Pogramación Paralela y Distribuida

Licenciatura en Ciencias de la Computación

UNCuyo – Facultad de Ingeniería

Early Adopters Awarded Fall 2011 (NSF/IEEE)

Programación Paralela y Distribuida

Herramientas para la Programación Paralela y Distribuida

Unidad 5

Contenido

Primera Parte

- Pasaje de mensajes
- PVM y MPI
- Introducción a MPI (*Message Passing Interface*)
  - MPD (*MultiProcessing Daemon*)
  - Comunicaciones: Punto a Punto, Colectivas, Bloqueantes, No Bloqueantes, otras

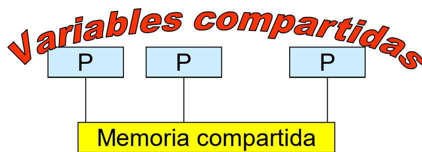
Contenido

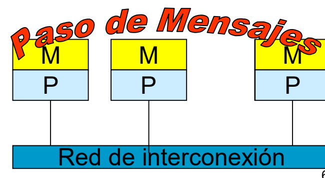
Primera Parte

- Pasaje de mensajes
- PVM y MPI
- Introducción a MPI (Message Passing Interface)
  - MPD (MultiProcessing Daemon)
  - Comunicaciones: Punto a Punto, Colectivas, Bloqueantes, No Bloqueantes, otras

Pasaje de Mensajes

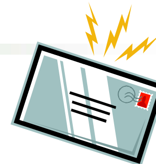
- Modelo de programación
  - Decidir el modelo de programación afecta la elección del **lenguaje** de programación y **librería** para implementar la aplicación
  - Las dos principales opciones originalmente fueron concebidas para utilizarse con la **arquitectura** paralela correspondiente





6

Pasaje de Mensajes



- **Paso de mensajes:**
  - Los datos se asocian a un nodo particular
  - De este modo para acceder a datos remotos es necesario establecer comunicaciones
  - En general para obtener datos remotos, el nodo propietario de los datos debe enviarlos y el nodo cliente debe recibirlos
  - En este modelo, las primitivas de envío y recepción encarnan el papel de la sincronización

8

Pasaje de Mensajes

- Las operaciones básicas en este paradigma son **send** y **receive**
- `send(void *sendbuf, int nelems, int dest)`
- `receive(void *recvbuf, int nelems, int source)`

```
P0

a = 100;
send(&a, 1 , 1);
a = 50;
```

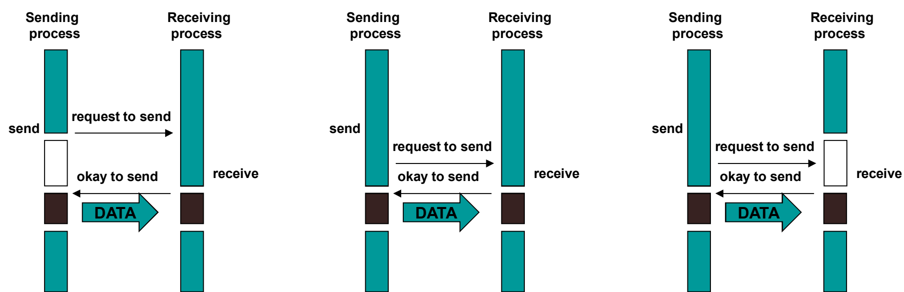
```
P1

a = 0;
receive(&a, 1 ,0);
printf("%d\n",a);
```

- ¿Qué imprime P1 por pantalla?

Pasaje de Mensajes

- Existen distintas posibilidades
  - Pasaje de mensajes bloqueante
    - Send/Receive bloqueante **sin buffer**



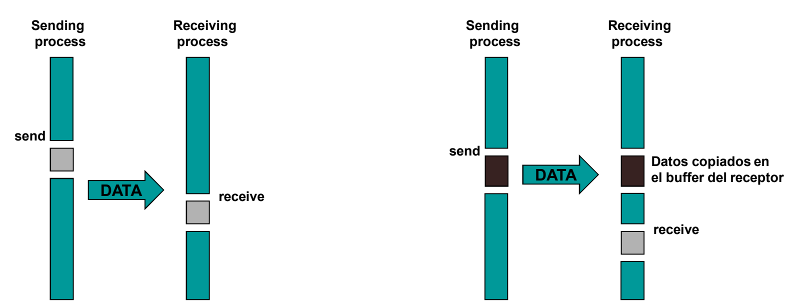
a) El sender llega primero; ocioso

b) Ambos llegan sincronizados

c) El receptor llega primero; ocioso

Pasaje de Mensajes

- Pasaje de mensajes bloqueante
  - Send/Receive bloqueante **con buffer**

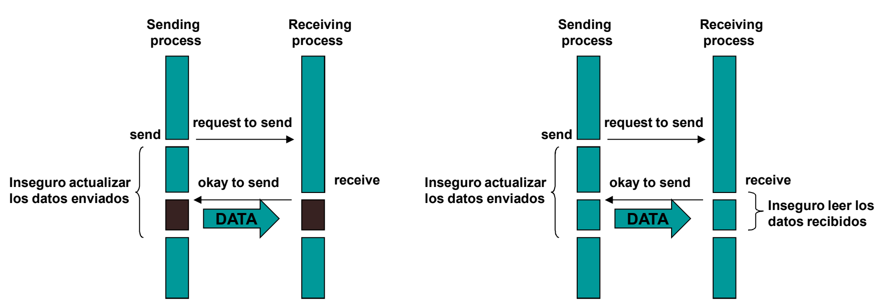


a) Con hardware de comunicación asíncrona y buffers en sender y receptor

b) Sin hardware de comunicación asíncrona; el sender interrumpe al receptor y deposita en buffer

Pasaje de Mensajes

- Pasaje de mensajes No bloqueante
  - **Sin buffer**



a) Sin hardware de soporte

a) Con hardware de soporte (asíncrono)

Contenido

Primera Parte

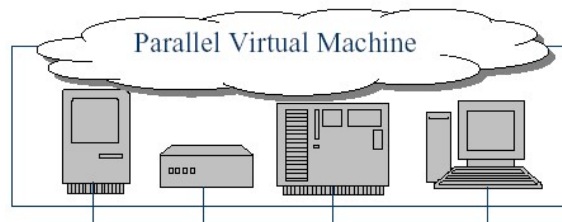
- Pasaje de mensajes
- PVM y MPI
- Introducción a MPI (Message Passing Interface)
  - MPD (MultiProcessing Daemon)
  - Comunicaciones: Punto a Punto, Colectivas, Bloqueantes, No Bloqueantes, otras

PVM y MPI

- Las librerías de comunicación que presentaremos pertenecen al caso: **paso de mensajes**
- Los **códigos** escritos utilizando este tipo de comunicación suelen ser conceptualmente **simples**, **pero** tanto el **debugging** como su operación pueden tornarse muy **complejos**
- Dos librerías que ofrecen un alto grado de portabilidad sobre distintas máquinas son:
  - **PVM**: Parallel Virtual Machine
  - **MPI**: Message Passing Interface

PVM (Parallel Virtual Machine)

- Desarrollado en *Oak Ridge National Laboratory*, fue la primera librería destinada a paralelización sobre redes de workstations Unix
- Permite a un conjunto de sistemas heterogéneos colaborar en la ejecución de un programa paralelo simulando una máquina virtual (**VM**)



PVM (Parallel Virtual Machine)

- Proporciona funciones para crear, comunicar y sincronizar procesos sobre la **VM**
- Estas funciones también incluyen estructuras de datos para enviar y recibir, como primitivas de alto nivel como **broadcast**, **scatter** – **gather** y **barreras** de sincronización
- Las aplicaciones se escriben utilizando un lenguaje procedural, como lo es **C** o **Fortran**
- Posee características de tolerancia a fallas: es fundamental que cuando se ejecutan aplicaciones y alguna tarea o host falla, el sistema continúe funcionando

PVM (Parallel Virtual Machine)

