Otimização do Desempenho em Memórias Transacionais com Aprendizado de Máquina

Tiago Perlin[1], **Andre Rauber Du Bois**[1]

[1]Centro de Desenvolvimento Tecnológico – Universidade Federal de Pelotas (UFPel)
Pelotas/RS

{tiago.perlin,dubois}@inf.ufpel.edu.br

Resumo. *Memórias Transacionais reduzem a possibilidade de erros na programação e a ocorrência de deadlocks durante a execução em sistemas com múltiplas threads, entretanto, o desempenho geral está relacionado à escolha de políticas e parâmetros de configuração. Neste trabalho são propostas melhorias no controle de paralelismo na Memória Transacional e mapeamento de threads, usando-se Aprendizado de Máquina.*

1. Introdução

Em sistemas computacionais atuais, com execução simultânea de múltiplas *threads*, as Memórias Transacionais (TM - *Transactional Memory*) [Frank and Chun 2008] têm sido usadas como alternativas aos mutexes e semáforos, facilitando-se a programação e evitando-se a ocorrência de *deadlocks*. As regiões críticas do código são delimitadas na forma de transações e podem ser executadas sem interrupções, ideia semelhante às transações em bancos de dados. O uso de mecanismos para identificação de conflitos e a possibilidade de reexecução, garantem a integridade nas transações confirmadas. Entretanto, o uso de TM pode gerar alguma sobrecarga e o desempenho depende da configuração do ambiente, da aplicação e das políticas de controle na TM. Algoritmos de Aprendizado de Máquina (ML - *Machine Learning*) são úteis na otimização de parâmetros de configuração em TM [Di Sanzo et al. 2019]. O objetivo deste trabalho é investigar o uso de ML na otimização simultânea de parâmetros da TM, como paralelismo e política de detecção de conflitos, e parâmetros do sistema, como política de mapeamento de *threads*.

2. Memórias Transacionais e Aprendizado de Máquina

Em alguns trabalhos [Rughetti et al. 2012] [Didona et al. 2016] [Di Sanzo et al. 2019], que propõem o uso de ML para melhorar o desempenho da TM, há a preocupação com o ajuste ótimo do nível de paralelismo, pois com paralelismo elevado observa-se a ocorrência de conflitos frequentes, enquanto que um menor paralelismo produz subaproveitamento da arquitetura. Outros trabalhos [Frank and Chun 2008] [Wang et al. 2012] investigaram como encontrar as melhores políticas (de detecção e resolução de conflitos) em uma TM, sendo que, a escolha da melhor política depende das características da aplicação e da arquitetura. No trabalho de [Castro et al. 2014] foi proposta a otimização da política de mapeamento de *threads*, utilizando-se métricas de desempenho coletadas da TM e métricas do próprio ambiente de execução. No trabalho de [Pasqualin et al. 2020] foi demonstrado que informações quanto a variáveis compartilhadas podem ser coletadas na própria TM e usadas para otimização do mapeamento das *threads*.

Neste trabalho são propostas melhorias no mecanismo de controle de paralelismo da TM, ajustando-se dinamicamente o nível ótimo de paralelismo e calculando-se periodicamente a probabilidade de conflito de cada transação, por meio da coleta de informações gerais, como métricas desempenho (como taxa de *commits* e *aborts*) e de informações específicas quanto a variáveis compartilhadas entre as *threads*. Essas informações, coletadas da própria TM, poderiam, então, ser usadas para a decisão de execução das transações em espera. A otimização simultânea do nível de paralelismo e da política de mapeamento de *threads* poderiam ser alcançadas com o uso de algoritmos de ML supervisionada, como modelos de Redes Neurais Artificiais.

3. Conclusões

Durante a revisão da literatura foram identificados parâmetros na TM que podem ser otimizados e como esses trabalhos estão relacionados. Neste trabalho, em andamento, está sendo proposta a otimização do nível de paralelismo e da política de mapeamento de *threads*, simultaneamente, usando-se algoritmos de ML, esperando-se melhoria de desempenho em uma TM.

Agradecimentos

O presente trabalho está sendo realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

Castro, M., Góes, L., and Méhaut, J.-F. (2014). Adaptive thread mapping strategies for transactional memory applications. *Journal of Parallel and Distributed Computing*, 74.

Di Sanzo, P., Pellegrini, A., Sannicandro, M., Ciciani, B., and Quaglia, F. (2019). Adaptive model-based scheduling in software transactional memory. *IEEE Transactions on Computers*, PP:1–1.

Didona, D., Diegues, N., Guerraoui, R., Kermarrec, A.-M., Neves, R., and Romano, P. (2016). ProteusTM: Abstraction Meets Performance in Transactional Memory. In *Twenty First International Conference on Architectural Support for Programming Languages and Operating Systems*, Atlanta, United States.

Frank, J. and Chun, R. (2008). Adaptive software transactional memory: A dynamic approach to contention management. pages 40–46.

Pasqualin, D. P., Diener, M., Du Bois, A. R., and Pilla, M. L. (2020). Thread affinity in software transactional memory. In *2020 19th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 180–187.

Rughetti, D., Di Sanzo, P., Ciciani, B., and Quaglia, F. (2012). Machine learning-based self-adjusting concurrency in software transactional memory systems. In *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 278–285.

Wang, Q., Kulkarni, S., Cavazos, J., and Spear, M. (2012). A transactional memory with automatic performance tuning. *TACO*, 8:54.