

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): Cristobal Lopez Pe;alver

Grupo de prácticas: B1

Fecha de entrega:

Fecha evaluación en clase:

[RECORDATORIO, quitar todo este texto en rojo del cuaderno definitivo–

1. COMENTARIO

Este cuaderno de prácticas se utilizará para asignarle una puntuación durante la evaluación continua de prácticas y también lo utilizará como material de estudio y repaso para preparar el examen de prácticas escrito. Luego redáctelo con cuidado, y sea ordenado y claro.

2. NORMAS SOBRE EL USO DE LA PLANTILLA

1) Usar **interlineado SENCILLO**.

2) Respetar los tipos de letra y tamaños indicados:

- Calibri-11 o Liberation Serif-11 para el texto

- Courier New-10 o Liberation Mono-10 para nombres de fichero, comandos, variables de entorno, etc., cuando se usan en el texto.

- Courier New o Liberation Mono de tamaño 8 o 9 para el código fuente en los listados de código fuente.

- Formatee el código fuente de los listados para que sea legible, limpio y claro. Consulte, como ejemplo, los Listados 1 y 2 del guion (tabule, comente, ...)

3) Insertar las capturas de pantalla donde se pidan y donde se considere oportuno

Recuerde que debe **adjuntar al zip de entrega, el pdf de este fichero, todos los ficheros con código fuente implementados/utilizados y el resto de ficheros que haya implementado/utilizado (scripts, hojas de cálculo, etc.), lea la Sección 1.4 del guion]**

Versión de gcc utilizada: *(respuesta)*

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:
 - a. Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos a partir de la modificación realizada.
 - b. Genere los programas en ensamblador para los programas modificados obtenidos en el punto anterior considerando las distintas opciones de optimización del compilador (-O1, -O2,...) e incorpórelos al cuaderno de prácticas. Compare los tiempos de ejecución de las versiones de código ejecutable obtenidas con las distintas opciones de optimización y explique las diferencias en tiempo a partir de las características de dichos códigos. Destaque las diferencias en el código ensamblador.
 - c. (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-modificado.c

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

| |
|----------|
| Adjunto. |
|----------|

MODIFICACIONES REALIZADAS:

Modificación a) –explicación–: He desenrollado teniendo en cuenta los distintos valores que puede tomar N a fin de evitar cores. Baja mejora.

Modificación b) –explicación–: He utilizado un algoritmo de multiplicación por bloques con subbloques de N/8 ya que eran los que mejores resultados daban.

...

NOTA: PARA N = 1000

| Modificación | -O0 | -O1 | -O2 | -O3 | -Os |
|---------------------|------------------|------------|------------|------------|------------|
| Sin modificar | 12.162796 | 7.790697 | 7.822186 | 7.810541 | 7.650664 |
| Modificación a) | 11.990881 | 8.051484 | 7.967990 | 7.963396 | 7.961131 |
| Modificación b) | 4.514788 | 1.368787 | 0.976531 | 1.029085 | 1.709673 |

COMENTARIOS SOBRE LOS RESULTADOS:

CAPTURAS DE PANTALLA:

(Optimización A)

```
cristobal@PG6: ~/Escritorio/COLE xd/2.2/AC/Prácticas/p4

cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -O0 -o pmm-secuencialA pmm-secuencial
-modificadoA.c; ./pmm-secuencialA 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:11.990881
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -O1 -o pmm-secuencialA pmm-secuencial
-modificadoA.c; ./pmm-secuencialA 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:8.051484
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -O2 -o pmm-secuencialA pmm-secuencial
-modificadoA.c; ./pmm-secuencialA 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:7.967990
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -O3 -o pmm-secuencialA pmm-secuencial
-modificadoA.c; ./pmm-secuencialA 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:7.963396
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -Os -o pmm-secuencialA pmm-secuencial
-modificadoA.c; ./pmm-secuencialA 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:7.961131
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ _
```

(Optimización B)

```

cristobal@PG6: ~/Escritorio/COLE xd/2.2/AC/Prácticas/p4

cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -O0 -o pmm-secuencial-modificado pmm-
secuencial-modificado.c; ./pmm-secuencial-modificado 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:4.514788
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -O1 -o pmm-secuencial-modificado pmm-
secuencial-modificado.c; ./pmm-secuencial-modificado 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:1.368787
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -O2 -o pmm-secuencial-modificado pmm-
secuencial-modificado.c; ./pmm-secuencial-modificado 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:0.976531
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -O3 -o pmm-secuencial-modificado pmm-
secuencial-modificado.c; ./pmm-secuencial-modificado 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:1.029085
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ gcc -Os -o pmm-secuencial-modificado pmm-
secuencial-modificado.c; ./pmm-secuencial-modificado 1000
Ultimo valor de i para N=1000: 999

matriz[0][0]:1000
matriz[N-1][N-1]:1000000

Tiempo en ejecutar matriz1xmatriz2:1.709673
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ _

```

B) CÓDIGO FIGURA 1:**CÓDIGO FUENTE:** figura1-modificado.c**(ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

Adjunto.

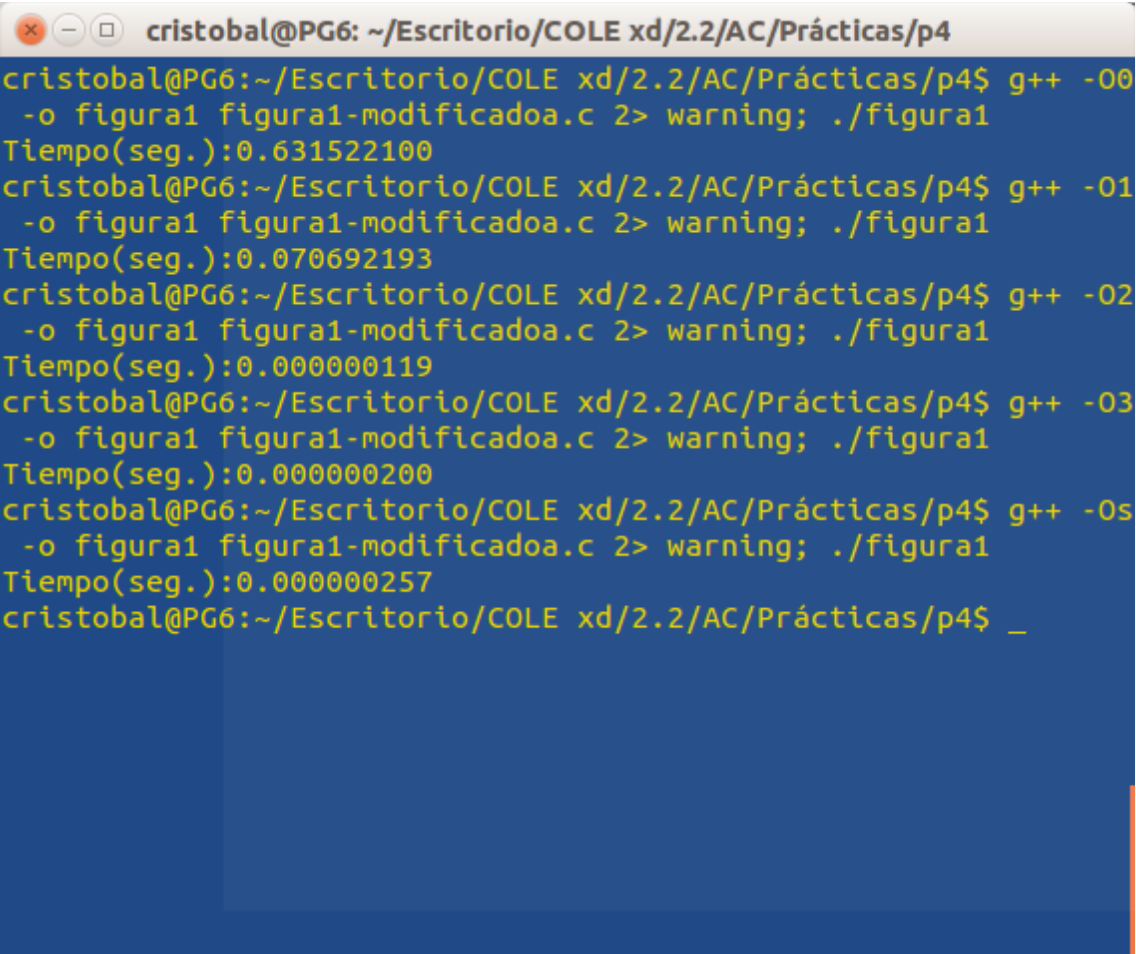
MODIFICACIONES REALIZADAS:

Modificación a) –explicación–: He fusionado en uno los 2 bucles que rellenaban X1 y X2.

Modificación b) –explicación–: He desenrollado el bucle obtenido anteriormente. Usar `_mm_prefetch()` no me ofrece mejoría. También he buscado alternativa al `+=`.

| Modificación | -O0 | -O1 | -O2 | -O3 | -Os |
|-----------------|--------------------|-------------|-------------|-------------|-------------|
| Sin modificar | 1.003782179 | 0.138296710 | 0.000000119 | 0.000000110 | 0.000000114 |
| Modificación a) | 0.631522100 | 0.070692193 | 0.000000119 | 0.000000200 | 0.000000257 |
| Modificación b) | 0.439123893 | 0.018008548 | 0.000000135 | 0.000000124 | 0.000000191 |

(OPTIMIZACIÓN A)



```
cristobal@PG6: ~/Escritorio/COLE xd/2.2/AC/Prácticas/p4
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -O0
-o figura1 figura1-modificadoa.c 2> warning; ./figura1
Tiempo(seg.):0.631522100
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -O1
-o figura1 figura1-modificadoa.c 2> warning; ./figura1
Tiempo(seg.):0.070692193
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -O2
-o figura1 figura1-modificadoa.c 2> warning; ./figura1
Tiempo(seg.):0.000000119
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -O3
-o figura1 figura1-modificadoa.c 2> warning; ./figura1
Tiempo(seg.):0.000000200
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -Os
-o figura1 figura1-modificadoa.c 2> warning; ./figura1
Tiempo(seg.):0.000000257
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ _
```

(OPTIMIZACIÓN B)

```

cristobal@PG6: ~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -O0
-o figura1 figura1-modificadob.c 2> warning; ./figura1
Tiempo(seg.):0.439123893
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -O1
-o figura1 figura1-modificadob.c 2> warning; ./figura1
Tiempo(seg.):0.018008548
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -O2
-o figura1 figura1-modificadob.c 2> warning; ./figura1
Tiempo(seg.):0.000000135
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -O3
-o figura1 figura1-modificadob.c 2> warning; ./figura1
Tiempo(seg.):0.000000124
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ g++ -Os
-o figura1 figura1-modificadob.c 2> warning; ./figura1
Tiempo(seg.):0.000000191
cristobal@PG6:~/Escritorio/COLE xd/2.2/AC/Prácticas/p4$ _

```

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

- a. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O1, -O2,..) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarreen. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

- b. (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)
{
...
}
```

| | -O0 | -O1 | -O2 | -O3 | -Os |
|----------------------|--------------------------------|-----|-----|-----|-----|
| Tiempos ejec. | <i>De 2 a 10 seg. aquí</i> | | | | |

CAPTURAS DE PANTALLA:

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

CÓDIGO EN ENSAMBLADOR: (ADJUNTAR AL .ZIP)

(LIMITAR AQUÍ EL CÓDIGO INCLUIDO A LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE SE REALIZA LA OPERACIÓN CON VECTORES)

daxpy00.s

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

daxpy01.s

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

daxpy02.s

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

daxpy03.s

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

daxpy0s.s

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