Proposta de Escalonamento de Transações do STM Haskell Para Máquinas de Arquitetura NUMA

Rodrigo M. Duarte[1], André R. Du Bois[1], Mauríco L. Pilla[1], Renata H. S. Reiser[1]

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

{rmduarte,dubois,pilla,reiser}@inf.ufpel.edu.br

Resumo. *As arquiteturas NUMA têm sido a alternativa ao problem de gargalo de memória das máquinas SMP. STM-Haskell é uma extensão da linguagem Haskell que provê a abstração de memórias transacionais facilitando o desenvolvimento de programas concorrentes. Este trabalho traz como proposta a implementação de um escalonador de transações, a nível de usuário, para arquiteturas do tipo NUMA.*

1. Introdução

Com o intuito de reduzir o problema do gargalo no barramento de memória e aumentar o nível de paralelismo, a arquitetura NUMA (*Non Uniforme Memory Access*) é uma alternativa que vem ganhando espaço. Nesta cada processador possui um banco de memória específico, porém o endereçamento total de memória é compartilhado entre todos os processadores. Isso acaba gerando diferentes latências ao acesso a memória principal. Com isso, os modelos de programação e escalonamento para esta arquitetura devem ser diferenciados para alcançar bons resultados.

Haskell é uma linguagem funcional pura (livre de efeitos colaterais). Esta característica é excelente para a criação de programas paralelos, visto que nesta, o resultado da computação independe da ordem da avaliação das expressões. Isso isenta o programador de focar na programação direcionada a arquitetura e sim no problema. STM-Haskell é uma biblioteca que fornece as primitivas para a programação usando memórias transacionais em Haskell. Esta aumenta a abstração da linguagem para o desenvolvimento de programas concorrentes.

A implementação mais usada do compilador e máquina virtual da linguagem Haskell é o GHC *Glasgow Haskell Compiler*. Esta ferramenta é a mais completa implementação para a linguagem Haskell, possuindo diversas primitivas para programação concorrentes nativas em sua máquina virtual, bem como inúmeras bibliotecas. Uma destas bibliotecas e a STM-Haskell.

Apesar do GHC ter uma implementação otimizada, a mesma ainda não possui um escalonador específico para arquiteturas NUMA, nem mesmo para transações. No GHC as transações somente são reconhecidas como tarefas comuns, o escalonador não tem nenhuma ação para evitar novos conflitos entre transações e nem otimizar a distribuição destas nas unidades de processadores em máquinas NUMA.

Uma modificação do GHC conhecida como LWC (*Lightweight Concurrency in GHC*) fornece abstrações para a programação concorrente. Entre estas abstrações, estão primitivas para a implementação de escalonadores a nível de usuário.

Este trabalho traz como proposta a implementação de um escalonador, a nível de usuário, usando o GHC e o LWC. A ideia principal é desenvolver um escalonador que reconheça a arquitetura NUMA e assuma outras formas de escalonamento. O objetivo é tentar reduzir as taxas de conflitos entre as transações, bem como explorar a escalonamento de transações levando em conta a localidade e a latência da arquitetura, reduzindo custos para melhorar o desempenho de programas concorrentes que usem memórias transacionais.

2. Arquitetura NUMA

As arquiteturas SMP (*Symetric Multi-Processor*) possuem a característica de fornecer o mesmo tempo de acesso a todas as posições de memória para todos os processadores. Assim, neste modelo de arquitetura, todos os processadores compartilham o mesmo barramento de memória. Com o aumento no número de processadores, surge um limitador para o ganho de desempenho, que é o gargalo gerado no acesso a memória [Kupferschmied et al. 2009].

Arquiteturas NUMA (*Non Uniforme Memory Access*), surgem como alternativa para evitar essa limitação de desempenho [Ribeiro 2011]. Nesta arquitetura cada processador (ou grupo de processadores) possui seu próprio banco de memória. Com isso, os acessos aos endereços de memórias possuem diferentes tempos. Isso se deve ao fato de que, se um processador tiver de acessar um bloco de memória de outro processador, esse terá a latência das redes de interconexão entre os diferentes processadores.

Assim o uso de arquiteturas NUMA tem se tornado uma opção atraente entre custo e desempenho [Annavaram et al. 2005]. Porém, para que se consiga desempenho nesta arquitetura, é necessário que se tenha conhecimento dos detalhes da mesma. Desenvolver uma aplicação (como um escalonador por exemplo) que tire proveito desta, sem levar em conta os custos provenientes das interconexões, pode comprometer o desempenho [Calciu et al. 2013].

3. STM Haskell

Software transactional Memory (STM) é um novo modelo de sincronização entre *threads* que simplifica a programação concorrente, permitindo que operações de acesso a memória possam ser compostas em uma única operação atômica. A ideia é fazer com que as operações sejam realizadas como transações parecidas com as transações de bancos de dados [Rigo et al. 2007]. Neste modelo, todo o sincronismo é realizado pelo sistema transacional, evitando assim problemas como *deadlocks*.

STM Haskell é uma extensão da linguagem Haskell que fornece primitivas para a programação usando STM [Harris et al. 2008]. Nela é definida um tipo de variável transacional (`TVar`), que é criada pela primitiva `newTVar`, e modificada pelas primitivas `readTVar` e `writeTVar`. Estas primitivas só podem ser executadas dentro de uma chamada a `atomically`. STM Haskell garante que operações que modificam uma `TVar`, sejam somente realizadas dentro de uma transação. Assim, o programador não

precisa se preocupar com o sincronismo, pois o sistema de tipos de Haskell garante que nenhuma variável `TVar` seja alterada fora de um bloco protegido, garantindo assim consistência e facilidade no desenvolvimento de programas paralelos.

