Evolución y nuevas funcionalidades de MPI

MPI el estándar que define la sintaxis y la semántica de las funciones.

Open MPI es una implementación libre de dicho estándar que nace de la fusión de:

- FT-MPI de la Universidad de Tennessee
- LA-MPI del Laboratorio Nacional de Los Álamos
- LAM/MPI de la Universidad de Indiana

Un poco de historia

MPI-1.0: Lanzado en 1993, sentó las bases aunque carecía de operaciones de comunicación colectiva y sus hebras no eran seguras.

MPI-1.1 y MPI-1.2 : Lanzado en 1995, nuevas funciones que no cambian significativamente la funcionalidad.

MPI-2.0: Lanzado en 1997. Añade funciones completamente nuevas y bindings para C++.

Un poco de historia

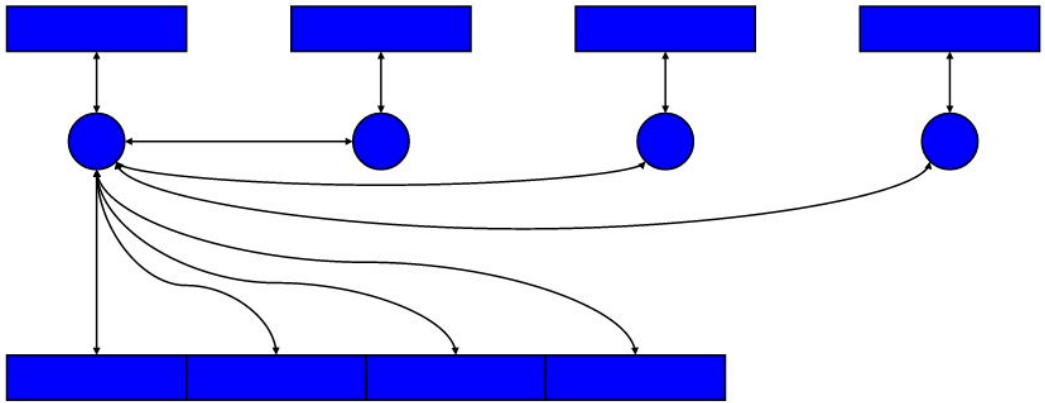
MPI-1.3 y MPI-2.1: Lanzados en 2008, MPI-1.3 es la última revisión de MPI-1. MPI-2.1. Corrige algunos errores de MPI-2.

MPI-2.2: Lanzado en 2009, busca corregir ambigüedades introducidas en el estándar MPI-2.1.

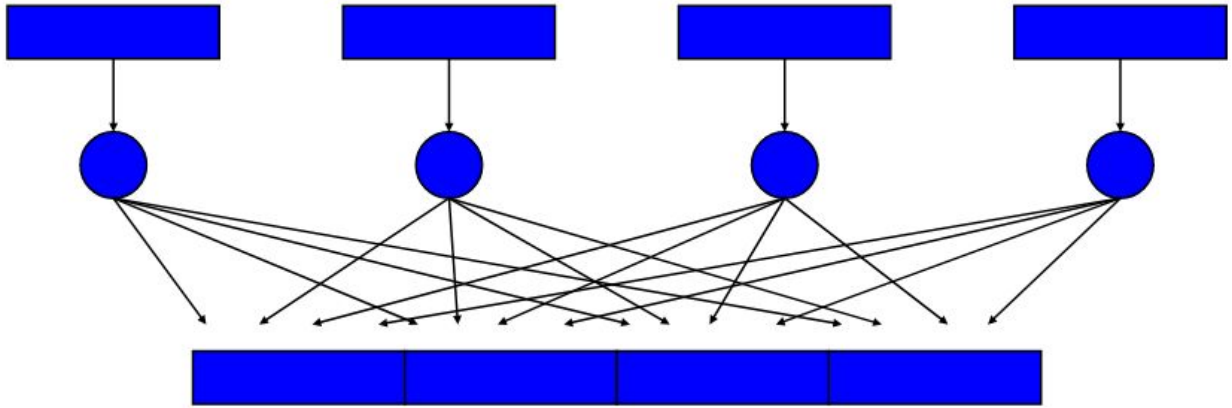
MPI-3.0: Lanzado en 2012, extiende la funcionalidad de varias funciones. Elimina funciones y objetos obsoletos así como los bindings de C++.

