Programación Paralela en MPI

Xavier Andrade V.

Introducción

- A pesar del aumento exponencial de la capacidad de los procesadores, esta no es suficiente para el estudio de muchos sistemas.

- Es necesario entonces usar varios procesadores para realizar una sola tarea.

- Aparece un factor importante, la comunicación entre procesos.

- La computación paralela es un área actual de estudio tanto en el desarrollo de sistemas como en la programación.

Sistemas Paralelos

Existen distintas maneras de interconectar varios procesadores en un sistema.

Sistemas de Memoria Compartida

- Están compuestos de procesadores y módulos de memoria interconectados.
- Existe un direccionamiento de memoria común para todos los procesadores.
- La comunicación entre procesos es muy rápida.
- Escalan a un máximo del orden 100 procesadores por problemas de rendimiento y costo.

Sistemas de Memoria Distribuida

- Cada procesador tiene su propia memoria.
- Un sistema de interconexión permite acceder la memoria de los otros procesadores.
- Escalan a miles de procesadores.

Programación en Paralelo

Existen distintas formas de programar aplicaciones que corran en varios procesadores, estas se diferencian en complejidad y escalabilidad.

OpenMP

- Es el compilador el que paraleliza el código.
- Debe ser asistido por directivas dadas por el programador.
- Es poco eficiente y solo capaz de generar código para sistemas de memoria distribuida.

Memoria Compartida y Threads

- Los distintos procesos comparten un área de memoria que usan para compartir información.
- Se adapta bien a sistemas de memoria compartida.
- Complicada e ineficiente de implementar en sistemas de memoria distribuida.

Paso de Mensajes

- La comunicación se hace explícitamente mediante mensajes que contienen información.
- Permite un fino control de la comunicación, la parte más costosa del cálculo en paralelo.
- La programación resulta más compleja.

MPI

MPI es un estándar de programación en paralelo mediante paso de mensajes que permite crear programas portables y eficientes

Introducción a MPI

- MPI fue creado en 1993 como un estándar abierto por fabricantes y usuarios de sistemas paralelos.

- Cada proveedor de un sistema paralelo implementa MPI para su sistema.

- Existen implementaciones de fuente abierta, que permitieron el desarrollo de sistemas paralelos de bajo costo basados en software libre.

- MPI-2 apunta a ampliarse a otras áreas de programación distribuida.

Características de MPI

- Interfaz genérica que permite una implementación optimizada en cualquier sistema paralelo.
- Es una biblioteca que incluye interfaces para FORTRAN, C y C++.
- Define varias formas de comunicación lo que permite programar de manera natural cualquier algoritmo en paralelo.
- Está pensado para crear bibliotecas paralelas.

Programación Básica en MPI

La programación usando MPI no es compleja, pero es relativamente distinta a la programación de un código serial y existen factores nuevos que tener en cuenta.

Estructura de un Programa MPI

- Se debe incluir el encabezado `<mpi.h>`
- Se debe inicializar y terminar MPI con las funciones `MPI_Init` y `MPI_Finalize` respectivamente.
- El código entre estas llamadas será ejecutado simultáneamente por todos los procesadores.
- Fuera de ese lapso no está definido el comportamiento del programa (depende de la implementación de MPI).

Referencia: Inicialización y Finalización de MPI

- C

```
int MPI_Init(int * pargv, char *** pargv)
int MPI_Finalize(void)
```

- C++

```
void MPI::Init(int & argv,char & ** argv)
void MPI::Init()
void MPI::Finalize()
```

Comunicadores

- Un comunicador corresponde a un grupo de procesos sobre el que se realiza la comunicación.
- Dentro de un comunicador cada proceso tiene un rango que lo identifica.
- Existe un comunicador básico `MPI_COMM_WORLD` que contiene a todos los procesos.
- Existen funciones que nos permiten saber el rango y el número de procesos dentro de un comunicador.

Referencia: Comunicadores y sus Funciones Básicas

- C

```
MPI_Comm MPI_COMM_WORLD
int MPI_Comm_size(MPI_Comm comm, int *psize)
int MPI_Comm_rank(MPI_Comm comm, int *prank)
```

- C++

```
MPI::Intracomm MPI::COMM_WORLD
int MPI::Intracomm::Get_size()
int MPI::Intracomm::Get_rank()
```

Funciones Básicas de Comunicación

- La forma de comunicación en MPI es a través de mensajes que contienen datos.

- La forma más simple es la comunicación punto a punto, donde se envía un mensaje de un proceso a otro.

- Esto se realiza usando las funciones `MPI_Send` y `MPI_Recv`.

Referencia: Funciones Básicas de Comunicación en C

```
int MPI_Send(void *buf, int count, MPI_Datatype dtype,
                        int dest,int tag, MPI_Comm comm)
int MPI_Recv(void *buf, int count, MPI_Datatype dtype,
                         int src,int tag, MPI_Comm comm,
                                          MPI_Status *stat)
```

Referencia: Funciones Básicas de Comunicación en C++

```
void MPI::Intracomm::Send(void *buf, int count,
                                        MPI::Datatype dtype,
                                           int dest,int tag)
MPI::Status MPI::Intracomm::Recv(void *buf, int count,
                                       MPI_Datatype dtype,
                                          int src,int tag)
```

Referencia: Funciones Básicas de Comunicación, Argumentos

`void * buf`	Buffer de envío o recepción
`int count`	número de datos
`MPI_Datatype dtype`	tipo de datos
`int dest/src`	rango del nodo al que se envía/recibe
`int tag`	etiqueta que diferencia al mensaje para ignorarla en `MPI_Recv` se puede pasar `MPI_ANY_TAG`
`MPI_Comm comm`	comunicador (en C++ corresponde a `this`)
`MPI_Status * stat`	status de la recepción (solo `MPI_Recv`) para ignorarlo pasar `MPI_STATUS_IGNORE` (en C++ es devuelto por la funcion)

Tipos de Datos

- Al enviar un dato es necesario especificar el tipo de este.
- Cada tipo tiene una equivalencia con un instancia del tipo `MPI_Datatype`.
- Esto permite el uso de MPI en ambientes heterogéneos.
- Es posible generar tipos de datos más complejos, característica que trataremos más adelante.

Referencia: Tipos de Datos

Tipo	Tipo de dato en C	Tipo de dato en C++
	MPI_Datatype	MPI::Datatype
char	MPI_CHAR	MPI::CHAR
bool		MPI::BOOL
int	MPI_INT	MPI::INT
unsigned int	MPI_UNSIGNED	MPI::UNSIGNED
float	MPI_FLOAT	MPI::FLOAT
double	MPI_DOUBLE	MPI::DOUBLE
long double	MPI_LONG_DOUBLE	MPI::LONG_DOUBLE
	MPI_BYTE	MPI::BYTE

Código Básico en C

```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char **argv){
  int rank,size;

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

  printf("Hola mundo, Este es el rango %d de %d\n",rank,size);

  if(rank==1){
    double data=3.14;
    MPI_Send(&data,1,MPI_DOUBLE,0,27,MPI_COMM_WORLD);
  }
  if(rank==0){
    double data;
    MPI_Recv(&data,1,MPI_DOUBLE,1,MPI_ANY_TAG,
     MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    printf("El rango 0 dice %g\n",data);
  }

  MPI_Finalize();
  return 0;
}
```

Código Básico en C++

```
#include <mpi.h>
#include <iostream>
using namespace std;

int main(int argc,char **argv){
  MPI::Init(argc,argv);
  const int size=MPI::COMM_WORLD.Get_size();
  const int rank=MPI::COMM_WORLD.Get_rank();

  cout << "Hola mundo, este es el rango " << rank
       << " de " << size << endl;

  if(rank==1){
    double data=3.14;
    MPI::COMM_WORLD.Send(&data,1,MPI::DOUBLE,0,27);
  }
  if(rank==0){
    double data;
    MPI::COMM_WORLD.Recv(&data,1,MPI::DOUBLE,1,MPI::ANY_TAG);
    cout << "El rango 0 dice " << data << endl;
  }

  MPI::Finalize();
  return 0;
}
```

Comportamiento del Envío de Mensajes

- Los envíos de datos en MPI están diseñados para lograr el máximo rendimiento posible.

- La función de envío `MPI_Send` tiene 2 comportamientos posibles, dependiendo del tamaño del mensaje.

  - Guardar el mensaje en un buffer y retornar inmediatamente.

  - Esperar a que el proceso de destino empieze a recibir.

- El tamaño de mensaje para el cual esto ocurre depende de la implementación y es muy variable.

Deadlocks

- Un deadlock ocurre cuando un proceso queda esperando un mensaje que nunca recibirá.

