The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

En multiprocesadores de memoria compartida, cada procesador puede direccionar memoria directamente la cual puede ser tambien direccionada por todos los otros procesadores. Este acceso uniforme requiere de algún método para asegurar exclusión mutua: la ejecución lógicamente atómica de las operaciones (secciones críticas) en una estructura de datos compartidos. La consistencia de la estructura de datos se garantiza al serializar las operaciones realizadas en ella. Dado que la exclusión mutua a partir de sólo software es costosa, prácticamente todos los multiprocesadores de memoria compartida proveen alguna forma de soporte de hardware para realizar exclusión mutua para acceder a estructuras de datos compartidas. Este soporte usualmente consiste en instrucciones que leen y escriben atómicamente a una locación de memoria. Si las operaciones en la estructura de datos compartida son suficientemente simples, se pueden encapsular dentro de una sola instrucción atómica. La exclusión mutua también puede ser garantizada por hardware. Si un número de procesadores intentan actualizar simultáneamente la misma locación, cada uno espera su turno sin volver el control del vuelta al software. Un *lock*(bloqueo) se necesario para las secciones críticas que toman más de una instrucción. Las instrucciones atómicas son usadas para arbitrar entre intentos simultáneos para adquirir el *lock*, pero si el *lock* está ocupado, la espera se hace en software. Cuando un *lock* esta ocupado, el proceso de espera puede ya sea bloquear o rotar (*spin* - espera activa) hasta que el *lock* sea liberado. *Spin-waiting* gasta mucho ciclos de procesador, es útil si la sección crítica es pequeña en donde el tiempo de espera es menor que el costo del bloqueo y reanudación del proceso, o si no otro trabajo está disponible.

Range of Multiprocessor Architectures Considered. Los *Spinning processors* pueden retardar a los procesadores activos/ocupados en cualquier multiprocesador en donde la espera activa consuma ancho de banda en la comunicación. El rendimiento de la espera activa cambia a lo largo de varias dimensiones de arquitectura: cómo los procesadores esta conectados a la memoria, ya sea que cada procesador tenga un cache coherente privado gestionado por el hardware o no, y si lo tiene, el protocolo de coherencia. *Common Hardware Support for Mutual Exclusion*: La mayoría de las arquitecturas soportan exclusión mutua al proveer instrucciones que leen, modifican y escriben en memoria de forma atómica. *Multistage networks* conectan múltiples procesadores con múltiples módulas de memoria. Solicitudes a memoria son reenviadas a través de una seria de *switches* al módulo de memoria correcto. Cuando un valor se lee de memoria como parte de una instrucción atómica, cualquier copia(s) "cacheada" de la localización debe ser invalidada y accesos subsecuentes a ese módulo de memoria tienen que ser retrasadas mientras en nuevo valor es calculado. En multiprocesadores de un sólo bus, el bus puede ser usado para arbitrar entre instrucciones atómicas simultáneas. Antes de iniciar una instrucción atómica, el procesador adquiere el bus y levanta una línea[1]. Esta línea es retenida mientras el nuevo valor de memoria es obtenido para evitar que otras peticiones atómicas sean iniciadas, pero el bus puede ser liberado para permitir otras peticiones normales de memoria proceder. En sistemas que no "cachean" datos compartidos, la transacción de bus usada para adquirir el *atomic bus line* puede ser superponer(*overlapped*) con la petición de lectura para los datos o con la señal de invalidación de otras copias de cache. Invalidación basada en coherencia *Write-back* evita una transacción extra de bus para escribir los datos. El

[1]La línea de bus atómica - atomic bus line

nuevo valor se almacena temporalmente en el cache del procesador. Cuando otro proceso necesita el valor, se obtiene el valor al mismo tiempo que se invalida la primer copia del procesador. En *Distributed-write write-back coherence*, la lectura inicial usualmente no se necesita. Esto porque copias en todos los "caches" son actualizadas en lugar de invalidadas cuando un procesador cambia un valor de memoria, el bloque de cache necesitado por la instrucción atómica se encontrará en el cache a menudo.

