Universisdad de Los Andes
Facultad de Ingeniería
Escuela de Sistemas

Message Passing Interface
MPI

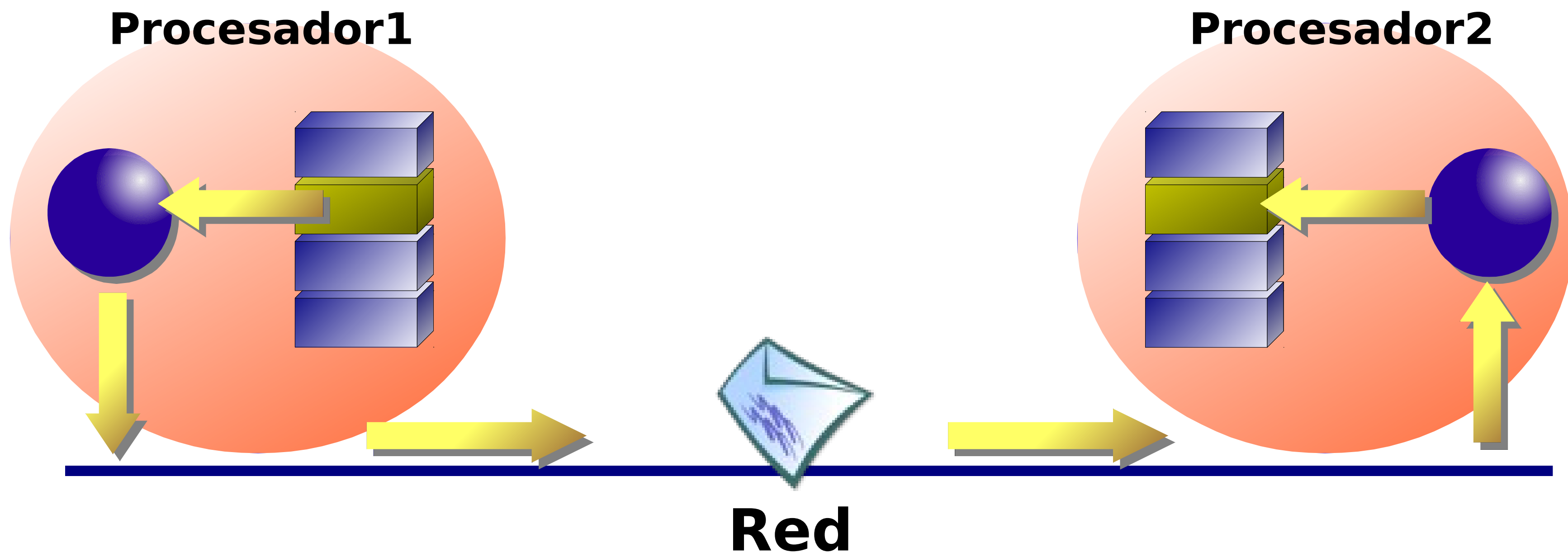
Prof. Gilberto Díaz
gilberto@ula.ve

Departamento de Computación, Escuela de Sistemas, Facultad de Ingeniería
Universidad de Los Andes, Mérida 5101 Venezuela

MPI es un estándar para la implementación de sistemas de pase de mensajes diseñado por un grupo de investigadores de la industria y la academia para funcionar en una amplia variedad de computadores paralelos y de forma tal que los códigos sean portables.

Message Passing Interface

Su diseño esta inspirado en máquinas con una arquitectura de memoria distribuida en donde cada procesador es propietario de cierta memoria y la única forma de intercambiar información es a través de mensajes.



Sin embargo, hoy en día también encontramos implemetaciones de MPI en máquinas de memoria compartida.

El estándar define la sintaxis y semántica de un conjunto de rutinas útiles

Dentro de las implantaciones más populares de MPI se encuentran:

- MPICH
- LAM
- OpenMPI

Trabajaremos con OpenMPI pues es auspiciada por un grupo importante de instituciones

Los principales objetivos de MPI:

- Proporcionar código portátil
- Proporcionar implementaciones eficientes
- Soportar arquitecturas paralelas heterogeneas

UNIVERSIDAD DE LOS ANDES MERIDA VENEZUELA

Instalación de OpenMPI

El proceso de instalación es bastante sencillo. Siga los siguientes pasos para llevar a cabo este proceso:

- Descargue los fuentes

```
cd /usr/local/src/
wget http://www.open-mpi.org/software/ompi/v1.3/downloads/ openmpi-1.3.2.tar.bz2
```

- Desempaquete los fuentes

```
tar xvjf openmpi-1.3.2.tar.bz2
```

Instalación de OpenMPI

El proceso de instalación es bastante sencillo. Siga los siguientes pasos para llevar a cabo este proceso:

- Configure el software

```
cd openmpi-1.3.2
./configure  - -prefix=/opt/openmpi/gnu
```

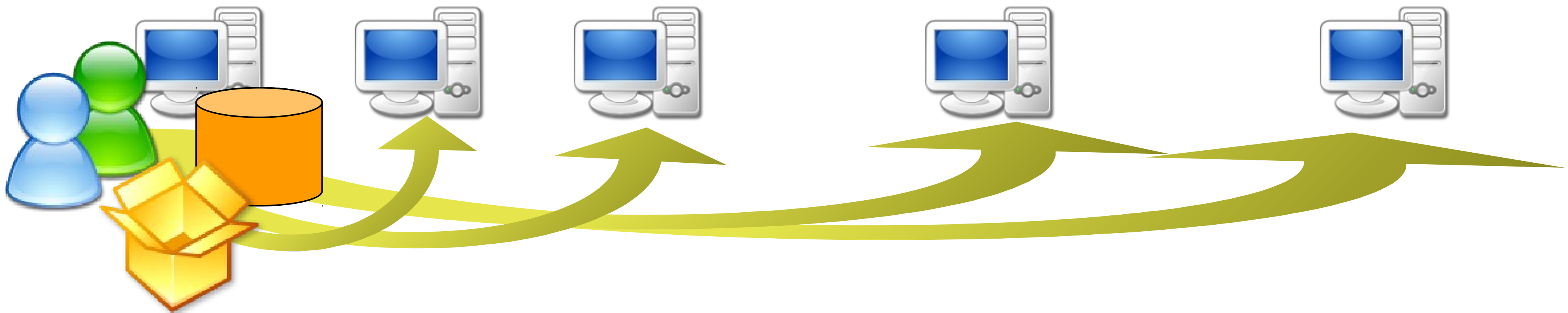
- Compile el software e instalelo

```
make && make install
```

