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.
Las partes de las matrices A y B son inicializadas en cada host de manera que cada host pueda calcular una parte de C con las matrices de las que es dueño. Y que solo necesite enviar cada parte una vez. A continuación se muestra como fueron distribuidas las distintas partes de las matrices, los numeros corresponden al identificador del proceso MPI.

Matriz A $\begin{matrix} 0 & 1 \\ 3 & 2 \end{matrix}$

Matriz B $\begin{matrix} 0 & 3 \\ 2 & 1 \end{matrix}$

Matriz C (Parte a calcular) $\begin{matrix} 0 & 1 \\ 2 & 3 \end{matrix}$

Parte de matriz A: Proceso 0 envía a 1, y 1 a 0. Proceso 2 envía a 3, y 3 a 2.
Parte de matriz B: Proceso 3 envía a 1, y 1 a 3. Proceso 2 envía a 0, y 0 a 2.

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.

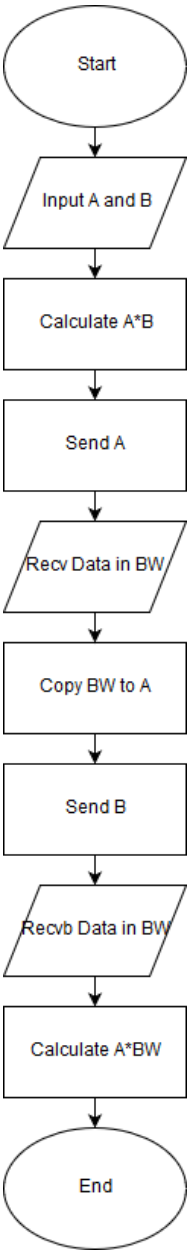


Figure 1: Proceso 0 y 3

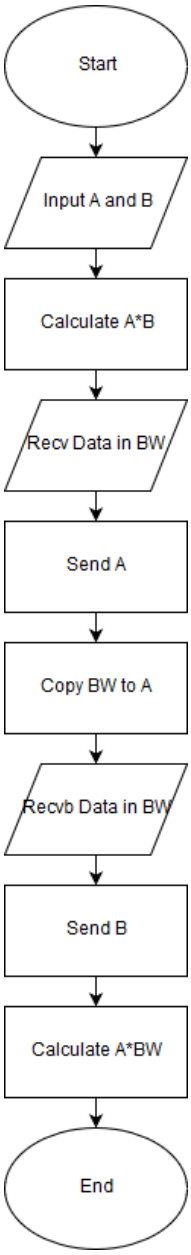


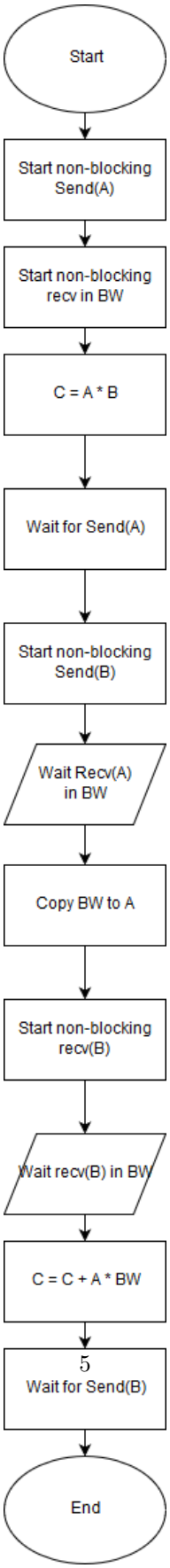
Figure 2: Proceso 1 y 2

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. Se usaron las maquinas rocky, pulqui, franky, sun-0-5.

4.1 Análisis de los resultados y corecciones

Al hacer un mejor analisis del programa y tiempos de ejecución se descubrió que en el programa no bloqueante algunos procesos no enviaban la parte de la matriz B hasta después de calcular la segunda parte de la matriz, provocando que los otros procesos se bloquearan hasta que estos terminaran el calculo. Por lo tanto, se cambio el orden de las últimas dos etapas. Cada proceso debe asegurarse de haber puesto a la matriz a enviar en el buffer correspondiente antes de comenzar a calcular la segunda parte de la matriz.
Se midieron también cuanto tardan las distintas etapas.

- Etapa 1: Calcular la primera parte de la matriz resultante.
- Etapa 2: Enviar A propio, recibir A ajeno en el buffer e intercambiar los punteros entre el buffer y A propio. (En el caso no bloqueante se hace una llamada no bloqueate para enviar B).
- Etapa 3: Enviar B y recibir B (en el caso no bloqueante hacer la llamada para asegurar que B se puso en el buffer para ser enviado).
- Etapa 4: Calcular el resto de la matriz resultante.

4.1.1 8.000 (bloqueante / no bloqueante)

- Etapa 1 [segundos]
  - Proceso 0: 161 / 162
  - Proceso 1: 345 / 385
  - Proceso 2: 122 / 120
  - Proceso 3: 338 / 332
- Etapa 2 [segundos]
  - Proceso 0: 187 / 224
  - Proceso 1: 2,18 / 1,18
  - Proceso 2: 218 / 212
  - Proceso 3: 1,09 / 0,76
- Etapa 3 [segundos]
  - Proceso 0: 1,15 / 0,85
  - Proceso 1: 2,18 / 1,16

  - Proceso 2: 9,07 / 54,6
  - Proceso 3: 10,1 / 54,7

- Etapa 4 [segundos]
  - Proceso 0: 161 / 163
  - Proceso 1: 339 / 350
  - Proceso 2: 121 / 122
  - Proceso 3: 338 / 332

El proceso más lento en el programa bloqueante tardo 689 segundos, y en el programa no bloqueante 737 segundos.

5 Conclusión

Es llamativo que el programa bloqueante es más rápido que el programa no bloqueante. Usar llamadas no bloqueantes puede resultar mejor para los procesos más rápidos pero resulta peor para los procesos más lentos.
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.
No siempre usar llamadas no bloqueantes significa que será más rápido. Además, uno no sabe como estan implementadas estas librerias.

6 Comandos

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

References