2º curso / 2º cuatr. Grado Ing. Inform.	**Arquitectura de Computadores (AC)** **Cuaderno de prácticas.** **Bloque Práctico 5. Optimización de código** Estudiante (nombre y apellidos): Alberto Llamas González Grupo de prácticas y profesor de prácticas: D3, Juan Carlos Gómez López Fecha de entrega: Fecha evaluación en clase: 7/06/2021

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) i3-3110M CPU @ 2.40GHz*

Sistema operativo utilizado: *Ubuntu 18.04.5 LTS "bionic"*

Versión de gcc utilizada: *gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)*

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

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4] 2021-05-31 lunes
$lscpu
Arquitectura:                        x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:                  Little Endian
CPU(s):                              4
Lista de la(s) CPU(s) en línea:      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:                              58
Nombre del modelo:                   Intel(R) Core(TM) i3-3110M CPU @ 2.40GHz
Revisión:                            9
CPU MHz:                             1479.432
CPU MHz máx.:                        2400,0000
CPU MHz min.:                        1200,0000
BogoMIPS:                            4789.13
Virtualización:                      VT-x
Caché L1d:                           32K
Caché L1i:                           32K
Caché L2:                            256K
Caché L3:                            3072K
CPU(s) del nodo NUMA 0:              0-3
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 rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cp
l vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer xsave avx f16c lahf_lm cpuid_fault epb pti ssbd ibrs ibpb stibp
 tpr_shadow vnmi flexpriority ept vpid fsgsbase smep erms xsaveopt dtherm arat pln pts md_clear flush_l1d
```

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`

pmm-secuencial.c

ejer1 > pmm-secuencial.c > ...

```c
/**
 * @author Alberto Llamas Gonzalez
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

//He realizado el programa con matrices dinamicas

//Vamos a calcular A = B*C
int main(int argc, char **argv)
{
    struct timespec cgt1, cgt2;
    double t; // para tiempos

    if (argc < 2)
    {
        printf("[ERROR]-Debe insertar tamaño matriz\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);
    double **A, **B, **C;

    //Reservamos espacio para las matrices
    A = (double **)malloc(N * sizeof(double *));
    B = (double **)malloc(N * sizeof(double *));
    C = (double **)malloc(N * sizeof(double *));
```

pmm-secuencial.c

ejer1 > pmm-secuencial.c > ...

```c
    if ((A == NULL) || (B == NULL) || (C == NULL))
        printf("\nNo hay suficiente espacio para la matriz\n");

    for (int e = 0; e < N; e++){
        A[e] = (double *) malloc(N * sizeof(double));
        B[e] = (double *)malloc(N * sizeof(double));
        C[e] = (double *)malloc(N * sizeof(double));

        if ((A[e] == NULL) || (B[e] == NULL) || (C[e] == NULL))
            printf("\nNo hay suficiente espacio para la matriz en la columna\n");
    }
    int i, j, k;
    double suma;

    srand48(time(NULL));

    // Inicialización matrices B, C
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
        {
            if (N < 9){
                B[i][j] = j + 1;
                C[i][j] = j + 1;
            } else {
                B[i][j] = drand48();
                C[i][j] = drand48();
            }
        }
```

pmm-secuencial.c

ejer1 > pmm-secuencial.c > ...

```c
    // Tiempo
    clock_gettime(CLOCK_REALTIME, &cgt1);

    // Cálculo de A=B*C
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
        {
            A[i][j] = 0;
            for (k = 0; k < N; k++)
                A[i][j] = A[i][j] + B[i][k] * C[k][j];
        }

    // Tiempo
    clock_gettime(CLOCK_REALTIME, &cgt2);

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

    // Impresión de tiempo de ejecución
    printf("Tiempo (seg): %0.9f\n", t); //%0.9f define los decimales a mostrar

