MPI 2

Introducción a Message-Passing Interface 2

Manuel Martín Salvador
draxus@gmail.com
http://draxus.org

Licencia CC

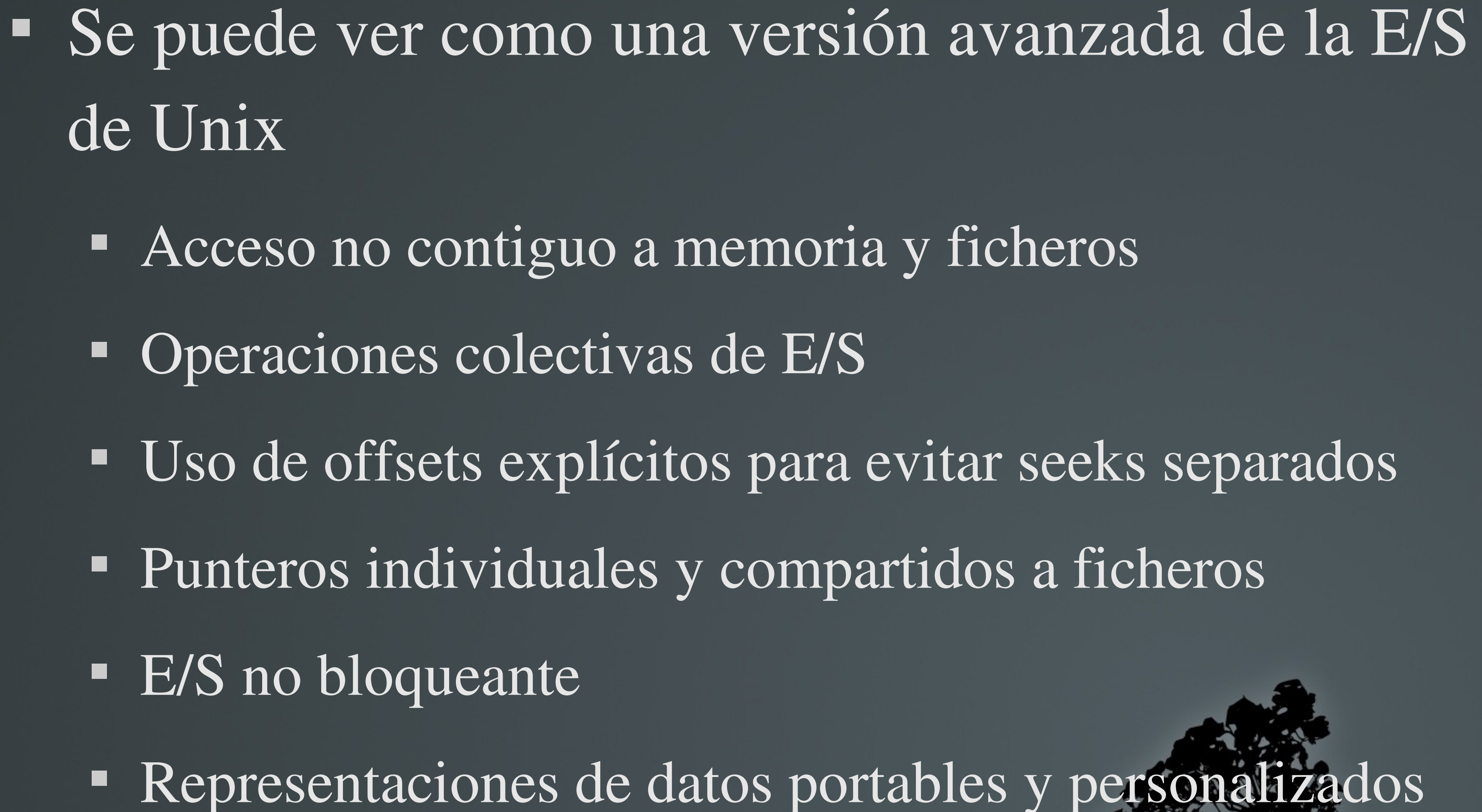
Índice

- Introducción
- E/S Paralela
- Acceso remoto a memoria
- Gestión dinámica de procesos
- Entorno de ejecución
- Referencias

