Aplicación de la metodología Foster en multiplicación de matrices cuadradas

Pedro Escobar, Sergio Rojas, Alexander Rodríguez,
Nicolás Rojas, Gonzalo Castillo, Israel Orozco

Pontificia Universidad Javeriana, Bogotá D.C

{pedro-escobar, s_rojas, jjrodriguezc, nicolasrojasg,
g.castillol, isteven.orozcor}

@javeriana.edu.co

Resumen. Con el fin de profundizar y entender la Metodología Foster, se pueden aplicar diferentes problemas, los cuales tienen características únicas y distribuciones óptimas de las tareas, comunicaciones y mapeo de los procesadores. El documento analiza detalladamente la multiplicación de matrices cuadradas utilizando la Metodología Foster, describiendo cada una de las etapas y analizando la solidez de cada una.

Palabras claves: tarea, comunicación, canal, procesador, paralelismo, complejidad, matriz, nodo, partición, aglomeramiento, mapeo.

1. Introducción

El presente documento realiza el proceso completo de la Metodología Foster aplicándolo al problema de multiplicación de matrices cuadradas. El algoritmo que se utiliza para la multiplicación es el ingenuo, es decir, aquel que realiza todas las iteraciones para las multiplicaciones y sumas. Se define la etapa de particionamiento, donde se reducen las tareas a la mínima cantidad, la etapa de comunicación, donde se establecen las uniones entre las tareas y los datos que se necesitan para cada tarea, la etapa de aglomeración, donde se unifican tareas en algunas más grandes con el fin de reducir la latencia y mejorar la localidad, la etapa de mapeo, donde se distribuyen las tareas aglomeradas de tal manera que se previene en mayor medida el tiempo de ocio en los procesadores. Por último, el documento analiza el tiempo de ejecución final del algoritmo, sumando cada uno de los tiempos de ejecución de las tareas según su tipo.

2. Teoría de la Multiplicación de Matrices

Una multiplicación de matrices consiste en una operación binaria que produce una matriz a partir de otras dos. El requerimiento principal para poder hacer esta operación es que la cantidad de columnas de la primera matriz debe ser igual a la cantidad de filas de la segunda matriz [2].

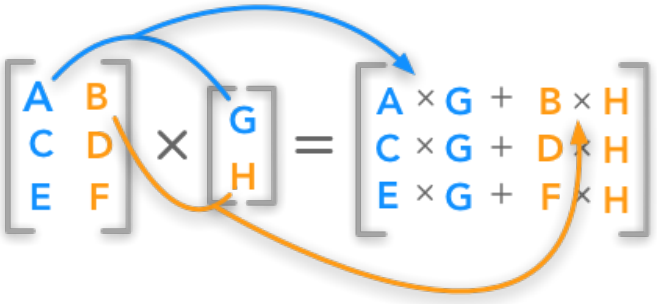

Figura 1. Proceso de la multiplicación de dos matrices.

Existen varios métodos para la multiplicación de matrices, los cuales buscan disminuir la complejidad del algoritmo y el costo computacional que se refleja cuando se multiplican matrices de gran tamaño.

En el contexto de las matrices cuadradas, que son aquellas con el mismo número de filas y columnas, la complejidad algorítmica de este proceso es $O(n^3)$ para la solución ***ingenua*** (recorrer todos los nodos y realizar todas las operaciones posibles), ya que hay *n x n* elementos en la matriz resultante y para cada elemento, se realizan *n* cantidad de multiplicaciones y sumas. En el año 2023 el mejor algoritmo descubierto reduce el exponente de 3 a 2.371552. Es importante destacar que además de hacer multiplicaciones, también se deben hacer adiciones de los productos para obtener el valor resultante en cada nodo de la matriz, en total son $(n-1)*n^2$ sumas. La complejidad sin simplificar es $O(n^3+(n-1)*n^2)$, con lo cual al ser n^3 significativamente mayor a la complejidad de las sumas, la complejidad total pasa de ser $O(n^3)$ solamente.

3. Teoría de la Metodología Foster

A continuación, se describe brevemente la metodología y sus etapas.

