



EBook Gratis

APRENDIZAJE

mpi

Free unaffiliated eBook created from
Stack Overflow contributors.

#mpi

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con mpi.....	2
Observaciones.....	2
Versiones.....	2
Examples.....	2
Rango y tamaño.....	2
Iniciar / Finalizar.....	3
Hola Mundo!.....	3
Valores de retorno de llamadas MPI.....	4
Capítulo 2: Colectivos.....	6
Observaciones.....	6
Examples.....	6
Emisión.....	6
Barrera.....	6
Dispersión.....	7
Capítulo 3: Compilando un programa de MPI.....	8
Observaciones.....	8
Examples.....	8
C Envoltorio.....	8
Capítulo 4: Creación y gestión de procesos.....	9
Examples.....	9
Desovar.....	9
Estableciendo conexión entre dos aplicaciones independientes.....	10
Capítulo 5: Ejecutando un programa MPI.....	12
Parámetros.....	12
Examples.....	12
Ejecuta tu trabajo.....	12
Capítulo 6: Implementaciones de MPI.....	13
Observaciones.....	13
Examples.....	13

Instalación de MPICH en Mac con Homebrew.....	13
Instalar Open MPI en Mac con Homebrew.....	13
Capítulo 7: Topologías de proceso.....	14
Examples.....	14
Creación y comunicación de topologías gráficas.....	14
Creditos.....	16

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [mpi](#)

It is an unofficial and free mpi ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official mpi.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con mpi

Observaciones

MPI es un estándar para la comunicación entre un grupo de procesos distribuidos (o locales). Incluye rutinas para enviar y recibir datos, comunicarse colectivamente y otras tareas más complejas.

El estándar proporciona una API para C y Fortran, pero también existen enlaces a otros lenguajes.

Versiones

Versión	Estándar	Fecha de lanzamiento
1	mpi-report-1.3-2008-05-30.pdf	1994-05-05
2.0	mpi2-report.pdf	2003-09-15
2.2	mpi22-report.pdf	2009-09-04
3.0	mpi30-report.pdf	2012-09-21
3.1	mpi31-report.pdf	2015-06-04

Examples

Rango y tamaño

Para obtener el tamaño de un comunicador (por ejemplo, `MPI_COMM_WORLD`) y el rango del proceso local dentro de él:

```
int rank, size;
int res;
MPI_Comm communicator = MPI_COMM_WORLD;

res = MPI_Comm_rank (communicator, &rank);
if (res != MPI_SUCCESS)
{
    fprintf (stderr, "MPI_Comm_rank failed\n");
    exit (0);
}
res = MPI_Comm_size (communicator, &size);
if (res != MPI_SUCCESS)
{
    fprintf (stderr, "MPI_Comm_size failed\n");
    exit (0);
}
```

Iniciar / Finalizar

Antes de ejecutar cualquier comando MPI, el entorno debe inicializarse y finalizarse al final:

```
int main(int argc, char** argv)
{
    int res;
    res = MPI_Init(&argc, &argv);
    if (res != MPI_SUCCESS)
    {
        fprintf(stderr, "MPI_Init failed\n");
        exit(0);
    }
    ...
    res = MPI_Finalize();
    if (res != MPI_SUCCESS)
    {
        fprintf(stderr, "MPI_Finalize failed\n");
        exit(0);
    }
}
```

Hola Mundo!

Tres cosas son generalmente importantes cuando se comienza a aprender a usar MPI. Primero, debe inicializar la biblioteca cuando esté listo para usarla (también debe finalizarla cuando haya terminado). Segundo, querrá saber el tamaño de su comunicador (lo que usa para enviar mensajes a otros procesos). Tercero, querrá saber su rango dentro de ese comunicador (en qué número de proceso está usted dentro de ese comunicador).

