Paralelización del algoritmo de multiplicación de matrices utilizando MPI

Documento de análisis

Presentado por: Juan Camilo Rojas Cortés

De forma similar a las entregas de talleres anteriores, en el presente ejercicio práctico se pretendía implementar el algoritmo de multiplicación de matrices de forma paralela. Para este caso particular se requería implementarlo utilizando MPI en una infraestructura computacional de tipo clúster. Por lo tanto, antes de explicar la implementación, se hará una breve explicación de la arquitectura de hardware utilizada:

Infraestructura computacional:

Para realizar la presente implementación se decidió utilizar la infraestructura computacional provista Amazon Web Services (aws). En este caso se montó virtualmente un clúster homogéneo con 5 máquinas, una con rol de maestro y 4 con role de trabajadores (worker nodes). Al ser un clúster homogéneo, todas las instancias de equipos asociadas tienen las mismas características de hardware y software. En este caso, las características utilizadas fueron las siguientes:

- **Núcleos de procesamiento por máquina:** 1.
- **Memoria RAM:** 1 GB.
- **Disco duro:** 8 GB.
- **Sistema operativo:** Ubuntu 18.04 (Linux/UNIX).

Implementación secuencial:

Para tener un parámetro de referencia respecto a la eficiencia de las implementaciones realizadas, se decidió montar la implementación secuencial hecha en entregas anteriores a la infraestructura computacional utilizada, ya que es la primera vez que se usa esta infraestructura en las entregas de la materia. Para medir el rendimiento se utilizó la misma metodología que en las entregas anteriores: Se lanzó un script que, en 10 iteraciones, ejecuta el algoritmo utilizando matrices de 10x10, 100x100, 200x200, 500x500, 1000x1000 y 2000x2000, midiendo el tiempo en cada una de estas ejecuciones. De esta forma se tiene una buena base de resultados para realizar un análisis. Los resultados de rendimiento para la implementación secuencial fueron los siguientes:

Implementación secuencial (tiempo en segundos)							
Iteración\Tamaño	**10**	**100**	**200**	**500**	**800**	**1000**	**2000**
1	0,000007	0,003854	0,031623	0,600099	3,21237	5,816839	125,35808
2	0,000006	0,003792	0,032324	0,600369	3,219024	5,797745	128,528061
3	0,000006	0,003786	0,032105	0,589271	3,204909	5,779063	125,511526
4	0,000005	0,003759	0,030914	0,583808	3,155053	5,79234	130,189123
5	0,000006	0,003762	0,031071	0,595891	3,100446	5,899994	125,591591
6	0,000005	0,003872	0,032325	0,598548	3,149181	5,842917	132,691128
7	0,000005	0,00376	0,031504	0,604328	3,039555	5,769106	135,160676
8	0,000005	0,003864	0,032744	0,600185	3,299282	5,809939	130,994478
9	0,000006	0,003809	0,033071	0,574759	3,197516	5,771828	128,613129
10	0,000006	0,003955	0,032773	0,58675	3,206256	5,792664	126,28311
Promedio	0,0000057	0,0038213	0,0320454	0,5934008	3,1783592	5,8072435	128,89209

Tabla 1. Tiempos de ejecución de la implementación secuencial

Implementación con MPI utilizando comunicación punto a punto

La primera implementación paralela con MPI realizada fue hecha haciendo uso del esquema de comunicación punto a punto. En este esquema se realiza el envío individual de datos desde el master a cada uno de los worker nodes para que estos realicen su parte respectiva. En el contexto de MPI, esto se hace utilizando las funciones *MPI_Send* y *MPI_Recieve*. Respecto a la técnica de paralelización utilizada, se decidió dividir las filas de las matrices a multiplicar en partes iguales según los worker nodes que había (4 en este caso). Por ejemplo, si se quería realizar el ejercicio con matrices de 1000x1000, en este caso el nodo master se encargará de repartir 250 filas a cada uno de los worker nodes y estos realizarán el procesamiento de la parte que les corresponde. En caso de que la división de filas no diera un resultado entero, el residuo de esta división se reparte equitativamente entre todos los worker nodes. Por ejemplo, si en el caso anterior no fuesen matrices de 1000x1000 sino de 1003x1003, entonces se añadirá una fila más a los workers 1, 2 y 3. De esta forma se garantiza que todos los worker nodes tendrán una carga computacional similar.

