Implementação de um esquema de invalidação mista na linguagem CMTJava

Rafael de Leão Bandeira[1], **André Rauber Du Bois**[1], **Maurício Lima Pilla**[1]

[1]Programa de Pós-Graduação em Computação
Centro de Desenvolvimento Tecnológico – UFPEL

{rdlbandeira, dubois, pilla}@inf.ufpel.edu.br

1. Introdução

A computação paralela permite um grande ganho no desempenho dos programas, dividindo-os em partes discretas resolvidas concorrentemente usando múltiplos recursos computacionais. Essa abordagem trouxe uma série de benefícios em relação a programação sequencial, permitindo resolver problemas cada vez mais complexos e em tempo cada vez menor. Apesar dos seus benefícios, esse paradigma aumenta a complexidade no desenvolvimento dos programas, pois é necessário levar em conta vários aspectos inexistentes nos algoritmos sequenciais, como por exemplo, garantir a exclusão mútua das tarefas executadas paralelamente que poderiam ter acessos com condições de corrida. O mecanismo mais usado para o controle do acesso a seções críticas é o *lock*. Contudo, o uso de técnicas baseadas em *locks* tem muitas desvantagens associadas como baixa escalabilidade, dificuldade de composição de código e a possibilidade de ocorrência de *deadlocks*.

As memórias transacionais são um modelo de programação alternativo aos mecanismos baseados em exclusão mútua, que fornecem uma abstração de mais alto nível para a escrita de programas concorrentes, deixando o programador concentrado no algoritmo, ao invés de na sincronização da execução. Além disso, memórias transacionais fornecem uma melhor relação entre escalabilidade e esforço de implementação, embora algoritmos usando primitivas de baixo nível pra sincronização obtenham melhor desempenho, ao custo de uma grande complexidade de desenvolvimento.

2. Estado-da-arte

CMTJava [Du Bois and Echevarria 2009] é uma extensão de Java para programação de memórias transacionais. O primeiro sistema transacional desenvolvido para a linguagem foi baseado no sistema transacional da linguagem STM Haskell [Harris et al. 2005]. Esse sistema possui implementação simples pois as transações precisam adquirir um bloqueio global sempre que vão validar o log da transação ou efetivar a mesma, limitando o paralelismo. Já o segundo sistema transacional foi desenvolvido baseado no algoritmo TL2 (*Transactional Locking II*) [Dice et al. 2006]. O algoritmo TL2 combina o uso de um relógio global, para controle de versão dos dados, com uma técnica de aquisição do bloqueio dos objetos em tempo de efetivação. Essa implementação não usa um bloqueio global, aumentando o paralelismo e consequentemente obtendo melhor desempenho que a implementação anterior.

3. Objetivos do trabalho

Este trabalho propõe o desenvolvimento de um novo sistema transacional para a CMTJava, com o objetivo de melhorar o desempenho dos programas na linguagem.

O algoritmo TL2 é bastante utilizado nas implementações de memória transacional, entretanto o esquema de detecção de conflitos tardio usado neste algoritmo, apesar de funcionar bem para transações curtas, é inefetivo para transações maiores. Essas transações desperdiçam uma quantidade significativa de trabalho devido a conflitos de escrita/escrita, que geralmente levam as transações a abortar e são detectados tarde de mais.

Um esquema alternativo à abordagem tardia é a detecção adiantada, que imediatamente aborta uma transação que tenta acessar uma posição de memória bloqueada por outra transação. Dessa forma, conflitos de leitura/escrita, que muitas vezes podem ser tratados sem cancelamentos, são detectados muito cedo e resolvidos abortando os leitores. Transações grandes que escrevem em posições de memória comumente lidas pelas demais, podem acabar bloqueando muitas outras transações, e por um longo período de tempo, assim diminuindo a performance do sistema [Dragojević et al. 2009].

Nesse contexto, o objetivo deste trabalho é a implementação de um esquema de invalidação mista (*mixed invalidation*) [Spear et al. 2006] para a detecção de conflitos na CMTJava. Esta estratégia combina as vantagens da detecção adiantada e tardia, pois os conflitos de escrita/escrita são detectados de forma adiantada, enquanto que conflitos de leitura/escrita são detectados tardiamente. Tal abordagem evita que transações com conflitos de escrita/escrita executem por um longo período antes da detecção, visto que somente uma poderá ser efetivada nesta situação. No entanto, permite que transações com conflitos de leitura/escrita continuem sua execução em paralelo, possibilitando ambas transações conflitantes efetivar, desde que o leitor o faça primeiro. Com o novo sistema transacional implementado, serão realizados testes para a comparação do desempenho desse sistema com os já implementados, e espera-se obter uma melhora considerável de desempenho na execução dos programas.

4. Agradecimentos

Este trabalho foi financiado pelos projetos FAPERGS/PRONEX/CNPq GREEN-GRID e FAPERGS/PESQUISADOR GAÚCHO.

Referências

Dice, D., Shalev, O., and Shavit, N. (2006). Transactional locking ii. In *In Proc. of the 20th Intl. Symp. on Distributed Computing*.

Dragojević, A., Guerraoui, R., and Kapalka, M. (2009). Stretching transactional memory. *Sigplan Notices*, 44.

Du Bois, A. R. and Echevarria, M. (2009). A domain specific language for composable memory transactions in java. In *DSL '09: Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages*, pages 170–186, Berlin, Heidelberg. Springer-Verlag.

Harris, T., Marlow, S., Peyton, S., and Herlihy, J. M. (2005). Composable memory transactions. pages 48–60. ACM Press.

Spear, M. F., Marathe, V. J., Iii, W. N. S., and Scott, M. L. (2006). Conflict detection and validation strategies for software transactional memory. In *In Proc. of the 20th Intl. Symp. on Distributed Computing*.