2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 5. Optimización de código

Estudiante (nombre y apellidos): Daniel Alconchel Vázquez
Grupo de prácticas y profesor de prácticas: 1
Fecha de entrega:
Fecha evaluación en clase:

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

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo):

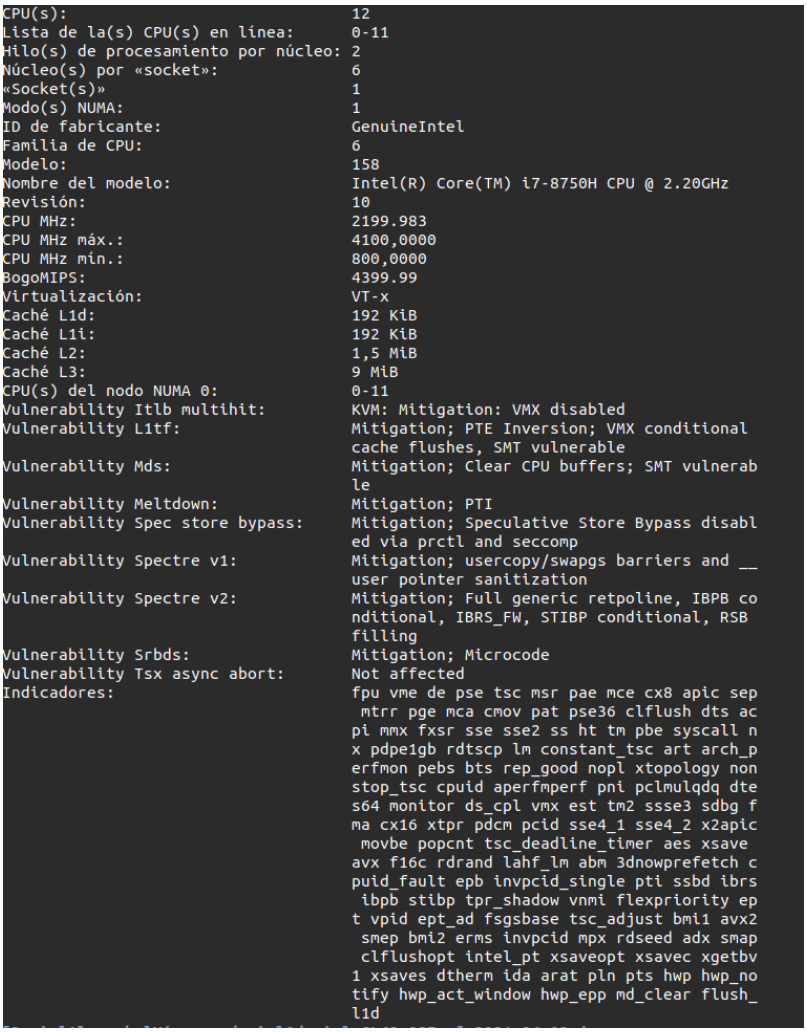
Intel i7-8750H (12) @ 2.200GHz

Sistema operativo utilizado: *Ubuntu 20.04.2 LTS x86_64*

Versión de gcc utilizada: *(respuesta)*

gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0

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



```
CPU(s):                          12
Lista de la(s) CPU(s) en línea:  0-11
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:          6
«Socket(s)»                      1
Modo(s) NUMA:                    1
ID de fabricante:                GenuineIntel
Familia de CPU:                  6
Modelo:                          158
Nombre del modelo:               Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
Revisión:                        10
CPU MHz:                         2199.983
CPU MHz máx.:                    4100,0000
CPU MHz mín.:                    800,0000
BogoMIPS:                        4399.99
Virtualización:                  VT-x
Caché L1d:                       192 KiB
Caché L1i:                       192 KiB
Caché L2:                        1,5 MiB
Caché L3:                        9 MiB
CPU(s) del nodo NUMA 0:          0-11
Vulnerability Itlb multihit:     KVM: Mitigation: VMX disabled
Vulnerability L1tf:              Mitigation; PTE Inversion; VMX conditional
                                 cache flushes, SMT vulnerable
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT vulnerab
                                 le
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabl
                                 ed via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __
                                 user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, IBPB co
                                 nditional, IBRS_FW, STIBP conditional, RSB
                                 filling
Vulnerability Srbds:             Mitigation; Microcode
Vulnerability Tsx async abort:   Not affected
Indicadores:                     fpu vme de pse tsc msr pae mce cx8 apic sep
                                  mtrr pge mca cmov pat pse36 clflush dts ac
                                 pi mmx fxsr sse sse2 ss ht tm pbe syscall n
                                 x pdpe1gb rdtscp lm constant_tsc art arch_p
                                 erfmon pebs bts rep_good nopl xtopology non
                                 stop_tsc cpuid aperfmperf pni pclmulqdq dte
                                 s64 monitor ds_cpl vmx est tm2 ssse3 sdbg f
                                 ma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic
                                  movbe popcnt tsc_deadline_timer aes xsave
                                 avx f16c rdrand lahf_lm abm 3dnowprefetch c
                                 puid_fault epb invpcid_single pti ssbd ibrs
