

2º curso / 2º cuatr.
Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 5. Optimización de código

Estudiante (nombre y apellidos): Mario Garcia Marquez

Grupo de prácticas y profesor de prácticas: Maribel Garcia Arenas

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): *AMD RYZEN 2700*

Sistema operativo utilizado: *Arch Linux*

Versión de gcc utilizada: 11.1.0

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

```
ac@bp4 on ~$ lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
Address sizes:            43 bits physical, 48 bits virtual
CPU(s):                  16
On-line CPU(s) list:      0-15
Thread(s) per core:       2
Core(s) per socket:       8
Socket(s):                1
NUMA node(s):            1
Vendor ID:                AuthenticAMD
CPU family:              23
Model:                   8
Model name:               AMD Ryzen 7 2700 Eight-Core Processor
Stepping:                 2
Frequency boost:          enabled
CPU MHz:                  3200.000
CPU max MHz:              3200.0000
CPU min MHz:              1550.0000
BogoMIPS:                 6389.24
Virtualization:           AMD-V
L1d cache:                256 KiB
L1i cache:                512 KiB
L2 cache:                 4 MiB
L3 cache:                 16 MiB
NUMA node0 CPU(s):       0-15
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf:       Not affected
Vulnerability Mds:        Not affected
Vulnerability Meltdown:   Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:  Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:  Mitigation; Full AMD retpoline, IBPB conditional, STIBP disabled, RSB filling
Vulnerability Srbds:       Not affected
Vulnerability Tsx async abort: Not affected
Flags:                     fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc
rep_good nopl nonstop_tsc cpuid extd_apicid aperfmperf pni pclmulqdq monitor sse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw skinit wdt tce topoext perfctr_core perfctr_nb bpext perfctr_llc mwaitx cpb hw_pstate s
shd ibpb vmmcall fsgsbase bmi1 avx2 smep bmi2 rdseed adx smap clflushopt sha_ni xsaveopt xsavec xgetbv1 xsaves clzero irperf xsaveerptr arat npt lbrv svm_lock
nrip save tsc_scale vmcb_clean flushbyasid decodeassists pausefilter pfthreshold avic vmsave_vmload vgif overflow_recov succor smca sme sev sev_es
```

1. (a) Implementar un código secuencial que calcule la multiplicación de dos matrices cuadradas. Utilizar como base el código de suma de vectores de BP0. Los datos se deben generar de forma aleatoria para un número de filas mayor que 8, como en el ejemplo de BP0, se puede usar `drand48()`.

MULTIPLICACIÓN DE MATRICES:

CAPTURA CÓDIGO FUENTE: `pmm-secuencial.c`

```

67 for (i = 0; i < N; i++) {
68     m1[i] = (double *)malloc(N * sizeof(double));
69     m2[i] = (double *)malloc(N * sizeof(double));
70     mr[i] = (double *)malloc(N * sizeof(double));
71 }
72 if ((m1 == NULL) || (m2 == NULL) || (mr == NULL)) {
73     printf("No hay suficiente espacio para los vectores \n");
74     exit(-2);
75 }
76
77 // Inicializar vectores
78 if (N < 9)
79     for (i = 0; i < N; i++) {
80         for (j = 0; j < N; j++) {
81             m1[i][j] = N * 0.1 + i * 0.1 + j * 0.1;
82             m2[i][j] = N * 0.1 - i * 0.1 - j * 0.1;
83             mr[i][j] = 0;
84         }
85     }
86 else {
87     srand(time(0));
88     for (i = 0; i < N; i++) {
89         for (j = 0; j < N; j++) {
90             m1[i][j] = drand48();
91             m2[i][j] = drand48();
92         }
93     }
94 }
95
96 clock_gettime(CLOCK_REALTIME, &cgt1);
97 // Producto matrices
98 for (i = 0; i < N; i++) {
99     for (j = 0; j < N; j++) {
100         suma = 0;
101         for (k = 0; k < N; k++) {
102             suma += m1[i][k] * m2[k][j];
103         }
104         mr[i][j] = suma;
105     }
106 }
107
108 clock_gettime(CLOCK_REALTIME, &cgt2);
109 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
110         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
111
112 // Imprimir resultado de la suma y el tiempo de ejecución
113 printf("Tiempo:%11.9f\t / Dim matrices:%u\t/ "
114        "mr[0][0](%8.6f) / / "
115        "\n");
116
117 COMMAND  @ ./pmm-secuencial.c
118 !:scro

```

(b) Modificar el código (solo el trozo que calcula la multiplicación) para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre -O2) a partir de la modificación realizada. Incorporar los códigos modificados en el cuaderno.

MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación–:

Modificación B) –explicación–:

...

CÓDIGOS FUENTE MODIFICACIONES

A) Captura de pmm-secuencial-modificado_A.c

```

39 }
38 if ((m1 == NULL) || (m2 == NULL) || (mr == NULL)) {
37     printf("No hay suficiente espacio para los vectores \n");
36     exit(-2);
35 }
34
33 // Inicializar vectores
32 if (N < 9)
31     for (i = 0; i < N; i++) {
30         for (j = 0; j < N; j++) {
29             m1[i][j] = N * 0.1 + i * 0.1 + j * 0.1;
28             m2[i][j] = N * 0.1 - i * 0.1 - j * 0.1;
27         }
26     }
25 else {
24     srand(time(0));
23     for (i = 0; i < N; i++) {
22         for (j = 0; j < N; j++) {
21             m1[i][j] = drand48();
20             m2[i][j] = drand48();
19         }
18     }
17     for (i = 0; i < N; i++) {
16         for (j = 0; j < N; j++) {
15             m2trans[j][i] = m2[i][j];
14         }
13     }
12 }
11
10 clock_gettime(CLOCK_REALTIME, &cgt1);
9 // Transponemos m2 para ponerla adecuadamente
8
7 for (i = 0; i < N; i++) {
6     for (j = 0; j < N; j++) {
5         suma = 0;
4         for (k = 0; k < N; k++) {
3             suma += m1[i][k] * m2trans[j][k];
2         }
1         mr[i][j] = suma;
68     }
1
2
3 clock_gettime(CLOCK_REALTIME, &cgt2);
4 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
5         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
6
7 // Imprimir resultado de la suma y el tiempo de ejecución
8 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

COMMAND `./ppm-secuencial.c`

main ac 74%

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

