

openMPI

Giovanni Ramírez García, PhD

Escuela de Ciencias Físicas y Matemáticas
Universidad de San Carlos de Guatemala

Guatemala, 9 febrero de 2021



Computación en paralelo

MPI: primeros pasos

MPI: pasos intermedios

MPI: pasos avanzados

Tipos de Computación en paralelo: taxonomía de Flynn

- ▶ SIMD (*single instruction, multiple data*): mismas operaciones sobre diferentes datos.
- ▶ MIMD (*multiple instructions, multiple data*): diferentes programas, diferentes datos

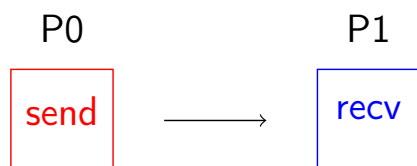
- ▶ MIMD=SPMD [Gropp, Tutorial on MPI]
- ▶ SIMD también es equivalente [Gropp, Tutorial on MPI]

MPI *funciona* principalmente en estructuras MIMD, sin embargo, en esquemas SIMD también reducen el consumo de recursos.

Esquemas de comunicación

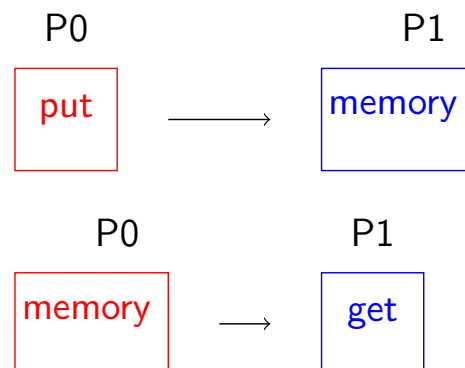
Intercambio cooperativo: *todos acuerdan transferir datos*

- ▶ el paso de mensajes se aproxima a un intercambio de datos de forma cooperativa
- ▶ los datos deben enviarse y recibirse explícitamente
- ▶ cualquier cambio en la memoria del receptor se hace con la participación del receptor



Operaciones unidireccionales

- ▶ incluyen operaciones de lectura y escritura de memoria
- ▶ ventaja: acceso inmediato, sin necesidad esperar otro proceso



MPI (I)

¿qué es?

- ▶ es una especificación para un biblioteca de *paso de mensajes*, un modelo de *paso de mensajes*
- ▶ funciona en computadoras, clústers, etc.
- ▶ diseñada para permitir el desarrollo de bibliotecas de software paralelo

¿por qué?

- ▶ MPI es un paradigma de programación
- ▶ resuelve problemas de portabilidad
- ▶ modularidad, portabilidad, heterogeneidad
- ▶ acceso a *peak performance*, topologías, herramientas de medida de desempeño

MPI (II)

Desarrollo

- ▶ Inicia en el Williamsburg Workshop (EEUU, abr/92)
- ▶ Se organiza en la Supercomputing (nov/92)
- ▶ *pre-final draft* distribuida en la Supercomputing (93)
- ▶ *final version* (may/94)

Consultores

- ▶ Vendedores: IBM, Intel, TMC, MEiko, Cray, Convex, Ncube
- ▶ Bibliotecas: PVM, p4, Zipcode, TCGMSG, Chameleon, Express, Linda
- ▶ Especialistas y consultores: Laboratorios (Argonne National Lab, Los Alamos, etc), Universidades (UC Santa Barbara, Southampton, Yale, Edinburgh, Cornell, Rice, San Francisco, etc)

Características de MPI (I)

- ▶ Comunicaciones combinando contexto y grupo para la seguridad de los mensajes
- ▶ *Threads* seguros, *buffers* estructurados
- ▶ Tipos de datos derivados
- ▶ Diferentes modos de comunicación: normal (bloqueo y sin bloqueo), síncronos, *buffered*
- ▶ Operaciones colectivas intrínsecas y también definidas por el usuario: rutinas para movimiento de grandes números de datos.
- ▶ Control de errores
- ▶ No tiene gestión de procesos.
- ▶ No tiene transferencias remotas de memoria.
- ▶ No tiene *mensajes activos*: el resto de un mensaje sólo se envía si el receptor acepta.
- ▶ No tiene memoria virtual compartida.
- ▶ **MPI incluye 125 funciones:** pero eso no implica que sea complicada de usar.
- ▶ Muchos de programas en paralelo pueden escribirse con **6 de esas funciones.**

openMPI

mpifort

- ▶ Open MPI Fortran wrapper compiler (también hay uno para C++)
- ▶ Pasa los argumentos de archivos de definiciones y bibliotecas necesarias para enlazar los objetos compilados.
- ▶ Es estrictamente recomendado usar mpifort y no enlazar manualmente los objetos compilados.

mpirun

- ▶ Execute serial and parallel jobs in Open MPI.
- ▶ **SPMD Model:**
`mpirun [options] <program> [<args>]`
- ▶ **MIMD Model:**
`mpirun [global_options] \
[local_options1] <program1> [<args1>] : \
[local_options2] <program2> [<args2>] : \
... : \
[local_optionsN] <programN> [<argsN>]`

Variables de entorno

¿quién soy? ¿quiénes están conmigo?

- ▶ Nuestros códigos pueden saber cuántos procesos están ejecutándose y quién ese proceso dentro de ese conjunto.
- ▶ Estas variables se definen en tiempo de ejecución, así que hay que revisarlas en cada ejecución.
- ▶ La variable **size** guarda el número de procesos paralelos.
- ▶ La variable **rank** guarda el número asignado a ese proceso.

```
PROGRAM hola
USE mpi
IMPLICIT NONE
INTEGER :: rank, size
INTEGER :: err

CALL MPI_Init(err)
IF (err.NE.0) STOP 'MPI_Init error'

CALL
MPI_Comm_size(MPI_COMM_WORLD,size,err)
IF (err.NE.0) STOP 'MPI_Comm_size error'

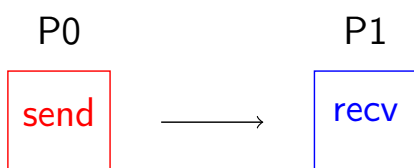
CALL
MPI_Comm_rank(MPI_COMM_WORLD,rank,err)
IF (err.NE.0) STOP 'MPI_Comm_rank error'

WRITE (*,*) "Hola mundo, soy:",rank,"de ",size

CALL MPI_Finalize(err)

END PROGRAM hola
```

Intercambio de mensajes (I)



- ▶ ¿a quién se envían los datos?
- ▶ ¿qué se envía?
- ▶ ¿cómo lo identifica el receptor?
- ▶ una función para enviar podría tener la forma
send (dest, type, address, length)

- ▶ **dest** un entero que identifique quién es el receptor
- ▶ **type** un entero positivo que diga cómo usar el mensaje
- ▶ **address, length** describen un área contigua de memoria donde está el mensaje que se quiere enviar.
- ▶ estas son unas buenas especificaciones para hardware y fáciles de entender, pero son poco flexibles.

Intercambio de mensajes (II)