A continuación, se presentan los resultados de tiempos obtenidos en la presente implementación:

Implementación con comunicación punto a punto (tiempo en segundos)							
Iteración\Tamaño	**10**	**100**	**200**	**500**	**800**	**1000**	**2000**
1	0,005752	0,008412	0,022005	0,213017	0,770255	1,689401	35,789894
2	0,005998	0,007549	0,020705	0,221286	0,762715	1,595179	32,022669
3	0,005997	0,007274	0,019536	0,21952	0,758253	1,552125	30,080351
4	0,005179	0,006098	0,020504	0,222214	0,756869	1,53344	30,317595
5	0,006123	0,007344	0,020433	0,220045	0,784072	1,537603	30,093357
6	0,003913	0,007291	0,019871	0,213193	0,779903	1,571785	30,094807
7	0,005715	0,00633	0,021257	0,220712	0,790195	1,654246	30,248235
8	0,005257	0,007056	0,021711	0,212178	0,789538	1,55668	34,882881
9	0,005512	0,0081	0,021211	0,220973	0,799789	1,683555	30,406013
10	0,0051	0,007758	0,02169	0,223284	0,795911	1,606236	30,2276
Promedio	0,0054546	0,0073212	0,0208923	0,2186422	0,77875	1,598025	31,4163402

Tabla 2. Registro de tiempos para la implementación con comunicación punto a punto

Para tener una mayor claridad en la mejora de rendimiento de esta implementación respecto a la versión secuencial del algoritmo, se calculó el Speed Up Rate con los tiempos obtenidos. Los resultados de Speed Up Rate se presentan en la siguiente gráfica:

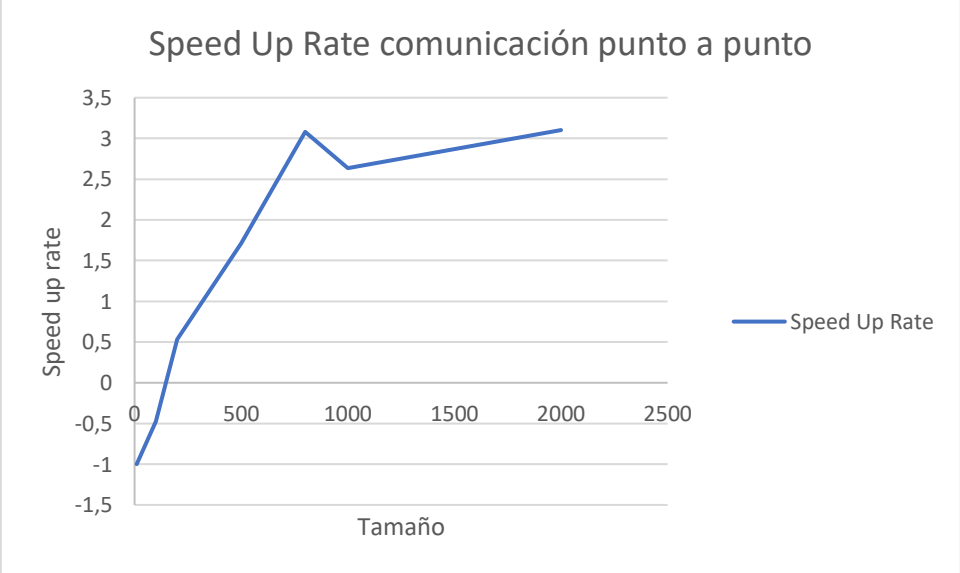

Gráfico 1. Speed Up Rate para la implementación con comunicación punto a punto

Como se puede ver en el Gráfico 1, la mejora en el rendimiento utilizando paralelización con comunicación punto a punto fue notable respecto a la implementación secuencial. Por ejemplo, para el caso de la ejecución con matrices de 2000 filas y 2000 columnas se llegó a un Speed Up Rate superior a 3. Esto indica que el algoritmo se ejecutó un 300% más rápido si se le compara con el algoritmo secuencial para el mismo tamaño de matrices.

Implementación con MPI utilizando comunicación colectiva

La segunda implementación realizada para esta entrega se hizo utilizando las herramientas de comunicación colectiva provistas por MPI. En este caso no se hace necesario repartir de manera separada los dato entre los miembros del clúster, sino que estos se pueden distribuir de una sola vez a través de ciertas funciones de MPI tales como *MPI_Bcast*, *MPI_Scatter* o *MPI_Gather*. Para esta implementación se eligió un método de repartición de las matrices parecido al utilizado en la implementación con comunicación punto a punto. Sin embargo, dadas las características de la comunicación colectiva, en este caso es más sencillo hacer que el nodo master también ayude a ejecutar procesamiento del algoritmo, mientras que en la implementación con comunicación punto a punto este solo se encargaba de repartir tareas. Por lo tanto, para esta implementación puntual (teniendo en cuenta la infraestructura computacional descrita anteriormente) se podía contar con 5 nodos haciendo el cálculo de la multiplicación, mientras que para el caso de la comunicación punto a punto el procesamiento se realizaba solo entre 4 instancias.

A continuación, se presentan los resultados de tiempos obtenidos en la presente implementación:

Implementación con comunicación colectiva (tiempo en segundos)							
Iteración\Tamaño	**10**	**100**	**200**	**500**	**800**	**1000**	**2000**
1	0,007663	0,009093	0,017842	0,173054	0,607777	1,267979	23,9496
2	0,006225	0,008198	0,016031	0,16802	0,600222	1,251438	23,988629
3	0,005442	0,010687	0,016751	0,170795	0,676361	1,270324	24,651728
4	0,007718	0,009001	0,016464	0,169152	0,613309	1,257804	24,026757
5	0,005542	0,009179	0,015993	0,169316	0,61259	1,252845	24,16076
6	0,00773	0,01061	0,016854	0,169544	0,610194	1,247178	24,159928
7	0,005258	0,008464	0,018594	0,172482	0,642008	1,254865	24,05797
8	0,007326	0,008335	0,016827	0,171945	0,631857	1,233188	23,936221
9	0,005408	0,010433	0,019143	0,165308	0,61816	1,268221	24,041534
10	0,007353	0,008286	0,016423	0,172854	0,60864	1,24445	24,019681
Promedio	0,0065665	0,0092286	0,0170922	0,170247	0,6221118	1,2548292	24,0992808

Tabla 3. Registro de tiempos para la implementación con comunicación colectiva

Para tener una mayor claridad en la mejora de rendimiento de esta implementación respecto a la versión secuencial del algoritmo, se calculó el Speed Up Rate con los tiempos obtenidos. Los resultados de Speed Up Rate se presentan en la siguiente gráfica:

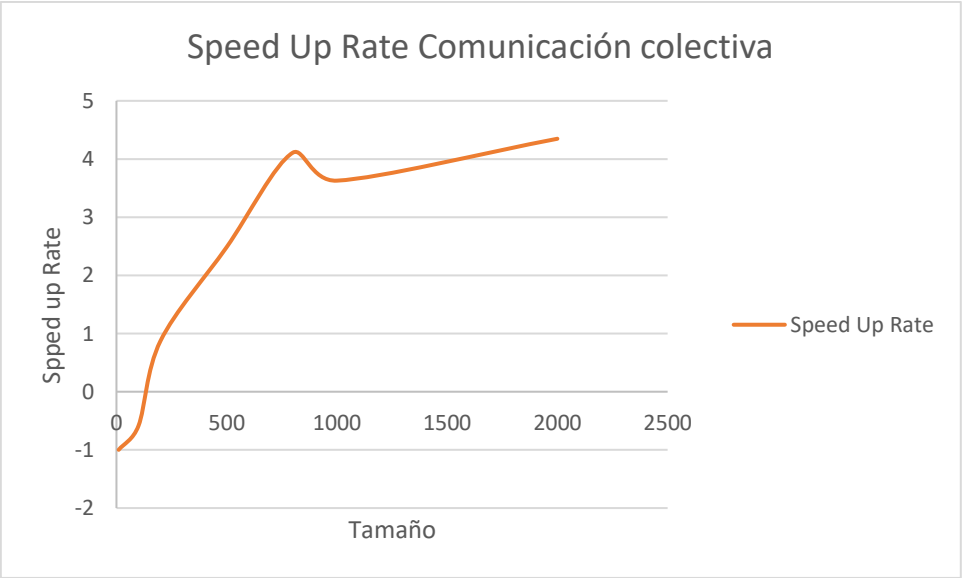

Gráfico 2. Speed Up Rate para la implementación con comunicación colectiva

Como se puede ver en el Gráfico 2, la mejora de rendimiento del algoritmo ejecutado con comunicación colectiva fue notable respecto a la implementación secuencial, pues se llegó a alcanzar un Speed Up Rate máximo superior a 4. Esto indica que, para el caso de matrices de 2000x2000, este algoritmo se ejecuta un 400% más rápido que el algoritmo secuencial.