Introducción

- MPI-1 se creó para agrupar las diferentes liberías de paso de mensajes bajo una misma sintaxis
- MPI-2 añade nuevas funcionalidades requeridas por los programadores
- Tres cambios principales
  - Entrada/Salida paralela
  - Operaciones con acceso remoto a memoria
  - Gestión dinámica de procesos

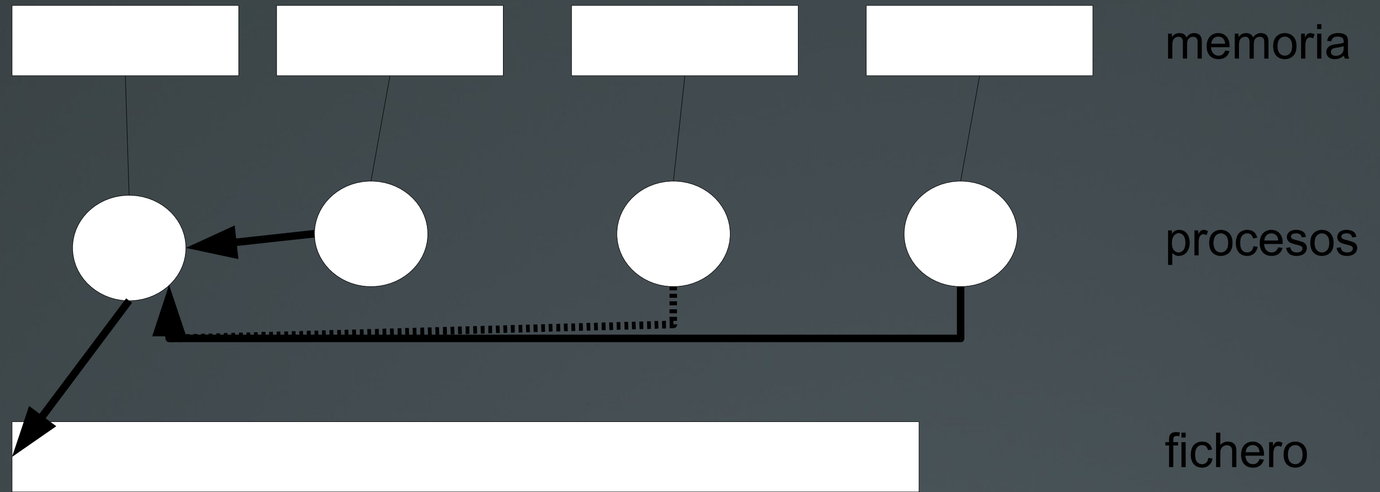
E/S Paralela



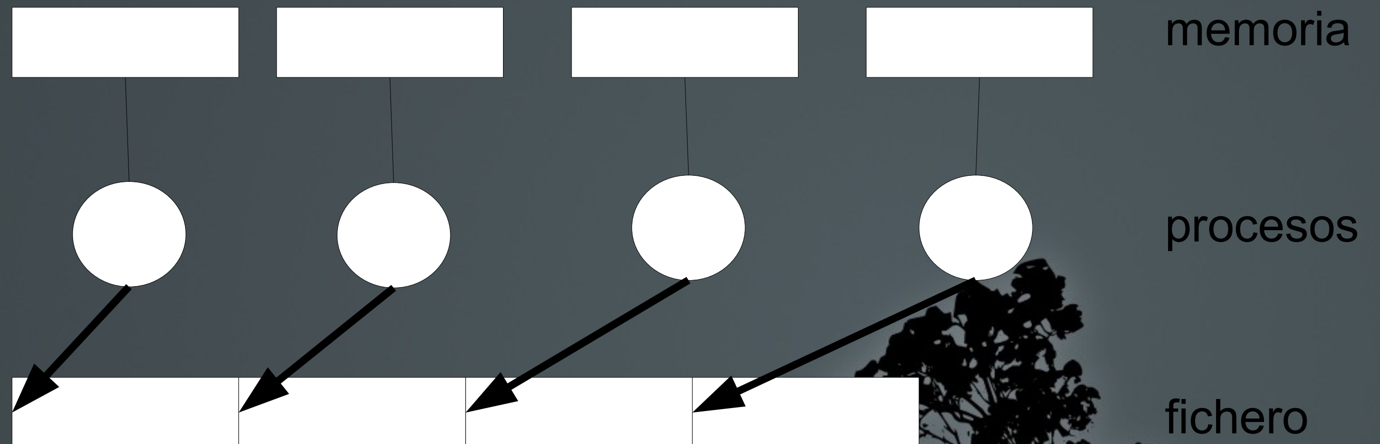
- Se puede ver como una versión avanzada de la E/S de Unix
  - Acceso no contiguo a memoria y ficheros
  - Operaciones colectivas de E/S
  - Uso de offsets explícitos para evitar seeks separados
  - Punteros individuales y compartidos a ficheros
  - E/S no bloqueante
  - Representaciones de datos portables y personalizados

