

2º curso / 2º  
cuatr.

Grado Ing.  
Inform.

## Arquitectura de Computadores (AC)

### Cuaderno de prácticas.

### Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos):

Grupo de prácticas y profesor de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

**Denominación de marca del chip de procesamiento o procesador (se encuentra en `/proc/cpuinfo`):** Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz

**Sistema operativo utilizado:** Manjaro Linux

**Versión de gcc utilizada:** gcc (GCC) 9.1.0

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

Arquitectura:	x86_64
modo(s) de operación de las CPUs:	32-bit, 64-bit
Orden de los bytes:	Little Endian
Tamaños de las direcciones:	39 bits physical, 48 bits virtual
CPU(s):	8
Lista de la(s) CPU(s) en línea:	0-7
Hilo(s) de procesamiento por núcleo:	2
Núcleo(s) por «socket»:	4
«Socket(s)»	1
Modo(s) NUMA:	1
ID de fabricante:	GenuineIntel
Familia de CPU:	6
Modelo:	60
Nombre del modelo:	Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz
Revisión:	3
CPU MHz:	4400.155
CPU MHz máx.:	4400,0000
CPU MHz mín.:	800,0000
BogoMIPS:	8003.28
Virtualización:	VT-x
Caché L1d:	32K
Caché L1i:	32K

Caché L2: 256K  
 Caché L3: 8192K  
 CPU(s) del nodo NUMA 0: 0-7

Indicadores: fpu vme de pse tsc msr pae mce cx8  
 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss  
 ht tm pbe syscall nx pdpe1gb rdtscp lm constant\_tsc arch\_perfmon pebs bts  
 rep\_good nopl xtopology nonstop\_tsc cpuid aperfmperf pni pclmulqdq dtes64  
 monitor ds\_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4\_1 sse4\_2  
 x2apic movbe popcnt tsc\_deadline\_timer aes xsave avx f16c rdrand lahf\_lm  
 abm cpuid\_fault invpcid\_single pti ssbd ibrs ibpb stibp tpr\_shadow vnmi  
 flexpriority ept vpid ept\_ad fsgsbase tsc\_adjust bmi1 avx2 smep bmi2 erms  
 invpcid xsaveopt dtherm ida arat pln pts md\_clear flush\_l1d

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

1.1 Modifique el código C para reducir el tiempo de ejecución (evalúe el tiempo y modifique sólo el trozo que hace la multiplicación y el trozo que se muestra en la Figura 1). Justifique los tiempos obtenidos (use -O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

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

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

**Figura 1 .** Código C++ que suma dos vectores

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

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

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

## **A) MULTIPLICACIÓN DE MATRICES:**

**CAPTURA CÓDIGO FUENTE:** pmm-secuencial.c

```
#include <chrono>
#include <cstdio>
```

```

using namespace std;

#define N 100
double A[N][N], B[N][N], Res[N][N];

int main(int argc, char **argv) {
    for (int i = 0; i < N; i++) {
        for (int j = i; j < N; j++) {
            A[i][j] = 2;
            B[i][j] = 3;
            Res[i][j] = 0;
        }
    }
    auto t0 = chrono::high_resolution_clock::now();
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                Res[i][j] += A[i][k] * B[k][j];

    auto tej = (chrono::high_resolution_clock::now() - t0).count();
    printf("Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n", tej,
Res[0][0], Res[N - 1][N - 1]);
}

```

### 1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

**Modificación a) –explicación–:** Desenrollado del bucle externo

**Modificación b) –explicación–:** Trasponer la matriz e invertir los accesos para mejorar la eficiencia del acceso a memoria

...