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

int main(int argc, char **argv) {
    int size, rank;
    int res;

    res = MPI_Init(&argc, &argv);
    if (res != MPI_SUCCESS)
    {
        fprintf(stderr, "MPI_Init failed\n");
        exit(0);
    }

    res = MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (res != MPI_SUCCESS)
    {
        fprintf(stderr, "MPI_Comm_size failed\n");
        exit(0);
    }
    res = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (res != MPI_SUCCESS)
    {
        fprintf(stderr, "MPI_Comm_rank failed\n");
        exit(0);
    }
}
```

```
fprintf(stdout, "Hello World from rank %d of %d~\n", rank, size);

res = MPI_Finalize();
if (res != MPI_SUCCESS)
{
    fprintf (stderr, "MPI_Finalize failed\n");
    exit (0);
}
}
```

Si ejecuta este programa de esta manera:

```
mpiexec -n 2 ./hello
```

Usted esperaría obtener una salida como esta:

```
Hello World from rank 0 of 2!
Hello World from rank 1 of 2!
```

También puede obtener esa salida hacia atrás (consulte <http://stackoverflow.com/a/17571699/491687>) para obtener más información sobre esto:

```
Hello World from rank 1 of 2!
Hello World from rank 0 of 2!
```

Valores de retorno de llamadas MPI

Casi cualquier llamada MPI devuelve un código de error entero, lo que significa el éxito de la operación. Si no se produce ningún error, el código de retorno es `MPI_SUCCESS` :

```
if (MPI_Some_op(...) != MPI_SUCCESS)
{
    // Process error
}
```

Si se produce un error, MPI llama a un controlador de errores asociado con el comunicador, la ventana o el objeto de archivo antes de regresar al código de usuario. Hay dos manejadores de errores predefinidos (el usuario puede definir manejadores de errores adicionales):

- `MPI_ERRORS_ARE_FATAL` : los errores resultan en la terminación del programa MPI
- `MPI_ERRORS_RETURN` : los errores provocan que el código de error se devuelva al usuario

El controlador de errores predeterminado para comunicadores y ventanas es `MPI_ERRORS_ARE_FATAL` ; para los objetos de archivo es `MPI_ERRORS_RETURN` . El controlador de errores para `MPI_COMM_WORLD` también se aplica a todas las operaciones que no están específicamente relacionadas con un objeto (por ejemplo, `MPI_Get_count`). Por lo tanto, verificar el valor de retorno de las operaciones sin E / S sin configurar el controlador de errores en `MPI_ERRORS_RETURN` es redundante ya que las llamadas MPI erróneas no se devolverán.

```
// The execution will not reach past the following line in case of error
int res = MPI_Comm_size(MPI_COMM_WORLD, &size);
if (res != MPI_SUCCESS)
{
    // The following code will never get executed
    fprintf(stderr, "MPI_Comm_size failed: %d\n", res);
    exit(EXIT_FAILURE);
}
```

Para habilitar el procesamiento de errores del usuario, primero se debe cambiar el controlador de errores de `MPI_COMM_WORLD` :

```
MPI_Comm_set_errhandler(MPI_COMM_WORLD, MPI_ERRORS_RETURN);

int res = MPI_Comm_size(MPI_COMM_WORLD, &size);
if (res != MPI_SUCCESS)
{
    fprintf(stderr, "MPI_Comm_size failed: %d\n", res);
    exit(EXIT_FAILURE);
}
```

El estándar MPI no requiere que las implementaciones de MPI puedan recuperarse de los errores y continuar la ejecución del programa.

Lea **Empezando con mpi en línea**: <https://riptutorial.com/es/mpi/topic/1943/empezando-con-mpi>

Capítulo 2: Colectivos

Observaciones

Las operaciones colectivas son llamadas MPI diseñadas para comunicar los procesos señalados por un comunicador en una sola operación o para realizar una sincronización entre ellos. Estos se utilizan a menudo para calcular uno o más valores basados en datos aportados por otros procesos o para distribuir o recopilar datos de todos los demás procesos.

Tenga en cuenta que todos los procesos en el comunicador deben invocar las mismas operaciones colectivas en orden, de lo contrario la aplicación se bloquearía.

Examples

Emisión

El siguiente código difunde el contenido en el `buffer` entre todos los procesos que pertenecen al comunicador `MPI_COMM_WORLD` (es decir, todos los procesos que se ejecutan en paralelo) utilizando la operación `MPI_Bcast`.

