Cuestionario Teorías 8 a 10

Pregunta 1

Explique brevemente los 7 paradigmas de interacción entre procesos en programación distribuida vistos en teoría. En cada caso ejemplifique, indique qué tipo de comunicación por mensajes es más conveniente y que arquitectura de hardware se ajusta mejor. Justifique sus respuestas.

Respuesta

Master / worker:
Conjunto de workers que comparten una bolsa de tareas independientes. Sacan una tarea y la ejecutan, y en algunos casos generan nuevas tareas y las almacenan en la bolsa.

- **Comunicación: Pasaje de Mensajes Asíncrono**.
  - **Justificación:** El Master envía tareas y recopila resultados sin necesidad de bloquearse esperando cada resultado, optimizando el rendimiento.
- **Arquitectura: Memoria Distribuida (Cluster)**.
  - **Justificación:** Las tareas son independientes (grano grueso), lo que minimiza la comunicación de datos y escala eficientemente en un clúster.

Hearbeat:
Usa un esquema “divide & conquer” se distribuye la carga (datos) entre los workers; cada uno es responsable de actualizar una parte.
Los nuevos valores dependen de los mantenidos por los workers o sus vecinos inmediatos.
Cada “paso” debiera significar un progreso hacia la solución.
Se asocia más con el paradigma de **Paralelismo de Datos** sobre estructuras de malla o matriz, en lugar de un *divide & conquer* recursivo. Sin embargo, el núcleo de la descripción es funcional.

- **Arquitectura: Memoria Distribuida (Grid/Malla)**.
  - **Justificación:** El intercambio de datos es intenso pero altamente **localizado** (solo con vecinos), por lo que una arquitectura distribuida de alto rendimiento es la más adecuada

Pipeline:
Un pipeline es un arreglo lineal de procesos "filtro" que reciben datos de un puerto (canal) de entrada y entregan resultados por un canal de salida.
Estos procesos ("workers") pueden estar en procesadores que operan en paralelo.

Probes y echoes:
Prueba-eco se basa en el envío de mensajes ("Probe") de un nodo al sucesor, y la espera posterior del mensaje de respuesta ("Echo").
Los probes se envían en paralelo a todos los sucesores.
Los algoritmos de prueba-eco son particularmente interesantes cuando se trata de recorrer redes donde no hay (no se conoce) un número fijo de nodos activos (ejemplo: redes móviles).

- **Comunicación: Pasaje de Mensajes Asíncrono**.
  - **Justificación:** El objetivo es el envío **en paralelo** y no bloqueante de los Probes para recorrer la red rápidamente.}

Broadcast;

Permiten alcanzar una información global en una arquitectura distribuida. Sirven para toma de decisiones descentralizadas.
Se puede usar broadcast para diseminar información o para resolver problemas de sincronización distribuida. Ejemplo: semáforos distribuidos, la base es un ordenamiento total de eventos de comunicación mediante el uso de relojes lógicos.

- **Comunicación: Pasaje de Mensajes Asíncrono (Multicast/Broadcast)**.
  - **Justificación:** El emisor necesita diseminar la información rápidamente sin esperar confirmación de todos.
- **Arquitectura: Memoria Distribuida (Cluster/Red)**.
  - **Justificación:** Se requiere hardware que soporte eficientemente la funcionalidad de *broadcast* a nivel de red para evitar sobrecargar al emisor.

Token Passing:

Se basa en un tipo especial de mensaje (“token”) que puede usarse para otorgar un permiso (control) o recoger información global de la arquitectura distribuida.
Un ejemplo del primer tipo de algoritmos es el caso de tener que controlar exclusión mutua distribuida.

- **Comunicación: Pasaje de Mensajes Asíncrono**.
  - **Justificación:** El *token* se pasa de forma secuencial y no requiere una respuesta inmediata para la transferencia.
- **Arquitectura: Memoria Distribuida (Anillo Lógico)**.
  - **Justificación:** El modelo de paso de *token* asume una topología de anillo (lógico o físico) para garantizar el orden y la equidad (*fairness*) en el acceso a recursos.

Servidores replicados:

Los servidores manejan (mediante múltiples instancias) recursos compartidos tales como dispositivos o archivos.
Un server puede ser replicado cuando hay múltiples instancias de un recurso: cada server maneja una instancia.

Pregunta 2

Describa el paradigma “bag of tasks”. ¿Cuáles son las principales ventajas del mismo?

