ESCALONAMENTO DE TRANSAÇÔES PARA O GLASGOW HASKELL COMPILER

RODRIGO MEDEIROS DUARTE[1]; GERSON G. H. CAVALHEIRO, MAURÍCIO LIMA PILLA[1]; ANDRÉ R. DU BOIS[1]

[1]Universidade Federal de Pelotas
{rmduarte, pilla, gerson.cavalheiro, dubois}@inf.ufpel.edu.br

1. INTRODUÇÃO

Memórias Transacionais (MT) é um modelo de sincronização entre threads que facilita a programação paralela. Neste as regiões críticas do código são tratadas como transações, parecidas com os presentes em bancos de dados (RIGO, 2007).

Apesar do elevado nível de abstração apresentado pelas MT, as mesmas ainda sofrem degradação no desempenho quando o programa apresenta elevada contenção de memória. Problemas como re-execução consecutivas devido aos mesmos conflitos e desperdício de recursos computacionais fazem parte desta perda de desempenho (YOO; LEE, 2008).

Com o intuito de resolver este problema, o escalonamento de transações vem sendo aplicado como alternativa (MALDONADO, 2010). A ideia principal é executar transações conflitantes sobre alguma ordem sequencial a fim de evitar novos conflitos.

O *Glasgow Haskell Compiler* (GHC, 2017), é um compilador e máquina virtual para a linguagem funcional Haskell. Este possui várias ferramentas para a geração de código concorrente, entre estas ferramentas, está a biblioteca do STM Haskell, que é uma extensão da linguagem que fornece a abstração de memórias transacionais (MARLOW, 2013). Esta define um tipo de variável transacional conhecida como **`TVar`**. Entre estas primitivas estão as funções **`readTVar`** e **`writeTVar`**, que serve para ler e escrever em uma **`TVar`** respectivamente. Ambas as funções somente podem ser executadas em uma função **`atomically`**. Este tipo de obrigatoriedade e definida pelo forte tratamento de tipos de Haskell, ou seja, mesmo que o programador inadvertidamente tente acessar uma variável transacional fora de uma ação atomically, o compilador retorna um erro informando que o programador deve corrigir seu código.

Apesar de o GHC ser estado da arte, o mesmo não possui escalonador de transações incluso em sua máquina virtual assim, o desempenho do mesmo é degradado quando a aplicação possui alta contenção de memória.

Com o objetivo de melhorar o desempenho das aplicações desenvolvidas usando STM-Haskell, este trabalho propõe a modificação do *Run Time System* (RTS) do GHC para incluir neste um escalonador de transações. O objetivo inicial é inserir um escalonador simples, usando heurísticas e, logo após, com o domínio do código do GHC, utilizar outros modelos de escalonamento de transações presentes na literatura.

Testes serão realizados utilizando *benchmarks* com aplicações usando memórias transacionais, como o STM-Haskell-Benchmark (PERFUMO, 2007), sobre diferentes tipos de contenção de dados, para verificar quais alterações nos desempenhos das aplicações e ao verificar que as mesmas surtiram efeito, será proposta a inclusão deste escalonador a máquina virtual do GHC.

2. METODOLOGIA

Para a realização deste trabalho, torna-se necessário o estudo na literatura sobre escalonamento de transações como também, o estudo do código fonte do GHC, para que se possam aplicar as técnicas estudadas. Como primeiro objetivo, pretende-se aplicar um modelo de escalonador simples, para entender o processo de modificação e compilação do GHC.

O GHC (GHC ,2017) é um compilador, interpretador e máquina virtual para a linguagem funcional Haskell. Com código fonte aberto e mantido por colaboradores, o mesmo é o estado da arte. Possui uma variedade de bibliotecas para a programação concorrente, entre elas o STM-Haskell, que possui uma parte implementada em alto nível (interface com o usuário) e outra no seu RTS. A parte implementada em seu RTS e realizada utilizando linguagem C e possui, mantida pelos desenvolvedores, uma *wiki* que contém informações sobre a estrutura geral do código do GHC. Facilitando assim o estudo para a modificação do mesmo.

A ideia de escalonamento de transações surge a partir da necessidade de reduzir o número de cancelamentos em sistemas transacionais. Um cancelamento ocorre quando duas ou mais transações acessam uma mesma área de memória e pelo menos um destes acessos é de escrita. A fim de evitar que conflitos ocorram repetidamente devido aos mesmos motivos o escalonamento de transações, em sua maioria, utiliza a ideia de serialização. Transações que conflitam são colocadas em uma mesma fila de execução. Executar transações conflitantes de forma sequencial, em sistemas de alta contenção, permite que outros *cores* do processador possam ser usados por outras transações que não são conflitantes, fazendo assim com que o sistema transacional utilize de forma mais racional os recursos computacionais.

Um dos primeiros trabalhos a apresentar o escalonamento de transações foi ATS (YOO; LEE, 2008). Neste trabalho os autores utilizam uma heurística que define a intensidade de conflitos no sistema. Se a intensidade de conflitos aumenta, o sistema começa a serializar todas as transações, quando a intensidade diminui, o sistema começa a aumentar a concorrência entre as transações novamente. Esta intensidade de conflitos é mensurada pelo histórico de cancelamentos das transações.

Outro trabalho apresenta CAR-STM (DOLEV et.al, 2008). Este escalonador opera de duas formas distintas, uma serializando transações conflitantes em uma mesma fila de execução de um mesmo processador. A outra é usando instrumentação, para analisar quais dados as transações estão acessando e tentar prever futuros conflitos entre as mesmas. Caso um conflito seja detectado, a transação é migrada para a fila de tarefas do processador que esta executando a transação conflitante.

