Proposta de um escalonador de transações para o Glasgow Haskell Compiler

Rodrigo M. Duarte[1], **André R. Du Bois**[1], **Gerson G. H. Cavalheiro**[1]

[1]CDTec – Universidade Federal de Pelotas - UFPEL
LUPS - Laboratory of Ubiquitous and Parallel Systems
Gomes Carneiro 1 – 96010-610 – Pelotas – RS – Brazil

{rmduarte,dubois,gerson.cavalheiro}@inf.ufpel.edu.br

Resumo. *Este artigo apresenta como proposta a implementação de um escalonador de transações no Run Time do Glasgow Haskell Compiler, com o intuito de prover melhor desempenho a aplicações concorrentes que utilizem STM-Haskell como modelo de sincronização.*

1. Introdução

Memória Transacional (MT) é um modelo de sincronização entre *threads* que facilita a programação concorrente. Neste as regiões criticas do código são tratadas como transações, parecida com as presente em banco de dados [Harris et al. 2005].

Apesar do elevado nível de abstração apresentado pelas MT, as mesmas ainda sofrem degradação no desempenho quando o programa apresenta elevada contenção de memória. Isso acaba fazendo com que as transações gerem muitos conflitos, levando a re-execução consecutiva das transações e deseperdício de recursos computacionais. [Yoo and Lee 2008]. Com o intuito de resolver este problema, o escalonamento de transações vem sendo aplicado como alternativa [Maldonado et al. 2010]. A ideia principal é executar transações conflitantes em alguma ordem sequencial a fim de evitar novos conflitos.

STM Haskell é uma extensão da linguagem funcional Haskell que fornece a abstração de MT para a programação concorrente. Nela um novo tipo de variável transacional e definida (TVar) [Harris et al. 2005]. Essa variável só pode ser modificada por ações de leitura e escrita dentro de um ação transacional, ou seja, o sistema de tipos de Haskell evita que uma variável transacional seja modificada fora de uma transação. Isso fornece segurança e facilidade no desenvolvimento de programas concorrente.

Glagow Haskell Compiler (GHC) é um compilador e máquina virtual para a linguagem Haskell. Por ser a ferramenta mais atualizada e contendo em seu RTS uma implementação robusta para a concorrência, o GHC é o estado da arte [Peyton-Jones et al. 2017]. Contudo o GHC não possuiu escalonador específico para transações, levando os programas concorrentes, que possuem alta contenção de memória e que usam STM-Haskell para sincronização, apresentarem perda de desempenho.

Alguns trabalhos tem apresentado a utilização de escalonadores de transações para reduzir a taxa de conflitos e tentar aumentar o desempenho de aplicações usando MT. Entre estes trabalhos esta o CAR-STM [Dolev et al. 2008], que se utiliza de instrumentação sobre as transações, para decidir quando deve executar as transações em sequência.

LUTS [Nicácio et al. 2013] que usa heurística e instrumentação para decidir quando as transações devem ser serializadas e até trabalhos como em [Di Sanzo et al. 2016], que utilizam de cadeias de markov para decidir quando as transações devem ser executadas de forma concorrente e quando as mesmas devem ser serializadas, com o escalonador operando dinamicamente sobre o sistema transacional. Os trabalhos apresentados anteriormente são todos escalonadores implementados para bibliotecas de MT em C.

2. Proposta

Este trabalho traz como proposta a modificação do RTS (Runtime System) do GHC, para a inserção de um escalonador especifico para transações. O objetivo principal é fornecer maior desempenho a programas usando STM-Haskell. Foi escolhido o GHC devido o mesmo possuir código fonte aberto e ser o mais atualizado. A ideia inicial e fornecer um escalonador de transações simples, que execute de forma sequencial todas as transações conflitantes e depois, com o domínio do código do RTS, experimentar escalonadores mais avançados como o utilizando cadeias de Markov. Outro objetivo também e fornecer uma interface de alto nível para que o programador posso desenvolver seus próprios modelos de escalonamento.

3. Agradecimentos

O presente trabalho foi realizado com apoio do Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES/Brasil.

Referências

Di Sanzo, P., Sannicandro, M., Ciciani, B., and Quaglia, F. (2016). Markov chain-based adaptive scheduling in software transactional memory. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 373–382. IEEE.

Dolev, S., Hendler, D., and Suissa, A. (2008). Car-stm: scheduling-based collision avoidance and resolution for software transactional memory. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 125–134. ACM.

Harris, T., Marlow, S., Peyton-Jones, S., and Herlihy, M. (2005). Composable memory transactions. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 48–60. ACM.

Maldonado, W., Marlier, P., Felber, P., Suissa, A., Hendler, D., Fedorova, A., Lawall, J. L., and Muller, G. (2010). Scheduling support for transactional memory contention management. In *ACM Sigplan Notices*, volume 45, pages 79–90. ACM.

Nicácio, D., Baldassin, A., and Araújo, G. (2013). Transaction scheduling using dynamic conflict avoidance. *International Journal of Parallel Programming*, pages 1–22.

Peyton-Jones, S., Marlow, S., et al. (2017). Glasgow haskell compiler. Disponivel em https://www.haskell.org/ghc/. Acesso em: Dezembro de 2017.

Yoo, R. M. and Lee, H.-H. S. (2008). Adaptive transaction scheduling for transactional memory systems. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 169–178. ACM.