MPI-3.1: Lanzado en 2015. Aclaraciones sobre el estándar anterior.

Nuevas funcionalidades: E/S Paralela



Nuevas funcionalidades: E/S Paralela



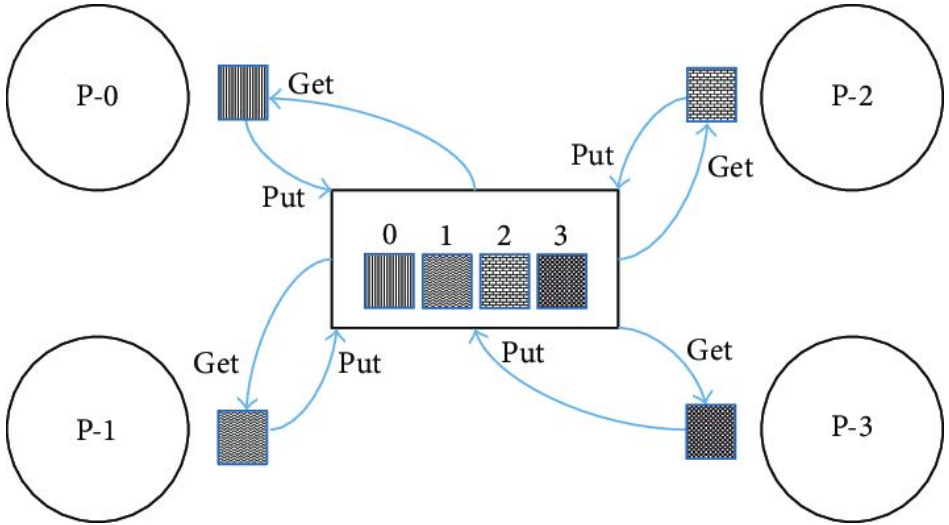
Nuevas funcionalidades: E/S Paralela

```
int MPI_File_open(MPI_Comm comm, const char *filename, int amode, MPI_Info
info, MPI_File *fh)

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

int MPI_File_write(MPI_File fh, const void *buf, int count, MPI_Datatype
datatype, MPI_Status *status)

int MPI_File_close(MPI_File *fh)
```

Nuevas funcionalidades: E/S Paralela

```c
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[]) {
 MPI_File fh;
 int buf[1000], rank;
 MPI_Init(0, 0);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_File_open(MPI_COMM_WORLD, "test.out", MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
 if (rank == 0)
   MPI_File_write(fh, buf, 1000, MPI_INT, MPI_STATUS_IGNORE);
 MPI_File_close(&fh);
 MPI_Finalize();
 return 0;
}
```

Nuevas funcionalidades: Comunicación unilateral



Nuevas funcionalidades: Comunicación unilateral

```
int MPI_Info_create(MPI_Info *info)

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

int MPI_Win_fence(int assert, MPI_Win win)

MPI_Put(const 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)

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)
```

Nuevas funcionalidades: Comunicación unilateral

```
#include <stdio.h>
#include <mpi.h>

#define NUM_ELEMENT 4

int main(int argc, char **argv){
 int i, id, num_procs, len,
localbuffer[NUM_ELEMENT],
sharedbuffer[NUM_ELEMENT];

 char name[MPI_MAX_PROCESSOR_NAME];
 MPI_Win win;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &id);
 MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
 printf("Rank %d\n", id);
```

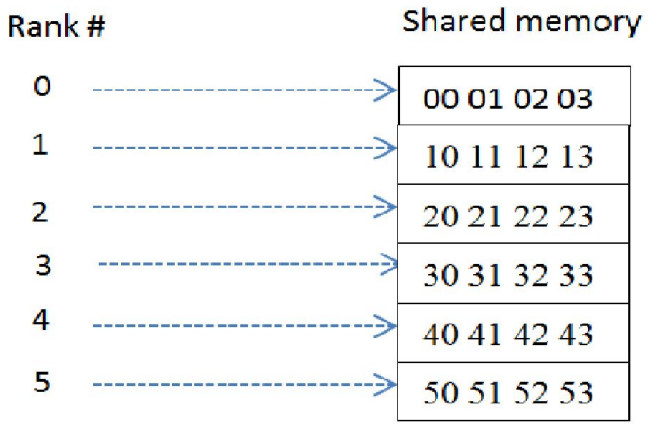
```
 MPI_Win_create(sharedbuffer, NUM_ELEMENT,
sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD, &win);

 for (i = 0; i < NUM_ELEMENT; i++){
   sharedbuffer[i] = 10 * id + i;
   localbuffer[i] = 0;
 }

 printf("Rank %d pone datos en memoria compartida:",id);
 for (i = 0; i < NUM_ELEMENT; i++)
   printf(" %02d", sharedbuffer[i]);
 printf("\n");
 MPI_Win_fence(0, win);
 if (id != 0)
   MPI_Get(&localbuffer[0], NUM_ELEMENT, MPI_INT, id -
1, 0, NUM_ELEMENT, MPI_INT, win);
```

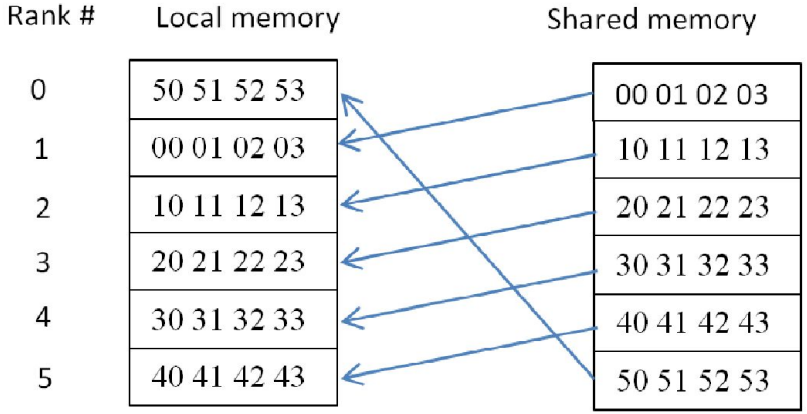
Nuevas funcionalidades: Comunicación unilateral

```
 else
   MPI_Get(&localbuffer[0], NUM_ELEMENT,
MPI_INT, num_procs - 1, 0, NUM_ELEMENT,
MPI_INT, win);

 MPI_Win_fence(0, win);
 printf("Rank %d recoge datos de memoria
compartida:", id);
 for (i = 0; i < NUM_ELEMENT; i++)
   printf(" %02d", localbuffer[i]);

 printf("\n");

 MPI_Win_fence(0, win);
```

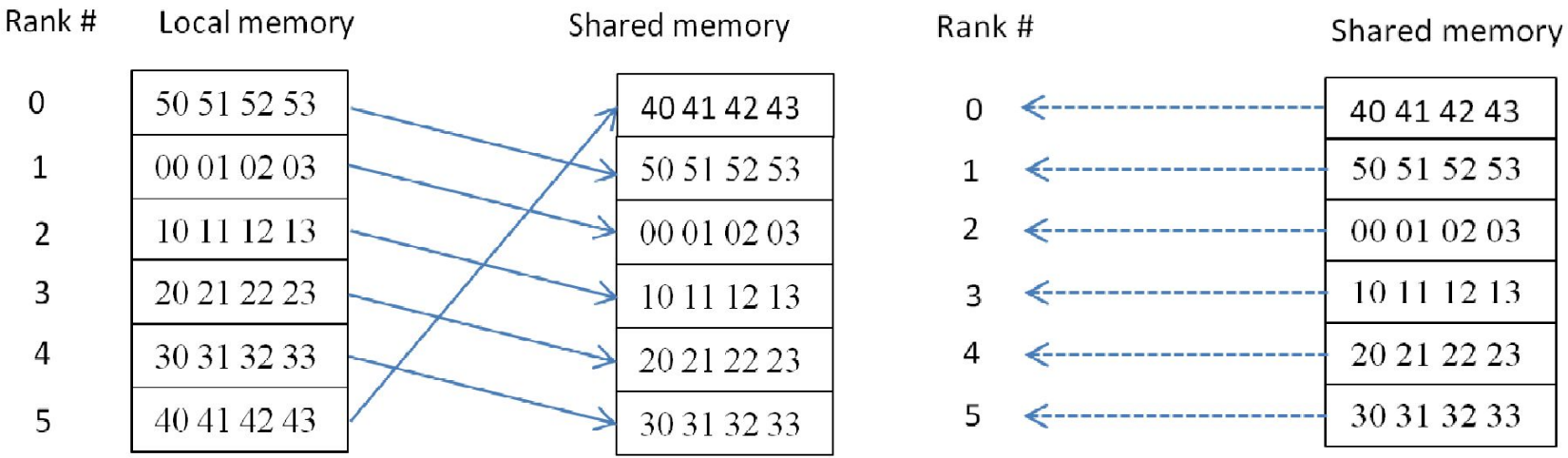
```
  if (id < num_procs - 1)
   MPI_Put(&localbuffer[0], NUM_ELEMENT, MPI_INT, id +
1, 0, NUM_ELEMENT, MPI_INT, win);
 else
   MPI_Put(&localbuffer[0], NUM_ELEMENT, MPI_INT, 0,
0, NUM_ELEMENT, MPI_INT, win);

 MPI_Win_fence(0, win);
 printf("Rank %d tiene nuevos datos en la memoria
compartida:", id);
 for (i = 0; i < NUM_ELEMENT; i++)
   printf(" %02d", sharedbuffer[i]);

 printf("\n");
 MPI_Win_free(&win);
 MPI_Finalize();
}
```

