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): David Carrasco Chicharro

Grupo de prácticas: B1

Fecha de entrega: 26/05/2018

Fecha evaluación en clase: 28/05/2018

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel® Core™ i5-5200U CPU @ 2.20GHz*

Sistema operativo utilizado: *Linux 4.13.0-41-generic #46 ~ 16.04.1-Ubuntu*

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

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

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-21 lunes
$ 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:2
Núcleo(s) por «socket»:2
Socket(s):             1
Modo(s) NUMA:          1
ID de fabricante:      GenuineIntel
Familia de CPU:        6
Modelo:                61
Model name:            Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
Revisión:              4
CPU MHz:               1017.964
CPU max MHz:           2700,0000
CPU min MHz:           500,0000
BogoMIPS:              4389.79
Virtualización:        VT-x
Caché L1d:             32K
Caché L1i:             32K
Caché L2:              256K
Caché L3:              3072K
NUMA node0 CPU(s):     0-3
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr p
ge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe sys
call nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nop
l 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 m
ovbe popcnt aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fau
lt epb invpcid_single pti retpoline intel_pt tpr_shadow vnmi flexpriority
 ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid rdseed adx
 smap xsaveopt dtherm ida arat pln pts
```

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`

```
15 int main(int argc, char **argv) {
16
17   struct timespec cgt1,cgt2;
18   double ncgt; //para tiempo de ejecución
19
20    if(argc < 2){
21       fprintf(stderr,"Falta el tamaño\n");
22       exit(-1);
23    }
24
25    unsigned int N = atoi(argv[1]);
26    int fil, col;
27
28    long long unsigned int **M1, **M2, **M3;
29    M1 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
30    M2 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
31    M3 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
32    for(long long unsigned int i=0 ; i<N ; i++){
33       M1[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
34       M2[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
35       M3[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
36    }
37
38    if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
39       printf("Error en la reserva de espacio para las matrices\n");
40       exit(-2);
41    }
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

```
15 int main(int argc, char **argv) {
16
17   struct timespec cgt1,cgt2;
18   double ncgt; //para tiempo de ejecución
19
20    if(argc < 2){
21       fprintf(stderr,"Falta el tamaño\n");
22       exit(-1);
23    }
24
25    unsigned int N = atoi(argv[1]);
26    int fil, col;
27
28    long long unsigned int **M1, **M2, **M3;
29    M1 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
30    M2 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
31    M3 = (long long unsigned int**) malloc(N*sizeof(long long unsigned int*));
32    for(long long unsigned int i=0 ; i<N ; i++){
33       M1[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
34       M2[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
35       M3[i] = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
36    }
37
38    if ( (M1==NULL) || (M2==NULL) || (M3==NULL) ){
39       printf("Error en la reserva de espacio para las matrices\n");
40       exit(-2);
41    }
42
43    // Inicialización de las matrices
44    for(fil=0 ; fil<N ; fil++)
45       for(col=0 ; col<N ; col++){
46          M1[fil][col]=fil+col;
47          M2[fil][col]=fil+col;
48          M3[fil][col]=0;
49       }
50
51    clock_gettime(CLOCK_REALTIME,&cgt1);
52    // Cálculo de M3
53    for (int i=0 ; i<N; i++)
54       for(int j=0 ; j<N ; j++)
55          for(int k=0 ; k<N ; k++)
56             M3[i][j] += M1[i][k] * M2[k][j];
57
58
59    clock_gettime(CLOCK_REALTIME,&cgt2);
60    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
61      (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
62
63    printf("Tiempo(seg.):%11.9f\t / Tamaño Matrices:%u\n",ncgt,N);
64    printf("M3[0][0]=%llu , M3[%d][%d]=%llu \n", \
65       M3[0][0], N-1, N-1, M3[N-1][N-1]);
66
67    for(int i=0 ; i<N ; i++){
68       free(M1[i]);
69       free(M2[i]);
70       free(M3[i]);
71    }
72    free(M1);   free(M2);   free(M3);
73
74    return 0;
75 }
```

Modificación a) –explicación-:

Desenrollado de bucles para romper secuencias de instrucciones dependientes intercalando otras instrucciones. En el último bucle de los tres utilizados para calcular el resultado de la matriz se ejecutan secuencialmente cuatro cálculos de las multiplicaciones realizadas para hallar el resultado de la matriz, de modo que se reduce el número de saltos.

Modificación b) –explicación-:

Intercambio de bucles para cambiar la forma de acceder a los datos según los almacena el compilador para aprovechar la localidad. De este modo se optimizan los accesos a memoria por localidad espacial y temporal.