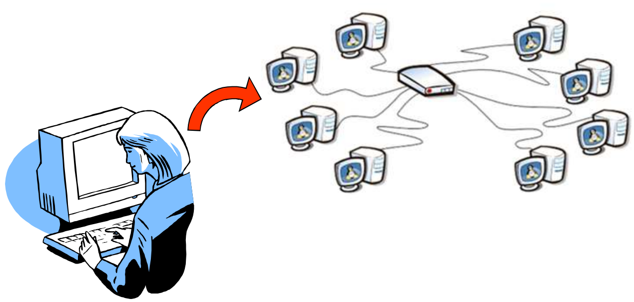
- PVM **no** es un estándar, pero es sumamente popular para realizar y desarrollar aplicaciones científicas complejas con esquema de programación en paralelo
- Componentes:
  - **pvm3d**: demonio residente en todos los hosts que conforman la máquina virtual
  - **Librería**: Repertorio de funciones para la cooperación entre tareas
- Modo de computación:
  - 1 aplicación = 1 conjunto de tareas

PVM (Parallel Virtual Machine)

- **1**. Un usuario escribe uno o más programas secuenciales en C, C++ o Fortran que contienen llamadas embebidas a la librería PVM. Cada programa se corresponde con una tarea de la aplicación.

PVM (Parallel Virtual Machine)

- **2**. Estos programas son compilados para cada arquitectura en el pool de host, y los archivos objeto resultantes son ubicados en un lugar accesible desde las máquinas en el pool.

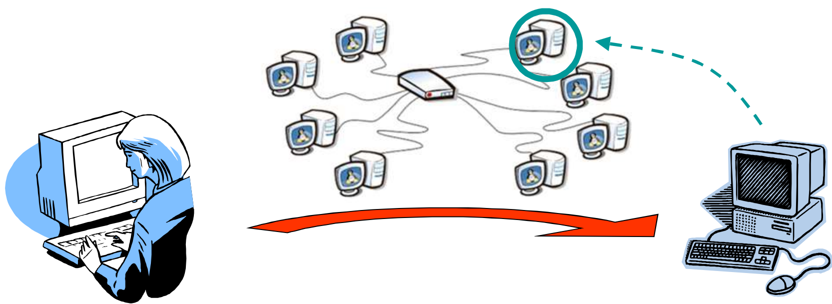


22

LICPaD (UTN-FRM) Dr. Germán Bianchini - Dra. Paola Caymes Scutari

PVM (Parallel Virtual Machine)

- **3**. Para ejecutar una aplicación, un usuario típicamente lanza una copia de una tarea (usualmente el "master" o tarea "iniciadora") manualmente desde una máquina en el pool.

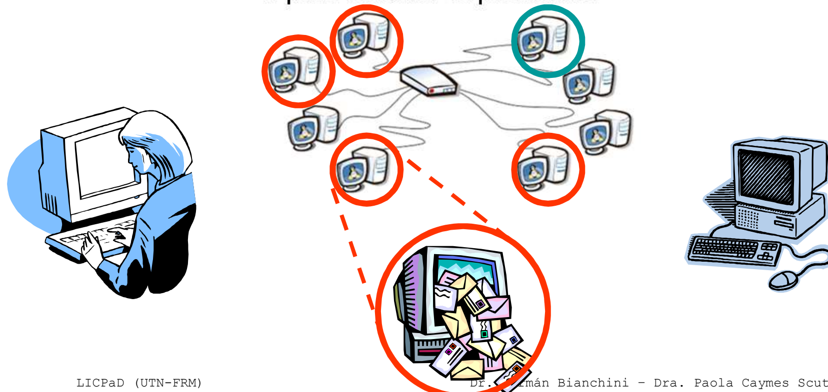


23

LICPaD (UTN-FRM) Dr. Germán Bianchini - Dra. Paola Caymes Scutari

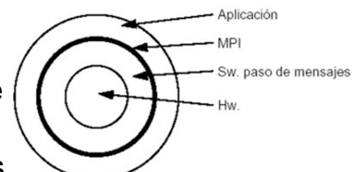
PVM (Parallel Virtual Machine)

- **4**. Este proceso subsecuentemente inicia otras tareas PVM, eventualmente resultando en una colección de tareas activas que entonces computan localmente e intercambian mensajes con las demás para resolver el problema.