### 1.1. CÓDIGOS FUENTE MODIFICACIONES

#### a) Captura de pmm-secuencial-modificado\_a.c

```

#include <chrono>
#include <cstdio>

using namespace std;

#define N 100

double A[N][N], B[N][N], Res[N][N];

int main(int argc, char **argv) {
    for (int i = 0; i < N; i++) {

```

```

        for (int j = i; j < N; j++) {
            A[i][j] = 2;
            B[i][j] = 3;
            Res[i][j] = 0;
        }
    }
    auto t0 = chrono::high_resolution_clock::now();
    for (int i = 0; i < N; i += 4) {
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                Res[i][j] += A[i][k] * B[j][j];
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                Res[i + 1][j] += A[i + 1][k] * B[k][j];
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                Res[i + 2][j] += A[i + 2][k] * B[k][j];
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                Res[i + 3][j] += A[i + 3][k] * B[k][j];
    }
    auto tej = (chrono::high_resolution_clock::now() - t0).count();
    printf("Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n", tej,
    Res[0][0], Res[N - 1][N - 1]);
}

```

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

```

ac p4: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc ac p4]$ g++ pmm-secuencial-modificado_a.cpp -o pmm-secuencial-modificado_a -O2
[arturo@arturo-pc ac p4]$ ./pmm-secuencial-modificado_a
Tiempo: 1938088
Primer componente: 6.000000
Ultimo componente: 6.000000
[arturo@arturo-pc ac p4]$

```

#### b) Captura de pmm-secuencial-modificado\_b.c

```

#include <chrono>
#include <cstdio>

using namespace std;

```

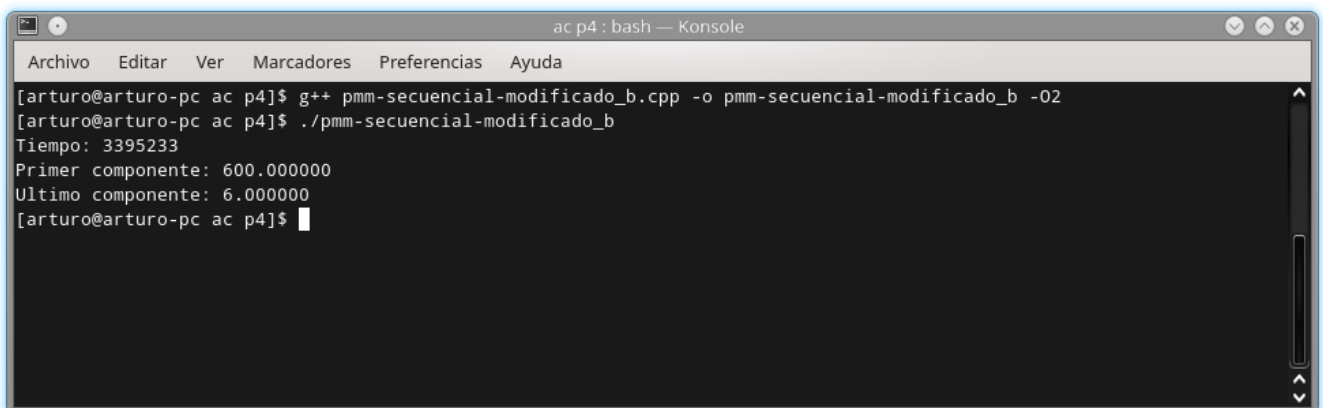
```

#define N 100

double A[N][N], B[N][N], Res[N][N];

int main(int argc, char **argv) {
    for (int i = 0; i < N; i++) {
        for (int j = i; j < N; j++) {
            A[i][j] = 2;
            B[j][i] = 3;
            Res[i][j] = 0;
        }
    }
    auto t0 = chrono::high_resolution_clock::now();
    for (int i = 0; i < N; i += 4) {
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                Res[i][j] += A[i][k] * B[j][k];
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                Res[i + 1][j] += A[i + 1][k] * B[j][k];
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                Res[i + 2][j] += A[i + 2][k] * B[j][k];
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                Res[i + 3][j] += A[i + 3][k] * B[j][k];
    }
    auto tej = (chrono::high_resolution_clock::now() - t0).count();
    printf("Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n", tej,
Res[0][0], Res[N - 1][N - 1]);
}

```



```

ac p4: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc ac p4]$ g++ pmm-secuencial-modificado_b.cpp -o pmm-secuencial-modificado_b -O2
[arturo@arturo-pc ac p4]$ ./pmm-secuencial-modificado_b
Tiempo: 3395233
Primer componente: 600.000000
Ultimo componente: 6.000000
[arturo@arturo-pc ac p4]$

```

**1.1. TIEMPOS:**

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		3408826ns
Modificación a)	Desenrollado de bucle	2997377ns
Modificación b)	Inversión de índices de la matriz	3000073ns
...		