E/S Paralela

SECUENCIAL



PARALELO



E/S Paralela

SECUENCIAL

```
int i, myrank, numprocs, buf[BUFSIZE];
MPI_Status status;
FILE *myfile;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
for(i=0; i<BUFSIZE; i++)
    buf[i] = myrank*BUFSIZE + i;
if(myrank!=0)
    MPI_Send(buf, BUFSIZE, MPI_INT, 0, 99, MPI_COMM_WORLD);
else{
    myfile = fopen(”testfile”, ”w”);
    fwrite(buf, sizeof(int), BUFSIZE, myfile);
    for(i=1; i<numprocs; i++){
        MPI_Recv(buf, BUFSIZE, MPI_INT, i, 99, MPI_COMM_WORLD, &status);
        fwrite(buf, sizeof(int), BUFSIZE, myfile);
}
MPI_Finalize();
```

E/S Paralela

PARALELO

```
int i, myrank, numprocs, buf[BUFSIZE];
MPI_File thefile;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
for(i=0; i<BUFSIZE; i++)
    buf[i] = myrank*BUFSIZE + i;
MPI_File_open(MPI_COMM_WORLD, ”testfile”,
             MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &thefile);
MPI_File_set_view(thefile, myrank*BUFSIZE*sizeof(int),
                 MPI_INT, MPI_INT, ”native”, MPI_INFO_NULL);
MPI_File_write(thefile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
MPI_File_close(&thefile);
MPI_Finalize();
```

E/S Paralela

Funciones en C

Abrir un fichero
int **MPI_File_open**(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh)

Ajustar la parte del fichero que ve cada proceso
int **MPI_File_set_view**(MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, char *datarep, MPI_Info info)

Escribir en el fichero
int **MPI_File_write**(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

Cerrar un fichero
int **MPI_File_close**(MPI_File *fh)

Obtener el tamaño de un fichero
int **MPI_File_get_size**(MPI_File fh, MPI_Offset *size)

Leer un fichero
int **MPI_File_read**(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

E/S Paralela

Funciones en C (continuación)

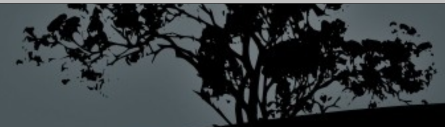
Posicionamiento en el fichero
int **MPI_File_seek**(MPI_file fh, MPI_Offset offset, int whence)

Posicionamiento explítico en lectura
int **MPI_File_read_at**(MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

Posicionamiento explícito en escritura
int **MPI_File_write_at**(MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

Lectura colectiva de un fichero
int **MPI_File_read_all**(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

Escritura colectiva de un fichero
int **MPI_File_write_all**(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)



E/S Paralela

Funciones en C (continuación)