MPI (Message Passing Interface)

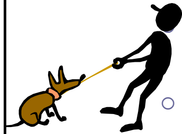
- Es un estándar desarrollado por el *MPI Forum*, un grupo formado por investigadores de empresas universidades y laboratorios involucrados en la computación de altas prestaciones
- Es una interfaz que ofrece al programador una colección de funciones para que éste diseñe su aplicación
- No es necesario conocer el hardware concreto sobre el que se va a ejecutar, ni la forma en la que se han implementado las funciones que emplea



Ubicación de MPI en el proceso de programación de aplicaciones paralelas

MPI (Message Passing Interface)

- Los objetivos fundamentales del *MPI Forum* fueron los siguientes:



  - Definir un entorno de programación único que garantice la portabilidad de las aplicaciones paralelas
  - Definir totalmente la interfaz de programación, sin especificar cómo debe ser la implementación de la misma
  - Brindar implementaciones de calidad, de dominio público, para favorecer la extensión del estándar
  - Convencer a los fabricantes de computadores paralelos para que ofrezcan versiones de MPI optimizadas para sus máquinas

MPI (Message Passing Interface)

- Razones para usar MPI en vez de PVM
  - Tiene varias implementaciones libres y de calidad (MPICH, LAM/MPI, CHIMP, OpenMPI, DeinoMPI, etc.)
  - Tiene comunicación full asíncrona, que permite solapar el cómputo
  - No hay condiciones de carrera causadas por procesos independientemente de que ingresen o abandonen un grupo
  - Administra los buffers de mensajes eficientemente
  - La sincronización protege al usuario de software de terceros
  - Se puede usar MPI eficientemente con MPP (*Massively Parallel Processors*) y clusters
  - Es altamente portable
  - MPI está especificado formalmente

MPI (Message Passing Interface)

- Concretamente ¿que agrega MPI 2?
  - E/S Paralela
  - Operaciones con acceso directo a memoria
  - Gestión dinámica de procesos

Contenido

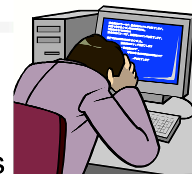
Primera Parte

- Pasaje de mensajes
- PVM y MPI
- Introducción a MPI (Message Passing Interface)
  - MPD (MultiProcessing Daemon)
  - Comunicaciones: Punto a Punto, Colectivas, Bloqueantes, No Bloqueantes, otras

Introducción a MPI

- El modelo de programación que subyace es MIMD (*Multiple Instruction Multiple Data*) aunque se dan especiales facilidades para la utilización del modelo SPMD (*Single Program Multiple Data*)
- Desde que fue completado en Junio de 1994, MPI ha ido creciendo tanto en aceptación como en uso
- Se han realizado implementaciones en una gran variedad de plataformas que van desde MPP's hasta NOW's
- Grandes compañías que se dedican a la fabricación de Supercomputadoras incluyen a MPI

Introducción a MPI



- Tamaño:
  - Contiene alrededor de **200** funciones
  - Esto no implica una relación directamente proporcional entre complejidad y número de funciones
  - Muchos programas paralelos pueden escribirse con sólo 6 funciones (*¡no es necesario ser un experto en todos los aspectos de MPI!*)
  - Sin embargo, MPI permite flexibilidad cuando es requerido, lo cual requiere mayor conocimiento

Introducción a MPI

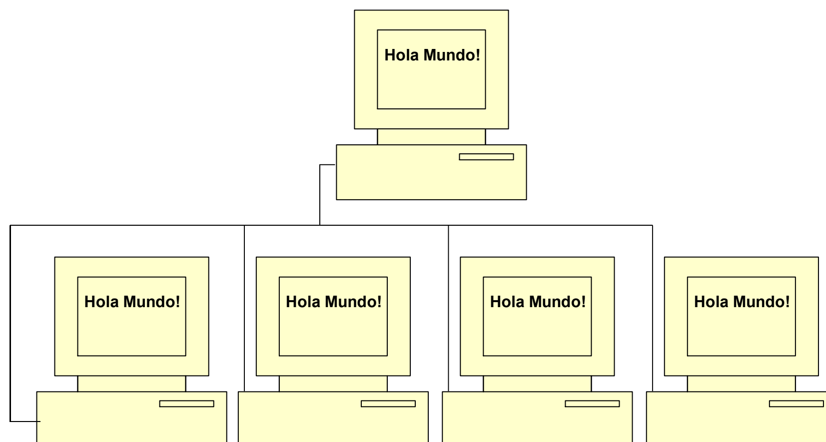
- Elementos básicos de la programación
  - MPI_Init: rutina de inicialización
  - MPI_Finalize: permite purgar o eliminar todos los estados de MPI

```
#include <stdio.h>
#include <mpi.h>
int main (int argc, char ** argv)
{
   MPI_Init(&argc, &argv);
   printf("Hola Mundo!\n");
   MPI_Finalize();
}
```

