SISTEMAS PARALELOS Y DISTRIBUIDOS

GRADO EN INGENIERÍA INFORMÁTICA – INGENIERÍA DE COMPUTADORES

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

UNIVERSIDAD DE SEVILLA

http://www.atc.us.es

Autor: José Luis Guisado Lizar

Noviembre de 2012

INTRODUCCIÓN A MPI

- MPI (Message-Passing Interface) es una biblioteca de funciones que implementan la **ejecución distribuida** de un programa mediante **paso de mensajes**
  - Pueden ser llamadas desde un programa en C o en FORTRAN
- MPI es realmente la especificación de una interfaz (API) de dicha biblioteca de funciones
- Estándar definido y mantenido por el “MPI Forum”, consorcio integrado por representantes de muchas organizaciones
- Existen múltiples implementaciones reales del estándar MPI:
  - MPICH, LAM/MPI, OpenMPI...

- MPI: biblioteca de funciones que implementan la **ejecución distribuida** de un programa mediante **paso de mensajes**

- MODELO DE PROGRAMACIÓN DE MEMORIA COMPARTIDA:
  - Distintas CPU/núcleos pueden acceder directamente a las variables del programa en memoria
  - Ejemplo: OpenMP

- MODELO DE PROGRAMACIÓN DE MEMORIA DISTRIBUIDA:
  - Asume que cada CPU sólo tiene acceso a su propio espacio de memoria
  - Ejemplos: MPI, PVM

- Ejemplo: Procesador multi-núcleo (Sistema SMP):
  - Un único espacio compartido de memoria
- MPI:
  - Ejecutar un programa paralelo en el que múltiples procesos cooperan.
  - Cada proceso (residiendo en una CPU/núcleo) sólo tiene acceso a su propio espacio de memoria.
  - Para acceder/alterar memoria, variables o información de otro proceso necesita intercambiar mensajes con él.
- Una ventaja de un modelo de programación distribuida (como MPI):
  - Le permite trabajar en cualquier tipo de entorno paralelo:
    - Procesador multinúcleo
    - Colección de servidores distribuidos (clúster)
    - Entorno grid o cloud
- OpenMP: Sólo puede trabajar en un sist. de memoria compartida (p. ej. un procesador multinúcleo)

MPI: CONCEPTOS BÁSICOS

- MPI opera a través de llamadas a funciones
- Estudiaremos cómo utilizar esas funciones desde programas en C
- Hay que indicar al compilador que enlace el código objeto del programa C con la biblioteca de funciones de MPI:
  - # include <mpi.h>
- Funciones MPI:
  - Nomenclatura: “MPI_Nombre-de-la-funcion”
  - Devuelven un código de error que indica si se han ejecutado con éxito:

```
if (err == MPI_SUCCESS)
{
   ... /* la rutina se ejecutó correctamente */
} else {
...
}
```

SISTEMAS PARALELOS Y DISTRIBUIDOS, GII-IC

MPI: CONCEPTOS BÁSICOS

- **COMUNICADORES**:
  - Un comunicador define un grupo de procesos a los cuales un comando en particular se aplicará
  - Todas las funciones de comunicación de MPI tienen como uno de sus parámetros un comunicador, que define el contexto en el que la comunicación tendrá lugar
  - El comunicador predefinido **MPI_COMM_WORLD**:
    - Consta de todos los procesos ejecutándose cuando comienza la ejecución del programa
    - Es decir: Incluye todos los procesos del programa
    - Es el utilizado al programar en MPI a nivel básico

MPI: FUNCIONES BÁSICAS

- **MPI_Init** – Inicializa MPI
- **MPI_Comm_size** – ¿Cuántos procesos hay?
  - Obtiene el nº de procesos lanzados por MPI, típicamente NCPUs - 1
- **MPI_Comm_rank** – Mi número de proceso
  - Solicita un nº de proceso (“rank” o rango) del proceso maestro MPI, que tiene rank = 0
- **MPI_Send** – Enviar un mensaje
- **MPI_Recv** – Recibir un mensaje
- **MPI_Finalize** – Cerrar el universo MPI