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

El desenrollado de bucles obtiene la mayor mejora de los tiempos, ya que aunque el tamaño del código generado es mayor, el número de instrucciones ejecutadas se reduce. Invertir los ejes de acceso a la matriz también produce una mejora similar ya que los accesos a memoria se realizan con mayor localidad espacial.

**B) CÓDIGO FIGURA 1:**

CAPTURA CÓDIGO FUENTE: figura1-original.c

```

#include <chrono>
#include <cstdio>

using namespace std;

#define N 40000

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

int main() {
    double R[N];
    auto t0 = chrono::high_resolution_clock::now();
    for (int ii = 0; ii < N; ii++) {
        double X1 = 0, X2 = 0;
        for (int i = 0; i < 5000; i++)
            X1 += 2 * s[i].a + ii;
        for (int i = 0; i < 5000; i++)
            X2 += 3 * s[i].b - ii;
        R[ii] = X1 < X2 ? X1 : X2;
    }

    auto tej = (chrono::high_resolution_clock::now() - t0).count();
    printf("Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n", tej,
R[0], R[N - 1]);
}

```

### 1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

**Modificación a) –explicación–:** Unificación de los dos bucles internos

**Modificación b) –explicación–:** Desenrollado del bucle interno

...

### 1.1. CÓDIGOS FUENTE MODIFICACIONES

#### a) Captura figura1-modificado\_a.c

```
#include <chrono>
#include <cstdio>

using namespace std;

#define N 40000

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

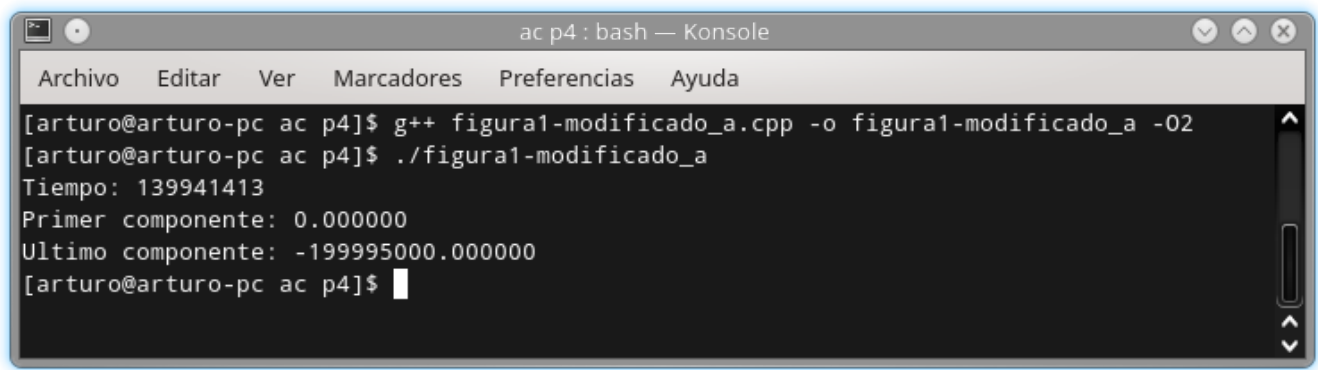
int main() {
    double R[N];
    auto t0 = chrono::high_resolution_clock::now();
    for (int ii = 0; ii < N; ii++) {
        double X1 = 0, X2 = 0;

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

        R[ii] = X1 < X2 ? X1 : X2;
    }

    auto tej = (chrono::high_resolution_clock::now() - t0).count();
    printf("Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n", tej,
R[0], R[N - 1]);
}
```