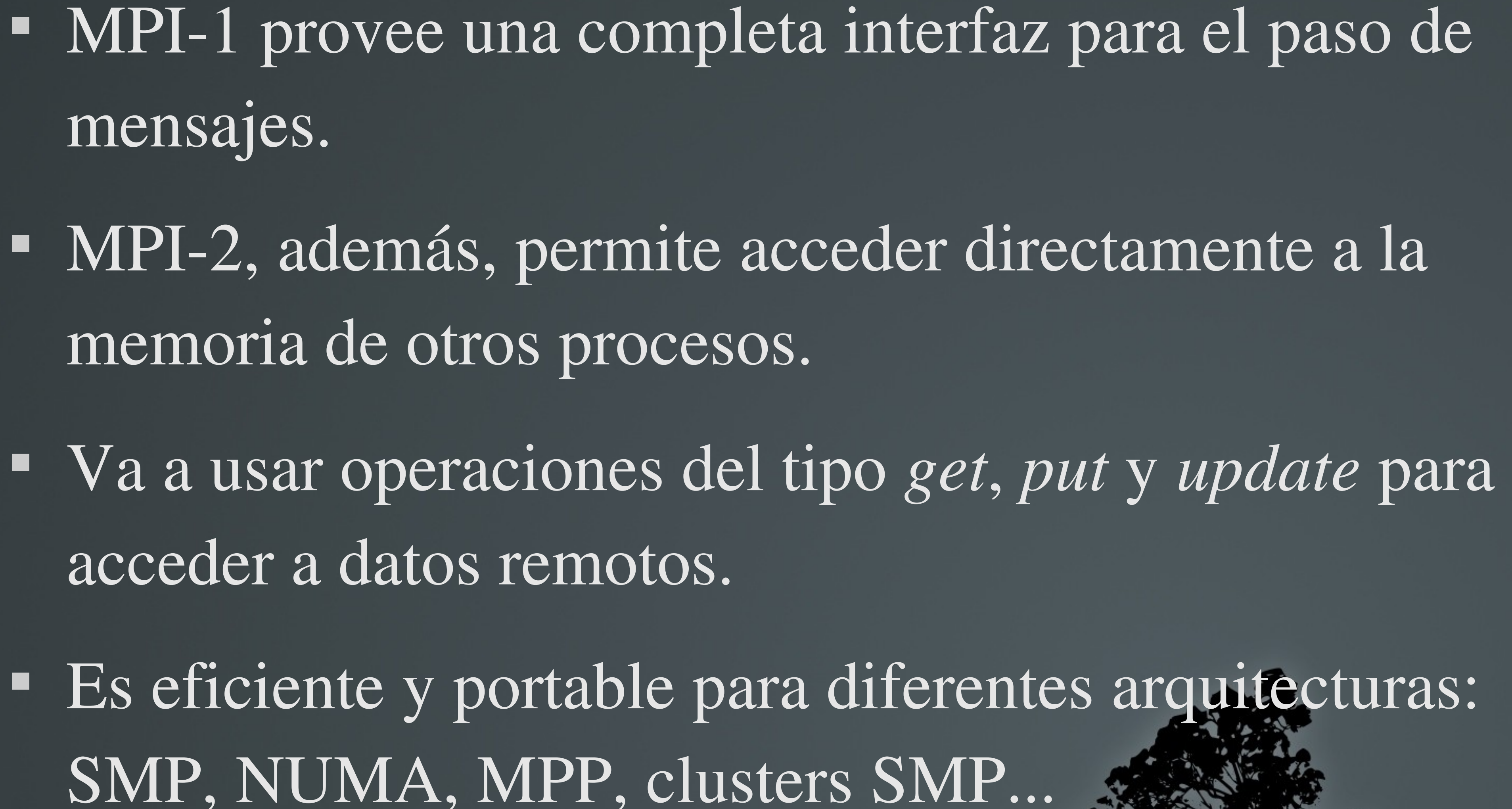
Escritura no bloqueante
int **MPI_File_iwrite_at**(MPI_file fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Request *request)

Escritura colectiva (inicio)
int **MPI_File_write_all_begin**(MPI_File fh, void *buf, int count, MPI_Datatype datatype)

Escritura colectiva (fin)
int **MPI_File_write_all_end**(MPI_File fh, void *buf, MPI_Status *status)

Escritura con puntero compartido
int **MPI_File_write_shared**(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

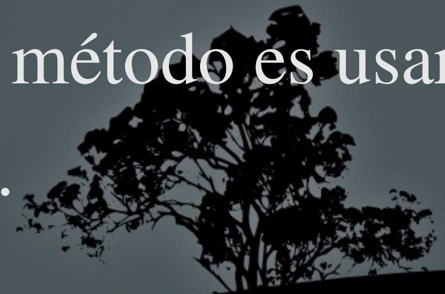
Acceso remoto a memoria



- MPI-1 provee una completa interfaz para el paso de mensajes.
- MPI-2, además, permite acceder directamente a la memoria de otros procesos.
- Va a usar operaciones del tipo *get*, *put* y *update* para acceder a datos remotos.
- Es eficiente y portable para diferentes arquitecturas: SMP, NUMA, MPP, clusters SMP...