3.1. Escalonador do RTS do GHC (Glasglow Haskell Compiler)

O Escalonador presente no RTS (*Run Time System*) do GHC utiliza um algoritmo de roubo de tarefas *work stealing*. Algoritmo este com eficiência comprovada na literatura [Blumofe and Leiserson 1999]. No RTS do GHC, para cada processador físico é instanciada uma *thread* de sistema a qual roda um processador virtual (*Capabilty*) [Peyton-Jones et al. 2016]. Cada *Capability* possui uma fila de *threads* de usuário a ser executada. Cada vez que uma destas filas fica vazia, Um *Capability* pode roubar tarefa de um outro que tenha tarefas sobrando em sua fila. Esse algoritmo tenta otimizar ao máximo o uso dos processadores disponíveis.

Apesar da implementação do escalonador do GHC apresentar desempenho satisfatório em máquinas SMP, o mesmo não possui otimizações para arquiteturas NUMA, nem mesmo faz distinção de *threads* que possuem transações das que não possuem. Na implementação atual do GHC, as transações são alocadas nas *threads* de usuário, para execução nos processadores sem considerar a localidade e nem as latências presentes nas arquiteturas NUMA. Com isso, as aplicações desenvolvidas em Haskell usando o GHC para esta arquitetura, acabam apresentando baixo desempenho.

4. LWC - Lightweight Concurrency in GHC

LWC é uma modificação do GHC onde são inseridas primitivas que realizam comunicação direta com o RTS [Li et al. 2007]. Estas primitivas fornecem ações que permitem a manipulação de *threads* para o desenvolvimento de escalonadores a nível de usuário. Em virtude da dificuldade da realização de modificações no RTS do GHC, (código com mais de 50 mil linhas), estas primitivas permitem que, a nível de código Haskell, consiga-se implementar e experimentar diferentes algoritmos de escalonamento. Assim o programador pode realizar testes sem a necessidade de realizar modificações em linguagem de baixo nível (C), e nem recompilar o código do GHC toda vez que realizar uma modificação do escalonador, permitindo assim a rápida prototipação de testes.

Entre as primitivas fornecidas por esta modificação estão funções para criação de *threads*, criação de *Capabilities*, escalonamento e preempção de *threads*, bem como a sincronização de ações por parte do RTS.

5. Proposta

Este trabalho traz como proposta a implementação de um escalonador de transações para máquinas de arquitetura NUMA. Utilizando as primitivas do LWC, pretende-se desenvolver um escalonador de transações que leve em consideração a localidade, latência e custo de migração das transações, tomando como base os dados adquiridos com a analise da estrutura da arquitetura NUMA da máquina onde serão executados os programas. Optou-se por usar as primitivas do LWC porque as mesmas apresentam um elevado nível de abstração para a prototipação de escalonadores, visto que as mesmas possuem interface direta para a linguagem Haskell.

Para verificar a validade da implementação do escalonador, pretende-se utilizar o STM Haskell Benchmark [Perfumo et al. 2007]. Após validados os resultados, pretende-se modificar o RTS do GHC para que o mesmo passe a reconhecer a arquitetura, explore melhor as características desta e obtenha melhor desempenho.

Referências

Annavaram, M., Grochowski, E., and Shen, J. (2005). Mitigating amdahl's law through epi throttling. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 298–309. IEEE.

Blumofe, R. D. and Leiserson, C. E. (1999). Scheduling multithreaded computations by work stealing. *Journal of the ACM (JACM)*, 46(5):720–748.

Calciu, I., Dice, D., Lev, Y., Luchangco, V., Marathe, V. J., and Shavit, N. (2013). Numa-aware reader-writer locks. In *ACM SIGPLAN Notices*, volume 48, pages 157–166. ACM.

Harris, T., Marlow, S., and Jones, S. P. (2005). Haskell on a shared-memory multiprocessor. In *Proceedings of the 2005 ACM SIGPLAN workshop on Haskell*, pages 49–61. ACM.

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

Kupferschmied, P., Stoess, J., and Bellosa, F. (2009). Numa-aware user-level memory management for microkernel-based operating systems. In *Poster/WiP session of the 4th ACM SIGOPS EuroSys Conference (EurosSys' 09)*.

Li, P., Marlow, S., Peyton Jones, S., and Tolmach, A. (2007). Lightweight concurrency primitives for ghc. In *Proceedings of the ACM SIGPLAN workshop on Haskell workshop*, pages 107–118. ACM.

Perfumo, C., Sonmez, N., Cristal, A., Unsal, O., Valero, M., and Harris, T. (2007). Dissecting transactional executions in haskell. In *TRANSACT 07: Second ACM SIGPLAN Workshop on Transactional Computing*, Portland, Oregon, USA. ACM.

Peyton-Jones, S., Marlow, S., et al. (2016). RTS do Glasgow Haskell Compiler. Disponível em `<https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts>`. Acesso em: Novembro de 2016.

Ribeiro, N. S. (2011). Explorando programação híbrida no contexto de clusters de máquinas numa.

Rigo, S., Centoducatte, P., and Baldassin, A. (2007). Memórias transacionais: Uma nova alternativa para programação concorrente. In *Minicursos do VIII Workshop em Sistemas Computacionais de Alto Desempenho, WSCAD 2007*.