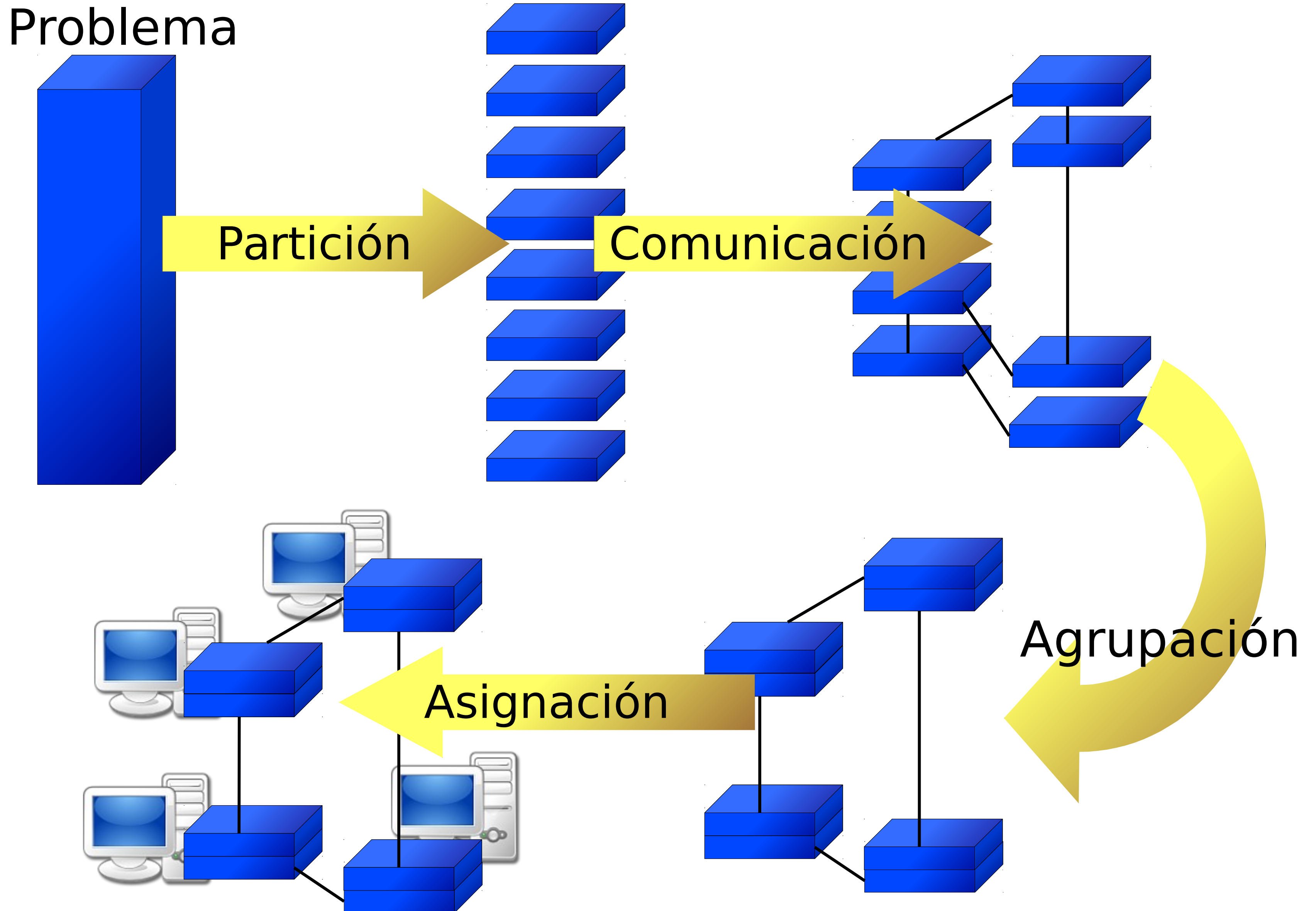
Cómo trabajar con MPI

Para poder ejecutar un programa paralelo es necesario configurar el ambiente apropiado:

