PRÁCTICA 1

Multiplicación de matrices de forma distribuida

Marc Provinciale Isach

Aaron Murillo Lort

Sistemas Distribuidos 19/20

Contenido

1. Introducción

En esta práctica, observaremos la mejora que supone procesar datos de forma masiva en la nube, haciendo uso del paralelismo que nos ofrece esta. Para poder hacerlo, haremos un cálculo simple: la multiplicación de dos matrices en la nube. Sin embargo, para poder llevar a cabo la comparación, lo haremos de dos formas: primero, llevaremos a cabo la operación de forma secuencial y, después, ese mismo cálculo de forma paralela haciendo uso de las *functions* ofrecidas por IBM Cloud.

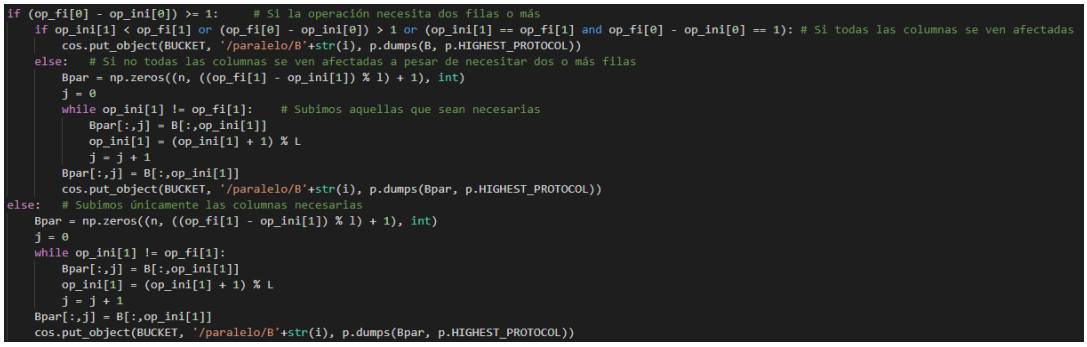
2. Solución implementada

Para implementar la multiplicación de matrices de forma distribuida hemos pensado en dos formas distintas de subir los datos.

En ambos casos solo se sube la cantidad necesaria de la matriz A que necesitará cada worker, las filas que va a calcular de C. Pero, en el primer enfoque, se sube completamente la matriz B en un solo paquete y cada worker va a descargarla entera, ya que casi siempre va a calcular mínimo una fila entera de C y por lo tanto va a necesitar todo B para calcularla; en el segundo enfoque, la matriz B se sube entera cuando un worker calcule una fila entera de C o más, y si calcula menos, solo descargará y subirá las columnas que va a necesitar: esto puede favorecer al rendimiento cuando el número de workers sea elevado y haya pocas operaciones a hacer.

A cada worker se le pasará el rango de qué posiciones de C tiene que calcular, y devolverá una lista con los resultados de las posiciones que haya calculado.

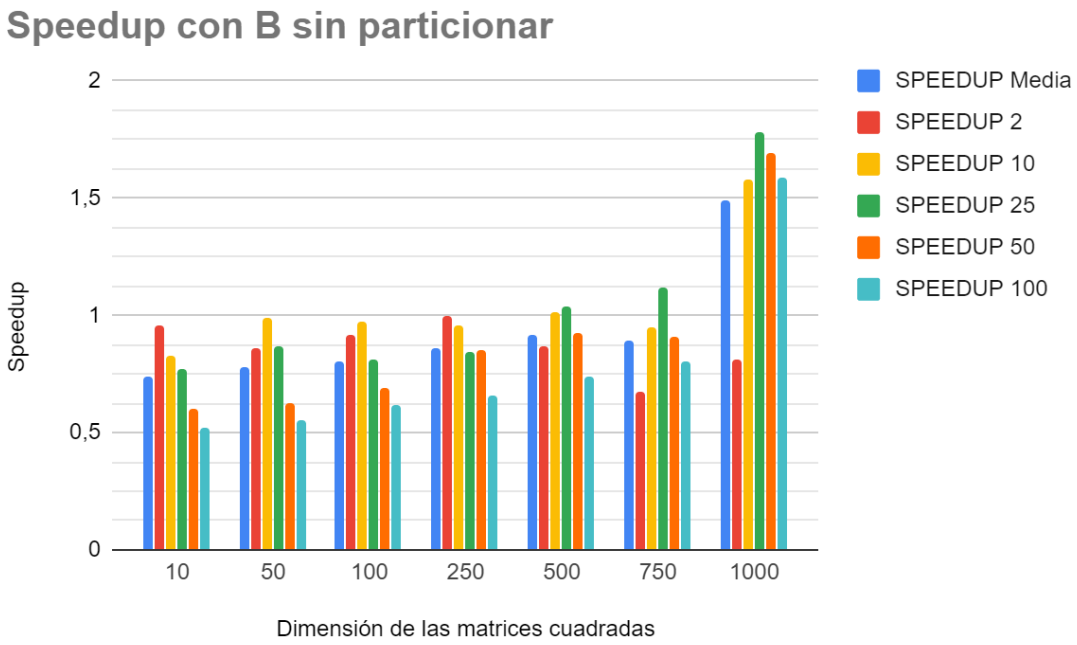
Por último, a la hora de juntar los resultados de todos los workers se recorre la matriz C posición por posición y se van poniendo los resultados de cada worker de forma ordenada.



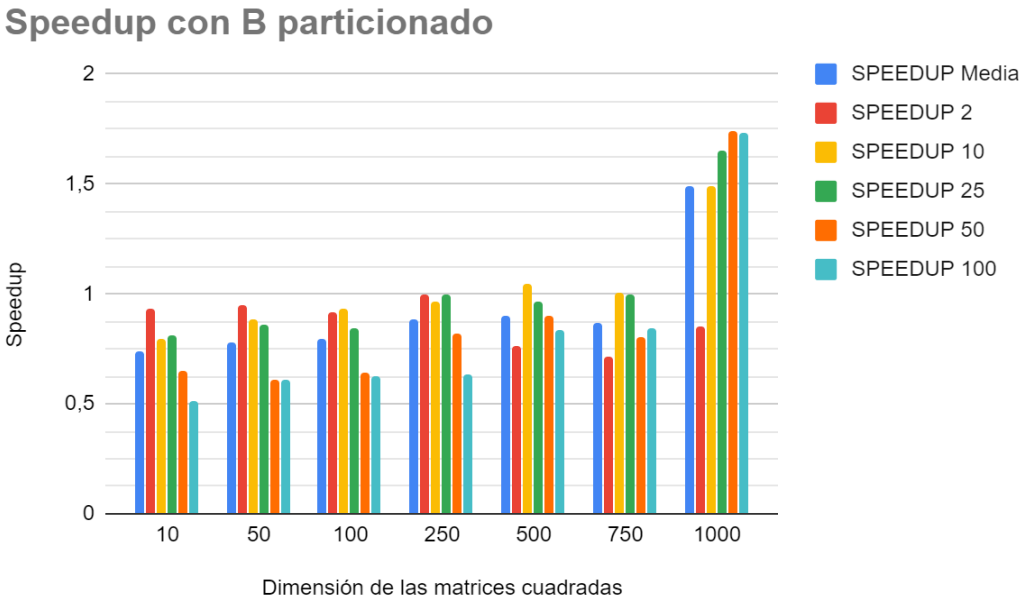
```
if (op_fi[0] - op_ini[0]) >= 1:     # Si la operación necesita dos filas o más
    if op_ini[1] < op_fi[1] or (op_fi[0] - op_ini[0]) > 1 or (op_ini[1] == op_fi[1] and op_fi[0] - op_ini[0] == 1): # Si todas las columnas se ven afectadas
        cos.put_object(BUCKET, '/paralelo/B'+str(i), p.dumps(B, p.HIGHEST_PROTOCOL))
    else:   # Si no todas las columnas se ven afectadas a pesar de necesitar dos o más filas
        Bpar = np.zeros((n, ((op_fi[1] - op_ini[1]) % l) + 1), int)
        j = 0
        while op_ini[1] != op_fi[1]:    # Subimos aquellas que sean necesarias
            Bpar[:,j] = B[:,op_ini[1]]
            op_ini[1] = (op_ini[1] + 1) % L
            j = j + 1
        Bpar[:,j] = B[:,op_ini[1]]
        cos.put_object(BUCKET, '/paralelo/B'+str(i), p.dumps(Bpar, p.HIGHEST_PROTOCOL))
else:   # Subimos únicamente las columnas necesarias
    Bpar = np.zeros((n, ((op_fi[1] - op_ini[1]) % l) + 1), int)
    j = 0
    while op_ini[1] != op_fi[1]:
        Bpar[:,j] = B[:,op_ini[1]]
        op_ini[1] = (op_ini[1] + 1) % L
        j = j + 1
    Bpar[:,j] = B[:,op_ini[1]]
    cos.put_object(BUCKET, '/paralelo/B'+str(i), p.dumps(Bpar, p.HIGHEST_PROTOCOL))
```

Creación y subida de paquetes de B

3. Resultados



Por un lado, como podemos observar en el gráfico, el tiempo de respuesta no mejora de forma considerable hasta llegar a matrices de 1000x1000, donde hay una subida notable del speedup relativo a la versión secuencial. Es decir, con un gran tamaño de datos, el paralelismo acelera el cálculo a llevar a cabo.



Por otro lado, podemos ver que, aún subiendo B de forma particionada y únicamente con las columnas requeridas por cada worker, el resultado es casi idéntico.

También observamos que no es necesario hacer uso de muchos workers para poder notar una mejora considerable: por ejemplo, con 25 workers obtenemos una mejora del Speedup prácticamente igual que con 50 workers o con 100, ya que el rendimiento se acaba estabilizando debido a que la creación de los workers empeora el rendimiento en lugar de mejorarlo.