    // Impresión de resultados
    //Para un numero de filas mayor que 8 solo imprimimos el primer y ultimo termino (en este caso sera
    printf("\n_________\nResultados:\n");

    printf("\nMatriz B: \n");
    if (N < 9)
        for (i = 0; i < N; i++)
```

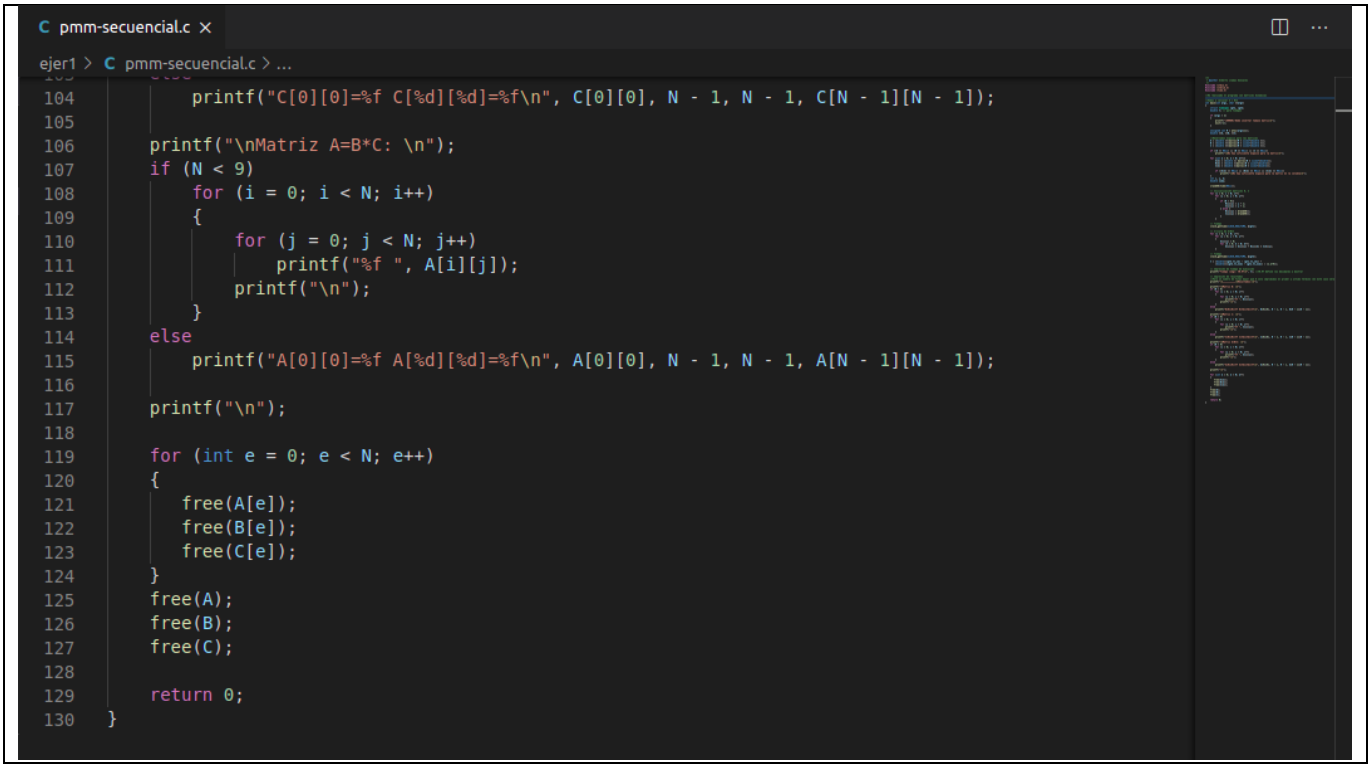
pmm-secuencial.c

ejer1 > pmm-secuencial.c > ...

```c
    printf("\nMatriz B: \n");
    if (N < 9)
        for (i = 0; i < N; i++)
        {
            for (j = 0; j < N; j++)
                printf("%f ", B[i][j]);
            printf("\n");
        }
    else
        printf("B[0][0]=%f B[%d][%d]=%f\n", B[0][0], N - 1, N - 1, B[N - 1][N - 1]);

    printf("\nMatriz C: \n");
    if (N < 9)
        for (i = 0; i < N; i++)
        {
            for (j = 0; j < N; j++)
                printf("%f ", C[i][j]);
            printf("\n");
        }
    else
        printf("C[0][0]=%f C[%d][%d]=%f\n", C[0][0], N - 1, N - 1, C[N - 1][N - 1]);

    printf("\nMatriz A=B*C: \n");
    if (N < 9)
        for (i = 0; i < N; i++)
        {
            for (j = 0; j < N; j++)
```



pmm-secuencial.c

ejer1 > pmm-secuencial.c > ...