```

ac/bp4/ejer1 on main [?] took 15s
> gcc ppm-secuencial.c -O2
ac/bp4/ejer1 on main [?]
> ./a.out 1000
Tiempo:4.462880374 / Dim matrices:1000

```

B) ...

```

25 for (j = 0; j < N; j++) {
24     m1[i][j] = N * 0.1 + i * 0.1 + j * 0.1;
23     m2[i][j] = N * 0.1 - i * 0.1 - j * 0.1;
22     mr[i][j] = 0;
21 }
20
19 else {
18     srand(time(0));
17     for (i = 0; i < N; i++) {
16         for (j = 0; j < N; j++) {
15             m1[i][j] = drand48();
14             m2[i][j] = drand48();
13             mr[i][j] = 0;
12         }
11     }
10     for (i = 0; i < N; i++) {
9         for (j = 0; j < N; j++) {
8             m2trans[j][i] = m2[i][j];
7         }
6     }
5 }
4
3 clock_gettime(CLOCK_REALTIME, &cgt1);
2 // Transponemos m2 para ponerla adecuadamente
1
65 for (i = 0; i < N; i++) {
1     for (j = 0; j < N; j++) {
2         for (k = 0; k < N; k++) {
3             mr[i][j] += m1[i][k] * m2trans[j][k];
4             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
5             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
6             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
7         }
8     }
9 }
10
11 clock_gettime(CLOCK_REALTIME, &cgt2);
12 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
13         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
14
15 // Imprimir resultado de la suma y el tiempo de ejecución
16 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
17
18     "mr[0][0] (%8.6f) / /",
19     "mr[%d][%d] (%8.6f) \n",
20     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
21     mr[Nviejo - 1][Nviejo - 1]);
22
23 for (i = 0; i < N; i++) {
24     for (j = 0; j < N; j++) {
25         for (k = 0; k < N; k++) {
26             mr[i][j] += m1[i][k] * m2trans[j][k];
27             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
28             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
29             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
30         }
31     }
32 }
33
34 clock_gettime(CLOCK_REALTIME, &cgt2);
35 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
36         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
37
38 // Imprimir resultado de la suma y el tiempo de ejecución
39 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
40
41     "mr[0][0] (%8.6f) / /",
42     "mr[%d][%d] (%8.6f) \n",
43     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
44     mr[Nviejo - 1][Nviejo - 1]);
45
46 for (i = 0; i < N; i++) {
47     for (j = 0; j < N; j++) {
48         for (k = 0; k < N; k++) {
49             mr[i][j] += m1[i][k] * m2trans[j][k];
50             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
51             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
52             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
53         }
54     }
55 }
56
57 clock_gettime(CLOCK_REALTIME, &cgt2);
58 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
59         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
60
61 // Imprimir resultado de la suma y el tiempo de ejecución
62 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
63
64     "mr[0][0] (%8.6f) / /",
65     "mr[%d][%d] (%8.6f) \n",
66     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
67     mr[Nviejo - 1][Nviejo - 1]);
68
69 for (i = 0; i < N; i++) {
70     for (j = 0; j < N; j++) {
71         for (k = 0; k < N; k++) {
72             mr[i][j] += m1[i][k] * m2trans[j][k];
73             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
74             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
75             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
76         }
77     }
78 }
79
80 clock_gettime(CLOCK_REALTIME, &cgt2);
81 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
82         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
83
84 // Imprimir resultado de la suma y el tiempo de ejecución
85 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
86
87     "mr[0][0] (%8.6f) / /",
88     "mr[%d][%d] (%8.6f) \n",
89     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
90     mr[Nviejo - 1][Nviejo - 1]);
91
92 for (i = 0; i < N; i++) {
93     for (j = 0; j < N; j++) {
94         for (k = 0; k < N; k++) {
95             mr[i][j] += m1[i][k] * m2trans[j][k];
96             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
97             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
98             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
99         }
100     }
101 }
102
103 clock_gettime(CLOCK_REALTIME, &cgt2);
104 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
105         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
106
107 // Imprimir resultado de la suma y el tiempo de ejecución
108 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
109
110     "mr[0][0] (%8.6f) / /",
111     "mr[%d][%d] (%8.6f) \n",
112     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
113     mr[Nviejo - 1][Nviejo - 1]);
114
115 for (i = 0; i < N; i++) {
116     for (j = 0; j < N; j++) {
117         for (k = 0; k < N; k++) {
118             mr[i][j] += m1[i][k] * m2trans[j][k];
119             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
120             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
121             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
122         }
123     }
124 }
125
126 clock_gettime(CLOCK_REALTIME, &cgt2);
127 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
128         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
129
130 // Imprimir resultado de la suma y el tiempo de ejecución
131 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
132
133     "mr[0][0] (%8.6f) / /",
134     "mr[%d][%d] (%8.6f) \n",
135     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
136     mr[Nviejo - 1][Nviejo - 1]);
137
138 for (i = 0; i < N; i++) {
139     for (j = 0; j < N; j++) {
140         for (k = 0; k < N; k++) {
141             mr[i][j] += m1[i][k] * m2trans[j][k];
142             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
143             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
144             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
145         }
146     }
147 }
148
149 clock_gettime(CLOCK_REALTIME, &cgt2);
150 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
151         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
152
153 // Imprimir resultado de la suma y el tiempo de ejecución
154 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
155
156     "mr[0][0] (%8.6f) / /",
157     "mr[%d][%d] (%8.6f) \n",
158     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
159     mr[Nviejo - 1][Nviejo - 1]);
160
161 for (i = 0; i < N; i++) {
162     for (j = 0; j < N; j++) {
163         for (k = 0; k < N; k++) {
164             mr[i][j] += m1[i][k] * m2trans[j][k];
165             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
166             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
167             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
168         }
169     }
170 }
171
172 clock_gettime(CLOCK_REALTIME, &cgt2);
173 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
174         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
175
176 // Imprimir resultado de la suma y el tiempo de ejecución
177 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
178
179     "mr[0][0] (%8.6f) / /",
180     "mr[%d][%d] (%8.6f) \n",
181     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
182     mr[Nviejo - 1][Nviejo - 1]);
183
184 for (i = 0; i < N; i++) {
185     for (j = 0; j < N; j++) {
186         for (k = 0; k < N; k++) {
187             mr[i][j] += m1[i][k] * m2trans[j][k];