Respuesta

El concepto de bag of tasks usando variables compartidas supone que un conjunto de workers comparten una "bolsa" con tareas independientes. Los workers sacan una tarea de la bolsa, la ejecutan y posiblemente crean nuevas tareas que ponen en la bolsa.
En memoria distribuida el proceso master es quien contiene las tareas y los procesos workers le "piden" trabajo.

Ventajas:

- **Balanceo de Carga Dinámico Superior:** Es la mayor ventaja. El modelo Master-worker permite que los procesadores más rápidos o aquellos que terminan primero tomen más tareas de la "bolsa", lo que equilibra automáticamente la carga de trabajo en tiempo de ejecución.
- **Alta Tolerancia a Fallos:** Si un trabajador falla, la tarea simplemente se pierde. El Maestro puede reinsertar esa tarea de nuevo en la bolsa para que sea procesada por otro trabajador, garantizando que el cómputo se complete.
- **Baja Sobrecarga de Comunicación:** Dado que las tareas son independientes, no hay comunicación entre los procesos trabajadores, solo entre el Maestro y cada Trabajador (para pedir/entregar tareas y resultados). Esto minimiza el *overhead*.
- **Escalabilidad Sencilla:** El modelo se adapta muy bien al aumentar el número de procesadores (p). Añadir más procesadores solo significa más trabajadores tomando tareas, sin requerir cambios complejos en el algoritmo o la lógica de comunicación.

Pregunta 3

Suponga que existe una impresora para ser utilizada por N procesos distribuidos, Cada proceso continuamente trabaja y cada tanto debe usar la impresora. Realizar una solución distribuida con el paradigma de interacción Token Passing (suponga que los procesos están “conectados” en forma de anillo).

Respuesta

```
chan work[N](int);
chan token[N](int);
```

```
chan respC[N](int)
chan devolver[N](int)
chan admin(int);
process aux {
    int token = ...;
    int idAux;
    recieve admin(idAux);
    send work[idAux](token);
}
process coordinador [id: 1..N]{
    int sig = 0;
    int t;
    send admin(id);
    while (true) {
        recieve work[id](t)
        if(!empty(token[id])) {
            recieve token[id](sig);
            send respC[id](t);
            recieve devolver[id](t);
        }
        sig = (id mod N) + 1;
        send work[sig](t);
    }
}
process worker[id:1..N]{
    int t;
    while (true){
        // hace otra cosa
        send token[id](1)
        recieve respC[id](t);
        // usa impresora
        send devolver[id](t);
    }

}
```

Pregunta 4

Suponga n^2 procesos organizados en forma de grilla cuadrada. Cada proceso puede comunicarse solo con los vecinos izquierdo, derecho, de arriba y de abajo (los procesos de las esquinas tienen solo 2 vecinos, y los otros en los bordes de la grilla tienen 3 vecinos). Cada proceso tiene inicialmente un valor local v.
a) Escriba un algoritmo heartbeat que calcule el máximo y el mínimo de los n2 valores. Al terminar el programa, cada proceso debe conocer ambos valores. (Nota: no es necesario que el algoritmo esté optimizado).
b) Analice la solución desde el punto de vista del número de mensajes.
c) Puede realizar alguna mejora para reducir el número de mensajes?

d) Modifique la solución de a) para el caso en que los procesos pueden comunicarse también con sus vecinos en las diagonales

Respuesta

```
chan compañeros[N,N](max: int, min: int);
process worker[i:1..N][j:1..N]{
    bool vecinos [N,N]; // matriz inicializada con true en los que son
vecinos
    int auxMax, auxMin;
    int min,max;
    int v = ...;
    auxMax = v;
    auxMin = v;
    for(r=0; r < 2*N; r++){
        for[q = 1 to n] {
            for [k = 1 to n]{
                if (vecinos[q][k]){
                    send compañeros[q][k](auxMax, auxMin);
                }
            }
        }
        for[q = 1 to n] {
            for [k = 1 to n]{
                if (vecinos[q][k]){
                    recieve compañeros[i][j](max, min)
                    if(min < auxMin) auxMin = min;
                    if(max > auxMax) auxMax = max;
                }
            }
        }

    }
}
```

Pregunta 5

Indicar las características generales de Pthreads.