- Un caso típico se puede observar en el código siguiente, que debido al comportamiento ambiguo de `MPI_Send` puede o no producir un deadlock.

```
if(rank==0) {
  MPI_COMM_WORLD.Send(vec1,vecsize,MPI::DOUBLE,1,0);
  MPI_COMM_WORLD.Recv(vec2,vecsize,MPI::DOUBLE,1,MPI::ANY_TAG);
}
if(rank==1) {
  MPI_COMM_WORLD.Send(vec3,vecsize,MPI::DOUBLE,0,0);
  MPI_COMM_WORLD.Recv(vec4,vecsize,MPI::DOUBLE,0,MPI::ANY_TAG);
}
```

Comunicación Colectiva

- En ocasiones es necesario transmitir información entre un grupo de procesadores.
- Para esto MPI provee varias rutinas de comunicación colectiva.
- Estas se basan en un comunicador y deben ser llamadas por todos los miembros de este.

Comunicación Colectiva: `MPI_Barrier`

- La rutina `MPI_Barrier` se bloquea hasta ser llamada por todos los procesos de un grupo.

- Permite sincronizar la ejecución de algunas tareas.

```
int MPI_Barrier(MPI_Comm comm)
void MPI::Intracomm::Barrier()
```

Comunicación Colectiva: MPI_Bcast

- MPI_Bcast copia los valores en un arreglo del nodo root al arreglo en todo el resto de los nodos.

```
int MPI_Bcast(void * buf, int count, MPI_Datatype dtype,
                                    int root, MPI_Comm comm)
                                    int root, MPI_Comm comm)
void MPI::Intracomm::Bcast(void * buf, int count,
                                          MPI_Datatype dtype,
                                                int root)
```

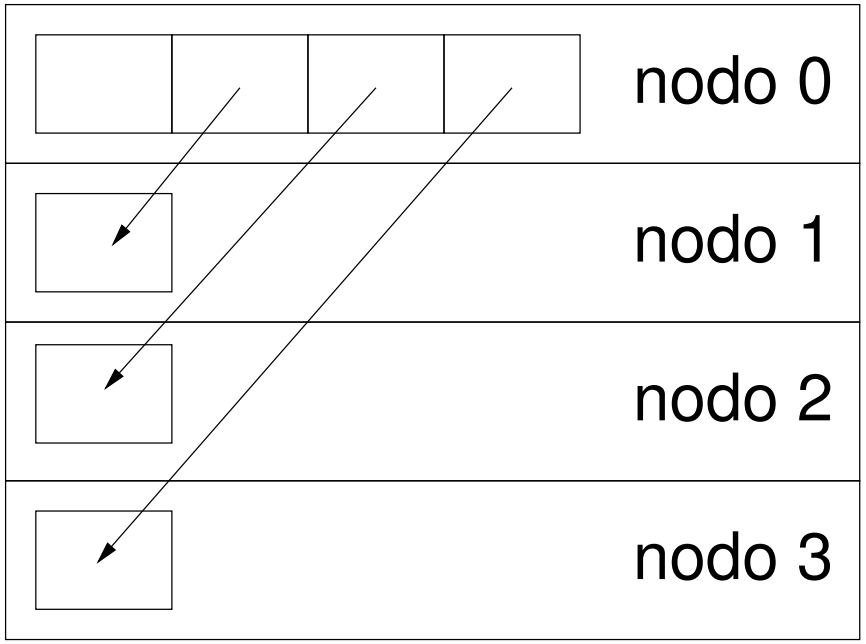
Comunicación Colectiva: `MPI_Gather` y `MPI_Scatter`

- `MPI_Scatter` toma un arreglo de datos en el nodo `root` y los distribuye a los nodos.

- `MPI_Gather` realiza la acción inversa, trayendo la información de los nodos a un arreglo en el nodo `root`.

- Existen variantes más sofisticadas de estas funciones.

Esquema de comunicación de `MPI_Scatter`



Referencia: `MPI_Scatter` y `MPI_Gather` en C

```
int MPI_Gather(void *sbuf, int scount, MPI_Datatype sdtype,
               void *rbuf, int rcount, MPI_Datatype rdtype,
                                       int root, MPI_Comm comm)
int MPI_Scatter(void *sbuf, int scount, MPI_Datatype sdtype,
               void *rbuf, int rcount, MPI_Datatype rdtype,
                                       int root, MPI_Comm comm)
```

Referencia: `MPI::Scatter` y `MPI::Gather` en C++

```
void MPI::Intracomm::Gather(void *sbuf, int scount,
                            MPI_Datatype sdtype,
                            void *rbuf, int rcount,
                            MPI_Datatype rdtype,
                            int root)
void MPI::Intracomm::Scatter(void *sbuf, int scount,
                            MPI_Datatype sdtype,
                            void *rbuf, int rcount,
                            MPI_Datatype rdtype,
                            int root)
```