                                  ibpb stibp tpr_shadow vnmi flexpriority ep
                                 t vpid ept_ad fsgsbase tsc_adjust bmi1 avx2
                                  smep bmi2 erms invpcid mpx rdseed adx smap
                                  clflushopt intel_pt xsaveopt xsavec xgetbv
                                 1 xsaves dtherm ida arat pln pts hwp hwp_no
                                 tify hwp_act_window hwp_epp 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

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

// Vectores globales
#define MAX 10000
double A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];


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]);
        if ( N > MAX )
                N = MAX;

        int i, j, k;
        double suma;

        // Inicialización matrices B, C
        if ( N<8 ){
    for ( i = 0; i < N; i++ )
      for ( j = 0; j < N; j++ ) {
        B[i][j]=j+1;
        C[i][j]=j+1;
      }
  }else{
    for ( i = 0; i < N; i++ )
      for ( j = 0; j < N; j++ ) {
        B[i][j]=drand48();
        C[i][j]=drand48();
      }
  }

        // 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);
```

NORMAL pmm-secuencial.c

(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-: Desenrollando bucles en la variable j, por ejemplo, 4 operaciones más por bucle. Permite recudir el número de saltos, aumenta la oportunidad de encontrar instrucciones independientes y facilita el insertar instrucciones para ocultar las latencias.

Modificación B) –explicación-: Localidad de accesos, intercambiamos las variables k y j para aprovechar la localidad ya que cambiaremos el acceso a los datos según los almacena el compilador.

Modificación C) –explicación-: Ambas. Tiene sentido que si cada una mejora el tiempo de ejecución, ambas lo mejorarán.

CÓDIGOS FUENTE MODIFICACIONES

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

```
for(int i = 0; i<N ; i++)
  for(int j = 0; j<N; j+=4)
    for(int k = 0; k<N; k++){
      A[i][j] += B[i][k] * C[k][j];
      A[i][j+1] += B[i][k] * C[k][j+1];
      A[i][j+2] += B[i][k] * C[k][j+2];
      A[i][j+3] += B[i][k] * C[k][j+3];
    }
```

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

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer1] 2021-06-03 jueves
$ gcc -O2 pmm-secuencial-modificadoA.c -o pmmA
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer1] 2021-06-03 jueves
$ ./pmmA 1200
Tiempo (seg): 3.053653432

__________
Resultados:

Matriz B:
B[0][0]=0.000000 B[1199][1199]=0.020978

Matriz C:
C[0][0]=0.000985 C[1199][1199]=0.014644

Matriz A=B*C:
A[0][0]=311.972803 A[1199][1199]=309.716224

[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer1] 2021-06-03 jueves
$
```

B) Captura de pmm-secuencial-modificado_B.c

```
clock_gettime(CLOCK_REALTIME,&cgt1);

for(int i = 0; i<N ; i++)
  for(int k= 0; k<N; k++)
    for(int j = 0; j<N; j++)
      A[i][j] += B[i][k] * C[k][j];


clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer1] 2021-06-03 jueves
$ gcc -O2 pmm-secuencial-modificadoB.c -o pmmB
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer1] 2021-06-03 jueves
$ ./pmmB 1200
Tiempo (seg): 1.195962831

