Generación y optimización de código

25 de diciembre de 2017

Índice

1. Introducción **1**

2. Balanceo de operaciones **1**
2.1. Ventajas e inconvenientes 1

3. Caso de estudio **2**

4. Mediciones y resultados **2**
4.1. Resultados 3

5. Análisis de resultados y conclusiones **3**

6. Bibliografía **5**

1. Introducción

Este informe tiene como objetivo analizar el beneficio de la optimización **balanceo de operaciones** y sus efectos cuando se utiliza a la vez con distintos niveles de optimización de gcc.

Todos los códigos han sido compilados con gcc 5.4.0 en una máquina con sistema operativo ubuntu 16.04 de 64 bits.

2. Balanceo de operaciones

Partiendo del el código ejemplo proporcionado por el profesor en la especificación de la Práctica 2, se considera que el Balanceo de operaciones es similar a la ***fusión de lazos****. Es de mencionar que no se encontró ninguna documentación que citase la técnica "Balanceo de operaciones".*

La fusión de bucle es una optimización del compilador y una transformación que reemplaza bucles múltiples con uno solo. Es posible cuando dos bucles no hacen referencia a los datos de los demás e iteran sobre el mismo rango (bucles contables). El objetivo principal es crear paralelismo asincrónico mediante la fusión de cálculos independientes utilizando únicamente un índice para controlar las operaciones (un bucle).

2.1. Ventajas e inconvenientes

A continuación se mencionan algunas de las ventajas e inconvenientes que puede tener la aplicación del *Balanceo de operaciones*. Es importante mencionar que el rendimiento en este caso se ve muy afectado por el tipo de operaciones y cálculos que se esté realizando, la localidad de los datos y las estructuras utilizadas.

- Se puede aprovechar el caso en el que se utilicen operaciones que utilicen módulos hardware específicos, lo cual agilizaría la ejecución al dinamizar el uso del pipeline.
- La fusión de bucle no siempre mejora la velocidad del tiempo de ejecución. En algunas arquitecturas, dos bucles en realidad pueden funcionar mejor que un bucle porque, por ejemplo, hay una mayor localidad de datos dentro de cada bucle.

- Si una variable en un bucle es una función lineal simple de la variable de índice, tal como $j := 4 * i + 1$, puede ser actualizado adecuadamente cada vez que la variable de bucle se cambia; además, si esta función lineal se hace en el mismo bucle que otra operación, mejora el rendimiento.
- El número de comprobaciones que se realiza para saber si se debe salir del lazo se reduce en función de que tanto se aplique la técnica.

3. Caso de estudio

A continuación se presenta el código 1 que será caso de estudio. Como podemos ver se itera sobre una matriz de tamaño N, realizando en primer lugar una raíz cuadrada y luego una suma.

Listing 1: Código sin la optimización

```
int i, j, k;
static float x[N], y[N];


for(k=0; k<ITER; k++){
  for(i=0; i<N; i++){
    x[i] = sqrt((float)i);
  }
  for(i=0; i<N; i++){
    y[i] = (float)i+2.0;
  }
}
```

La optimización se realiza como podemos observar en el código 2. En lugar de realizar bucles separados para cada una de las operaciones, se realizan en el mismo lazo.

Listing 2: Código con la optimización

```
int i, j, k;
static float x[N], y[N];

for(k=0; k<ITER; k++){
  for(i=0; i<N; i++) {
    x[i] = sqrt((float)i);
    y[i] = (float)i+2.0;
  }
}
```

4. Mediciones y resultados

Se realizaron las optimizaciones O0, O1, O2 y O3: sin ninguna opción de optimización (O0) el objetivo del compilador es reducir el costo de la compilación y hacer que la depuración produzca los resultados esperados. Al activar los indicadores de optimización, el compilador intenta mejorar el rendimiento y / o el tamaño del código a expensas del tiempo de compilación y posiblemente la capacidad de depurar el programa.

Para la medida del tiempo de ejecución se toman en cuenta únicamente las operaciones correspondientes a la raíz cuadrada y a la suma, *T. Total*. Para obtener medidas significativas, los lazos se repiten un numero `ITER=100000` veces. Consideraremos también el tiempo que supondría la ejecución de un solo bucle, *T. Medio*, para ello se divide el tiempo obtenido de todas las iteraciones entre el numero de estas. El tiempo se mide segundos.

En las mediciones se tiene en cuenta el *calentamiento de la caché*. Durante la ejecución de las operaciones, la primera iteración solo provoca fallos, pues ninguno de los datos está cargado en caché. Para evitarlo se realiza una iteración sobre los dos arrays antes de ejecutar las operaciones de interés mencionadas en 3.

