2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 5. Optimización de código

Estudiante (nombre y apellidos): Juan Carlos Ruiz Fernández
Grupo de prácticas y profesor de prácticas: B3 Mancia Anguita
Fecha de entrega: 9 de junio de 2021
Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): ***Intel Core i7 7700k 4.2GHz***

Sistema operativo utilizado: ***Windows 10 pro x64***

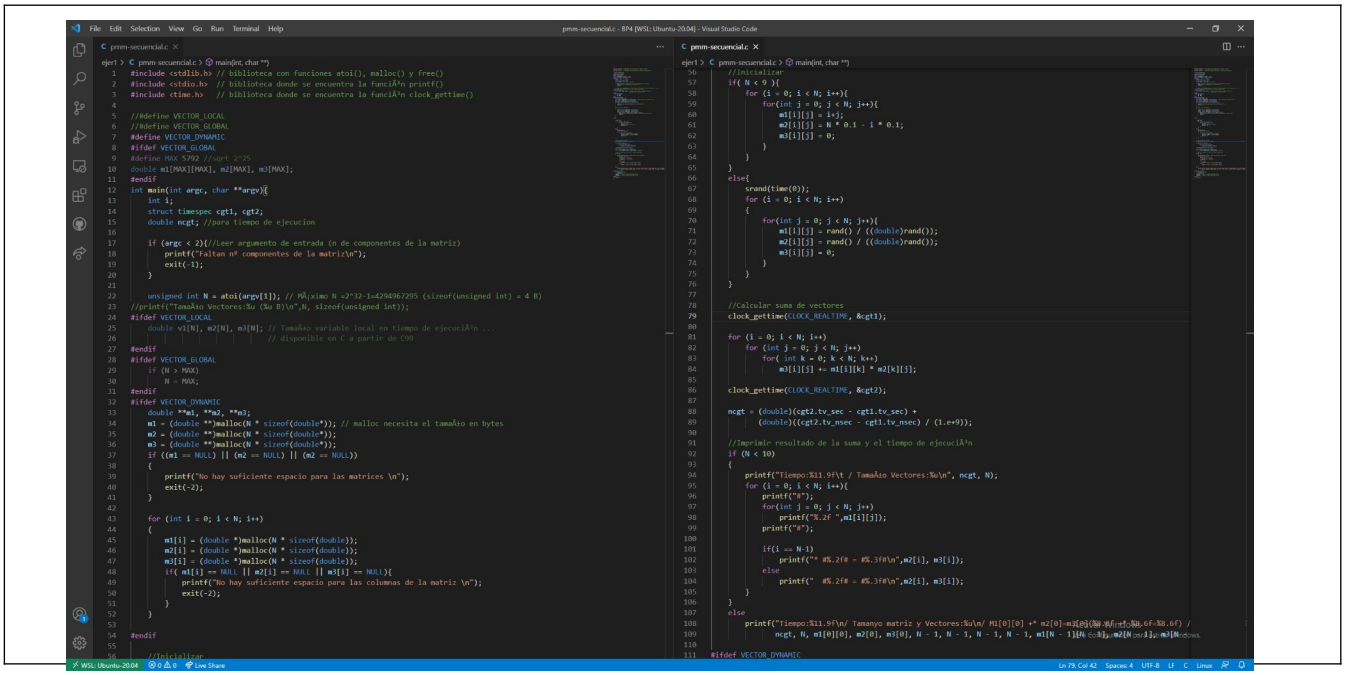
Versión de gcc utilizada: ***gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)***

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

```
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
Address sizes:           36 bits physical, 48 bits virtual
CPU(s):                  8
On-line CPU(s) list:     0-7
Thread(s) per core:      2
Core(s) per socket:      4
Socket(s):               1
Vendor ID:               GenuineIntel
CPU family:              6
Model:                   158
Model name:              Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
Stepping:                9
CPU MHz:                 4201.000
CPU max MHz:             4201.0000
BogoMIPS:                8402.00
Hypervisor vendor:       Windows Subsystem for Linux
```

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()`).

   CAPTURA CÓDIGO FUENTE: `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.