```
int rank;
int res;

res = MPI_Comm_rank (MPI_COMM_WORLD, &rank);
if (res != MPI_SUCCESS)
{
    fprintf (stderr, "MPI_Comm_rank failed\n");
    exit (0);
}

int buffer[100];
if (rank == 0)
{
    // Process with rank id 0 should fill the buffer structure
    // with the data it wants to share.
}

res = MPI_Bcast (buffer, 100, MPI_INT, 0, MPI_COMM_WORLD);
if (res != MPI_SUCCESS)
{
    fprintf (stderr, "MPI_Bcast failed\n");
    exit (0);
}
```

Barrera

La operación `MPI_Barrier` realiza una sincronización entre los procesos que pertenecen al comunicador dado. Es decir, todos los procesos de un comunicador determinado esperarán dentro del `MPI_Barrier` hasta que todos estén dentro y, en ese momento, dejarán la operación.

```

int res;

res = MPI_Barrier (MPI_COMM_WORLD); /* all processes will wait */
if (res != MPI_SUCCESS)
{
    fprintf (stderr, "MPI_Barrier failed\n");
    exit (0);
}

```

Dispersión

El proceso raíz `sendbuf` el contenido en `sendbuf` a todos los procesos (incluso a sí mismo) utilizando la operación `MPI_Scatter`.

```

int rank;
int size;
int sendcount = 1;
int recvcount = sendcount;
int sendbuf[3];
int recvbuf;
int root = 0;

MPI_Comm_size (MPI_COMM_WORLD, &size);

if (size != 3)
{
    fprintf (stderr, "Number of processes must be 3\n");
    exit (0);
}

MPI_Comm_rank (MPI_COMM_WORLD, &rank);

if (rank == 0)
{
    sendbuf[0] = 3;
    sendbuf[1] = 5;
    sendbuf[2] = 7;
}

MPI_Scatter (sendbuf, sendcount, MPI_INT, &recvbuf, recvcount, MPI_INT, root, MPI_COMM_WORLD);

printf ("rank: %i, value: %i\n", rank, recvbuf);

/* Output:
 * rank: 0, value: 3
 * rank: 1, value: 5
 * rank: 2, value: 7
 */

```

Lea Colectivos en línea: <https://riptutorial.com/es/mpi/topic/2602/colectivos>

Capítulo 3: Compilando un programa de MPI

Observaciones

MPI necesita agregar bibliotecas adicionales e incluir directorios a su línea de compilación al compilar su programa. En lugar de hacer un seguimiento de todos ellos, por lo general, puede usar uno de los envoltorios del compilador.

Examples

C Envoltorio

```
mpicc -o my_prog my_prog.c
```

Lea [Compilando un programa de MPI en línea](https://riptutorial.com/es/mpl/topic/3650/compilando-un-programa-de-mpl):

<https://riptutorial.com/es/mpl/topic/3650/compilando-un-programa-de-mpl>

Capítulo 4: Creación y gestión de procesos.

Examples

Desovar

El proceso maestro genera dos procesos de trabajo y dispersa el `sendbuf` a los trabajadores.

master.c

```
#include "mpi.h"

int main(int argc, char *argv[])
{
    int n_spawns = 2;
    MPI_Comm intercomm;

    MPI_Init(&argc, &argv);

    MPI_Comm_spawn("worker_program", MPI_ARGV_NULL, n_spawns, MPI_INFO_NULL, 0, MPI_COMM_SELF,
&intercomm, MPI_ERRCODES_IGNORE);

    int sendbuf[2] = {3, 5};
    int recvbuf; // redundant for master.

    MPI_Scatter(sendbuf, 1, MPI_INT, &recvbuf, 1, MPI_INT, MPI_ROOT, intercomm);

    MPI_Finalize();
    return 0;
}
```

trabajador.c

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

int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);

    MPI_Comm intercomm;
    MPI_Comm_get_parent(&intercomm);

    int sendbuf[2]; // redundant for worker.
    int recvbuf;

    MPI_Scatter(sendbuf, 1, MPI_INT, &recvbuf, 1, MPI_INT, 0, intercomm);
    printf("recvbuf = %d\n", recvbuf);

    MPI_Finalize();
    return 0;
}
```

Ejecución

```
mpicc master.c -o master_program
mpicc worker.c -o worker_program
mpirun -n 1 master_program
```

Estableciendo conexión entre dos aplicaciones independientes.

El proceso maestro genera aplicaciones de servidor y cliente con un proceso único para cada aplicación. El servidor abre un puerto y el cliente se conecta a ese puerto. Luego, el cliente envía datos al servidor con `MPI_Send` para verificar que se haya establecido la conexión.

master.c

```
#include "mpi.h"

int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);

    MPI_Comm intercomm;
    // Spawn two applications with a single process for each application.
    // Server must be spawned before client otherwise the client will complain at
    MPI_Lookup_name().
    MPI_Comm_spawn("server", MPI_ARGV_NULL, 1, MPI_INFO_NULL, 0, MPI_COMM_SELF, &intercomm,
    MPI_ERRCODES_IGNORE);
    MPI_Comm_spawn("client", MPI_ARGV_NULL, 1, MPI_INFO_NULL, 0, MPI_COMM_SELF, &intercomm,
    MPI_ERRCODES_IGNORE);

    MPI_Finalize();
    return 0;
}
```

servidor.c

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

int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);

    // Open port.
    char port_name[MPI_MAX_PORT_NAME];
    MPI_Open_port(MPI_INFO_NULL, port_name);

    // Publish port name and accept client.
    MPI_Comm client;
    MPI_Publish_name("name", MPI_INFO_NULL, port_name);
    MPI_Comm_accept(port_name, MPI_INFO_NULL, 0, MPI_COMM_WORLD, &client);

    // Receive data from client.
    int recvbuf;
    MPI_Recv(&recvbuf, 1, MPI_INT, 0, 0, client, MPI_STATUS_IGNORE);
    printf("recvbuf = %d\n", recvbuf);

    MPI_Unpublish_name("name", MPI_INFO_NULL, port_name);

    MPI_Finalize();
}
```

```
    return 0;
}
```

cliente.c

```
#include "mpi.h"

int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);

    // Look up for server's port name.
    char port_name[MPI_MAX_PORT_NAME];
    MPI_Lookup_name("name", MPI_INFO_NULL, port_name);

    // Connect to server.
    MPI_Comm server;
    MPI_Comm_connect(port_name, MPI_INFO_NULL, 0, MPI_COMM_SELF, &server);

    // Send data to server.
    int sendbuf = 3;
    MPI_Send(&sendbuf, 1, MPI_INT, 0, 0, server);

    MPI_Finalize();
    return 0;
}
```

Línea de comando

```
mpicc master.c -o master_program
mpicc server.c -o server
mpicc client.c -o client
mpirun -n 1 master_program
```

Lea Creación y gestión de procesos. en línea: <https://riptutorial.com/es/mpi/topic/9416/creacion-y-gestion-de-procesos->

Capítulo 5: Ejecutando un programa MPI

Parámetros

Parámetro	Detalles
<code>-n <num_procs></code>	El número de procesos MPI para iniciar el trabajo.

Examples

Ejecuta tu trabajo

La forma más sencilla de ejecutar su trabajo es usar `mpiexec` o `mpirun` (usualmente son la misma cosa y alias entre sí).

```
mpiexec -n 2 ./my_prog
```

Lea [Ejecutando un programa MPI en línea](https://riptutorial.com/es/mmpi/topic/2877/ejecutando-un-programa-mpi): <https://riptutorial.com/es/mmpi/topic/2877/ejecutando-un-programa-mpi>

Capítulo 6: Implementaciones de MPI

Observaciones

MPI es un estándar, no una biblioteca de programación. Hay muchas implementaciones de la norma. Los de código abierto más comunes son MPICH y Open MPI. Hay muchos derivados de estas dos bibliotecas que son de código abierto o comerciales (o ambos).

Es importante saber qué implementación tiene porque la forma en que compila o ejecuta su programa puede cambiar sutilmente.

Examples

Instalación de MPICH en Mac con Homebrew

```
brew install mpich
```

Instalar Open MPI en Mac con Homebrew

```
brew install open-mpi
```

Lea Implementaciones de MPI en línea:

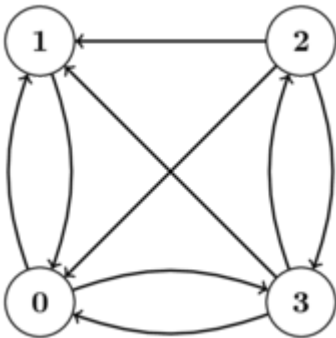
<https://riptutorial.com/es/mpi/topic/2870/implementaciones-de-mpi>

Capítulo 7: Topologías de proceso

Examples

Creación y comunicación de topologías gráficas.

Crea una topología de gráfico de manera distribuida para que cada nodo defina sus vecinos. Cada nodo comunica su rango entre los vecinos con `MPI_Neighbor_allgather`.



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

#define nnode 4

int main()
{
    MPI_Init(NULL, NULL);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int source = rank;
    int degree;
    int dest[nnode];
    int weight[nnode] = {1, 1, 1, 1};
    int recv[nnode] = {-1, -1, -1, -1};
    int send = rank;

    // set dest and degree.
    if (rank == 0)
    {
        dest[0] = 1;
        dest[1] = 3;
        degree = 2;
    }
    else if(rank == 1)
    {
        dest[0] = 0;
        degree = 1;
    }
    else if(rank == 2)
    {
        dest[0] = 3;
```

```

        dest[1] = 0;
        dest[2] = 1;
        degree = 3;
    }
    else if(rank == 3)
    {
        dest[0] = 0;
        dest[1] = 2;
        dest[2] = 1;
        degree = 3;
    }

    // create graph.
    MPI_Comm graph;
    MPI_Dist_graph_create(MPI_COMM_WORLD, 1, &source, &degree, dest, weight, MPI_INFO_NULL, 1,
&graph);

    // send and gather rank to/from neighbors.
    MPI_Neighbor_allgather(&send, 1, MPI_INT, recv, 1, MPI_INT, graph);

    printf("Rank: %i, recv[0] = %i, recv[1] = %i, recv[2] = %i, recv[3] = %i\n", rank,
recv[0], recv[1], recv[2], recv[3]);

    MPI_Finalize();
    return 0;
}

```

Lea Topologías de proceso en línea: <https://riptutorial.com/es/mpi/topic/9423/topologias-de-proceso>

Creditos

S. No	Capítulos	Contributors
1	Empezando con mpi	Community , foxcub , Harald , haraldkl , Hristo Iliev , Wesley Bland
2	Colectivos	foxcub , Harald , Shibli , Wesley Bland
3	Compilando un programa de MPI	Harald , Wesley Bland
4	Creación y gestión de procesos.	Shibli
5	Ejecutando un programa MPI	Wesley Bland
6	Implementaciones de MPI	Wesley Bland
7	Topologías de proceso	Shibli