DEFINICIONES DE FUNCIONES MPI

- **MPI_Init(int *argc, char **argv[])**
  - Inicializa MPI. Debe ser llamada una única vez antes de emplear cualquier otra función MPI.
  - Lanza los procesos MPI en cada nodo
  - Sus parámetros son punteros a los parámetros de la función *main* del programa: argc y argv. Es decir, toma como parámetros la dirección de argc y la de argv.
  - Pasa los argumentos recibidos en la línea de comando al proceso lanzador de MPI
- **MPI_Comm_size(MPI_Comm comm, int *size)**
  - [IN **comm**]: Comunicador
  - [OUT **size**]: Número de procesos en el grupo de comm (integer)
    - Devuelve el número de procesos que participan en un comunicador. Si éste es el comunicador global predefinido MPI_COMM_WORLD, indica el número total de procesos involucrados en el programa

DEFINICIONES DE FUNCIONES MPI

- **MPI_Comm_rank(MPI_Comm comm, int *rank)**
  - [IN **comm**]: Comunicador
  - [OUT **rank**] : rango del proceso que llama en el grupo del comunicador
    - Devuelve el rango del proceso actual en un comunicador.
    - Cada comunicador contiene un grupo de procesos, cada uno de los cuales tiene un identificador único (rango), que es un número entero que comienza en 0: (0,1,2,3,...).

- **int MPI_Finalize(void)**
  - Función que debe ser llamada al finalizar el programa, cierra adecuadamente el entorno MPI.
  - El usuario debe asegurarse que todas las comunicaciones han sido completadas antes de llamar a MPI_Finalize.
  - Después de ella, ninguna otra rutina MPI puede ser llamada (incluyendo MPI_Init)

PRIMER EJEMPLO BÁSICO : programa hola_mundo.c


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

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

  MPI_Init(&argc, &argv);               /* Inicializa MPI */
  printf("Hola mundo\n");
  MPI_Finalize();                       /* Cierra el universo MPI */
  return(0);
}
~
~
~
~
~
~
~
~
~
~
~
~
~
"hola_mundo.c" 10L, 213C
```

PRIMER EJEMPLO BÁSICO : programa hola_mundo.c

- COMPILACIÓN: mpicc nombre_archivo.c –o nombre_archivo
- EJECUCIÓN: mpirun –np 4 nombre_archivo

```
user@pelican:~/exercises_mpi$ ls hola_mundo.c -la
-rwxr-xr-x 1 user user 213 Nov 10 13:29 hola_mundo.c
user@pelican:~/exercises_mpi$ mpicc hola_mundo.c -o hola_mundo
user@pelican:~/exercises_mpi$ mpirun -np 4 hola_mundo
```

- Para ver más detalles: mpirun –v ... (verbose)

SISTEMAS PARALELOS Y DISTRIBUIDOS, GII-IC

MPI → EJECUCIÓN EN PARALELO

- TRATE DE PENSAR EN PARALELO:
  - Al ejecutar `$ mpirun -np N` cada uno de los N procesos ejecuta una copia del código fuente
  - Cada variable se duplica N veces y puede tomar diferentes valores en los distintos procesos
  - Si se quiere chequear los valores de variables con printf, hay que tener en cuenta:
    - No en todas las implementaciones de MPI se cumple que todos los nodos tengan acceso a E/S
    - En caso de que sí lo tengan, si chequea los valores de variables con printf muestre siempre el rango del proceso en la salida de cada instrucción printf

PROGRAMA hola_mundo_avanzado.c


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

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

  int mi_rango, tamanno;

  MPI_Init(&argc, &argv);               /* Inicializa MPI */
  MPI_Comm_rank(MPI_COMM_WORLD, &mi_rango);
  MPI_Comm_size(MPI_COMM_WORLD, &tamanno);

  if ( mi_rango == 0 )
        printf("Soy el proceso %i de %i: !Hola mundo!\n", mi_rango, tamanno);
  else
        printf("Yo soy el proceso %i de %i.\n", mi_rango, tamanno);

  MPI_Finalize();                       /* Cierra el universo MPI */
  return(0);
}
~
~
~
~
"hola_mundo_avanzado.c" 19L, 468C
```

