# Introducción Breve a MPI

Colaboratorio Nacional de Computación Avanzada (CNCA)


2014

# Contenidos

# Definición

- MPI = Message Passing Interface
- Es una *especificación*, no una biblioteca.
- Utilizado en el modelo de PASO DE MENSAJES.
- Aplicable en C, C++, Fortran y otros lenguajes.
- Paralelismo explícito.
- Modelo de memoria distribuida.

# Justificación

- Estandarización: soportado por todas las plataformas HPC.
- Portabilidad: código fuente no cambia.
- Rendimiento: optimizaciones de proveedores (eg. Intel).
- Accesibilidad: muchas implementaciones disponibles.

# Paso de Mensajes

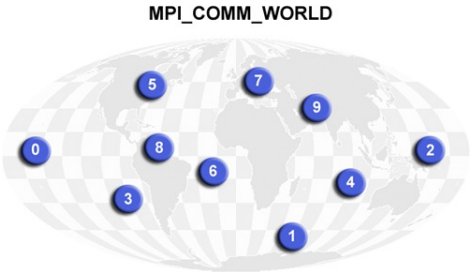
- Envío de datos, sincronización y cálculo a traves de paquetes.
- Paralelismo explícito.
- Datos viajan por un medio físico: buses internos, red, etc.
- Es importante el manejo adecuado de la memoria para preservar la integridad de los datos.
- Por lo general se utiliza con paralelismo de *grano grueso*, ya que la comunicación es costosa.

# Estructura general

1. MPI "Include"
2. Declaración de variables y prototipos de funciones.
3. Inicializar ambiente MPI.
4. Código paralelo y paso de mensajes.
5. Finalizar ambiente MPI.
6. Fin del programa.

# Estructura general

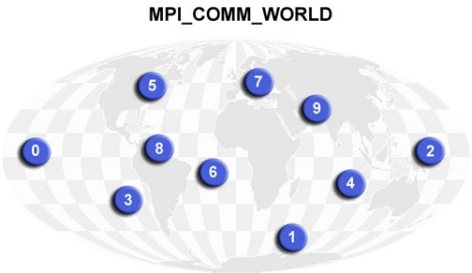
1. Variables y funciones comienzan con MPI_.
2. Procesos se agrupan en *communicators*.




3. En un comunicador cada proceso tiene un número de identificación, llamado usualmente *rank*.
4. Este *id* puede usarse para especificar origen y destino de los mensajes.

# Estructura general

1. Variables y funciones comienzan con MPI_.
2. Procesos se agrupan en *communicators*.




3. En un comunicador cada proceso tiene un número de identificación, llamado usualmente *rank*.
4. Este *id* puede usarse para especificar origen y destino de los mensajes.

# Manejo del ambiente

- MPI_Init(&argc,&argv): Inicializa el ambiente de MPI. Se llama sólo UNA vez.
- MPI_Comm_size(comm,&size): Calcula la cantidad de procesos en un comunicador, y lo guarda en *size*.
- MPI_Comm_rank(comm,&rank): Retorna el *id* del proceso actual en la variable *rank*.
- MPI_Get_processor_name(&name,&resultlength): Retorna el nombre del procesador (máquina) asignada al proceso actual (*name*) y su largo (*resultlength*).
- MPI_Wtime(): Retorna la hora del sistema (double).
- MPI_Finalize(): Termina el ambiente MPI. Es la última función MPI del programa.

# Manejo del ambiente

- MPI_Init(&argc,&argv): Inicializa el ambiente de MPI. Se llama sólo UNA vez.
- MPI_Comm_size(comm,&size): Calcula la cantidad de procesos en un comunicador, y lo guarda en *size*.
- MPI_Comm_rank(comm,&rank): Retorna el *id* del proceso actual en la variable *rank*.
- MPI_Get_processor_name(&name,&resultlength): Retorna el nombre del procesador (máquina) asignada al proceso actual (*name*) y su largo (*resultlength*).
- MPI_Wtime(): Retorna la hora del sistema (double).
- MPI_Finalize(): Termina el ambiente MPI. Es la última función MPI del programa.

# Ejercicio

Copie el archivo 'Hola.c' en su directorio home
Compile con:

```
$ mpicc Hola.c -o Hola
```

Ejecute con:

```
$ mpiexec -np [np] ./Hola
```

donde np es el número de procesos.

# Tipos de datos

Para facilitar la portabilidad MPI define tipos de datos propios:

- MPI_CHAR
- MPI_SHORT, MPI_INT, MPI_LONG
- MPI_DOUBLE, MPI_FLOAT
- Muchos más...

La implementación de los tipos depende de la implementación de MPI.

Se utilizan para paso de mensajes, pero corresponden a los tipos equivalentes en C/C++. i.e.: MPI_FLOAT = float, MPI_INT = int, etc.

# Comunicación Punto a Punto

- Comunicación directa entre dos procesos.
- Con/sin bloqueo, envío sincronizado o con buffer, etc.

Enviar:

| | |
|---|---|
| Bloqueo | MPI_Send(&buffer,count,type,dest,tag,comm) |
| No Bloqueo | MPI_Isend(&buffer,count,type,dest,tag,comm,request) |

Recibir:

| | |
|---|---|
| Bloqueo | MPI_Recv(&buffer,count,type,source,tag,comm,status) |
| No Bloqueo | MPI_Irecv(&buffer,count,type,source,tag,comm,request) |

# Comunicación Punto a Punto

- Comunicación directa entre dos procesos.
- Con/sin bloqueo, envío sincronizado o con buffer, etc.

Enviar:

| | |
|---|---|
| Bloqueo | MPI_Send(&buffer,count,type,dest,tag,comm) |
| No Bloqueo | MPI_Isend(&buffer,count,type,dest,tag,comm,request) |

Recibir:

| | |
|---|---|
| Bloqueo | MPI_Recv(&buffer,count,type,source,tag,comm,status) |
| No Bloqueo | MPI_Irecv(&buffer,count,type,source,tag,comm,request) |

# Comunicación colectiva

- Involucra *todos* los procesos en un comunicador.
- Aplicaciones: sincronización, envío de datos, cálculos colectivos.

Sincronización:

- MPI_Barrier(comm): Bloquea un proceso hasta que TODOS los otros alcancen este punto.

# Comunicación colectiva

- Involucra *todos* los procesos en un comunicador.
- Aplicaciones: sincronización, envío de datos, cálculos colectivos.

Sincronización:

- MPI_Barrier(comm): Bloquea un proceso hasta que TODOS los otros alcancen este punto.

Envío:

- MPI_Bcast: Envía el mensaje en buffer desde root a cada proceso en comm.
- MPI_Scatter: Distribuye datos equitativamente entre procesos.
- MPI_Gather: Reúne datos de varios procesos en uno.

# Comunicación colectiva

- Involucra *todos* los procesos en un comunicador.
- Aplicaciones: sincronización, envío de datos, cálculos colectivos.

Sincronización:

- MPI_Barrier(comm): Bloquea un proceso hasta que TODOS los otros alcancen este punto.

Envío:

- MPI_Bcast: Envía el mensaje en buffer desde root a cada proceso en comm.
- MPI_Scatter: Distribuye datos equitativamente entre procesos.
- MPI_Gather: Reúne datos de varios procesos en uno.

# Comunicación colectiva – Envío

Sintaxis:

- MPI_Bcast(&buffer,count,datatype,root,comm)
- MPI_Scatter(&sendbuf,sendcnt,sendtype,&recvbuf,recvcnt,recvtype,root,comm)
- MPI_Gather(&sendbuf,sendcnt,sendtype,&recvbuf,recvcnt,recvtype,root,comm)

Importante recordar que:

1. Los campos como 'count' se refieren a la *cardinalidad* o cantidad de elementos, y no a la cantidad de variables.
2. 'sendcnt' y 'recvcnt' corresponden al mismo número.

# Comunicación colectiva – Cálculo

Cálculo:

- MPI_Reduce: "Reduce" los datos de *n* procesos y deja el resultado en uno o más procesos.
- Sintaxis:
  MPI_Reduce(&sendbuf,&recvbuf,count,datatype,op,root,comm):
- Algunas operaciones posibles:
  - MPI_MAX
  - MPI_MIN
  - MPI_SUM
  - MPI_PROD
  - MPI_LAND

# Comunicación colectiva

Esquema: