Memoria distribuida / Pasaje de mensajes

Práctico Nº 6

Objetivos

Este práctico es una aproximación práctica al diseño e implementación de soluciones, a fin y efecto que el estudiante sea capaz de resolver los problemas reales aplicando el paradigma de memoria distribuida a través de la programación con pasaje de mensajes.

Temas a tratar

Computación Paralela de Memoria Distribuida: Programación con Pasaje de Mensajes. Creación Dinámica de Procesos. Rutinas básicas de Pasaje de Mensajes. Concepto de Procesos Demonios. Grupos. Administración de grupos. Tipos de Comunicaciones. Comunicaciones Colectivas. Casos de estudio: MPI.

Metodología

Resolución de práctico de máquina. A través de distintos problemas simples, algunos resueltos en paralelo y otros a resolver, el alumno se aproximará en el primer caso a las facilidades provistas por la biblioteca respecto a la programación con pasaje de mensajes.

Conceptos Preliminares

En la actualidad, cada vez es más frecuente observar la existencia de sistemas altamente complejos que requieren mayor tiempo de computo. Para afrontar este tipo de problemas se hace uso de la computación de alto desempeño, en donde un único problema es dividido y resuelto de manera simultánea por un grupo de procesadores, donde el principal objetivo es mejorar la velocidad de procesamiento de la aplicación.

En estos casos, el modelo de memoria utilizado hace que la programación de aplicaciones paralelas para cada caso sea esencialmente diferente. En este práctico estudiaremos la biblioteca de pasaje de mensajes MPI, la cual idealmente es utilizada en aplicaciones de memoria distribuida donde los procesadores no pueden comunicarse a través de la memoria.

Ejercicio 1: Hello World.

a) El siguiente código resuelve un saludo en paralelo de todos los procesos que trabajan en paralelo

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

main(intargc, char **argv){
  int node;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &node);
  printf("Hello World fromNode %d\n",node);
  MPI_Finalize();
}
```

Ejecutar el programa para distinta cantidad de procesos. Analizar la salida de las ejecuciones y compararlas entre ellas.

b) Modificar el programa anterior de manera que los procesos paralelos envíen el mensaje de saludo *"Saludos desde el proceso #"* al Proceso 0, quien será el responsable de imprimirlo en la salida estándar. La salida tendrá las siguientes características:

Hola, soy el proceso 0 (hay "n" procesos) y recibo: (Hola desde el proceso #)
(Hola desde el proceso #)
…

c) Modificar el programa anterior, de manera tal que sea posible recibir e imprimir los mensajes enviados al Proceso 0 en orden ascendente. Es decir, para cuatro procesos la salida debería ser la siguiente:

Hola, soy el proceso 0 (hay 4 procesos) y recibo: (Hola desde el proceso 1)
(Hola desde el proceso 2)
(Hola desde el proceso 3)
(Hola desde el proceso 4)

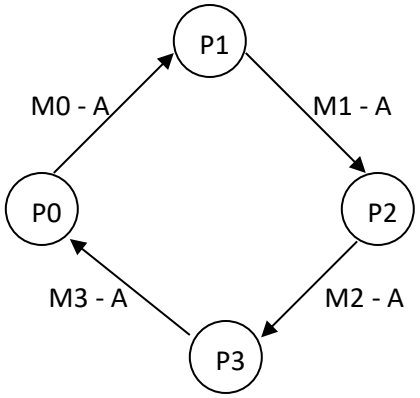
d) ¿Qué modificaciones debe hacer al programa anterior para que el proceso responsable de imprimir es otro proceso distinto del Proceso 0? Realícelas.

Ejercicio 2: Anillo.

Desarrollar un programa MPI denominado "anillo", en donde los procesos deben hacer circular un mensaje a través de un "anillo lógico". El programa deberá recibir como parámetro un entero n que indicará la cantidad de vueltas que el mensaje debe dar al anillo. Ejemplo con cuatro procesos:

El proceso 0 envía el M0 con el dato "A" al proceso 1
El proceso 1 envía el M1 con el dato "A" al proceso 2
El proceso 2 envía el M2 con el dato "A" al proceso 3
El proceso 3 envía el M3 con el dato "A" al proceso 0

Este proceso será repetido tantas veces como lo indique El parámetro "n".



Ejercicio 3: Multiplicación de matriz por un vector.
Analizar diferentes soluciones paralelas para resolver la multiplicación de una matriz por un vector. Analice soluciones paralelas considerando los distintos tipos de particionado de datos.
Ejemplo:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 \\ 31 & 32 & 33 & 34 & 35 & 36 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 91 \\ 217 \\ 343 \\ 469 \\ 595 \\ 721 \end{pmatrix}$$

a) Resuelva el problema planteado considerando que cada proceso realiza la multiplicación de una fila de la matriz por el vector completo. Considera en la solución el siguiente esquema de código:

   1. El proceso 0 generará la matriz y el vector.
   2. Las filas de la matriz se repartirán entre los procesos participantes.
   3. El vector se replicará en los procesos.
   4. Cada proceso realizará las operaciones que le corresponden.
   5. El vector resultado es reunido por el proceso 0.
   6. El proceso 0 mostrará el resultado.

Observación: El programa debe ser parametrizable, es decir que el mismo debe solicitar el tamaño de la matriz antes de generarla. Asuma el caso más simple donde cuenta con tantos procesos como filas tenga la matriz.

b) Modifique el código anterior pensando que existen más filas que procesos para tratarlas. Considere dos casos:
   - El número de filas de la matriz es múltiplo de la cantidad de procesos.
   - El número de filas de la matriz no es múltiplo de la cantidad de procesos.

Ejercicio 4: Generación dinámica de procesos.
Transcriba, ejecute y analice el ejemplo dado teoría para realizar la generación dinámica de procesos.

Ejercicio 5: Comunicaciones Colectivas.
Implemente cada comunicación colectiva descripta en teoría con comunicaciones punto a punto únicamente.

Ejercicio 6: Juego de la Vida.
El juego de la vida es un autómata celular diseñado por el matemático británico John Horton Conway en 1970. Es un juego de cero jugadores, lo que quiere decir que su evolución está determinada por el estado inicial y no necesita ninguna entrada de datos posterior.
El "tablero de juego" es una matriz de N x N formada por "células". Cada célula tiene 8 células vecinas, que son las que están próximas a ella, incluidas las diagonales.
Las células tienen dos estados: están "vivas" o "muertas" (1 representa una casilla con vida y 0 sin vida).
El estado de la matriz evoluciona a lo largo de unidades de tiempo discretas (se podría decir que por turnos). El estado de todas las células se tiene en cuenta para calcular el estado de las mismas al turno siguiente. Todas las células se actualizan simultáneamente.

Las transiciones dependen del número de células vecinas vivas según las siguientes reglas:

- Una célula muerta con exactamente 3 células vecinas vivas "nace" (al turno siguiente estará viva).
- Una célula viva con 2 ó 3 células vecinas vivas sigue viva, en otro caso muere o permanece muerta (por "soledad" o "superpoblación").

a) Analice el problema planteado. ¿Considera que es posible encontrar una solución paralela para el mismo? Justifique su respuesta.

b) Realice una implementación del juego de a vida de manera secuencial y otra en forma paralela utilizando MPI. Ambas implementaciones serán en pseudocódigo. Considere que el estado inicial es generado aleatoriamente por el programa. Se debe parametrizar el tamaño de la matriz y el número de ciclos que ejecutará el juego.