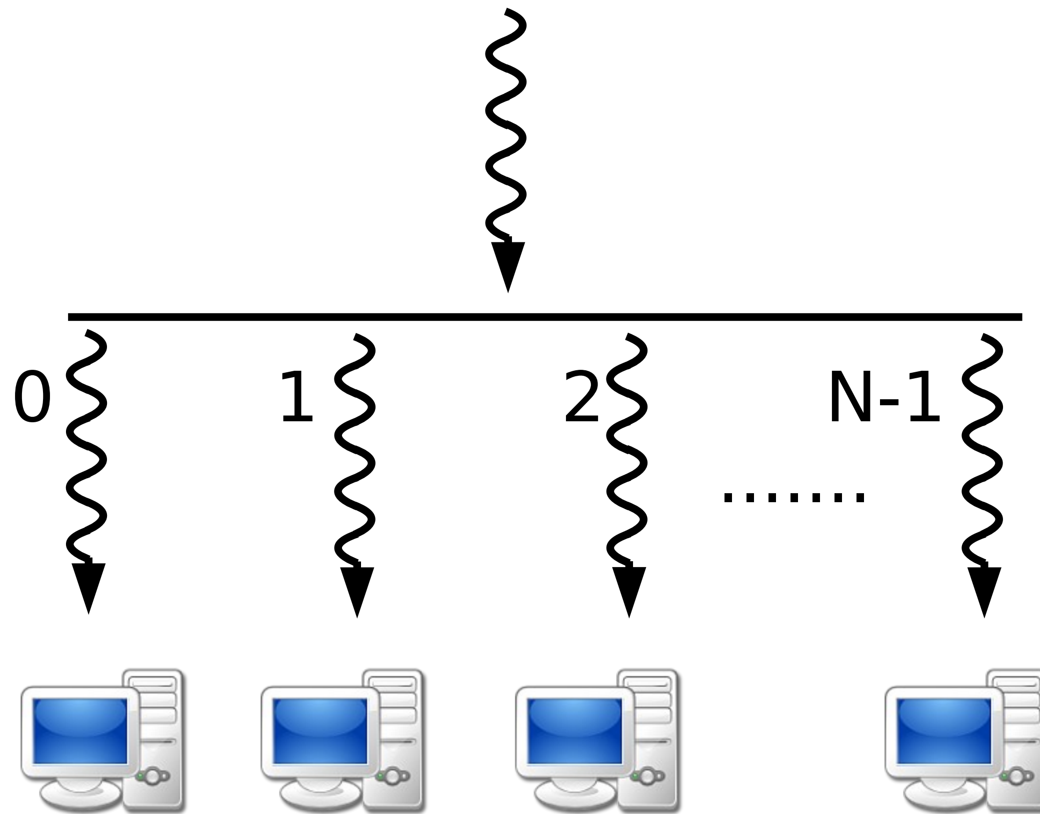
- Un sistema de archivos común donde se encuentren todos los ficheros involucrados en la ejecución
- Un sistema de autenticación de usuarios que permita el acceso sin contraseña a los nodos participantes



Cómo trabajar con MPI



Cómo trabajar con MPI



Paralelismo a nivel de datos

$$
\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}
\times
\begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}
=
\begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix}
$$

Paralelismo a nivel de datos

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} x \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix}$$

Paralelismo a nivel de datos

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix}$$

Paralelismo a nivel de datos

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix}$$

Paralelismo a nivel de datos

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$$

$$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$$

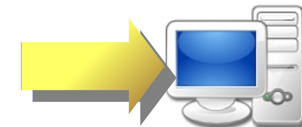
$$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$$

....

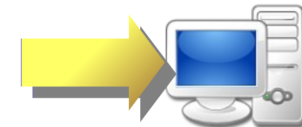
$$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$$

Paralelismo a nivel de datos

$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$



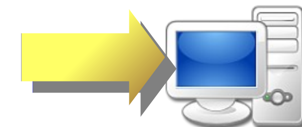
$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$



$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$

....

$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$



Cómo trabajar con MPI

Las implementaciones de MPI son bibliotecas

- Todas las operaciones son ejecutadas realizando invocaciones a funciones
- Debemos incluir los siguientes archivos cabecera (headers)
  - mpi.h en C y C++
  - mpif.h para Fortran77 y Fortran90

Cómo trabajar con MPI

Podemos categorizar las funciones de MPI de la siguiente manera:

- Funciones de gestión
  - Inicializar y terminar el ambiente paralelo
- Funciones para la comunicación entre pares de procesos (comunicación punto a punto)
- Funciones para la comunicación entre grupos de procesos (comunicación colectiva)
- Funciones para crear tipos de datos

Funciones de Gestión del Ambiente Paralelo

- Inicializar el ambiente de ejecución paralelo

  MPI_Init(&argc, &argv)

- Identificación dentro del ambiente paralelo

  MPI_Comm_rank(MPI_COMM_WORLD, &my_id)

- Obtener el número de procesos que realmente se pudo iniciar

  MPI_Comm_size(MPI_COMM_WORLD, &nproc)

Inicio del programa

- Finalizar el ambiente de ejecución paralelo

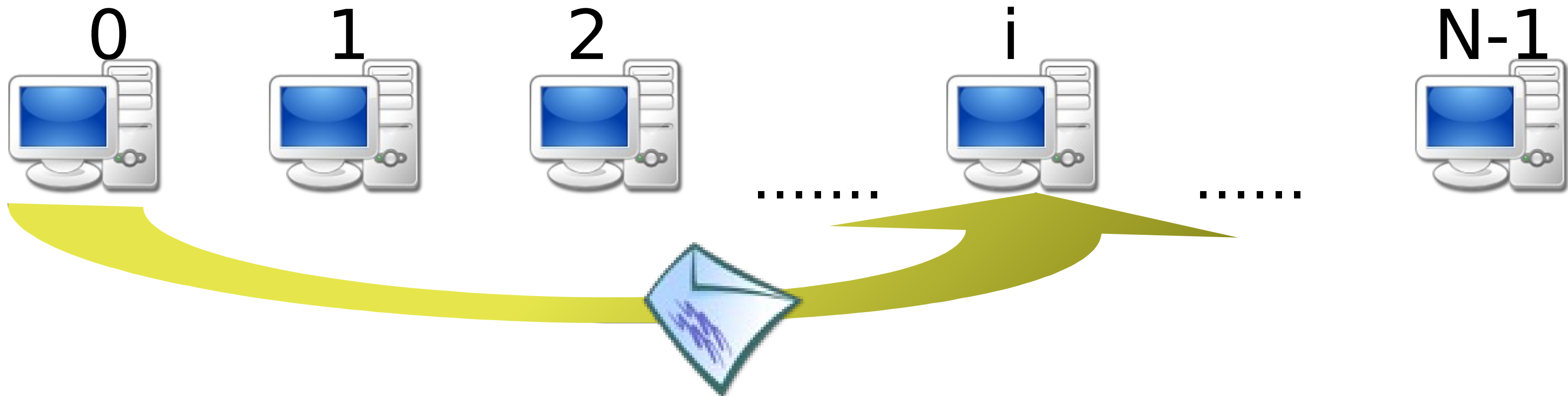
  MPI_Finalize()