Acceso remoto a memoria

- Tres pasos principales:
  - Definir en el proceso qué memoria va a usarse para operaciones RMA. Se crea un objeto **MPI_Win**.
  - Especificar qué datos se van a mover y a dónde. Se usan las operaciones **MPI_Put**, **MPI_Get** y **MPI_Accumulate**.
  - Verificar que los datos se han recibido por completo. Se usa la función **MPI_Win_fence**. Otro método es usar **MPI_Win_lock** y **MPI_Win_unlock**.



Acceso remoto a memoria

- Memory Windows (ventanas de memoria)
  - Son los espacios de memoria reservados para ser usados en las operaciones de acceso remoto a memoria.

Crea una ventana.
int **MPI_Win_create**(void *base, MPI_Aint size, int disp_unit, MPI_Info info,
MPI_Comm comm, MPI_Win *win)

Liberar una ventana. Sólo debe llamarse cuando han terminado todas las operaciones RMA.
int **MPI_Win_free**(MPI_Win *win)

Ambas son operaciones colectivas, deben ser llamadas por todos los procesos involucrados.

Acceso remoto a memoria

- Moviendo datos

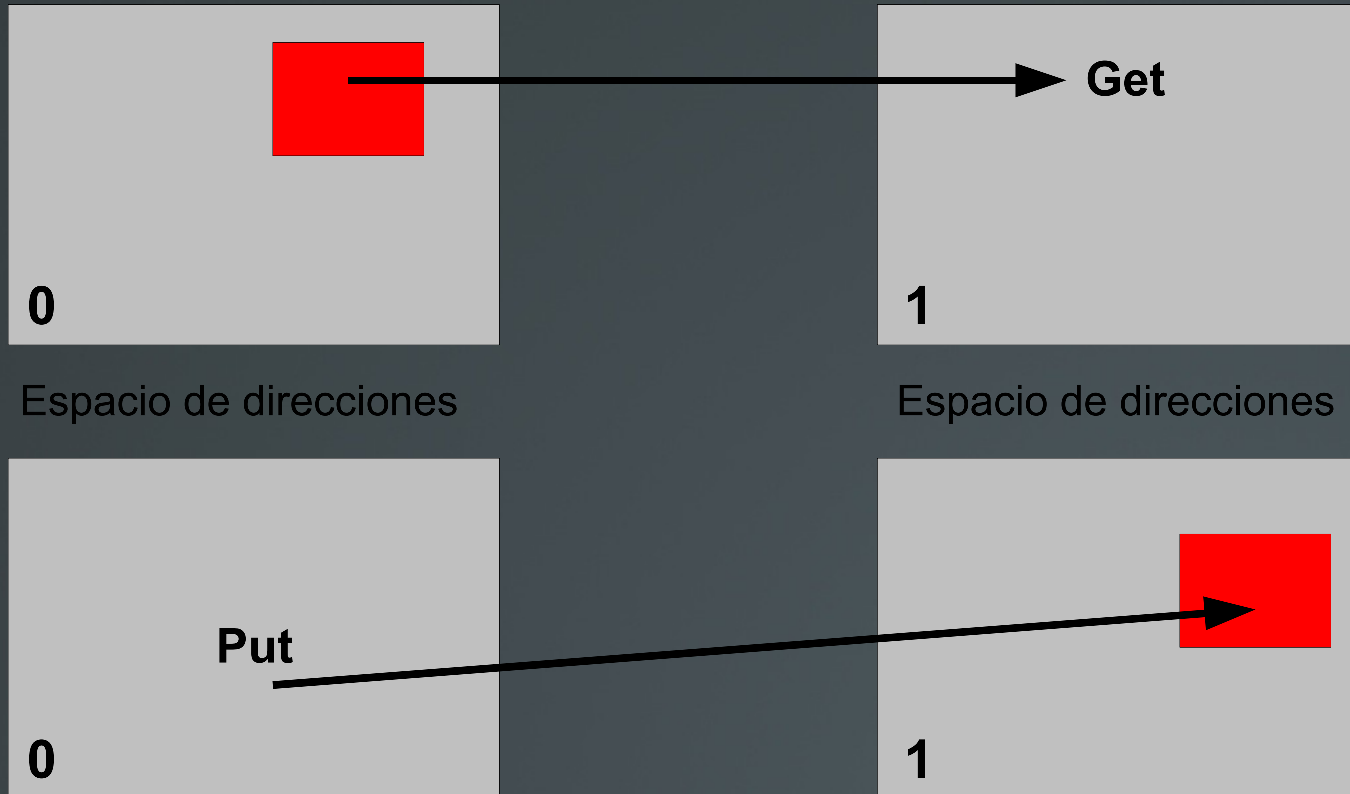
Pone datos en una ventana de memoria (no bloqueante)
int **MPI_Put**(void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_blank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win)

Obtiene datos de una ventana de memoria (no bloqueante)
int **MPI_Get**(void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win)

Actualiza los datos de una ventana de memoria
int **MPI_Accumulate**(void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)

Completa una transferencia de datos
int **MPI_Win_fence**(int assert, MPI_Win win)

Acceso remoto a memoria



local

ventana RMA

Acceso remoto a memoria

Ejemplo usando paso de mensajes

```
if(rank==0){
    MPI_Isend(outbuf, n, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
}
else if(rank==1){
    MPI_Irecv(inbuf, n, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
}

/* ...Otras operaciones... */

MPI_Wait(&request, &status);
```

Ejemplo usando RMA

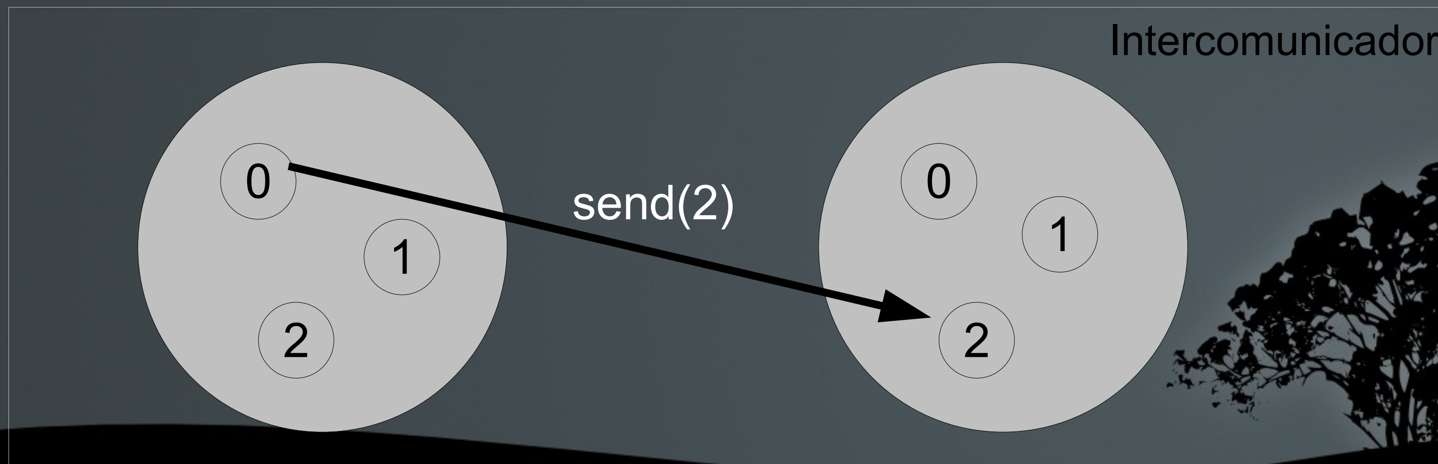
```
if(rank==0){
    MPI_Win_create(NULL, 0, sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD, &win);
}
else if(rank==1){
    MPI_Win_create(inbuf, n*sizeof(int), sizeof(int), MPI_INFO_NULL,
                   MPI_COMM_WORLD, &win);
}
MPI_Win_fence(0, win); //"obligatorio" antes de un MPI_Put
if(rank==0) MPI_Put(outbuf, n, MPI_INT, 1, 0, n, MPI_INT, win);
//Otra forma sería usar MPI_Get en el proceso 1

/* ...Otras operaciones... */
MPI_Win_fence(0, win);
MPI_Win_free(&win);
```

Gestión dinámica de procesos

- En MPI-1 el número de procesos de COMM_WORLD es fijo durante toda la ejecución
- En MPI-2 podemos crear procesos usando la opración colectiva MPI_Comm_spawn
- Esta función devuelve un intercomunicador que desde el punto de vista de los ”padres”, agrupa a los padres, mientras que para los ”hijos” sería su propio COMM_WORLD.