188             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
189             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
190             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
191         }
192     }
193 }
194
195 clock_gettime(CLOCK_REALTIME, &cgt2);
196 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
197         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
198
199 // Imprimir resultado de la suma y el tiempo de ejecución
200 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
201
202     "mr[0][0] (%8.6f) / /",
203     "mr[%d][%d] (%8.6f) \n",
204     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
205     mr[Nviejo - 1][Nviejo - 1]);
206
207 for (i = 0; i < N; i++) {
208     for (j = 0; j < N; j++) {
209         for (k = 0; k < N; k++) {
210             mr[i][j] += m1[i][k] * m2trans[j][k];
211             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
212             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
213             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
214         }
215     }
216 }
217
218 clock_gettime(CLOCK_REALTIME, &cgt2);
219 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
220         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
221
222 // Imprimir resultado de la suma y el tiempo de ejecución
223 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
224
225     "mr[0][0] (%8.6f) / /",
226     "mr[%d][%d] (%8.6f) \n",
227     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
228     mr[Nviejo - 1][Nviejo - 1]);
229
230 for (i = 0; i < N; i++) {
231     for (j = 0; j < N; j++) {
232         for (k = 0; k < N; k++) {
233             mr[i][j] += m1[i][k] * m2trans[j][k];
234             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
235             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
236             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
237         }
238     }
239 }
240
241 clock_gettime(CLOCK_REALTIME, &cgt2);
242 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
243         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
244
245 // Imprimir resultado de la suma y el tiempo de ejecución
246 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
247
248     "mr[0][0] (%8.6f) / /",
249     "mr[%d][%d] (%8.6f) \n",
250     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
251     mr[Nviejo - 1][Nviejo - 1]);
252
253 for (i = 0; i < N; i++) {
254     for (j = 0; j < N; j++) {
255         for (k = 0; k < N; k++) {
256             mr[i][j] += m1[i][k] * m2trans[j][k];
257             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
258             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
259             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
260         }
261     }
262 }
263
264 clock_gettime(CLOCK_REALTIME, &cgt2);
265 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
266         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
267
268 // Imprimir resultado de la suma y el tiempo de ejecución
269 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
270
271     "mr[0][0] (%8.6f) / /",
272     "mr[%d][%d] (%8.6f) \n",
273     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
274     mr[Nviejo - 1][Nviejo - 1]);
275
276 for (i = 0; i < N; i++) {
277     for (j = 0; j < N; j++) {
278         for (k = 0; k < N; k++) {
279             mr[i][j] += m1[i][k] * m2trans[j][k];
280             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
281             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
282             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
283         }
284     }
285 }
286
287 clock_gettime(CLOCK_REALTIME, &cgt2);
288 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
289         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
290
291 // Imprimir resultado de la suma y el tiempo de ejecución
292 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
293
294     "mr[0][0] (%8.6f) / /",
295     "mr[%d][%d] (%8.6f) \n",
296     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
297     mr[Nviejo - 1][Nviejo - 1]);
298
299 for (i = 0; i < N; i++) {
300     for (j = 0; j < N; j++) {
301         for (k = 0; k < N; k++) {
302             mr[i][j] += m1[i][k] * m2trans[j][k];
303             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
304             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
305             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
306         }
307     }
308 }
309
310 clock_gettime(CLOCK_REALTIME, &cgt2);
311 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
312         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
313
314 // Imprimir resultado de la suma y el tiempo de ejecución
315 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
316
317     "mr[0][0] (%8.6f) / /",
318     "mr[%d][%d] (%8.6f) \n",
319     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
320     mr[Nviejo - 1][Nviejo - 1]);
321
322 for (i = 0; i < N; i++) {
323     for (j = 0; j < N; j++) {
324         for (k = 0; k < N; k++) {
325             mr[i][j] += m1[i][k] * m2trans[j][k];
326             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
327             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
328             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
329         }
330     }
331 }
332
333 clock_gettime(CLOCK_REALTIME, &cgt2);
334 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
335         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
336
337 // Imprimir resultado de la suma y el tiempo de ejecución
338 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
339
340     "mr[0][0] (%8.6f) / /",
341     "mr[%d][%d] (%8.6f) \n",
342     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
343     mr[Nviejo - 1][Nviejo - 1]);
344
345 for (i = 0; i < N; i++) {
346     for (j = 0; j < N; j++) {
347         for (k = 0; k < N; k++) {
348             mr[i][j] += m1[i][k] * m2trans[j][k];
349             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
350             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
351             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
352         }
353     }
354 }
355
356 clock_gettime(CLOCK_REALTIME, &cgt2);
357 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
358         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
359
360 // Imprimir resultado de la suma y el tiempo de ejecución
361 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
362
363     "mr[0][0] (%8.6f) / /",
364     "mr[%d][%d] (%8.6f) \n",
365     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
366     mr[Nviejo - 1][Nviejo - 1]);
367
368 for (i = 0; i < N; i++) {
369     for (j = 0; j < N; j++) {
370         for (k = 0; k < N; k++) {
371             mr[i][j] += m1[i][k] * m2trans[j][k];
372             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
373             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
374             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
375         }
376     }
377 }
378
379 clock_gettime(CLOCK_REALTIME, &cgt2);
380 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
381         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
382
383 // Imprimir resultado de la suma y el tiempo de ejecución
384 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
385
386     "mr[0][0] (%8.6f) / /",
387     "mr[%d][%d] (%8.6f) \n",
388     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
389     mr[Nviejo - 1][Nviejo - 1]);
390
391 for (i = 0; i < N; i++) {
392     for (j = 0; j < N; j++) {
393         for (k = 0; k < N; k++) {
394             mr[i][j] += m1[i][k] * m2trans[j][k];