Fin del programa

Funciones Básicas de Envío y Recepción

- Función para enviar mensajes

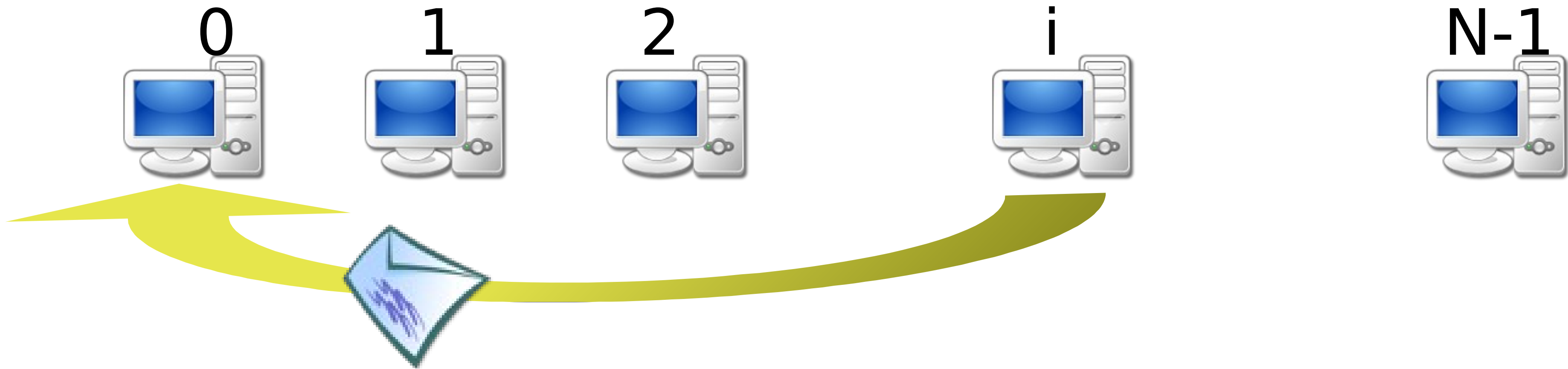
MPI_Send(&a,size,MPI_INT,i,tag,COMM)



Funciones Básicas de Envío y Recepción

- Función para recibir mensajes

MPI_Recv(&r,1,MPI_INT,i,tag,comm,&stat)



Un Programa ejemplo

```
#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,tag,
MPI_COMM_WORLD, &status);
    printf("%d recibio mensaje de %d\n",my_id,source);
    MPI_Finalize();
}
```

Cómo Compilar un programa en MPI

El proceso de compilación difiere entre las distintas implantaciones de MPI. Generalmente, el comando para compilar incluye todas las banderas del compilador necesarias para incluir tanto los archivos cabecera como las bibliotecas.

Ejemplos:

```
mpicc programa.c -o prog.exe

mpic++ programa.c -o prog.exe

mpif77 programa.f -o prog.exe

pif90 programa.f90 -o prog.exe
```

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Cómo Ejecutar un programa en MPI

El proceso de ejecución en la mayoría de las implantaciones se realiza de la siguiente manera:

mpirun *–machinefile* machines *-np* 4 p.exe

En general, ejecutar un programa en MPI es independiente de la implantación y puede necesitar varios scripts, argumentos, y variables de ambiente

Cómo Ejecutar un programa en MPI

mpirun *–machinefile* machines *-np* 4 p.exe

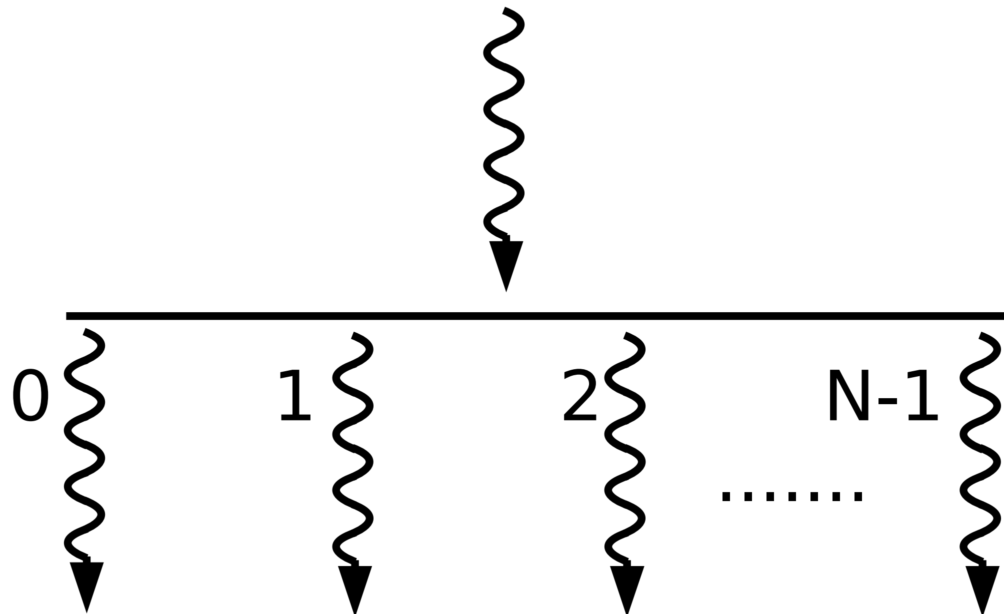
Archivo de computadores donde se ejecutaran los procesos.

Se coloca el nombre o dirección IP de un computador en cada línea

Cómo Ejecutar un programa en MPI

mpirun *–machinefile* machines *-np* 4 p.exe

Número de procesos que se crearán



Cómo Ejecutar un programa en MPI

En el estándar MPI-2 existe una forma distinta de ejecutar un programa paralelo

mpiexec argumentos

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Elementos importantes de un Programa MPI

Encontramos los siguientes elementos:

- Los archivos cabecera (headers)
- El entorno de comunicación (MPI Communicator)
- Formato de las funciones de MPI
- Tamaño del entorno de comunicación
- El identificador de cada proceso dentro del entorno de comunicación
- Inicialización y finalización del programa

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

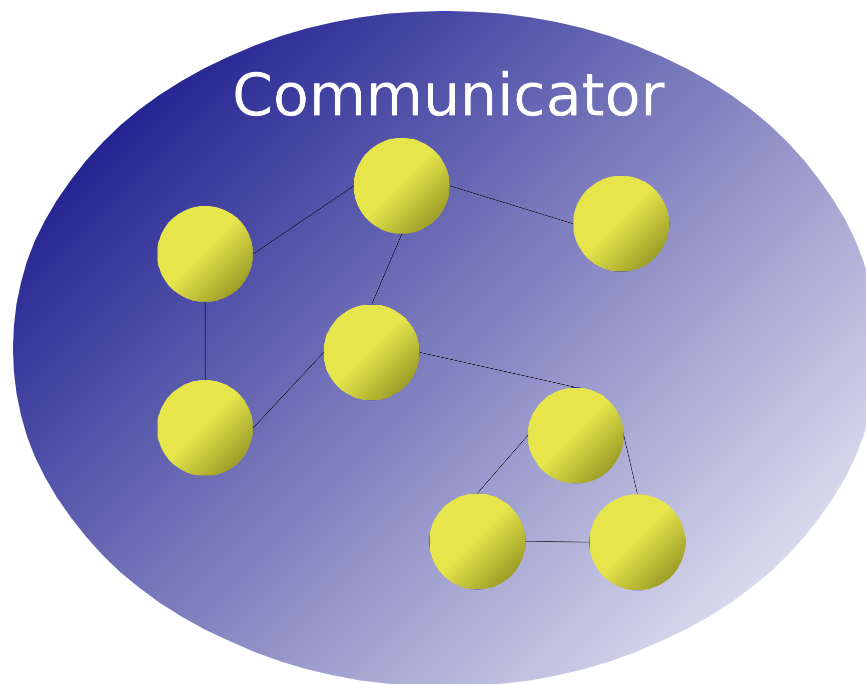
Elementos importantes de un Programa MPI

Los archivos cabecera contienen

- Definiciones de los tipos de datos
- Los prototipos de las funciones
    - mpi.h en C
    - mpif.h para Fortran77 y Fortran90

Elementos importantes de un Programa MPI

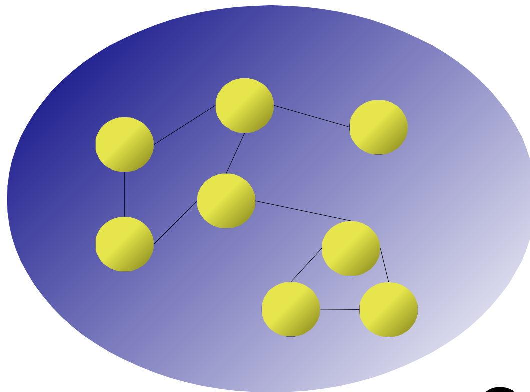
El entorno de comunicación (MPI Communicator) es definido como una variable mediante la cual se puede hacer referencia a un grupo de procesos que se comunican entre si.



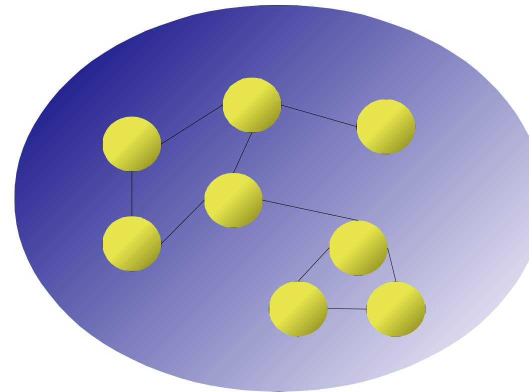
Elementos importantes de un Programa MPI

El programador puede definir múltiples entornos de comunicación

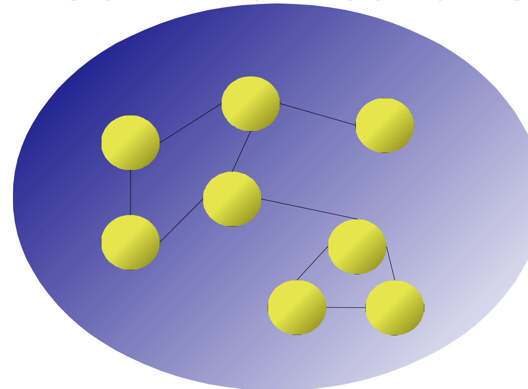
Communicator 1



Communicator 2



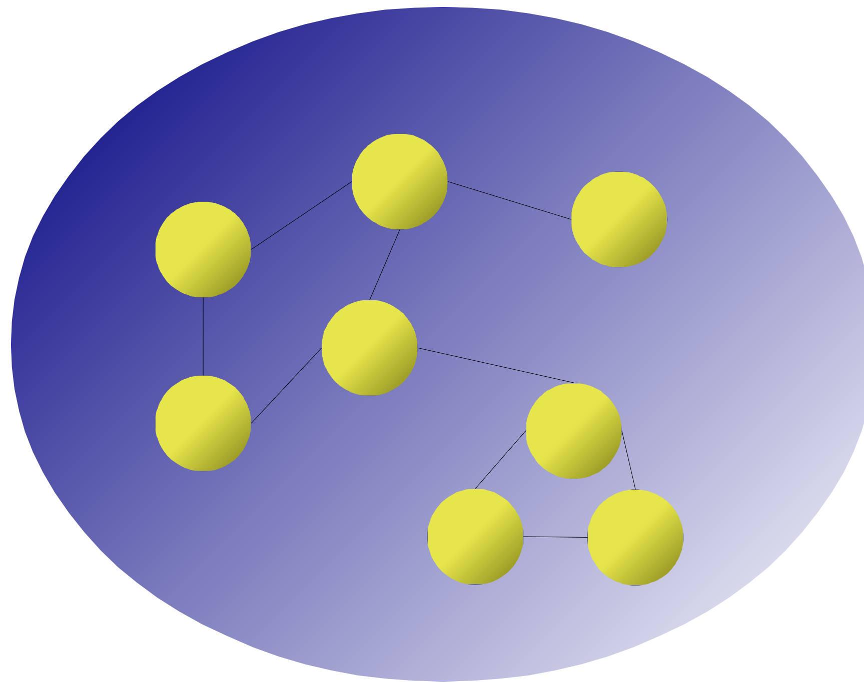
Communicator 3



Elementos importantes de un Programa MPI

Existe un entorno de comunicación **predefinido**

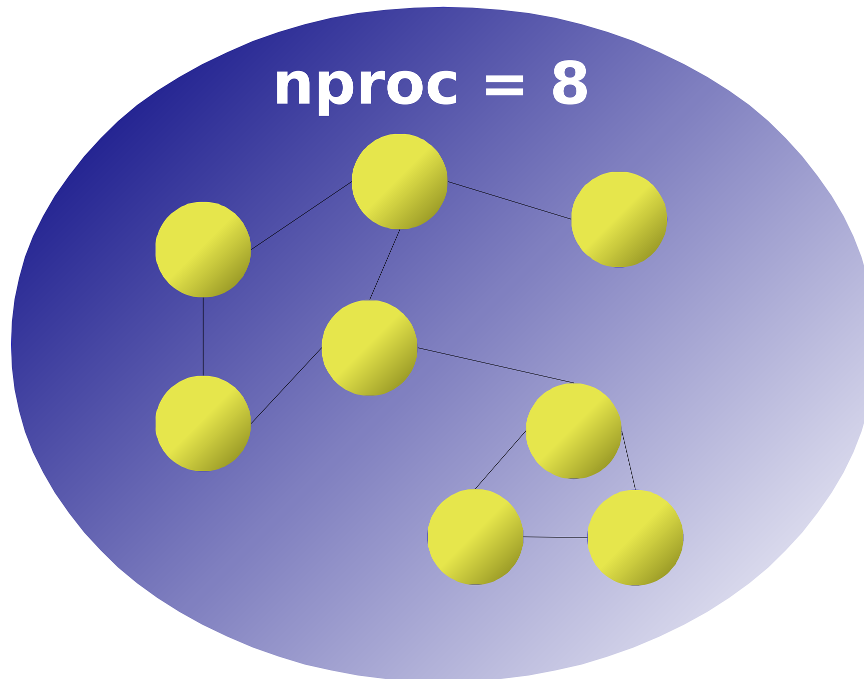
MPI_COMM_WORLD



Elementos importantes de un Programa MPI

El tamaño del entorno de comunicación se refiere al número de procesos que se lograron arrancar al momento de la inicialización.

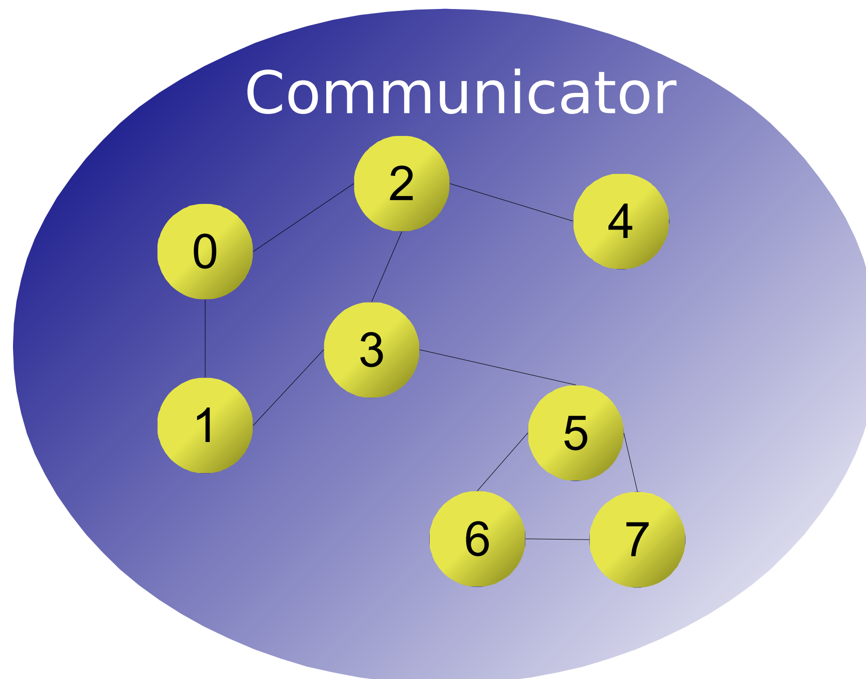
MPI_Comm_size(MPI_COMM_WORLD, &nproc)



Elementos importantes de un Programa MPI

El identificador distingue cada proceso dentro del entorno de comunicación:

```
MPI_Comm_rank(MPI_COMM_WORLD, &my_id)
```



UNIVERSIDAD DE LOS ANDES MERIDA VENEZUELA

Elementos importantes de un Programa MPI

- Los procesos pueden ser reunidos en un grupo
- Los mensajes se transmiten dentro de un contexto determinado
- Un grupo y un contexto conforman el entorno de comunicación

Elementos importantes de un Programa MPI

- Cada proceso es identificado dentro del grupo que pertenece al entorno de comunicación a través de su ID

- El entorno predefinido contiene todos los procesos iniciales MPI_COMM_WORLD

Tipos de Datos en MPI

Las funciones de envío y recepción gestionan los datos utilizando los tres primeros argumentos:

- Dirección de memoria donde están los datos
- Tamaño de la memoria
- Tipo de dato

```
MPI_Send(&buffer, size, MPI_INT,.....
MPI_Recv(&buffer, size, MPI_INT,.....
```

Tipos de Datos en MPI

Dentro de los tipos de datos que podemos encontrar en MPI tenemos:

MPI_BYTE
MPI_INT
MPI_CHAR
MPI_SHORT
MPI_FLOAT
MPI_DOUBLE
MPI_LONG_DOUBLE
MPI_UNSIGNED.....