__________
Resultados:

Matriz B:
B[0][0]=0.000000 B[1199][1199]=0.020978

Matriz C:
C[0][0]=0.000985 C[1199][1199]=0.014644

Matriz A=B*C:
A[0][0]=311.972803 A[1199][1199]=309.716224

[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer1] 2021-06-03 jueves
```

C) Captura de pmm-secuencial-modificado_C.c

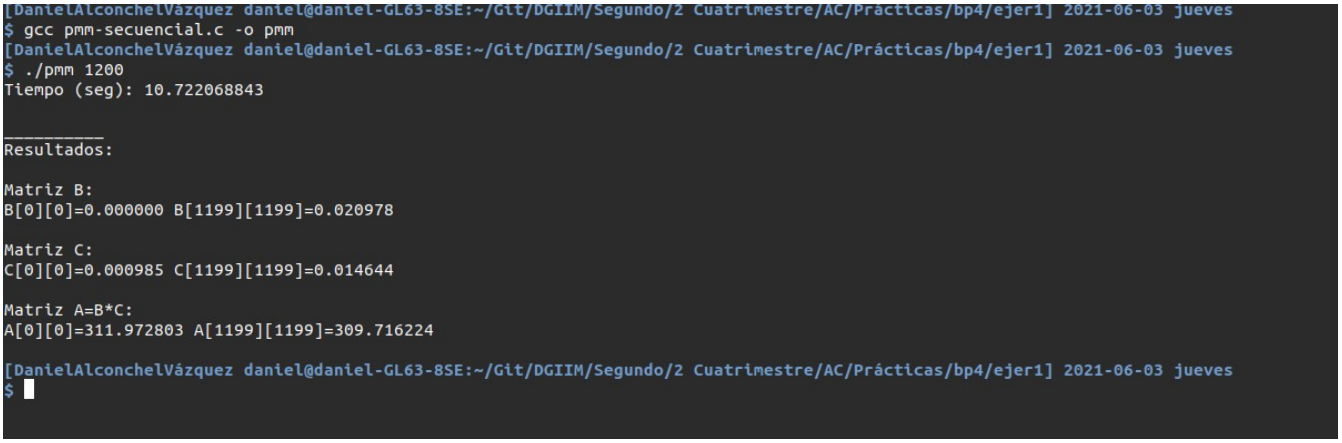
```
clock_gettime(CLOCK_REALTIME,&cgt1);

for(int i = 0; i<N ; i++)
  for(int k = 0; k<N; k++)
    for(int j = 0; j<N; j+=4){
      A[i][j] += B[i][k] * C[k][j];
      A[i][j+1] += B[i][k] * C[k][j+1];
      A[i][j+2] += B[i][k] * C[k][j+2];
      A[i][j+3] += B[i][k] * C[k][j+3];
    }
```

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

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer1] 2021-06-03 jueves
$ gcc -O2 pmm-secuencial-modificadoC.c -o pmmC
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer1] 2021-06-03 jueves
$ ./pmmC 1200
Tiempo (seg): 1.115805556

__________
Resultados:

Matriz B:
B[0][0]=0.000000 B[1199][1199]=0.020978

Matriz C:
C[0][0]=0.000985 C[1199][1199]=0.014644

Matriz A=B*C:
A[0][0]=311.972803 A[1199][1199]=309.716224

[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer1] 2021-06-03 jueves
```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		10,722068843
Modificación A)	Desenrollado de bucles en la variable j	3.053653432
Modificación B)	Cambio de lugar la variable j y k para aprovechar la localidad de acceso	1.195962831
Modificación C)	Ambas	1.115805556



COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:
La justificación de las mejoras está incluida cuando describo que mejoras voy a usar.
Tal y como esperábamos, se produce una mejora en el tiempo de ejecución. Podemos destacar la diferencia entre el desenrollado de bucles y el cambiar de lugar la variable j, ya que hay una mejora de 2 segundos.

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`

```c
int main(int argc, char **argv){
        struct timespec cgt1,cgt2; double t;
        int i, ii, X1, X2;
    if (argc < 3){
        printf("ERROR EN LOS PARAMETROS\n");
        exit(EXIT_FAILURE);
    }

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

    struct bp4 s[N];
    int R[M];

        if (N>8 || M>8){
        for ( i = 0; i < N; i++ ) {
                s[i].a = drand48();
                s[i].b = drand48();
            }
    }else {
        for ( i = 0; i < N; i++ ) {
            s[i].a = i;
            s[i].b = i;
            }
    }
        // 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);
        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): %f\n", t);
        // Impresión de resultados
        printf("\n_________\nResultados:\n");
        printf("\nVector R: \n");
        printf("R[0]=%d R[39999]=%d\n", R[0], R[M-1]);

        return 0;
}
```

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-: Unificación de ambos bucles, uniremos los bucles for para evitar recorrer ambos las mismas veces.