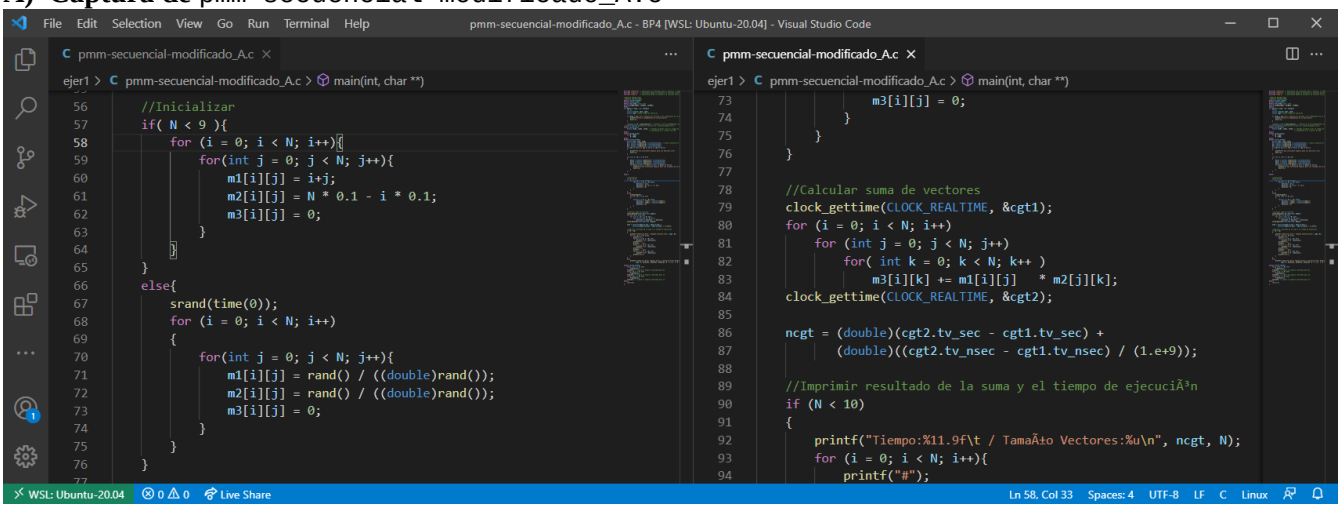
MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación-: He intercambiado las variables del bucle j y k para hacer menor numero de acceso a memoria

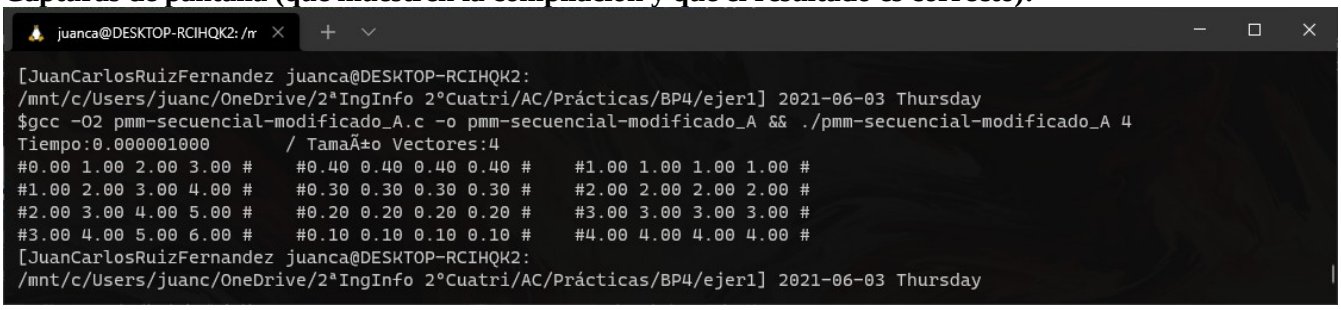
Modificación B) –explicación-: He desenrollado los bucles en 4 y comprobado que N sea múltiplo de 4

CÓDIGOS FUENTE MODIFICACIONES

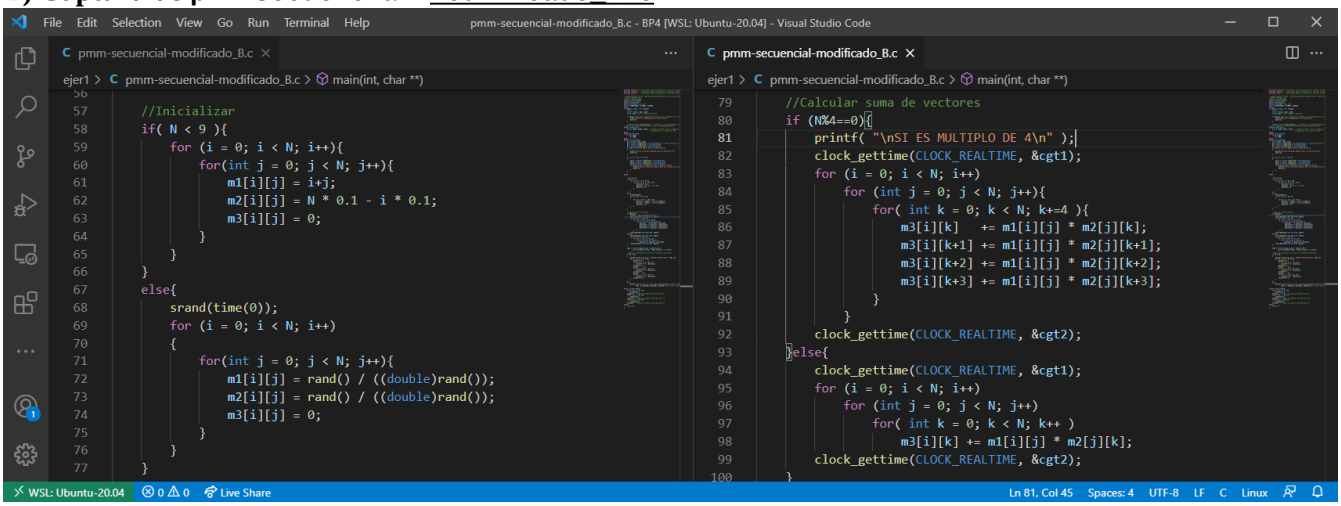
A) Captura de `pmm-secuencial-modificado_A.c`

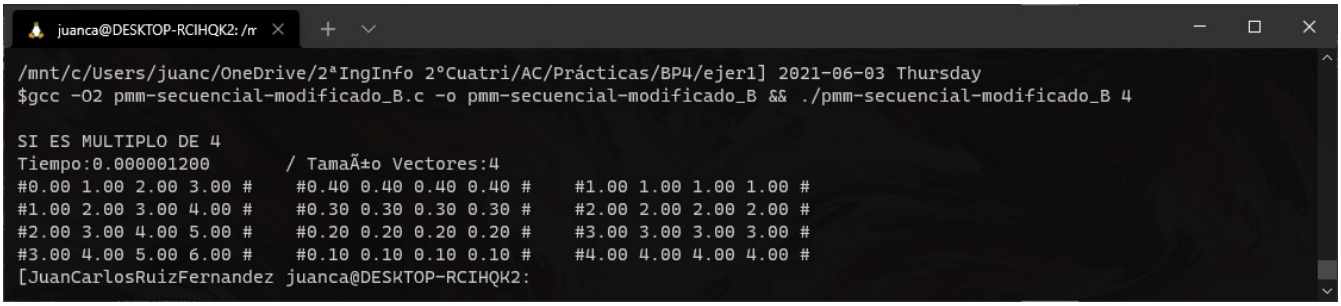


Capturas de pantalla (que muestren la compilación y que el resultado es correcto):



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





TIEMPOS: He recalculado las ejecuciones para un tamaño N==400

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	-	0.060923700
Modificación A)	Minimizar saltos en memoria, posiciones no consecutivas, intercambiando el uso de las variables en el ultimo bucle: k por j y viceversa	0.032524800
Modificación B)	Desenrollado de bucle para hacer 4 iteraciones. Comprobación de que sea múltiplo de 4 para la multiplicación y si no lo es aplicar los bucles sin desenrollar	0.029016500

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

Donde se aprecia unas diferencias mas significativas es entre el secuencial y la modificación A. En la cual con ese intercambio de variables accedo a posiciones de memoria consecutivas (columnas de una misma fila) y no como en el secuencial que accedo de fila en fila.
Con la modificación b no es tan significativa la diferencia pero si se aprecia cierta mejora. Teniendo en cuenta que hay más código a ejecutar.

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`

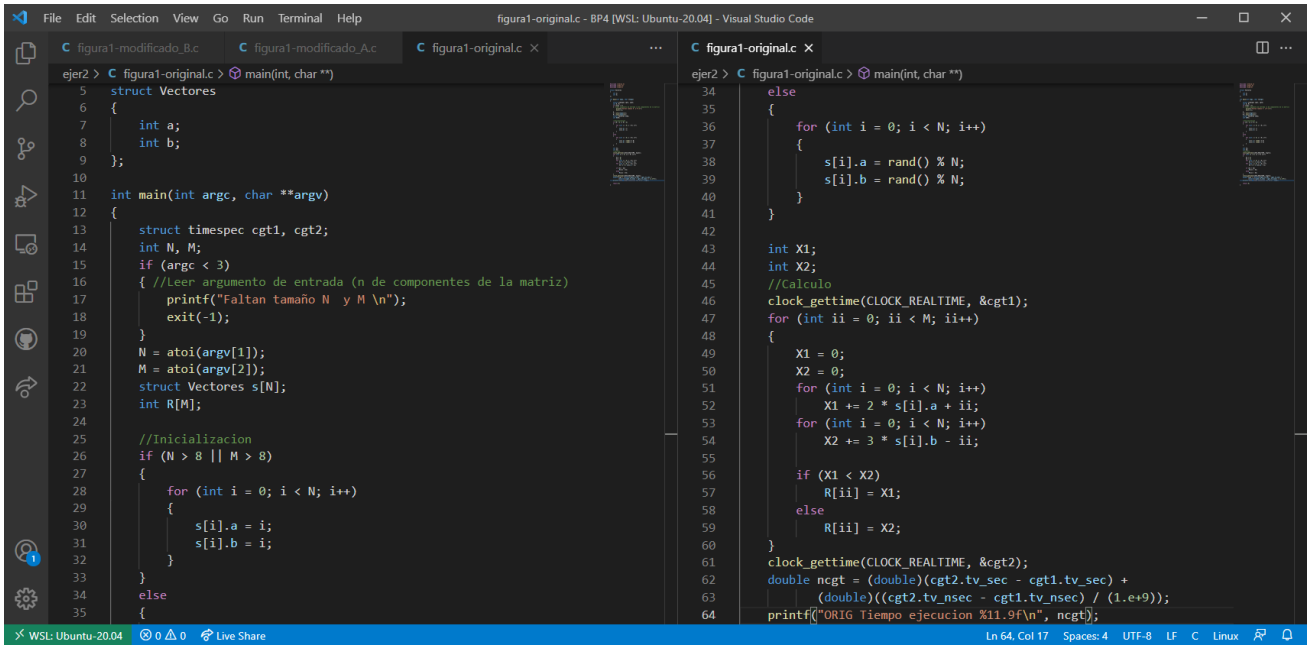


Figura 1 . Código C++ que suma dos vectores. M y N deben ser parámetros de entrada al programa, usar valores mayores que 1000 en la evaluación.

```
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.

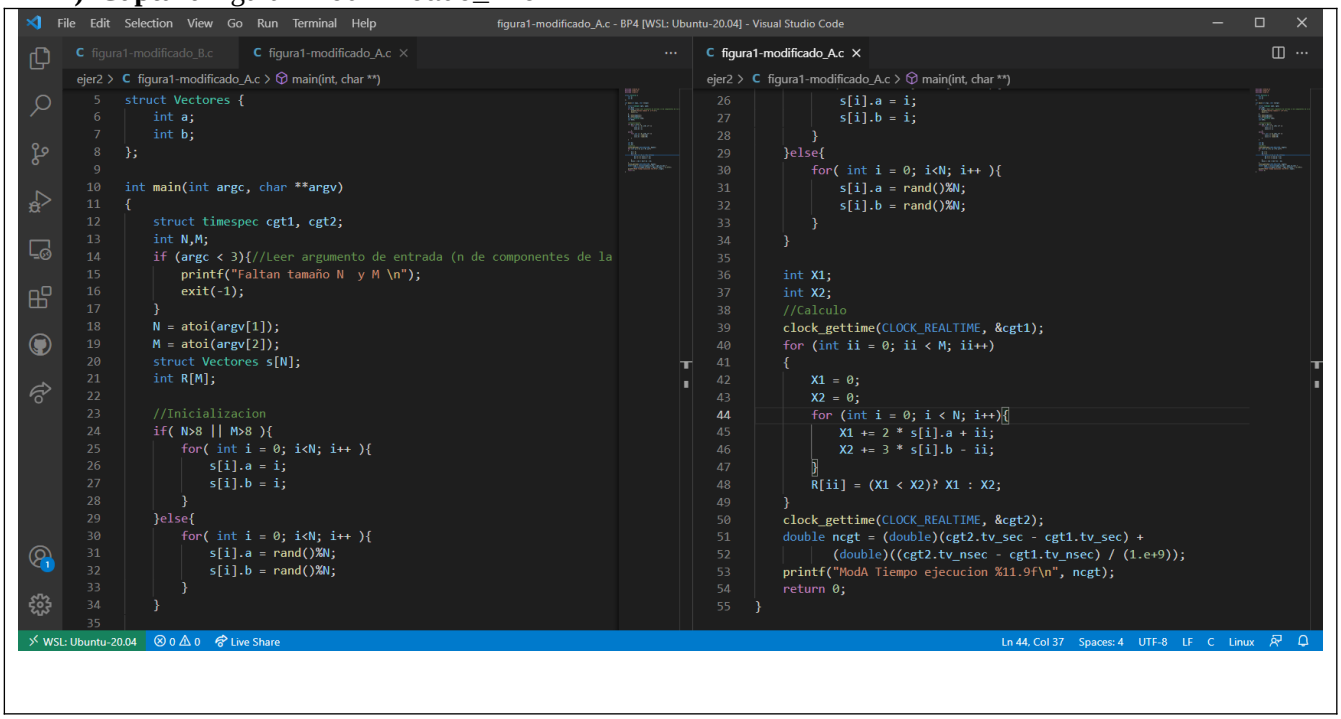
MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación-: El struct tal como esta es mas optimo recorrer ambos elementos en un solo bucle para minimizar los saltos.

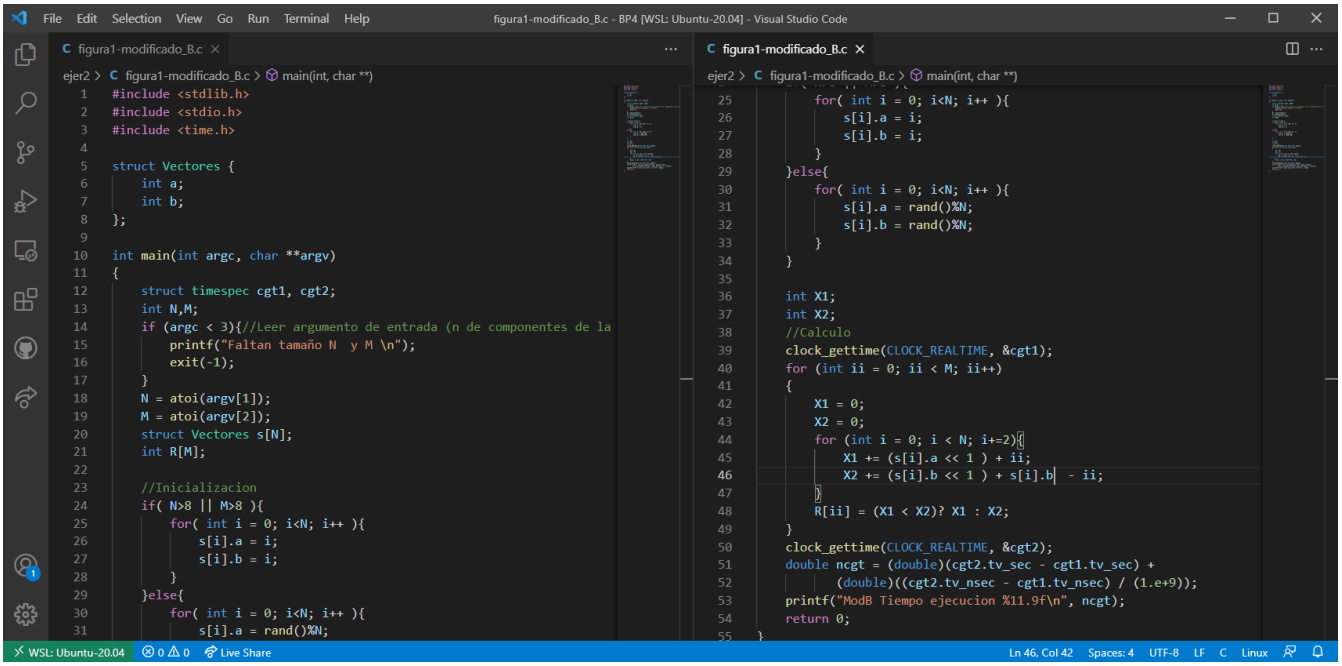
Modificación B) –explicación-: Cambiado las multiplicaciones por desplazamientos y sumas

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c



B) Captura figura1-modificado_B.c




```
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer2] 2021-06-03 Thursday
$gcc -O2 figura1-original.c -o figura1-original && gcc -O2 figura1-modificado_A.c -o figura1-modificado_A && gcc -O2 figura1-modificado_B.c -o figura1-modificado_B && ./figura1-original 1000 1000 && ./figura1-modificado_A 1000 1000 && ./figura1-modificado_B 1000 1000
ORIG Tiempo ejecucion 0.001080100
ModA Tiempo ejecucion 0.000855600
ModB Tiempo ejecucion 0.000275700
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer2] 2021-06-03 Thursday
$
```

TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	-	*0.001080100*
Modificación A)	**El struct tal como esta es mas optimo recorrer ambos elementos en un solo bucle para minimizar los saltos.**	0.000855600
Modificación B)	**Cambiado las multiplicaciones por desplazamientos y sumas**	0.000275700

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

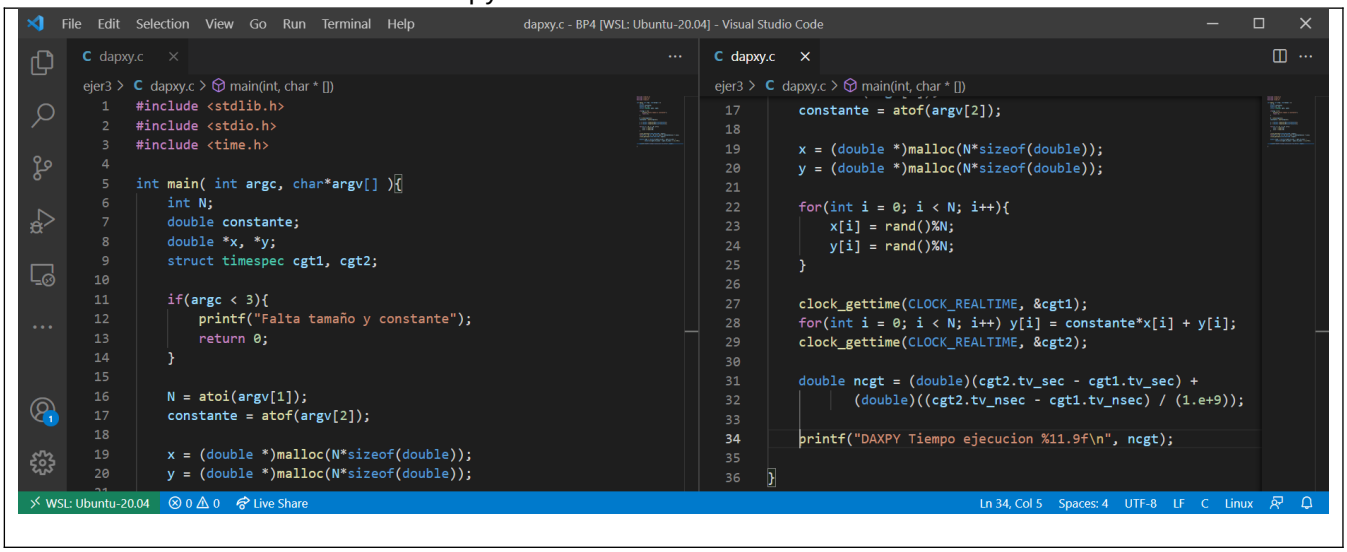
Las diferencias son bastantes considerables al cambiar la forma de recorrer el array del struct. Tal como estaba era mejor recorrer ambos valores en un mismo bucle, posiciones de memoria consecutivas. Mas considerable es la diferencia sustituyendo la multipliacion por el desplazamiento y la suma.

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`



Tiempos ejec. Longitud vectores=9000000	-O0	-Os	-O2	-O3
	0.021555900	0.019251700	0.019052200	0.018430800

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

```
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer3] 2021-06-09 Wednesday
$gcc -O0 dapxy.c -o dapxy0
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer3] 2021-06-09 Wednesday
$gcc -Os dapxy.c -o dapxys
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer3] 2021-06-09 Wednesday
$gcc -O2 dapxy.c -o dapxy2
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer3] 2021-06-09 Wednesday
$gcc -O3 dapxy.c -o dapxy3
```

```
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer3] 2021-06-09 Wednesday
$./dapxy0 9000000 5
DAXPY Tiempo ejecucion 0.021555900
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer3] 2021-06-09 Wednesday
$./dapxys 9000000 5
DAXPY Tiempo ejecucion 0.019251700
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer3] 2021-06-09 Wednesday
$./dapxy2 9000000 5
DAXPY Tiempo ejecucion 0.019052200
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP4/ejer3] 2021-06-09 Wednesday
$./dapxy3 9000000 5
DAXPY Tiempo ejecucion 0.018430800
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

A parte de los tamaños, siendo el -O3 el más largo porque desenrolla bucles, tenemos las comprobaciones:

O0 hace la comprobacion en L6 para volver a L7

Os hace la comprobacion al principio para saltar a L12 y al final de L6 siempre salta a si mismo (L6)

O2 la hace solamente al final de su etiqueta L7

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

```
call    clock_gettime@PLT
        movl    $0, -88(%rbp)
        jmp     .L6
.L7:
        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)
.L6:
        movl    -88(%rbp), %eax
        cmpl    -84(%rbp), %eax
        jl      .L7
        leaq    -32(%rbp), %rax
        movq    %rax, %rsi
        movl    $0, %edi
        call    clock_gettime@PLT
```

daxpy0s.s

```
call    clock_gettime@PLT
        xorl    %eax, %eax
.L6:
        cmpl    %eax, %ebx
        jle     .L12
        movsd   8(%rsp), %xmm0
        mulsd   0(%r13,%rax,8), %xmm0
        addsd   0(%rbp,%rax,8), %xmm0
        movsd   %xmm0, 0(%rbp,%rax,8)
        incq    %rax
        jmp     .L6
.L12:
        leaq    40(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
```

daxpy02.s

```
call    clock_gettime@PLT
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L7:
        movsd   8(%rsp), %xmm0
        mulsd   (%r12,%rax,8), %xmm0
        movq    %rax, %rdx
        addsd   (%r14,%rax,8), %xmm0
        movsd   %xmm0, (%r14,%rax,8)
        addq    $1, %rax
        cmpq    %rdx, %r13
        jne     .L7
.L8:
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
```

daxpy03.s

```
call    clock_gettime@PLT
        cmpl    $1, %r12d
        je      .L9
        movsd   8(%rsp), %xmm1
        shrl    %r12d
        xorl    %edx, %edx
        salq    $4, %r12
        unpcklpd        %xmm1,
%xmm1
        .p2align 4,,10
        .p2align 3
.L7:
        movupd  0(%r13,%rdx), %xmm0
        movupd  (%r15,%rdx), %xmm2
        mulpd   %xmm1, %xmm0
        addpd   %xmm2, %xmm0
        movups  %xmm0, (%r15,%rdx)
        addq    $16, %rdx
        cmpq    %rdx, %r12
        jne     .L7
.L9:
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        movq    40(%rsp), %rax
        pxor    %xmm0, %xmm0
        subq    24(%rsp), %rax
        cvtsi2sdq       %rax, %xmm0
        pxor    %xmm1, %xmm1
        movq    32(%rsp), %rax
        subq    16(%rsp), %rax
        cvtsi2sdq       %rax, %xmm1
        movl    $1, %edi
        movl    $1, %eax
        divsd   .LC1(%rip), %xmm0
        leaq    .LC2(%rip), %rsi
        addsd   %xmm1, %xmm0
        call    __printf_chk@PLT
.L3:
        movq    56(%rsp), %rax
        xorq    %fs:40, %rax
        jne     .L18
        addq    $72, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 56
        xorl    %eax, %eax
        popq    %rbx
        .cfi_def_cfa_offset 48
        popq    %rbp
        .cfi_def_cfa_offset 40
        popq    %r12
        .cfi_def_cfa_offset 32
        popq    %r13
        .cfi_def_cfa_offset 24
        popq    %r14
        .cfi_def_cfa_offset 16
        popq    %r15
        .cfi_def_cfa_offset 8
        ret
.L17:
        .cfi_restore_state
        leaq    .LC0(%rip), %rsi
        movl    $1, %edi
        call    __printf_chk@PLT
        jmp     .L3
.L4:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
```