Existen funciones dentro de MPI para construir tipos de datos propios del usuario

Etiquetas de los Mensajes

En MPI los mensajes contienen una etiqueta (TAG) que ayuda al proceso receptor a identificar el mensaje.

Cada mensaje puede ser categorizados en la recepción por una etiqueta específica o simplemente indicarse que puede tener cualquier categoría MPI_ANY_TAG

Funciones Básicas de Envío y Recepción

- Función para enviar mensajes

MPI_Send(*&a,n,MPI_INT,dest,tag,COMM*)

Donde

a son los datos a enviar

n es el número de elementos en *a*

MPI_INT el tipo de datos

dest identificador del proceso al que se le enviará los datos

tag es la etiqueta del mensaje

COMM es el entorno de comunicación

Funciones Básicas de Envío y Recepción

- Función para recibir mensajes

MPI_Recv(*&a,n,MPI_INT,source,tag,comm,&stat*)

Donde

a variable donde se reciben los datos
n es el número de elementos en *a*
MPI_INT el tipo de datos
source identificador del proceso que envió los datos
tag es la etiqueta del mensaje
comm es el entorno de comunicación
stat arreglo de enteros con información sobre el mensaje en caso de error

Funciones Básicas de Envío y Recepción

La completación del proceso de comunicación significa que la memoria utilizada en la transferencia del mensaje se puede reutilizar.

- En el emisor: la variable enviada se puede reutilizar
- En el receptor: la variable recibida se puede leer

Modos de comunicación

El proceso de comunicación en MPI tiene dos formas o modos de llevarse a cabo

- Bloqueante: las rutinas de envío y recepción retornan cuando se ha completado la comunicación.
- No bloqueante: las funciones retornan inmediatamente. El usuario debe chequear si se ha completado la comunicacón antes de utilizar el contenido de una variable.

Funciones Básicas de Envío y Recepción

Las funciones MPI_Send y MPI_Recv son bloqueantes:

Cuando MPI_Send retorna entonces se puede seguir utilizando el contenido del buffer (datos enviados) y modificarlos si es necesario.

No se tiene conocimiento si el nodo destino recibió el mensaje.

Funciones Básicas de Envío y Recepción

Las funciones MPI_Send y MPI_Recv son bloqueantes:

MPI_Recv retornará sólo cuando se hayan recibido todos los datos en el buffer de recepción.

Funciones Básicas de Envío y Recepción

La implementaciones de MPI proporcionan también funciones no bloqueantes para en envío y recepción de mensajes.

MPI_Isend(....MPI_Request **req*)

MPI_Irecv(....MPI_Request **req)*

Donde:

req es el manejador de la solicitud de la comunicación

Funciones Básicas de Envío y Recepción

MPI_Isend y MPI_Irecv permiten que la ejecución del programa continue y se pueda realizar otras tareas.

Sin embargo, es necesario que el proceso de comunicación sea completado para poder utilizar el contenido de la variable que se envió o se recibió.

El usuario debe chequear si se ha completado la comunicacón antes de utilizar el contenido de una variable.

Verificando la Completación de la Comunic.

Cuando se utilizan las funciones no bloqueantes el usuario debe agregar código para verificar si la comunicación se ha completado.

MPI_Test(MPI_Request *req, int *flag, MPI_Status *stat)

Donde:

req es el manejador de la comunicación
flag es verdadera si se completó la comunic.
stat es el estado del objeto

Verificando la Completación de la Comunic.

Si no se puede realizar otra tarea en el programa, pues se necesita el dato del mensaje, entonces se debe esperar por este.

MPI_Wait(MPI_Request *req, MPI_Status *stat)

Donde:

req es el manejador de la comunicación
stat es el estado del objeto

UNIVERSIDAD DE LOS ANDES MERIDA VENEZUELA

Transferencia Sincrónica de Mensajes

Las implantaciones de MPI proporcionan también una forma de comunicación sincrónica.

Esto quiere decir que la comunicación no se completará hasta que el mensaje sea recibido por el nodo destino.

MPI_Ssend	bloqueante
MPI_Issend	no bloqueante

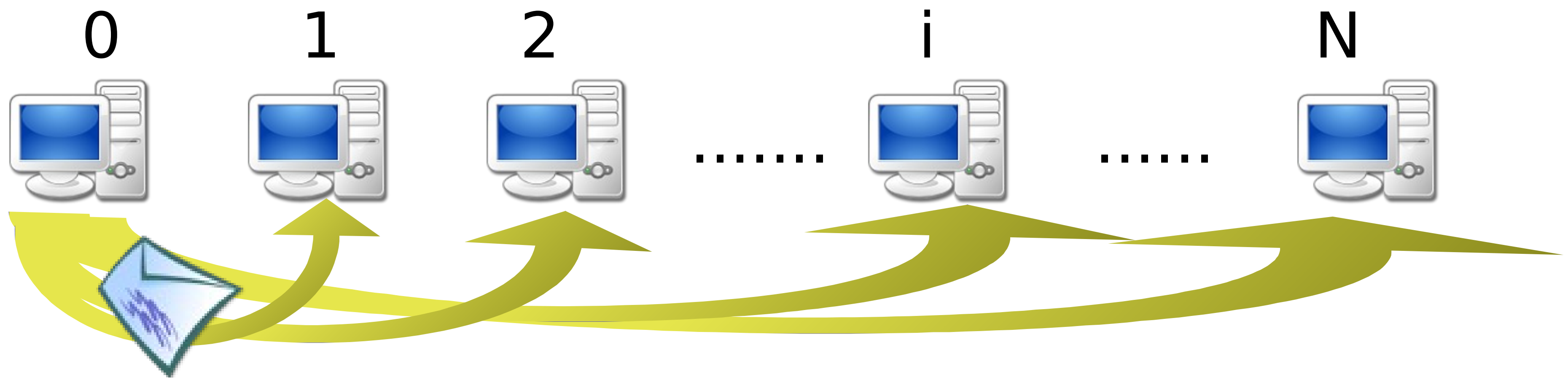
Comunicaciones Colectivas

En ciertas aplicaciones se requiere enviar un conjunto de datos a múltiples procesos o recibir en un único proceso datos de varios remitentes.