The Performance of the Simple Approaches to Spin-Waiting. El rendimiento cuando hay contención en el *lock* depende de la minimización del ancho de banda de la comunicación usada por los *spinning processors*, dado que esto puede causar que los procesadores hagan trabajo útil de forma lenta. El retraso que hay cuando un *lock* es liberado y luego re-adquirido por un *spinning processor* también debe ser minimizado, dado que ningún procesador estás ejecutando la sección crítica durante este tiempo. Inconveniente: entre más frecuentemente un procesador intenta adquirir un *lock*, más rápido será adquirido pero también los otros procesadores estarán más interrumpidos. Latencia, el tiempo que le toma a un procesador en adquirir un *lock* en la ausencia de contención es también un criterio importante en aplicaciones con bloqueos frecuentes. Un rendimiento pobre en contención puede evitar que una aplicación alcance su rendimiento máximo. Podría ser que no siempre sea posible modificar un programa para usar un número óptimo de procesadores. Un sistema operativo, tiene muy poco control sobre la tasa en la que los usuarios hacen llamadas al sistema. Durante cargas intensivas, *locks* que normalmente no son un problema pueden convertirse en fuente de contención. *Spin on Test-and-Set*[2]. El algoritmo más simple de espera activa es que cada procesador ejecute repetidamente la instrucción *test-and-set* hasta que tenga éxito en la adquisición. El rendimiento cae conforme el número de *spinning processors* aumenta. Causas: (1) con el fin de liberar un *lock*, el poseedor del *lock* debe lidiar con *spinning processors* por acceso exclusivo a la locación del *lock*. (2) En arquitecturas en donde las peticiones de *test-and-set* comparten el mismo bus como referencias de memoria normales, las peticiones de *spinning processors* pueden retrasar acessos a otras locaciones por el possedor del *lock* u otros procesos activos. (3) En arquitecturas *multistage network* la espera activa puede causar un retraso en los accesos a los módulos de memoria que contienen la localización del *lock* así como otros módulos. *Spin on Read (Test-and-Test-and-Set).* Los *spinning processors* hacen una lectura cíclica del valor del *lock* y solamente cuando el *lock* está libre, ejecutan una instrucción *test-and-set*; esto elimina la necesidad de ejecutar *test-and-set* repetitivamente mientras que el *lock* está retenido. Mientras que el *lock* está ocupado, el *spinning* se hace en el cache sin consumir ciclos de red o bus. Cuando el *lock* es liberado, cada copia es actualizada a un nuevo valor o invalidada, causando un *read miss* el cual obtiene el nuevo valor. El procesador que espera ve el cambio en el estado y realiza un *test-and-set*. Si alguien adquiere el *lock* en el interim, el procesador puede reanudar el *spinning* en su cache. *Reasons for the Poor Performance os Spin on Read:* razones por las cuáles el rendimiento de *spinning* en memoria puede ser peor que el esperado: (1) Existe una separación entre la detección que el *lock* ha sido liberado e intentar adquirirlo por medio de la instrucción *test-and-set*. (2) Copias de cache del *lock* son invalidadas por la instrucción *test-and-set* incluso si el valor no es cambiado. (3) Coherencia de cache basada en invalidación requiere de $O(P)$ ciclos de bus o red para emitir un valor a P procesadores en espera. *Measurement Results:* el rendimiento se degrada conforme los procesadores giran(*spin*) en *test-and-set*. Conforme la sección crítica se convierte en un cuello de botella, el número promedio de *spinning processors* aumenta, lo que hace significativamente más lento al procesador el ejecutar la sección crítica. Como resultado el rendimiento tope nunca es alcanzado.

New Software Alternatives. *Delay Alternatives:* (1) *Delay after Spinning Processor Notices Lock has been Released:* se puede reducir el número *test-and-set* fallidos en una lectura al insertar un retraso entre el momento en el que el procesador lee que el *lock* fue liberado y cuando se compromete a intentar el *test-and-set*. Si otro procesador adquiere el *lock* durante este retraso, entonces el procesador puede reanundar el *spinning*, sino entonces el procesador puede intentar el *test-and-set* con mayor probabilidad que el *lock* será adquirido. De esta forma el numero de *test-and-set* fallidos y por lo tanto invalidaciones, pueden ser reducidas. A cada procesador se le puede asignar estáticamente una cantidad de tiempo de retraso que va de 0 a $P-1$ en donde P es el número de procesadores. El *spinning processor* con el menor retraso asignado revisa el *lock*, ve si está libre y lo adquiere. Procesadores con tiempos de retraso más largos caducan, ven que el *lock* está ocupado y reanundan el *spinning*. La asignación estática de retrasos se puede asegurar de que al menos un procesador caduque en cualquier instante. (2) *Delay Between Each Memory Reference:* un enfoque alternativo es reducir el costo de la espera activa es insertar un retraso entre cada referencia

de memoria. Esto puede ser usado en arquitecturas sin coherencia de cache o con invalidación basada en coherencia para limitar el ancho de banda de la comunicación que consumen los *spinning processors*. *Queing in Shared Memory:* Usar la memoria compartida para almacenar el estado de la actividad de los *spinning procesors* podría resultar más costoso puesto que se necesitarían dos instrucciones atómicas extra por sección crítica. Se propone un nuevo método de cola de espera (queueing) para procesadores en espera activa que solamente requiere una instrucción atómica por cada ejecución de una sección crítica. Cada procesador que arriba realiza un *read-and-increment* atómico para obtener un número único de secuencia. Cuando un procesador finaliza el *lock*, le asigna el procesador el número de secuencia siguien más alto; ahora el procesador es dueño del *lock*. Dado que los procesadores están secuenciados, no se necesitan instrucciones *read-modify-write* atómicas para pasarle el control al *lock*.