395             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
396             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
397             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
398         }
399     }
400 }
401
402 clock_gettime(CLOCK_REALTIME, &cgt2);
403 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
404         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
405
406 // Imprimir resultado de la suma y el tiempo de ejecución
407 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
408
409     "mr[0][0] (%8.6f) / /",
410     "mr[%d][%d] (%8.6f) \n",
411     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
412     mr[Nviejo - 1][Nviejo - 1]);
413
414 for (i = 0; i < N; i++) {
415     for (j = 0; j < N; j++) {
416         for (k = 0; k < N; k++) {
417             mr[i][j] += m1[i][k] * m2trans[j][k];
418             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
419             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
420             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
421         }
422     }
423 }
424
425 clock_gettime(CLOCK_REALTIME, &cgt2);
426 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
427         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
428
429 // Imprimir resultado de la suma y el tiempo de ejecución
430 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
431
432     "mr[0][0] (%8.6f) / /",
433     "mr[%d][%d] (%8.6f) \n",
434     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
435     mr[Nviejo - 1][Nviejo - 1]);
436
437 for (i = 0; i < N; i++) {
438     for (j = 0; j < N; j++) {
439         for (k = 0; k < N; k++) {
440             mr[i][j] += m1[i][k] * m2trans[j][k];
441             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
442             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
443             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
444         }
445     }
446 }
447
448 clock_gettime(CLOCK_REALTIME, &cgt2);
449 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
450         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
451
452 // Imprimir resultado de la suma y el tiempo de ejecución
453 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
454
455     "mr[0][0] (%8.6f) / /",
456     "mr[%d][%d] (%8.6f) \n",
457     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
458     mr[Nviejo - 1][Nviejo - 1]);
459
460 for (i = 0; i < N; i++) {
461     for (j = 0; j < N; j++) {
462         for (k = 0; k < N; k++) {
463             mr[i][j] += m1[i][k] * m2trans[j][k];
464             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
465             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
466             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
467         }
468     }
469 }
470
471 clock_gettime(CLOCK_REALTIME, &cgt2);
472 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
473         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
474
475 // Imprimir resultado de la suma y el tiempo de ejecución
476 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
477
478     "mr[0][0] (%8.6f) / /",
479     "mr[%d][%d] (%8.6f) \n",
480     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
481     mr[Nviejo - 1][Nviejo - 1]);
482
483 for (i = 0; i < N; i++) {
484     for (j = 0; j < N; j++) {
485         for (k = 0; k < N; k++) {
486             mr[i][j] += m1[i][k] * m2trans[j][k];
487             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
488             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
489             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
490         }
491     }
492 }
493
494 clock_gettime(CLOCK_REALTIME, &cgt2);
495 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
496         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
497
498 // Imprimir resultado de la suma y el tiempo de ejecución
499 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
500
501     "mr[0][0] (%8.6f) / /",
502     "mr[%d][%d] (%8.6f) \n",
503     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
504     mr[Nviejo - 1][Nviejo - 1]);
505
506 for (i = 0; i < N; i++) {
507     for (j = 0; j < N; j++) {
508         for (k = 0; k < N; k++) {
509             mr[i][j] += m1[i][k] * m2trans[j][k];
510             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
511             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
512             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
513         }
514     }
515 }
516
517 clock_gettime(CLOCK_REALTIME, &cgt2);
518 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
519         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
520
521 // Imprimir resultado de la suma y el tiempo de ejecución
522 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
523
524     "mr[0][0] (%8.6f) / /",
525     "mr[%d][%d] (%8.6f) \n",
526     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
527     mr[Nviejo - 1][Nviejo - 1]);
528
529 for (i = 0; i < N; i++) {
530     for (j = 0; j < N; j++) {
531         for (k = 0; k < N; k++) {
532             mr[i][j] += m1[i][k] * m2trans[j][k];
533             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
534             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
535             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
536         }
537     }
538 }
539
540 clock_gettime(CLOCK_REALTIME, &cgt2);
541 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
542         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
543
544 // Imprimir resultado de la suma y el tiempo de ejecución
545 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
546
547     "mr[0][0] (%8.6f) / /",
548     "mr[%d][%d] (%8.6f) \n",
549     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
550     mr[Nviejo - 1][Nviejo - 1]);
551
552 for (i = 0; i < N; i++) {
553     for (j = 0; j < N; j++) {
554         for (k = 0; k < N; k++) {
555             mr[i][j] += m1[i][k] * m2trans[j][k];
556             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
557             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
558             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
559         }
560     }
561 }
562
563 clock_gettime(CLOCK_REALTIME, &cgt2);
564 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
565         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
566
567 // Imprimir resultado de la suma y el tiempo de ejecución
568 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
569
570     "mr[0][0] (%8.6f) / /",
571     "mr[%d][%d] (%8.6f) \n",
572     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
573     mr[Nviejo - 1][Nviejo - 1]);
574
575 for (i = 0; i < N; i++) {
576     for (j = 0; j < N; j++) {
577         for (k = 0; k < N; k++) {
578             mr[i][j] += m1[i][k] * m2trans[j][k];
579             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
580             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
581             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
582         }
583     }
584 }
585
586 clock_gettime(CLOCK_REALTIME, &cgt2);
587 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
588         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
589
590 // Imprimir resultado de la suma y el tiempo de ejecución
591 printf("Tiempo:%11.9f\t / Dim matrices:%u\t\n",
592
593     "mr[0][0] (%8.6f) / /",
594     "mr[%d][%d] (%8.6f) \n",
595     ncgt, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
596     mr[Nviejo - 1][Nviejo - 1]);
597
598 for (i = 0; i < N; i++) {
599     for (j = 0; j < N; j++) {
600         for (k = 0; k < N; k++) {
601             mr[i][j] += m1[i][k]
```

```
ac/bp4/ejer1 on ↩ main [?] took 5m16s
> g++ ppm-secuencial.c -O2
ac/bp4/ejer1 on ↩ main [?]
> ./a.out 1000
Tiempo:0.891208334 / Dim matrices:1000
```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	<i>Ninguna</i>	1000: 4.61s
Modificación A)	Guardamos m 2 transpuesta de forma que recorremos por columnas	1000: 4.46s
Modificación B)	Se ha desarrollado el bucle y se ha adaptado N	1000: 0.89s
...		

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

La principal mejora se debe a reducir los saltos del bucle y hacer el código mucho más directo ahorrando instrucciones de salto y comprobaciones.

- (a) Usando como base el código de BP0, generar un programa para evaluar un código de la Figura 1. M y N deben ser parámetros de entrada al programa. Los datos se deben generar de forma aleatoria para valores de M y N mayores que 8, como en el ejemplo de BP0.

CÓDIGO FIGURA 1:**CAPTURA CÓDIGO FUENTE:** figura1-original.c

```

25 for (j = 0; j < N; j++) {
26     m1[i][j] = N * 0.1 + i * 0.1 + j * 0.1;
27     m2[i][j] = N * 0.1 - i * 0.1 - j * 0.1;
28     mr[i][j] = 0;
29 }
30 else {
31     srand(time(0));
32     for (i = 0; i < N; i++) {
33         for (j = 0; j < N; j++) {
34             m1[i][j] = drand48();
35             m2[i][j] = drand48();
36             mr[i][j] = 0;
37         }
38     }
39     for (i = 0; i < N; i++) {
40         for (j = 0; j < N; j++) {
41             m2trans[j][i] = m2[i][j];
42         }
43     }
44 }
45
46 clock_gettime(CLOCK_REALTIME, &cgt1);
47 // Transponemos m2 para ponerla adecuadamente
48
49 65 for (i = 0; i < N; i++) {
50     for (j = 0; j < N; j++) {
51         for (k = 0; k < N; k++) {
52             mr[i][j] += m1[i][k] * m2trans[j][k];
53             mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
54             mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
55             mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
56         }
57     }
58 }
59
60 clock_gettime(CLOCK_REALTIME, &cgt2);
61 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
62         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
63
64 // Imprimir resultado de la suma y el tiempo de ejecución
65 printf("Tiempo:%11.9f\t / Dim matrices:%u\t / "
66        "mr[0][0] (%8.6f) / \n",
67        ncgt, N, mr[0][0], N, N);
68
69 for (i = 0; i < N; i++) {
70     for (j = 0; j < N; j++) {
71         mr[i][j] = 1;
72     }
73 }
74
75 COMMAND @ ./ppm-secuencial.c
76 !scrot modificado8codigo.png

```

Figura 1. Código C++ que suma dos vectores. M y N deben ser parámetros de entrada al programa, usar valores mayores

que 1000 en la evaluación.

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

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

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

(b) Modificar el código C (solo el trozo a evaluar) para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre -O2) a partir de la modificación realizada. En las ejecuciones de evaluación usar valores de N y M mayores que 1000. Incorporar los códigos modificados en el cuaderno.

MODIFICACIONES REALIZADAS (al menos dos modificaciones):

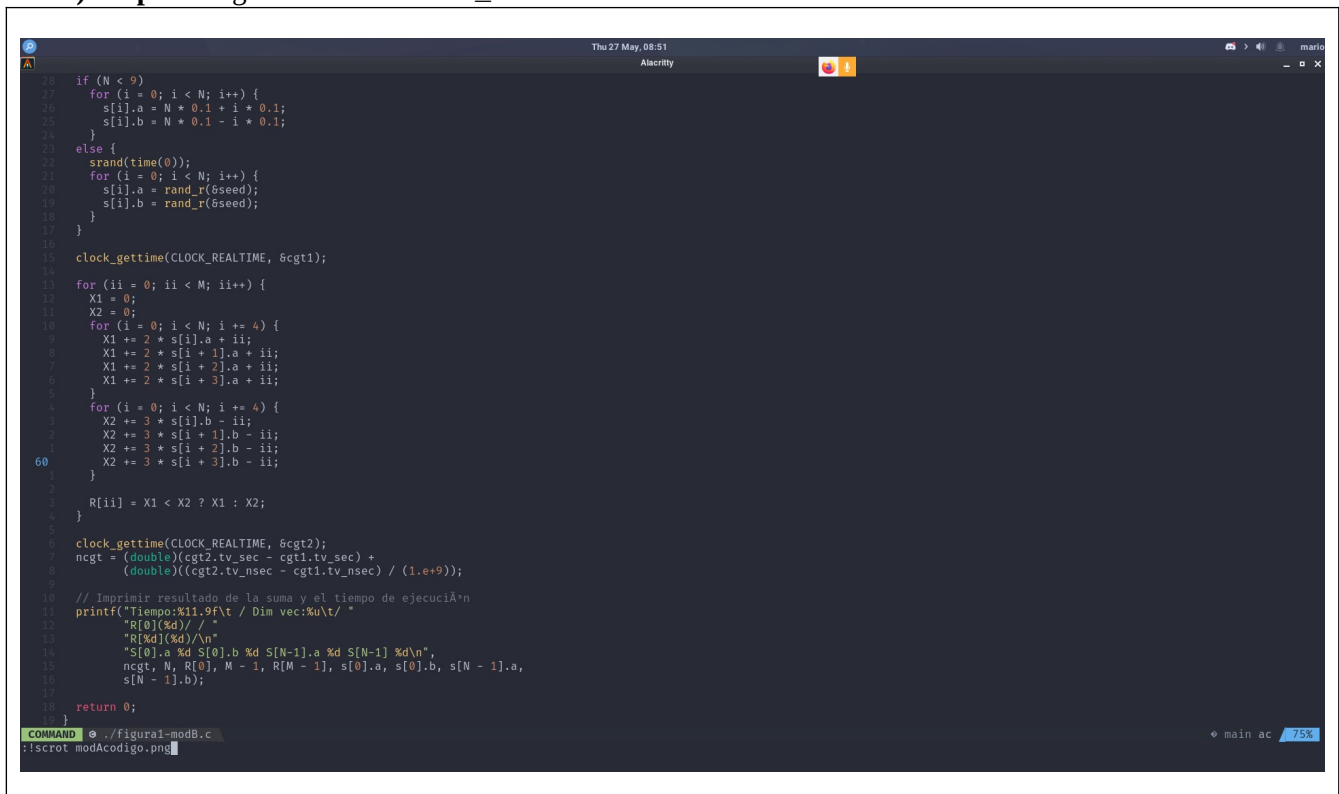
Modificación A) –explicación–:

Modificación B) –explicación–:

...

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c



```
Thu 27 May, 08:51
Alacritty
mario

28 if (N < 9)
29     for (i = 0; i < N; i++) {
30         s[i].a = N * 0.1 + i * 0.1;
31         s[i].b = N * 0.1 - i * 0.1;
32     }
33 else {
34     srand(time(0));
35     for (i = 0; i < N; i++) {
36         s[i].a = rand_r(&seed);
37         s[i].b = rand_r(&seed);
38     }
39 }
40
41 clock_gettime(CLOCK_REALTIME, &cgt1);
42
43 for (ii = 0; ii < M; ii++) {
44     X1 = 0;
45     X2 = 0;
46     for (i = 0; i < N; i += 4) {
47         X1 += 2 * s[i].a + ii;
48         X1 += 2 * s[i + 1].a + ii;
49         X1 += 2 * s[i + 2].a + ii;
50         X1 += 2 * s[i + 3].a + ii;
51     }
52     for (i = 0; i < N; i += 4) {
53         X2 += 3 * s[i].b - ii;
54         X2 += 3 * s[i + 1].b - ii;
55         X2 += 3 * s[i + 2].b - ii;
56         X2 += 3 * s[i + 3].b - ii;
57     }
58     R[ii] = X1 < X2 ? X1 : X2;
59 }
60
61 clock_gettime(CLOCK_REALTIME, &cgt2);
62 ncgt = ((double)(cgt2.tv_sec - cgt1.tv_sec) +
63         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9)));
64
65 // Imprimir resultado de la suma y el tiempo de ejecución
66 printf("Tiempo:%11.9f\t / Dim vec:%u\t / "
67        "R[0](%d) / "
68        "R[%d](%d) / \n",
69        "S[0].a %d S[0].b %d S[N-1].a %d S[N-1].b %d\n",
70        ncgt, N, R[0], M - 1, R[M - 1], s[0].a, s[0].b, s[N - 1].a,
71        s[N - 1].b);
72
73 return 0;
74 }
75
76 COMMAND @ ./Figura1-modB.c
77 !:scrot modAcodigo.png
```

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

```
ac/bp4/ejer2 on main [?]
> gcc -O2 figura1-modA.c
ac/bp4/ejer2 on main [?]
> ./a.out 100000 100000
Tiempo:5.663290820 / Dim vec:100000 / R[0](-899010246)/ / R[99999](228902132)/
S[0].a 1460154716 S[0].b 2098222173 S[N-1].a 831638063 S[N-1] 682130416
```

B) ...

```
ac/bp4/ejer2 on main [?] took 4m37s
> gcc -O2 figura1-modB.c
ac/bp4/ejer2 on main [?]
> ./a.out 100000 100000
Tiempo:5.432829394 / Dim vec:100000 / R[0](-1654327960)/ / R[99999](-1234210941)/
S[0].a 1460154716 S[0].b 2098222173 S[N-1].a 831638063 S[N-1] 682130416
```

```
47 unsigned int Nnuevo = N + 4 - (N % 4);
48 unsigned seed = 24567;
49
50 struct componentes s[Nnuevo];
51 int R[N];
52
53 // Inicializar vectores
54 if (N < 9)
55     for (i = 0; i < N; i++) {
56         s[i].a = N * 0.1 + i * 0.1;
57         s[i].b = N * 0.1 - i * 0.1;
58     }
59 else {
60     srand(time(0));
61     for (i = 0; i < N; i++) {
62         s[i].a = rand_r(&seed);
63         s[i].b = rand_r(&seed);
64     }
65 }
66
67 clock_gettime(CLOCK_REALTIME, &cgt1);
68
69 for (ii = 0; ii < M; ii++) {
70     X1 = 0;
71     X2 = 0;
72     for (i = 0; i < N; i += 4) {
73         X1 += 2 * s[i].a + ii;
74         X1 += 2 * s[i + 1].a + ii;
75         X1 += 2 * s[i + 2].a + ii;
76         X1 += 2 * s[i + 3].a + ii;
77         X2 += 3 * s[i].b - ii;
78         X2 += 3 * s[i + 1].b - ii;
79         X2 += 3 * s[i + 2].b - ii;
80         X2 += 3 * s[i + 3].b - ii;
81     }
82     R[ii] = X1 < X2 ? X1 : X2;
83 }
84
85 clock_gettime(CLOCK_REALTIME, &cgt2);
86 ncgt = ((double)(cgt2.tv_sec - cgt1.tv_sec) +
87         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9)));
88
89 // Imprimir resultado de la suma y el tiempo de ejecución
90 printf("Tiempo:%11.9f\t / Dim vec:%u\t / "
91        "R[0](%d) / "
92        "R[99999](%d)\n",
93        "S[0].a %d S[0].b %d S[N-1].a %d S[N-1] %d\n",
94        S[0].a, S[0].b, S[N-1].a, S[N-1]);
95
96 COMMAND ac/bp4/ejer2
97 !scrot modBcodigo.png
```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	Ninguna	100000: 9.45s
Modificación A)	Desarrollar el bucle	100000: 5.66s
Modificación B)	Se han juntado ambos bloques en uno	100000: 5.46
...		

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

Como en el ejercicio anterior el principal aumento en ganancias se debe a desarrollar el bucle, luego unificar ambos ha traído una leve mejora al código también.

3. 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=0;i<N;i++) y[i]= a*x[i] + y[i];
```

Generar 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. Incorporar los códigos al cuaderno de prácticas y destacar las diferencias entre ellos. Sólo se debe evaluar el tiempo del núcleo DAXPY. N deben ser parámetro de entrada al programa.

CAPTURA CÓDIGO FUENTE: daxpy.c

```

1 // Linpack benchmark
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6 #include <unistd.h>
7
8 // Leer argumento de entrada (nº de componentes del vector)
9 if (argc < 3) {
10     printf("Faltan nº componentes del vector\n");
11     exit(-1);
12 }
13
14 unsigned int N = atoi(argv[1]);
15 unsigned seed = 24567;
16 double a = drand48();
17
18 double x[N], y[N];
19
20 // Inicializar vectores
21 if (N < 9)
22     for (i = 0; i < N; i++) {
23         x[i] = N * 0.1 + i * 0.1;
24         y[i] = N * 0.1 + i * 0.1;
25     }
26 else {
27     srand(time(0));
28     for (i = 0; i < N; i++) {
29         x[i] = rand_r(&seed);
30         y[i] = rand_r(&seed);
31     }
32 }
33
34 clock_gettime(CLOCK_REALTIME, &cgt1);
35
36 for (i = 0; i < N; i++) {
37     y[i] = a * x[i] + y[i];
38 }
39
40 clock_gettime(CLOCK_REALTIME, &cgt2);
41 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
42         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
43
44 // Imprimir resultado de la suma y el tiempo de ejecución
45 printf("Tiempo:%11.9f\t / Dim vec:%u\t / "
46        "y[0](%f) // "
47        "y[N-1](%f)\n",
48        ncgt, N, y[0], y[N - 1]);
49
50 return 0;
51 }
52
53 COMMAND @ ./daxpy.c
54 !!scrot daxpySource.png

```

Tiempos ejec. Longitud vectores=5000000 00	-O0	-Os	-O2	-O3
	2.46s	0.56s	0.54s	0.53s

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

```
ac/bp4/ejer3 on $ main [?] took 44s
> gcc -O2 daxpy.c
ac/bp4/ejer3 on $ main [?]
> ./a.out 1000000
Tiempo:0.001112082 / Dim vec:1000000 / y[0](2098222173.000057)/ / y[N-1](1776869888.000057)/
ac/bp4/ejer3 on $ main [?]
> ./a.out 5000000
Tiempo:0.005356662 / Dim vec:5000000 / y[0](2098222173.000057)/ / y[N-1](1265952085.000038)/
ac/bp4/ejer3 on $ main [?]
> ./a.out 500000000
Tiempo:0.053293378 / Dim vec:500000000 / y[0](2098222173.000057)/ / y[N-1](2042628256.000006)/
ac/bp4/ejer3 on $ main [?]
> ./a.out 5000000000
Tiempo:0.548521556 / Dim vec:5000000000 / y[0](2098222173.000057)/ / y[N-1](109378441.000078)/
ac/bp4/ejer3 on $ main [?] took 7s
> gcc -O0 daxpy.c
ac/bp4/ejer3 on $ main [?]
> ./a.out 5000000000
Tiempo:2.467433228 / Dim vec:5000000000 / y[0](2098222173.000057)/ / y[N-1](109378441.000078)/
ac/bp4/ejer3 on $ main [?] took 9s
> gcc -Os daxpy.c
ac/bp4/ejer3 on $ main [?]
> ./a.out 5000000000
Tiempo:0.568560242 / Dim vec:5000000000 / y[0](2098222173.000057)/ / y[N-1](109378441.000078)/
ac/bp4/ejer3 on $ main [?] took 7s
> gcc -O3 daxpy.c
ac/bp4/ejer3 on $ main [?]
> ./a.out 5000000000
Tiempo:0.533375289 / Dim vec:5000000000 / y[0](2098222173.000057)/ / y[N-1](109378441.000078)/
ac/bp4/ejer3 on $ main [?] took 7s
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Como vemos O0, la opción por defecto, genera un código en ensamblador muy largo que se ve reducido al utilizar el método de compilación Os. Con O2 vemos que se ha quitado un salto para dejar el código en ensamblador con solo uno y finalmente O3 pasa a usar instrucciones vectoriales.

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> .L14: movl -88(%rbp), %eax cltq leaq 0(,%rax,8), %rdx movq -72(%rbp), %rax addq %rdx, %rax movsd (%rax), %xmm0 movapd %xmm0, %xmm1 mulsd -80(%rbp), %xmm1 movl -88(%rbp), %eax cltq leaq 0(,%rax,8), %rdx movq -64(%rbp), %rax addq %rdx, %rax movsd (%rax), %xmm0 movl -88(%rbp), %eax cltq leaq 0(,%rax,8), %rdx movq -64(%rbp), %rax addq %rdx, %rax addsd %xmm1, %xmm0 movsd %xmm0, (%rax) addl \$1, -88(%rbp) .L13: movl -88(%rbp), %eax cmpl %eax, -84(%rbp) ja .L14 leaq -32(%rbp), %rax movq %rax, %rsi movl \$0, %edi </pre>	<pre> .L8: cmpl %eax, %ebp jbe .L18 movq %r15, %xmm0 mulsd 0(%r13,%rax,8), %xmm0 addsd (%r12,%rax,8), %xmm0 movsd %xmm0, (%r12,%rax,8) incq %rax jmp .L8 .L18: </pre>	<pre> .L8: movsd (%rsp), %xmm0 mulsd (%r12,%rax,8), %xmm0 addsd 0(%rbp,%rax,8), %xmm0 movsd %xmm0, 0(%rbp,%rax,8) addq \$1, %rax cmpl %eax, %r15d ja .L8 </pre>	<pre> .L8: xorl %edi, %edi leaq 32(%rsp), %rsi call clock_gettime@PLT testl %r13d, %r13d movl \$1, %ecx cmovne %r13d, %ecx xorl %eax, %eax subl \$1, %r13d je .L14 jmp .L16 .L4: leaq 32(%rsp), %rsi xorl %edi, %edi movl \$4294967295, %ebx call clock_gettime@PLT jmp .L15 </pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4. (a) Paralarizar con OpenMP en la CPU el código de la multiplicación resultante en el Ejercicio 1.(b). NOTA: usar para generar los valores aleatorios, por ejemplo, `drand48_r ()`.

(b) Calcular la ganancia en prestaciones que se obtiene en `atcgrid4` para el máximo número de procesadores físicos con respecto al código inicial no optimizado del Ejercicio 1.(a) para dos tamaños de la matriz.

(a) **MULTIPLICACIÓN DE MATRICES PARALELO:**

CAPTURA CÓDIGO FUENTE: `pmm-paralelo.c`

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <omp.h>
4
5  int main() {
6      int N = 1024;
7      int m1[N][N];
8      int m2[N][N];
9      int mr[N][N];
10
11      for (i = 0; i < N; i++) {
12          for (j = 0; j < N; j++) {
13              m1[i][j] = drand48();
14              m2[i][j] = drand48();
15              mr[i][j] = 0;
16          }
17      }
18
19      for (i = 0; i < N; i++) {
20          for (j = 0; j < N; j++) {
21              m2trans[j][i] = m2[i][j];
22          }
23      }
24
25      t1 = omp_get_wtime();
26      // Transponemos m2 para ponerla adecuadamente
27
28      #pragma omp parallel for private(j, k)
29      for (i = 0; i < N; i++) {
30          for (j = 0; j < N; j++) {
31              for (k = 0; k < N; k += 4) {
32                  mr[i][j] += m1[i][k] * m2trans[j][k];
33                  mr[i][j] += m1[i][k + 1] * m2trans[j][k + 1];
34                  mr[i][j] += m1[i][k + 2] * m2trans[j][k + 2];
35                  mr[i][j] += m1[i][k + 3] * m2trans[j][k + 3];
36              }
37          }
38      }
39
40      t2 = omp_get_wtime();
41
42      // Imprimir resultado de la suma y el tiempo de ejecución
43      printf("Tiempo: %11.9f\t / Dim matrices: %u\t / ",
44             "mr[0][0](%8.6f) / \n",
45             t2 - t1, Nviejo, mr[0][0], Nviejo - 1, Nviejo - 1,
46             mr[Nviejo - 1][Nviejo - 1]);
47
48      for (i = 0; i < N; i++) {
49          free(m1[i]);
50          free(m2[i]);
51          free(mr[i]);
52          free(m2trans[i]);
53      }
54
55      free(m1); // libera el espacio reservado para m1
56      free(m2); // libera el espacio reservado para m2
57      free(mr); // libera el espacio reservado para mr
58
59      return 0;
60 }
61
62 COMMAND @ ./pmm-paralelo.c
63 !scrot codigoParalelo.png

```

(b) RESPUESTA

Para 1000:

Paralelo = 0.764s

Secuencial = 1.59s

Ganancia = 2.08

Para 2000:

Paralelo = 5.73s

secuencial = 41.34s

Ganancia = 7.21