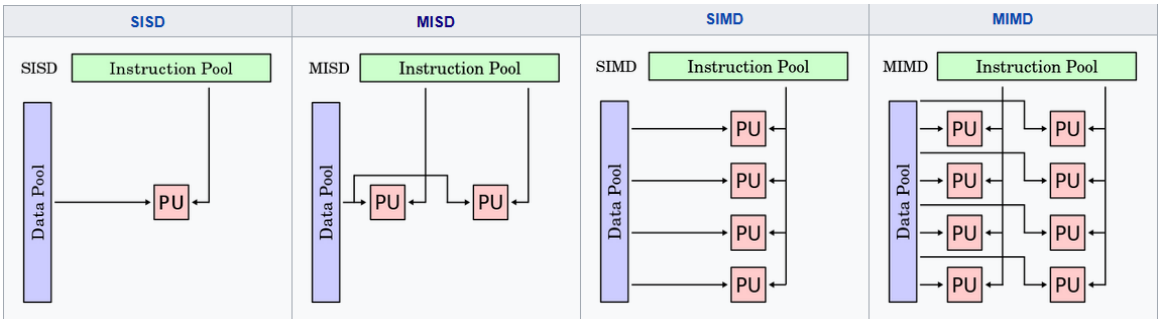
Introducción

Después de lo realizado en la práctica anterior ahora aprenderemos otra de las maneras de enviar y recibir datos entre procesadores, estas funciones son "*MPI_Bcast*" y "*MPI_Gather*", la primera se enfoca en el envío de mensajes a todos los procesadores, por lo que cada uno recibirá una copia de los datos originales que podrán modificar; la otra instrucción es la enfocada a la recepción, en ella los procesadores enviaran sus mensajes y el que sea designado como raíz será el encargado de juntarlos en un orden dado por el pid (Processor Identifier) en un arreglo.

Otra de las cosas necesarias para comprender de mejor manera esto es conocer lo que es la taxonomía de Flynn la cual es una clasificación de arquitectura de computadoras, se basan en las instrucciones y datos que manejan y por combinaciones tenemos 4 clasificaciones:

- **SISD (Simple Instruction Simple Data)**: Un único procesador ejecuta un solo flujo de instrucciones para operar datos en una única memoria. Se ejecuta una única instrucción y un dato en cada ciclo de reloj.
- **SIMD (Simple Instruction Multiple Data)**: Todas las unidades ejecutan la misma instrucción sincronizadamente, pero con datos distintos.
- **MISD (Multiple Instruction Simple Data)**: Todas las unidades ejecutan diferentes instrucciones, pero con datos iguales.
- **MIMD (Multiple Instruction Simple Data)**: Varios procesadores autónomos que ejecutan simultáneamente instrucciones diferentes sobre datos diferentes.

Aquí se puede ver una pequeña explicación de manera gráfica del cómo son cada una:



Preguntas

- Las comunicaciones colectivas facilitan la programación y simplifican el código. ¿Se puede pensar que acortan el tiempo de ejecución de los programas?
- Explicar qué refleja la medida de tiempo realizada
- Plantear otras posibilidades de medida de tiempos de ejecución que permitan distinguir los tiempos invertidos en comunicación entre procesos y los tiempos dedicados al cálculo.
- Adjunta tu código y la impresión de pantalla de lo que ocurre.

Respuestas

En principio si, ya que una de las maneras de atacar los problemas es por el conocido divide y vencerás, ya que al dividir el problema y compartirlo con los diferentes procesadores hace que sea aparte de más sencillo de resolver, se llega a acortar el tiempo de ejecución siempre y cuando lo que tenga que ejecutar un procesador no dependa de lo que hagan los demás, ya que eso crearía un atraso en la ejecución lo que la podría llegar a tardar más a que si se ejecutara de manera secuencial.

El tiempo medido nos refleja la duración de la ejecución del programa desde que comienza a distribuir las matrices a los procesadores y culmina en el retorno de los vectores resultados.

La posibilidad que se utilizó fue la de ocupar diferentes temporizadores uno para el solo envío de datos, otro para la recepción y otro que se utiliza en se ocupa para medir el tiempo de las sumas en cada procesador, al final de que se muestra los resultados de la suma y las matrices, se muestran los tiempos obtenidos en las diferentes partes que se mencionan arriba.

El código se anexa al final del documento.

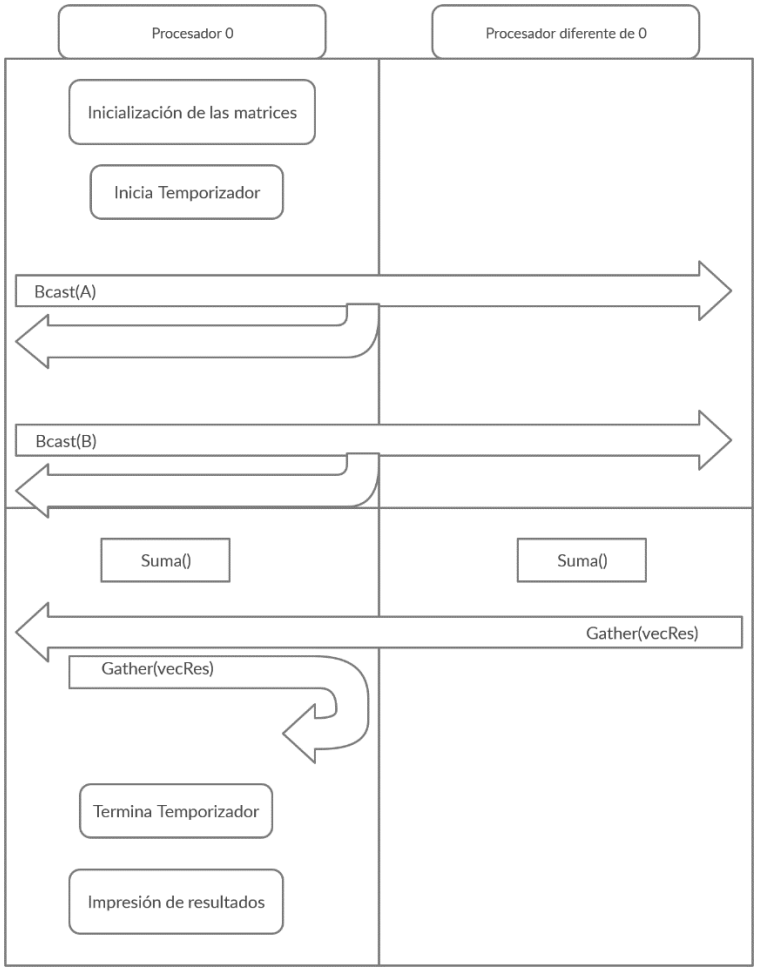
Aquí se utilizaron 3 vectores por lo que el tamaño de las matrices terminó siendo de 3 * 3 como se puede ver en la captura de abajo, junto a ello están los mensajes de tiempos de ejecución de cada instancia que se cronometró

```
ramy@Ramy:~/Distribuidos/Pr3$ mpirun -np 3 ./pr3
--------------------------------------------------------------------------
WARNING: Linux kernel CMA support was requested via the
btl_vader_single_copy_mechanism MCA variable, but CMA support is
not available due to restrictive ptrace settings.

The vader shared memory BTL will fall back on another single-copy
mechanism if one is available. This may result in lower performance.

  Local host: Ramy
--------------------------------------------------------------------------
Matriz A
[3 0 6 ]
[1 3 2 ]
[5 7 2 ]
Matriz B
[5 9 1 ]
[2 0 7 ]
[9 3 7 ]
Matriz Resultado
[8 9 7 ]
[3 3 9 ]
[14 10 9 ]
El tiempo total fue del envio: 0.000039
El tiempo total fue de la recepción: 0.000090
El tiempo total : 0.000132
El tiempo total de la operación en el procesador[0]: 0.000001
El tiempo total de la operación en el procesador[1]: 0.000001
El tiempo total de la operación en el procesador[2]: 0.000001
[Ramy:00523] 2 more processes have sent help message help-btl-vader.txt / cma-permission-denied
[Ramy:00523] Set MCA parameter "orte_base_help_aggregate" to 0 to see all help / error messages
```

En este otro ejemplo es ejecutandola con 100 procesadores por lo que se puede ver que en comparación a la anterior los tiempos son mayores, no se muestra la impresión de las matrices dado que el tamaño es demasiado grande para ser mostrado en pantalla.

```
El tiempo total fue del envio: 0.021657
El tiempo total fue de la recepción: 0.003165
El tiempo total : 0.024830
El tiempo total de la operación en el procesador[0]: 0.000005
```

Diagrama UML



Conclusión

En conclusión, tenemos que ahora hemos aprendido un poco más sobre las funciones de MPI respecto los envíos y recepciones y las posibles aplicaciones en que las podríamos utilizar como en este caso fue la suma de las columnas de dos matrices, pero a su vez esto lo podemos ampliar a que cada procesador reciba una lista de datos y los procesen de manera distinta ya que basándonos en la taxonomía de Flynn el envío broadcast es del tipo MISD (Multiple Instruction Simple Data) ya que el procesador raíz envía a todos una copia de los datos y cada procesador puede ocupar de manera diferente los datos que le llegaron, mientras que la recepción de datos es del tipo SIMD (Simple Instruction Multiple Data) ya que el nodo raíz recibe todos los datos de los procesadores y les aplica la misma instrucción a todos después.