4. Conclusiones

Sin duda, el paralelismo ofrecido por estos servicios de la nube ofrece una mejora notable, pero con ciertos matices. Con matrices de dimensiones pequeñas o no significativas, el paralelismo no ofrece ninguna mejoría, es más, el tiempo de respuesta empeora. Dado que tenemos muchos workers cuyas operaciones a llevar a cabo son escasas, si sumamos el tiempo que tarda cada uno en descargar los datos necesarios, hacer el cálculo que debe hacer y luego el tiempo dedicado a reunir los resultados, podemos deducir que es más fácil hacer el cálculo secuencial. Al fin y al cabo, paralelizar un trabajo con datos tan absurdamente pequeños es un sinsentido teniendo en cuenta que la mayoría de nuestros computadores son capaces ya de llevar a cabo dichos cálculos rápidamente y de golpe.

Sin embargo, a partir de matrices de 1000x1000, es decir, matrices con un millón de elementos o más, podemos notar el efecto del paralelismo. Ahora, podemos separar la matriz en matrices tan pequeñas como las matrices de las que hablábamos en el párrafo anterior para poder así agilizar el trabajo y calcular distintos fragmentos de C de forma paralela, sin tener que operar con tantísimos datos a la vez y de forma secuencial.

En síntesis, la manipulación de datos de forma paralela en la nube conlleva sin duda una mejoría al realizar cualquier cálculo, pero sólo cuando la cantidad de datos implicados en dicho cálculo es considerablemente alta.

5. Anexo

Tablas mediante las cuales hemos podido calcular el speedup relativo y hemos podido recopilar los datos requeridos:

Secuencial	Tiempo(Media)
10	2,908080912
50	2,993372488
100	2,929051685
250	3,542289066
500	4,274890184
750	5,278008366
1000	11,013474178

Paralelo_B	W(2)	W(10)	W(25)	W(50)	W(100)
10	3,031006384	3,506559324	3,760076141	4,825938702	5,611126852
50	3,484849358	3,034225368	3,457050085	4,784439850	5,422704554
100	3,195803213	3,009960985	3,626043081	4,250720215	4,768288088
250	3,557226896	3,711977196	4,187816286	4,165683413	5,369305325
500	4,920792723	4,229841042	4,119254017	4,642002773	5,801988459
750	7,833088064	5,567683363	4,723071527	5,816294527	6,567350960
1000	13,505802250	6,971541977	6,177143717	6,523207283	6,929969645

Media Workers	SPEEDUP Media	SPEEDUP 2	SPEEDUP 10	SPEEDUP 25	SPEEDUP 50	SPEEDUP 100
4,146941481	0,7366088748	0,9594440075	0,8293260267	0,7734101125	0,6025938354	0,5182703917
4,036653843	0,7778066054	0,8589675423	0,9865359769	0,8658747818	0,6256474283	0,5520072979
3,770163116	0,8001562522	0,9165306779	0,9731194855	0,8077818216	0,6890718601	0,614277416
4,198401823	0,8612044906	0,9958007093	0,9542863222	0,8458558887	0,8503500424	0,6597294905
4,742775803	0,9149771088	0,8687401452	1,010650316	1,03778261	0,9209150432	0,7367974299
6,101497688	0,8900804596	0,6738093996	0,9479720777	1,117494905	0,9074520455	0,8036738706
8,021532974	1,491156691	0,8154624194	1,579775925	1,782939605	1,688352631	1,589252875

Paralelo_Paquetes	W(2)	W(10)	W(25)	W(50)	W(100)
10	3,106415606	3,641148996	3,572863150	4,486912203	5,674617052
50	3,144047070	3,389407063	3,483372116	4,884955597	4,941026402
100	3,201241541	3,138384199	3,463172150	4,587442541	4,661104441
250	3,546940374	3,656792212	3,538486338	4,338496161	5,609669399
500	5,598343801	4,102518511	4,429231310	4,765976381	5,133528852
750	7,373544455	5,251735020	5,301510477	6,565959644	6,283577251
1000	12,898979378	7,401332378	6,670751905	6,335897970	6,367329741

Media Workers	SPEEDUP Media	SPEEDUP 2	SPEEDUP 10	SPEEDUP 25	SPEEDUP 50	SPEEDUP 100
4,096391401	0,7418714178	0,9361532006	0,7986712201	0,813935712	0,6481252095	0,5124717465
3,968561649	0,7826313926	0,9520762323	0,8831552047	0,8593318165	0,6127737353	0,6058199744
3,810268974	0,7921881118	0,9149736588	0,933299271	0,8457713214	0,6384933782	0,6284029295
4,138076897	0,8832781408	0,9986886422	0,968687544	1,001074677	0,8164785528	0,6314612884
4,805919771	0,9000936428	0,7635990814	1,042016062	0,9651539704	0,896960002	0,8327390977
6,155265369	0,8720372023	0,7158034237	1,005002794	0,9955669028	0,8038441677	0,8399687239
7,934858274	1,492164934	0,8538252412	1,488039398	1,651009412	1,738265709	1,729684911

6. Webgrafía

1. Github con ficheros y archivos informativos de PyWren (2018): https://github.com/pywren/pywren-ibm-cloud
2. Manual de la librería Numpy (2018): https://docs.scipy.org/doc/numpy-1.15.1/index.html
3. Serialización y deserialización de objetos con Pickle (2001-2020): https://docs.python.org/3/library/pickle.html