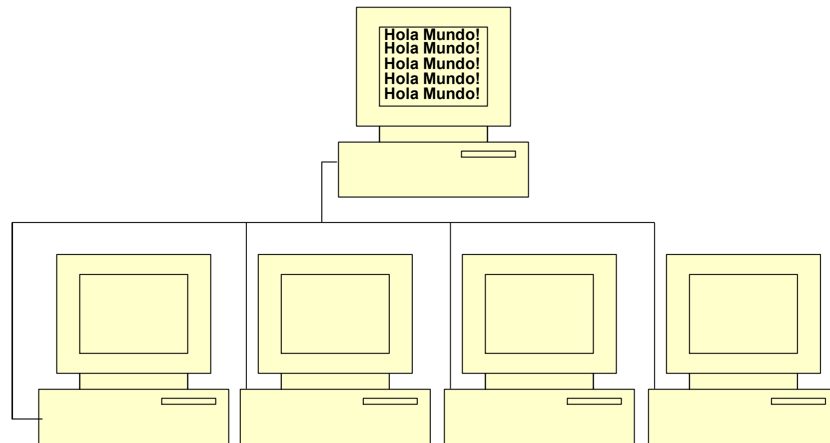
```
program main
include 'mpif.h'
integer ierr

call MPI_INIT( ierr )
print *, 'Hola Mundo!'
call MPI_Finalize( ierr )
end
```


Introducción a MPI



Introducción a MPI



Introducción a MPI

- Otras tres funciones muy útiles son:
  - MPI_Comm_size: para averiguar el número de procesos que participan en la aplicación
  - MPI_Comm_rank: para que cada proceso averigüe su dirección (identificador) dentro de la colección de procesos que componen la aplicación
  - MPI_Get_Processor_Name: retorna el nombre del procesador sobre el cual se está ejecutando

Introducción a MPI

```
#include <mpi.h>
int main (int argc, char ** argv)

 proc;    /* Numero de procesos */
 ;        /* Mi identificacion: 0<=yo<=(n
 name[30];
 sultlen;

 _Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &nproc);
 MPI_Comm_rank(MPI_COMM_WORLD, &yo);
 MPI_Get_Processor_Name(name, &resultlen);

 /* Cuerpo del programa */

 MPI_Finalize();
}
```

La mayor parte de las funciones retornan un entero, que es un diagnóstico (si es MPI_SUCCESS se ha realizado con éxito)

Esta palabra hace referencia al comunicador universal (incluye a todos los procesos)

37

Introducción a MPI

- Comando de compilación (MPICH y LAM)
  mpicc –o holamundo holamundo.c
- Ejecución (no es parte del estándar)
  mpirun –np 8 –machinefile maquinas ./holamundo
  ó mpdboot –n 8 –f maquinas
  mpiexec –n 8 ./holamundo
- ¿Y el resto de las funciones?
  - Se dispone de funciones de comunicación punto a punto (que involucran sólo a dos procesos), y de funciones u operaciones colectivas (que involucran a múltiples procesos)
  - Los procesos pueden agruparse y formar comunicadores, lo que permite una definición del ámbito de las operaciones colectivas, así como un diseño modular

38

Contenido

Primera Parte

- Pasaje de mensajes
- PVM y MPI
- Introducción a MPI (Message Passing Interface)
  - MPD (MultiProcessing Daemon)
  - Comunicaciones: Punto a Punto, Colectivas, Bloqueantes, No Bloqueantes, otras

MPD

- Convierten las workstation en una máquina virtual
- Un MPD debe correr en cada workstation
- En dicha máquina virtual se pueden ejecutar aplicaciones MPI
- Cuando se ejecuta un programa MPI en una máquina, los requerimientos son enviados a los demonios MPD para que inicien las copias del programa en las máquinas señaladas
- Una vez que las copias han arrancado, éstas pueden usar MPI para comunicarse con las demás copias dentro de la máquina virtual

