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

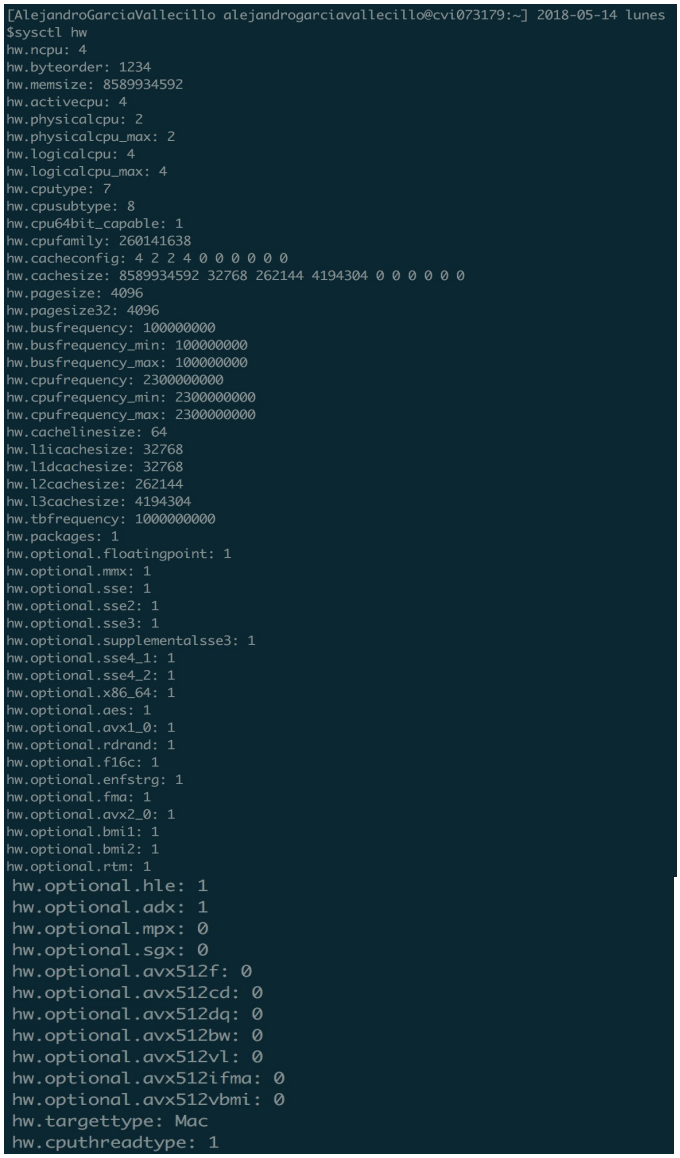
Estudiante (nombre y apellidos):
Grupo de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i5-7360U CPU @ 2.30GHz*

Sistema operativo utilizado: *MacOS*

Versión de gcc utilizada: *Version 7*

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



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

#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
                     // globales (su longitud no estará limitada por el ...
                     // tamaño de la pila del programa)
//#define VECTOR_DYNAMIC    // descomentar para que los vectores sean variables ...
                            // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 15000
double m1[MAX][MAX], m2[MAX][MAX], m3[MAX][MAX];
#endif
```

```c
void multi(int n){

    int i, j, k;
    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];
            }
    }

}

int main(int argc, char ** argv)
{
    if(argc < 2){
        printf("Falta el número de filas/columnas. \n");
        exit(-1);
    }

    int n = atoi(argv[1]);

    #ifdef VECTOR_GLOBAL
    if (n>MAX) n=MAX;
    #endif
    #ifdef VECTOR_DYNAMIC
    double **m1, **m2, **m3;
    m1 = (double**) malloc(n*sizeof(double*));
    m2 = (double**) malloc(n*sizeof(double*));
    m3 = (double**) malloc(n*sizeof(double*));
    for(int 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));
    }
    if ( ( m1==NULL) || (m2==NULL) || (m3==NULL) ){
        printf("Error en la reserva de espacio para los vectores y matriz\n");
        exit(-2);
    }
    #endif

    int i, j, k;
    struct timespec cgt1, cgt2; double ncgt; //para tiempo de ejecución

   for(i=0; i<n; i++)
       for(j=0; j<n; j++){
           m1[i][j] = 5;
           m2[i][j] = 2;
           m3[i][j] = 0;
       }

   clock_gettime(CLOCK_REALTIME,&cgt1);

   multi(n);

   clock_gettime(CLOCK_REALTIME,&cgt2);

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

   printf("El componente 0 del vector resultante es: %.0f \n",m3[0][0]);
   printf("El componente %d del vector resultante es: %.0f \n", n-1, m3[n-1][n-1]);

   printf("Tiempo(seg.):%11.9f \n", ncgt);

   #ifdef VECTOR_DYNAMIC
   free(m1); // libera el espacio reservado para m1
   free(m2); // libera el espacio reservado para m2
   free(m3); // libera el espacio reservado para m3
   #endif

   return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: La primera modificación que he realizado ha sido cambiar el bucle de j por el de k y viceversa. Con esta modificación hemos hecho que sea mas fácil recorrer la matriz 2 y 3 en memoria.

Modificación b) –explicación-: Desenrollado del bucle de j de 5 en 5.

1.1. CÓDIGOS FUENTE MODIFICACIONES

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

```
void multi(int n){

    int i, j, k;
    for(i=0; i<n; i++){
        for(k=0; k<n; k++)
            for(j=0; j<n; j++){
                m3[i][j] += m1[i][k]*m2[k][j];
            }
    }

}
```

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

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -O2 pmm-secuencial.c -o pmm-secuencial
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -O2 pmm-secuencial-modificado_a.c -o pmm-secuencial-modificado_a
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./pmm-secuencial 1500
El componente 0 del vector resultante es: 15000
El componente 1499 del vector resultante es: 15000
Tiempo(seg.):9.955650000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./pmm-secuencial-modificado_a 1500
El componente 0 del vector resultante es: 15000
El componente 1499 del vector resultante es: 15000
Tiempo(seg.):2.275844000
```

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

```
void multi(int n){

    int i, j, k;
    for(i=0; i<n; i++){
        for(k=0; k<n; k++){
            for(j=0; j<n; j+=5){
                m3[i][j] += m1[i][k]*m2[k][j];
                m3[i][j+1] += m1[i][k]*m2[k][j+1];
                m3[i][j+2] += m1[i][k]*m2[k][j+2];
                m3[i][j+3] += m1[i][k]*m2[k][j+3];
                m3[i][j+4] += m1[i][k]*m2[k][j+4];
            }
        }
    }

}
```

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

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -O2 pmm-secuencial-modificado_b.c -o pmm-secuencial-modificado_b
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./pmm-secuencial-modificado_b 1500
El componente [0][0] de la matriz resultante es: 15000
El componente [1499][1499] de la matriz resultante es: 15000
Tiempo(seg.):2.166628000
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*9,96*
Modificación a)	2,27
Modificación b)	2,16

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

a) Al hacer esta modificación optimizamos el acceso a memoria, ya que las matrices se guardan por filas y de esta manera no salta de una a otra antes de terminar la fila anterior. Y en la captura podemos ver como el tiempo se reduce bastante.

b) He desenrollado el bucle de 5 en 5, por lo que me ahorro bastantes saltos hacia atrás y, por ende, tiempo.

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

```
_multi:
LFB4:
testl           %edi, %edi
jle             L2
leal            -1(%rdi), %edx
movq   _m2@GOTPCREL(%rip), %rcx
imulq     $-120000, %rdx, %r8
movq   _m3@GOTPCREL(%rip), %rax
movq   _m1@GOTPCREL(%rip), %rdi
leaq        120000(%rax), %r10
movq            %r8, %rsi
subq            %rcx, %r10
subq            $120000, %r8
subq            %rcx, %rsi
addq            %rax, %rsi
imulq       $120000, %rdx, %rax
imulq       $120008, %rdx, %rdx
leaq    120000(%rcx,%rax), %r11
leaq    120008(%rcx,%rdx), %r9
.align 4,0x90
L3:
movq            %r11, %rcx
.align 4,0x90
L6:
movsd  -120000(%rsi,%rcx), %xmm1
leaq          (%r8,%rcx), %rax
movq            %rdi, %rdx
.align 4,0x90
L4:
movsd           (%rdx), %xmm0
addq            $120000, %rax
addq            $8, %rdx
mulsd       -120000(%rax), %xmm0
cmpq            %rax, %rcx
```

pmm-secuencial-modificado_a.s