La metodología consiste en un diseño de cuatro etapas. La entrada de este proceso es el problema a resolver [1]. En el caso del presente documento, el problema es la multiplicación de dos matrices cuadradas. Las cuatro etapas de la metodología Foster son:

- **Particionamiento:** El proceso de dividir los datos y los procesos computacionales en piezas más pequeñas.
- **Comunicación:** El proceso para determinar cómo se comunican las tareas entre sí. Es necesario realizar la distinción entre la comunicación local y global.
- **Aglomeración:** El proceso de agrupar tareas en tareas más largas, mejorando el rendimiento o simplificando la densidad en la programación.
- **Mapeo:** El proceso de asignación de tareas a los procesadores físicos.

Para cada una de las etapas de la metodología hay una lista de chequeo con la cual se comprueba que el resultado sea adecuado.

4. Aplicación de la metodología en la multiplicación de matrices

Para fines de un buen entendimiento del proceso se tomará como ejemplo para la aplicación dos matrices de tamaño 2x2, utilizando una máquina que posee 4 procesadores:

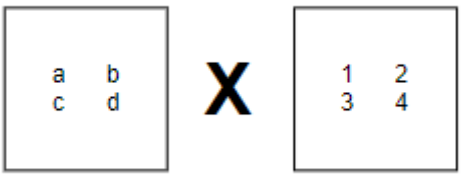

Figura 2. Matriz de ejemplo para aplicar la metodología Foster.

a. Particionamiento

Es necesario establecer cuáles son las tareas que se deben hacer para realizar la multiplicación de las matrices. Según la sección 2, la multiplicación *ingenua* de matrices tiene una complejidad de $O(n^3)$, es decir, se realizan n^3 multiplicaciones y n^2 adiciones. En el ejemplo propuesto, la cantidad de operaciones son 2^3 multiplicaciones y 2^2 adiciones. En la siguiente figura se puede observar la matriz resultante en función de las operaciones a realizar:

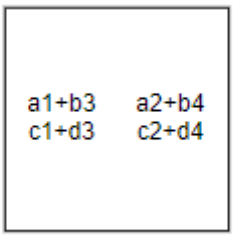

Figura 3. Matriz resultante de la multiplicación.

Según la matriz resultante, cada nodo se obtiene realizando dos multiplicaciones y una adición. Dado que el particionamiento necesita que se establezca la mayor cantidad de paralelismo posible, las tareas más pequeñas que se pueden extraer son:

- Distribución de los valores de las matrices para las operaciones.
- Multiplicación de dos nodos.
- Suma de dos nodos.
- Agrupación de los resultados en la matriz final.

Las matrices son de dos dimensiones, por lo tanto, para mayor paralelismo se realizará una descomposición 2D. La primera dimensión corresponde a la fila y la segunda a la columna.

El particionamiento propuesto para la multiplicación de dos matrices cuadradas realiza tanto descomposición de dominio como descomposición funcional.

- **Descomposición de dominio:** Los datos de cada matriz se dividen en n^3 parejas, asignando una *tarea primitiva* a cada una. La acción de dividir los datos en partes más pequeñas se denomina *scatter*. Para el ejemplo propuesto los datos se dividen en 2^3 parejas. La tarea primitiva asignada para cada pareja es la multiplicación. Luego de cada multiplicación se realizan n^2 sumas de los productos obtenidos, es decir, 2^2 sumas. Por último, se reúnen los resultados de las sumas para conformar la matriz resultante, es decir, se realiza una tarea de tipo *gather*. Hay un total de 2^3 multiplicaciones y 2^2 adiciones, o como lo denomina la metodología, tareas de tipo *reduction*, una sola operación de *gather* y una sola operación de *scatter*, dando un total de **14 *tareas primitivas***.

La cantidad de tareas para la solución con máxima partición sigue la siguiente fórmula que puede ser utilizada para diferentes tamaños del problema:

$$n^3+n^2+2$$

donde n representa el tamaño de la matriz cuadrada.

A continuación, se verifica la solidez del particionamiento con la lista de chequeo:

1. ¿La partición define al menos un orden de magnitud más de tareas que procesadores?

   Se definieron 4 procesadores para este ejemplo, y se obtuvieron 14 operaciones, por lo tanto, **SÍ** se tiene un número superior de tareas que de procesadores.

2. ¿Se evita en lo posible la redundancia en el almacenamiento de datos y en las computaciones realizadas?