Em SHRINK (DRAGOJEVIĆ, 2009), é apresentado um escalonador que serializa as transações através da analise das regiões de memória que as transações acessam. Se as transações acessam as mesmas áreas de memória e o nível de contenção começa a aumentar, o escalonador começa a serializar todas as transações, caso o nível de contenção diminua as transações voltam a ser executadas de forma concorrente.

No trabalho de (NICÁCIO, 2013), um escalonador de transações com dois modelos de decisão é apresentado. O primeiro modelo utiliza heurística para decidir a quantidade de transações concorrentes e o segundo, utiliza uma tabela de probabilidades para decidir qual transação deve ser escalada. A heurística é usada para transações pequenas, para não onerar o sistema transacional. Já a

tabela é usada para transações longas, pois o tempo de execução das transações sobrepõe o *overhead* do escalonador.

Já em (DI SANZO, 2016), é apresentado um escalonador que define a quantidade de concorrência entre as transações, através de uma cadeia de Markov. Quando uma transação é iniciada, o escalonador muda de estado na cadeia, modificando a probabilidade para iniciar uma nova transação, através do histórico anterior. As transições da cadeia definem a probabilidade de quantas transações podem ser executadas concorrentemente no sistema, evitando o efeito de *thrashing* (aumento do número de cancelamentos).

Os trabalhos apresentados acima serão utilizados como base para implementar dois modelos de escalonamento no RTS do GHC, um que serializa todas as transações conflitantes na mesma fila de execução de um único processador e outro, que migra a transação cancelada para a fila de tarefas do processador que está executando a que efetivou.

3. RESULTADOS E DISCUSSÃO

O trabalho ainda não apresenta resultados, pois se trata de uma proposta de implementação. Porém é esperado que o uso do escalonador de transações no RTS do GHC diminua o efeito de *thrashing* em programas que utilizam memórias transacionais para realizar a sincronização. A ideia é inicialmente incluir no GHC um escalonador simples onde as transações conflitantes são serializadas em uma única fila de um mesmo *core,* tentando evitar que novos conflitos ocorram.

Após o domínio do código do RTS e a verificação dos resultados obtidos com a implementação do escalonador simples, outros modelos de escalonamento de transações poderão ser utilizados para saber qual apresentará melhor desempenho.

Uma vez verificado qual o modelo de escalonamento ideal, será proposta a inclusão definitiva para as novas versões do GHC.

4. CONCLUSÕES

Haskell é uma linguagem funcional que possui elevado nível de abstração e esta qualidade facilita o desenvolvimento de programas concorrentes (MARLOW, 2013). Aliada ao forte tratamento de tipos, Haskell também se torna uma linguagem ideal para implementar memórias transacionais (HARRIS et al.,2005). Porém, em aplicações de alta contenção a mesma ainda apresenta perda de desempenho devido ao RTS do GHC não possui escalonador especifico para transações.

A ideia de inserir um escalonador de transações no GHC surge da necessidade de melhorar o desempenho das aplicações escritas em Haskell. Isso pode tornar a linguagem mais atrativa para o uso não só no meio acadêmico, mas como também no meio comercial.

Neste artigo é proposta a implementação de um escalonador de transações para a linguagem funcional Haskell, na máquina virtual do GHC. Espera-se que a utilização deste escalonador reduza a quantidade de conflitos presentes em aplicações que apresentam alta contenção de memória, melhorando seu desempenho geral.

5. REFERÊNCIAS BIBLIOGRÁFICAS

GHC. **Glasgow Haskell Compiler**. Acessado em 29 sept. 2017. Online. Disponível em: https://www.haskell.org/ghc/

RIGO, Sandro; CENTODUCATTE, Paulo; BALDASSIN, Alexandro. **Memórias transacionais: Uma nova alternativa para programação concorrente**. Laboratório de Sistemas de Computação–Instituto de Computação–Unicamp, 2007.

MARLOW, Simon. **Parallel and concurrent programming in Haskell: Techniques for multicore and multithreaded programming**. "O'Reilly Media", 2013.

MALDONADO, Walther et al. Scheduling support for transactional memory contention management. In: **ACM Sigplan Notices**. ACM, 2010. p. 79-90.

DOLEV, Shlomi; HENDLER, Danny; SUISSA, Adi. 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**. ACM, 2008. p. 125-134.

DI SANZO, Pierangelo et al. Markov chain-based adaptive scheduling in software transactional memory. In: **Parallel and Distributed Processing Symposium, 2016 IEEE International**. IEEE, 2016. p. 373-382.

NICÁCIO, Daniel; BALDASSIN, Alexandro; ARAÚJO, Guido. Transaction scheduling using dynamic conflict avoidance. **International Journal of Parallel Programming**, p. 1-22, 2013.

PERFUMO, Cristian et al. Dissecting transactional executions in Haskell. In: **TRANSACT'07: Second ACM SIGPLAN Workshop on Transactional Computing**. 2007.

YOO, Richard M.; LEE, Hsien-Hsin S. Adaptive transaction scheduling for transactional memory systems. In: **Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures**. ACM, 2008. p. 169-178.

DRAGOJEVIĆ, Aleksandar et al. Preventing versus curing: avoiding conflicts in transactional memories. In: **Proceedings of the 28th ACM symposium on Principles of distributed computing**. ACM, 2009. p. 7-16.

DI SANZO, Pierangelo et al. Markov chain-based adaptive scheduling in software transactional memory. In: **Parallel and Distributed Processing Symposium, 2016 IEEE International**. IEEE, 2016. p. 373-382.

HARRIS, Tim et al. Composable memory transactions. In: **Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming**. ACM, 2005. p. 48-60.