Modificación B) –explicación-: Localidad de los accesos, ya que la forma en la que se declaran los arrays determina como se almacenan en memoria, por lo tanto, almacenaremos todas las a y las b juntas, en vez de un array con ambas variables.

Modificación C) –explicación-: Junto con la unión de ambos bucles, podremos usar, al igual que en el primer ejercicio, el desenrollado de bucles.

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c

```
clock_gettime(CLOCK_REALTIME,&cgt1);
for(ii = 0; ii<M; ii++){
    X1 = 0; X2 = 0;
    for(int i=0; i<N; i++){
        X1 += 2*s[i].a+ii;
        X2 += 3*s[i].b-ii;
    }
    if(X1 < X2) R[ii]=X1; else R[ii] = X2;
}
clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$ gcc -O2 figura1-original-modificadoA.c -o figA
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$ ./figA 5000 40000
Tiempo (seg): 0.128848

__________
Resultados:

Vector R:
R[0]=0 R[39999]=-199995000
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$
```

B) Captura figura1-modificado_B.c

```
clock_gettime(CLOCK_REALTIME,&cgt1);
for(ii = 0; ii<M; ii++){
    X1 = 0; X2 = 0;
    for(int i=0; i<N; i++) X1 += 2*s.a[i]+ii;
    for(int i=0; i<N; i++) X2 += 3*s.b[i]-ii;

    if(X1 < X2) R[ii]=X1; else R[ii] = X2;
}
clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$ gcc figura1-original-modificadoB.c -o figB
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$ ./figB 5000 40000
Tiempo (seg): 0.952118

__________
Resultados:

Vector R:
R[0]=0 R[39999]=-199995000
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$
```

C) Captura figura1-modificado_C.c

```
clock_gettime(CLOCK_REALTIME,&cgt1);
for(ii = 0; ii<M; ii++){
    X1 = 0; X2 = 0;
    for(int i=0; i<N; i+=4){
        X1 += 2*s[i].a+ii;
        X2 += 3*s[i].b-ii;
        X1 += 2*s[i+1].a+ii;
        X2 += 3*s[i+1].b-ii;
        X1 += 2*s[i+2].a+ii;
        X2 += 3*s[i+2].b-ii;
        X1 += 2*s[i+3].a+ii;
        X2 += 3*s[i+3].b-ii;
    }

    if(X1 < X2) R[ii]=X1; else R[ii] = X2;
}
clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$ gcc figura1-original-modificadoC.c -o figC
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$ ./figC 5000 40000
Tiempo (seg): 0.386977


_________
Resultados:

Vector R:
R[0]=0 R[39999]=-199995000
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	--	0,741744
Modificación A)	Unificación del bucle for	0,128848
Modificación B)	Cambio del Struct	0,952118
Modificación C)	Unión del bucle for + desenrollado del bucle	0,386977

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$ gcc figura1-original.c -o figorig
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
$ ./figorig 5000 40000
Tiempo (seg): 0.741744


_________
Resultados:

Vector R:
R[0]=0 R[39999]=-199995000
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer2] 2021-06-03 jueves
```

COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:
Nuevamente, la justificación ha sido añadida junto a la indicación de que modificaciones se han hecho. Además, los cambios son los que esperábamos. Un resultado extraño es que el struct apenas genera mejoría y en algunas ejecuciones, como en este caso, tarda hasta más que el código original.

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
#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+

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

Tiempos ejec. Longitud vectores=XXXX	**-O0**	**-Os**	**-O2**	**-O3**
	0,009293	0,002907	0,002461	0,000363

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

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ gcc -O0 daxpy.c -o daxpyO0
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ gcc -O2 daxpy.c -o daxpyO2
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ gcc -O3 daxpy.c -o daxpyO3
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ gcc -Os daxpy.c -o daxpyOs
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ gcc -S -O0 daxpy.c -o daxpyO0.s
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ gcc -S -O2 daxpy.c -o daxpyO2.s
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ gcc -S -O3 daxpy.c -o daxpyO3.s
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ gcc -S -Os daxpy.c -o daxpyOs.s
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$
```

