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

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Gil Bautista David
Grupo de prácticas: C1
Fecha de entrega:
Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz*

4 Cores físicos y 2 hilos por núcleo.

Sistema operativo utilizado: *Windows 10 usando el bash de ubuntu*

Versión de gcc utilizada: *gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609*

Adjunte el contenido del fichero /proc/cpuinfo de la máquina en la que ha tomado las medidas

1. Para el núcleo que se muestra en la Figura 1 (ver guion de prácticas), y para un programa que implemente la multiplicación de matrices (use variables globales):

   1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos (use –O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

   1.2 Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.

   1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

A) MULTIPLICACIÓN DE MATRICES:
CÓDIGO FUENTE: `pmm-secuencial.c` `(Mismo código del bloque anterior)`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Convertir la matriz en un vector para hacer alineamiento de memoria.

Modificación b) –explicación-: Cambiar los bucles **j** y **k** para optimizar el acceso a memoria en la multiplicación.

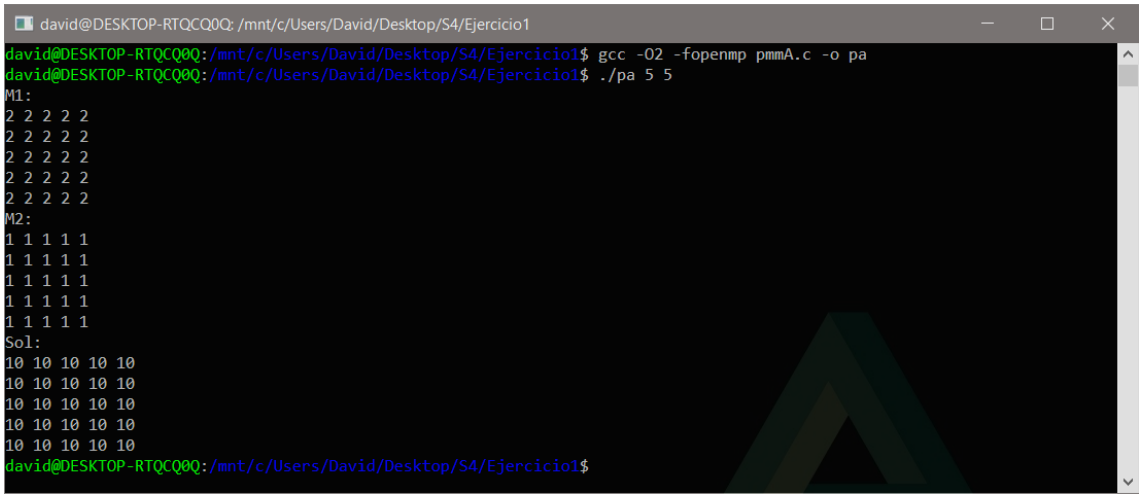
1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmmA.c`

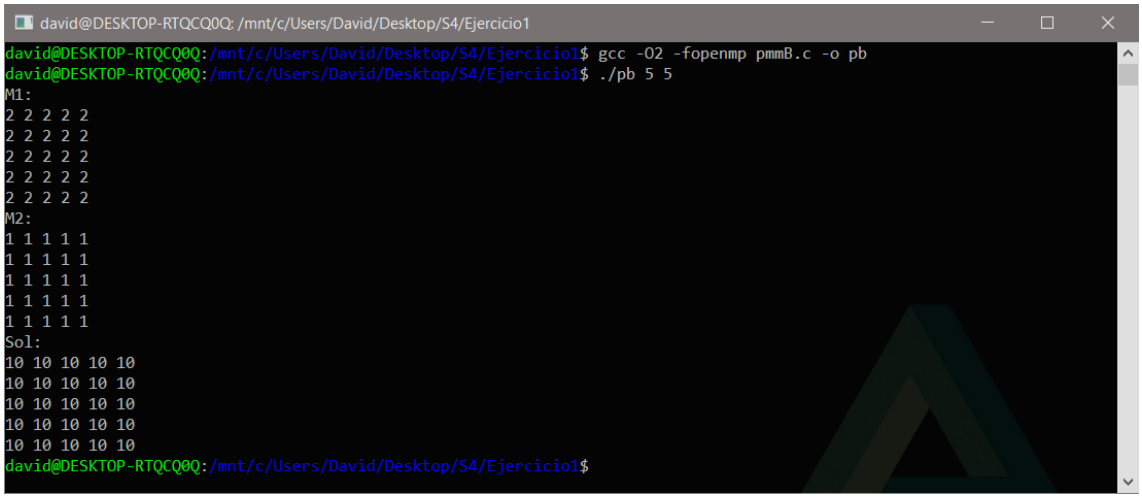
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */
```

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



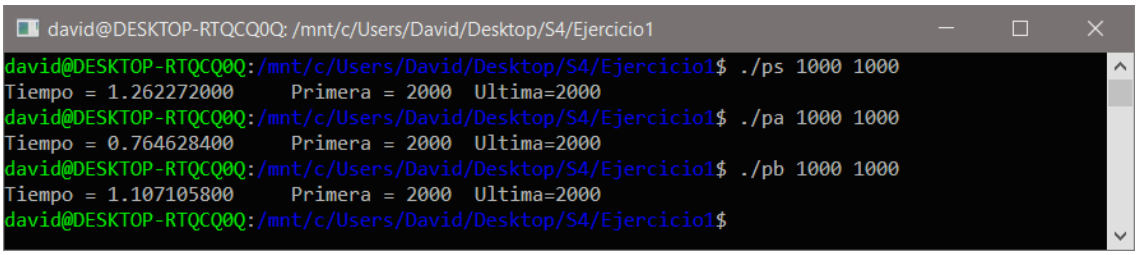
b) pmmB.c

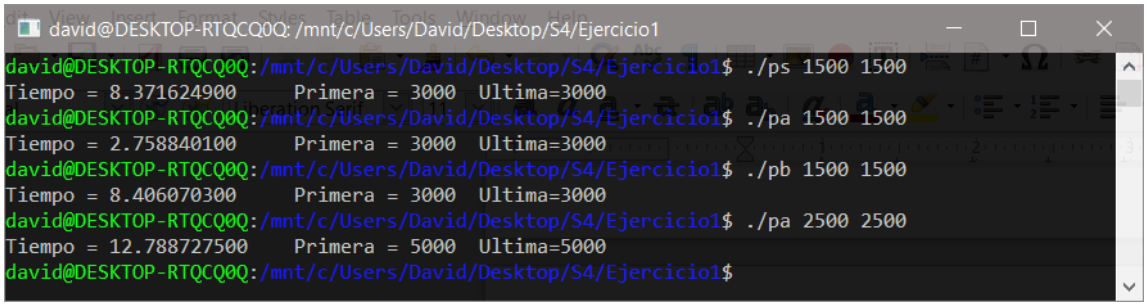


1.1. TIEMPOS:

Tamaño: 1000

Modificación	**-O2**
Sin modificar	1.26227200
Modificación a)	0.764628400
Modificación b)	1.107105800





1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Observamos un notable beneficio en la modificación “a” ya que al convertir la matriz en vector se requieren muchos menos accesos a memoria que son lo que realmente produce que el tiempo de ejecución aumente ya que los cálculos para la multiplicación se realizan muy rápido.

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

`pmm-secuencial.s`	`pmm-secuencial-modificado_b.s`	`pmm-secuencial-modificado_c.s`

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-original.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
for(ii = 1; ii <= 40000; ii++){
        X1 = 0; X2 = 0;

        // for (i = 0; i < 5000; i++) {
        //      X1 += 2*s[i].a+ii;
        //      X2 += 2*s[i].b-ii;
        // }

        for (i = 0; i < 5000; 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;

        //R[ii] = (X1 < X2) ? X1 : X2;
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Se unen los bucles internos para realizar la mitad de iteraciones a pesar de hacer más cálculos.

Modificación b) –explicación-: Eliminar un bucle for y unirlo a otro ya que hacían lo mismo.

Modificación c) –explicación-: Al final del bloque he sustituido la estructura if – else por una condición ternaria R[ii] = (X1 < X2) ? X1 : X2; pero aumentaba más el tiempo de ejecución (aunque puede que fuera solo una casualidad).

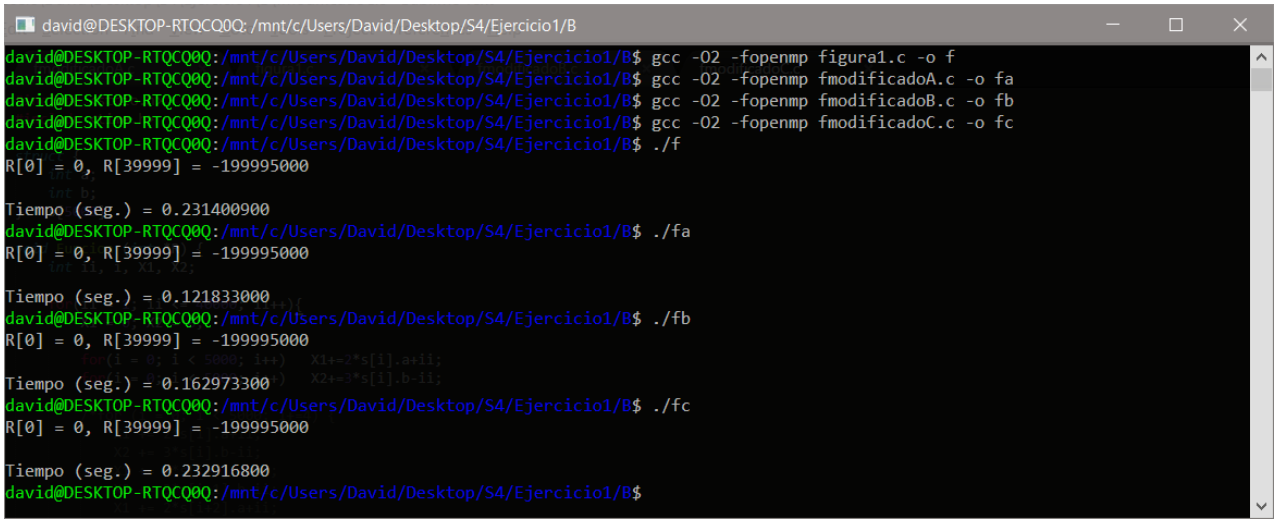
1.1. CÓDIGOS FUENTE MODIFICACIONES

a) **`fa.c`**
`fb.c`
`fc.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */
```

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



1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.231400900*
Modificación a)	*0.121833000*
Modificación b)	*0.162973300*
Modificación c)	*0.232916800*

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Como se puede observar la modificación a es la que mejor resultado obtiene ya que al unificar todos los cálculos en un solo bucle se reduce bastante el número de iteraciones.

La modificación c, como he comentado antes, aumenta el tiempo de ejecución respecto al programa original por lo que he optado por no añadirla.

Comparando la modificación a y b se puede observar que aunque la a realiza ¼ de las iteraciones que la b el tiempo de ejecución varía muy poco y complica bastante el entendimiento del código.

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

pmm-secuencial.s	pmm-secuencial-modificado_b.s	pmm-secuencial-modificado_c.s
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */	/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */	/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 */ /* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/ /* INTERLINEADO SENCILLO */

2. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina denominada DAXPY (***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=1;i<=N,i++) y[i]= a*x[i] + y[i];
```

2.1. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O0, -O2, -O3) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo

del número de operaciones en coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CÓDIGO FUENTE: `daxpy.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

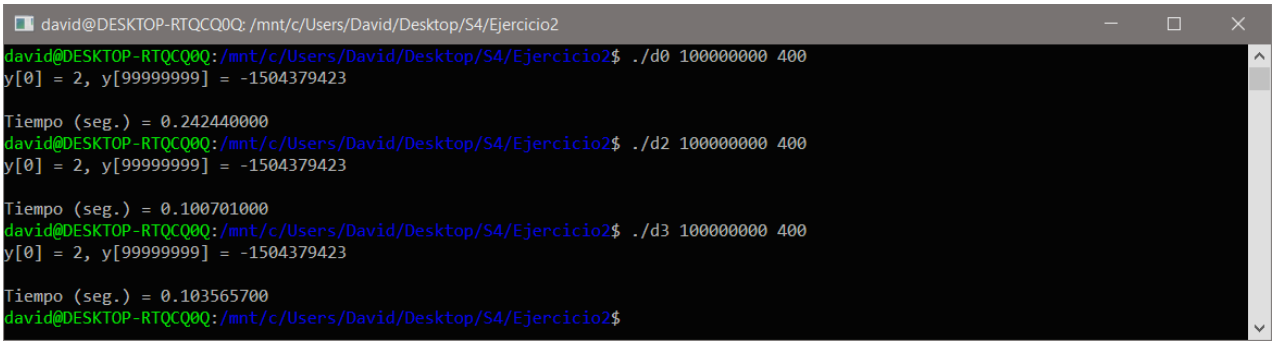
```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 y 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

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

Tiempos ejec.	**-O0**	**-O2**	**-O3**
	0.242440000	0.100701000	0.103565700

CAPTURAS DE PANTALLA:



COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

CÓDIGO EN ENSAMBLADOR (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

`daxpyO0.s`	`daxpyO2.s`	`daxpyO3.s`