```
_multi:
LFB4:
testl           %edi, %edi
jle             L9
leal            -1(%rdi), %eax
movq   _m3@GOTPCREL(%rip), %rdx
pushq           %rbx
LCFI0:
imulq       $120000, %rax, %rcx
movq   _m2@GOTPCREL(%rip), %r11
movq   _m1@GOTPCREL(%rip), %r10
leaq          8(,%rax,8), %rsi
leaq        120000(%rdx), %r9
leaq          (%r9,%rcx), %rbx
leaq    120000(%r11,%rcx), %r8
.align 4,0x90
L3:
movq            %r11, %rcx
movq            %r10, %rdi
.align 4,0x90
L6:
movsd           (%rdi), %xmm1
xorl            %eax, %eax
.align 4,0x90
L4:
movsd       (%rcx,%rax), %xmm0
mulsd            %xmm1, %xmm0
addsd       (%rdx,%rax), %xmm0
movsd      %xmm0, (%rdx,%rax)
addq            $8, %rax
cmpq            %rax, %rsi
jne             L4
addq            $120000, %rcx
addq            $8, %rdi
```

pmm-secuencial-modificado_b.s

```
_multi:
LFB4:
testl           %edi, %edi
jle             L9
leal            -1(%rdi), %eax
movq   _m3@GOTPCREL(%rip), %r9
pushq           %r12
LCFI0:
imulq       $120000, %rax, %rax
pushq           %rbp
LCFI1:
movq   _m2@GOTPCREL(%rip), %rbp
pushq           %rbx
LCFI2:
movq   _m1@GOTPCREL(%rip), %rbx
leaq         120000(%r9), %r11
leaq          (%r11,%rax), %r12
leaq    120000(%rbp,%rax), %r10
.align 4,0x90
L3:
movq            %rbp, %rsi
movq            %rbx, %r8
.align 4,0x90
L6:
movsd           (%r8), %xmm0
movq            %rsi, %rdx
movq            %r9, %rax
xorl            %ecx, %ecx
.align 4,0x90
L4:
movsd           (%rdx), %xmm1
addl            $5, %ecx
addq            $40, %rax
addq            $40, %rdx
```

```
addsd           %xmm0, %xmm1
jne             L4
movsd  %xmm1, -120000(%rsi,%rcx)
addq            $8, %rcx
cmpq            %r9, %rcx
jne             L6
addq            $120000, %rsi
addq            $120000, %rdi
cmpq            %r10, %rsi
jne             L3
L2:
leaq            lC0(%rip), %rdi
xorl            %eax, %eax
jmp             _printf
```

```
cmpq            %r8, %rcx
jne             L6
addq            $120000, %r10
cmpq            %rbx, %r9
movq            %r9, %rdx
je              L2
addq            $120000, %r9
jmp             L3
L2:
popq            %rbx
LCFI1:
leaq            lC0(%rip), %rdi
jmp             _puts
L9:
leaq            lC0(%rip), %rdi
jmp             _puts
```

```
mulsd           %xmm0, %xmm1
addsd       -40(%rax), %xmm1
movsd          %xmm1, -40(%rax)
movsd       -32(%rdx), %xmm1
mulsd           %xmm0, %xmm1
addsd       -32(%rax), %xmm1
movsd          %xmm1, -32(%rax)
movsd       -24(%rdx), %xmm1
mulsd           %xmm0, %xmm1
addsd       -24(%rax), %xmm1
movsd          %xmm1, -24(%rax)
movsd       -16(%rdx), %xmm1
mulsd           %xmm0, %xmm1
addsd       -16(%rax), %xmm1
movsd          %xmm1, -16(%rax)
movsd           -8(%rdx), %xmm1
mulsd           %xmm0, %xmm1
addsd           -8(%rax), %xmm1
movsd           %xmm1, -8(%rax)
cmpl            %ecx, %edi
jg              L4
addq            $120000, %rsi
addq            $8, %r8
cmpq            %r10, %rsi
jne             L6
addq            $120000, %rbx
cmpq            %r12, %r11
movq            %r11, %r9
je              L2
addq            $120000, %r11
jmp             L3
L2:
popq            %rbx
LCFI3:
leaq            lC0(%rip), %rdi
popq            %rbp
LCFI4:
popq            %r12
LCFI5:
jmp             _puts
L9:
leaq            lC0(%rip), %rdi
jmp             _puts
```

Como podemos ver, que el programa sea más rápido no es sinónimo de que sea mas corto.

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

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

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

int main()
{
    int ii, X1, X2, i;
    int R[40000];

    struct timespec cgt1, cgt2;
    double ncgt;
```

```
    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("El componente 0 de R es: %d \n", R[0]);
    printf("El componente %d de R es: %d \n", 39999, R[39999]);

    printf("Tiempo(seg.):%11.9f \n", ncgt);

    return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Ya que los dos for hacen lo mismo, dejamos solo uno con las dos operacines.

Modificación b) –explicación-: Desenrrollamos el bucle como en el ejercicio anterior.

1.1. CÓDIGOS FUENTE MODIFICACIONES

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

