Práctica 1 - GEMM

Computación paralela y cálculo distribuido

Mario García Mayo

1. GEMM secuencial y validador

El cálculo del producto de matrices generalizado secuencial y el validador están implementados en las funciones `producto` y `validador`.

2. GEMM paralelo

En la primera versión del GEMM en paralelo se ha paralelizado el primer bucle en

```
for i in range(m):
    for j in range(n):
        for k in range(K):
```

De este modo, cada thread se ocupa de calcular un determinado número de filas de la matriz de forma independiente. Para ello, se ultiliza la función `calcular_filas`.

En la segunda versión se ha paralelizado el primer bucle en

```
for j in range(n):
    for i in range(m):
        for k in range(K):
```

Funciona de forma similar a la primera versión, pero esta vez los threads van calculando columnas de la matriz en lugar de filas. Para calcularlas se usa la función `calcular_cols`.

En la tercera versión, se ha paralelizado el tercer bucle en

```
for k in range(K):
    for j in range(n):
        for i in range(m):
```

Es decir, una vez que tenemos fijados un k y un j cada thread se va ocupando de calcular ciertos elementos $a_{ik} \cdot b_{kj}$ de la columna j. Para calcularlos se usa la función `calcular_filas_col`. Sin embargo, esta forma de paralelizar el producto de matrices es muy lenta ya que la carga que se le asigna a cada proceso es muy baja y hay que inicializar los procesos muchas veces, luego el tiempo que se requiere para tantas inicializaciones no nos compensa.

3. Casos en los que no se puede paralelizar

Si intentamos paralelizar uno de los bucles correspondientes a K, el número de columnas de A y de filas de B, el resultado que obtendríamos sería incorrecto. Veamos que pasa si tratamos de paralelizar el tercer bucle en

```
for i in range(m):
    for j in range(n):
        for k in range(K):
```

Tras fijar la fila i y la columna j con la que estamos trabajando, cada thread se ocuparía de calcular uno de los productos $a_{ik} \cdot b_{kj}$ $(k = 1, \ldots, K)$ y de sumarlo al elemento c_{ij}.
Sin embargo, al no realizarse estas operaciones de forma secuencial lo que sucedería es que todos los threads tratarían de obtener el valor de c_{ij} y si uno accediese a este valor antes de que el anterior hubiese terminado de modificarlo entonces estaría tomando un valor no actualizado de c_{ij} y dicha modificación se perdería.

Alternando el orden de los bucles pero paralelizando igualmente en el de K sucedería algo similar solo que en vez de tomarse valores no actualizados de un solo elemento de la matriz se tomarían valores no actualizados de filas, columnas o de la matriz entera (cada thread se iría ocupando de sumar $a_{ik} \cdot b_{kj}$ a todos los elementos de una fila, una columna o de toda la matriz).

4. Análisis de escalabilidad

Para realizar el análisis de estabilidad, tomaremos medidas con matrices 500×500 utilizando la primera versión del GEMM paralelo. Tras ejecutar 5 veces el algoritmo con 16, 8, 4, 2 y 1 thread obtenemos los siguientes tiempos paralelos:

Cantidad de hilos	Tp_1	Tp_2	Tp_3	Tp_4	Tp_5	MEDIA
1 (secuencial)	54.2158	53.0852	54.8534	54.0066	53.4651	53.9252
2	27.3620	28.1996	27.9723	27.6287	28.1779	27.8681
4	14.9247	15.2708	15.3352	15.4176	15.3602	15.2617
8	10.3967	10.4570	10.5746	10.6909	10.6687	10.5576
16	10.2091	10.2126	10.1893	10.2587	10.1987	10.2137

En la siguiente gráfica se ilustra la escalabilidad observada.

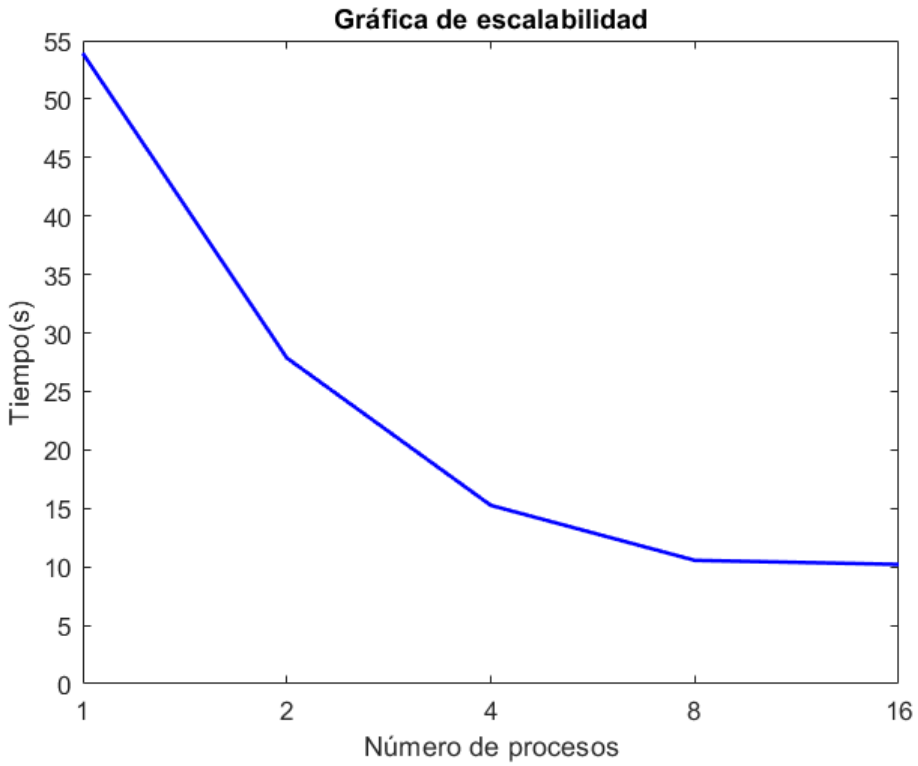

Figura 1: Escalabilidad

5. Speedup y eficiencia

A continuación vamos a calcular el speedup S y la eficiencia E usando los datos registrados en la sección anterior. Recordamos que vienen dados por

$$S = \frac{T_s}{T_p}, \qquad E = \frac{S}{P}$$

donde T_p es el tiempo paralelo, T_s el tiempo secuencial y P el número de procesos.

Cantidad de hilos	Speedup	Eficiencia
1 (secuencial)	1	1
2	1.9350	0.9675
4	3.5334	0.8833
8	5.1077	0.6385
16	5.2797	0.3300

6. Conclusiones

A la hora de paralelizar un algoritmo, tenemos que analizar primero qué bucles podemos paralelizar y cuáles no y luego ver si todas las posibles formas de paralelizar son realmente útiles. En el caso del GEMM, hemos visto que no se pueden paralelizar los bucles correspondientes a K y que si paralelizamos los bucles interiores el tiempo de ejecución puede llegar a ser muy alto.

Respecto a la escalabilidad, observamos un mayor speedup cuantos más threads usemos, pero a su vez la eficiencia va siendo cada vez más pequeña. Además, llega un punto en el que el tiempo de ejecución que obtenemos parece que apenas se reduce al aumentar el número de threads (con 8 y con 16 la diferencia es de décimas de segundo), aunque cuánto más grandes sean las matrices más se tarda en alcanzar este estancamiento.