CONCURRENTE MESA SEPTIEMBRE 2018

PREGUNTAS:

1) Defina el problema de sección crítica. Compare los algoritmos para resolver este problema (spin locks, Tie Breaker, Ticket, Bakery). Marque ventajas y desventajas de cada uno.

2) Defina el concepto de "sincronización barrier". ¿Cuál es su utilidad?

a) ¿Qué es una barrera simétrica?

b) ¿Qué es una "butterfly barrier"? Ejemplifica el funcionamiento para 16 procesos.

3) a) ¿En qué consiste la comunicación guardada y cuál es su utilidad? Ejemplifique.

b) Describa cómo es la ejecución de sentencias de alternativa e iteración que contienen comunicaciones guardadas.

c) Dado el siguiente bloque de código, indique para cada inciso qué valor quedó en Aux, o si el código quedó bloqueado.

Aux = -1;

....

if (A==0); P2?(Aux) > Aux = Aux +2;

(A==1); P3?(Aux) > Aux = Aux +5;

(B==0); P3?(Aux) > Aux = Aux +7;

endif;

….

i. Si el valor de A = 1 y B = 2 antes del if, y solo P2 envia el valor 6.

ii. Si el valor de A = 0 y B = 2 antes del if, y solo P2 envia el valor 8.

iii. Si el valor de A = 2 y B = 0 antes del if, y solo P3 envia el valor 6.

iv. Si el valor de A = 2 y B = 1 antes del if, y solo P3 envia el valor 9.

v. Si el valor de A = 1 y B = 0 antes del if, y solo P3 envia el valor 14.

vi. Si el valor de A = 0 y B = 0 antes del if, P3 envia el valor 9 y P2 el valor 5.

4) Describa brevemente en qué consisten los mecanismos de RPC y Rendezvous. ¿Para qué tipo de problemas son más adecuados?

¿Por qué es necesario proveer sincronización dentro de los módulos en RPC? ¿Cómo puede realizarse esta sincronización?

¿Qué elementos de la forma general de Rendezvous no se encuentran en el lenguaje ADA?

5) a) ¿Cuál es el objetivo de la programación paralela?

b) Defina las métricas de speedup y eficiencia.

¿Cuál es el significado de cada una de ellas (qué miden)?

c) 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.

d) Suponga que el tiempo de ejecución de un algoritmo secuencial es de 10000 unidades de tiempo, de las cuales 95% corresponden a código paralelizable. ¿Cuál es el límite en la mejora que puede obtenerse paralelizando el algoritmo?

6) Suponga que N procesos poseen inicialmente cada uno un valor. Se debe calcular la suma de todos los valores y al finalizar la computación todos deben conocer dicha suma.

a) Analice (desde el punto de vista del número de mensajes y la performance global) las soluciones posibles con memoria distribuida para arquitecturas en Estrella (centralizada), Anillo Circular, Totalmente Conectada y Árbol.

b) Implemente al menos dos de las soluciones mencionadas.

c) Numero de mensajes enviados por implementación y tiempo de ejecución

d) Repecto a lo obtenido en c) analice con N=4, N=8, N=16, N=32 y N=64. Analice la performance de las soluciones y compárelas

RESOLUCION:

1) Defina el problema de sección crítica. Compare los algoritmos para resolver este problema (spin locks, Tie Breaker, Ticket, Bakery). Marque ventajas y desventajas de cada uno.

En este problema, n procesos repetidamente ejecutan una sección crítica de código, luego una sección no crítica. La sección crítica está precedida por un protocolo de entrada y seguido de un protocolo de salida.

Spin locks

Se utiliza una variable booleana, llamada lock, que indica cuando un proceso está en su SC. Se utilizan instrucciones especiales como Test & Set (TS), Fetch & Add (FA) o Compare & Swap, presentes en casi todas las máquinas, que permiten implementar el protocolo de entrada a la sección critico.