- MPI_COMM_WORLD: El “comunicador universal” → todos los procesos de la aplicación paralela pertenecen a él

EJECUCIÓN Y PROGRAMACIÓN SPMD

- El comando de ejecución depende de la implementación concreta de MPI que se use, pero la esencia es la misma:
  - Se coloca una copia del ejecutable en cada procesador
  - Cada procesador comienza la ejecución de su copia del ejecutable
  - Procesos diferentes pueden ejecutar instrucciones distintas incluyendo en el programa una bifurcación condicional basada en el valor del rango del propio proceso (Por ejemplo: programa hola_mundo_avanzado.c)

- Programación SPMD (single-program multiple-data):
  - En la forma más general de programación MIMD, cada proceso ejecuta un programa diferente, pero puede conseguirse lo mismo con un único programa que incluye bifurcaciones según el rango del propio proceso
  - La programación SPMD es una forma particular de programación MIMD. No hay que confundirla con la programación SIMD

MPI_GET_PROCESSOR_NAME

- **MPI_Get_processor_name(char* name, int* resultlen)**
  - [OUT **name**] : nombre del nodo físico en que corre el proceso (cadena de caracteres, cuyo tamaño debe ser al menos igual a MPI_MAX_PROCESSOR_NAME)
  - [OUT **resultlen**] : número de elementos en el búffer de recepción (entero)
    - Devuelve el nombre del procesador en que la función fue llamada.

FUNCIONES DE COMUNICACIÓN PUNTO A PUNTO

- Incluyen llamadas SEND y RECEIVE entre dos procesos.
- Existen dos categorías básicas de funciones send y receive:
  - Bloqueantes: Retornan cuando el send (o el receive) se ha completado:
    - MPI_SEND
    - MPI_RECV
  - No bloqueantes: Retornan inmediatamente y queda al arbitrio del programador chequear o no si las llamadas se han completado:
    - MPI_ISEND
    - MPI_IRECV
    - Para verificar si las llamadas no bloqueantes se han completado se emplean los tests:
      - MPI_TEST
      - MPI_WAIT
- Veremos en principio sólo los send y receive bloqueantes estándar:
  - MPI_SEND y MPI_RECV.

MPI_SEND

- **int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)**
  - [IN **buf**]: dirección inicial del búffer de envío
  - [IN **count**]: número de elementos en el búffer de envío (entero positivo)
  - [IN **datatype**]: tipo de dato de cada elemento en el búffer de envío
  - [IN **dest**]: rango del proceso destino (entero)
  - [IN **tag**]: etiqueta del mensaje (entero)
  - [IN **comm**]: comunicador

    - Envía un mensaje conteniendo *count* elementos de un tipo de dato especificado, que comienza en la dirección de memoria *buf*, usando la etiqueta de mensaje *tag*, al proceso cuyo rango es *dest*, en el comunicador *comm*.

MPI_RECV

- **MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)**
  - [OUT **buf**]: dirección inicial del búffer de recepción
  - [IN **count**]: número de elementos en el búffer de recepción (entero positivo)
  - [IN **datatype**]: tipo de dato de cada elemento en el búffer de envío
  - [IN **source**]: rango del proceso fuente (entero) o MPI_ANY_SOURCE
  - [IN **tag**]: etiqueta del mensaje (entero) o MPI_ANY_TAG
  - [IN **comm**]: comunicador
  - [OUT **status**]: información de estado (estructura con tres campos que contienen el rango del proceso fuente, la etiqueta y el código de error producido)

