Práctica 3

Sesión 1

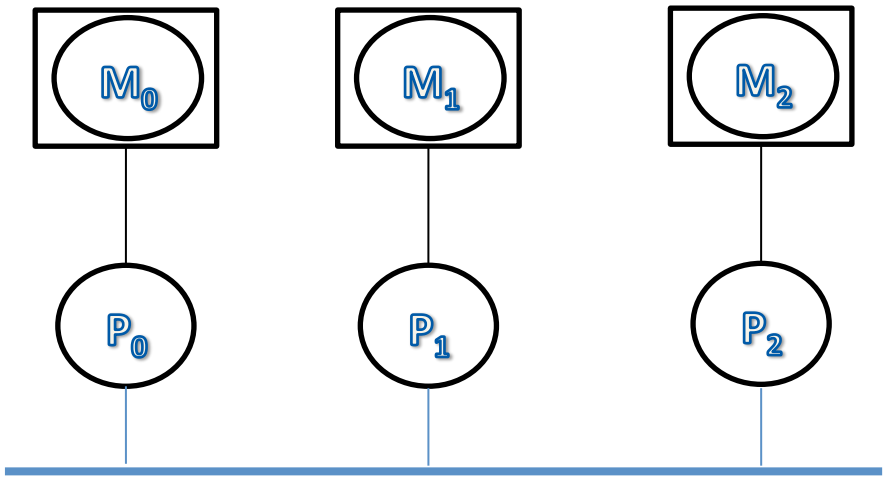
Contenido

- Primeros pasos con MPI:
  - programa hello.c
  - Modificación hello.c (id de proceso y número de procesos)

- Comunicaciones punto a punto (MPI_Send,MPI_Recv):
  - Cálculo del número Pi
  - Determinación del modelo de tiempo de comunicaciones: programa ping-pong

Recordatorio

- Arquitectura de memoria distribuida

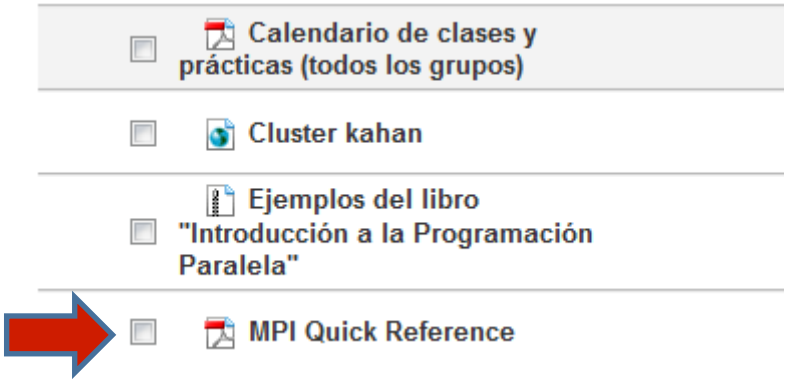


  - Todos los procesos tienen la misma copia del programa que se va a ejecutar
  - Todas las variables son locales
  - La única manera de intercambiar datos es el envío/recepción de mensajes
  - Pueden ejecutar diferentes instrucciones dependiendo de sus identificadores
  - Para que tenga éxito una comunicación es necesario que uno de los dos procesos implicados haga la petición del envío (**send**) y el otro la petición de recepción (**rec**):

$$P_0 : send(\ldots,1,\ldots) \longleftrightarrow P_1 : rec(\ldots,0,\ldots)$$

Recordatorio

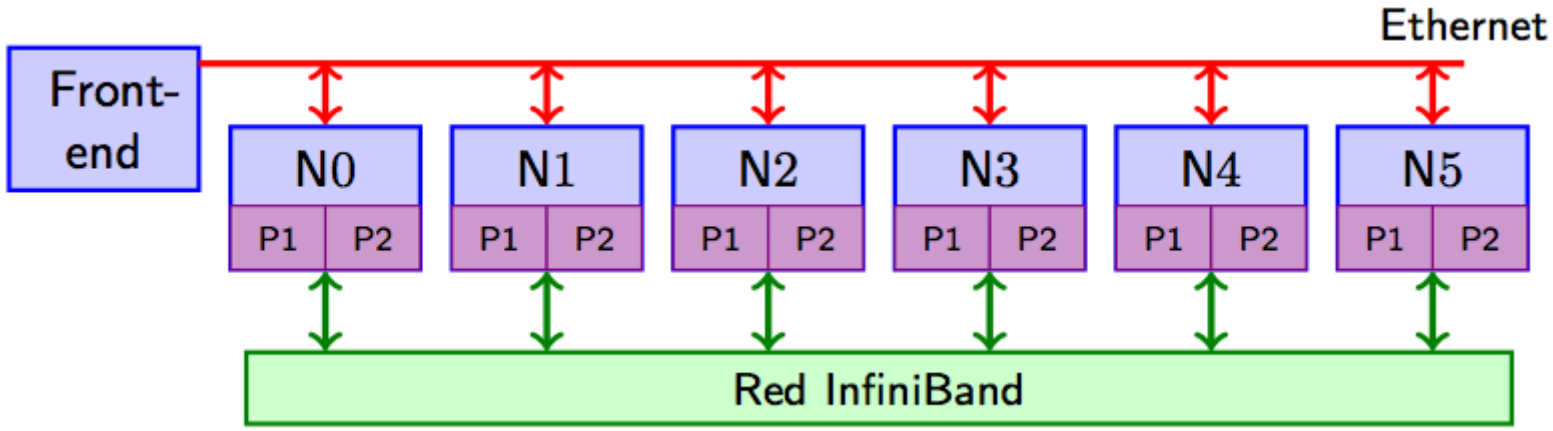
- En Recursos de Poliformat de la asignatura, hay una guía de referencia rápida que puede ser usada en el examen de MPI y es de interés para el seguimiento de las clases



..\..\Teoria\mpi-qref.pdf

Recordatorio

- Cluster Kahan



- La compilación del código se realiza mediante el comando `mpicc`
  - `mpicc` es una utilidad que invoca al compilador con las opciones apropiadas para la instalación particular de MPI
  - El entorno MPI solo está instalado en Kahan (y en el frontend)
  - Se puede consultar el manual en línea (`man`), para cualquier función.

Ejercicios 1-2 (I)

- Implementar programa hello.c y ejecutarlo:

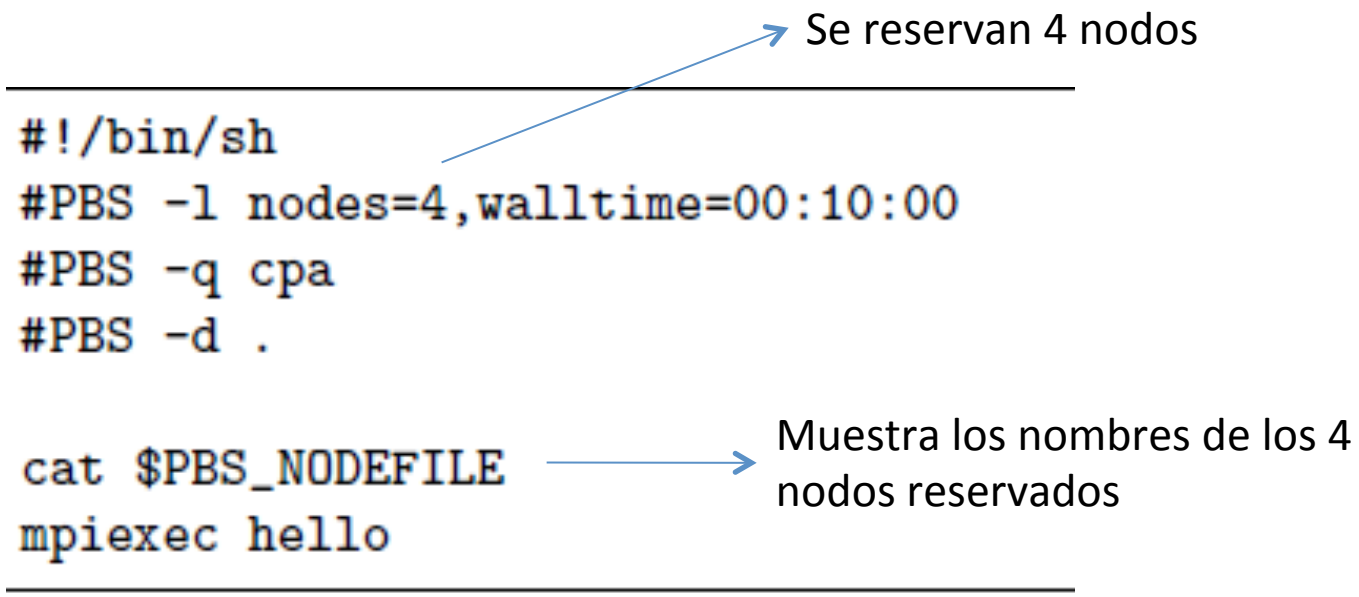
```
#include <stdio.h>
#include <mpi.h>
int main (int argc, char *argv[])
{
  MPI_Init(&argc, &argv);
  printf("Hello\n”);
  MPI_Finalize();
 return 0;
 }
```

  - **Compilación**: mpicc −Wall -o hello hello.c
  - Con “mpicc −show” (se muestran las opciones usadas por el script)
  - **Ejecución**: mpiexec hello
  - Si se ejecuta en la línea de comandos, se puede indicar el número de procesos que se usan:

    mpiexec −n 4 hello (se ejecuta con 4 procesos)
  - De esta forma, todos los procesos se lanzan sobre el mismo nodo

Ejercicios 1-2 (II)

- Scripts para la cola



Ejercicios 1 y 2 (III)

Dado que cada nodo tiene 32 cores, puede ser interesante lanzar varios procesos en el mismo nodo. Para ello modificaremos, el script indicando la cantidad de procesos por nodo (ppn) y también añadir una opción adicional

```
#PBS -l nodes=2:ppn=16,walltime=00:10:00
#PBS -W x="NACCESSPOLICY:SINGLEJOB"
```

Ejercicio 2

- Implementar programa hello.c y ejecutarlo:

```
#include <stdio.h>
#include <mpi.h>
int main (int argc, char *argv[])
{
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(…);
  MPI_Comm_size(…);
  printf("Hello world from process %d of %d\n", rank, size);
  MPI_Finalize();
 return 0;
 }
```

  - **Compilación**: mpicc -o hello hello.c
  - **Ejecución**: mpiexec hello
  - Si se ejecuta en la línea de comandos, se puede indicar el número de procesos que se usan:

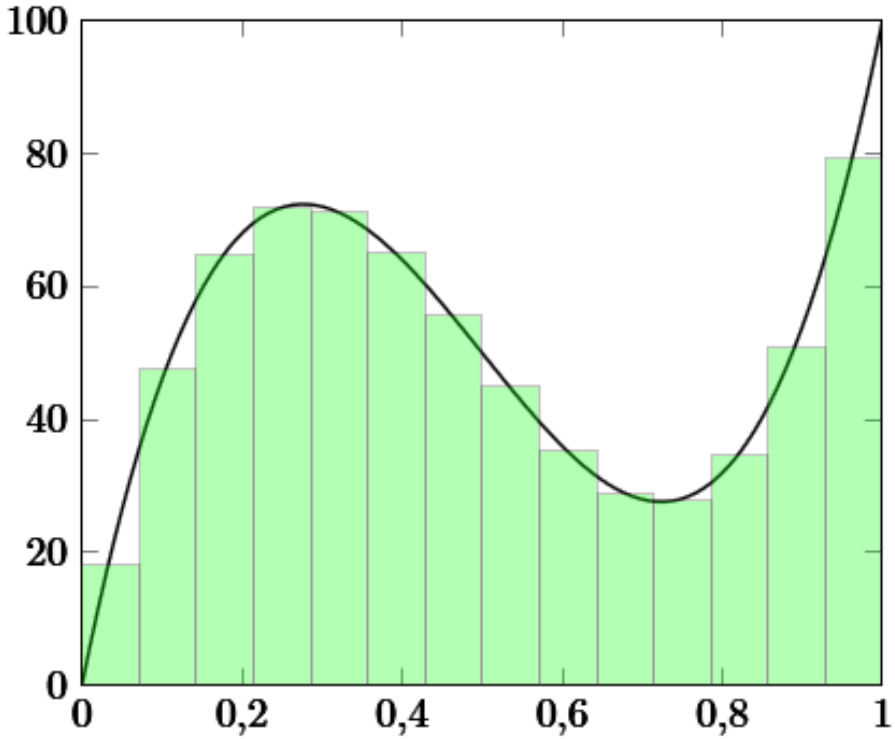
mpiexec −np 3 hello (se ejecuta con 3 procesos)

Calculo de Pi

- Cálculo del número pi/4 en MPI (suponiendo que hay 3 procesos)

$$\int_0^1 \frac{1}{1+x^2}\,dx = \frac{\pi}{4} \cong h\sum_{i=1}^{n} f(x_i) =$$

$$h\sum_{P_0} f(x_i) + h\sum_{P_1} f(x_i) + h\sum_{P_2} f(x_i)$$



h = tamaño del intervalo

x_i = puntos medios

$$f(x) = \frac{1}{1+x^2}$$

Calculo de Pi. Programa

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

int main(int argc,char *argv[])
{
  int    n, myid, numprocs, i;
  double mypi, pi, h, sum, x;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);

  if (argc==2) n = atoi(argv[1]);
  else n = 100;
  if (n<=0) MPI_Abort(MPI_COMM_WORLD,MPI_ERR_ARG);

  /* Cálculo de PI. Cada proceso acumula la suma parcial de un subintervalo */
  h   = 1.0 / (double) n;
  sum = 0.0;
  for (i = myid + 1; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += (4.0 / (1.0 + x*x));
  }
  mypi = h * sum;
  /* Reducción: el proceso 0 obtiene la suma de todos los resultados */
  MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

  if (myid==0) {
    printf("Cálculo de PI con %d procesos\n",numprocs);
    printf("Con %d intervalos, PI es aproximadamente %.16f (error = %.16f)\n",n,pi,fabs(pi-M_PI));
  }

  MPI_Finalize();
  return 0;
}
```

$$\text{mypi}(P_0) = h\sum_{P_0} f(x_i)$$

$$\text{mypi}(P_1) = h\sum_{P_1} f(x_i)$$

$$\text{mypi}(P_2) = h\sum_{P_2} f(x_i)$$

P_0 recibe en la variable pi el valor mypi(P_1) de P_1 y el valor mypi(P_2) de P_2, sumándoselos a mypi(P_0)

Ejercicio 3 (Cálculo de Pi)

- Transformar la llamada a MPI_Reduce en llamadas a comunicaciones punto a punto:

Si soy P0 (recibir en la variable pi los valores mypi de $P_1, P_2, \ldots, P_{numprocs-1}$ y acumularlos en mypi):

Para i=1:numprocs-1

rec(s, Pi) (mejor: recibir de cualquiera→ rec(s,any))

mypi= mypi+s;

Fin para

Escribir el resultado obtenido y el error cometido

En caso contrario (procesadores restantes, enviar a P_0 el valor de mypi)

send(mypi,P_0)

Recordatorio (envío estándar)

- **MPI_Send**. Se comporta como:
  - MPI_Bsend: envío de mensaje cortos (envíos con buffer)
  - MPI_Ssend: envío de mensaje largos (envíos síncronos)
- Explicación de los argumentos:

MPI_Send(void ***buf**, int **count,** MPI_Datatype **datatype**, int **dest**, int **tag**, MPI_Comm **comm**)

buf: puntero al dato que se envía (dato simple lleva delante &; array no tiene delante &)

datatype : tipo MPI de dato (MPI_DOUBLE, MPI_INT, MPI_CHAR, etc.)

count: número de datos (dato simple: 1 dato, array: número de elementos del array)

dest: identificador del proceso que recibirá el mensaje

tag: identificador/etiqueta del mensaje

comm: comunicador. Suele usarse el comunicador **MPI_COMM_WORLD**, el cual contiene a todos los procesos que se están ejecutando

Recordatorio (recepción estándar)

- **MPI_Recv**

(void ***buf,** int **count,**MPI_Datatype **datatype**, int **source**, int **tag**, MPI_Comm **comm**, MPI_Status ***status**)

buf: puntero al dato que se envía (dato simple lleva delante &; array no tiene delante &)

datatype : tipo MPI de dato (MPI_DOUBLE, MPI_INT, MPI_CHAR, etc.)

count: número de datos (dato simple: 1, array: número de elementos del array)

source : Identificador del proceso que envía el mensaje. Puede usarse la etiqueta **MPI_ANY_SOURCE** para indicar que se espera recibir desde cualquier proceso

tag: identificador/etiqueta del mensaje. Puede usarse la etiqueta **MPI_ANY_TAG** para indicar que se espera recibir mensajes con cualquier etiqueta

comm: comunicador (**MPI_COMM_WORLD** Comunicador universal)

status: información del mensaje:

status.MPI_SOURCE: proceso que ha realizado el envío

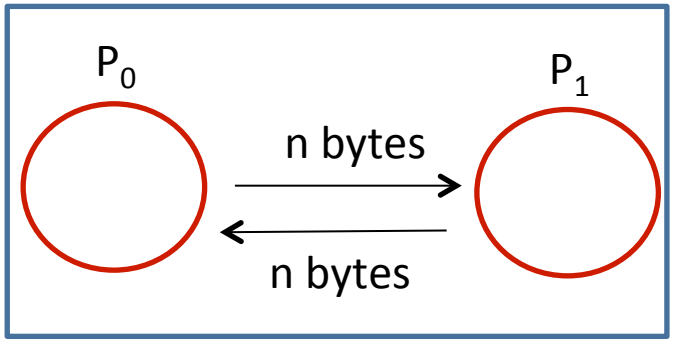
status.MPI_TAG: etiqueta del mensaje recibido

MPI_STATUS_IGNORE: etiqueta que se usa para el argumento **status** cuando nos da igual obtener información del mensaje recibido

El programa Ping-Pong (I)

- Sirve para obtener experimentalmente la fórmula que relaciona el envío de *n* bytes con el tiempo tardado en las comunicaciones (t_c)
  - t_s: tiempo de establecimiento de la señal
  - t_w: tiempo de envío de 1 byte
  - n: número de bytes
  - t_c: tiempo de comunicaciones

$$t_c = t_s + t_w n$$



- Determinación de t_s y t_w:
- Completar el programa ping-pong.c, teniendo en cuenta que:
  - El programa tiene como argumento el tamaño de mensaje
  - Usar `MPI_Wtime()` para medir tiempos. Se usa como la llamada de OpenMP `omp_get_time()`.
  - Para que los tiempos medidos sean significativos, el programa debe repetir la operación `NREPS` veces y mostrar el tiempo medio.
  - Las operaciones de envío y recepción se realizan con `MPI_Send` y `MPI_Recv` con `MPI_BYTE` como tipo de dato de envío/recepción.

El programa Ping-Pong (II)

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define NMAX 1000000
#define NREPS 100
int main(int argc,char *argv[])
{
  int n, myid, numprocs;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);

/* The program takes 1 argument: message size (n), with a default size of 100
    bytes and a maximum size of NMAX bytes*/
  if (argc==2) n = atoi(argv[1]);
  else n = 100;
  if (n<0 || n>NMAX) n=NMAX;
                .
```

El programa Ping-Pong (III)

```
/* COMPLETE: Get current time, using MPI_Wtime() */

/* COMPLETE: loop of NREPS iterations.
In each iteration, P0 sends a message of n bytes to P1, and P1 sends the same
message back to P0. The data sent is taken from array buf and received into
the same array. */

/* COMPLETE: Get current time, using MPI_Wtime() */

/* COMPLETE: Only in process 0.
Compute the time of transmission of a single message (in milliseconds) and
print it. Take into account there have been NREPS repetitions, and each
repetition involves 2 messages. */

  MPI_Finalize();
  return 0;
}
```