Comunicación Colectiva: `MPI_Reduce`

- Esta función realiza una operación matemática de reducción sobre un grupo de datos disperso en los nodos.

```
int MPI_Reduce(void *sbuf, void* rbuf, int count,
                                   MPI_Datatype dtype, MPI_Op op,
                                         int root, MPI_Comm comm)
void MPI::Intracomm::Reduce(void *sbuf, void* rbuf, int count,
                                   MPI_Datatype dtype, MPI_Op op,
                                                       int root)
```

Tipos de Operaciones Para `MPI_Reduce`

Nombre C	Nombre C++	Operación
`MPI_MAX`	`MPI::MAX`	Busca el máximo
`MPI_MIN`	`MPI::MIN`	Busca el mínimo
`MPI_SUM`	`MPI::SUM`	Suma los valores
`MPI_PROD`	`MPI::PROD`	Multiplica los valores
`MPI_LAND`	`MPI::LAND`	And lógico
`MPI_BAND`	`MPI::BAND`	And por bits
`MPI_LOR`	`MPI::LOR`	Or lógico
`MPI_BOR`	`MPI::BOR`	Or por bits
`MPI_LXOR`	`MPI::LXOR`	Xor lógico
`MPI_BXOR`	`MPI::BXOR`	Xor por bits
`MPI_MAXLOC`	`MPI::MAXLOC`	Busca el máximo y su ubicación
`MPI_MINLOC`	`MPI::MINLOC`	Busca el mínimo y su ubicación

Programación Avanzada

MPI tiene muchas características, pero no es necesario conocerlas todas para escribir un programa; sin embargo las capacidades más avanzadas permiten simplificar la programación y lograr el máximo rendimiento.

Tipos de Datos Compuestos

- A veces es necesario enviar datos que no están en un arreglo.
  - Una estructura o clase compuesta de distintos tipos de datos.
  - Un subarreglo de un arreglo multidimensional.
- Es posible enviarlos usando varios mensajes, lo que resulta ineficiente.
- En MPI es posible crear nuevos tipos de datos, compuestos de tipos base y que pueden estar separados en memoria.

Formas Avanzadas de Comunicación Punto a Punto

- La función `MPI_Sendrecv` realiza un envío y una recepción simultáneas.

- Es posible realizar envíos con un comportamiento explícito.
  - `MPI_Bsend` copia el mensaje a un *buffer* especificado por el usuario y retorna antes de que el mensaje se haya enviado.
  - `MPI_Ssend` y `MPI_Rsend` esperan a que el envío ya haya sido realizado antes de retornar.
  - `MPI_Rsend` fallará si no hay un recibo correspondiente al momento de ejecutarse.

Envíos y Recibos No Bloqueantes

- Esta forma de envío permite realizar de manera simultánea comunicación y cómputo.
- La comunicación se inicia con `MPI_Isend` o `MPI_Irecv`, de los cuales se recibe un handler con el que se verifica que la comunicación haya finalizado.
- El programador es responsable de no modificar los datos que están siendo manipulados mientras se completa el envío.
- También es posible crear de esta forma canales persistentes de comunicación que pueden ser usados varias veces.

Comunicadores

- A partir del comunicador básico `MPI_COMM_WORLD` es posible construir copias de este o comunicadores con un subconjunto de los procesos.

- Esto permite aislar la comunicación entre distintas partes de un programa.

- Separar lógicamente los procesos.

- Y realizar rutinas de comunicación colectiva en un subconjunto de procesos.

Topologías Virtuales

- Muchas veces en un algoritmo de paralelización se hace una distribución espacial de los procesos donde la comunicación se hace más frecuentemente a primeros vecinos.

- Las Topologías Virtuales facilitan la programación y permiten optimizar la distribución de acuerdo a la topología física de interconexión del sistema.

- Una topología se implementa con un comunicador donde cada proceso puede ser identificado con su posición.

- Existen 2 tipos de topologías virtuales.

Tipos de Topologías Virtuales

- Cartesianas:
  - Cada proceso corresponde a un punto en una grilla cuadrada o toroidal de dimensión arbitraria.
  - Es posible construir comunicadores para las subgrillas en cada dimensión.
- Grafos
  - Una forma general donde el programador especifica todas las conexiones que tiene la topología.

Recursos

- **MPI Forum:** `http://www.mpi-forum.org/`
- **LAM-MPI:** `http://www.lam-mpi.org/`
- **MPICH:** `http://www-unix.mcs.anl.gov/mpi/mpich/`
- **MPI: The Complete Reference** Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, Jack Dongarra.

  `http://www.netlib.org/utk/papers/mpi-book/mpi-book.html`