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



```
ac p4: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc ac p4]$ g++ figura1-modificado_a.cpp -o figura1-modificado_a -O2
[arturo@arturo-pc ac p4]$ ./figura1-modificado_a
Tiempo: 139941413
Primer componente: 0.000000
Ultimo componente: -199995000.000000
[arturo@arturo-pc ac p4]$
```

**b)Captura figura1-modificado\_b.c**

```
#include <chrono>
#include <cstdio>

using namespace std;

#define N 40000

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

int main() {
    double R[N];
    auto t0 = chrono::high_resolution_clock::now();

    for (int ii = 0; ii < N; ii++) {
        double X1 = 0, X2 = 0;

        for (int i = 0; i < 5000; i+=4) {
            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;
    }

    R[ii] = X1 < X2 ? X1 : X2;
}

auto tej = (chrono::high_resolution_clock::now() - t0).count();
printf("Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n",tej,
R[0], R[N - 1]);
}

```

```

ac p4: bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc ac p4]$ g++ figura1-modificado_b.cpp -o figura1-modificado_b -O2
[arturo@arturo-pc ac p4]$ ./figura1-modificado_b
Tiempo: 141265137
Primer componente: 0.000000
Ultimo componente: -199995000.000000
[arturo@arturo-pc ac p4]$

```

### 1.1. TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		271368891ns
Modificación a)	Unificación de los bucles internos	141669717ns
Modificación b)	Unificación y desenrollado de los bucles internos	141265137ns
...		

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

Ambas versiones tienen unos tiempos muy similares, por lo que lo más probable es que el desenrollado del bucle no haya tenido mucho efecto y la optimización con más impacto ha sido la de unificar los bucles,

**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.**

Código ensamblador de pmm-secuencial.c	Código ensamblador de pmm-secuencial-modificado_a.c	Código ensamblador de pmm-secuencial-modificado_b.c
<pre> .LC3: .string "Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n" main: pushq %rbx movsd .LC0(%rip), %xmm1 movl \$800, %esi xorl %ecx, %ecx movsd .LC1(%rip), %xmm0 xorl %edx, %edx .L3: movq %rcx, %rax .L2: movq \$0x000000000, Res(%rax) addq \$8, %rax movsd %xmm1, A-8(%rax) movsd %xmm0, B-8(%rax) cmpq %rsi, %rax jne .L2 addl \$1, %edx addq \$808, %rcx leaq 800(%rax), %rsi cmpl \$100, %edx jne .L3 call std::chrono::_V2::system_cloc k::now() movl \$B, %esi movl \$80000, %r8d movl \$A, %edi movq %rax, %rbx negq %rsi subq \$B, %r8 .L6: movl \$B+80000, %ecx .L5: movsd Res-80000(%rsi,%rcx), %xmm1 leaq -80000(%rcx), %rax movq %rdi, %rdx .L4: movsd (%rdx), %xmm0 mulsd (%rax), %xmm0 addq \$800, %rax addq \$8, %rdx addsd %xmm0, %xmm1 cmpq %rcx, %rax </pre>	<pre> .LC3: .string "Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n" main: pushq %r14 movsd .LC0(%rip), %xmm1 xorl %esi, %esi xorl %ecx, %ecx pushq %r13 movsd .LC1(%rip), %xmm0 movl \$800, %edx pushq %r12 pushq %rbp pushq %rbx .L3: movq %rsi, %rax .L2: movq \$0x000000000, Res(%rax) addq \$8, %rax movsd %xmm1, A-8(%rax) movsd %xmm0, B-8(%rax) cmpq %rdx, %rax jne .L2 addl \$1, %ecx addq \$808, %rsi leaq 800(%rax), %rdx cmpl \$100, %ecx jne .L3 call std::chrono::_V2::system_cloc k::now() movl \$2400, %ecx movl \$B, %r8d movl \$1600, %edi movq %rax, %rbx movl \$82400, %eax movl \$800, %esi negq %r8 subq \$B, %rax subq \$B, %rcx movl \$A+2400, %ebp movl \$A, %r11d subq \$B, %rdi movl \$A+1600, %r10d subq \$B, %rsi movq %rax, %r14 movl \$A+800, %r9d </pre>	<pre> .LC3: .string "Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n" main: pushq %rbx movsd .LC0(%rip), %xmm1 movl \$800, %edi xorl %edx, %edx movsd .LC1(%rip), %xmm0 xorl %ecx, %ecx .L3: leaq B(%rdx), %rsi movq %rdx, %rax .L2: addq \$8, %rax movsd %xmm0, (%rsi) addq \$800, %rsi movsd %xmm1, A-8(%rax) movq \$0x000000000, Res- 8(%rax) cmpq %rdi, %rax jne .L2 addl \$1, %ecx addq \$808, %rdx leaq 800(%rax), %rdi cmpl \$100, %ecx jne .L3 call std::chrono::_V2::system_cloc k::now() movl \$B, %esi movl \$80000, %r8d movl \$A, %edi movq %rax, %rbx negq %rsi subq \$B, %r8 .L6: movl \$B+80000, %ecx .L5: movsd Res-80000(%rsi,%rcx), %xmm1 leaq -80000(%rcx), %rax movq %rdi, %rdx .L4: movsd (%rdx), %xmm0 mulsd (%rax), %xmm0 addq \$800, %rax </pre>