comunicaciones mediadas por un buffer

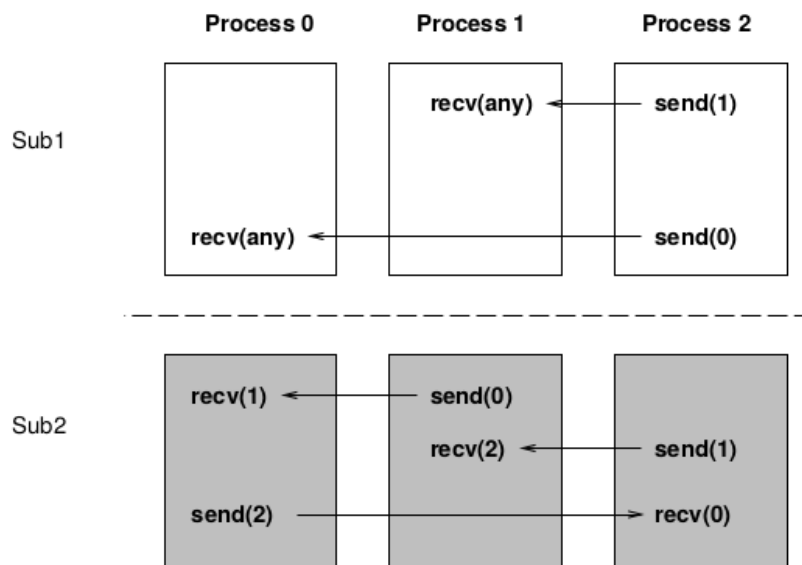
- ▶ Sólo se especifica la *dirección de inicio*, el tipo de datos y el número de datos que se enviarán
- ▶ Esto permite comunicaciones heterogéneas.
- ▶ Resuelve el problema de las representaciones de los tipos de datos en distintas computadoras.

MPI datatype

- ▶ MPI_INTEGER: INTEGER
- ▶ MPI_REAL: REAL(4)
- ▶ MPI_DOUBLE_PRECISION: REAL(8)
- ▶ MPI_COMPLEX: COMPLEX
- ▶ MPI_LOGICAL: LOGICAL
- ▶ MPI_CHARACTER: CHARACTER(1)
- ▶ MPI_BYTE
- ▶ MPI_PACKED

Intercambio de mensajes (III)

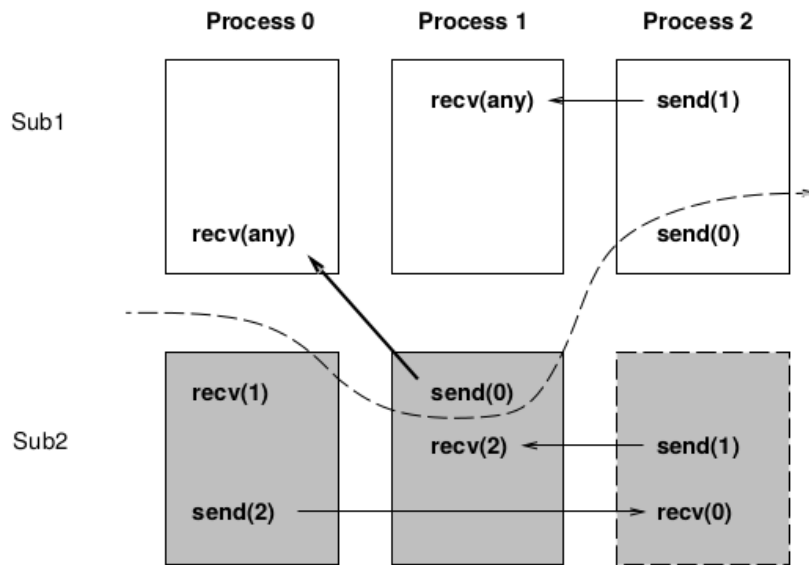
Buena práctica de comunicaciones



[Gropp, Tutorial on MPI]

Intercambio de mensajes (III)

Mala práctica de comunicaciones



[Gropp, Tutorial on MPI]

Intercambio de mensajes con bloqueo

MPI_Send(start, count, datatype, dest, tag, comm)
MPI_Recv(start, count, datatype, source, tag, comm, status)

- ▶ **start:** la variable que se envía/la variable donde se guarda el mensaje recibido.
- ▶ **count:** cuántas posiciones se envían/reciben.
- ▶ **datatype:** el tipo de datos MPI.
- ▶ **dest:** a quién se le envía el mensaje.
- ▶ **source:** de quién se recibe el mensaje.
- ▶ **tag:** etiqueta para diferenciar mensajes.
- ▶ **comm:** el *handler* para el intercambio de mensajes.
- ▶ **status:** información del estado de la recepción.