Análisis comparativo y conclusiones

Para determinar las ventajas o desventajas de utilizar paralelización con comunicación punto a punto o colectiva, se realizó un gráfico comparativo con los Speed Up Rate obtenidos por cada una de las implementaciones. Este gráfico se presenta a continuación:

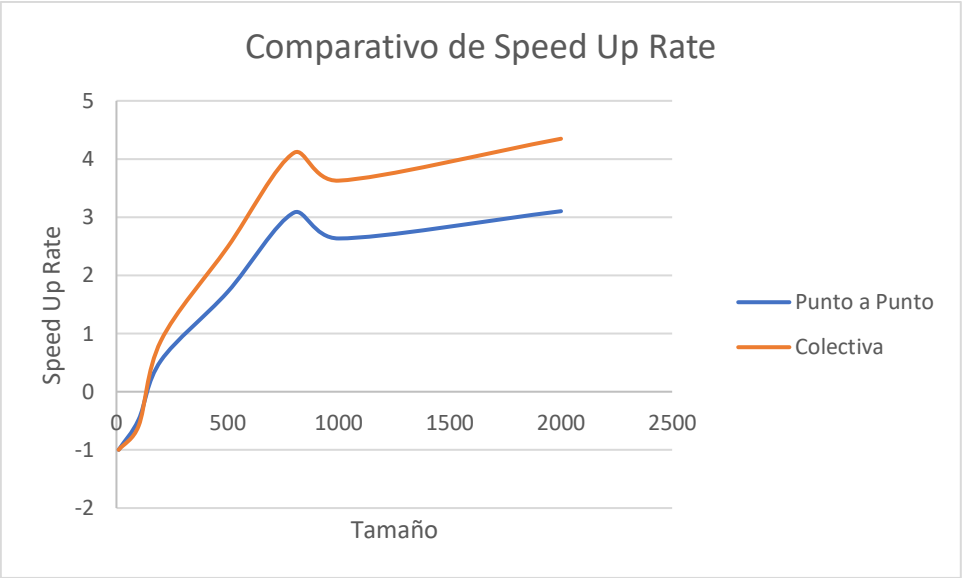

Gráfico 3. Comparativo de Speed Up Rate para las dos implementaciones hechas

Como se puede apreciar en el Gráfico 3, el Speed Up Rate de las dos implementaciones hechas tiene un comportamiento notablemente similar. Sin embargo, la implementación con comunicación colectiva alcanza valores superiores por un buen margen. Por ejemplo, para el caso de las matrices de 2000 filas y 2000 columnas, el Speed Up Rate con comunicación colectiva fue un 33% mayor que el Speed Up Rate obtenido por la implementación con comunicación punto a punto. Esta diferencia puede deberse, entre otras cosas, a que en la comunicación colectiva no es necesario repartir individualmente los procesos entre los worker nodes y esperar una respuesta de estos, pues la repartición se da de manera automática y en un solo paso. Además, para este caso particular en la implementación colectiva se pudo hacer procesamiento con 5 nodos, mientras que en la comunicación punto a punto solamente se realizaron los cálculos con los 4 worker nodes, pues el nodo master únicamente se ocupó de repartir la memoria y recibir los resultados finales.

Sin importar el tipo de comunicación elegida, este trabajo permite concluir que la paralelización con memoria distribuida y paso de mensajes puede llegar a tener un rendimiento de calidad considerable. Por ejemplo, las implementaciones hechas con memoria compartida en un solo equipo a través de hilos o procesos del sistema operativo no llegaron a superar nunca un Speed Up Rate de 2, mientras que con estas implementaciones hechas con MPI se llegó a superar un Speed Up Rate de 4. Además, este rendimiento puede seguir creciendo con el número de instancias que se añadan al clúster, siendo mucho más sencillo el escalamiento de esta arquitectura que el escalamiento de un solo procesador, haciendo referencia a los métodos de paralelización utilizados en entregas anteriores.