P1- COMPILADORES

Armando Nogueira Rio

```
C:\Users\armando\workspace\c\COMP4>gcc --version
gcc (GCC) 5.4.0
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Las prácticas se harán con la versión 5.4 del compilador de gcc para Windows.

1. **Las medidas de tiempo de ejecución se harán utilizando la función gettimeofday().**
    a. **Lee el manual de dicha función.**
    b. **Si queremos medir el tiempo en una parte del código haremos lo siguiente:**

```
#include <sys/time.h>

…
struct timeval inicio, final;
double tiempo;
…
gettimeofday(&inicio, NULL);
{ …} /*Código a medir*/
gettimeofday(&final, NULL);
tiempo = (final.tv_sec - inicio.tv_sec + (final.tv_usec - inicio.tv_usec) / 1.e6);
```

    c. **Comprueba la influencia del tiempo empleado por la función gettimeofday.**
    d. **Mide el tiempo de ejecución con dos ejemplos simples: uno con muchas operaciones aritméticas, y otro con muchas operaciones de entrada/salida. Haz varias medidas.**

2. **Usaremos el compilador gcc.**
    a. **Lee el manual de dicho compilador.**
    b. **El ensamblador que genera es el de la arquitectura IA32 de Intel. Su manual está aquí . Hojéalo. Tienes información interesante aquí y aquí.**
    c. **Para los siguientes apartados considera el siguiente código que realiza el producto de dos matrices cuadradas. Comprueba que funciona correctamente.**

```c
#include <stdio.h>

#define Nmax 600

void producto(float x, float y, float *z) {
    *z = x * y;
}

main() {
    float A[Nmax][Nmax], B[Nmax][Nmax], C[Nmax][Nmax], t, r;
    int i, j, k;
    for (i = 0; i < Nmax; i++) /* Valores de las matrices */ for (j = 0; j < Nmax;
j++) {
            A[i][j] = (i + j) / (j + 1.1);
            B[i][j] = (i - j) / (j + 2.1);
        }
    for (i = 0; i < Nmax; i++) /* Producto matricial */ for (j = 0; j < Nmax; j++) {
            t = 0;
            for (k = 0; k < Nmax; k++) {
                producto(A[i][k], B[k][j], &r);
                t += r;
            }
            C[i][j] = t;
        }
}
```

    d. **La opción –E realiza solamente el preprocesado. Comprueba cómo se sustituyen las constantes.**
    e. **La opción –S genera el código en ensamblador. Compruébalo. Analiza la sintáxis de este ensamblador, cómo se implementan los lazos, cómo se hacen las llamadas a funciones y como son las operaciones en punto flotante.**
    f. **La opción –c genera el código objeto. Compruébalo. Un fichero objeto se puede enlazar con gcc invocando el fichero .o directamente.**
    g. **El enlazado estático se puede hacer con la opción –static de gcc. Comprueba el tamaño del ejecutable. Los ficheros compilados de esta manera son autónomos al quedar incorporadas a su código las librerías.**
    h. **Compíla con las opciones -O0, -O1, -O2, -O3, -Os. Compara el tamaño de los códigos objeto de cada compilación. Compara los tiempos de ejecución. Compara los códigos en ensamblador.**

Opción	-O0	-O1	-O2	-O3	-Os
Tiempo	1.413806	0.083442	0.000001	0.000000	0.000000
Tamaño	4K	2K	2K	2K	2K

Para no engordar demasiado esta sección los códigos ensamblador se ubicarán al final de este documento en los anexos del ejercicio 2, lo más sorprendente del código ensamblador es la gran optimización que realiza el compilador dado que con -O1 no llama a la función producto, y con -O2 en adelante ni siquiera realiza ningún tipo de bucle

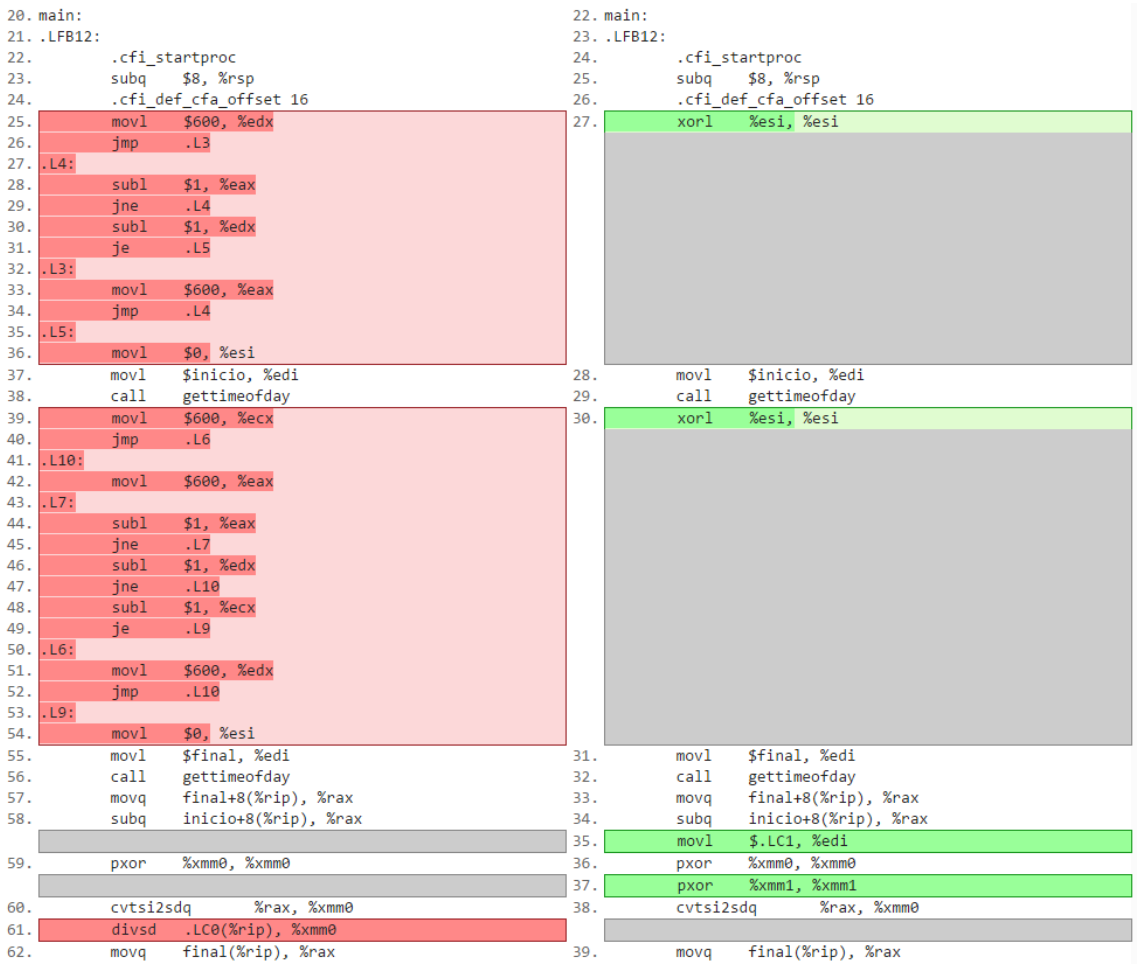

Figura 1: O1 vs O2

El motivo por el que puede realizar tantas optimizaciones es porque el resultado de todo el cálculo no se usa para nada y por lo tanto se puede ignorar todo el cálculo y el programa es prácticamente un printf y dos mediciones.

Para comprobar esta teoría hemos hecho la misma prueba añadiendo un printf al final del cálculo, aunque fuera de la medida de tiempos, evidentemente, consiguiendo así que el compilador no sea capaz de optimizar tanto el código y obteniendo los siguientes resultados:

Opción	-O0	-O1	-O2	-O3	-Os
Tiempo	1.417668	0.237451	0.229912	0.064996	0.228796
Tamaño	4K	3K	4K	4K	3K

En este resultado resulta curioso que la opción O3 consiga una bajada tan grande de tiempo aun cuando muestra todos los resultados correctamente. Si bien es cierto, que el nivel de compilación O3 es considerado peligroso por mucha gente dado que en ocasiones proporciona optimizaciones que no son equivalentes al código inicial, en este caso el resultado es sorprendente y exacto.

El motivo de este resultado debe de ser el uso de operaciones vectoriales dado que una primera vista sobre el resultado de la operación matricial no nos ha permitido distinguir ningún patrón.

3. **Considera el código adjunto. Compílalo primero con la opción -O1 y posteriormente con –O1 –funroll-loops. Compara los códigos en ensamblador obtenidos. Analiza el comportamiento (en términos de tiempo de ejecución) de ambas versiones para diferentes valores de N**