Las comunicaciones colectivas permiten la transferencia de datos entre todos los procesos que pertenecen a un grupo específico.

Comunicaciones Colectivas

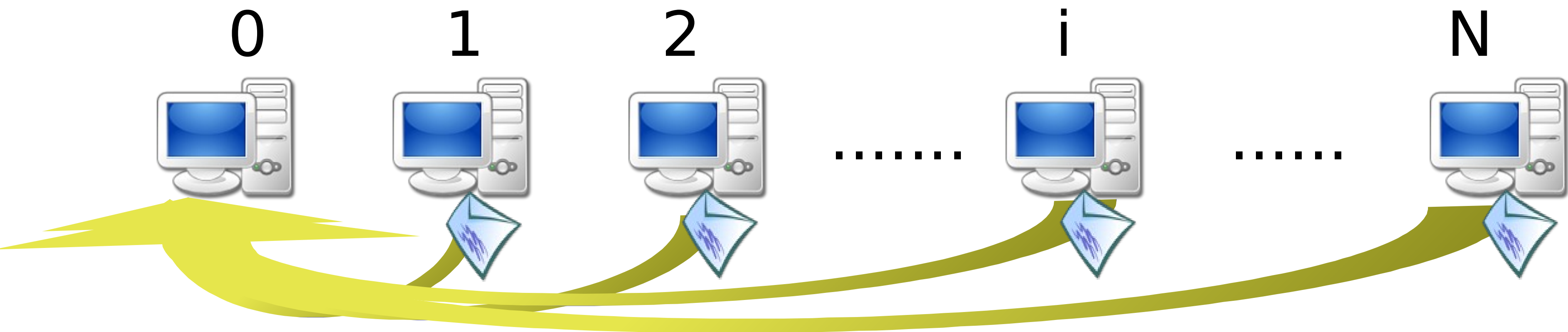
Por ejemplo: enviar un mismo mensaje al resto de los procesos



MPI_Bcast(buffer, count, datatype, src, comm)

Comunicaciones Colectivas

Por ejemplo: recibir mensajes del resto de los procesos



MPI_Reduce(*sendbuf, *recvbuf, count, type, MPI_Op op, src, comm)

Comunicaciones Colectivas

MPI_Op especifica el tipo de operación que se va a ejecutar con los datos que se reciben de cada nodo.

MPI tiene operaciones predefinidas como:

- Suma
- Máximo
- AND lógico, etc

El usuario tiene la posibilidad de definir sus propias operaciones

Comunicaciones Colectivas

Las operaciones colectivas de MPI pueden ser clasificadas en tres clases:

- Sincronización: Barreras para sincronizar.
- Movimiento: (transferencia) de datos. Operaciones para difundir, recolectar y esparcir.
- Cálculos colectivos: Operaciones para reducción global, tales como suma, máximo, mínimo o cualquier función definida por el usuario.

Comunicaciones Colectivas

MPI_Barrier(MPI_Comm comm)

MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Tipos de Datos y Operaciones del Usuario

Las implantaciones de MPI permiten definir tipos de datos por parte del usuario.

Esto es útil a la hora de trabajar con tipos de datos compuestos como estructuras o registros

Tipos de Datos y Operaciones del Usuario

Asi mismo, las implantaciones de MPI permiten crear nuevas operaciones para las comunicaciones colectivas.

Comunicaciones Colectivas

Las rutinas de comunicación colectiva proporcionan un alto nivel para organizar el programa paralelo.

Cada proceso ejecuta las mismas operaciones de comunicación.

MPI proporciona un conjunto importante de funciones para la comunicación colectiva.

Eliminación Gaussiana

Es uno de los método más efectivos para resolver sistemas de ecuaciones del tipo

$$Ax = b$$

Donde A es la matriz de coeficientes y b es el vector de términos independientes

Eliminación Gaussiana

Consideremos el siguiente sistema de ecuaciones:

$$x_1 + 1/2x_2 + 1/3x_3 = 3$$

$$1/2x_1 + 1/3x_2 + 1/4x_3 = 2$$

$$1/3x_1 + 1/4x_2 + 1/5x_3 = 1$$

Eliminación Gaussiana

Si seleccionamos el coeficiente a_{11} como pivote

$$x_1 + 1/2x_2 + 1/3x_3 = 3$$

$$1/2x_1 + 1/3x_2 + 1/4x_3 = 2$$

$$1/3x_1 + 1/4x_2 + 1/5x_3 = 1$$

Eliminación Gaussiana

Dividimos los coeficientes de la misma columna por a_{11}

$$l_{21} = a_{21}/a_{11} = (1/2)/1 = 1/2$$

$$l_{31} = a_{31}/a_{11} = 1/3$$

Luego, multiplicamos la primera fila por l_{21} y se la restamos a la segunda fila.
Despues, multiplicamos la primera fila por l_{31} y se la restamos a la tercera fila

Eliminación Gaussiana

El sistema queda de la siguiente forma:

$$x_1 + 1/2x_2 + 1/3x_3 = 3$$

$$1/12x_2 + 1/12x_3 = 1/2$$

$$1/12x_2 + 4/45x_3 = 0$$

Eliminación Gaussiana

Ahora tomamos como pivote a_{22}

$$x_1 + 1/2x_2 + 1/3x_3 = 3$$

$$1/12x_2 + 1/12x_3 = 1/2$$

$$1/12x_2 + 4/45x_3 = 0$$

Eliminación Gaussiana

Dividimos los coeficientes de la misma columna por a_{22}

$$l_{32} = a_{32}/a_{22}$$

Luego, multiplicamos la segunda fila por l_{32} y se la restamos a la tercera fila.

Eliminación Gaussiana

El sistema de la siguiente forma

$$x_1 + 1/2x_2 + 1/3x_3 = 3$$

$$1/12x_2 + 1/12x_3 = 1/2$$

$$1/1805x_3 = -1/2$$