1.1. CÓDIGOS FUENTE MODIFICACIONES

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

```
53    // Cálculo de M3
54    for (int i=0 ; i<N; i++)
55       for(int j=0 ; j<N ; j+=4)
56          for(int k=0 ; k<N ; k++){
57             M3[i][j] += M1[i][k] * M2[k][j];
58             M3[i][j+1] += M1[i][k] * M2[k][j+1];
59             M3[i][j+2] += M1[i][k] * M2[k][j+2];
60             M3[i][j+3] += M1[i][k] * M2[k][j+3];
61          }
```

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

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:2018
-05-21 lunes
$ gcc -O2 pmm-secuencial-modificado_a.c -o pmm-secuencial-modificado_a -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:2018
-05-21 lunes
$ ./pmm-secuencial-modificado_a 1500
Tiempo(seg.):5.734103343          / Tamaño Matrices:1500
M3[0][0]=1123875250 , M3[1499][1499]=7864878250
```

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

```
53    // Cálculo de M3
54    for (int i=0 ; i<N; i++)
55       for(int k=0 ; k<N ; k++)
56          for(int j=0 ; j<N ; j++)
57             M3[i][j] += M1[i][k] * M2[k][j];
58
```

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

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:2018
-05-21 lunes
$ gcc -O2 pmm-secuencial-modificado_b.c -o pmm-secuencial-modificado_b -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:2018
-05-21 lunes
$ ./pmm-secuencial-modificado_b 1500
Tiempo(seg.):3.825377118          / Tamaño Matrices:1500
M3[0][0]=1123875250 , M3[1499][1499]=7864878250
```

1.1. TIEMPOS:

Modificación	-O2
Sin modificar	*10,82 seg*
Modificación a)	*5,73 seg*
Modificación b)	*3,82 seg*

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

En ambas modificaciones se han obtenido mejores resultados ya que en la primera se ha reducido en cuatro partes el número de instrucciones de salto, y en la segunda se ha conseguido optimizar los accesos a memoria para aprovechar la localidad.

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)

En color rojo se destacan las diferencias entre `pmm-secuencial.s` y `pmm-secuencial-modificado_b.s`, y con los colores azul, verde y naranja las secuencias de código que se repiten en `pmm-secuencial-modificado_a.s` con el desenrollado de código.

pmm-secuencial.s	**pmm-secuencial-modificado_a.s**	**pmm-secuencial-modificado_b.s**

