
movl %ecx, 48(%rsp,%r8,4)
addq $1, %r8
cmpq $40000, %r8
jne .L3
leaq 32(%rsp), %rsi
xorl %edi, %edi
call clock_gettime
```

```
subl %edi, %edx
addl %edx, %ecx
cmpq $s+40000, %rax
jne .L4
cmpl %ecx, %esi
cmovl %esi, %ecx
movl %ecx, 48(%rsp,%r8,4)
addq $1, %r8
cmpq $40000, %r8
jne .L3
leaq 32(%rsp), %rsi
xorl %edi, %edi
call clock_gettime
```

```
movl -12(%rax), %ecx
leal (%rcx,%rcx,2), %ecx
subl %edx, %ecx
addl %ecx, %r12d
movl -4(%rax), %ecx
leal (%rcx,%rcx,2), %ecx
subl %edx, %ecx
addl %ecx, %esi
cmpq %rax, %r13
jne .L4
leal (%rbx,%rbp), %eax
addl %r12d, %esi
cmpl %esi, %eax
cmovl %eax, %esi
movl %esi, 48(%rsp,%rdi,4)
addq $1, %rdi
cmpq $40000, %rdi
jne.L3
leaq 32(%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 siguientes opciones de optimización del compilador: -O0, -Os, -O2, -O3. Explique las diferencias que se observan en el código justificando al mismo tiempo las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

