Programación Paralela: MPI

Agustin Colazo

Septiembre 2018

1 Introducción

MPI (Message Passing Interface) es un estándar que define la sintaxis y semántica de las funciones contenidas en una bibliteca de paso de mensajes diseñado para ser usada en programas que son ejecutados en sistemas distribuidos.
Esta biblioteca aporta sincronización entre procesos, permite la exclusión mutua, transferencia de datos entre procesos, etc.

2 Descripción

Se implementará un algoritmo de multiplicación de matrices cuadradas usando cuatro procesos MPI y diviendo los datos de cada una de las matrices en 4 bloques cuadrados, cada uno de los cuales será poseido por cada uno de los procesos.
Se hará un algoritmo que utilice primitivas bloqueantes, y otro no bloqueantes.

3 Desarrollo

Cada programa inicializa sus datos y metadatos. Los datos serian la porciones de las matrices A y B que les corresponden. Los metadatos describen a que procesos deben enviar estos datos, y de quien recibir datos.
En ambos programas se aloca memoria para cada matriz, y se aloca un buffer para la recepción de datos. Los programas son híbridos entre OpenMP y MPI. Se usa OpenMP solo en la multiplicación de las matrices, el resto de las funciones son ejecutados por un solo hilo.

3.1 Bloqueante

En el caso del programa que usa primitivas bloqueantes se hace uso de las funciones Send y Recv.
MPI_Send: Send bloqueante. La rutina devuelve cuando el buffer de la aplicación esta lista para ser reusado. No garantiza que los datos hayan sido recibido por el receptor.

MPI_SSend: Send bloqueante sincrono. Envia un mensaje y bloquea asta que el buffer del proceso que envía este libre para ser reusado y el proceso destino haya comenzado a recibir el mensaje.
MPI_Recv: Recibe un mesaje y bloquea hasta que los datos esten disponibles en el buffer del proceso receptor.
Para evitar el bloqueo mutuo dos procesos envian datos, mientras otros dos reciben. La separación de los datos se hizo de tal manera que solo hay comunicación dentro de dos pares de procesos. Los procesos 0 y 1 se comunican entre sí, y los procesos 2 y 3 también.

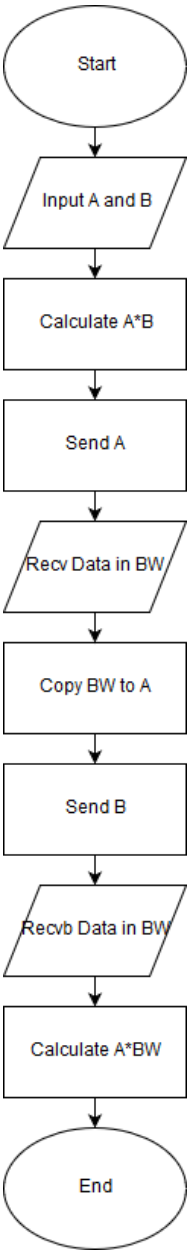


Figure 1: Proceso 0 y 3

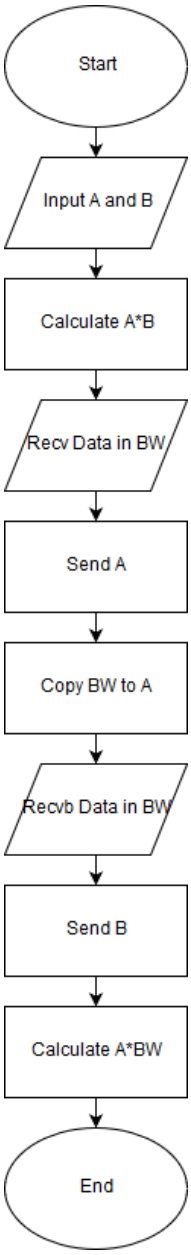


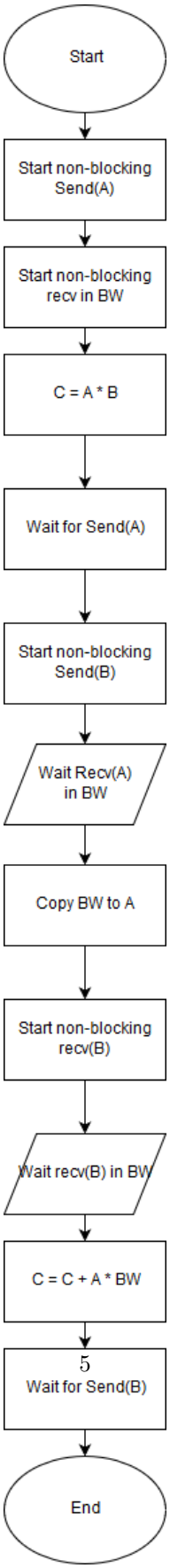
Figure 2: Proceso 1 y 2

3.2 No Bloqueante

El programa con primitivas no bloqueantes.
MPI_ISend: Se identifica un area de memoria como buffer de envío. El pro-

grama continua sin esperar que el mensaje sea copiado del buffer.
MPI_IRecv: Identifica un area de memoria como buffer receptor. El programa continua sin tener que esperar que el mensaje sea recibido y copiado en el buffer.
MPI_Wait: Bloquea el programa hasta que una operación de envío o recibida de datos no bloquante se haya completado.
MPI_Test: Revisa el estado de una operación de envío o recibida no bloqueante. Devuelve un valor que indicará el estado.

El programa no bloqueante se comienza envía una matriz antes de realizar el calculo de los datos. Mientras se envían los datos se realiza el calculo. Luego de realizar el calculo se espera hasta que hayan sido recibido los datos correspondientes. Una vez recibidos se copian en el buffer correspondiente y se comienza a enviar la otra parte de los datos, al ser recibidos se termina de calcular el resultado.


Start
Start non-blocking Send(A)
Start non-blocking recv in BW
C = A * B
Wait for Send(A)
Start non-blocking Send(B)
Wait Recv(A) in BW
Copy BW to A
Start non-blocking recv(B)
Wait recv(B) in BW
C = C + A * BW
Wait for Send(B)
End

4 Resultados

El tiempo de ejecución del programa se mide por el tiempo que tardo el proceso más lento.
Los procesos se ejecutaron en los hosts: Franky, Rocky, sun-0-5 y sun-0-7.

4.1 Tamaño de las matrices

Se realizo la multiplicación de matrices con dos tamaños de matrices distintos.

4.1.1 8.000

- 1 proceso: 679 segundos.
- 4 procesos, bloqueante: 265 segundos.
- 4 procesos, no bloqueante: 380 segundos.

4.1.2 12.000

- 4 procesos, bloqueante: 1022 segundos.
- 4 procesos, no bloqueante: 1310 segundos.

5 Conclusión

Se observo un speedup de 2.56 para el programa bloqueante y 1.79 para el no bloqueante. Es llamativo que el programa con llamadas bloqueantes es más rápido que el programa con llamadas no bloqueantes, en ambos casos.
Desarrollar programas híbridos usando MPI y OpenMP da como resultado poder multiplicar matrices de dimensiones muy grandes de manera rápida. Pero optimizar OpenMP cuando las arquitecturas de los distintos hosts son distintas es un desafío.
Comparado con OpenMP, MPI tiene que enviar los datos de un host a otro, lo cual resulta en un overhead. Pero dependiendo del tamaño de los datos, esto se justifica.

6 Comandos

mpicc -fopenmp name.c -o name
mpirun -np process_number -host host1,host2,host3,host4 ./name

References