Hardware Solutions. Tal y como en las soluciones de software, las soluciones de hardware también tienen sus pros y contras. Por ejemplo, el mejor mecanismo de coherencia de cache para *spin locks* puede no ser el mejor para referencias de memoria normales. *Multistage Interconnection Network Multiprocessors:* combinación de redes, por medio de acceso paralelo a un localización de memoria, puede mejorar el rendimiento del *spinning* directo en *test-and-set*. Peticiones al mismo lugar que arriban al mismo *switch* de red se combinan y reenvían como una sola petición; el resultado es el mismo como si dos peticiones fueran hechas de forma secuencial al módulo de memoria. Colas de espera de hardware en el módulo de memoria, como en colas de espera en software, pueden eliminar el *polling* a través de la red – también pueden hacer más rápido el paso de control de un *lock*. Para esto, los procesadores deben emitir instrucciones "enter" y "exit" explícitos en secciones críticas al módulo de memoria, el cual podría mantener colas de los procesadores esperando para cada *lock*. Cuando una solicitud "enter" del procesador retorna, se tiene el *lock*; no es necesario hacer *polling* en la red. En sistemas con coherencia de cache y colas de espera de software, el procesador que libera el *lock* le notifica al siguiente por medio de la escritura de esta bandera (*flag*). Una invalidación seguida de un *read miss* es necesaria antes de que el *spinning processor* puede iniciar la ejecución de la sección crítica. Al manejar de forma especial las solicitudes de sección crítica, la cola de espera en hardware elimina una viaje(ida y vuelta) en la red para pasar el control del *lock*. La latencia de *lock* es probable que sea mejor en hardware que con colas de espera en software. Las colas de espera en hardware incrementan la complejidad en el módulo de memoria, reducen el número de instrucciones necesarias para adquirir el *lock*. *Single Bus Multiprocessors:* (1) *Read Broadcast:* puede eliminar solicitudes duplicadas de *read miss*. El controlador de cache de cada procesador monitorea el bus; si se produce una lectura correspondiente a un bloque inválido en su cache se toman los datos del bus y se establece el bloque como válido. Asi que, cuando las copias de cache de los *spinning processors* están invalidadas, la primera lectura va a llenar todos los caches. (2) Por medio de un manejo especial de las solicitudes *test-and-set* en el cache y en el bus de los controladores, se puede eliminar la necesidad de que *test-and-set* fallidos usen el bus. De esta forma, un procesador puede rotar(*spin*) en *test-and-set*, adquirir el *lock* lo más rápidamente cuando está libre, sin consumir ancho de banda del bus cuando el *lock* está ocupado. Al proveer este manejo especial de instrucciones *test-and-set*, no se incrementa el bus o los ciclos de cache, el rendimiento es mejor que con software *backoff*[3] o colas de espera.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

Este artículo examina la siguiente pregunta: dado un hardware que soporta instrucciones atómicas, existen algoritmos eficientes para software de espera activa (*sping waiting*), o hay más tipos más complejo de soporte de hardware necesarios para el rendimiento?

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

La mayoría de arquitecturas de multiprocesadores proveen soporte de hardware para excluir mutuamente accesos a estruturas de datos compartidas. Este soporte usualmente consiste de instrucciones que leen y luego escriben en una localización de memoria de forma atómica. Estas instrucciones atómicas son utilizadas para manipular bloqueos (*locks*): cuando un procesador está accediendo una estructura de datos, su *lock* esta ocupado y otros procesadores que necesiten acceso tienen que esperar. Para secciones críticas pequeñas, esperar activamente para que un *lock* sea

[3]Reducir el uso de otros otros recursos de sistemas.

liberado es más eficiente que renunciar al procesador para hacer otro trabajo. Desafortunadamente, la espera activa puede retrasar otros procesadores al consumir ancho de banda de comunicación.

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

Muchos multiprocesadores de memoria compartida han sido diseñados en el pasado reciente: el *Sequent Simmetry*, Alliant FX, BBN Butterfly están entre los que han tenido mayor éxito comercial. En investigación: DEC SRC Firefly, Illinois Cedar, IBM RP3 y el Wisconsin Multicube. Todos los procesadores anteriores soportan instrucciones atómicas, aunque algunos, como el Multicube también provee otros mecanismos.

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

Se propone un nuevo método para poner en una cola de espera explítica los *spinning processors*. Conforme los procesadores arriban al *lock*, cada uno de ellos adquiere un número único de secuencia especificando el orden en que ellos ejecutarán la sección crítica. Cuado el bloqueo es liberado, el control puede ser pasado directamente al siguiente procesador en la línea sin mayor sincronización y con efectos mínimos en otros procesadores. También se examina el rendimiento de varias soluciones de hardware. Se propone una adición a protocolos de *snoopy cache* para explotar la semántica de las solicitudes *spin lock* para obtener mejor rendimiento.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?