MPD

- Operaciones básicas
  - **mpd** inicia un demonio mpd.
  - **mpdboot** inicia un conjunto de demonios sobre una lista de máquinas.
  - **mpdtrace** lista todos los demonios mpd que están corriendo. La opción –l muestra los nombres completos de los hosts y el puerto donde el mpd está escuchando.
  - **mpdallexit** termina todos los demonios mpd de la máquina virtual

MPD

- Ejemplo sobre una máquina

```
mpd &
mpiexec -n 3 ./miprogramampi
mpdallexit
```

- Ejemplo sobre tres máquinas

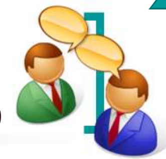
```
mpdboot -n 3 -f mismaquinas
mpiexec -n 3 ./miprogramampi
mpdallexit
```

Contenido

Primera Parte

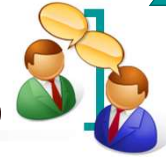
- Pasaje de mensajes
- PVM y MPI
- Introducción a MPI (Message Passing Interface)
  - MPD (MultiProcessing Daemon)
  - Comunicaciones: Punto a Punto, Colectivas, Bloqueantes, No Bloqueantes, otras

Comunicación punto a punto



- Es el mecanismo básico de transferencia de mensajes entre un par de procesos
- Funciones de envío y recepción permiten transferir datos de cierto tipo con una etiqueta asociada
- El tipo de dato del contenido se hace necesario para dar soporte a la heterogeneidad
- La información del tipo es utilizada para realizar las conversiones en la representación cuando se envían datos de una arquitectura a otra
- La etiqueta, del lado del receptor, le permite seleccionar un mensaje en particular

Comunicación punto a punto



MPI_Send(buf, count, datatype, dest, tag, comm)

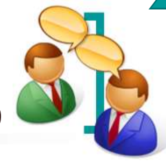
- buf: dirección del buffer de envío
- count: número de elementos a enviar
- datatype: tipo de datos de los elementos del buffer de envío
- dest: identificador del proceso destino
- tag: etiqueta del mensaje
- comm: comunicador

MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

45

LICPaD (UTN-FRM) Dr. Germán Bianchini - Dra. Paola Caymes Scutari

Comunicación punto a punto



MPI_Recv(buf, count, datatype, source, tag, comm, status)

- buf: dirección del buffer de recepción
- count: número de elementos a recibir
- datatype: tipo de datos de los elementos del buffer de recepción
- source: identificador del proceso que envía o MPI_ANY_SOURCE
- tag: etiqueta del mensaje o MPI_ANY_TAG
- comm: comunicador
- status: objeto status

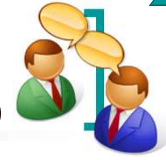
MPI_Recv(void *buf, int count,MPI_Datatype datatype,int source,int tag,MPI_Comm comm, MPI_Status status)

status.MPI_SOURCE = identificador del enviador
status.MPI_TAG = la etiqueta
status.MPI_ERROR = el código de error

46

LICPaD (UTN-FRM) Dr. Germán Bianchini - Dra. Paola Caymes Scutari

Comunicación punto a punto



- Ejemplo "saluda al sucesor"

```
#include <stdio.h>
#include <mpi.h>
int my_id, nproc, tag=99, source;
MPI_Status status;
int main (int argc, char** argv)
{
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
  MPI_Comm_size(MPI_COMM_WORLD, &nproc);

  MPI_Send(&my_id,1,MPI_INT,(my_id+1)%nproc,tag,MPI_COMM_WORLD);
  MPI_Recv(&source,1,MPI_INT,MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,status)
  printf("%d recibo mensaje de %d\n", my_id,source);
  MPI_Finalize();
}
```

(my_id+nproc-1)%nproc

¡Sólo hemos usado 6 funciones!

