Introducción a MPI

- ¿Qué es MPI?
- Características básicas de la programación con MPI
- Ejemplos
- LAM-MPI
- Enlaces de interés

¿Qué es MPI?

MPI (iniciales de Message Passing Interface)

MPI es una especificación para programación de paso de mensajes, que proporciona una librería de funciones para C, C++ o Fortran que son empleadas en los programas para comunicar datos entre procesos.

MPI es la primera librería de paso de mensajes estándar y portable, especificada por consenso por el MPI Forum, con unas 40 organizaciones participantes, como modelo que permita desarrollar programas que puedan ser migrados a diferentes computadores paralelos.

http://www.mpi-forum.org

El primer estándar MPI 1.0 fue acabado y publicado en mayo 1994. El estándar ha sido actualizado desde entonces, estando actualmente en desarrollo el MPI 2.

Características de MPI:

- Estandarización.
- Portabilidad: multiprocesadores, multicomputadores, redes, heterogéneos, ...
- Buenas prestaciones.
- Amplia funcionalidad.
- Existencia de implementaciones libres (mpich, LAM-MPI, ...)

La especificación detalla las funciones que se pueden utilizar, no el modo como se compilan y lanzan-ejecutan los programas, lo cual puede variar de una implementación a otra.

Características básicas de la programación con MPI

Siguiendo el modelo SPMD, el usuario escribirá su aplicación como un proceso secuencial del que se lanzarán varias instancias que cooperan entre sí.

Los procesos invocan diferentes funciones MPI que permiten

- iniciar, gestionar y finalizar procesos MPI
- comunicar datos entre dos procesos
- realizar operaciones de comunicación entre grupos de procesos
- crear tipos arbitrarios de datos

Funciones básicas de MPI

Cualquier programa paralelo con MPI puede implementarse con tan sólo 6 funciones, aunque hay muchas más funciones para aspectos avanzados. Todas ellas empiezan por MPI_ y obligan a que los programas MPI tengan `#include "mpi.h"`

Los programas MPI deben ser obligatoriamente inicializados y finalizados en MPI (MPI_Init, MPI_Finalize).

Los procesos en ejecución pueden saber cuántos procesos MPI forman parte de un grupo de procesos -***communicator*** en la terminología MPI- (MPI_Comm_size) y qué número de orden -empezando por 0- tiene en ese grupo de procesos (MPI_Comm_rank).

Los mensajes punto a punto deben ser enviados explícitamente por el emisor y recibidos explícitamente por el receptor (más adelante se explicarán las operaciones de comunicación colectivas), para lo cual pueden emplearse dos funciones básicas (MPI_Send y MPI_Recv).

Fichero cabecera:

#include <mpi.h>

Formato de las funciones:

codigo_error = MPI_nombre(parámetros ...)

Inicialización:

int MPI_Init (int *argc , char ***argv)

Comunicador:

Conjunto de procesos que se intercomunican. Por defecto podemos utilizar MPI_COMM_WORD , en cuyo caso el grupo de procesos es el conjunto de procesos lanzados conjuntamente para resolver un problema

Identificación de procesos:

MPI_Comm_rank (MPI_Comm comm , int *rank)

Procesos en el comunicador:

MPI_Comm_size (MPI_Comm comm , int *size)

Finalización:

int MPI_Finalize ( )

Mensajes:

Un mensaje estará formado por un cierto número de elementos de un mismo tipo MPI.

Tipos MPI básicos:

MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHOT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Tipos MPI derivados: los construye el programador.

Envío de un mensaje a otro proceso:

```
int MPI_Send ( void *posicion_de_memoria , int contador ,
  MPI_Datatype tipo , int destino , int etiqueta ,
  MPI_Comm comunicador )
```

Recepción de un mensaje de otro proceso:

```
int MPI_Recv ( void *posicion_de_memoria , int contador ,
  MPI_Datatype tipo , int origen , int etiqueta,
  MPI_Comm comunicador , MPI_Status *estado)
```

El receptor puede emplear MPI_ANY_TAG y/o MPI_ANY_SOURCE

Ejemplos

Ejemplo 1: hola.c **(guardar con el botón derecho)**

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

int main(int argc, char *argv[])
{
  int rank, size;
```

```
  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  printf("Hola! Soy el %d de %d\n", rank, size);

  MPI_Finalize();

  return 0;
}
```

Ejemplo 2: anillo.c (guardar con el botón derecho)

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

int main(int argc, char *argv[])
{
  MPI_Status status;
  int num, rank, size, tag, next, from;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  /* Usaremos una etiqueta arbitraria de valor 201.
     Calculamos el identificador (rango) del siguiente y del anterior, suponiendo un anillo */

  tag = 201;
  next = (rank + 1) % size;
  from = (rank + size - 1) % size;

  /* En uno de los procesos, el "primario", preguntamos un parametro */

  if (rank == 0) {
    printf("Introduce el numero de vueltas al anillo: ");
    scanf("%d", &num);

    printf("Proceso %d envia %d al proceso %d\n", rank, num, next);
    MPI_Send(&num, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
  }

  /* Los procesos "pasan" el numero de vueltas que faltan.
  Cuando llega al proceso 0, se descuenta una vuelta.
  Cuando un proceso recibe un 0, lo pasa y termina. */

  do {

    MPI_Recv(&num, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);
    printf("Proceso %d ha recibido %d\n", rank, num);

    if (rank == 0) {
      --num;
      printf("Proceso 0 descuenta una vuelta\n");
    }

    printf("Proceso %d envia %d al proceso %d\n", rank, num, next);
    MPI_Send(&num, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
  } while (num > 0);
  printf("Proceso %d termina\n", rank);

  /* El proceso "primario debe esperar el ultimo envio del ultimo proceso antes de terminar */
```

```
  if (rank == 0)
    MPI_Recv(&num, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);

  MPI_Finalize();
  return 0;
}
```

LAM-MPI

LAM-MPI (LAM viene de Local Area Multicomputer) es una implementación libre de MPI. Es un simple pero poderoso entorno para ejecutar y monitorizar aplicaciones MPI sobre redes de ordenadores.

http://www.lam-mpi-org

El sistema LAM-MPI está compuesto de tres partes

- una librería de funciones
- un *daemon*, lamd, que se ejecutará en todos los procesadores del multicomputador (virtual)
- una serie de órdenes para gestionar y monitorizar el multicomputador (virtual)

Las diferentes órdenes son explicadas con su manual mediante el correspondiente **`man`**

Arranque de LAM

El usuario crea un fichero que indique las máquinas que vayan a formar parte del multicomputador virtual.

`lamboot` `[-v ficherohosts]`

Arranca LAM (el multicomputador virtual) en una máquina o en un conjunto de máquinas.

`recon` `[-v ficherohosts]`

Verifica que se puede arrancar LAM en una máquina o en un conjunto de máquinas.

`lamnodes`

muestra los nodos (procesadores) que forman parte del multicomputador virtual.

Compilación de programas MPI en LAM

`mpicc`: es un "atajo" para ejecutar el compilador `cc` que además encuentra los ficheros "include" de MPI y enlaza con las librerías MPI necesarias.

ejemplo: `mpicc -o programa programa.c`

Ejecución de programas MPI en LAM

`mpirun`. Una aplicación MPI con el modelo SPM se lanza con una simple orden que indica cuántas

instancias del programa se ejecutarán.

ejemplo: `mpirun -np 4 programa`

Monitorización de aplicaciones MPI en LAM

El estado de los diferentes procesos MPI y de los mensajes que se intercambian puede ser monitorizado en todo momento.

`mpitask`

muestra el estado de los procesos en ejecución.

`mpimsg`

muestra los mensajes que estén enviándose o que no hayan sido recibidos todavía.

Limpieza de LAM

`lamclean` `[-v]`

Elimina todos los procesos MPI lanzados por el usuario y que no hayan terminado.

Terminación de LAM

`lamhalt`

Termina la sesión de LAM, eliminando el *daemon* `lamd`, y debe ser ejecutado cuando no se vaya a seguir trabajando con LAM-MPI.

Enlaces de interés

Algunas páginas con contenidos similares

en castellano

Introducción a MPI - U.Autónoma de Madrid : explicación bastante completa en HTML con aspectos de implementación de LAM-MPI

Algoritmos y Programación Paralela - U. Murcia - Programación en memoria distribuida: MPI : pantallazos de clase

Programación Distribuida y Paralela - U. Granada - Introducción a MPI : pantallazos de clase; algunos aspectos se refieren a la implementación MPICH

Curso de Programación en Paralelo con MPI - U. Sevilla : estilo pantallazos

Programación de Multicomputadores con MPI - UJI : pantallazos de clase, con explicación medianamente detallada

Programación de aplicaciones paralelas con MPI - Universidad del País Vasco - es de

1997, pero es texto completo y detallado

en inglés

LAM/MPI Parallel Computing en http://www.lam-mpi.org

Curso tutorial de MPI en el NCSA - Introduction to MPI : (necesita registro, que es libre)

University of Notre Dame - MPI Tutorial