<pre> jne .L4 movsd %xmm1, Res-80000(%rsi, %rax) leaq 8(%rax), %rcx cmpq \$B+80792, %rax jne .L5 addq \$800, %rsi addq \$800, %rdi cmpq %r8, %rsi jne .L6 call std::chrono::_V2::system_cloc k::now() movsd Res+79992(%rip), %xmm1 movl \$.LC3, %edi movsd Res(%rip), %xmm0 subq %rbx, %rax movq %rax, %rsi movl \$2, %eax call printf xorl %eax, %eax popq %rbx ret Res: .zero 80000 B: .zero 80000 A: .zero 80000 .LC0: .long 0 .long 1073741824 .LC1: .long 0 .long 1074266112 </pre>	<pre> .L12: movl \$B+80000, %edx movq %rdx, %r13 .L5: movsd Res-80000(%r8,%r13), %xmm1 leaq -80000(%r13), %rax movq %r11, %r12 .L4: movsd (%r12), %xmm0 mulsd (%rax), %xmm0 addq \$800, %rax addq \$8, %r12 addsd %xmm0, %xmm1 cmpq %r13, %rax jne .L4 movsd %xmm1, Res- 80000(%r8,%rax) leaq 8(%rax), %r13 cmpq \$B+80792, %rax jne .L5 movl \$B+80000, %r13d .L7: movsd Res-80000(%rsi,%r13), %xmm1 leaq -80000(%r13), %rax movq %r9, %r12 .L6: movsd (%r12), %xmm0 mulsd (%rax), %xmm0 addq \$800, %rax addq \$8, %r12 addsd %xmm0, %xmm1 cmpq %r13, %rax jne .L6 movsd %xmm1, Res-80000(%rsi, %rax) leaq 8(%rax), %r13 cmpq \$B+80792, %rax jne .L7 movl \$B+80000, %r13d .L9: movsd Res-80000(%rdi,%r13), %xmm1 leaq -80000(%r13), %rax movq %r10, %r12 .L8: movsd (%r12), %xmm0 mulsd (%rax), %xmm0 addq \$800, %rax addq \$8, %r12 addsd %xmm0, %xmm1 cmpq %rax, %r13 jne .L8 movsd %xmm1, Res-80000(%rdi, </pre>	<pre> addq \$8, %rdx addsd %xmm0, %xmm1 cmpq %rcx, %rax jne .L4 movsd %xmm1, Res-80000(%rsi, %rax) leaq 8(%rax), %rcx cmpq \$B+80792, %rax jne .L5 addq \$800, %rsi addq \$800, %rdi cmpq %r8, %rsi jne .L6 call std::chrono::_V2::system_cloc k::now() movsd Res+79992(%rip), %xmm1 movl \$.LC3, %edi movsd Res(%rip), %xmm0 subq %rbx, %rax movq %rax, %rsi movl \$2, %eax call printf xorl %eax, %eax popq %rbx ret Res: .zero 80000 B: .zero 80000 A: .zero 80000 .LC0: .long 0 .long 1073741824 .LC1: .long 0 .long 1074266112 </pre>
--	--	--