Comunicaciones colectivas

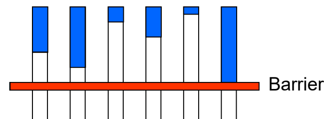
- En ciertas aplicaciones se requiere enviar un conjunto de datos a múltiples procesos
- Las comunicaciones colectivas permiten la transferencia de datos entre todos los procesos que pertenecen a un grupo específico
- En este caso, no se usan etiquetas, se sustituyen por comunicadores
- Las operaciones colectivas se pueden clasificar en tres clases:
  - **Sincronización**. Barreras para sincronizar
  - **Movimiento de datos** (difundir, recolectar, esparcir)
  - **Cálculos colectivos**. Operaciones para reducción global

Comunicaciones colectivas

- Sincronización

MPI_Barrier(MPI_Comm comm)

  - Es una operación que no exige ningún cambio de información
  - Es una operación puramente de sincronización
  - Detiene la ejecución de cada proceso que la llama hasta tanto todos los procesos incluidos en el grupo asociado a *comm* la hayan llamado



Comunicaciones colectivas

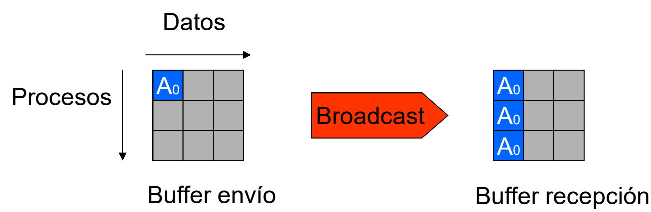
- Movimiento de datos
  - Broadcast
    - Sirve para que un proceso, el raíz, envíe un mensaje a todos los miembros del comunicador

MPI_Bcast(void *inbuf, int incnt, MPI_Datatype intype, int root, MPI_Comm comm)

- inbuf: dirección del buffer de recepción, o envío en root
- incnt: número de elementos en el buffer de envío
- intype: tipo de dato de los elementos del buffer de envío
- root: rango del proceso root
- comm: comunicador

Comunicaciones colectivas

- Esquema de la operación **MPI_Broadcast**



Comunicaciones colectivas

- Movimiento de datos
  - Gather
    - Realiza una recolección de datos en el proceso raíz
    - Este proceso recopila un vector de datos al que contribuyen todos los procesos del comunicador con la misma cantidad de datos
    - El proceso raíz almacena las contribuciones de forma consecutiva

Comunicaciones colectivas

MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype senddatatype, void *rcvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);

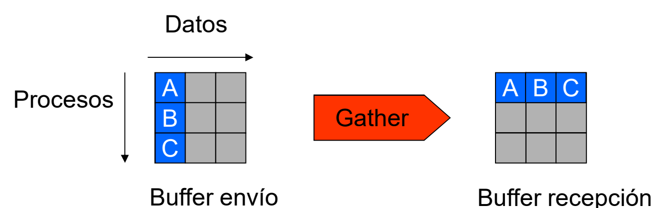
- sendbuf: dirección del buffer de envío
- sendcount: número de eltos. del buffer de envío
- senddatatype: tipo de dato de los eltos. del buffer
- recvbuf: dirección del buffer de recepción
- recvcount: número de eltos. a recibir de cada uno
- recvtype: tipo de dato de los eltos. del buffer de recepción
- root: rango del proceso raíz
- comm: comunicador

53

LICPaD (UTN-FRM) Dr. Germán Bianchini - Dra. Paola Caymes Scutari

Comunicaciones colectivas

- Esquema de la operación **MPI_Gather**



54

LICPaD (UTN-FRM) Dr. Germán Bianchini - Dra. Paola Caymes Scutari

Comunicaciones colectivas

- Movimiento de datos
  - Scatter
    - Realiza la operación simétrica a MPI_Gather
    - El proceso raíz posee un vector de elementos, uno por cada proceso del comunicador
    - Tras realizar la distribución, cada proceso tiene un elemento del vector inicial

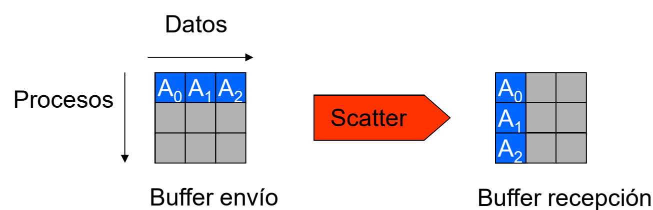
MPI_Scatter(void* sendbuf, int sendcount, MPI_Datatype senddatatype, void *rcvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);

Comunicaciones colectivas

- Esquema de la operación **MPI_Scatter**

Datos

Procesos



Buffer envío

Buffer recepción

Comunicaciones colectivas

- Cálculos colectivos
  - Reduce
    - Combina los elementos provistos en el buffer de entrada (*sendbuf*) de cada proceso en el grupo, usando la operación *op*
    - Se obtiene un resultado final combinado que se almacena en el proceso raíz (*recvbuf*)