.L16:
movl $0, -120(%rbp)
jmp .L12
.L15:
movl $0, -116(%rbp)
jmp .L13
.L14:
movl -124(%rbp), %eax
cltq
leaq 0(,%rax,8), %rdx
movq -80(%rbp), %rax
addq %rdx, %rax
movq (%rax), %rax
movl -120(%rbp), %edx
movslq %edx, %rdx
salq $3, %rdx
addq %rdx, %rax
movl -124(%rbp), %edx
movslq %edx, %rdx
leaq 0(,%rdx,8), %rcx
movq -80(%rbp), %rdx
addq %rcx, %rdx
movq (%rdx), %rdx
movl -120(%rbp), %ecx
movslq %ecx, %rcx
salq $3, %rcx
addq %rcx, %rdx
movq (%rdx), %rcx
movl -124(%rbp), %edx
movslq %edx, %rdx
leaq 0(,%rdx,8), %rsi
movq -96(%rbp), %rdx
addq %rsi, %rdx
movq (%rdx), %rdx
movl -116(%rbp), %esi
movslq %esi, %rsi
salq $3, %rsi
addq %rsi, %rdx
movq (%rdx), %rsi
movl -116(%rbp), %edx
movslq %edx, %rdx
leaq 0(,%rdx,8), %rdi
movq -88(%rbp), %rdx
addq %rdi, %rdx
	.L16:
	movl $0, -120(%rbp)
	jmp .L12
	.L15:
	movl $0, -116(%rbp)
	jmp .L13
	.L14:
	movl -124(%rbp), %eax
	cltq
	leaq 0(,%rax,8), %rdx
	movq -80(%rbp), %rax
	addq %rdx, %rax
	movq (%rax), %rax
	movl -120(%rbp), %edx
	movslq %edx, %rdx
	salq $3, %rdx
	addq %rdx, %rax
	movl -124(%rbp), %edx
	movslq %edx, %rdx
	leaq 0(,%rdx,8), %rcx
	movq -80(%rbp), %rdx
	addq %rcx, %rdx
	movq (%rdx), %rdx
	movl -120(%rbp), %ecx
	movslq %ecx, %rcx
	salq $3, %rcx
	addq %rcx, %rdx
	movq (%rdx), %rcx
	movl -124(%rbp), %edx
	movslq %edx, %rdx
	leaq 0(,%rdx,8), %rsi
	movq -96(%rbp), %rdx
	addq %rsi, %rdx
	movq (%rdx), %rdx
	movl -116(%rbp), %esi
	movslq %esi, %rsi
	salq $3, %rsi
	addq %rsi, %rdx
	movq (%rdx), %rsi
	movl -116(%rbp), %edx
	movslq %edx, %rdx
	leaq 0(,%rdx,8), %rdi
	movq -88(%rbp), %rdx
	addq %rdi, %rdx
		.L16:
		movl $0, -120(%rbp)
		jmp .L12
		.L15:
		movl $0, -116(%rbp)
		jmp .L13
		.L14:
		movl -124(%rbp), %eax
		cltq
		leaq 0(,%rax,8), %rdx
		movq -80(%rbp), %rax
		addq %rdx, %rax
		movq (%rax), %rax
		movl -116(%rbp), %edx
		movslq %edx, %rdx
		salq $3, %rdx
		addq %rdx, %rax
		movl -124(%rbp), %edx
		movslq %edx, %rdx
		leaq 0(,%rdx,8), %rcx
		movq -80(%rbp), %rdx
		addq %rcx, %rdx
		movq (%rdx), %rdx
		movl -116(%rbp), %ecx
		movslq %ecx, %rcx
		salq $3, %rcx
		addq %rcx, %rdx
		movq (%rdx), %rcx
		movl -124(%rbp), %edx
		movslq %edx, %rdx
		leaq 0(,%rdx,8), %rsi
		movq -96(%rbp), %rdx
		addq %rsi, %rdx
		movq (%rdx), %rdx
		movl -120(%rbp), %esi
		movslq %esi, %rsi
		salq $3, %rsi
		addq %rsi, %rdx
		movq (%rdx), %rsi
		movl -120(%rbp), %edx
		movslq %edx, %rdx
		leaq 0(,%rdx,8), %rdi
		movq -88(%rbp), %rdx
		addq %rdi, %rdx

```
  movq  (%rdx), %rdx
  movl  -120(%rbp), %edi
  movslq  %edi, %rdi
  salq  $3, %rdi
  addq  %rdi, %rdx
  movq  (%rdx), %rdx
  imulq  %rsi, %rdx
  addq  %rcx, %rdx
  movq  %rdx, (%rax)
  addl  $1, -116(%rbp)
.L13:
  movl  -116(%rbp), %eax
  cmpl  -108(%rbp), %eax
  jb  .L14
  addl  $1, -120(%rbp)
.L12:
  movl  -120(%rbp), %eax
  cmpl  -108(%rbp), %eax
  jb  .L15
  addl  $1, -124(%rbp)
```

```
  movq  (%rdx), %rdx
  movl  -120(%rbp), %edi
  movslq  %edi, %rdi
  salq  $3, %rdi
  addq  %rdi, %rdx
  movq  (%rdx), %rdx
  imulq  %rsi, %rdx
  addq  %rcx, %rdx
  movq  %rdx, (%rax)
  movl  -124(%rbp), %eax
  cltq
  leaq  0(,%rax,8), %rdx
  movq  -80(%rbp), %rax
  addq  %rdx, %rax
  movq  (%rax), %rax
  movl  -120(%rbp), %edx
  movslq  %edx, %rdx
  addq  $1, %rdx
  salq  $3, %rdx
  addq  %rdx, %rax
  movl  -124(%rbp), %edx
  movslq  %edx, %rdx
  leaq  0(,%rdx,8), %rcx
  movq  -80(%rbp), %rdx
  addq  %rcx, %rdx
  movq  (%rdx), %rdx
  movl  -120(%rbp), %ecx
  movslq  %ecx, %rcx
  addq  $1, %rcx
  salq  $3, %rcx
  addq  %rcx, %rdx
  movq  (%rdx), %rcx
  movl  -124(%rbp), %edx
  movslq  %edx, %rdx
  leaq  0(,%rdx,8), %rsi
  movq  -96(%rbp), %rdx
  addq  %rsi, %rdx
  movq  (%rdx), %rdx
  movl  -116(%rbp), %esi
  movslq  %esi, %rsi
  salq  $3, %rsi
  addq  %rsi, %rdx
  movq  (%rdx), %rsi
  movl  -116(%rbp), %edx
  movslq  %edx, %rdx
  leaq  0(,%rdx,8), %rdi
  movq  -88(%rbp), %rdx
  addq  %rdi, %rdx
  movq  (%rdx), %rdx
  movl  -120(%rbp), %edi
  movslq  %edi, %rdi
  addq  $1, %rdi
  salq  $3, %rdi
  addq  %rdi, %rdx
  movq  (%rdx), %rdx
  imulq  %rsi, %rdx
  addq  %rcx, %rdx
  movq  %rdx, (%rax)
  movl  -124(%rbp), %eax
  cltq
  leaq  0(,%rax,8), %rdx
  movq  -80(%rbp), %rax
```

```
  movq  (%rdx), %rdx
  movl  -116(%rbp), %edi
  movslq  %edi, %rdi
  salq  $3, %rdi
  addq  %rdi, %rdx
  movq  (%rdx), %rdx
  imulq  %rsi, %rdx
  addq  %rcx, %rdx
  movq  %rdx, (%rax)
  addl  $1, -116(%rbp)
.L13:
  movl  -116(%rbp), %eax
  cmpl  -108(%rbp), %eax
  jb  .L14
  addl  $1, -120(%rbp)
.L12:
  movl  -120(%rbp), %eax
  cmpl  -108(%rbp), %eax
  jb  .L15
  addl  $1, -124(%rbp)
```

```
addq  %rdx, %rax
movq  (%rax), %rax
movl  -120(%rbp), %edx
movslq  %edx, %rdx
addq  $2, %rdx
salq  $3, %rdx
addq  %rdx, %rax
movl  -124(%rbp), %edx
movslq  %edx, %rdx
leaq  0(,%rdx,8), %rcx
movq  -80(%rbp), %rdx
addq  %rcx, %rdx
movq  (%rdx), %rdx
movl  -120(%rbp), %ecx
movslq  %ecx, %rcx
addq  $2, %rcx
salq  $3, %rcx
addq  %rcx, %rdx
movq  (%rdx), %rcx
movl  -124(%rbp), %edx
movslq  %edx, %rdx
leaq  0(,%rdx,8), %rsi
movq  -96(%rbp), %rdx
addq  %rsi, %rdx
movq  (%rdx), %rdx
movl  -116(%rbp), %esi
movslq  %esi, %rsi
salq  $3, %rsi
addq  %rsi, %rdx
movq  (%rdx), %rsi
movl  -116(%rbp), %edx
movslq  %edx, %rdx
leaq  0(,%rdx,8), %rdi
movq  -88(%rbp), %rdx
addq  %rdi, %rdx
movq  (%rdx), %rdx
movl  -120(%rbp), %edi
movslq  %edi, %rdi
addq  $2, %rdi
salq  $3, %rdi
addq  %rdi, %rdx
movq  (%rdx), %rdx
imulq  %rsi, %rdx
addq  %rcx, %rdx
movq  %rdx, (%rax)
movl  -124(%rbp), %eax
cltq
leaq  0(,%rax,8), %rdx
movq  -80(%rbp), %rax
addq  %rdx, %rax
movq  (%rax), %rax
movl  -120(%rbp), %edx
movslq  %edx, %rdx
addq  $3, %rdx
salq  $3, %rdx
addq  %rdx, %rax
movl  -124(%rbp), %edx
movslq  %edx, %rdx
leaq  0(,%rdx,8), %rcx
movq  -80(%rbp), %rdx
addq  %rcx, %rdx
movq  (%rdx), %rdx
```

```
  movl  -120(%rbp), %ecx
  movslq  %ecx, %rcx
  addq  $3, %rcx
  salq  $3, %rcx
  addq  %rcx, %rdx
  movq  (%rdx), %rcx
  movl  -124(%rbp), %edx
  movslq  %edx, %rdx
  leaq  0(,%rdx,8), %rsi
  movq  -96(%rbp), %rdx
  addq  %rsi, %rdx
  movq  (%rdx), %rdx
  movl  -116(%rbp), %esi
  movslq  %esi, %rsi
  salq  $3, %rsi
  addq  %rsi, %rdx
  movq  (%rdx), %rsi
  movl  -116(%rbp), %edx
  movslq  %edx, %rdx
  leaq  0(,%rdx,8), %rdi
  movq  -88(%rbp), %rdx
  addq  %rdi, %rdx
  movq  (%rdx), %rdx
  movl  -120(%rbp), %edi
  movslq  %edi, %rdi
  addq  $3, %rdi
  salq  $3, %rdi
  addq  %rdi, %rdx
  movq  (%rdx), %rdx
  imulq  %rsi, %rdx
  addq  %rcx, %rdx
  movq  %rdx, (%rax)
  addl  $1, -116(%rbp)
.L13:
  movl  -116(%rbp), %eax
  cmpl  -108(%rbp), %eax
  jb  .L14
  addl  $4, -120(%rbp)
.L12:
  movl  -120(%rbp), %eax
  cmpl  -108(%rbp), %eax
  jb  .L15
  addl  $1, -124(%rbp)
```