```
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ ./daxpyO0 1000000
Tiempo(seg): 0.009293
y[0]=0, y[N-1]=47999952
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ ./daxpyO2 1000000
Tiempo(seg): 0.002461
y[0]=0, y[N-1]=47999952
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ ./daxpyO3 1000000
Tiempo(seg): 0.000363
y[0]=0, y[N-1]=47999952
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$ ./daxpyOs 1000000
Tiempo(seg): 0.002907
y[0]=0, y[N-1]=47999952
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp4/ejer3] 2021-06-03 jueves
$
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:
O0: No tenemos optimización alguna.
O2: Vemos que el número de instrucciones es reducido y observamos cambios positivos considerables en la eficiencia de las instrucciones que se usen..
O3: El código ensamblador es mucho más complejo y difícil de entender, aumenta también el número de subrutinas llamadas y se usan instrucciones vectoriales.
Os: Disminuimos el tamaño del ejecutable y el tamaño de archivo también 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)

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

.L9:
movl -144(%rbp), %eax
cmpl -148(%rbp), %eax
jle .L10
leaq -96(%rbp), %rax
movq %rax, %rsi
movl $0, %edi
call clock_gettime@PLT
movl $1, -144(%rbp)
jmp .L11
.L12:
movq -128(%rbp), %rax
movl -144(%rbp), %edx
movslq %edx, %rdx
movl (%rax,%rdx,4), %eax
imull -140(%rbp), %eax
movl %eax, %ecx
movq -112(%rbp), %rax
movl -144(%rbp), %edx
movslq %edx, %rdx
movl (%rax,%rdx,4), %eax
addl %eax, %ecx
movq -112(%rbp), %rax
movl -144(%rbp), %edx
movslq %edx, %rdx
movl %ecx, (%rax,%rdx,4)
addl $1, -144(%rbp)
.L11:
movl -144(%rbp), %eax
cmpl -148(%rbp), %eax
jle .L12
leaq -80(%rbp), %rax
movq %rax, %rsi
movl $0, %edi
call clock_gettime@PLT
	.L22:
	leaq -72(%rbp), %rsi
	xorl %edi, %edi
	call clock_gettime@PLT
	xorl %eax, %eax
	.L11:
	incq %rax
	cmpl %eax, %ebx
	jl .L23
	imull $47, (%r14,%rax,4), %edx
	addl %edx, 0(%r13,%rax,4)
	jmp .L11
	.L23:
	xorl %edi, %edi
	leaq -56(%rbp), %rsi
	decl %ebx
	call clock_gettime@PLT
		call clock_gettime@PLT
		movl $1, %eax
		.p2align 4,,10
		.p2align 3
		.L12:
		imull $47, (%r15,%rax,4), %edx
		addl %edx, (%rbx,%rax,4)
		addq $1, %rax
		cmpq %r14, %rax
		jne .L12
		.L13:
		xorl %edi, %edi
		leaq -80(%rbp), %rsi
		movslq %r12d, %r12
		call clock_gettime@PLT
			.L9:
			leaq -96(%rbp), %rsi
			xorl %edi, %edi
			leal -1(%r15), %r12d
			call clock_gettime@PLT
			jmp .L19
			.L10:
			movl $1, 4(,%rcx,4)
			movl $2, %eax
			movl $1, 4(,%r13,4)
			cmpl $1, %r15d
			jne .L20
			leaq -96(%rbp), %rsi
			xorl %edi, %edi
			call clock_gettime@PLT

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.