	<pre> %r13) addq \$8, %r13 cmpq \$B+80800, %r13 jne .L9 .L11: movsd Res-80000(%rcx,%rdx), %xmm1 leaq -80000(%rdx), %rax movq %rbp, %r12 .L10: movsd (%r12), %xmm0 mulsd (%rax), %xmm0 addq \$800, %rax addq \$8, %r12 addsd %xmm0, %xmm1 cmpq %rax, %rdx jne .L10 movsd %xmm1, Res-80000(%rcx, %rdx) addq \$8, %rdx cmpq \$B+80800, %rdx jne .L11 addq \$3200, %rcx addq \$3200, %r8 addq \$3200, %rbp addq \$3200, %r11 addq \$3200, %rdi addq \$3200, %r10 addq \$3200, %rsi addq \$3200, %r9 cmpq %r14, %rcx jne .L12 call std::chrono::_V2::system_cloc k::now() movsd Res+79992(%rip), %xmm1 movl \$.LC3, %edi movsd Res(%rip), %xmm0 subq %rbx, %rax movq %rax, %rsi movl \$2, %eax call printf popq %rbx xorl %eax, %eax popq %rbp popq %r12 popq %r13 popq %r14 ret Res: .zero 80000 B: .zero 80000 A: .zero 80000 </pre>	
--	---	--

	<pre> .LC0: .long 0 .long 1073741824 .LC1: .long 0 .long 1074266112 </pre>	
--	--	--

Entre pmm-secuencial.c y pmm-secuencial-modificado\_b no hay diferencias en el bucle, supongo que porque las optimizaciones del compilador. Sin embargo la diferencia con pmm-secuencial-modificado\_b es considerable, el código ensamblador ha aumentado bastante de tamaño. LC12 es la etiqueta de inicio del bucle desenrollado y todas las etiquetas detrás de esta hasta llegar a la llamada a `std::chrono::_V2::system_clock::now()` corresponden a los bucles internos resultantes de haber desenrollado el bucle exterior.

Código ensamblador de figura1.cpp	Código ensamblador de figura1-modificado_a.cpp	Código ensamblador de figura1-modificado_b.cpp
<pre> .LC1: .string "Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n" main: pushq %rbx subq \$320000, %rsp call std::chrono::_V2::system_cloc k::now() xorl %ecx, %ecx pxor %xmm3, %xmm3 movq %rax, %rbx .L5: pxor %xmm1, %xmm1 movl %ecx, %edx movapd %xmm3, %xmm0 movl \$5000, %eax cvtsi2sd %ecx, %xmm1 .L2: addsd %xmm1, %xmm0 subl \$1, %eax jne .L2 movl %edx, %eax pxor %xmm2, %xmm2 movapd %xmm3, %xmm1 negl %eax cvtsi2sd %eax, %xmm2 movl \$5000, %eax .L3: addsd %xmm2, %xmm1 subl \$1, %eax jne .L3 minsd %xmm1, %xmm0 movsd %xmm0, (%rsp,%rcx,8) addq \$1, %rcx cmpq \$40000, %rcx jne .L5 call std::chrono::_V2::system_cloc k::now() </pre>	<pre> .LC1: .string "Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n" main: pushq %rbx subq \$320000, %rsp call std::chrono::_V2::system_cloc k::now() xorl %edx, %edx pxor %xmm4, %xmm4 movq %rax, %rbx .L4: pxor %xmm3, %xmm3 pxor %xmm2, %xmm2 movapd %xmm4, %xmm1 movl %edx, %eax cvtsi2sd %edx, %xmm3 negl %eax movapd %xmm4, %xmm0 cvtsi2sd %eax, %xmm2 movl \$5000, %eax .L2: addsd %xmm3, %xmm0 addsd %xmm2, %xmm1 subl \$1, %eax jne .L2 minsd %xmm1, %xmm0 movsd %xmm0, (%rsp,%rdx,8) addq \$1, %rdx cmpq \$40000, %rdx jne .L4 call std::chrono::_V2::system_cloc k::now() movsd (%rsp), %xmm0 movl \$.LC1, %edi movsd 319992(%rsp), %xmm1 subq %rbx, %rax movq %rax, %rsi </pre>	<pre> .LC1: .string "Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n" main: pushq %rbx subq \$320000, %rsp call std::chrono::_V2::system_cloc k::now() xorl %edx, %edx pxor %xmm4, %xmm4 movq %rax, %rbx .L4: pxor %xmm3, %xmm3 pxor %xmm2, %xmm2 movapd %xmm4, %xmm1 movl %edx, %eax cvtsi2sd %edx, %xmm3 negl %eax movapd %xmm4, %xmm0 cvtsi2sd %eax, %xmm2 movl \$1250, %eax .L2: addsd %xmm3, %xmm0 addsd %xmm2, %xmm1 addsd %xmm3, %xmm0 addsd %xmm2, %xmm1 addsd %xmm3, %xmm0 addsd %xmm2, %xmm1 addsd %xmm3, %xmm0 addsd %xmm2, %xmm1 addsd %xmm3, %xmm0 addsd %xmm2, %xmm1 subl \$1, %eax jne .L2 minsd %xmm1, %xmm0 movsd %xmm0, (%rsp,%rdx,8) addq \$1, %rdx cmpq \$40000, %rdx jne .L4 call std::chrono::_V2::system_cloc </pre>