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

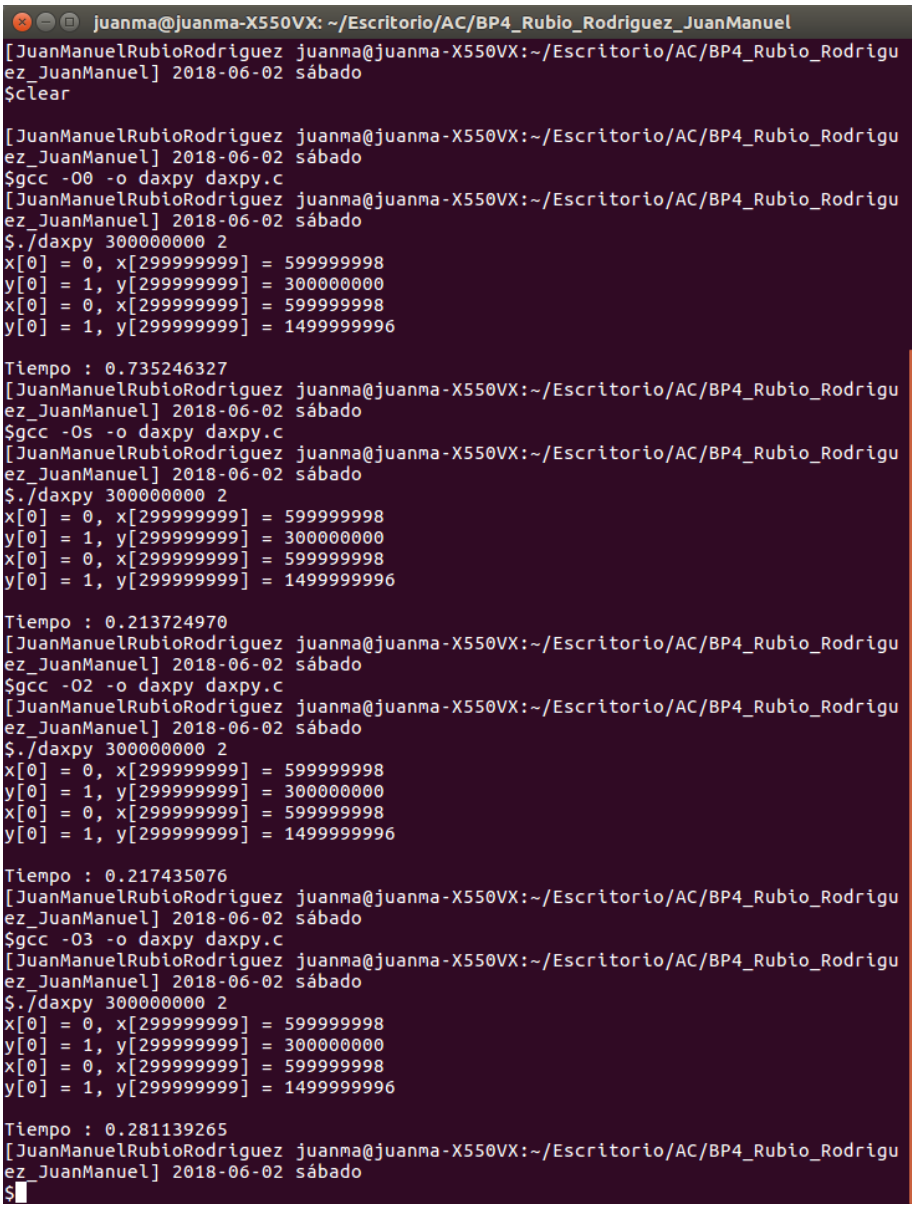
CAPTURA CÓDIGO FUENTE: `daxpy.c`

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

void daxpy(int *y, int *x, int a, unsigned n, struct timespec *cgt1, struct timespec
*cgt2)
{
    clock_gettime(CLOCK_REALTIME,cgt1);
    unsigned i;
    for (i=0; i<n; i++)
        y[i] += a*x[i];
    clock_gettime(CLOCK_REALTIME,cgt2);
}

int main(int argc, char *argv[])
{
    if (argc < 3)
    {
        fprintf(stderr, "ERROR: falta tamano del vector y constante\n");
        exit(1);
    }
```

```
    unsigned n = strtol(argv[1], NULL, 10);
    int a = strtol(argv[2], NULL, 10);
    int *y, *x;
    y = (int*) malloc(n*sizeof(int));
    x = (int*) malloc(n*sizeof(int));

    unsigned i;
    for (i=0; i<n; i++){
        y[i] = i+1;
        x[i] = i*2;
    }

    printf("x[0] = %i, x[%i] = %i\n", x[0], n-1, x[n-1]);
    printf("y[0] = %i, y[%i] = %i\n", y[0], n-1, y[n-1]);

    struct timespec cgt1,cgt2; double ncgt;

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

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

    printf("x[0] = %i, x[%i] = %i\n", x[0], n-1, x[n-1]);
    printf("y[0] = %i, y[%i] = %i\n", y[0], n-1, y[n-1]);
    printf("\nTiempo : %11.9f\n", ncgt);

    free(y);
    free(x);

    return 0;
}
```

	-O0	**-Os**	**-O2**	**-O3**
Tiempos ejec.	0.735246 327	0.21372 4970	0.217435 076	0.281139 265

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



COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:
La primera diferencia y más evidente, es que con O0, el compilador trabaja con la pila y con el resto trabaja con registros, incrementando notablemente la velocidad.
En O3, aparentemente se realiza un desenrollado del bucle for que, en este caso, no nos da una mejora de tiempo sino que lo empeora.
La compilación con Os y O2 en este caso es muy similar, incluido en tiempo.

CÓDIGO EN ENSAMBLADOR (no es necesario introducir aquí el código como captura de pantalla, ajustar el tamaño de la letra para que una instrucción no ocupe más de un renglón)**:**
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpy00.s	daxpy0s.s	daxpy02.s	daxpy03.s
.LFB2: .cfi_startproc pushq %rbp cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp	.LFB20: .cfi_startproc pushq %r13 .cfi_def_cfa_offset 16 .cfi_offset 13, -16 pushq %r12	.LFB38: .cfi_startproc pushq %r14 .cfi_def_cfa_offset 16 .cfi_offset 14, -16 pushq %r13	.LFB38: .cfi_startproc pushq %r14 .cfi_def_cfa_offset 16 .cfi_offset 14, -16 pushq %r13

```
.cfi_def_cfa_register 6
subq $64, %rsp
movq %rdi, -24(%rbp)
movq %rsi, -32(%rbp)
movl %edx, -36(%rbp)
movl %ecx, -40(%rbp)
movq %r8, -48(%rbp)
movq %r9, -56(%rbp)
movq -48(%rbp), %rax
movq %rax, %rsi
movl $0, %edi
call clock_gettime
movl $0, -4(%rbp)
jmp .L2
.L3:
movl -4(%rbp), %eax
leaq 0(,%rax,4), %rdx
movq -24(%rbp), %rax
addq %rax, %rdx
movl -4(%rbp), %eax
leaq 0(,%rax,4), %rcx
movq -24(%rbp), %rax
addq %rcx, %rax
movl (%rax), %ecx
movl -4(%rbp), %eax
leaq 0(,%rax,4), %rsi
movq -32(%rbp), %rax
addq %rsi, %rax
movl (%rax), %eax
imull -36(%rbp), %eax
addl %ecx, %eax
movl %eax, (%rdx)
addl $1, -4(%rbp)
.L2:
movl -4(%rbp), %eax
cmpl -40(%rbp), %eax
jb .L3
movq -56(%rbp), %rax
movq %rax, %rsi
movl $0, %edi
call clock_gettime
nop
leave
.cfi_def_cfa 7, 8
ret
.cfi_endproc
```

```
.cfi_def_cfa_offset 24
.cfi_offset 12, -24
movq %rsi, %r12
pushq %rbp
.cfi_def_cfa_offset 32
.cfi_offset 6, -32
pushq %rbx
.cfi_def_cfa_offset 40
.cfi_offset 3, -40
movq %r8, %rsi
movq %rdi, %rbx
xorl %edi, %edi
movl %edx, %r13d
subq $24, %rsp
.cfi_def_cfa_offset 64
movl %ecx, %ebp
movq %r9, 8(%rsp)
call clock_gettime
movq 8(%rsp), %r9
xorl %eax, %eax
.L2:
cmpl %eax, %ebp
jbe .L6
movl (%r12,%rax,4), %edx
imull %r13d, %edx
addl %edx, (%rbx,%rax,4)
incq %rax
jmp.L2
.L6:
addq $24, %rsp
.cfi_def_cfa_offset 40
movq %r9, %rsi
xorl %edi, %edi
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
jmpclock_gettime
.cfi_endproc
.LFE20:
```

```
.cfi_def_cfa_offset 24
.cfi_offset 13, -24
movq %rsi, %r13
pushq %r12
.cfi_def_cfa_offset 32
.cfi_offset 12, -32
 pushq %rbp
.cfi_def_cfa_offset 40
.cfi_offset 6, -40
movq %r8, %rsi
pushq %rbx
.cfi_def_cfa_offset 48
.cfi_offset 3, -48
movl %ecx, %ebp
movq %rdi, %rbx
xorl %edi, %edi
movl %edx, %r12d
movq %r9, %r14
call clock_gettime
xorl %eax, %eax
testl %ebp, %ebp
je .L4
.p2align 4,,10
.p2align 3
.L5:
movl 0(%r13,%rax,4), %esi
imull %r12d, %esi
addl %esi, (%rbx,%rax,4)
addq $1, %rax
cmpl %eax, %ebp
ja .L5
.L4:
popq %rbx
.cfi_def_cfa_offset 40
movq %r14, %rsi
xorl %edi, %edi
popq %rbp
.cfi_def_cfa_offset 32
popq %r12
.cfi_def_cfa_offset 24
popq %r13
.cfi_def_cfa_offset 16
popq %r14
.cfi_def_cfa_offset 8
jmp clock_gettime
.cfi_endproc
```

```
.cfi_def_cfa_offset 24
.cfi_offset 13, -24
movl %ecx, %r14d
pushq %r12
.cfi_def_cfa_offset 32
.cfi_offset 12, -32
pushq %rbp
.cfi_def_cfa_offset 40
.cfi_offset 6, -40
movq %rsi, %r12
pushq %rbx
.cfi_def_cfa_offset 48
.cfi_offset 3, -48
movq %r8, %rsi
movq %rdi, %rbx
xorl %edi, %edi
movl %edx, %r13d
movq %r9, %rbp
subq $16, %rsp
.cfi_def_cfa_offset 64
call clock_gettime
testl %r14d, %r14d
je .L10
leaq 16(%r12), %rax
cmpq %rax, %rbx
leaq 16(%rbx), %rax
setnb %dl
cmpq %rax, %r12
setnb %al
orb %al, %dl
je .L3
cmpl $6, %r14d
jbe .L3
movq %rbx, %rax
andl $15, %eax
shrq $2, %rax
negq %rax
andl $3, %eax
cmpl %r14d, %eax
cmova %r14, %rax
xorl %edx, %edx
testl %eax, %eax
je .L4
movl (%r12), %edx
imull %r13d, %edx
addl %edx, (%rbx)
cmpl $1, %eax
movl $1, %edx
je .L4
movl 4(%r12), %edx
imull %r13d, %edx
addl %edx, 4(%rbx)
cmpl $3, %eax
movl $2, %edx
jne .L4
movl 8(%r12), %edx
imull %r13d, %edx
addl %edx, 8(%rbx)
movl $3, %edx
.L4:
movl %r14d, %edi
movl %r13d, 12(%rsp)
xorl %ecx, %ecx
subl %eax, %edi
movd 12(%rsp), %xmm4
salq $2, %rax
leal -4(%rdi), %esi
leaq (%rbx,%rax), %r10
xorl %r9d, %r9d
pshufd $0, %xmm4, %xmm2
addq %r12, %rax
shrl $2, %esi
addl $1, %esi
movdqa %xmm2, %xmm3
leal 0(,%rsi,4), %r8d
psrlq $32, %xmm3
.L6:
movdqu (%rax,%rcx), %xmm0
addl $1, %r9d
movdqa %xmm0, %xmm1
psrlq $32, %xmm0
pmuludq %xmm3, %xmm0
pshufd $8, %xmm0, %xmm0
pmuludq %xmm2, %xmm1
pshufd $8, %xmm1, %xmm1
punpckldq %xmm0, %xmm1
movdqa (%r10,%rcx), %xmm0
paddd %xmm1, %xmm0
movaps %xmm0, (%r10,%rcx)
addq $16, %rcx
cmpl %esi, %r9d
jb .L6
addl %r8d, %edx
```

			cmpl %r8d, %edi
			je .L10
			movl %edx, %eax
			movl (%r12,%rax,4), %ecx
			imull %r13d, %ecx
			addl %ecx, (%rbx,%rax,4)
			leal 1(%rdx), %eax
			cmpl %eax, %r14d
			jbe .L10
			movl (%r12,%rax,4), %ecx
			addl $2, %edx
			imull %r13d, %ecx
			addl %ecx, (%rbx,%rax,4)
			cmpl %edx, %r14d
			jbe .L10
			movl %edx, %eax
			imull (%r12,%rax,4), %r13d
			addl %r13d, (%rbx,%rax,4)
			.L10:
			addq $16, %rsp
			.cfi_remember_state
			.cfi_def_cfa_offset 48
			movq %rbp, %rsi
			xorl %edi, %edi
			popq %rbx
			.cfi_def_cfa_offset 40
			popq %rbp
			.cfi_def_cfa_offset 32
			popq %r12
			.cfi_def_cfa_offset 24
			popq %r13
			.cfi_def_cfa_offset 16
			popq%r14
			.cfi_def_cfa_offset 8
			jmp clock_gettime
			.p2align 4,,10
			.p2align 3