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

```
13 struct {
14    int a;
15    int b;
16 } s[5000];
17
18 int main(){
19    struct timespec cgt1,cgt2;
20    double ncgt; //para tiempo de ejecución
21    int ii, i, X1, X2;
22    long long unsigned int R[40000];
23
24    // Inicialización
25    for(int k=0 ; k<5000 ; k++){
26       s[k].a = k;
27       s[k].b = k;
28    }
29
30    // Cómputos
31    clock_gettime(CLOCK_REALTIME,&cgt1);
32
33    for(ii=0 ; ii<40000 ; ii++){
34       X1=0; X2=0;
35       for(i=0 ; i<5000 ; i++)       X1 += 2*s[i].a + ii;
36       for(i=0 ; i<5000 ; i++)       X2 += 3*s[i].b - ii;
37
38       if(X1<X2)          R[ii]=X1;
39       else         R[ii]=X2;
40    }
41
42    clock_gettime(CLOCK_REALTIME,&cgt2);
43    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
44      (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
45
46    printf("Tiempo(seg.):%11.9f \n",ncgt);
47    printf("R[0]=%llu\t R[39999]=%llu\n",R[0],R[39999]);
48
49    return 0;
50 }
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Se han fusionados los dos bucles de los cómputos en uno sólo para disminuir a la mitad el número de instrucciones de salto.

Modificación b) –explicación-: Se han desenrollado los bucles para romper secuencias de instrucciones dependientes. Se ejecutan secuencialmente cuatro operaciones para hallar el resultado de cada una de las componentes del vector, de modo que se reduce el número de saltos.

Modificación c) –explicación-: Esta modificación consiste en una combinación de las dos anteriores.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1`-modificado_a.c`