   SÍ. A pesar de que cada nodo de la matriz se repite *n* veces, es necesario realizar esto para que haya la mayor cantidad de paralelismo posible. Si se quisiera anular la redundancia en el almacenamiento, se realizaría la operación en un sólo procesador, con lo cual no se aplicaría la metodología de Foster. En cuanto a la computación, todas las operaciones hechas entre los nodos de las matrices corresponden a operaciones únicas, ya que una operación utiliza siempre una combinación diferente de nodos.

3. ¿Las tareas primitivas son del mismo tamaño?

   SÍ, asumiendo que el costo computacional de realizar una multiplicación y una suma son el mismo o prácticamente el mismo.

4. ¿El número de tareas incrementa en función del tamaño del problema?

   SÍ. El número de tareas depende directamente de la complejidad original de la multiplicación *ingenua* de matrices. Son n^3 multiplicaciones y n^2 adiciones, por lo tanto, entre mayor sea el tamaño de las matrices, mayor cantidad de tareas se realizarán para la multiplicación.

5. ¿Se han considerado otras alternativas para las particiones?

   SÍ. A pesar de que en el presente documento se muestra la partición que el grupo de trabajo considera la más apropiada, se exploraron otras alternativas tales como: partición con cero redundancia en el almacenamiento de datos: esta causa que no haya paralelismo puesto que no se puede repetir la data, a pesar de que se use en diferentes operaciones. No hay soluciones más allá de estas que realmente den valor a la solución, puesto que el problema está limitado a que se tiene que resolver utilizando la estrategia *ingenua* de la multiplicación de matrices.

b. Comunicación

Para la comunicación es necesario recordar que en el modelo computacional adecuado para la metodología Foster no hay memoria compartida, ya que son las tareas las encargadas de enviar datos por medio de los mensajes. Para computar tanto la suma como la multiplicación de dos valores de tareas T_1 y T_2, una de ellas debe enviar su valor al otro para que se efectúe la operación. Cuando el algoritmo termina, el resultado de la multiplicación de matrices debe estar en una tarea individual, esta se conoce como la tarea raíz, siguiendo la metodología de Quinn.

La matriz resultante tendrá el mismo tamaño que las matrices que hicieron parte de la multiplicación, lo que significa que tendrá uno o más nodos. La tarea final que posea los datos de la matriz debe corresponder a una tarea *gather*, la cual obtiene los valores de las penúltimas tareas, dando como resultado un arreglo de estos elementos. Por otro lado, las operaciones matemáticas propias de la multiplicación de matrices corresponden a tareas *reduction*, las cuales obtienen sus valores de operaciones anteriores o de la asignación inicial de los datos.

La comunicación propuesta para la multiplicación de dos matrices cuadradas, cumpliendo con el particionamiento con mayor paralelismo se muestra en la siguiente figura:

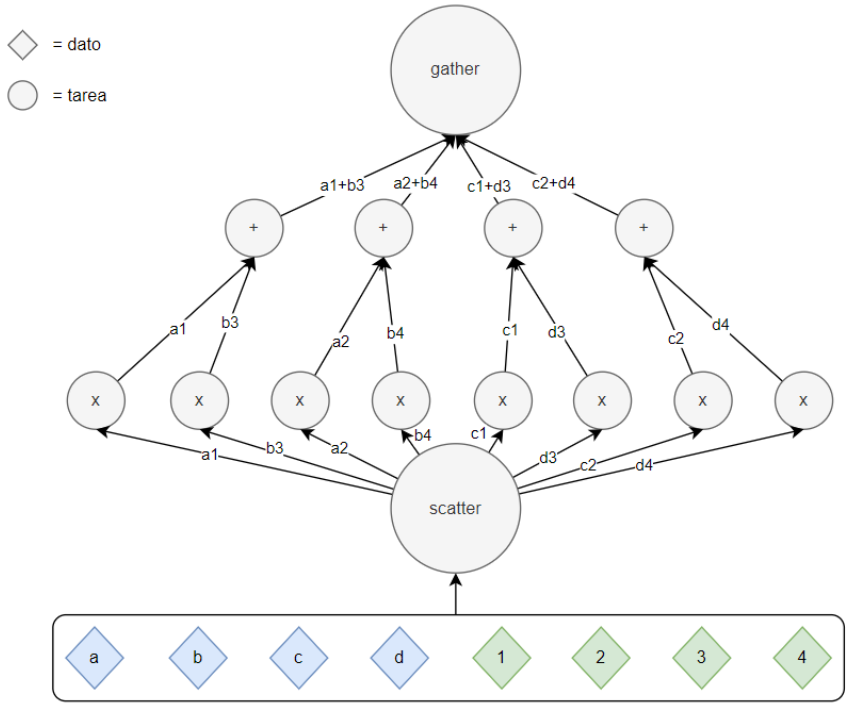

Figura 4. Comunicación entre tareas para la multiplicación de dos matrices cuadradas.

En la figura 4 se pueden observar los nodos y las comunicaciones entre estos. En el particionamiento se hizo un particionamiento por descomposición de dominio, por lo tanto, la figura debe leerse de la siguiente manera: cada nodo representa una **tarea primitiva**. Al ser un árbol, se evidencian las computaciones paralelas realizadas sobre los datos. Se pueden contar 12 comunicaciones a nivel de suma y multiplicación, junto con otras 8 comunicaciones a nivel de asignación inicial de los datos o *scatter*, y una comunicación para la tarea encargada de I/O, dando un total de 21 comunicaciones.

Se puede deducir la fórmula para las comunicaciones totales:

$$2n^3+n^2+1$$

siendo n^3 el número de comunicaciones para la asignación inicial de datos para la multiplicación y n^3+n^2 las comunicaciones a nivel de suma y *gather* y *1* la tarea de I/O.

La estructura del árbol combina los conocimientos adquiridos por el estado del arte utilizando n-arios (el árbol cuya raíz es el resultado de la multiplicación de las matrices).

A continuación, se verifica la solidez de la comunicación con la lista de chequeo:

1. ¿Todas las tareas tienen el mismo número de comunicaciones?

   Sí y no. Las tareas y sus comunicaciones conforman un árbol n-ario, el cual, **SÍ** tiene la misma cantidad de comunicaciones para los nodos de un mismo nivel. Sin embargo, a medida se van realizando las tareas, son menos comunicaciones puesto que las operaciones son de tipo **reduction**. La comunicación es balanceada puesto que en cada iteración se tiene la misma cantidad de comunicaciones, por lo que se puede deducir que no hay inconvenientes al escalar con tamaños más grandes.

2. ¿Las tareas tienen el menor número de comunicaciones posible?

   SÍ. No existen comunicaciones redundantes entre las tareas, puesto que todas ellas corresponden a combinaciones únicas de valores de las matrices para producir el resultado. No obstante, en la etapa de aglomeración se pretende ver si es posible reducir comunicaciones al agrupar tareas, por lo que el valor puede disminuir aún más.

3. ¿Las comunicaciones entre las operaciones se pueden ejecutar de forma concurrente?

   SÍ. Las comunicaciones para la multiplicación se realizan de forma concurrente y paralela. Hay un detalle en la asignación de los datos, puesto que se tienen que asignar *n* veces los nodos de las matrices para efectuar las multiplicaciones. Por ejemplo, en la matriz 2x2, el nodo en la fila 1 y columna 1 se utiliza en dos multiplicaciones independientes, por lo que la concurrencia en cuanto al acceso al mismo dato sí está presente.

4. ¿Las tareas pueden ejecutar sus operaciones de forma concurrente?

   SÍ. Cada multiplicación de los valores de las matrices se puede realizar de forma paralela y concurrente, puesto que no hay dependencias entre ellas. En cuanto a las operaciones de adición, estas dependen de los resultados de las multiplicaciones, pero pueden hacerse de forma paralela y concurrente una vez se satisfagan dichas dependencias.

c. Aglomeración

Para esta etapa es necesario recordar los datos que se tienen para el ejemplo propuesto:

- 4 procesadores.
- 14 tareas primitivas, siguiendo la fórmula n^3+n^2+2.
- 21 comunicaciones, siguiendo la fórmula $2n^3+n^2+1$.

Para reducir la cantidad de comunicaciones, se pueden agrupar tareas de multiplicación y suma para cada nodo de la matriz resultante:

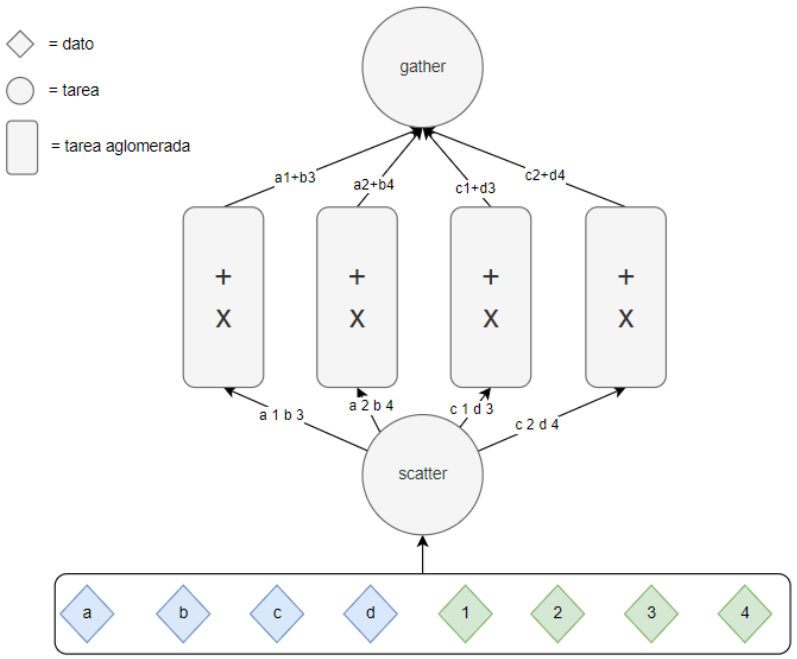

Figura 5. Comunicación entre tareas luego de aplicar aglomeración.

Se puede observar que las operaciones de suma y multiplicación se aglomeran en tareas más grandes, con el fin de reducir la comunicación. Ahora, las comunicaciones se reducen de 21 a 9, siendo la fórmula:

$$n^3+1$$

lo cual reduce significativamente las comunicaciones a nivel de operaciones. Las comunicaciones a nivel de asignación de valores son las de mayor cantidad, y no es posible aglomerar las comunicaciones de *scatter* puesto que cada pareja que se multiplica se conforma de un elemento de una matriz y el otro de la otra matriz. A diferencia de otros problemas como el de *n-body*, la multiplicación de matrices realiza operaciones tanto de *scatter*, *reduction* y *gather*, por lo que no se pueden aplicar las mismas estrategias para la aglomeración.

La cantidad de tareas disminuye de 14 a 6, siguiendo la siguiente fórmula:

$$n^2+2$$

La solución disminuye la latencia puesto que se realizan menos comunicaciones, disminuyendo el tiempo de *start-up*. Por otro lado, el tiempo de transmisión aumenta puesto que al agrupar las multiplicaciones y sumas de un nodo de la matriz resultante, el tamaño de los bytes a enviar aumenta. Estos *trade-offs* se deben tener en cuenta y modificarse según la arquitectura del problema, en el caso de este trabajo, al ser 4 procesadores es posible realizar esta aglomeración, dejando un nodo de la matriz resultante para cada operación.

A continuación, se verifica la solidez de la aglomeración con la lista de chequeo:

1. ¿Se reducen los costos de comunicación al incrementar la localidad?

   SÍ. Se aglomeraron las tareas de multiplicación y suma para cada nodo de la matriz resultante, lo cual incrementa la localidad y reduce las comunicaciones totales de 21 a 9.

2. Si la aglomeración replicó computación, los beneficios de dicha replicación deberían tener mayor peso que los costos, para un número de problemas con tamaños y procesadores diferentes.

   La aglomeración repite procesos de adición y multiplicación, aumentando la localidad, lo cual disminuye los costos derivados por la latencia. Adicionalmente, la solución permite que se pueda utilizar con menos procesadores, comparado con la solución sin aglomeración, donde habría mucha más comunicación interna.

3. Si la aglomeración hace réplicas de los datos, no debería comprometerse la escalabilidad del algoritmo al restringir la cantidad de problemas con tamaños y procesadores diferentes que este pueda soportar.

Para la aglomeración propuesta no se realiza réplica alguna de los datos, puesto que las multiplicaciones y sumas hechas utilizan nodos diferentes de las matrices.

4. ¿La aglomeración produce tareas con lógica similar y costos de comunicación similares?

   SÍ. Todas las aglomeraciones realizadas tienen la misma cantidad de comunicaciones y lógica interna.

5. ¿El número de tareas sigue aumentando al aumentar el tamaño del problema?

   SÍ. Las tareas varían siguiendo la fórmula n^2+2 la cual depende del tamaño de la matriz cuadrada, por tanto, es escalable.

6. ¿Hay suficiente concurrencia para la computadora actual y computadores diferentes en el futuro?

   SÍ. Hay múltiple acceso al mismo dato, en concreto *n* veces se accede al mismo dato de la matriz para ser utilizado en las operaciones. Adicionalmente, las operaciones en paralelo siguen presentes sin importar el tamaño de los datos.

7. ¿El número de tareas no puede reducirse más, previniendo el desbalance, reduciendo los costos de implementación y la escalabilidad?

   SÍ. No es posible reducir más la densidad de tareas y comunicaciones puesto que se perdería la capacidad de programar de forma paralela, y se requeriría tener procesadores de mayor capacidad, desaprovechando el paralelismo de la solución actual.

8. El *trade-off* entre la aglomeración escogida y el costo de la modificación del código secuencial es razonable?

   SÍ. Para ambas soluciones la complejidad es de $O(n^3)$, por lo que el paralelismo reduce n/p (n= tamaño de la matriz cuadrada, p = número de procesadores) veces aproximadamente la duración de la ejecución. Es importante recalcar que no es exactamente esa la reducción, puesto que se deben tomar en cuenta los tiempos de *start-up* y transmisión.

d. Mapeo

En esta etapa se distribuyen las tareas en los procesadores físicos. Para el problema actual, se tiene la siguiente información:

- 4 procesadores.
- 6 tareas.
- 9 canales de comunicación, entre los que se encuentran 4 canales de comunicación para enviar los datos para las multiplicaciones, 4 canales para el *gather* y 1 canal para I/O.

La idea del mapeo es minimizar el tiempo de ejecución. Esto se logra maximizando la utilización del procesador y minimizando la comunicación interna. Con lo anterior se deduce que no es óptimo poner todas las tareas en un sólo procesador, ya que a pesar de utilizarlo en una mayor medida que en un proceso paralelo, se maximiza la comunicación interna. Se debe tener un buen *trade-off* en cuanto al procesamiento y la comunicación interna, y asegurarse de que los procesadores no están ociosos durante gran parte de la ejecución.

En la aglomeración ya se estuvo tomando en cuenta el posible mapeo para las tareas, asegurándose de poner tareas que se comunican frecuentemente lo más juntas posible, reduciendo la latencia. Adicionalmente se están poniendo tareas que se pueden hacer en paralelo al mismo nivel, por lo que se ejecutan de forma concurrente. El mapeo propuesto para los 4 procesadores es el siguiente:

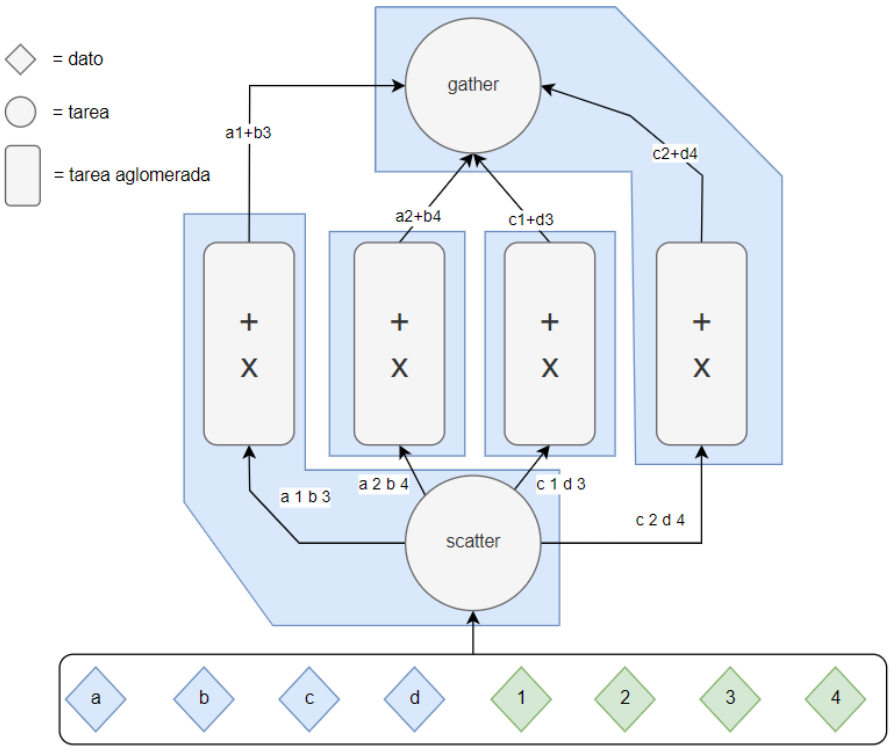

Figura 6. Mapeo de las tareas en procesadores físicos.

En la figura 6 se observan 4 polígonos azules que corresponden a los procesadores, los cuales realizan diferentes tareas. Se tiene que:

- Todos los procesadores en algún momento ejecutarán $\frac{n^2}{4}$ tareas de multiplicación y suma. Para el ejemplo actual, todos los procesadores ejecutarán $\frac{2^2}{4} = 1$ tareas de multiplicación y suma
- 1 procesador distribuye los datos de las matrices originales.
- 1 procesador recibe los resultados y conforma la matriz resultante.

A continuación, se verifica la solidez de el mapeo con la lista de chequeo:

1. ¿Se han contemplado diseños basados en una sola tarea por procesador y múltiples tareas por procesador?

SÍ. La solución actual es fruto del particionamiento, comunicación y aglomeración propuesto. Diseños con una sola tarea por procesador tienen un inconveniente y es que una vez el procesador de *scatter* termina su ejecución, este se vuelve ocioso por el resto de la ejecución, y a su vez, el procesador de *gather* se mantiene ocioso hasta que todos los procesadores hayan terminado su ejecución. Es inevitable el tiempo al inicio cuando se espera hasta distribuir toda la data, pero se puede prevenir que a partir de ahí algún procesador quede ocioso.

Diseños con múltiples tareas por procesador son los más adecuados para reducir el tiempo de ocio. Sin embargo, estos pueden aumentar la transmisión inter-procesador, lo cual se considera mucho menor que el tiempo de ocio, por lo tanto el riesgo se asume.

2. ¿Se han considerado ubicaciones estáticas y dinámicas para las tareas de los procesadores?

   SÍ. La solución actual propone una ubicación estática, ya que la multiplicación de matrices es un proceso que requiere de tener los datos completos de las dos matrices cuadradas, por lo tanto, no se espera que haya dinamismo en las operaciones. Así mismo, las comunicaciones de la solución propuesta se encuentran altamente balanceadas, por lo que la ubicación estática es la más adecuada.

3. Si se escogió un esquema de balanceo de carga centralizado, ¿está seguro de que el *manager* no se convertirá en un cuello de botella?

   No se está aplicando balanceo de carga dinámica, por lo tanto no se tiene un algoritmo de agendamiento de tareas. Por lo anterior, no hay un *manager* que controle el mapeo puesto que para este problema es sólo uno.

4. Si se escogió un esquema de balanceo de carga dinámico, ¿ha considerado los costos relativos de las diferentes estrategias y tiene en cuenta los costos de la implementación en su análisis?

   No se está aplicando balanceo de carga dinámica, por lo tanto no se tiene un algoritmo de agendamiento de tareas. No obstante, un análisis del tiempo de ejecución será realizado en la sección Análisis.

5. Input de data:

La tarea de *scatter* debe leer todos los datos de las dos matrices cuadradas para poder enviarlos hacia las tareas correspondientes. Según las secciones anteriores, cada envío necesita una pareja de datos, el primero de la primera matriz, y el segundo de la segunda matriz. A su vez, estas parejas se mezclan con otras, dando como resultado un nodo de la matriz final. Para cualquier tamaño de la matriz, el tiempo que le toma a la tarea *scatter* para leer los datos es:

$$\lambda_{io} + \frac{2n^2}{\beta_{io}}$$

Donde λ es la latencia, β es la banda ancha del canal y $2n^2$ corresponde a la cantidad de datos que hay en las dos matrices. El procesador I/O lee cada uno de estos datos y los envía en diferentes combinaciones hacia las tareas de multiplicación y suma para que realicen las operaciones.

6. Análisis

Siempre que se implementa un algoritmo de ejecución en paralelo, se debe analizar su rendimiento. En particular, se quiere saber el tiempo de ejecución del algoritmo en función del tamaño del problema y el número de procesadores.