Seis funciones de MPI

1. MPI_Init
2. MPI_Finalize
3. MPI_Comm_size
4. MPI_Comm_rank
5. MPI_Send
6. MPI_Recv

Tarea

1. Hacer un programa “hola mundo”
2. Hacer un programa “ping pong”, la pelota lleva cuenta de cuántas veces ha rebotado
3. Hacer un programa “pong”, la pelota lleva cuenta de cuántas veces ha rebotado y sabe la posición 1D del otro pong.

Intercambio de mensajes sin bloqueo

Las operaciones de intercambio de mensajes sin bloqueo regresan inmediatamente el control a la siguiente instrucción que está después de su llamada.

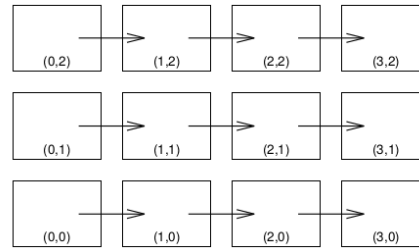
- ▶ **MPI_Isend**(start, count, datatype, dest, tag, comm, request)
- ▶ **MPI_Irecv**(start, count, datatype, dest, tag, comm, request)
- ▶ **MPI_Wait**(request, status)

Otros modos de comunicación

- ▶ Modo síncrono: **MPI_Ssend** el envío no se completa hasta que se envía una señal de *recibido*. **Usar con cuidado.**
- ▶ Modo *buffered*: **MPI_Bsend** el usuario provee el buffer al sistema.
- ▶ Modo *ready*: **MPI_Rsend** el usuario garantiza que la señal de *recibido* ha sido generada. Permite protocolos rápidos.

Topologías: ejemplo de topología cartesiana

- ▶ Topología cartesiana: es una red
- ▶ Se define usando
`MPI_Cart_create(MPI_COMM_WORLD, ndim, dims, periods, reorder, comm2d, ierr)`, que genera un nuevo *handler* de comunicaciones
- ▶ Los vecinos de la red se obtienen con `MPI_CART_SHIFT(comm2d, 0, 1, nbrleft, nbrright, irr)` y con `MPI_CART_SHIFT(comm2d, 1, 1, nbrbottom, nbrtop, irr)`



[Gropp, Tutorial on MPI]

- ▶ Las coordenadas del proceso se obtienen con
`MPI_CART_COORDS(comm1d, myrank, 2, coords, irr)`

Otras topologías y grupos

- ▶ Una red general se crea con `MPI_Graph_create`
- ▶ Las topologías sirven para aprovechar que algunos procesadores están más *cerca* que otros
- ▶ Se pueden hacer redes periódicas en una o más dimensiones.
- ▶ Se pueden hacer redes no periódicas
- ▶ Los grupos pertenecen son procesos que pertenecen a un mismo *handler*. Los grupos se crean con `MPI_Comm_create`
- ▶ Los miembros de un grupo se manipulan con `MPI_Group_incl`, `MPI_Group_excl`, `MPI_Group_union`, `MPI_Group_intersection`

otras cosas que ya no da tiempo

- ▶ Objetos de MPI
- ▶ Manejo de etiquetas para separar mensajes
- ▶ Comunicaciones privadas
- ▶ *Timers*: **MPI_Wtime()** toma tiempos locales en los procesos.
- ▶ Interface de perfilado
- ▶ Conexión mediante MPI de varios programas que usan MPI
- ▶ Comunicaciones persistentes
- ▶ Manejo de errores

¡Muchas gracias!

Contacto:
Giovanni Ramírez García, PhD
ramirez@ecfm.usac.edu.gt
<http://ecfm.usac.edu.gt/ramirez>