```
33    for(ii=0 ; ii<40000 ; ii++){
34       X1=0; X2=0;
35       for(i=0 ; i<5000 ; i++){
36          X1 += 2*s[i].a + ii;
37          X2 += 3*s[i].b - ii;
38       }
39
40       if(X1<X2)          R[ii]=X1;
41       else         R[ii]=X2;
42    }
```

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

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:2018
-05-21 lunes
$ gcc -O2 figura1-modificado_a.c -o figura1-modificado_a -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:2018
-05-21 lunes
$ ./figura1-modificado_a
Tiempo(seg.):0.232453261
R[0]=24995000    R[39999]=18446744073547049116
```

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

```
33    for(ii=0 ; ii<40000 ; ii++){
34       X1=0; X2=0;
35       for(i=0 ; i<5000 ; i+=4){
36          X1 += 2*s[i].a + ii;
37          X2 += 2*s[i+1].a + ii+1;
38          X3 += 2*s[i+2].a + ii+2;
39          X4 += 2*s[i+3].a + ii+3;
40       }
41       for(i=0 ; i<5000 ; i+=4){
42          X5 += 3*s[i+1].b - ii;
43          X6 += 3*s[i+1].b - ii+1;
44          X7 += 3*s[i+1].b - ii+2;
45          X8 += 3*s[i+1].b - ii+3;
46       }
47
48       if(X1<X5)          R[ii]=X1;
49       else         R[ii]=X5;
50
51       if(X2<X6)          R[ii]=X2;
52       else         R[ii]=X6;
53
54       if(X3<X7)          R[ii]=X3;
55       else         R[ii]=X7;
56
57       if(X4<X8)          R[ii]=X4;
58       else         R[ii]=X8;
59
60    }
```

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

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-21 lunes
$ gcc -O2 figura1-modificado_b.c -o figura1-modificado_b -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-21 lunes
$ ./figura1-modificado_b
Tiempo(seg.):0.171606658
R[0]=9375000     R[39999]=393713424
```

c) Captura figura1-**`modificado_c.c`**

```
33    for(ii=0 ; ii<40000 ; ii++){
34       X1=0; X2=0;
35       for(i=0 ; i<5000 ; i+=4){
36          X1 += 2*s[i].a + ii;
37          X2 += 2*s[i+1].a + ii+1;
38          X3 += 2*s[i+2].a + ii+2;
39          X4 += 2*s[i+3].a + ii+3;
40          X5 += 3*s[i+1].b - ii;
41          X6 += 3*s[i+1].b - ii+1;
42          X7 += 3*s[i+1].b - ii+2;
43          X8 += 3*s[i+1].b - ii+3;
44       }
45
46       if(X1<X5)          R[ii]=X1;
47       else         R[ii]=X5;
48
49       if(X2<X6)          R[ii]=X2;
50       else         R[ii]=X6;
51
52       if(X3<X7)          R[ii]=X3;
53       else         R[ii]=X7;
54
55       if(X4<X8)          R[ii]=X4;
56       else         R[ii]=X8;
57    }
```

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

```
$ gcc -O2 figura1-modificado_c.c -o figura1-modificado_c -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-21 lunes
$ ./figura1-modificado_c
Tiempo(seg.):0.145225053
R[0]=9375000     R[39999]=393713392
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0,3396*
Modificación a)	*0,2324*
Modificación b)	*0,1716*

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

La primera modificación no significa una excesiva mejora, pues la fusión de dos bucles tan sólo supone reducir algunos saltos, pero en la segunda la mejora de resultados es más notoria, pues se reducen bastantes instrucciones de salto. Por último, la tercera mejora únicamente supone un avance de la segunda, ya que ésta no acarrea un cambio excesivo.

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-original.s`	**`figura1-modificado_a.s`**	**`figura1-modificado_b.s`**

`.L11:`
`  movl  $0, -320068(%rbp)`
`  movl  $0, -320064(%rbp)`
`  movl  $0, -320072(%rbp)`
`  jmp  .L5`
	`.L9:`
	`  movl  $0, -320068(%rbp)`
	`  movl  $0, -320064(%rbp)`
	`  movl  $0, -320072(%rbp)`
	`  jmp  .L5`
		`.L17:`
		`  movl  $0, -320092(%rbp)`
		`  movl  $0, -320088(%rbp)`
		`  movl  $0, -320096(%rbp)`
		`  jmp  .L5`

```
.L6:
  movl  -320072(%rbp), %eax
  cltq
  movl  s(,%rax,8), %eax
  leal  (%rax,%rax), %edx
  movl  -320076(%rbp), %eax
  addl  %edx, %eax
  addl  %eax, -320068(%rbp)
  addl  $1, -320072(%rbp)
.L5:
  cmpl  $4999, -320072(%rbp)
  jle  .L6
  movl  $0, -320072(%rbp)
  jmp  .L7
.L8:
  movl  -320072(%rbp), %eax
  cltq
  movl  s+4(,%rax,8), %edx
  movl  %edx, %eax
  addl  %eax, %eax
  addl  %edx, %eax
  subl  -320076(%rbp), %eax
  addl  %eax, -320064(%rbp)
  addl  $1, -320072(%rbp)
.L7:
  cmpl  $4999, -320072(%rbp)
  jle  .L8
  movl  -320068(%rbp), %eax
  cmpl  -320064(%rbp), %eax
  jge  .L9
  movl  -320068(%rbp), %eax
  movslq  %eax, %rdx
  movl  -320076(%rbp), %eax
  cltq
  movq  %rdx, -320016(%rbp,
%rax,8)
  jmp  .L10
.L9:
  movl  -320064(%rbp), %eax
  movslq  %eax, %rdx
  movl  -320076(%rbp), %eax
  cltq
  movq  %rdx, -320016(%rbp,
%rax,8)
.L10:
  addl  $1, -320076(%rbp)
```

```
.L6:
  movl  -320072(%rbp), %eax
  cltq
  movl  s(,%rax,8), %eax
  leal  (%rax,%rax), %edx
  movl  -320076(%rbp), %eax
  addl  %edx, %eax
  addl  %eax, -320068(%rbp)
  movl  -320072(%rbp), %eax
  cltq
  movl  s+4(,%rax,8), %edx
  movl  %edx, %eax
  addl  %eax, %eax
  addl  %edx, %eax
  subl  -320076(%rbp), %eax
  addl  %eax, -320064(%rbp)
  addl  $1, -320072(%rbp)
.L5:
  cmpl  $4999, -320072(%rbp)
  jle  .L6
  movl  -320068(%rbp), %eax
  cmpl  -320064(%rbp), %eax
  jge  .L7
  movl  -320068(%rbp), %eax
  movslq  %eax, %rdx
  movl  -320076(%rbp), %eax
  cltq
  movq  %rdx, -320016(%rbp,
%rax,8)
  jmp  .L8
.L7:
  movl  -320064(%rbp), %eax
  movslq  %eax, %rdx
  movl  -320076(%rbp), %eax
  cltq
  movq  %rdx, -320016(%rbp,
%rax,8)
.L8:
  addl  $1, -320076(%rbp)
```

```
.L6:
  movl  -320096(%rbp), %eax
  cltq
  movl  s(,%rax,8), %eax
  leal  (%rax,%rax), %edx
  movl  -320100(%rbp), %eax
  addl  %edx, %eax
  addl  %eax, -320092(%rbp)
  movl  -320096(%rbp), %eax
  addl  $1, %eax
  cltq
  movl  s(,%rax,8), %eax
  leal  (%rax,%rax), %edx
  movl  -320100(%rbp), %eax
  addl  %edx, %eax
  addl  $1, %eax
  addl  %eax, -320088(%rbp)
  movl  -320096(%rbp), %eax
  addl  $2, %eax
  cltq
  movl  s(,%rax,8), %eax
  leal  (%rax,%rax), %edx
  movl  -320100(%rbp), %eax
  addl  %edx, %eax
  addl  $2, %eax
  addl  %eax, -320084(%rbp)
  movl  -320096(%rbp), %eax
  addl  $3, %eax
  cltq
  movl  s(,%rax,8), %eax
  leal  (%rax,%rax), %edx
  movl  -320100(%rbp), %eax
  addl  %edx, %eax
  addl  $3, %eax
  addl  %eax, -320080(%rbp)
  addl  $4, -320096(%rbp)
.L5:
  cmpl  $4999, -320096(%rbp)
  jle  .L6
  movl  $0, -320096(%rbp)
  jmp  .L7
.L8:
  movl  -320096(%rbp), %eax
  addl  $1, %eax
  cltq
  movl  s+4(,%rax,8), %edx
  movl  %edx, %eax
  addl  %eax, %eax
  addl  %edx, %eax
  subl  -320100(%rbp), %eax
  addl  %eax, -320076(%rbp)
  movl  -320096(%rbp), %eax
  addl  $1, %eax
  cltq
  movl  s+4(,%rax,8), %edx
  movl  %edx, %eax
  addl  %eax, %eax
  addl  %edx, %eax
  subl  -320100(%rbp), %eax
  addl  $1, %eax
  addl  %eax, -320072(%rbp)
  movl  -320096(%rbp), %eax
```

```
  addl  $1, %eax
  cltq
  movl  s+4(,%rax,8), %edx
  movl  %edx, %eax
  addl  %eax, %eax
  addl  %edx, %eax
  subl  -320100(%rbp), %eax
  addl  $2, %eax
  addl  %eax, -320068(%rbp)
  movl  -320096(%rbp), %eax
  addl  $1, %eax
  cltq
  movl  s+4(,%rax,8), %edx
  movl  %edx, %eax
  addl  %eax, %eax
  addl  %edx, %eax
  subl  -320100(%rbp), %eax
  addl  $3, %eax
  addl  %eax, -320064(%rbp)
  addl  $4, -320096(%rbp)
.L7:
  cmpl  $4999, -320096(%rbp)
  jle  .L8
  movl  -320092(%rbp), %eax
  cmpl  -320076(%rbp), %eax
  jge  .L9
  movl  -320092(%rbp), %eax
  movslq  %eax, %rdx
  movl  -320100(%rbp), %eax
  cltq
  movq  %rdx, -320016(%rbp,
%rax,8)
  jmp  .L10
.L9:
  movl  -320076(%rbp), %eax
  movslq  %eax, %rdx
  movl  -320100(%rbp), %eax
  cltq
  movq  %rdx, -320016(%rbp,
%rax,8)
.L10:
  movl  -320088(%rbp), %eax
  cmpl  -320072(%rbp), %eax
  jge  .L11
  movl  -320088(%rbp), %eax
  movslq  %eax, %rdx
  movl  -320100(%rbp), %eax
  cltq
  movq  %rdx, -320016(%rbp,
%rax,8)
  jmp  .L12
.L11:
  movl  -320072(%rbp), %eax
  movslq  %eax, %rdx
  movl  -320100(%rbp), %eax
  cltq
  movq  %rdx, -320016(%rbp,
%rax,8)
.L12:
  movl  -320084(%rbp), %eax
  cmpl  -320068(%rbp), %eax
  jge  .L13
```