<pre> movsd (%rsp), %xmm0 movl \$.LC1, %edi movsd 319992(%rsp), %xmm1 subq %rbx, %rax movq %rax, %rsi movl \$2, %eax call printf addq \$320000, %rsp xorl %eax, %eax popq %rbx ret </pre>	<pre> movl \$2, %eax call printf addq \$320000, %rsp xorl %eax, %eax popq %rbx ret </pre>	<pre> k::now() movsd (%rsp), %xmm0 movl \$.LC1, %edi movsd 319992(%rsp), %xmm1 subq %rbx, %rax movq %rax, %rsi movl \$2, %eax call printf addq \$320000, %rsp xorl %eax, %eax popq %rbx ret </pre>
---	---	--

Entre figura1.cpp y figura1-modificado\_a.cpp vemos que en el segundo se ha eliminado una de las etiquetas usadas para los bucles, así que en efecto la unificación de los bucles se ve reflejada en el código ensamblador. En figura1-modificado\_b.cpp vemos que además de haber una etiqueta menos que en figura1.cpp, las instrucciones de suma han aumentado bastante gracias al desenrollado de bucle

### 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

figura1:

.LC1:

.string "Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n"

main:

```

pushq %rbx
subq $320000, %rsp
call std::chrono::_V2::system_clock::now()
xorl %edx, %edx
pxor %xmm4, %xmm4
movq %rax, %rbx

```

.L4:

```

pxor %xmm3, %xmm3
pxor %xmm2, %xmm2
movapd %xmm4, %xmm1
movl %edx, %eax
cvtsi2sdl %edx, %xmm3
negl %eax
movapd %xmm4, %xmm0
cvtsi2sdl %eax, %xmm2
movl $2500, %eax

```

.L2:

```

addsd %xmm3, %xmm0
addsd %xmm2, %xmm1
addsd %xmm3, %xmm0
addsd %xmm2, %xmm1
subl $1, %eax
jne .L2
minsd %xmm1, %xmm0
movsd %xmm0, (%rsp,%rdx,8)
addq $1, %rdx
cmpq $40000, %rdx
jne .L4
call std::chrono::_V2::system_clock::now()
movsd (%rsp), %xmm0

```

```

movl  $.LC1, %edi
movsd 319992(%rsp), %xmm1
subq  %rbx, %rax
movq  %rax, %rsi
movl  $2, %eax
call  printf
addq  $320000, %rsp
xorl  %eax, %eax
popq  %rbx
ret

```

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

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

2.1. Genere los programas en ensamblador para cada una de las siguientes opciones de optimización del compilador: -O0, -Os, -O2, -O3. Explique las diferencias que se observan en el código justificando al mismo tiempo las mejoras en velocidad que acarrearán. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos. Sólo se debe evaluar el tiempo del núcleo DAXPY

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

#### CAPTURA CÓDIGO FUENTE: daxpy.c

```

#include <chrono>
#include <cstdio>

using namespace std;

#define N 10000
#define a 3.3

double y[N + 1], x[N + 1];
int main(int argc, char **argv) {

    for (int i = 0; i <= N; i++) {
        y[i] = 2.1 * i;
        x[i] = 3.2 * i;
    }
}

```

```

auto t0 = chrono::high_resolution_clock::now();

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

auto tej = (chrono::high_resolution_clock::now() - t0).count();
printf("Tiempo: %ld\nPrimer componente: %f\nUltimo componente: %f\n", tej,
y[0], y[N - 1]);

return 0;
}

```

Tiempos ejec.	-O0	-Os	-O2	-O3
	79373ns	17765ns	27292ns	8742ns

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

```

ac p4 : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[arturo@arturo-pc ac p4]$ g++ daxpy.cpp -o daxpy -O2
[arturo@arturo-pc ac p4]$ ./daxpy
Tiempo: 27292
Primer componente: 0.000000
Ultimo componente: 126587.340000
[arturo@arturo-pc ac p4]$

```

**COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:**

Al pasar de O0 a Os vemos que el código reduce considerablemente su tamaño además de que deja de usar instrucciones `cltq`. De Os a O2 no hay diferencia en esta sección concreta del código generado. Al usar O3 se generan un mayor número de instrucciones que en O2 y Os, además se usan instrucciones de manejo de números de doble precisión empaquetados.

**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
<pre> call    std::chrono::_V2::system_clock::now() movq    %rax, -40(%rbp) movl    \$1, -8(%rbp) .L8: </pre>	<pre> call    std::chrono::_V2::system_clock::now() movsd   .LC2(%rip),%xmm1 movq    %rax, %rbx movl    \$8, %eax </pre>	<pre> call    std::chrono::_V2::system_clock::now() movsd   .LC2(%rip), %xmm1 movq    %rax, %rbx movl    \$8, %eax </pre>	<pre> call    std::chrono::_V2::system_clock::now() movapd  .LC6(%rip),%xmm1 movq    %rax, %rbx </pre>



<pre> cmpl \$10000, -8(%rbp) jg .L7 movl -8(%rbp), %eax cltq movsd x(,%rax,8), %xmm1 movsd .LC2(%rip), %xmm0 mulsd %xmm0, %xmm1 movl -8(%rbp), %eax cltq movsd y(,%rax,8), %xmm0 addsd %xmm1, %xmm0 movl -8(%rbp), %eax cltq movsd %xmm0, y(,%rax,8) addl \$1, -8(%rbp) jmp .L8 .L7: call std::chrono::_V2::system_clock::now() </pre>	<pre> .L3: movsd x(%rax), %xmm0 addq \$8, %rax mulsd %xmm1, %xmm0 addsd y-8(%rax), %xmm0 movsd %xmm0, y-8(%rax) cmpq \$80008, %rax jne .L3 call std::chrono::_V2::system_clock::now() </pre>	<pre> .L3: movsd x(%rax), %xmm0 addq \$8, %rax mulsd %xmm1, %xmm0 addsd y-8(%rax), %xmm0 movsd %xmm0, y-8(%rax) jne .L3 call std::chrono::_V2::system_clock::now() </pre>	<pre> movl \$8, %eax .L3: movupd x(%rax), %xmm0 movupd y(%rax), %xmm7 addq \$16, %rax mulpd %xmm1, %xmm0 addpd %xmm7, %xmm0 movups %xmm0, y-16(%rax) cmpq \$80008, %rax jne .L3 call std::chrono::_V2::system_clock::now() </pre>
--	--	---	---

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

Rmax=4.577657

Nmax=12600

Rmax/2=2.2888

N1/2=914500

Rpico=5.53

