Auto-tuning de Granularidade em Memórias Transacionais

Silvio R. Cordeiro[1], **Nicolas Maillard**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{srcordeiro,nmaillard}@inf.ufrgs.br

1. Contexto Científico

A proliferação de processadores multicore trouxe consigo uma forte demanda pela paralelização de código sequencial, onde cada core recebe parte do trabalho e deve sincronizar sua execução com os demais. Entretanto, o modelo de sincronização mais difundido, utilizando locks, impede a composição de código atômico a partir de blocos menores de que também executam atomicamente. Por exemplo, uma sub-rotina que remove um elemento de uma estrutura de dados de acesso atômico e o adiciona em outra estrutura similar só pode garantir que nenhum estado intermediário será visto se ela adquirir um lock interno para cada estrutura, quebrando as suas abstrações [Harris et al. 2005, Larus and Rajwar 2006].

A fim de se eliminar os locks para simplificar a programação paralela, surgiu uma abstração de atomicidade denominada *Memórias Transacionais* [Cascaval et al. 2008]. Um programa que utiliza Memórias Transacionais é similar a um programa sequencial, com algumas seções do código marcadas como *transações*. Em contraste com as seções críticas tradicionais, as transações não exigem um identificador (como um objecto lock) para que possam garantir exclusão mútua. Alterações efetuadas em memória por uma transação são vistas por outras transações de uma só vez, como se a transação fosse uma grande operação atômica.

Os esforços iniciais empregados na implementação de sistemas de memórias transacionais foram divididos entre Memória Transacional baseada apenas em hardware e Memória Transacional baseada apenas em software. Com o tempo, as limitações físicas do hardware e o alto sobrecusto incorrido pelo software levaram à implementação de sistemas híbridos, que executam grande parte das transações diretamente sobre um hardware com suporte a Memórias Transacionais, usando uma implementação em software para os casos excepcionais que o hardware não consegue tratar [Kumar et al. 2006, Baugh et al. 2008].

Uma das preocupações a se considerar no desenvolvimento de uma implementação de Memória Transacional é a de qual deve ser o limiar que determina a escolha entre hardware e software para a execução de uma transação, considerando-se as limitações do hardware em questão. Problemas como a definição do tamanho ideal para cada transação, a granularidade de detecção de conflitos no acesso à memória, a taxa de acessos de escrita em memória e a frequência de conflitos entre transações, compõem um espaço de busca consideravelmente grande, tendo sido pouco explorado [Damron et al. 2006].

2. Auto-tuning

A técnica de *auto-tuning* surgiu como uma forma de permitir que o software automaticamente acompanhe o desenvolvimento do hardware, baseando-se em um conjunto de parâmetros para definir e percorrer um espaço de possibilidades de execução [Asanovic et al. 2009, Williams 2008]. Seu uso está crescendo significativamente na área de computação de alto desempenho [Asanovic et al. 2009], mas a sua aplicação para a otimização de Memórias Transacionais ainda é um problema pouco estudado.

3. Objetivos do Trabalho

Este trabalho busca realizar a exploração de parâmetros em *auto-tuning* sobre aplicações de Memória Transacional. Esses parâmetros podem ser encontrados tanto no nível de algoritmos quanto no do sistema de *runtime* empregado para a execução.

Em relação ao algoritmo, pode-se executar o *auto-tuning* sobre a granularidade das transações, agrupando execuções subsequentes de transações, gerando transações maiores para reduzir o custo gasto entrando e saindo de transações. Nesse caso, o número de transações agrupadas em alguma região de código pode servir como parâmetro de otimização. Já do ponto de vista do sistema de *runtime*, os parâmetros a serem explorados podem variar de acordo com a implementação usada. Por exemplo, em um sistema híbrido, a decisão de quando executar uma transação em software ou hardware pode ser feita a partir de um parâmetro de *auto-tuning*.

Este trabalho tem como contribuição a implementação de um *autotuner* e a avaliação de parâmetros de *auto-tuning* para aplicações de Memórias Transacionais. A partir disso, será possível encontrar dinamicamente o conjunto de características parametrizáveis que mais influenciam o tempo de execução para qualquer determinada plataforma.

Referências

Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., and Yelick, K. (2009). A view of the parallel computing landscape. *Commun. ACM*, 52:56–67.

Baugh, L., Neelakantam, N., and Zilles, C. (2008). Using hardware memory protection to build a high-performance, strongly-atomic hybrid transactional memory. In *ISCA '08*, pages 115–126, Washington, DC, USA. IEEE Computer Society.

Cascaval, C., Blundell, C., Michael, M., Cain, H. W., Wu, P., Chiras, S., and Chatterjee, S. (2008). Software transactional memory: why is it only a research toy? *Commun. ACM*, 51:40–46.

Damron, P., Fedorova, A., Lev, Y., Luchangco, V., Moir, M., and Nussbaum, D. (2006). Hybrid transactional memory. In *ASPLOS '06*, pages 336–346, New York, NY, USA. 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*, PPoPP '05, pages 48–60, New York, NY, USA. ACM.

Kumar, S., Chu, M., Hughes, C. J., Kundu, P., and Nguyen, A. D. (2006). Hybrid transactional memory. In *Principles and Practice of Parallel Programming (PPOPP'06)*, pages 209–220.

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

Williams, S. W. (2008). *Auto-tuning Performance on Multicore Computers*. PhD thesis, EECS Department, University of California, Berkeley.