		movl -320084(%rbp), %eax
		movslq %eax, %rdx
		movl -320100(%rbp), %eax
		cltq
		movq %rdx, -320016(%rbp, %rax,8)
		jmp .L14
		.L13:
		movl -320068(%rbp), %eax
		movslq %eax, %rdx
		movl -320100(%rbp), %eax
		cltq
		movq %rdx, -320016(%rbp, %rax,8)
		.L14:
		movl -320080(%rbp), %eax
		cmpl -320064(%rbp), %eax
		jge .L15
		movl -320080(%rbp), %eax
		movslq %eax, %rdx
		movl -320100(%rbp), %eax
		cltq
		movq %rdx, -320016(%rbp, %rax,8)
		jmp .L16
		.L15:
		movl -320064(%rbp), %eax
		movslq %eax, %rdx
		movl -320100(%rbp), %eax
		cltq
		movq %rdx, -320016(%rbp, %rax,8)
		.L16:
		addl $1, -320100(%rbp)

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

   if(argc < 3){
      fprintf(stderr,"Falta el tamaño y la constante\n");
      exit(-1);
   }

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

   unsigned int N = atoi(argv[1]);
   unsigned int a = atoi(argv[2]);
   unsigned int i;

   long long unsigned int *x, *y;
   x = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));
   y = (long long unsigned int*) malloc(N*sizeof(long long unsigned int));

   if ( (x==NULL) || (y==NULL) ){
      printf("Error en la reserva de espacio para las matrices\n");
      exit(-2);
   }

   // Inicialización del vector
   for(i=0;i<N;i++)  x[i] = i;

   clock_gettime(CLOCK_REALTIME,&cgt1);
   // Cómputos
   for (i=0;i<N;i++) y[i]= a*x[i] + y[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("Tiempo(seg.):%11.9f\t / Tamaño de N:%u\n",ncgt,N);

   free(x);
   free(y);

   return 0;
}
```

Tiempos ejec.	-O0	-Os	-O2	-O3
	1,95	*0,85*	*0,86*	*1,37*

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

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-25 viernes
$ gcc -O0 daxpy.c -o daxpy-O0 -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-25 viernes
$ ./daxpy-O0 200000000 5
Tiempo(seg.):1.950863346         / Tamaño de N:200000000
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-25 viernes
$ gcc -O2 daxpy.c -o daxpy-O2 -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-25 viernes
$ ./daxpy-O2 200000000 5
Tiempo(seg.):0.867336042         / Tamaño de N:200000000
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-25 viernes
$ ./daxpy-O2 200000000 5
Tiempo(seg.):0.848860415         / Tamaño de N:200000000
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-25 viernes
$ gcc -O3 daxpy.c -o daxpy-O3 -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-25 viernes
$ ./daxpy-O3 200000000 5
Tiempo(seg.):1.371631149         / Tamaño de N:200000000
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP4:
2018-05-25 viernes
$ ./daxpy-O3 200000000 5
Tiempo(seg.):1.370957617         / Tamaño de N:200000000
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:
En la compilación con `-O0` se puede observar que se usan direcciones relativas a pila, lo que supone utilizar una gran cantidad de operaciones `move`. Para `-Os` y `-O2` tenemos un código muy parecido, con apenas algunas diferencias en cuanto a los registros usados, pero ambos mucho más escuetos que el código generado con `-O0`, lo cual explica que el tiempo de ejecución de estos dos últimos sean prácticamente iguales y ambos dos menores que el tiempo de ejecución para `-O0`. Para la compilación con la opción `-O3` se genera un código en ensamblador más extenso que los anteriores donde se puede apreciar que el compilador realiza un desenrollado de bucle; además utiliza SSE (Streaming SIMD Extensions) que es esencialmente el equivalente de coma flotante de las instrucciones MMX. Por esto el programa que utiliza optimización `-O3` termina tardando más que los dos anteriores, aunque no tanto como con `-O0`.

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	daxpy0s.s	daxpy02.s	daxpy03.s

.L9:
movl -80(%rbp), %eax
leaq 0(,%rax,8),%rdx
movq -64(%rbp), %rax
addq %rdx, %rax
movl -76(%rbp), %ecx
movl -80(%rbp), %edx
	.L6:
	cmpl %edx, %ebx
	jbe .L13
	movq %r13, %rax
	imulq 0(%rbp, %rdx,8), %rax
	addq %rax,
		.L8:
		movq %rax, %rcx
		imulq 0(%rbp,%rdx,8), %rcx
		addq %rcx, (%r12,%rdx,8)
		addq $1, %rdx
			.L16:
			movdqu(%rdi,%rax),%xmm2
			addl $1, %ecx
			movdqa %xmm2, %xmm1
			movdqa %xmm2, %xmm4
			pmuludq %xmm5, %xmm2
			psrlq $32, %xmm1

```
  leaq  0(,%rdx,8),%rsi
  movq  -72(%rbp), %rdx
  addq  %rsi, %rdx
  movq  (%rdx), %rdx
  imulq  %rdx, %rcx
  movl  -80(%rbp), %edx
  leaq  0(,%rdx,8),%rsi
  movq  -64(%rbp), %rdx
  addq  %rsi, %rdx
  movq  (%rdx), %rdx
  addq  %rcx, %rdx
  movq  %rdx, (%rax)
  addl  $1, -80(%rbp)
```

```
(%r12,%rdx,8)
  incq  %rdx
  jmp  .L6
```

```
  cmpl  %edx, %ebx
  ja  .L8
```

```
  movdqa  %xmm1, %xmm0
  pmuludq  %xmm3, %xmm4
  pmuludq  %xmm3, %xmm0
  paddq  %xmm2, %xmm0
  psllq  $32, %xmm0
  paddq  %xmm4, %xmm0
  paddq (%rdx,%rax),%xmm0
  movaps %xmm0,(%rdx,rax)
  addq  $16, %rax
  cmpl  %esi, %ecx
  jb  .L16
  jmp  .L24
```