Melhorando o desempenho da CMTJava com versionamento de dados adiantado

Marcos Gonçalves Echevarria[1], **André Rauber Du Bois**[1]

[1]Programa de Pós-Graduação em Informática – Mestrado em Ciência da Computação
Universidade Católica de Pelotas (UCPEL) – Pelotas – RS – Brazil

{marcosge,dubois}@ucpel.tche.br

1. Introdução

CMTJava [Du Bois and Echevarria 2009] é uma linguagem de domínio específico para programação de memórias transacionais em Java. A linguagem foi desenvolvida adaptando as abstrações da linguagem STM Haskell [Harris et al. 2008] para um contexto orientado a objetos, visando facilitar a programação de máquinas multi-core.

Memórias transacionais fornecem um novo modelo de controle de concorrência que não apresenta as mesmas dificuldades encontradas no uso de bloqueios. Construções em linguagens transacionais são fáceis de serem usadas e podem gerar programas fortemente escaláveis [Adl-Tabatabai et al. 2006].

Transações na linguagem CMTJava são implementadas como uma mônada de passagem de estados e, para isso, é usada uma extensão Java para *closures* [jav 2008]. CMTJava também permite que ações transacionais possam ser combinadas para gerar novas transações. CMTJava apresenta todas as construções de memórias transacionais (*atomic*, *retry*, *OrElse*) e é a primeira extensão Java para transações que suporta a construção *OrElse*.

Este trabalho tem como objetivo apresentar uma proposta para melhorar o desempenho da linguagem CMTJava, através da implementação de um mecanismo de versionamento de dados adiantado. A implementação atual da CMTJava possui um mecanismo de versionamento de dados tardio.

2. A proposta

Hoje a CMTJava, assim como a STM Haskell, utiliza um versionamento de dados tardio, onde os valores alterados por uma transação são armazenados em um buffer enquanto que a memória contém o valor antigo. Se a transação for efetivada, o valor do buffer (valor novo) é escrito na memória e se torna permanente. Caso contrário, se a transação falhar, nada precisa ser feito e o valor que estava no buffer será descartado.

Com o versionamento de dados adiantado, sempre que um dado sofre alteração o novo valor do dado é atualizado diretamente na memória e o valor antigo do dado é guardado em um log. Se a transação for efetivada, o valor do log (valor antigo) é descartado e o valor na memória fica permanente. Caso contrário, se a transação falhar o valor que estava na memória é descartado e o valor do log é escrito na memória.

Muitos sistemas baseados em memórias transacionais utilizavam o versionamento de dados tardio, mas atualmente alguns sistemas vem adotando o versionamento de dados adiantado porque é mais eficiente [Larus and Rajwar 2006].

Quando uma transação precisa ler um dado, com o versionamento de dados tardio é necessário varrer o log da transação para achar o valor correto do dado, com o versionamento de dados adiantado o valor correto já está na memória. Outra diferença é que o versionamento de dados adiantado elimina uma ou duas referências de memória para cada leitura ou escrita da transação.

Além disso, um sistema que utiliza versionamento de dados tardio, ao efetivar uma transação, precisa validar a lista de escritas e leituras da transação para ter certeza de que nenhuma outra transação alterou um dado que foi lido/escrito por essa transação. Também no processo de efetivação, o versionamento de dados tardio perde muito tempo copiando os valores corretos para a memória, o que não é necessário utilizando o versionamento de dados adiantado porque os valores corretos já estão na memória.

Para implementar esse mecanismo é necessário verificar a forma como a detecção de conflitos é feita. Na implementação atual da CMTJava é usada uma detecção de conflitos tardia, pois é verificado se existiu conflito durante a efetivação da transação. Uma detecção de conflitos tardia não funciona com um versionamento de dados adiantado [Adl-Tabatabai et al. 2006]. Desta forma, será necessário, além de implementar um mecanismo de versionamento adiantado, implementar um mecanismo para detecção de conflitos adiantado. Existem três tipos de conflitos que devem ser resolvidos usando uma detecção de conflitos adiantada. Um conflito será detectado sempre que uma transação escrever em uma posição de memória e alguma outra transação ler/escrever nessa mesma posição. Considerando duas transações (T1 e T2), podemos ter os seguintes conflitos:

- **Leitura/Escrita:** T1 lê uma posição de memória. Quando T2 escreve nessa posição, um conflito é detectado.
- **Escrita/Leitura:** T1 escreve em uma posição de memória. Quando T2 lê essa posição, um conflito é detectado.
- **Escrita/Escrita:** T1 escreve em uma posição de memória. Quando T2 escreve nessa posição, um conflito é detectado.

3. Conclusões

Este trabalho apresentou uma proposta para melhorar o desempenho da linguagem CMTJava [Du Bois and Echevarria 2009], implementando um mecanismo de versionamento de dados adiantado. Se essa proposta se mostrar mais eficiente que a atual, esse mecanismo pode ser implementado também na idéia original da linguagem STM Haskell [Harris et al. 2008], a qual serviu como base para a implementação da CMTJava.

Referências

(2008). Java Closures. WWW page, http://www.javac.info/.

Adl-Tabatabai, A.-R., Kozyrakis, C., and Saha, B. (2006). Unlocking concurrency. *ACM Queue*, 4(10):24–33.

Du Bois, A. and Echevarria, M. (2009). A domain specific language for composable memory transactions in java. In Taha, W. M., editor, *DSL*, volume 5658 of *Lecture Notes in Computer Science*, pages 170–186. Springer.

Harris, T., Marlow, S., Jones, S. L. P., and Herlihy, M. (2008). Composable memory transactions. *Commun. ACM*, 51(8):91–100.

Larus, J. and Rajwar, R. (2006). *Transactional Memory*. Morgan & Claypool.