MASTER SIANI ULPGC

Computación Paralela

Paseo por las optimizaciones

Optimización basada en el paralelismo de instrucciones

Sergio Marrero Marrero
Universidad de Las Palmas de Gran Canaria

14/03/2016

Índice

1. Ejercicio 3. Paralelismo de Instrucciones. Desenrollado

En este apartado tenemos dos optimizaciones superpuestas. Por una parte tenemos el desenrollado de bucles(UNROLLING) y por otra parte el paralelismo de instrucciones.

1.1. Paralelismo de instrucciones o segmentación

La segemenctación es una técnica que nos permite solapar instrucciones. Para explicar el concepto se utilizará la analogía de cómo hacer varias coladas de ropa.

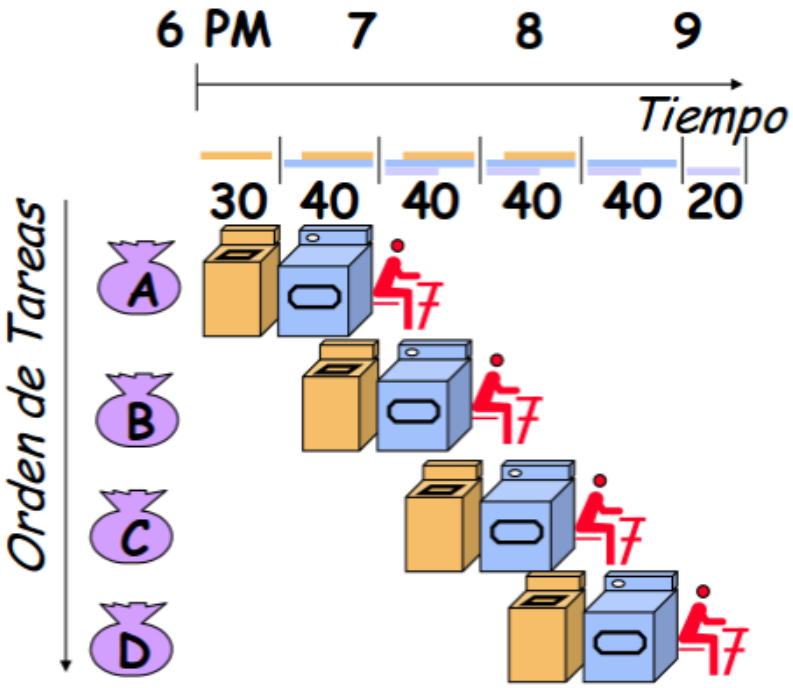

Figura 1: Paralelismo en la colada

En esta imagen se puede apreciar lo siguiente. Las tareas A,B,C,D se deben de terminar lo antes posible. Para ello se dispone de una lavadora, una secadora y una plancha. El método consiste en utilizar cada herramienta en

cuanto se quede libre.De manera simplificada se puede decir que las tareas básicas de una ordenador son las siguientes.

1. Búsqueda de las instrucción
2. Lectura de registros mientras se decodifica la instrucción
3. Ejecución de la operación o cálculo de una dirección
4. Acceso a un operando en la memoria de datos.
5. Escritura del resultado en un registro.

Si hacemos la analogía de la colada con las tareas que realiza el procesador podríamos conseguir optimizar los recursos del mismo. **Hay que tener en cuenta que la segmentación incrementa el rendimiento incrementando la productividad (throughput), en lugar de reducir el tiempo de ejecución de cada instrucción individual.**. Obviamente, el ritmo estará limitado por la ruta hardware más lenta.

Por otro lado, los procesadores superescalares permiten ejecutar más de una instrucción a la vez en cada ciclo de reloj. Volviendo a la analogía, es como si la lavadora permitiera ejecutar muchas coladas de una sola vez.

1.2. Desenrollamiento de bucles.Unrolling

Esta forma de optimización incrementa la velocidad de los bucles eliminando la condición de terminación del bucle en cada iteración. Por ejemplo, el siguiente bucle de 0 a 7 comprueba la condición i <8 en cada iteración:

```
for (i = 0; i < 8; i++)
{
y[i] = i;
}
```

Al final del bucle, esa condición ha sido comprobada 9 veces, y gran parte del tiempo de ejecución fue gastado en la comprobación.

Sin embargo, al desenrollarlo podemos ejecutar las instrucciones directamente:

```
y[0] = 0;
y[1] = 1;
y[2] = 2;
y[3] = 3;
y[4] = 4;
y[5] = 5;
y[6] = 6;
y[7] = 7;
```

Este tipo de código no requiere ningún test, y se ejecuta a la máxima velocidad. La única pega sería que su uso viene acompañado de un aumento en el tamaño del archivo.

Por otro lado, y es aquí donde engancha con la segmentación, al desenrrollar el bucle permite descubrir a ojos del compilador las operaciones que son independientes y por esta razón, por ser independientes, podrán ser segmentadas (paralelizadas).

Una vez que cada asignación es independiente, permite ser compialdo para usar procesamiento.

1.3. Compilación del código y análisis

El código que se deberá implementar es el siguiente:

```
void dgemm (int n, float* A, float* B, float* C)
{
 int i,j,k,x;
 for (i = 0; i < n; i+=UNROLL*4)
    for (j = 0; j < n; j++)
    {
          __m128 c[4];
          for (x=0; x < UNROLL; x++) c[x]=_mm_load_ps(C+i+x*4+j*n);
          for (k=0;k<n;k++) {
          __m128 b=_mm_load_ps1(B+k+j*n); /*replica 4 veces B[k][j]*/
          for (x=0;x<UNROLL;x++)
          c[x]=_mm_add_ps(c[x],/*c[x] += A[i+x*4][k]*B[k][j]*/_mm_mul_ps
    }
 for ( x=0;x<UNROLL;x++)
```

```
            _mm_store_ps(C+i+x*4+j*n, c[x]); /*guarda C[i+x*4][j]*/
    }
}
```

Se puede observar que en este código aparece la variable *x*. Esta variable nos permite ir desenrrollando los bucles e ir destapando aquellas instrucciones que son independientes entre sí. Por otro lado, se observa la variable UNROLL, la cual contiene el numero de instrucciones a desenrrollar.

La forma de compilar esta función será la siguiente:

```
gcc -msse -o matrix4desenrollamiento4096 matrix4desenrollamiento4096.
```

Se observa que no han cambiado las instrucciones respecto de la anterior forma de compilar. Esto es debido a dos cosas, por una parte la optimización anterior se sigue realizando, y por ende es necesario la subinstrucción *-msse* y por otro lado,como ya se dijo, la instrucción *-O3* ya es el grado máximo de optimización que incluye el compilador GSS. Esto significa que *-O3* incluye todos los métodos de optimización del compilador.

A continuación se exponen los resultados obtenidos, comparandolo con la mejor versión secuencial, la cual se utilizará de referencia, como ya se apuntó anteriormente:

Optimización	-gcc	tiempo(s)	MFlops/s	Archivo (Kb)	Speed up
Sin optimizar	-O	1284.4 (21.408)	107.0	8.7	—-
-O2	-O2	739.6 (12.327)	185.8	8.7	1
Par. datos	msse,-O3	174.9 (2.92)	785.7	8.7	4.2287
Segmentación	-O3	65.1 (1.09)	2111.5	8.7	11.361

Cuadro 1: Ejecución con segmentacion o paralelismo de instrucciones

Se observa que el fichero ejecutable no ha aumentado su tamaño.