Por ejemplo la instrucción TS toma una variable compartida lock como argumento y devuelve un resultado booleano. Accion= reserva de recurso, de forma atómica. 10

Se utilizan loops que no terminan hasta que lock es false, y por lo tanto TS setea initial a falso. Si ambos procesos están tratando de entrar a su SC, solo uno puede tener éxito en ser el primero en setear lock en true; por lo tanto, solo uno terminará su protocolo de entrada. Cuando se usa una variable de lockeo de este modo, se lo llama spin lock pues los procesos “dan vueltas” (spinning) mientras esperan que se libere el lock.

Funcionamiento:

Si el recurso esta libre, entonces TS(lock=false). Al setear lock = true, lo estamos reservando. Como devuelve el valor original de lock, false, al retornar se sale del while interno, ingresando a la sección critica del proceso.

Si el recurso está ocupado, entonces TS(lock=true). El seteo no produce ningún cambio. Al retornar con un valor true, el while interno sigue iterando.

Desventaja:

1) En multiprocesadores puede llevar a baja performance si varios procesos están compitiendo por el acceso a una SC. Esto es porque lock es una variable compartida y todo proceso demorado continuamente la referencia. Esto causa “memory contention”, lo que degrada la performance de las unidades de memoria y las redes de interconexión procesador-memoria.

2) El scheduling debe ser fuertemente fair para asegurar la eventual entrada

3) La instrucción TS escribe en lock cada vez que es ejecutada, aun cuando el valor de lock no cambie. Dado que la mayoría de los multiprocesadores emplean caches para reducir el tráfico hacia la memoria primaria, esto hace a TS mucho más cara que una instrucción que solo lee una variable compartida (al escribir un valor en un procesador, deben invalidarse o modificarse los caches de los otros procesadores). Aunque el overhead por invalidación de cache puede reducirse modificando el entry protocol para usar un protocolo test-and-test-and-set como sigue:

Tie-Breaker

Si hay n procesos, el protocolo de entrada en cada proceso consiste de un loop que itera a través de n-1 etapas. En cada etapa, determinamos si el proceso o un segundo proceso avanzan a la siguiente etapa. Si aseguramos que a lo sumo a un proceso a la vez se le permite ir a través de las n-1 etapas, entonces a lo sumo uno a la vez puede estar en su SC.

Sean in[1:n] y ultimo[1:n] arreglos enteros, donde n>1. El valor de in[i] indica cuál etapa está ejecutando p[i]; el valor de ultimo[j] indica cuál proceso fue el último en comenzar la etapa j. Estas variables son usadas de la siguiente manera: 11

Ventajas:

1) Solo requiere scheduling incondicionalmente fair para satisfacer la propiedad de eventual entrada.

2) No requiere instrucciones especiales del tipo Test-and-Set

Desventajas:

1) El algoritmo es mucho más complejo que la solución spin lock.

Ticket

Se basa en repartir tickets (números) y luego esperar turno. Este algoritmo es implementado por un repartidor de números y por un display que indica qué cliente está siendo servido.

Sean número y próximo enteros inicialmente 1, y sea turno[1:n] un arreglo de enteros, cada uno de los cuales es inicialmente 0. Para entrar a su SC, el proceso P[i] primero setea turn[i] al valor corriente de número y luego incrementa número. El proceso P[i] luego espera hasta que el valor de próximo es igual a su número.

Algunas máquinas tienen instrucciones que retornan el viejo valor de una variable y la incrementan o decrementan como una operación indivisible simple. Como ejemplo específico, Fetch-and-Add es una instrucción con el siguiente efecto:

Solucion Ticket con instrucción Fetch-and-add

Ventaja:

1) Es más fácil de comprender que el algoritmo Tie-Breaker.

Desventaja:

1) Si el algoritmo corre un tiempo largo, se puede alcanzar un overflow en las variables número y próximo. Aunque se puede resolver reseteando los contadores a un valor chico (digamos 1) cada vez que sean demasiado grandes. Si el valor más grande es al menos tan grande como n, entonces los valores de turno[i] se garantiza que son únicos.

