Práctica 3: Paralelización con MPI
Sesión 1: Primeros pasos con MPI

Mónica Chillarón

Computación Paralela (CPA)

Curso 2020/2021

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice

Práctica 3

- La práctica 3 consta de 4 sesiones de trabajo y una de examen
- No se entrega nada, se evalúa sólo mediante examen
- Fecha examen: 11 de Enero
- El examen sustituye a la sesión 3.5

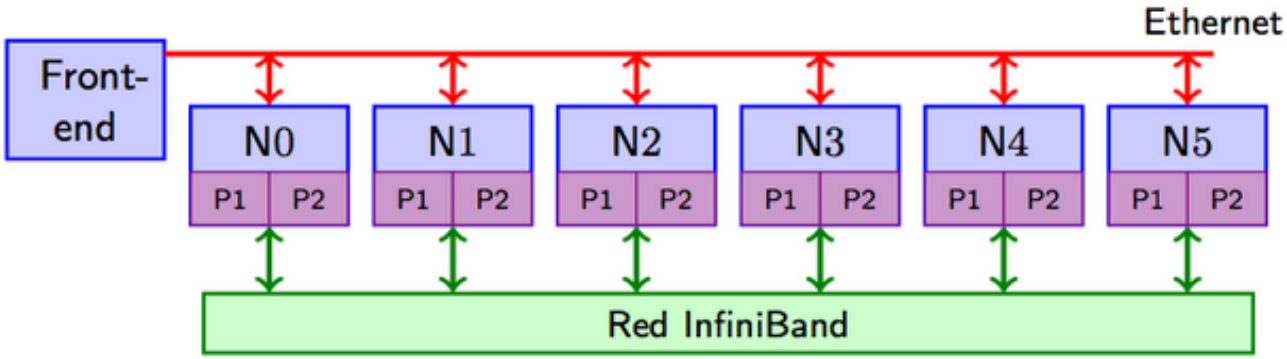
Cluster de Prácticas: Kahan

Especificaciones del cluster Kahan: http://personales.upv.es/jroman/kahan.html

Cada nodo consta de:

- 2 procesadores AMD Opteron 16 Core 6272, 2.1GHz, 16MB
- 32GB de memoria DDR3 1600
- Disco 500GB, SATA 6 GB/s
- Controladora InfiniBand QDR 4X (40Gbps, tasa efectiva de 32Gbps)

Agregado: 12 procesadores, 192 núcleos, 192 GB



Importante: Actualmente sólo hay activos 4 nodos (2 fallan). NUNCA USAR MÁS DE DOS NODOS PARA UN TRABAJO

Hola Mundo

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

int main (int argc, char *argv[])
{
  MPI_Init(&argc, &argv);
  printf("Hello world\n");
  MPI_Finalize();
  return 0;
}
```

- Programa en el que cada proceso imprime un hola mundo
- Compilar: **mpicc hello.c -o hello**
- Ejecutar en el frontend (para pequeñas pruebas): **mpiexec -n 4 hello**
- Con -n 4 indicamos que se ejecute el programa con 4 procesos

Hola Mundo

- Para mandar el trabajo al sistema de colas:

```
#!/bin/sh
#PBS -l nodes=2,walltime=00:10:00
#PBS -q cpa
#PBS -d .

cat $PBS_NODEFILE
mpiexec ./hello
```

- Reservamos máximo 2 nodos
- Con **cat** $*PBS_NODEFILE* podemos saber en qué nodos se está ejecutando
- De esta manera, se ejecutaría un proceso por nodo
- Si queremos más de un proceso por nodo:

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

Hola Mundo

- Ejercicio 1: Probar con distintos números de procesos, y también cambiando el script del trabajo para ejecutar varios procesos por nodo en el sistema de colas

- Ejercicio 2: Modifica el programa para obtener el identificador de proceso y el número de procesos, y mostrar esa información en el saludo:

  int MPI_Comm_rank(MPI_Comm comm, int *rank) para el identificador
  int MPI_Comm_size(MPI_Comm comm, int *size) para el número de procesos

Cálculo de Pi

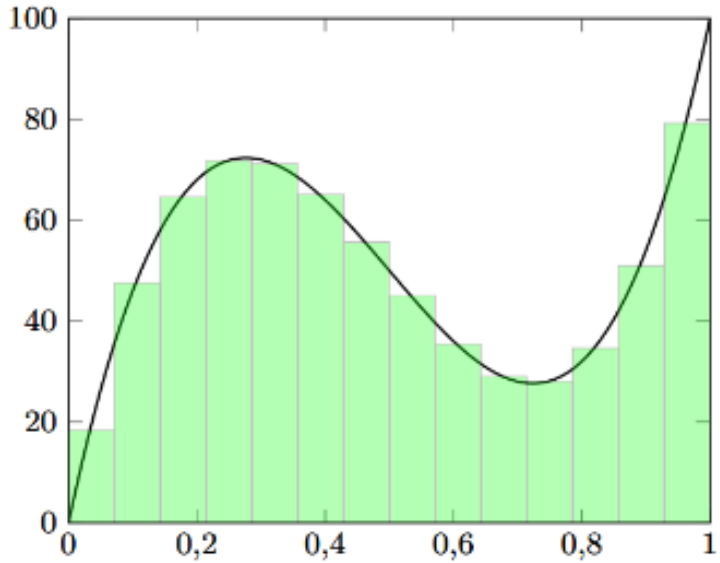


Figura 1: Interpretación geométrica de una integral.

$$\int_0^1 \frac{1}{1+x^2} dx = \frac{\pi}{4}$$

- Se hace un reparto cíclico manual tal y como hicimos en OpenMP:

```
for (i = 1; i <= n; i ++)
```
->
```
for (i = myid + 1; i <= n; i += numprocs)
```

- Cada proceso calcula su suma parcial y la guarda en mypi
- El proceso 0 suma todos los resultados parciales y lo guarda en pi:

```
/* Reducción: el proceso 0 obtiene la suma de todos los resultados */
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

Cálculo de Pi

Ejercicio 3: Modifica el programa, sustituyendo la llamada a la función MPI_Reduce por un fragmento de código que sea equivalente pero utilice solo comunicaciones punto a punto (MPI_Send y MPI_Recv). Para ello, lo más sencillo es hacer que todos los procesos envíen su suma parcial (mypi) al proceso 0, el cual se encargará de recibir cada valor y acumularlo sobre la variable pi.

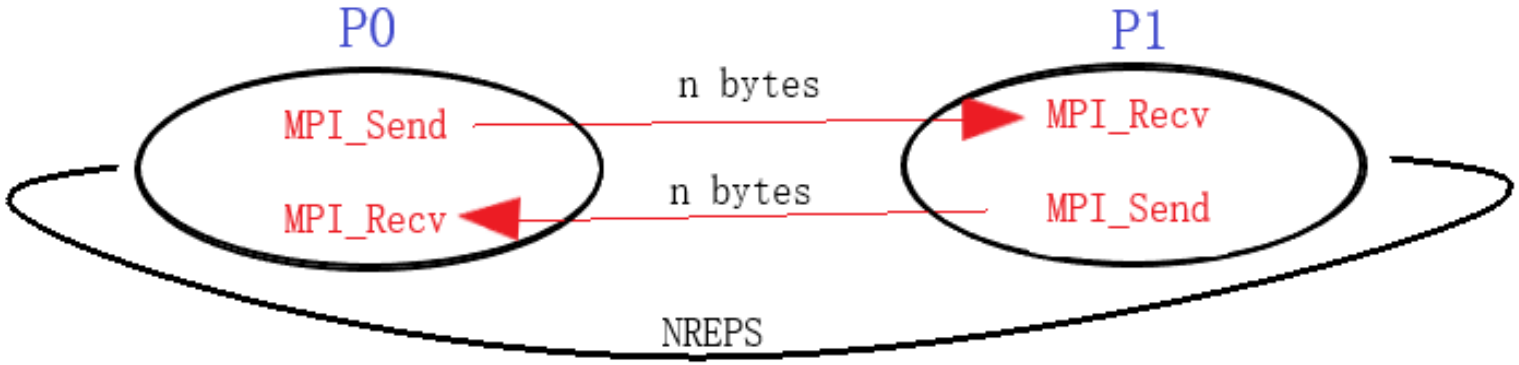
```
Si myid==0
    sumar mi resultado al resultado final

    para cada proceso
        recibir su resultado parcial
        sumar al resultado final
    fin para

Sino
   enviar mi resultado al proceso 0
```

El programa ping-pong

Programa para medir la velocidad de la red



- El P_0 envía (MPI_Send) un mensaje a P_1, que lo recibe (MPI_Recv) y se lo devuelve a P_0
- Se envían n bytes (n elementos de tipo MPI_BYTE) en cada mensaje (argumento del programa)
- Se mide el tiempo entre que P_0 empieza a enviar y termina de recibir
- Para medir tiempo: MPI_Wtime() funciona como omp_get_wtime()
- Para tener una medida más fiable, repetimos los envíos NREPS veces y sacamos la media
- Hay que tener en cuenta que estamos midiendo el tiempo de envío de 2 mensajes de n bytes cada vez (envío y recepción)

El programa ping-pong

- Ejercicio 4: Completa el programa ping-pong.c para que haga lo que se explica en la trasparencia anterior. El programa tiene como argumento el tamaño del mensaje, n (en bytes).

- Ejercicio 5: ¿Por qué se envían dos mensajes en cada iteración del bucle? ¿se podría eliminar el mensaje de respuesta de a P_1 a a P_0?

- Ejercicio 6: En cada iteración, el proceso P_0 tiene que hacer un envío y una recepción. ¿Podría utilizar para ello la función MPI_Sendrecv_replace? ¿y el proceso a P_1?