MPI_Reduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm);

Comunicaciones colectivas

- MPI define una colección de operaciones del tipo MPI_Op:
  - **MPI_MAX**: máximo
  - **MPI_MIN**: mínimo
  - **MPI_SUM**: suma
  - **MPI_PROD**: producto
  - **MPI_LAND**: and lógico
  - **MPI_BAND**: and bit a bit
  - **MPI_LOR**: or lógico
  - **MPI_BOR**: or bit a bit
  - **MPI_LXOR**: xor lógico
  - **MPI_BXOR**: xor bit a bit

Com. bloq. y no bloqueantes

- Las funciones básicas de MPI para el envío y recepción bloqueado son **MPI_Send** y **MPI_Recv**
- Esto significa que una llamada al *send* bloquea al proceso hasta que el buffer pueda ser reutilizado (dicho de otra forma, liberado)
- En el caso del *receive*, se bloquea hasta que el buffer de recepción realmente contenga el mensaje enviado
- Por lo tanto, el orden en el que aparecen las llamadas es importante: **1º send y 2º receive**

Com. bloq. y no bloqueantes

- MPI también provee comunicación **no bloqueante**
- Son funciones que permiten solapar la transmisión del mensaje con otros cómputos, o con la transmisión de otros mensajes
- Las funciones no bloqueantes tienen dos partes:
  - La función a realizar (envío o recepción)
  - El test de completitud de la función
- Manejan un objeto ***request*** del tipo MPI_Request
  - Es una especie de recibo de la operación
  - Sirve para consultar si la operación se ha terminado

Com. bloq. y no bloqueantes

MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)

Inmediato

- buf: dirección del buffer de envío
- count: número de elementos a enviar
- datatype: tipo de datos de los elementos del buffer de envío
- dest: identificador del proceso destino
- tag: etiqueta del mensaje
- comm: comunicador
- request: recibo para consulta

61

Com. bloq. y no bloqueantes

- La función MPI_Wait toma como entrada un recibo y bloquea al proceso hasta que la operación termina

  MPI_Wait(MPI_Request *request, MPI_Status *status)
  - request: manejador del pedido (recibo)
  - status: objeto estado

- La función MPI_Test indica si la operación ha terminado, pero no se bloquea

  MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
  - flag: contendrá true si la operación se ha completado

62

Enviar y recibir a la vez

- La comunicación entre dos procesos (del tipo intercambio) suele aparecer frecuentemente, por tal motivo surge:

  MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype senddatatype, int dest, int sendtag, void * recvbuf, int recvcount, MPI_Datatype recvdatatype, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

  - Los argumentos son esencialmente la combinación de los argumentos de MPI_Send y MPI_Recv
  - Los buffers de envío y destino deben ser disjuntos

Enviar y recibir a la vez

- A veces, este tipo de comunicación puede forzar un uso de buffer temporal, y por ende se incrementa la cantidad de memoria, por eso, surge otra opción:

  MPI_Sendrecv_replace(void *buf, int count, MPI_Datatype datatype, int dest, int sendtag, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

  - Esta función realiza un *send* bloqueante y efectúa un *receive* usando el mismo buffer
  - En este caso, es importante notar que ambas operaciones (*send* y *receive*) deben transferir datos del mismo tipo

Referencias

- **Geist Al et al.** – **"***PVM: Parallel Virtual Machine – A User's guide an tutorial for networked parallel computing"* The MIT Press, 1995 Massachusetts Institute of Technology
- **Grama A. et al**. – "*Introduction to Parallel Computing Second Edition*" Addison-Wesley, 2003
- **Gropp W**. et al. – "*MPICH2 User's Guide Version 1.0.7 - Mathematics and Computer Science Division*" Argonne National Laboratory. 2008.
- **Morrison R. S. – "***Cluster Computing*", 2002
- **Pacheco P.S.** – "Parallel Programming with MPI" Morgan Kaufmann Publishers, Inc. 1997
- **Snir Marc et al**. – "*MPI: The Complete Reference*" The MIT Press, 1996 Cambridge Massachusetts, London England