Respuesta

En principio estos mecanismos fueron heterogéneos y poco portables. A mediados de los 90 la org POSIX auspició el desarrollo de una biblioteca en C para multithreading (Pthreads). Pthreads es una biblioteca para programación paralela en **memoria compartida**, se pueden crear threads, asignarles atributos, darlos por terminados, identificarlos, etc.

Las secciones críticas se implementan en Pthreads utilizando mutex locks (bloqueo por exclusión mutua) por medio de variables mutex.

Una variable mutex tienen dos estados: locked (bloqueado) and unlocked (desbloqueado). En cualquier instante, sólo UN thread puede bloquear un mutex. Lock es una operación atómica.
Para entrar en la sección crítica un Thread debe lograr tener control del mutex (bloquearlo). Cuando un Thread sale de la SC debe desbloquear el mutex.

Pregunta 6

Explicar cómo se maneja la sincronización por exclusión mutua y por condición en Pthreads. Indicar la relación entre ambas. Explicar cómo se pueden simular los monitores en Pthreads.

Respuesta

Las secciones críticas se implementan en Pthreads utilizando mutex locks (bloqueo por exclusión mutua) por medio de variables mutex.
Una variable mutex tienen dos estados: locked (bloqueado) and unlocked (desbloqueado). En cualquier instante, sólo UN thread puede bloquear un mutex. Lock es una operación atómica
Pthreads soporta tres tipos de Mutexs (Locks): Normal, Recursive y Error check

- Un mutex con el atributo Normal no permite que un thread que lo tiene bloqueado vuelva a hacer un lock sobre él (deadlock).
- Un Mutex con el atributo Recursive SI permite que un thread que lo tiene bloqueado vuelva a hacer un lock sobre él. Simplemente incrementa una cuenta de control.
- Un Mutex con el atributo ErrorCheck responde con un reporte de error al intento de un segundo bloqueo por el mismo thread.
  Podemos utilizar variables de condición para que un thread se autobloquee hasta que se alcance un estado determinado del programa.
  Cada variable de condición estará asociada con un predicado. Cuando el predicado se convierte en verdadero (TRUE) la variable de condición da una señal para el/los threads que están esperando por el cambio de estado de la condición.

Pregunta 7

Explicar la clasifican de las comunicaciones punto a punto en las librerías de Pasaje de Mensajes en general: bloqueante y no bloqueante, con y sin buffering. Indicar a cual pertenece PMA y PMS.

Respuesta

- Send/Receive bloqueantes sin buffering.
  - el proceso espera a que el mensaje sea recibido o enviado por el proceso con el que me quiero comunicar
- Send/Receive bloqueantes con buffering.

    - El proceso espera a que el mensaje llegue o se envíe a un buffer intermedio y finaliza la comunicación ahi
- Send/Receive no bloqueantes sin buffering.
    - El proceso envía el mensaje o lo recibe, e inmediatamente continúa la ejecución
- Send/Receive no bloqueantes con buffering
    - El proceso envía el mensaje o lo recibe e inmediatamente continúa la ejecución

En PMA se utiliza send no bloqueante y recieve bloqueante. Y PM send y recieve bloqueante

Pregunta 8

Describa sintéticamente las características de sincronización y comunicación de MPI. Indicar las diferentes opciones de comunicación punto a punto que tiene MPI.

Respuesta

Un comunicador define el dominio de comunicación.

- Cada proceso puede pertenecer a muchos comunicadores.
- Existe un comunicador que incluye a todos los procesos de la aplicación MPI_COMM_WORLD.
- Son variables del tipo MPI_Comm → almacena información sobre que procesos pertenecen a él.

Protocolos para envios:

- Diferentes protocolos para send.
    - Send bloqueantes con buffering (Bsend).
    - Send bloqueantes sin buffering (Ssend).
    - Send no bloqueantes (Isend).
- Diferentes protocolos para Recv.
    - Recv bloqueantes (Recv).
    - Recv no bloqueantes (Irecv)

Pregunta 9

Explicar en qué consiste y por qué son tan eficientes las comunicaciones colectivas en MPI, mostrar
gráficamente un ejemplo de Broadcast entre 8 procesos. Cuales son y que hacen las principales opciones de
comunicación colectiva en MPI

Respuesta

Las comunicaciones colectivas consisten en un conjunto de funciones que permiten a un grupo de procesos realizar tareas de movimiento de datos (como enviar, recibir, recolectar, dispersar) y computación (como reducción) de forma coordinada.