```c
#include <stdio.h>

#define N 10000
double res[N];

main() {
    int i;
    double x;
    for (i = 0; i < N; i++) res[i] = 0.0005 * i;
    for (i = 0; i < N; i++) {
        x = res[i];
        if (x < 10.0e6) x = x * x + 0.0005; else x = x - 1000;
        res[i] += x;
    }
    printf("resultado= %e\n", res[N - 1]);
}
```

N = 100000		
Opciones	**-O1**	**-O1 -funroll-loops**
Tiempo	2.549950e+03	2.549950e+03
Tamaño	2K	4K

N = 10000		
Opciones	**-O1**	**-O1 -funroll-loops**
Tiempo	2.999500e+01	2.999500e+01
Tamaño	2K	4K

N = 1000		
Opciones	**-O1**	**-O1 -funroll-loops**
Tiempo	7.495002e-01	7.495002e-01
Tamaño	2K	4K

Los tiempos de ejecución son iguales, puesto que el número de instrucciones que realiza son el mismo, sin embargo, el código ensamblador cambia sustancialmente debido a que la opción unroll-loops, como su propio nombre indica, desenrolla los bucles haciendo que el código resultante no tenga ningún bucle, por lo cual será mucho más largo.

Normalmente en esta prueba podría aparecer cierta mejoría al desarrollar los bucles debido a que el procesador conoce siempre la instrucción siguiente y las puede introducir en el pipeline de manera que se empiecen a ejecutar.

Anexo

1. Códigos ensamblador del Ejercicio 2
2. Códigos ensamblador del Ejercicio 2 modificado para que imprima el valor del cálculo
3. Códigos ensamblador del Ejercicio 3