```
104         printf("C[0][0]=%f C[%d][%d]=%f\n", C[0][0], N - 1, N - 1, C[N - 1][N - 1]);
105
106     printf("\nMatriz A=B*C: \n");
107     if (N < 9)
108         for (i = 0; i < N; i++)
109         {
110             for (j = 0; j < N; j++)
111                 printf("%f ", A[i][j]);
112             printf("\n");
113         }
114     else
115         printf("A[0][0]=%f A[%d][%d]=%f\n", A[0][0], N - 1, N - 1, A[N - 1][N - 1]);
116
117     printf("\n");
118
119     for (int e = 0; e < N; e++)
120     {
121        free(A[e]);
122        free(B[e]);
123        free(C[e]);
124     }
125     free(A);
126     free(B);
127     free(C);
128
129     return 0;
130 }
```

(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-: desenrollado de bucles, con 4 cálculos por iteración, teniendo en cuenta que el tamaño N no tiene por qué ser múltiplo de 4 (desenrollamos el for hasta N-N%4 y luego calculamos de forma independiente, fuera del for, desde N%4 hasta N).

Modificación B) –explicación-: cambio en el orden de ejecución: i,k,j en lugar de i,j,k, para disminuir las penalizaciones por caché.

CÓDIGOS FUENTE MODIFICACIONES

A) Captura de `pmm-secuencial-modificado_A.c`

Se muestra solo el código que se ha modificado respecto al programa inicial:

```c
    clock_gettime(CLOCK_REALTIME, &cgt1);

    int mod = N % 4;
    // Cálculo de A=B*C
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
        {
            A[i][j] = 0;
            for (k = 0; k < N - mod; k += 4)
            {
                A[i][j] = A[i][j] + B[i][k] * C[k][j];
                A[i][j] = A[i][j] + B[i][k + 1] * C[k + 1][j];
                A[i][j] = A[i][j] + B[i][k + 2] * C[k + 2][j];
                A[i][j] = A[i][j] + B[i][k + 3] * C[k + 3][j];
            }

            if (N - mod + 2 < N)
            {
                A[i][j] = A[i][j] + B[i][N - mod] * C[N - mod][j];
                A[i][j] = A[i][j] + B[i][N - mod + 1] * C[N - mod + 1][j];
                A[i][j] = A[i][j] + B[i][N - mod + 2] * C[N - mod + 2][j];
            }
            else if (N - mod + 1 < N)
            {
                A[i][j] = A[i][j] + B[i][N - mod] * C[N - mod][j];
                A[i][j] = A[i][j] + B[i][N - mod + 1] * C[N - mod + 1][j];
            }
            else if (N - mod < N)
            {
                A[i][j] = A[i][j] + B[i][N - mod] * C[N - mod][j];
            }
        }

    // Tiempo
    clock_gettime(CLOCK_REALTIME, &cgt2);
```

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

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$gcc -O2 -o pmm-secuencial-modificadoA pmm-secuencial-modificadoA.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$./pmm-secuencial-modificadoA 13
Tiempo (seg): 0.000005345

__________
Resultados:

Matriz B:
B[0][0]=0.668860 B[12][12]=0.990862

Matriz C:
C[0][0]=0.285291 C[12][12]=0.006513

Matriz A=B*C:
A[0][0]=3.733834 A[12][12]=3.348559
```

Vemos que disminuye el tiempo respecto al programa sin modificar:

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$./pmm-secuencial 13
Tiempo (seg): 0.000031154