2) Limitado porque es necesario utilizar instrucciones de máquina como Fetch-and-add. Aunque se podría resolver la sección crítica con un algoritmo spin locks o tie-breaker

Bakery

Es un algoritmo de tipo ticket, donde los procesos puede obtener el mismo número, pero hay una técnica de desempate en caso de que efectivamente ocurra eso.

Como en el algoritmo ticket, sea turno[1:n] un arreglo de enteros, cada uno de los cuales es inicialmente 0. Para entrar a su SC, el proceso p [i] primero setea turno[i] a uno más que el máximo de los otros valores de turno. Luego p[i] espera hasta que turno[i] sea el más chico de los valores no nulos de turno.

Ventaja:

1) No necesita de instrucciones de maquina especiales.

2) Defina el concepto de "sincronización barrier". ¿Cuál es su utilidad?

Como muchos problemas pueden ser resueltos con algoritmos iterativos paralelos en los que cada iteración depende de los resultados de la iteración previa es necesario proveer sincronización entre los procesos al final de cada iteración, para realizar esto se utilizan las barreras. Las barreras son un punto de encuentro de todos los procesos luego de cada iteración, aquí se demoran los procesos hasta que todos hayan llegado a dicho punto y cuando lo hagan recién pueden continuar su ejecución.

Es útil para poner un punto de encuentro entre los procesos de un programa concurrente a la espera de una determinada condición que le permita proseguir su curso normal. Dicha condición puede ser para mantener la integridad de los datos o bien para realizar una nueva iteración de los procesos.

- La sincronización barrier establece que el punto de demora al final de cada iteración es una barrera a la que deben llegar todos antes de permitirles pasar.

a) ¿Qué es una barrera simétrica?

Una barrera simétrica es un conjunto de barreras entre pares de procesos que utilizan sincronización barrier. En cada etapa los pares de procesos que interactúan van cambiando dependiendo a algún criterio establecido.

Otra:

Una barrera simétrica para n procesos se construye a partir de pares de barreras simples para dos procesos. Consiste en que cada proceso tiene un flag que setea cuando arriba a la barrera. Luego espera a que el otro proceso setee su flag y finalmente limpia la bandera del otro.

b) ¿Qué es una “butterfly barrier”? Ejemplifica el funcionamiento para 16 procesos.

Con butterfly barrier lo que se hace es en cada etapa diferente (son log2 n etapas) cada proceso sincroniza con uno distinto. Es decir, si s es la etapa cada proceso sincroniza con uno a distancia 2s-1. Así al final de las log2 n etapas cada proceso habrá sincronizado directa o indirectamente con el resto de los procesos.

3) a) ¿En qué consiste la comunicación guardada y cuál es su utilidad? Ejemplifique.

Con frecuencia un proceso se quiere comunicar con más de un proceso, quizás por distintos canales y no sabe el orden en el cual los otros procesos podrían querer comunicarse con él. Es decir, podría querer recibir información desde diferentes canales y no querer quedar bloqueado si en algún canal no hay mensajes. Para poder comunicarse por distintos canales se utilizan sentencias de comunicación guardadas.

Las sentencias de comunicación guardada soportan comunicación no determinística: B; C->S;

 B es una expresión booleana que puede omitirse y se asume true.

 B y C forman la guarda.

 La guarda tiene éxito si B es true y ejecutar C no causa demora.

 La guarda falla si B es falsa.

 La guarda se bloquea si B es true pero C no puede ejecutarse inmediatamente.

b) Describa cómo es la ejecución de sentencias de alternativa e iteración que contienen comunicaciones guardadas.

Sea la sentencia de alternativa con comunicación guardada de la forma:

if B1; comunicacion1 -> S1

□ B2; comunicacion2 -> S2

fi

Primero, se evalúan las expresiones booleanas, Bi y la sentencia de comunicación.

•Si todas las guardas fallan (una guarda falla si B es false), el if termina sin efecto.

•Si al menos una guarda tiene éxito, se elige una de ellas (no determinísticamente).

•Si algunas guardas se bloquean (B es true pero la ejecución de la sentencia de comunicación no puede ejecutarse inmediatamente), se espera hasta que alguna de ellas tenga éxito.

Segundo, luego de elegir una guarda exitosa se ejecuta la sentencia de comunicación asociada.

Tercero, y último, se ejecutan las sentencias S relacionadas.

La ejecución de la iteración es similar realizando los pasos anteriores hasta que todas las guardas fallen.

c) Dado el siguiente bloque de código, indique para cada inciso qué valor quedó en Aux, o si el código quedó bloqueado.

Aux = -1;

....

if (A==0); P2?(Aux) > Aux = Aux +2;

(A==1); P3?(Aux) > Aux = Aux +5;

(B==0); P3?(Aux) > Aux = Aux +7;

endif;

…

i. Si el valor de A = 1 y B = 2 antes del if, y solo P2 envia el valor 6.

Se queda bloqueado en la guarda A==1 ya que la condición es la única verdadera pero P3 no se ejecutó.

ii. Si el valor de A = 0 y B = 2 antes del if, y solo P2 envia el valor 8.

Entra en la primer guarda (única con condición verdadera) y se ejecuta P2. Luego el valor de aux será 10.

iii. Si el valor de A = 2 y B = 0 antes del if, y solo P3 envia el valor 6.

Entra en la tercer guarda (única con condición verdadera) y se ejecuta P3. Luego el valor de aux será 13.

iv. Si el valor de A = 2 y B = 1 antes del if, y solo P3 envia el valor 9.

El if no tiene efecto ya que ninguna condición es verdadera. Aux mantiene el valor (-1).

v. Si el valor de A = 1 y B = 0 antes del if, y solo P3 envia el valor 14.

Puede entrar en la segunda o tercer guarda (elección no determinista) ya que ambas son verdaderas y la sentencia de comunicación puede ejecutarse inmediatamente (no hay bloqueo).

Caso A==1: el valor de aux será 19.

Caso B==0: el valor de aux será 21.

vi. Si el valor de A = 0 y B = 0 antes del if, P3 envia el valor 9 y P2 el valor 5.

Tanto la condición de la primer guarda y la condición de la última son verdaderas; además P3 y P2 se ejecutaron, por lo tanto no hay bloqueo. Como la elección es no determinista, pueden suceder dos casos:

Caso A==0: el valor de aux será 7.

Caso B==0: el valor de aux será 16.

4) Describa brevemente en qué consisten los mecanismos de RPC y Rendezvous. ¿Para qué tipo de problemas son más adecuados?

RPC sólo provee un mecanismo de comunicación y la sincronización debe ser implementada por el programador utilizando algún método adicional. En cambio, Rendezvous es tanto un mecanismo de comunicación como de sincronización.

En RPC la comunicación se realiza mediante la ejecución de un CALL a un procedimiento, este llamado crea un nuevo proceso para ejecutar lo solicitado. El llamador se demora hasta que el proceso ejecute la operación requerida y le devuelva los resultados. En Rendezvous la diferencia con RPC está en cómo son atendidos los pedidos: el CALL es atendido por un proceso ya existente, no por uno nuevo. Es decir, con RPC el proceso que debe atender el pedido no se encuentra activo sino que se crea para responder al llamado y luego termina; en cambio, con Rendezvous el proceso que debe atender los requerimientos se encuentra continuamente activo. Esto hace que RPC permita servir varios pedidos al mismo tiempo y que con Rendezvous sólo puedan ser atendidos de a uno por vez (se precisaría un pool de procesos aceptando el mismo CALL para poder atender a más de un pedido simultáneamente).

Estos mecanismos son más adecuados para problemas con interacción del tipo cliente/servidor donde la comunicación entre ellos debe ser bidireccional y sincrónica, el cliente solicita un servicio y el servidor le responde con lo solicitado ya que el cliente no debe realizar ninguna otra tarea hasta no obtener una respuesta del servidor

¿Por qué es necesario proveer sincronización dentro de los módulos en RPC? ¿Cómo puede realizarse esta sincronización?

Por sí mismo, RPC es puramente un mecanismo de comunicación. Aunque un proceso llamador y su server sincronizan, el único rol del server es actuar en nombre del llamador. Conceptualmente, es como si el proceso llamador mismo estuviera ejecutando el llamado, y así la sincronización entre el llamador y el server es implícita.

También necesitamos alguna manera para que los procesos en un módulo sincronicen con cada uno de los otros. Estos incluyen tanto a los procesos server que están ejecutando llamados remotos como otros procesos declarados en el módulo. Como es habitual, esto comprende dos clases de sincronización: exclusión mutua y sincronización por condición. Hay dos aproximaciones para proveer sincronización en módulos, dependiendo de si los procesos en el mismo módulo ejecutan con exclusión mutua o ejecutan concurrentemente. Si ejecutan con exclusión (es decir, a lo sumo hay uno activo por vez) entonces las variables compartidas son protegidas automáticamente contra acceso concurrente. Sin embargo, los procesos necesitan alguna manera de programar sincronización por condición. Para esto podríamos usar automatic signalig (await B) o variables condición.

Si los procesos en un módulo pueden ejecutar concurrentemente (al menos conceptualmente), necesitamos mecanismos para programar tanto exclusión mutua como sincronización por condición. En este caso, ada módulo es en sí mismo un programa concurrente, de modo que podríamos usar cualquiera de los métodos descriptos anteriormente. Por ejemplo, podríamos usar semáforos dentro de los módulos, monitores locales, o podríamos usar Rendezvous (o usar MP).

¿Qué elementos de la forma general de Rendezvous no se encuentran en el lenguaje ADA?

Rendezvous provee la posibilidad de asociar sentencias de scheduling y de poder usar los parámetros formales de la operación tanto en las sentencias de sincronización como en las sentencias de scheduling. ADA no provee estas posibilidades.

ADA no provee la expresión de Scheduling "ei" la cual se usa para alterar el orden de servicio de invocaciones por default en las guardas.

5) **a) ¿Cuál es el objetivo de la programación paralela?**

El objetivo principal es reducir el tiempo de ejecución, o resolver problemas más grandes o con mayor precisión en el mismo tiempo, usando una arquitectura multiprocesador en la que se pueda distribuir la tarea global en tareas que puedan ejecutarse en distintos procesadores (simultáneamente).

b) Defina las métricas de speedup y eficiencia.

¿Cuál es el significado de cada una de ellas (qué miden)?

Ambas son métricas asociadas al procesamiento paralelo.

El speedup es una medida de la mejora de rendimiento (performance) de una aplicación al aumentar la cantidad de procesadores, comparado con el rendimiento al utilizar un solo procesador.

Speedup⇒ S = Ts/ Tp

S es el cociente entre el tiempo de ejecución secuencial del algoritmo secuencial conocido más rápido (Ts) y el tiempo de ejecución paralelo del algoritmo elegido (Tp).

El rango de valores de S va desde 0 a p, siendo p el número de procesadores.

La eficiencia es una medida relativa que permite la comparación de desempeño en diferentes entornos de computación paralela.

Eficiencia⇒ E = S/P

Es el cociente entre speedup y número de procesadores. El valor está entre 0 y 1, dependiendo de la efectividad en el uso de los procesadores. Cuando es 1 corresponde al speedup perfecto

c) 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.

Más eficientemente (eficiencia): E = S/p

Entonces:

E(p-1) = (p-1) / p= 1 - (1/p)

E(p/2) = (p/2) / p= 1/p

Sabemos que un E ideal es 1.

Como vemos, para S=p-1 conforme p crece, E se acerca a 1. Para S=p/2 en cambio, conforme p crece, E tiende a 0.

Por ello, p-1 es la solución más eficiente.

d) Suponga que el tiempo de ejecución de un algoritmo secuencial es de 10000 unidades de tiempo, de las cuales 95% corresponden a código paralelizable. ¿Cuál es el límite en la mejora que puede obtenerse paralelizando el algoritmo?

El límite de mejora sería utilizando 9500 procesadores los cuales ejecutan una unidad de tiempo cada procesador paralelamente obteniendo un tiempo paralelo de 501 unidades de tiempo.

Explicacion: 500 unidades de tiempo que no eran paralelizables más esa unidad de tiempo que tardaron los 9500 procesadores en ejecutar el 95% del código paralelizable.

El speedup mide la mejora del tiempo obtenida con un algoritmo paralelo comparándola con el secuencial.

Speedup = TiempoSecuencial / TiempoParalelo = 10000 / 501 = 19,9 ~ 20

Aunque se ejecute en más procesadores, el mayor speedup alcanzable es este, cumpliéndose así la *Ley de Amdahl* diciendo para todo problema hay un límite de paralelización, dependiendo el mismo no de la cantidad de procesadores, sino de la cantidad de código secuencial.

6) **Suponga que N procesos poseen inicialmente cada uno un valor. Se debe calcular la suma de todos los valores y al finalizar la computación todos deben conocer dicha suma.**

a) Analice (desde el punto de vista del número de mensajes y la performance global) las soluciones posibles con memoria distribuida para arquitecturas en Estrella (centralizada), Anillo Circular, Totalmente Conectada y Árbol.

b) Implemente al menos dos de las soluciones mencionadas.

Arquitectura en estrella (centralizada)

En este tipo de arquitectura todos los procesos (workers) envían su valor local al procesador central (coordinador), este suma los N datos y reenvía la información de la suma al resto de los procesos. Por lo tanto se ejecutan 2(n-1) mensajes. Si el procesador central dispone de una primitiva broadcast se reduce a N mensajes.

En cuanto a la performance global, los mensajes al coordinador se envían casi al mismo tiempo. Estos se quedaran esperando hasta que el coordinador termine de computar la suma y envíe el resultado a todos.

(Promedio: el coordinador hace: send resultado[i] (sum/n);)

Anillo circular

Se tiene un anillo donde P[i] recibe mensajes de P[i-1] y envía mensajes a P[i+1]. P[n-1] tiene como sucesor a P[0]. El primer proceso (P[0]) envía su valor local ya que es lo único que conoce.

Este esquema consta de dos etapas:

1) Cada proceso recibe un valor y lo suma con su valor local, transmitiendo la suma local a su sucesor.

2) Todos reciben la suma global.

P[0] debe ser algo diferente para poder "arrancar" el procesamiento: debe envía su valor local ya que es lo único que conoce. Se requerirán 2(n-1) mensajes.

A diferencia de la solución centralizada, esta reduce los requerimientos de memoria por proceso pero tardará más en ejecutarse, por más que el número de mensajes requeridos sea el mismo. Esto se debe a que cada proceso debe esperar un valor para computar una suma parcial y luego enviársela al siguiente proceso; es decir, un proceso trabaja por vez, se pierde el paralelismo.

(Promedio: Process P[0] cambia el último send: send valor[1] (suma/n);)

Totalmente conectada (simétrica)

Todos los procesos ejecutan el mismo algoritmo. Existe un canal entre cada par de procesos.

Cada uno transmite su dato local v a los n-1 restantes. Luego recibe y procesa los n-1 datos que le faltan, de modo que en paralelo toda la arquitectura está calculando la suma total y tiene acceso a los N datos.

Se ejecutan n(n-1) mensajes. Si se dispone de una primitiva de broadcast, serán N mensajes. Es la solución más corta y sencilla de programar, pero utiliza el mayor número de mensajes si no hay broadcast.

Árbol

Se tiene una red de procesadores (nodos) conectados por canales de comunicación bidireccionales. Cada nodo se comunica directamente con sus vecinos. Si un nodo quiere enviar un mensaje a toda la red, debería construir un árbol de expansión de la misma, poniéndose a él mismo como raíz.

El nodo raíz envía un mensaje por broadcast a todos los hijos, junto con el árbol construido. Cada nodo examina el árbol recibido para determinar los hijos a los cuales deben reenviar el mensaje, y así sucesivamente.

Se envían n-1 mensajes, uno por cada padre/hijo del árbol.