La información para el análisis es la siguiente:

- 2 matrices de tamaño $n=2$.
- 4 procesadores.
- 6 tareas.
- 9 canales de comunicación, entre los cuales hay comunicación interna de tareas en un procesador o externa hacia tareas en otro procesador.

El tiempo total de ejecución del algoritmo consta de la suma del tiempo para realizar el input y el *scatter* de la data, más el tiempo para la computación de la multiplicación de matrices, más el tiempo para reunir los resultados y mostrarlos. Se asume que se realizan M cantidad de iteraciones. Según las secciones anteriores, se puede decir que:

- La tarea inicial para *scatter* tiene un tiempo de ejecución de $\lambda_{io} + \frac{2n^2}{\beta_{io}}$

- La tarea final para *gather* recoge el resultado de la multiplicación de las dos matrices, los cuales corresponden a una matriz con las mismas dimensiones, por tanto, el tiempo de *gather* es $\lambda_{io} + \frac{n^2}{\beta_{io}}$.
- Las tareas de multiplicación y suma tienen el mismo tiempo de ejecución, ya que realizan exactamente la misma cantidad de sumas y multiplicaciones. Cada tarea recibe *2n* números y cada procesador recibe $\frac{n^2}{p}$ tareas, por tanto, cada procesador recibe $\frac{2n^3}{p}$ números aproximadamente.

  La cantidad de multiplicaciones realizada por procesador se extrae de la propia complejidad del algoritmo, es decir $\frac{n^3}{p}$.

  La cantidad de sumas realizada por procesador se extrae de la parte de la complejidad del algoritmo que se cancela al ser menor que las multiplicaciones, es decir $\frac{(n-1)n^2}{p}$.

  Si se define χ como la cantidad de tiempo para realizar cada operación binaria, y asumiendo que el tiempo para la multiplicación es el mismo que para la suma, se puede concluir que el tiempo para cada tarea es de:

$$\chi\left(\frac{(n-1)n^2}{p} + \frac{n^3}{p}\right) = \chi\left(\frac{2n^3-n^2}{p}\right)$$

  El tiempo para todas las tareas *reduction*, por tanto, es:

$$\chi\left(\frac{2n^3-n^2}{p}\right) * \frac{n^2}{p} = \chi\left(\frac{2n^5-n^4}{p^2}\right)$$

  Para el ejemplo actual, con *n=2*, se tienen un total de 12 operaciones de *reduction*, 8 multiplicaciones y 4 sumas.

  En total, el tiempo de ejecución del algoritmo paralelo, asumiendo que *n* es divisible por *p* es:

$$\lambda_{io} + \frac{2n^2}{\beta_{io}} + \chi\left(\frac{2n^5-n^4}{p^2}\right) + \lambda_{io} + \frac{n^2}{\beta_{io}}$$

Simplificando la fórmula final del tiempo de ejecución del algoritmo aplicando la Metodología Foster es:

$$2\lambda_{io} + \frac{3n^2}{\beta_{io}} + \chi(\frac{2n^5 - n^4}{p^2})$$

7. Conclusiones

Luego de realizar el experimento con la Metodología Foster, es evidente el potencial de esta estrategia para realizar una distribución adecuada de las tareas, las comunicaciones y la asignación con los procesadores. La metodología permite aprovechar de la mejor forma posible la cantidad de procesadores, para reducir costos y tiempos.

Adicionalmente, en cada una de las etapas es posible realizar una auditoría para comprobar que se tiene un balance adecuado entre la cantidad de comunicaciones y el aprovechamiento del procesador. No obstante, hay que tener en cuenta los *trade-offs* en los que al incrementar una de las características, provoca el decremento de otra.

Por otro lado, previo a la elaboración de la Metodología Foster, es necesario tener una implementación inicial adecuada del algoritmo, buscando su optimización. Para el contexto de la multiplicación de matrices cuadradas, ya existen algoritmos que disminuyen la complejidad, por lo que es posible tener un mejor rendimiento.

8. Referencias

[1] Weiss, S. (s.f.). Chapter 3 Parallel Algorithm Design. En S. Weiss, *Parallel Computing* (pág. 27).

[2] Wikipedia. (s.f.). *Computational complexity of matrix multiplication*. Obtenido de https://en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication