Informe práctica 01

Pedro Bueno Aldrey

Ejercicio 2h

Compilamos el código 3 veces usando las opciones de compilación dadas:

```
$ gcc -S -O0 -masm=intel main.c -o main0.s
$ gcc -S -O1 -masm=intel main.c -o main1.s
$ gcc -S -O2 -masm=intel main.c -o main2.s
$ gcc -S -O3 -masm=intel main.c -o main3.s
$ gcc -S -Os -masm=intel main.c -o mains.s
```

Podemos observar como el tamaño del código es mayor en la opción 0 (la opción sin optimizar), a partir de ahí vemos que las sucesivas opciones disminuyen también el tamaño aunque lo hacen de manera menos significativa:

```
$wc -l main*.s
 160 main0.s
  52 main1.s
  29 main2.s
  29 main3.s
  27 mains.s
 297 total
```

Realizaremos un pequeño análisis de cada código para ver lo que está pasando:

0. La optimización 0 no optimiza nada según el manual, es así que vemos tanto ambos bucles compilados enteramente con todas las instrucciones interiores.
1. La optimización 1 detecta que no se hace nada en los bucles utilizable, y por lo tanto no ejecuta las operaciones de su interior a pesar de que seguimos viendo ambas estructuras.
2. La segunda optimización detecta que las bucles no se utilizan y por lo tanto los elimina.
3. Ya que prácticamente no existe código con la segunda optimización los niveles 2 y 3 son equivalentes.
4. La optimización -Os se utiliza para hacer el código más pequeño, pero como apenas queda código solo elimina dos líneas ambas con la misma directiva: `.p2align 4,,15`.

Ejercicio 3

La opción "funroll-loops" descompone los bucles cuyas iteraciones se conocen en tiempo de ejecución a sus correspondientes instrucciones en ensamblador sin realizar el salto. Como hay 10000 iteraciones el compilador no crea las instrucciones de las 10000 sino que hace una descomposición parcial. Por ejemplo para el primer for hace una descomposición de 8 instrucciones seguidas para luego volver a saltar. Como es natural el código resultante es más largo.

Realizaremos las pruebas de ejecución precalentando la caché, para que esté en la medida de lo posible el array cargado en niveles altos de la jerarquía de memoria y este no interfiera con las pruebas. Los resultados mostrados a continuación son la media de 20 mediciones. Tomamos mediciones desde 1024 asta 5*10^8 siguiendo las potencias de 2, obtenemos los siguientes resultados:

Vemos como el supuestamente optimizado (en naranja) es a la larga ligeramente más rápido que el no optimizado, ya que reduce el número de saltos y operaciones que debe de hacer para recorrer el mismo bucle.