Nuevas funcionalidades: Comunicación unilateral

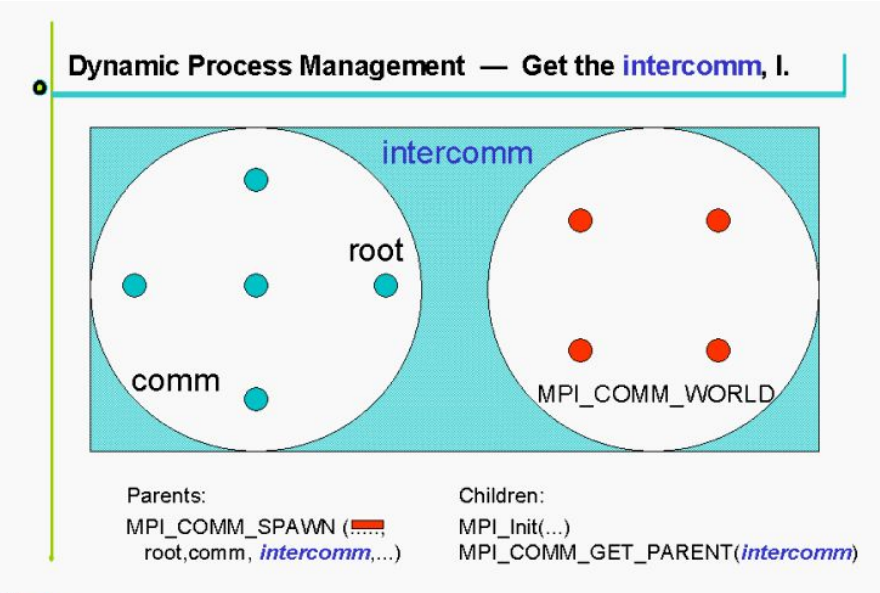




Nuevas funcionalidades: Comunicación unilateral



Nuevas funcionalidades: Administración dinámica de procesos



Nuevas funcionalidades: Administración dinámica de procesos

```
int MPI_Comm_spawn(const char *command, char *argv[], int maxprocs, MPI_Info info, int root,
MPI_Comm comm, MPI_Comm *intercomm, int array_of_errcodes[])

int MPI_Open_port(MPI_Info info, char *port_name)

int MPI_Comm_connect(const char *port_name, MPI_Info info, int root, MPI_Comm comm, MPI_Comm
*newcomm)

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

int MPI_Comm_join(int fd, MPI_Comm *intercomm)
```

Nuevas funcionalidades: Administración dinámica de procesos

```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

#define NUM_SPAWNS 2

int main(int argc, char *argv[]) {
 int np = NUM_SPAWNS;
 int errcodes[NUM_SPAWNS];
 MPI_Comm parentcomm, intercomm;
 MPI_Init(&argc, &argv);
 MPI_Comm_get_parent(&parentcomm);
 if (parentcomm == MPI_COMM_NULL) {
   MPI_Comm_spawn("ejecutable", MPI_ARGV_NULL, np,
MPI_INFO_NULL, 0,MPI_COMM_WORLD, &intercomm, errcodes);
   printf("Soy el padre.\n");
 } else {
   printf("Soy el hijo.\n");
 }
 fflush(stdout);
 MPI_Finalize();
 return 0;
}
```

FIN