PRIMERA PRÁCTICA CALIFICADA DE PROGRAMACIÓN PARALELA

Apellidos y Nombres: Esquivel Grados Luis Germán
Código: 17200154
Ciclo: 2020-0
Profesor: Herminio Paucar

Teoría

1. Explique con sus palabras ¿qué es un proceso de una computadora?
2. Explique a qué se refieren cuando hablamos de una comunicación punto a punto entre 2 procesos, proponer un ejemplo de código.
3. ¿Qué es una memoria Ram (principal), Cache y Virtual? E indicar ¿cómo funcionan?
4. ¿En qué consiste la **programación en Memoria Distribuida** y la **programación en Memoria Compartida**?
5. Describa en 3 líneas como máximo e indicar los parámetros de los siguientes comandos del MPI:

a) MPI_Send(…)
b) MPI_Recv(…)
b) MPI_Reduce(…)
b) MPI_AllReduce(…)

Práctica

6. Utilizando **MPI**, implemente un algoritmo que determine el número de veces que un elemento x aparezca en un vector **A** con **n** elementos enteros. Se puede asumir que su algoritmo comienza con los elementos ya distribuidos entre los **p** procesos (**n/p** para cada uno)
7. Desarrolle un algoritmo en **MPI**, utilizando **p** procesadores para calcular **n!**
8. Suponga que **comm_sz = 8** y la cantidad de elementos es **n = 16.**

a) Diseñe un diagrama que explique como **MPI_Scatter** puede ser implementado usando comunicaciones basadas en árboles. Puede suponer que el origen del Scatter es el proceso con rank 0.

9. Hacer lo mismo para el **MPI_Gather**, en este caso con el proceso 0 como destino.

Resolución

1. Un proceso es la ejecución de un programa, es un conjunto de datos almacenados en memoria e instrucciones realizadas sucesivamente para lograr un fin.
2. Es el paso de mensajes entre únicamente dos procesos: uno envía y el otro recibe. El que envía lo hace a través de la función MPI_Send(…) y el que recibe, con la función MPI_Recv(…).

Ejemplo: Leer un entero n y enviarlo a todos los procesos.

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

```
int main(int argc, char *argv[]){
        int n, rank, size;
        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        if (rank == 0){
                cin>>n;
                for(int i=0;i<size;i++)
                        MPI_Send(&n,1,MPI_INT,i,0,MPI_COMM_WORLD);
        }
        else
             MPI_Recv(&n,1, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
        MPI_Finalize();
        return 0;
}
```

3. RAM: Memoria de acceso aleatorio, es aquella a la que los procesos envían sus datos e instrucciones.
Cache: Memoria de menor tamaño que la RAM, pero más rápida. Se usa para acceder a los datos más usados. Generalmente tiene 3 niveles, el primer nivel está en la CPU y es la más rápida de todas.
Virtual: Memoria de mayor tamaño que la RAM, Memoria en la que los procesos guardan sus datos e instrucciones cuando no hay espacio en la memoria RAM.
4. Programación en Memoria Distribuida: Es cuando todos los procesos tienen separada su propia porción de memoria, por lo cual un proceso no puede leer, editar ni eliminar los datos de otros procesos.
Programación en Memoria Compartida: Todos los procesos comparten la misma porción de memoria, por lo tanto, es posible que varios procesos lean, editen y/o eliminen datos que serán usados por otros procesos.

5.
a)
```
MPI_Send(const void *buf,
         int count,
         MPI_Datatype datatype,
         int dest,
         int tag,
         MPI_Comm comm)
```
Envía un dato o conjunto de datos a otro proceso a través del mismo comunicador.

Parámetros

buf
Dirección del elemento a enviar
count
Número de elementos a enviar
datatype

Tipo de dato del mensaje

dest

Rank del proceso receptor

tag

Etiqueta del mensaje

comm

Comunicador

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

Recibe un dato o conjunto de datos de otro proceso a través del mismo comunicador.

Parámetros

buf

Dirección de la variable que recibirá el mensaje

count

Número de elementos a recibir

datatype

Tipo de dato del mensaje

dest

Rank del proceso emisor

tag

Etiqueta del mensaje

comm

Comunicador

status

Estado de la recepción del mensaje

c) MPI_Reduce(const void *sendbuf,
void *recvbuf,
int count,
MPI_Datatype datatype,
MPI_Op op,
int root,
MPI_Comm comm)

Recibe datos de todos los procesos, les aplica un operador y los envía a otro proceso.

Parámetros

sendbuf

Dirección del elemento que envía cada proceso

recvbuf

Dirección de la variable que recibirá el mensaje

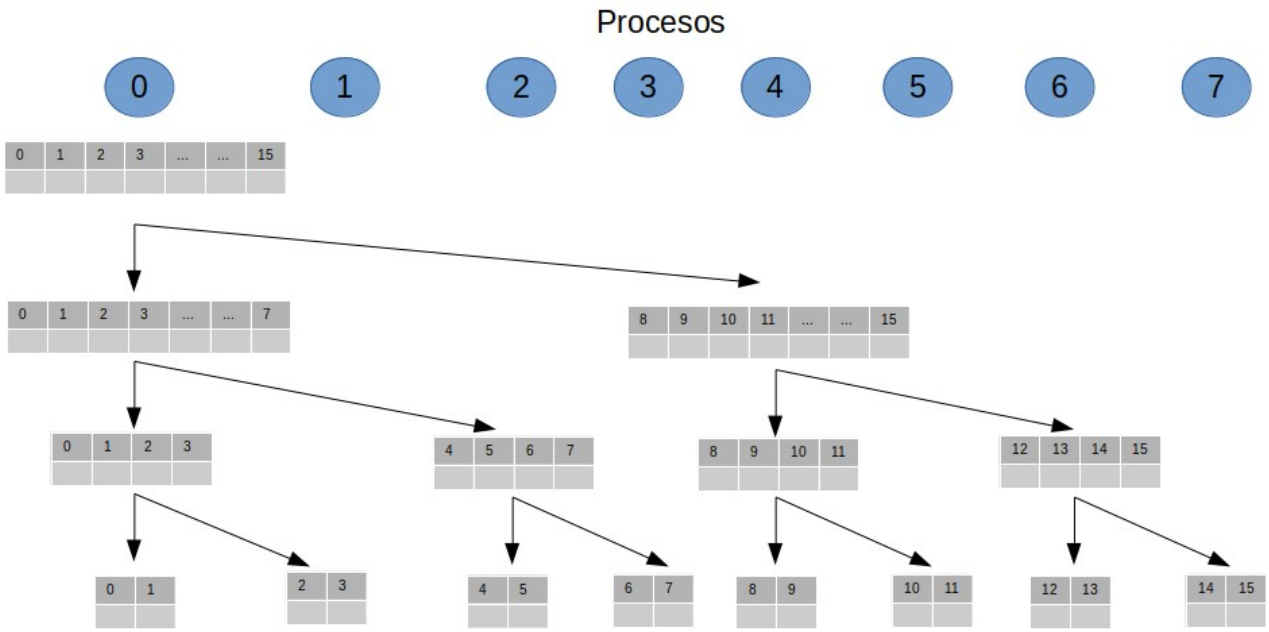
count
Número de elementos que se envían/reciben
datatype
Tipo de dato del mensaje
op
Operación a realizar
root
Rank del proceso que recibirá el resultado de la operación
comm
Comunicador

c) MPI_Allreduce(const void *sendbuf,
void *recvbuf,
int count,
MPI_Datatype datatype,
MPI_Op op,
int root,
MPI_Comm comm)
Recibe datos de todos los procesos, les aplica un operador y los envía todos los procesos.

Parámetros
sendbuf
Dirección del elemento que envía cada proceso
recvbuf
Dirección de la variable que recibirá el mensaje
count
Número de elementos que se envían/reciben
datatype
Tipo de dato del mensaje
op
Operación a realizar
comm
Comunicador

8.

a) MPI_Scatter



b) MPI_Gather