__________
Resultados:

Matriz B:
B[0][0]=0.152069 B[12][12]=0.693568

Matriz C:
C[0][0]=0.103653 C[12][12]=0.295822

Matriz A=B*C:
A[0][0]=3.449452 A[12][12]=4.454223

[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$
```

Añadimos otra prueba con un valor < 9 para comprobar que calcula bien los resultados de hacer A=B*C

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$./pmm-secuencial-modificadoA 4
Tiempo (seg): 0.000001240

__________
Resultados:

Matriz B:
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000

Matriz C:
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000

Matriz A=B*C:
10.000000 20.000000 30.000000 40.000000
10.000000 20.000000 30.000000 40.000000
10.000000 20.000000 30.000000 40.000000
10.000000 20.000000 30.000000 40.000000

[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$./pmm-secuencial 4
Tiempo (seg): 0.000001790

__________
Resultados:

Matriz B:
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000

Matriz C:
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000
1.000000 2.000000 3.000000 4.000000

Matriz A=B*C:
10.000000 20.000000 30.000000 40.000000
10.000000 20.000000 30.000000 40.000000
10.000000 20.000000 30.000000 40.000000
10.000000 20.000000 30.000000 40.000000
```

B) Captura de `pmm-secuencial-modificado_B.c`

```
65      // Cálculo de A=B*C
66      for (i = 0; i < N; i++)
67          for (k = 0; k < N; k++)
68          {
69              A[i][j] = 0;
70              for (j = 0; j < N; j++)
71                  A[i][j] = A[i][j] + B[i][k] * C[k][j];
72          }
73
74      // Tiempo
```

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

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$gcc -O2 -o pmm-secuencial-modificadoB pmm-secuencial-modificadoB.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$./pmm-secuencial-modificadoB 56
Tiempo (seg): 0.000404259

__________
Resultados:

Matriz B:
B[0][0]=0.093832 B[55][55]=0.559334

Matriz C:
C[0][0]=0.304918 C[55][55]=0.153229

Matriz A=B*C:
A[0][0]=12.805890 A[55][55]=15.209803
```

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$./pmm-secuencial 56
Tiempo (seg): 0.002367576

__________
Resultados:

Matriz B:
B[0][0]=0.801855 B[55][55]=0.088784

Matriz C:
C[0][0]=0.350823 C[55][55]=0.208710

Matriz A=B*C:
A[0][0]=19.324364 A[55][55]=12.433459
```

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$./pmm-secuencial-modificadoB 5
Tiempo (seg): 0.000000812

__________
Resultados:

Matriz B:
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000

Matriz C:
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000

Matriz A=B*C:
15.000000 30.000000 45.000000 60.000000 75.000000
15.000000 30.000000 45.000000 60.000000 75.000000
15.000000 30.000000 45.000000 60.000000 75.000000
15.000000 30.000000 45.000000 60.000000 75.000000
15.000000 30.000000 45.000000 60.000000 75.000000
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer1] 2021-06-04 viernes
$./pmm-secuencial 5
Tiempo (seg): 0.000002807

__________
Resultados:

Matriz B:
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000

Matriz C:
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000
1.000000 2.000000 3.000000 4.000000 5.000000

Matriz A=B*C:
15.000000 30.000000 45.000000 60.000000 75.000000
15.000000 30.000000 45.000000 60.000000 75.000000
15.000000 30.000000 45.000000 60.000000 75.000000
15.000000 30.000000 45.000000 60.000000 75.000000
15.000000 30.000000 45.000000 60.000000 75.000000
```

TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar	*Dos tiempos para cada modificación respectivamente*	*0,000031154-0,002367576*
Modificación A)	Desenrollado de bucle	0,000005345
Modificación B)	Cambio de iteradores en el for	0,000404259

COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:
Es claro que las modificaciones mejoran el tiempo. De entre ellas, la que más mejora el tiempo es la modificación 2, lo cual tiene sentido por la penalización que se aprecia en caché.

2. **(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`

figura1-original.c × figura1-modificado_A.c pmm-secuencial.c

ejer2 > figura1-original.c > main(int, char **)

```c
/**
 * @author Alberto Llamas Gonzale
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX 1001
// Vectores globales
struct s
{
    int a;
    int b;
} s[MAX];

int main(int argc, char **argv)
{
    if (argc < 3)
        printf("\nIntroduzca el numero de parametros correcto N,M\n");

    int N = atoi(argv[1]);
    int M = atoi(argv[2]);

    struct timespec cgt1, cgt2;
    double t; // para tiempos

    struct s s[N];
    int R[M];

    int i, ii, X1, X2;
    srand48(time(NULL));

    // Inicialización
    for (i = 0; i < N; i++)
    {
```

figura1-original.c × figura1-modificado_A.c pmm-secuencial.c

ejer2 > figura1-original.c > main(int, char **)

```c
    int i, ii, X1, X2;
    srand48(time(NULL));

    // Inicialización
    for (i = 0; i < N; i++)
    {
        if (N < 9 && M < 9){
            s[i].a = i;
            s[i].b = i;
        } else {
            s[i].a = drand48();
            s[i].b = drand48();
        }
    }

    // Tiempo
    clock_gettime(CLOCK_REALTIME, &cgt1);

    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;
    }

    // Tiempo
    clock_gettime(CLOCK_REALTIME, &cgt2);

```

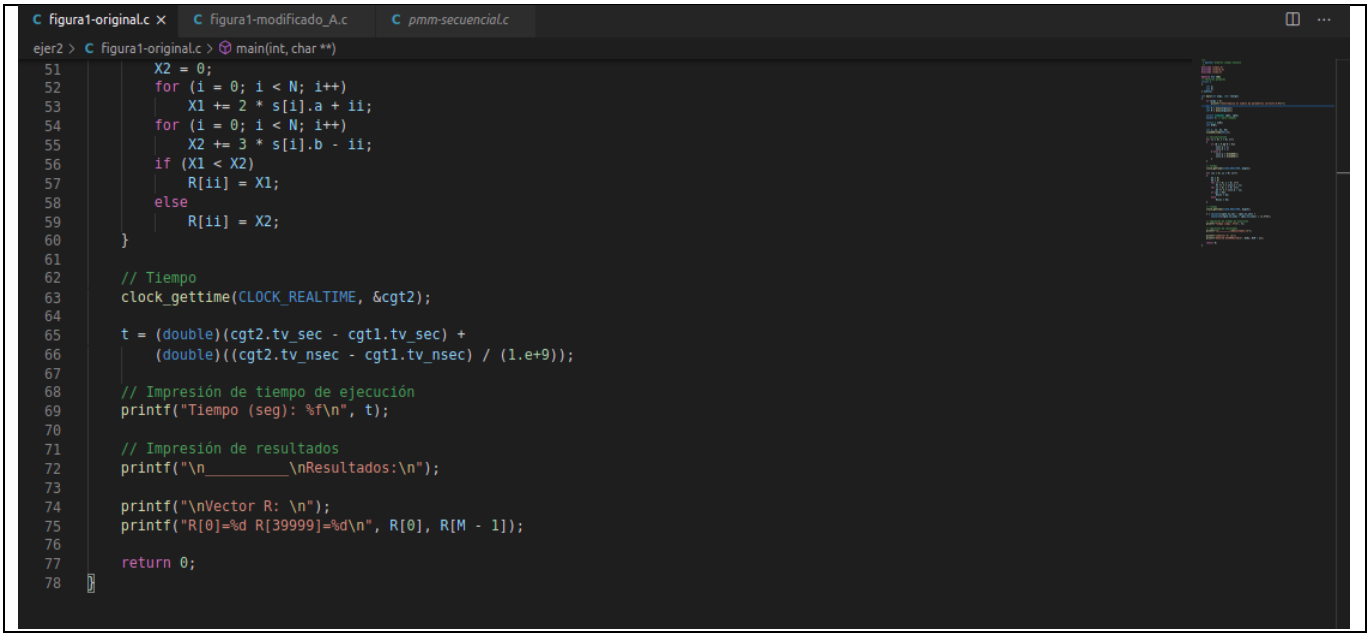


Figura 1 . Código C++ que suma dos vectores. `M` y `N` deben ser parámetros de entrada al programa, usar valores mayores que 1000 en la evaluación.

```
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-: recorremos los vectores a y b en el mismo bucle. Por cómo están colocados en el struct, tenemos en memoria {a0, b0, a1, b1, ..., an, bn}. Con el código original estamos trayendo de caché todos los bloques del struct dos veces, para recorrer los vectores a y b, respectivamente. Sin embargo, con esta modificación sólo traemos el struct completo de caché una sola vez, porque vamos recorriendo la memoria de forma contigua, sin saltos.

Modificación B) –explicación-: añadimos a la modificación A un desenrollado del for en 4 como en la modificación A del ejercicio 1

CÓDIGOS FUENTE MODIFICACIONES

A) **Captura** figura1-`modificado_A.c`

```
52      for (ii = 0; ii < M; ii++)
53      {
54          X1 = 0;
55          X2 = 0;
56          //Modificacion:
57          for (i = 0; i < N; i++){
58              X1 += 2 * s[i].a + ii;
59              X2 += 3 * s[i].b - ii;
60          }
61          if (X1 < X2)
62              R[ii] = X1;
63          else
64              R[ii] = X2;
65      }
66
```

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

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer2] 2021-06-04 viernes
$./figura1-original 5000 40000
Tiempo (seg): 0.344647

_________
Resultados:

Vector R:
R[0]=0 R[39999]=-199995000
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer2] 2021-06-04 viernes
$gcc -O2 -o figura1-modificado_A figura1-modificado_A.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer2] 2021-06-04 viernes
$./figura1-modificado_A 5000 40000
Tiempo (seg): 0.256312

_________
Resultados:

Vector R:
R[0]=0 R[39999]=-199995000
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer2] 2021-06-04 viernes
$
```

B) **Captura** figura1-`modificado_B.c`

```
50      int mod = N % 4;
51
52      for (ii = 0; ii < M; ii++)
53      {
54          X1 = 0;
55          X2 = 0;
56          for (i = 0; i < N - mod; i+=4){
57
58              X1 += 2 * s[i].a + ii;
59              X1 += 2 * s[i + 1].a + ii;
60              X1 += 2 * s[i + 2].a + ii;
61              X1 += 2 * s[i + 3].a + ii;
62
63              X2 += 3 * s[i].b - ii;
64              X2 += 3 * s[i + 1].b - ii;
65              X2 += 3 * s[i + 2].b - ii;
66              X2 += 3 * s[i + 3].b - ii;
67
68              if (N - mod + 2 < N)
69              {
70                  X1 += 2 * s[i].a + ii;
71                  X1 += 2 * s[i + 1].a + ii;
72                  X1 += 2 * s[i + 2].a + ii;
73                  X2 += 3 * s[i].b - ii;
74                  X2 += 3 * s[i + 1].b - ii;
75                  X2 += 3 * s[i + 2].b - ii;
```

```
76              }
77              else if (N - mod + 1 < N)
78              {
79                  X1 += 2 * s[i].a + ii;
80                  X1 += 2 * s[i + 1].a + ii;
81                  X2 += 3 * s[i].b - ii;
82                  X2 += 3 * s[i + 1].b - ii;
83              }
84              else if (N - mod < N)
85              {
86                  X1 += 2 * s[i].a + ii;
87                  X2 += 3 * s[i].b - ii;
88              }
89          }
90          if (X1 < X2)
91              R[ii] = X1;
92          else
93              R[ii] = X2;
94      }
95
```

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

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer2] 2021-06-04 viernes
$gcc -O2 -o figura1-modificado_B figura1-modificado_B.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer2] 2021-06-04 viernes
$./figura1-modificado_B 5000 40000
Tiempo (seg): 0.215002

__________
Resultados:

Vector R:
R[0]=0 R[39999]=-199995000
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer2] 2021-06-04 viernes
$./figura1-modificado_A 5000 40000
Tiempo (seg): 0.256414

__________
Resultados:

Vector R:
R[0]=0 R[39999]=-199995000
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer2] 2021-06-04 viernes
$./figura1-original 5000 40000
Tiempo (seg): 0.343076

__________
Resultados:

Vector R:
R[0]=0 R[39999]=-199995000
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer2] 2021-06-04 viernes
$
```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		*0,343076*
Modificación A)	a y b en mismo bucle	0,256414
Modificación B)	desenrollado del bucle	0,215002

COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:
Como vemos, el desenrollado de bucles con la unión en un mismo for es la mejor opción.

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` (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

```
for (i=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 acarrean. 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`

```c
/**
 * @author Alberto Llamas Gonzalez
 * */

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

int main(int argc, char **argv)
{
    if (argc < 2)
    {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }

    int N = atoi(argv[1]);

    struct timespec ini, fin;
    double tiempo;

    // Creación de vector y matriz
    int i, a = 47;
    int x[N], y[N];

    // Inicialización
    for (i = 1; i <= N; i++)
    {
        x[i] = i;
        y[i] = i;
    }

    // Cálculo del resultado
    clock_gettime(CLOCK_REALTIME, &ini);

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

    clock_gettime(CLOCK_REALTIME, &fin);

    tiempo = (double)(fin.tv_sec - ini.tv_sec) + (double)((fin.tv_nsec - ini.tv_nsec) / (1.e+9));

    // Resultados
    printf("Tiempo(seg): %f\ny[0]=%d, y[N-1]=%d \n", tiempo, y[0], y[N - 1]);
}
```

Tiempos ejec. Longitud vectores=1000000	-O0	-Os	-O2	-O3
	De 0,25 a 10 seg. aquí	0,001343	0,001322	0,001268

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

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer3] 2021-06-04 viernes
$gcc -O2 -o daxpy daxpy.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer3] 2021-06-04 viernes
$./daxpy 1000000
Tiempo(seg): 0.001322
y[0]=0, y[N-1]=47999952
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer3] 2021-06-04 viernes
$gcc -Os -o daxpy daxpy.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer3] 2021-06-04 viernes
$./daxpy 1000000
Tiempo(seg): 0.001343
y[0]=0, y[N-1]=47999952
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer3] 2021-06-04 viernes
$gcc -O3 -o daxpy daxpy.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer3] 2021-06-04 viernes
$./daxpy 1000000
Tiempo(seg): 0.001268
y[0]=0, y[N-1]=47999952
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

-O0: no tenemos optimización alguna.
-O2: vemos que el número de instrucciones es parecido, pero observamos cambios considerables en las instrucciones utilizadas, que serán más eficientes.
-O3: el código ensamblador es mucho más complejo y difícil de entender, aumenta también el número de subrutinas llamadas.
-Os: disminuimos el tamaño del ejecutable, el código se parece mucho al de -O1 y -O2, pero podemos ver en el tamaño de archivo que este es menor.

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)
Nota: He cambiado el formato de las tablas para que se vean bien las imágenes.

daxpyO0.s

```
109    jmp .L3
110  .L4:
111    movq  -128(%rbp), %rax
112    movl  -144(%rbp), %edx
113    movslq  %edx, %rdx
114    movl  -144(%rbp), %ecx
115    movl  %ecx, (%rax,%rdx,4)
116    movq  -112(%rbp), %rax
117    movl  -144(%rbp), %edx
118    movslq  %edx, %rdx
119    movl  -144(%rbp), %ecx
120    movl  %ecx, (%rax,%rdx,4)
121    addl  $1, -144(%rbp)
122  .L3:
123    movl  -144(%rbp), %eax
124    cmpl  -148(%rbp), %eax
125    jle .L4
```

daxpyOs.s

```
 52  .L3:
 53    cmpl  %eax, %ebx
 54    jl  .L10
 55    movl  %eax, (%r14,%rax,4)
 56    movl  %eax, 0(%r13,%rax,4)
 57    incq  %rax
 58    jmp .L3
```

daxpyO2.s

```
 60  .L4:
 61    movl  %ebx, (%r15,%rbx,4)
 62    movl  %ebx, (%r14,%rbx,4)
 63    addq  $1, %rbx
 64    cmpq  %rax, %rbx
 65    jne .L4
```

daxpy03.s