```
    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);
```

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

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -O2 figura1-original.c -o figura1-original
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -O2 figura1-modificado_a.c -o figura1-modificado_a
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./figura1-original
El componente 0 de R es: 0
El componente 39999 de R es: -199995000
Tiempo(seg.):0.200522000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./figura1-modificado_a
El componente 0 de R es: 0
El componente 39999 de R es: -199995000
Tiempo(seg.):0.144857000
```

b) Captura figura1 **`-modificado_b.c`**

```
    clock_gettime(CLOCK_REALTIME, &cgt1);
    for (ii = 0; ii < 40000; ii++) {

        X1 = 0; X2 = 0;

        for(i = 0; i < 5000; i+=5){
            X1 += 2 * s[i].a + ii;
            X2 += 3 * s[i].b - ii;
            X1 += 2 * s[i+1].a + ii;
            X2 += 3 * s[i+1].b - ii;
            X1 += 2 * s[i+2].a + ii;
            X2 += 3 * s[i+2].b - ii;
            X1 += 2 * s[i+3].a + ii;
            X2 += 3 * s[i+3].b - ii;
            X1 += 2 * s[i+4].a + ii;
            X2 += 3 * s[i+4].b - ii;
        }

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

    }
    clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -O2 figura1-modificado_b.c -o figura1-modificado_b
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./figura1-modificado_b
El componente 0 de R es: 0
El componente 39999 de R es: -199995000
Tiempo(seg.):0.113019000
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0,21*
Modificación a)	*0,14*
Modificación b)	*0,11*

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Como se puede observar en la tabla de tiempos, agrupando los dos for nos ahorramos muchos saltos y con el desenrollado muchos más, lo que hace que la ejecución sea más rapida.

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

```
call            _clock_gettime
movq    _s@GOTPCREL(%rip), %r11
leaq            48(%rsp), %rbx
xorl            %r10d, %r10d
leaq       40000(%r11), %r9
leaq       40004(%r11), %r8
.align 4
L2:
movl            %r10d, %edi
movq            %r11, %rax
xorl            %esi, %esi
.align 4
L3:
movl            (%rax), %edx
addq            $8, %rax
leal      (%rdi,%rdx,2), %edx
addl            %edx, %esi
cmpq            %r9, %rax
jne             L3
leaq            4(%r11), %rax
xorl            %ecx, %ecx
.align 4
L4:
movl            (%rax), %edx
addq            $8, %rax
leal      (%rdx,%rdx,2), %edx
subl            %edi, %edx
addl            %edx, %ecx
cmpq            %r8, %rax
jne             L4
cmpl            %esi, %ecx
cmovg           %esi, %ecx
movl      %ecx, (%rbx,%r10,4)
addq            $1, %r10
cmpq            $40000, %r10
jne             L2
leaq            32(%rsp), %rsi
xorl            %edi, %edi
call            _clock_gettime
```

pmm-secuencial-modificado_a.s

```
call            _clock_gettime
movq    _s@GOTPCREL(%rip), %r10
xorl            %r9d, %r9d
leaq            48(%rsp), %r11
leaq          40000(%r10), %r8
.align 4
L2:
movl            %r9d, %edi
movq            %r10, %rax
xorl            %ecx, %ecx
xorl            %esi, %esi
.align 4
L3:
movl            (%rax), %edx
addq            $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, %r8
jne             L3
cmpl            %ecx, %esi
cmovl           %esi, %ecx
movl         %ecx, (%r11,%r9,4)
addq            $1, %r9
cmpq            $40000, %r9
jne             L2
leaq            32(%rsp), %rsi
xorl            %edi, %edi
call            _clock_gettime
```

pmm-secuencial-modificado_b.s

```
call            _clock_gettime
movq     _s@GOTPCREL(%rip), %r9
xorl            %r8d, %r8d
leaq            48(%rsp), %r10
leaq          40000(%r9), %rdi
.align 4
L2:
movl            %r8d, %edx
movq            %r9, %rax
xorl            %ecx, %ecx
xorl            %r11d, %r11d
.align 4
L3:
movl            (%rax), %esi
addq            $40, %rax
leal      (%rdx,%rsi,2), %ebx
movl            -36(%rax), %esi
addl            %ebx, %r11d
leal      (%rsi,%rsi,2), %esi
subl            %edx, %esi
addl            %esi, %ecx
movl            -32(%rax), %esi
leal      (%rdx,%rsi,2), %ebx
movl            -28(%rax), %esi
addl            %r11d, %ebx
leal      (%rsi,%rsi,2), %esi
subl            %edx, %esi
addl            %esi, %ecx
movl            -24(%rax), %esi
leal      (%rdx,%rsi,2), %r11d
movl            -20(%rax), %esi
addl            %r11d, %ebx
leal      (%rsi,%rsi,2), %esi
subl            %edx, %esi
addl            %ecx, %esi
movl            -16(%rax), %ecx
leal      (%rdx,%rcx,2), %r11d
movl            -12(%rax), %ecx
addl            %ebx, %r11d
leal      (%rcx,%rcx,2), %ecx
subl            %edx, %ecx
addl            %ecx, %esi
movl            -8(%rax), %ecx
leal      (%rdx,%rcx,2), %ecx
addl            %ecx, %r11d
movl            -4(%rax), %ecx
leal      (%rcx,%rcx,2), %ecx
subl            %edx, %ecx
addl            %esi, %ecx
cmpq            %rax, %rdi
jne             L3
cmpl            %ecx, %r11d
cmovl           %r11d, %ecx
movl       %ecx, (%r10,%r8,4)
addq            $1, %r8
cmpq            $40000, %r8
jne             L2
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`

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

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

    if(argc < 2){
        printf("Falta el tamaño del vector\n");
        exit(-1);
    }

    int n = atoi(argv[1]);
    int a = 5;
    int i;

    int y[n], x[n];

    for(i=0; i<n; i++){
        x[i] = 5;
        y[i] = 2;
    }

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

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for(i = 0; i < n; i++){

        y[i] += a * x[i];

    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

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

    printf("El componente 0 del vector es: %d \n",y[0]);
    printf("El componente %d del vector es: %d \n", n-1, y[n-1]);

    printf("Tiempo(seg.):%11.9f \n", ncgt);

}
```

	-O0	**-Os**	**-O2**	**-O3**
Tiempos ejec.	*0.002689000*	*0.000814000*	*0.000957000*	*0.000468000*

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

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -O0 daxpy.c -o daxpyO0
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./daxpyO0 1000000
El componente 0 del vector es: 27
El componente 999999 del vector es: 27
Tiempo(seg.):0.002689000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -Os daxpy.c -o daxpyOs
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./daxpyOs 1000000
El componente 0 del vector es: 27
El componente 999999 del vector es: 27
Tiempo(seg.):0.000814000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -O2 daxpy.c -o daxpyO2
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./daxpyO2 1000000
El componente 0 del vector es: 27
El componente 999999 del vector es: 27
Tiempo(seg.):0.000957000
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$gcc-7 -O3 daxpy.c -o daxpyO3
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MBPdeAlejandro:~/Dropbox/Carrera/2Cuatrimestre/Arquitectura de computadore
s/AC---Practicas/Practica 4] 2018-05-29 martes
$./daxpyO3 1000000
El componente 0 del vector es: 27
El componente 999999 del vector es: 27
Tiempo(seg.):0.000468000
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Con -O0 no se optimiza y se usa lo predeterminado. Con -Os optimiza el tamaño del código. Con -O2 se intenta aumentar el rendimiento del código sin aumentar el tamaño. Y con -O3 se activan optimizaciones que son caras en términos de tiempo de compilación y uso de memoria, de forma que puede que no mejore el rendimiento.

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)**

daxpyO0.s	daxpyOs.s	daxpyO2.s	daxpyO3.s

Call _clock_gettime
movl $0, -52(%rbp)
jmp L5
L6:
movq -80(%rbp), %rax
movl -52(%rbp), %edx
movslq %edx, %rdx
movl (%rax,%rdx,4), %ecx
movq -96(%rbp), %rax
movl -52(%rbp), %edx
movslq %edx, %rdx
movl (%rax,%rdx,4), %eax
imull -60(%rbp), %eax
addl %eax, %ecx
movq -80(%rbp), %rax
	Call _clock_gettime
	xorl %eax, %eax
	L5:
	cmpl %eax, %ebx
	jle L10
	imull $5,(%r14,%rax,4), %edx
	addl %edx, 0(%r13,%rax,4)
	incq %rax
	jmp L5
	L10:
	leaq -48(%rbp), %rsi
	xorl %edi, %edi
	call _clock_gettime
		Call _clock_gettime
		xorl %edx, %edx
		.align 4
		L6:
		movl (%r15,%rdx), %eax
		leal (%rax,%rax,4), %eax
		addl %eax, (%r12,%rdx)
		addq $4, %rdx
		cmpq %rbx, %rdx
		jne L6
		L7:
		leaq -64(%rbp), %rsi
		xorl %edi, %edi
		call _clock_gettime
			Call _clock_gettime
			movq %r12, %rdx
			xorl %esi, %esi
			shrq $2, %rdx
			negq %rdx
			andl $3, %edx
			leal 3(%rdx), %eax
			cmpl %r14d, %eax
			ja L12
			testl %edx, %edx
			je L13
			movq -88(%rbp), %rcx
			imull $5, 0(,%rcx,4), %eax
			addl %eax, (%r12)
			cmpl $1, %edx
			je L22
			imull $5, 4(,%rcx,4), %eax

