Teoria 7- mpi

lunes, 12 de mayo de 2025 18:07

Memoria distribuida:
Conjutno de nodos de procresamiento, cada uno con su propio espaciod e datos
Los nodos se comunican pasándose mensajes tanto para sincronizarse como para hablarse.
Pasaje de mensajes

- Cada proceso con su propio espaciod e direcciones
- Cada dato está en una partición, para interactuar tienen que cooperar los dos procesos.

Es bajo nivel, el programador tiene que distribuir los datos y acomodar los procesos, dificil de programar, mantener. En general se iusa SPMD (Single Program Multiple Data): no todos ejecutan lo mismo y los procesos no se sincronizan todo el tiempo

Send y receive
Bloqueante:

- Send bloquea al proceso hasta que el mensaje se copia a un buffer del sistema o el receptor lo recibe directamente.
- Receive bloquea al proceso hasta que el mensaje llega.
- **Ventaja:** Garantiza sincronización, evitando el uso de datos antiguos.

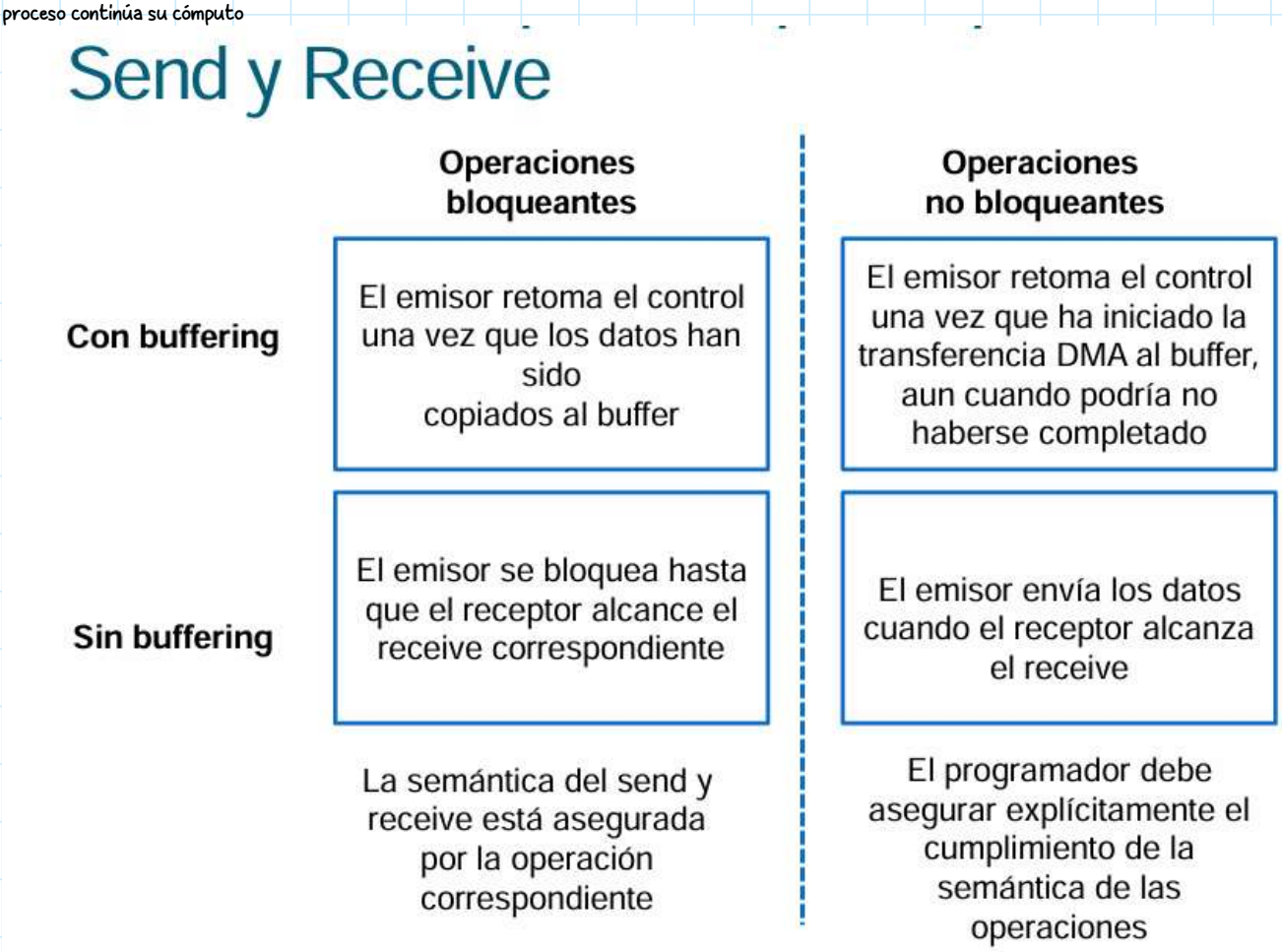
No bloquante:

- Send inicia la operación y regresa inmediatamente, sin esperar a que se complete el envio.
- Receive también puede retornar sin haber recibido el mensaje completo.
- **Ventaja:** Permite realizar cálculos mientras los datos se transmiten.
- Requiere un posterior chequeo para asegurarse la finalización de la comunicación

Bloqueante: con/sin buffer
Sin: una cagada, bloqueas el send hasta que no recibe, tiempo ocioso y deadlocks si no están bien las sentencias de comunición
Con: el send se bloquea hasta que llega al buffer (es otro bufer extra,no el del receptor). Si hay hardware para comunicación asincrónica (no se ocupa la cpu), manda y listo. Si no hay, el emisor tiene que meter el mensaje en el buiffer y ahgi se desbloquea.

Tener buffer hace menos tiempo ocioso pero aumentas el manejo de buffers. Los buffers tienen un tañano x

No bloqueante con/sin buffer
Sin:inicia comunicación al llegar al receive
Con:el emisor utiliza acceso directo a memoria (DMA) para copiar los datos a un buffer prealocado mientras el proceso continúa su cómputo



- *MPI_Init*: inicializa el entorno MPI. Debe ser invocada por todos los procesos antes que cualquier otro llamado a rutinas MPI.

```
MPI_Init (int *argc, char **argv)
```

Algunas implementaciones de MPI requieren argc y argv para inicializar el entorno

- *MPI_Finalize*: cierra el entorno MPI. Debe ser invocado por todos los procesos como último llamado a rutinas MPI.

```
MPI_Finalize ()
```

Mpi_comm se uisa para comunuicar procesos. Es un comunciador, gruarsda l,a data de los procesos que pretenecen a él.
Hay un comunicador globakl que incluye a todos: mpi_comm_world

· MPI soporta:
· Comunicaciones punto a punto: operaciones de comunicación que involucran a dos procesos → bloqueantes y no bloqueantes
Comunicaciones colectivas: operaciones de comunicación que pueden involucrar a dos o más procesos → bloqueantes y no bloqueantes

- Sobre orden:
  - MPI asegura que los mensajes no se *sobrepasarán* entre ellos.
  - Si un proceso envía 2 mensajes seguidos a un mismo receptor (M1 y M2), y ambos coinciden con el mismo receive, el orden de recepción será: M1, M2.
  - Si un proceso ejecuta 2 receive seguidos (R1 y R2), y hay un mensaje pendiente que coincide con ambos, R1 recibirá antes que R2.

- Sobre *fairness* (justicia):
  - MPI no asegura fairness → es responsabilidad del programador que un proceso no sufra *inanición*
  - Ejemplo: P0 le envía un mensaje a P2. Sin embargo, P1 envía otro mensaje a P2 que compite con el de P0 (coincide con el receive). P2 sólo recibirá uno de los 2 mensajes.

Declarar una variable arriba del todo es una variable global, cada proceso va a tener un copia de ella. NO es compartida.
Se toma tiempo completo, desde que arranca primer proceso hasta el ùltimo
Matrices van a estar en un solo proceso (master), distribuye partes de matrices, què le engo que dar a cada proceso, distribuye a wowrkers y a si mismo y el master une todo despuès. Nadie arranca hasta que el master no distribuye. Se pone barrera antes de arrancar con lo de if rank ==0. Master hace gather después para unir, entonces el ùltimo que termina es el master.
Hay que hacer que mida el master. Gather es bloqueante y asincrònico

Mpi init vs mpi init thread
Thread dice al so que voy a usar threads, forma especial de bloquear procesos a nivel so. Que no haya deadlock a nivel so.