```
.L15:
  leaq  -80(%rbp), %rsi
  xorl  %edi, %edi
  movslq  %r13d, %r13
  call  clock_gettime@PLT
  movq  -72(%rbp), %rax
  pxor  %xmm0, %xmm0
  subq  -88(%rbp), %rax
  pxor  %xmm1, %xmm1
  movl  (%rbx,%r13,4), %ecx
  movl  0(,%r12,4), %edx
  leaq  .LC4(%rip), %rsi
  movl  $1, %edi
  cvtsi2sdq %rax, %xmm0
  movq  -80(%rbp), %rax
  subq  -96(%rbp), %rax
  cvtsi2sdq %rax, %xmm1
  movl  $1, %eax
  divsd .LC3(%rip), %xmm0
  addsd %xmm1, %xmm0
  call  __printf_chk@PLT
  xorl  %eax, %eax
  movq  -56(%rbp), %rbx
  xorq  %fs:40, %rbx
  jne .L34
  leaq  -40(%rbp), %rsp
  popq  %rbx
  popq  %r12
  popq  %r13
  popq  %r14
  popq  %r15
  popq  %rbp
  .cfi_remember_state
  .cfi_def_cfa 7, 8
  ret
.L19:
  .cfi_restore_state
  movl  $2, -104(%rbp)
  jmp .L5
.L23:
  movl  $2, %esi
  jmp .L13
.L3:
  leaq  -96(%rbp), %rsi
  xorl  %edi, %edi
  leal  -1(%r14), %r13d
  call  clock_gettime@PLT
  jmp .L15
```

4. **(a)** Paralizar 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:

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer4] 2021-06-04 viernes
$gcc -O2 -fopenmp -o pmm-paralelo pmm-paralelo.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer4] 2021-06-04 viernes
$./pmm-paralelo 800
Tiempo (seg): 0.348659183

__________
Resultados:

Matriz B:
B[0][0]=0.978918 B[799][799]=0.170992

Matriz C:
C[0][0]=0.267670 C[799][799]=0.774878

Matriz A=B*C:
A[0][0]=197.423879 A[799][799]=199.641848
```

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer4] 2021-06-04 viernes
$./pmm-paralelo 1000
Tiempo (seg): 0.680940225

__________
Resultados:

Matriz B:
B[0][0]=0.945337 B[999][999]=0.484302

Matriz C:
C[0][0]=0.904393 C[999][999]=0.657849

Matriz A=B*C:
A[0][0]=244.427919 A[999][999]=250.892041

[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer4] 2021-06-04 viernes
$./pmm-paralelo 1200
Tiempo (seg): 1.181779566

__________
Resultados:

Matriz B:
B[0][0]=0.299348 B[1199][1199]=0.998181

Matriz C:
C[0][0]=0.427346 C[1199][1199]=0.270577

Matriz A=B*C:
A[0][0]=291.242407 A[1199][1199]=301.811363
```

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer4] 2021-06-04 viernes
$gcc -O2 -o pmm-secuencial pmm-paralelo.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer4] 2021-06-04 viernes
$./pmm-secuencial 800
Tiempo (seg): 0.584785129

__________
Resultados:

Matriz B:
B[0][0]=0.839464 B[799][799]=0.380681

Matriz C:
C[0][0]=0.656869 C[799][799]=0.903980

Matriz A=B*C:
A[0][0]=203.127451 A[799][799]=214.012322
```

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer4] 2021-06-04 viernes
$./pmm-secuencial 1000
Tiempo (seg): 1.137294283

_________
Resultados:

Matriz B:
B[0][0]=0.193475 B[999][999]=0.475885

Matriz C:
C[0][0]=0.179821 C[999][999]=0.193966

Matriz A=B*C:
A[0][0]=247.927030 A[999][999]=252.875365

[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp4/ejer4] 2021-06-04 viernes
$./pmm-secuencial 1200
Tiempo (seg): 1.941785111

_________
Resultados:

Matriz B:
B[0][0]=0.547487 B[1199][1199]=0.311931

Matriz C:
C[0][0]=0.702774 C[1199][1199]=0.486120

Matriz A=B*C:
A[0][0]=298.483705 A[1199][1199]=309.393959
```

CAPTURA CÓDIGO FUENTE: `pmm-paralelo.c`

```
69      // Cálculo de A=B*C
70      #pragma omp parallel for
71      for (i = 0; i < N; i++)
72          for (k = 0; k < N; k++)
73          {
74              A[i][j] = 0;
75              for (j = 0; j < N; j++)
76                  A[i][j] = A[i][j] + B[i][k] * C[k][j];
77          }
78
```

La única modificación realizada en el código es la sentencia añadida #pragma omp parallel for, que se ve en la imagen.

(b) RESPUESTA

Como podemos observar en la captura de la ejecución, podemos calcular tres ganancias, ya que hemos probado tres tamaños de matriz.

G1 = 0,584785129 / 0,348659183 = 1,6772400026

G2 = 1,137294283 / 0,680940225 = 1,6701822587

G3 = 1,94178511 / 1,181776566 = 1,6431067986

Como vemos, la ganancia oscila entre 1,64 y 1,67.