Su **eficiencia** radica en que la implementación de la biblioteca MPI está **optimizada** para el hardware específico de la arquitectura. En lugar de que el programador tenga que implementar estas operaciones complejas con múltiples llamadas punto a punto (`MPI_Send` y `MPI_Recv`), las funciones colectivas de MPI aprovechan algoritmos internos altamente eficientes, como la estrategia de **árbol binario** para la difusión de datos. Esto reduce la complejidad del código para el desarrollador y permite que la transferencia de datos y la sincronización se realicen en un **tiempo mínimo**.

Pregunta 10

¿Qué relación encuentra entre el paralelismo recursivo y la estrategia de “dividir y conquistar”? ¿Cómo aplicaría este concepto a un problema de ordenación de un arreglo?

Respuesta

La relación es que la estrategia de **Dividir y Conquistar (DyC)** es el **modelo algorítmico** que facilita el **Paralelismo Recursivo**.

- DyC descompone un problema grande en subproblemas independientes.
- El **Paralelismo Recursivo** aprovecha esta estructura para ejecutar esos subproblemas **simultáneamente** en el *hardware* concurrente (Multicores, Multiprocesadores).

Aplicación a la Ordenación de un Arreglo

Para ordenar un arreglo (ejemplo de *Merge Sort*), la estrategia se aplica así:

1. **Dividir:** El arreglo se descompone recursivamente en sub-arreglos pequeños.
2. **Conquistar (Paralelismo):** Cada sub-arreglo se ordena por un **proceso o hilo distinto** de forma **concurrente**. El objetivo es el **menor tiempo para completar el trabajo**.
3. **Combinar (Cooperación):** Los sub-arreglos ordenados se fusionan. Esta fase requiere la **comunicación y sincronización** de los procesos para resolver la tarea común.

Pregunta 12

a) ¿Cuál es el objetivo de la programación paralela?
b) ¿Cuál es el significado de las métricas de speedup y eficiencia? ¿Cuáles son los rangos de valores en cada caso?
c) ¿En qué consiste la “ley de Amdahl”?
d) Suponga que la solución a un problema es paralelizada sobre p procesadores de dos maneras diferentes. En un caso, el speedup (S) está regido por la función S=p-1 y en el otro por la función S=p/2. ¿Cuál de las dos soluciones se comportará más eficientemente al crecer la cantidad de procesadores? Justifique claramente

Respuesta

Inciso A

El objetivo de la **Programación Paralela** es **reducir el tiempo de ejecución** de un programa.

Esto se logra a través de:

- La **ejecución concurrente** en múltiples procesadores.
- El aprovechamiento del *hardware* que utiliza **más procesadores por chip para mayor potencia de cómputo** (Multicores o Multiprocesadores).
- Alcanzando un **menor tiempo para completar el trabajo**.

Inciso B

Las métricas de **Speedup** y **Eficiencia** son esenciales en la Computación Paralela para evaluar la **ganancia de rendimiento** obtenida al utilizar múltiples procesadores en comparación con la ejecución secuencial.

El *Speedup* (Sp) mide la **mejora de velocidad** lograda por un algoritmo paralelo al ejecutarse en p procesadores, en relación con la mejor versión secuencial conocida del mismo algoritmo.

La **Eficiencia** (Ep) mide el **grado de utilización** de los procesadores disponibles. Indica la fracción de tiempo en que los p procesadores están realizando trabajo útil para el cómputo, y no esperando o comunicándose.

Inciso C

La **Ley de Amdahl** es una fórmula fundamental en la computación paralela que permite estimar el **máximo *speedup* (aceleración)** que se puede lograr al paralelizarr un programa o sistema.
La Ley de Amdahl consiste en establecer que la mejora de rendimiento que se puede conseguir con la paralelización está **limitada por la porción del código que debe ejecutarse de forma secuencial** (la parte no paralelizable).

Elementos Clave:

1. **Porción Secuencial (α):** Es el porcentaje del tiempo total de ejecución del algoritmo secuencial (Tsec) que no puede ser paralelizado (ej. inicialización, cálculos de dependencia de datos, entrada/salida).
2. **Porción Paralela (1−α):** Es el porcentaje del tiempo total de ejecución que sí puede distribuirse y ejecutarse en paralelo.
3. **Procesadores (p):** La cantidad de unidades de procesamiento.