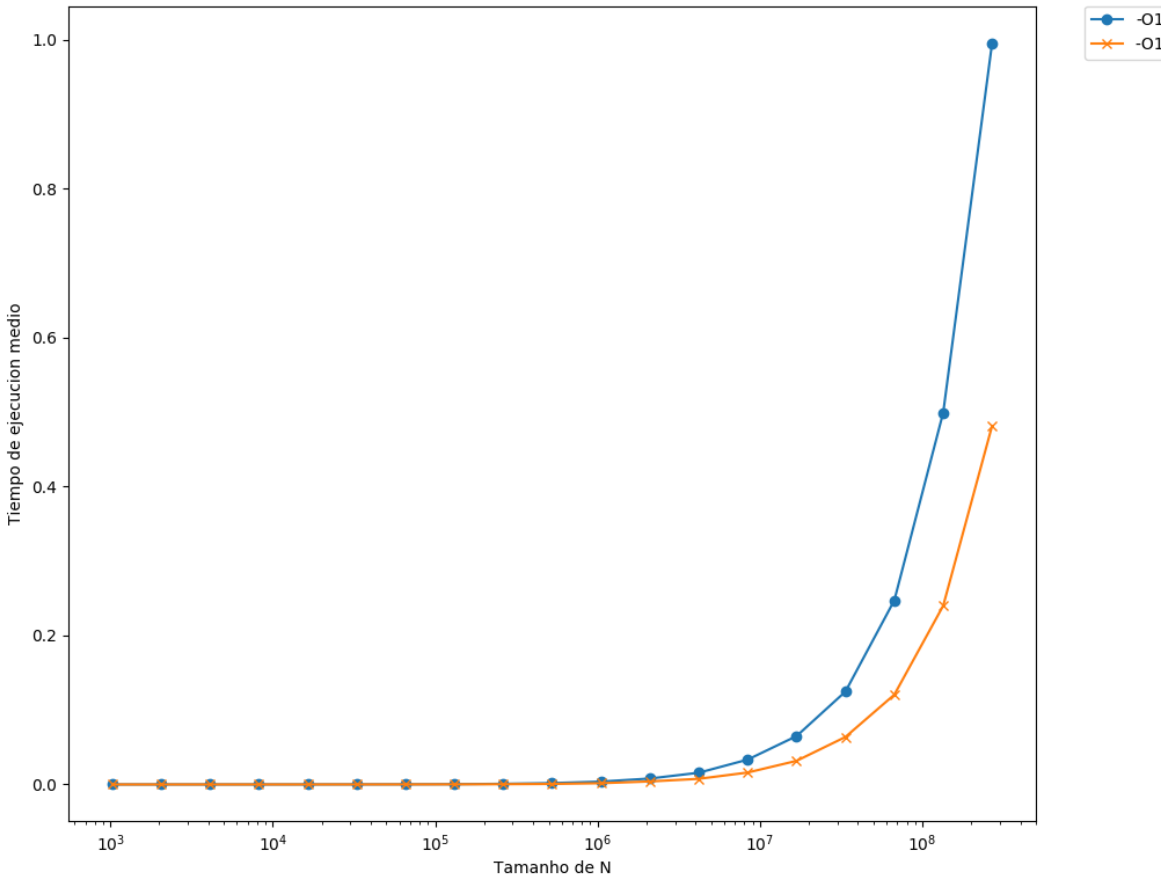
-O1
-O1
1.0
0.8
0.6
0.4
0.2
0.0
Tiempo de ejecucion medio
10^3
10^4
10^5
10^6
10^7
10^8
Tamanho de N

Figure 1:

Códigos utilizados para las mediciones:

Script de medición

```
import sys
import os
import subprocess
import matplotlib.pyplot as plt

opt = []
no_opt = []
x_axis = []

i = 1024
while i < 536870912:
    command1 = "cat main.c | sed -r 's/aqui/\#define N " + str(i) + "/g' > m.c"
    command2 = "gcc -O1 -funroll-loops m.c"
    command3 = "./a.out"
    os.system(command1)
    os.system(command2)
    a = 0
    for j in range(20):
        result = subprocess.check_output(command3, shell=True)
        a = a + float(result.split("tiempo:")[1])
    opt.append(a / 20)
    x_axis.append(i)
    print(i)
    i = i * 2

i = 1024
while i < 536870912:
    command1 = "cat main.c | sed -r 's/aqui/\#define N " + str(i) + "/g' > m.c"
    command2 = "gcc -O1 m.c"
    command3 = "./a.out"
    os.system(command1)
    os.system(command2)
    a = 0
    for j in range(20):
        result = subprocess.check_output(command3, shell=True)
        a = a + float(result.split("tiempo:")[1])
    no_opt.append(a / 20)
    i = i * 2

plt.semilogx(x_axis, no_opt, label="-O1", marker="o")
plt.semilogx(x_axis, opt, label="-O1 -funroll-loops", marker="x")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.ylabel('Tiempo de ejecucion medio')
plt.xlabel('Tamanho de N')
plt.show()
```

Código C

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

aqui

```
double res[N];
int main() {


  struct timeval inicio, final;
  double tiempo;

  for(int i = 0; i < N; i++)
    res[i] = 0;

  gettimeofday(&inicio,NULL);
  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;
  }
  gettimeofday(&final,NULL);

  tiempo = (final.tv_sec-inicio.tv_sec+(final.tv_usec-inicio.tv_usec)/1.e6);
  printf("resultado= %e tiempo: %lf \n", res[N - 1], tiempo);
  return 0;
}
```