4.1. Resultados

Cuadro 1: Tiempos de ejecución para cada optimización y valor de N

Optimización	N	T. total	T. total optimizado	T. medio	T. medio optimizado
O0	10	0.454498	0.452115	0.000005	0.000005
	100	0.537071	0.530515	0.000005	0.000005
	1000	2.164715	2.162903	0.000022	0.000022
	10000	18.345381	18.467451	0.000183	0.000185
	100000	180.288948	181.513590	0.001803	0.001815
O1	10	0.453372	0.453740	0.000005	0.000005
	100	0.470021	0.453693	0.000005	0.000005
	1000	1.514734	1.040017	0.000015	0.000010
	10000	11.830518	7.219330	0.000118	0.000072
	100000	114.942187	69.044294	0.001149	0.000690
O2	10	0.455741	0.451568	0.000005	0.000005
	100	0.455904	0.454760	0.000005	0.000005
	1000	1.336401	0.878906	0.000013	0.000009
	10000	10.128875	5.608550	0.000101	0.000056
	100000	98.030437	52.884574	0.000980	0.000529
O3	10	0.453480	0.454091	0.000005	0.000005
	100	0.457381	0.458074	0.000005	0.000005
	1000	0.990776	0.878866	0.000010	0.000009
	10000	6.630820	5.604977	0.000066	0.000056
	100000	63.181213	52.887055	0.000632	0.000529

En la figura 1 está obtenida a partir de los datos de la tabla 1. Podemos observar la comparación, según el valor de N, de los tiempos de ejecución para cada optimización cuando se aplica o no el *balanceo de operaciones*.

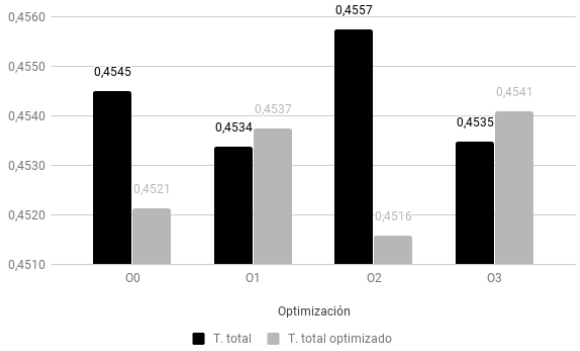
5. Análisis de resultados y conclusiones

Podemos observar que sin aplicar ninguna optimización del compilador, para tamaños de N pequeños ($N \leq 1000$), la aplicación de la técnica supone un mejor rendimiento por si sola. Sin embargo, cuando el valor de N es mayor, el desempeño es, por lo menos, un poco peor. Aún cuando las mediciones consideraron como máximo N=100000, no sería disparatado suponer que esta tendencia continuará para valores superiores.

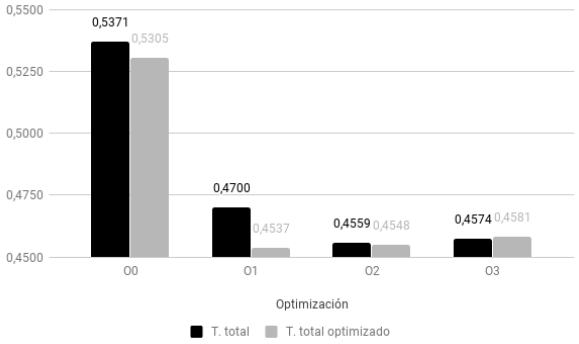
Cuando se aplica la opción -O1, podemos apreciar que el desempeño en general para valores de $N>10$ es bastante mejor, bien si comparamos únicamente los valores obtenidos para el algoritmo sin optimizar (barras negras), para el algoritmo con la optimización (barras grises), o el desempeño entre ambos algoritmos; por ejemplo, la diferencia en el caso de N=100000 entre ambos algoritmos es del orden de segundos.

Podemos destacar el caso en el que se usa esta opción para el valor N=10. Como vemos en 1 en este caso el algoritmo optimizado tiene un tiempo de ejecución ligeramente mayor que el no optimizado. Esto se puede deber bien a la existencia de interrupciones y bloqueos o a que, efectivamente, las decisiones del compilador implican un peor desempeño. Es curioso también como se puede observar el mismo comportamiento para -O3, esto indica que en general es mas probable que para valores pequeños de N sea mejor limitarse únicamente a aplicar el balanceo de operaciones pues, incluso cuando el comportamiento para -O2 nos pueda hacer pensar que esta es la opción conveniente, si observamos los datos la diferencia entre O0 y O2 apenas es significativa al tratarse de centésimas de segundo en un total de 100000 iteraciones.

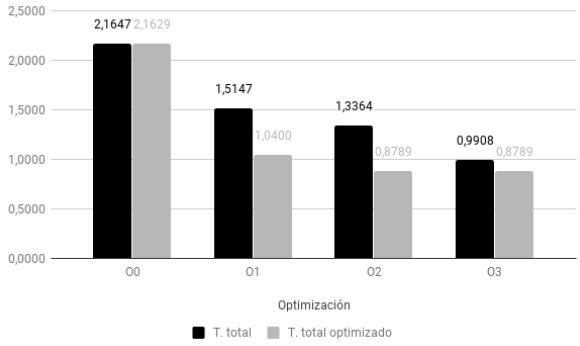
En cuanto la opción -O2, además de lo mencionado anteriormente, agregar que para valores de grandes, supone una mejora del rendimiento con y sin la aplicación de la técnica. Además la mezcla de ambas optimizaciones supone una mejora considerable con respecto a las otras dos opciones anteriores.



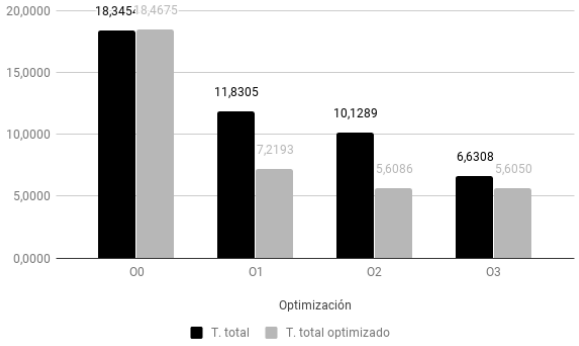
(a) N = 10



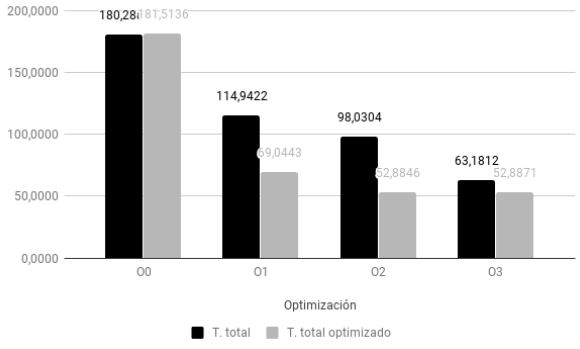
(b) N = 100



(c) N = 1000



(d) N = 10000



(e) N = 100000

Figura 1: Gráficos correspondientes a los tiempos de ejecución en función de N

En el caso de -O3, podemos hablar de dos comportamientos diferenciados. En el caso de N≤100 (tamaños pequeños) el uso de esta opción con el balance de operaciones implica un peor desempeño. Incluso parece que cuanto menor es el valor de N, peor es el desempeño, basta con comparar las barras de O0 y O3 para N=10 y N=100 (1). Es importante recordar que esta opción incluye optimizaciones muy bruscas y que es probable que el balance de operaciones no favorezca al resto de técnicas que aplica el compilador con esta opción.

Por otro lado, para el caso de N>100, se mantiene la misma tendencia mencionada para las otras opciones. En general, se observa que el rendimiento mejora, tanto en el algoritmo en el que no se aplica la técnica como en el que sí, y también en el contraste entre el tiempo de ejecución para el algoritmo sin optimización y el del optimizado. Podemos destacar que la diferencia entre el rendimiento del algoritmo optimizado es apenas considerable para los casos de -O2 y -O3.

En conclusión, como podemos ver en los datos, para valores pequeños de N el uso del balance de operaciones supone una mejora considerable en el rendimiento. Esto se puede deber, principalmente a que es mucho menor el número de comparaciones que se realizan para verificar si es necesario salir del lazo. Al tratarse de vectores relativamente pequeños, el peso que tienen estas comparaciones puede resultar significante frente al tiempo que se tardan en realizar el resto de cálculos. Sin embargo cuando se aplican las opciones de optimización del compilador, el rendimiento empeora. Esto se puede deber a que la predicción es más compleja al tratarse de dos operaciones, aún cuando una es una función lineal que depende del índice del bucle. Así pues, para el compilador puede que resulte más sencillo realizar dichas predicciones cuando se trata de valores de N pequeños, lo cual le permita aplicar técnicas como el desenrollamiento de bucles de forma satisfactoria.

En general, para valores grandes de N podemos ver que el uso del balance de operaciones prácticamente no supone ninguna mejora frente al algoritmo en el que no se aplica esta técnica. Sin embargo, si esta se usa al mismo tiempo que alguna de las opciones del compilador, el rendimiento mejora considerablemente.

En resumen, el uso de una técnica de optimización como puede ser el balance de operaciones es especialmente útil cuando se trata de estructuras de datos con pocos elementos. Dicha ganancia se puede apreciar también en el caso de estructuras con muchos elementos si además se hace uso de las optimizaciones del compilador, en cuyo caso la ganancia apreciada será bastante considerable. Por ejemplo, en el caso de N=100000 y la optimización -O3, el tiempo de ejecución es de menos de un tercio del tiempo que tardaría el mismo algoritmo con la opción -O0.

6. Bibliografía

- *Loop fission and fusion. [online].* Varios Autores (2017).Fecha de acceso: 25 Dec. 2017. `https://en.wikipedia.org/wiki/Loop_fission_and_fusion`.
- *Loop Fusion. [online].* Varios autores (2017). Fecha de acceso: 25 Dec. 2017. `http://www.compileroptimizations.com/category/loop_fusion.htm`