Memórias Transacionais em Arquiteturas NUMA: Explorando Localidade de Dados e Processos

Douglas Pereira Pasqualin[1], **André Rauber Du Bois**[1], **Maurício Lima Pilla**[1]

[1]CDTec – Universidade Federal de Pelotas (UFPel)
Rua Gomes Carneiro 1 – 96.010-610 – Pelotas – RS – Brazil

{dp.pasqualin,dubois,pilla}@inf.ufpel.edu.br

Resumo. *Memórias transacionais (MTs) são um paradigma para programação concorrente, que utilizam metadados, normalmente centralizados, para garantir propriedades de consistência. Esta característica leva a excesso de contenção, sobretudo quando executados em arquiteturas NUMA (Non-Uniform Memory Architectures). Este artigo apresenta um estudo inicial sobre trabalhos com foco no aumento de desempenho de MT em arquiteturas NUMA.*

1. Introdução

Memória transacional (MT) é um paradigma para programação concorrente que visa a substituição do uso de bloqueios para gerenciar a concorrência dos programas. MT delimita trechos de códigos que devem ser executados de forma atômica, sendo que trechos de códigos que executem sem conflitos devem efetuar um *commit*, ou seja, efetivar as alterações na memória. Em caso de conflitos, uma das transações deve efetuar *abort* e reiniciar a execução, até que consiga efetivar o *commit* [Grahn 2010]. Apesar de MT poder ser desenvolvida tanto em *hardware* (MTH) quanto em *software* (MTS), este trabalho está focando somente em MTS. Muitos algoritmos de MTS utilizam um relógio global para versionamento de locais de memória, ou outros metadados centralizados com o objetivo de garantir propriedades de consistência [Grahn 2010]. Estes metadados geram muita contenção, sobretudo em arquiteturas NUMA (*Non-Uniform Memory Access*).

Este artigo apresenta um estudo de revisão bibliográfica sobre trabalhos com foco no aumento de desempenho de MTS em arquiteturas NUMA. Além disso, são comentadas oportunidades de pesquisa que não foram abordadas nos trabalhos pesquisados.

2. Trabalhos relacionados e oportunidades

Os trabalhos relacionados com MTS em arquiteturas NUMA ou com características similares à NUMA, procuram explorar a localidade de dados ou processos e podem ser divididos em três grupos. O primeiro é a separação do relógio global, por exemplo, um individual por processador físico ou por *thread* [Avni and Shavit 2008, Chan and Wang 2011, Marlier et al. 2014, Mohamedin et al. 2016]. Essa abordagem elimina a contenção do relógio centralizado, porém adiciona efeitos colaterais como a incidência de falsos *aborts*, em virtude dos relógios separados não estarem sincronizados. E caso se opte pela sincronização, um custo adicional de comunicação será adicionado. A segunda abordagem é utilizar formas de versionamento diferenciados (atrasado ou adiantado) entre conflitos que ocorram entre transações executando dentro do próprio nó NUMA ou entre nós [Wang et al. 2012]. Por fim, a última abordagem é a criação de heurísticas para efetuar a migração de *threads* entre nós, levando em consideração somente a localidade da memória

[Chan et al. 2015]. Entretanto, novas pesquisas [Gaud et al. 2015, Diener et al. 2015] sobre arquiteturas NUMA, demostram que a localidade, apesar de ainda ser importante, não é a principal causa de perda de desempenho, e sim a falta de balanceamento entre os nós.

3. Considerações finais

Apesar de MT ser um tópico bastante pesquisado, ainda existem questões com oportunidades de pesquisa, como por exemplo, melhorar o desempenho dos algoritmos quando executados em arquiteturas NUMA. Os trabalhos pesquisados focam em garantir a melhor localidade de memória e processos, evitando o acesso a nós remotos. Porém, pesquisas mais recentes apontam que o balanceamento de carga deve ser priorizado, sendo que localidade de dados deve ser o segundo parâmetro.

Com base nesses argumentos, existem algumas opções que podem ser escolhidas para a continuação do trabalho a ser desenvolvido em MTS: incluir técnicas para balanceamento de carga em arquiteturas NUMA; aprimorar a exploração da localidade dados, porém tentando reduzir a incidência de falsos *aborts*; estudar técnicas utilizadas para sincronização de relógios em sistemas distribuídos e aplicá-las em arquiteturas NUMA.

Referências

Avni, H. and Shavit, N. (2008). Maintaining consistent transactional states without a global clock. In *Proceedings of the 15th International Colloquium on Structural Information and Communication Complexity*, SIROCCO '08, pages 131–140, Berlin, Heidelberg. Springer-Verlag.

Chan, K., Lam, K. T., and Wang, C. L. (2015). Cache affinity optimization techniques for scaling software transactional memory systems on multi-CMP architectures. In *2015 14th International Symposium on Parallel and Distributed Computing*, pages 56–65.

Chan, K. and Wang, C. L. (2011). TrC-MC: Decentralized software transactional memory for multi-multicore computers. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 292–299.

Diener, M., Cruz, E. H. M., and Navaux, P. O. A. (2015). Locality vs. Balance: Exploring data mapping policies on NUMA systems. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 9–16.

Gaud, F., Lepers, B., Funston, J., Dashti, M., Fedorova, A., Quéma, V., Lachaize, R., and Roth, M. (2015). Challenges of memory management on modern NUMA systems. *Commun. ACM*, 58(12):59–66.

Grahn, H. (2010). Transactional memory. *J. Parallel Distrib. Comput.*, 70(10):993–1008.

Marlier, P., Sobe, A., and Sutra, P. (2014). A locality-aware software transactional memory. Euro-TM Workshop on Transactional Memory (WTM 2014).

Mohamedin, M., Palmieri, R., Peluso, S., and Ravindran, B. (2016). On designing NUMA-aware concurrency control for scalable transactional memory. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '16, pages 45:1–45:2, New York, NY, USA. ACM.

Wang, R.-b., Lu, K., and Lu, X.-c. (2012). Aware conflict detection of non-uniform memory access system and prevention for transactional memory. *Journal of Central South University*, 19(8):2266–2271.