    - Bloquea un proceso hasta que recive un mensaje del proceso cuyo rango es *source*, en el comunicador *comm*, con etiqueta de mensaje *tag*.
    - Se pueden usar los comodines *MPI_ANY_SOURCE* y *MPI_ANY_TAG* para recibir mensajes. En ese caso, el *status* de salida puede ser usado para saber la fuente y etiqueta del mensaje recibido.
    - El mensaje recibido se guarda en un búffer consistente en un espacio de memoria que contiene *count* elementos consecutivos del tipo especificado por *datatype*, comenzando en la dirección de memoria *buf*.
    - La longitud del mensaje recibido debe ser menor o igual que la longitud del búffer de recepción disponible.

Tipos de datos básicos de MPI

- MPI tiene constantes que definen los tipos de datos básicos
- Cada tipo de dato básico de C tiene su equivalente MPI, de tipo MPI_tipo, que debe ser usado en las llamadas en C:

TIPO DE DATO MPI	TIPO DE DATO C
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_SHORT	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	--

PROGRAMA saludos.c


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

main(int argc, char* argv[]) {
  int           mi_rango;          // rango de mi proceso
  int           p;                 // numero de procesos
  int           fuente;            // rango del emisor
  int           dest;              // rango del destinatario
  int           etiqueta = 0;      // etiqueta para los mensajes
  char          mensaje[100];      // almacenamiento para el mensaje
  MPI_Status    status;            // devuelve el status para el receptor

  MPI_Init(&argc, &argv);          // Inicializa MPI
  MPI_Comm_rank(MPI_COMM_WORLD, &mi_rango);
  MPI_Comm_size(MPI_COMM_WORLD, &p);

  if (mi_rango != 0) {
      /* Crea mensaje */
      sprintf(mensaje, "Saludos desde el proceso %d!", mi_rango);
      dest = 0;
      /* Usa strlen+1 para que '\0' sea transmitido */
      MPI_Send(mensaje, strlen(mensaje)+1, MPI_CHAR, dest, etiqueta, MPI_COMM_WORLD);
  } else {                         // mi_rango == 0
      for (fuente=1; fuente < p; fuente++) {
              MPI_Recv(mensaje,   100,   MPI_CHAR,   fuente,   etiqueta,   MPI_COMM_WORLD,
&status);
              printf("%s\n", mensaje);
      }
  }
  MPI_Finalize();                              // Cierra el universo MPI
}                                  // main
```

EVALUACIÓN DEL RENDIMIENTO

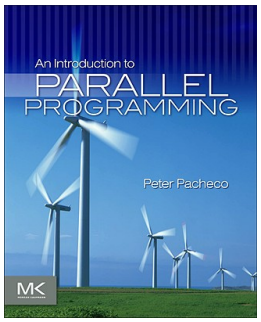
- MPI define una función para medir tiempos de ejecución con la que se puede evaluar el rendimiento de los programas
- Al ser una función del estándar MPI, tiene como ventaja su portabilidad

- **double MPI_Wtime(void)**
  - Devuelve un número de segundos en punto flotante, que representan una referencia de tiempo (wall-clock time).

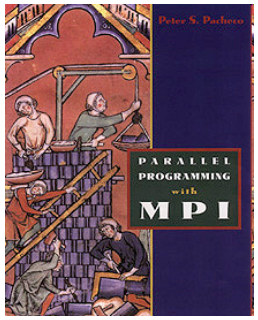
- Para medir la duración de un proceso, se puede llamar a *MPI_Wtime* justo antes de que comience y justo despueś de que termine. A continuación, se calcula la diferencia entre los dos tiempos.

MPI: BIBLIOGRAFÍA

- "An Introduction to Parallel Programming", Peter S. Pacheco. Elsevier - Morgan Kaufmann, 2011.



- "Parallel Programming with MPI", Peter S. Pacheco. Elsevier - Morgan Kaufmann, 1996.



- "MPI: A Message-Passing Interface Standard". Version 2.2. Message Passing Interface Forum. September 4, 2009. http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf