MASTER SIANI ULPGC

Computación Paralela

Paseo por las optimizaciones

Multiplicación de matrices

Sergio Marrero Marrero
Universidad de Las Palmas de Gran Canaria

14/03/2016

Índice

0.1. Trazando el código

Este pequeño paseo de optimización se va a realizar acompañado de la siguiente multiplicación de matrices.

$$C_{n\times n} = B_{n\times n}A_{n\times n} = \begin{pmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{pmatrix} = \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{pmatrix} \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix},$$

Las filas serán denotadas con el subíndice j y las columnas con i. Lo que significa que un elemento de la matriz vendrá mejor representado por c_{ji}. El siguiente código permite multiplicar estas matrices.

```
void dgemm (int n, float* A, float* B, float* C)
{
 int i,j,k;
 for (i = 0; i < n; ++i)
   for (j = 0; j < n; ++j)
   {
    float cij = C[i+j*n]; /* cij = C[i][j] */
    for(k = 0; k < n; k++ )
      cij += A[i+k*n] * B[k+j*n]; /* cij += A[i][k]*B[k][j] */
    C[i+j*n] = cij; /* C[i][j] = cij */
   }
}
```

Veamos la traza del código omitiendo el bucle que mueve la variable k.

i	j	c_{ji}	C[i+j*3]	C[posición]
0	0	c_{00}	C[0+0*3]	C[0]
0	1	c_{10}	C[0+1*3]	C[3]
0	2	c_{20}	C[0+2*3]	C[6]
1	0	c_{01}	C[1+0*3]	C[1]
1	1	c_{11}	C[1+1*3]	C[4]
1	2	c_{21}	C[1+2*3]	C[7]
2	0	c_{02}	C[2+0*3]	C[2]
2	1	c_{12}	C[2+1*3]	C[5]
2	2	c_{22}	C[2+2*3]	C[8]

Cuadro 1: Comprendiendo la traza 1

A continuación vemos dos matrices. La matriz de la izquierda trata de resaltar el orden en el que se van a ir creando los elementos c_{ji}, en la matriz de la derecha se trata de resaltar como los subíndices comienzan en 0. Por otra parte, en la última columna de la tabla 1 se ve las posiciones que ocupan en memoria.

$$OrdenLlenado = \begin{pmatrix} 1^o & 4^o & 7^o \\ 2^o & 5^o & 8^o \\ 3^o & 6^o & 9^o \end{pmatrix} \qquad C_{3\times 3} = \begin{pmatrix} c_{00} & c_{10} & c_{20} \\ c_{01} & c_{11} & c_{21} \\ c_{02} & c_{12} & c_{22} \end{pmatrix}$$

Veamos ahora la traza del bucle que desarrolla el producto de cada para fila-columna, en cada elemento c_{ji} de la matriz C. Supongamos que nos encontramos en el siguiente elemento $j = 1, i = 3$.

k	B[k+1*2]	A[3+k*2]	b_{ji}	a_{ji}
0	B[2]	A[3]	b_{02}	a_{10}
1	B[3]	A[5]	b_{12}	a_{11}
2	B[4]	A[7]	b_{22}	a_{12}

Cuadro 2: Comprendiendo la traza 2

que como se ve, es la multiplicación de la tercera fila de B y la segunda columna de A. Se puede observar también que las filas de B están ordenadas en memoria, sin embargo al recorrer la matriz A por columnas, hay que hacerlo dando saltos de n.

0.2. Cálculo del número de operaciones

En primer lugar vamos a hacerlo para el caso particular que venimos manejando $n = 3$ y los resultados los extrapolaremos a n. Observando la tabla 1, vemos que tiene 9 filas. Es decir, los nueve elementos de C que queremos calculas. Esto es $3x3$ elementos. Por otro lado, observando la tabla 0.1, vemos que tiene 3 filas. Cada una de estas filas implica una mutiplicación entre el elemento de A y de B correspondiente, que para el caso que no ocupa son 3 multiplicaciones. Cada una de estas multiplicaciones se debe de sumas con la siguiente, es decir, tendremos que hacer 2 sumas. Pero si tenemos en cuenta que la primera multiplicación que se incorporó a la sumatoria, tuvo que ser añadida a la variable c_{ji} correspondiente, se verá entonces que en realidad hay 3 sumas y no dos.

Si extrapolamos esta situacion particular de $n = 3$ a n, entonces se tendrá que:

$$flops = n^o elementos * n^o mult + n^o elementos * n^o sum = n * n * n + n * n * n = 2n^3$$

0.3. Introducción a la compilación con optimizacion en GCC

GCC es un compilador con el que se pueden incorporar distintas estrategias de optimización. Suministra un amplio rango de opciones que ayudan a incrementar la velocidad, o reducir el tamaño, de los archivos ejecutables generados.Hay que tener en cuenta que la optimización es un proceso complejo. Para cada comando de alto nivel en el código fuente normalmente hay muchas combinaciones posibles de instrucciones máquina que pueden ser usadas para alcanzar los resultados finales apropiados. El compilador debe considerar todas estas posibilidades y elegir entre ellas.

La primera forma de optimización usada por GCC sucede a nivel de código fuente, y no requiere ningún conocimiento de las instrucciones máquina. Hay muchas técnicas de optimización a nivel de código fuente, como por ejemplo evitar el cálculo repetido de una función, pudiendo previamente descargar su contenido sobre una varible. Es decir, cambiar esto

```
x = cos(v)*(1+sin(u/2)) + sin(w)*(1-sin(u/2))
```

por esto

```
t = sin(u/2)
x = cos(v)*(1+t) + sin(w)*(1-t)
```

A este tipo de optimización se le conoce como 'eliminación de subexpresiones comunes (CSE)' y se realiza de forma automática cuando la optimización está activa. Este tipo de optimización es beneficiosa porque simultáneamente incrementa la velocidad y reduce el tamaño del código. Otras optimización a nivel de fuente es la 'expansión de función en linea' la cual incrementa la eficiencia de la llamada a funciones frecuentes, reduciendo así la llamada 'sobrecarga de la llamada a función'.

Por otro lado hay que tener en cuenta que algunas formas de optimización son capaces simultáneamente de incrementar la velocidad y reducir el tamaño de un programa, mientras que otros tipos de optimización producen código más rápido a costa de incrementar el tamaño del ejecutable.

Para controlar los tiempos de compilación y el uso de memoria, y la compensación entre espacio y velodidad para el ejecutable resultante, GCC suministra un rango de nivelenes generales de optimizaivón, numerados desde 0 a 3, así como opciones individuales para tipos específicos de optimización. El nivel de optimizavión se eligen con la opción '-ONIVEL', donde NIVEL es un número del 0 al 3.

0.3.1. Compilación del código y análisis

0.3.2. Sin optimización o nivel -O

El nivel de optimización '-O0' o sin opción 'O'(por defecto) no realiza ninguna optimización y compila el código fuente de la form más sencilla posible. Cada comando en el código fuente es convertido directamente a sus correspondientes intruciones en el archivo ejecutable, sin reorganización. Esta es la mejor opción a usar cuand se depura un programa y es la opción por defecto si no se espcifica una opción de nivel de opcimización.

Para comprobar los distintos niveles de optimización, se operará con una matriz de n=4096. El número de operaciones necesarias para calcular esta matriz será:

$$flops = 2n^3 = 2 * 4096^3 = 1{,}3744x10^{11} flops$$

Para compilar el código se insertará en consola la siguiente instrucción:

```
gcc -o matrix1sin4096 matrix1sin.c -O
```

Los resultados son los siguientes.

Optimización	-gcc	tiempo(s)	MFlops/s	Archivo (Kb)	Speed up
Sin optimizar	—	1284.4 (21.408)	107.0	8.6	—-

Cuadro 3: Ejecución secuencial sin optimización

0.3.3. Nivel de optimización -O2

Este nivel de optimización añade a las optimizaciones incorporadas en '-01' los métodos de optimización que no impliquen un aumento del tamaño del código. Incluye hasta casi 30 tipos de optimización. En el caso que nos ocupa, las dos optimizaciones que se explicaron al principio se llevarán a cabo en el proceso de optimización.

Probablemente, por las características del código fuente implementado, el método de optimización fruto de la compilación con -O2 que más provoque optimización del código, será la expansión de las funciones, ya que se eliminarán las llamadas 'sobrecarga de la llamada a la función'.

A continuación se volverá a compilar el archivo anterior, pero ahora introduciendo la siguiente linea en consola para su compilación.

Para compilar el código se insertará en consola la siguiente instrucción:

```
gcc -o matrix2serial4096 matrix1sin.c -O2
```

Los resultados son los siguientes.

Optimización	-gcc	tiempo(s)	MFlops/s	Archivo (Kb)	Speed up
Sin optimizar	—	1284.4 (21.408)	107.0	8.7	1
-O2	—	739.6 (12.327)	185.8	8.7	1.73

Cuadro 4: Ejecución secuencial con optimización -O2

Se puede observar en la tabla 4 que hay una optimización bastante grande con respecto al inicial. Sin embargo, de ahora en adelante, el que se utilizará como referencia será la versión mas optimizada de la compilación secuencial, y este es el -O2.