```
movl     -52(%rbp), %edx
movslq         %edx, %rdx
movl  %ecx, (%rax,%rdx,4)
addl         $1, -52(%rbp)
L5:
movl     -52(%rbp), %eax
cmpl     -56(%rbp), %eax
jl             L6
leaq    -144(%rbp), %rax
movq           %rax, %rsi
movl           $0, %edi
call      _clock_gettime
```

```
addl           %eax, 4(%r12)
cmpl             $3, %edx
jne              L23
imull  $5, 8(,%rcx,4), %eax
movl             $3, %esi
addl           %eax, 8(%r12)
L13:
movl             %r13d, %r8d
xorl             %eax, %eax
subl             %edx, %r8d
salq             $2, %rdx
xorl             %ecx, %ecx
leaq      (%r12,%rdx), %r9
movl             %r8d, %edi
addq             %r15, %rdx
shrl             $2, %edi
.align 4
L10:
movdqu    (%rdx,%rax), %xmm1
addl             $1, %ecx
movdqa           %xmm1, %xmm0
pslld            $2, %xmm0
paddd            %xmm1, %xmm0
paddd       (%r9,%rax), %xmm0
movaps      %xmm0, (%r9,%rax)
addq             $16, %rax
cmpl             %edi, %ecx
jb               L10
movl             %r8d, %eax
andl             $-4, %eax
addl             %eax, %esi
cmpl             %eax, %r8d
je               L15
L12:
movslq           %esi, %rax
imull $5, (%r15,%rax,4), %edx
addl      %edx, (%r12,%rax,4)
leal            1(%rsi), %eax
cmpl             %r13d, %eax
jge              L15
cltq
imull $5, (%r15,%rax,4), %edx
addl      %edx, (%r12,%rax,4)
leal            2(%rsi), %eax
cmpl             %eax, %r13d
jle              L15
cltq
imull $5, (%r15,%rax,4), %edx
addl      %edx, (%r12,%rax,4)
leal            3(%rsi), %eax
cmpl             %eax, %r13d
jle              L15
cltq
imull $5, (%r15,%rax,4), %edx
addl      %edx, (%r12,%rax,4)
leal            4(%rsi), %eax
cmpl             %eax, %r13d
jle              L15
cltq
addl             $5, %esi
imull $5, (%r15,%rax,4), %edx
addl      %edx, (%r12,%rax,4)
cmpl             %esi, %r13d
jle              L15
movslq           %esi, %rsi
imull $5, (%r15,%rsi,4), %eax
addl      %eax, (%r12,%rsi,4)
L15:
xorl             %edi, %edi
leaq        -64(%rbp), %rsi
call         _clock_gettime
```