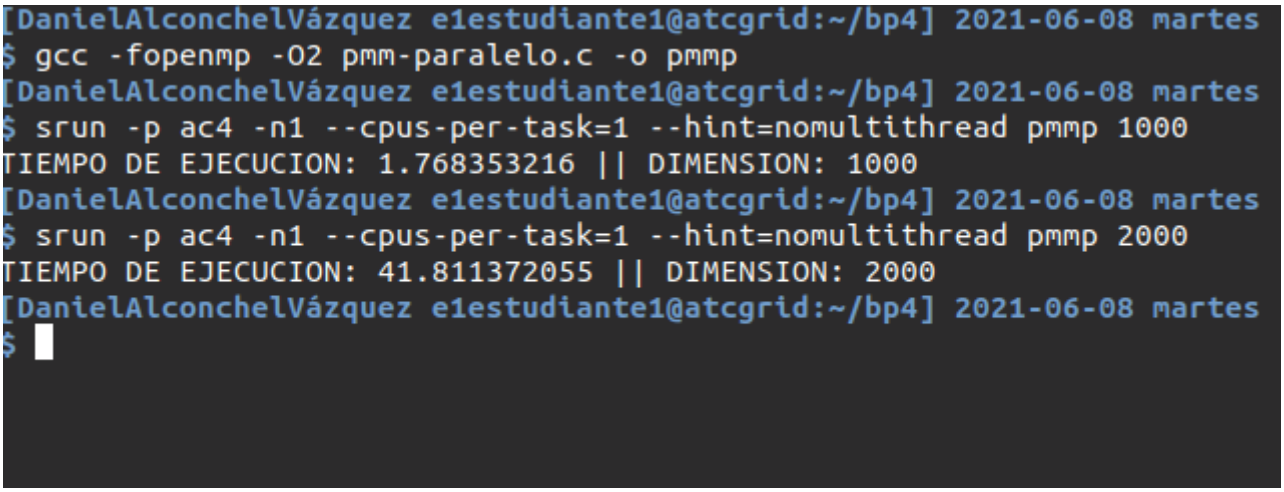
   (a) MULTIPLICACIÓN DE MATRICES PARALELO:

   CAPTURA CÓDIGO FUENTE: `pmm-paralelo.c`

```c
#define VAR_DYNAMIC

int main(int argc, char **argv){
  struct timespec cgt1, cgt2;
  double ncgt;
  int k=0;

  if (argc < 2)
  {
    printf("FALTA LA NENSION");
    exit(-1);
  }

  int N = atoi(argv[1]);

  double **m1, **m2, **mm;

  m1 = (double **)malloc(N * sizeof(double *));
  m2 = (double **)malloc(N * sizeof(double *));
  mm = (double **)malloc(N * sizeof(double *));

  for (int i = 0; i < N; i++){
    m1[i] = (double *)malloc(N * sizeof(double));
    m2[i] = (double *)malloc(N * sizeof(double));
    mm[i] = (double *)malloc(N * sizeof(double));
  }

  double t_inicial, t_final;
  int i = 0, j = 0;


  for (i = 0; i < N; i++){
    for (j = 0; j < N; j++){
      m1[i][j] = drand48();
      m2[i][j] = drand48();
      mm[i][j] = 0;
    }
  }

  t_inicial = omp_get_wtime();

  #pragma omp parallel for private(j,k)
      for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
          for (int k = 0; k < N; k++)
            mm[i][j] += m1[i][k] * m2[k][j];
```

(b) RESPUESTA

```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp4] 2021-06-08 martes
$ ls
pmm  pmm-paralelo.c  pmms  pmm-secuencial.c
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp4] 2021-06-08 martes
$ gcc -fopenmp -O2 pmm-secuencial.c -o pmms
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp4] 2021-06-08 martes
$ srun -p ac4 -n1 --cpus-per-task=1 --hint=nomultithread pmms 1000
Tiempo (seg): 1.664338759


_________
Resultados:

Matriz B:
B[0][0]=0.000000 B[999][999]=0.677952

Matriz C:
C[0][0]=0.000985 C[999][999]=0.221306

Matriz A=B*C:
A[0][0]=261.400093 A[999][999]=256.234227

[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp4] 2021-06-08 martes
$ srun -p ac4 -n1 --cpus-per-task=1 --hint=nomultithread pmms 2000
Tiempo (seg): 13.970224730


_________
Resultados:

Matriz B:
B[0][0]=0.000000 B[1999][1999]=0.469179

Matriz C:
C[0][0]=0.000985 C[1999][1999]=0.272035

Matriz A=B*C:
A[0][0]=492.399304 A[1999][1999]=512.803629
```


[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp4] 2021-06-08 martes
$ gcc -fopenmp -O2 pmm-paralelo.c -o pmmp
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp4] 2021-06-08 martes
$ srun -p ac4 -n1 --cpus-per-task=1 --hint=nomultithread pmmp 1000
TIEMPO DE EJECUCION: 1.768353216 || DIMENSION: 1000
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp4] 2021-06-08 martes
$ srun -p ac4 -n1 --cpus-per-task=1 --hint=nomultithread pmmp 2000
TIEMPO DE EJECUCION: 41.811372055 || DIMENSION: 2000
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp4] 2021-06-08 martes
$