Gestión dinámica de procesos

- Intracomunicador: conjunto de procesos (MPI-1)
- Intercomunicador: engloba 2 conjuntos de procesos (local y remoto). La idea existe en MPI-1, pero la especificación de operaciones colectivas en intercomunicadores aparece en MPI-2.



Gestión dinámica de procesos

Creación de procesos hijos
int **MPI_Comm_spawn**(char *command, char *argv[], int maxprocs, MPI_Info info, int root, MPI_Comm comm, MPI_Comm *intercomm, int array_of_errcodes[])

Debe ser ejecutada por todos los procesos de comm

Obtiene el comunicador de los padres
int **MPI_Comm_get_parent**(MPI_Comm *parent)

Sólo ejecutado por los hijos

Creación de múltiples procesos hijos
int **MPI_Comm_spawn_multiple**(int count, char *array_of_commands[], char **array_of_argv[], int array_of_maxprocs[], MPI_Info array_of_info[], int root, MPI_Comm comm, MPI_Comm *intercomm, int array_of_errcodes[])

Gestión dinámica de procesos

- Filosofía cliente/servidor

Abrir puerto
int **MPI_Open_port**(MPI_Info info, char *port_name)

Cerrar puerto
int **MPI_Close_port**(char *port_name)

Aceptar conexión
int **MPI_Comm_accept**(char *port_name, MPI_Info info, int root, MPI_Comm comm, MPI_Comm *newcomm)

Conectar a un puerto
int **MPI_Comm_connect**(char *port_name, MPI_Info info, int root, MPI_Comm comm, MPI_Comm *newcomm)

Desconectar
int **MPI_Comm_disconnect**(MPI_Comm *comm)

Gestión dinámica de procesos

Servidor

```
MPI_Comm client;

MPI_Open_port(MPI_INFO_NULL
, port_name);

MPI_Comm_accept(port_name,
MPI_INFO_NULL, 0,
MPI_COMM_WORLD, &client);

MPI_Send(......., client);

MPI_Close_port(port_name);
```

Cliente

```
MPI_Comm server;

MPI_Comm_connect(port_name,
MPI_INFO_NULL, 0,
MPI_COMM_WORLD, &server);

MPI_Comm_remote_size(server,
&nprocs);

MPI_Recv(...., server);

MPI_Comm_disconnect(&server);
```

Entorno de ejecución

- He usado LAM/MPI, que soporta MPI-2
- Preparamos las máquinas:
  - Instalamos el servidor lam y los ejecutables y librerías de mpich.
  - Configuramos ssh y generamos claves rsa en todos los ordenadores. Crearemos el fichero `~/.ssh/authorized_keys2`, que contendrá las claves de todos los ordenadores que vamos a usar. Es importante que se creen sin contraseña.
  - En el ordenador raíz, creamos un archivo ”maquinas” de forma que incluya las direcciones IP de los ordenadores que formarán parte del procesamiento. Ahora ejecutamos el servidor lam pasando como parámetro dicho archivo: `lamboot maquinas`. Comprobaremos que el demonio `lamd` se ha iniciado en todas las máquinas afectadas.
  - Hay que tener en cuenta que el proceso a ejecutar debe estar en la misma ruta en todas las máquinas, y compilado para su arquitectura.
  - Para finalizar, lanzamos `mpirun -np X programa`, siendo X el número de procesadores.

Referencias

- William Gropp, Ewing Lusk, Rajeev Thakur: *Using MPI-2: Advanced Features of the Message Passing Interface* (1999)
- LAM/MPI http://www.lam-mpi.org/
- OpenMPI http://www.open-mpi.org/
- MPICH2 http://www.mcs.anl.gov/research/projects/mpich2/

Fin

GRACIAS