Performance Pathologies in Hardware Transactional Memory
Aluno: João Batista Correa Gomes Moreira - RA: 087331
Jayaram Bobba, Kevin E. Moore, Haris Volos, Luke Yen, Mark D. Hill, Michael M. Swift, David A. Wood, “Performance Pathologies in Hardware Transactional Memory," IEEE Micro, vol. 28, no. 1, pp. 32-41, Jan./Feb. 2008, doi:10.1109/MM.2008.11

Memória Transacional (TM) é um recurso que simplifica o desenvolvimento de algoritmos paralelos através da definição de blocos atômicos chamados transações. O sistema TM executa transações especulativamente, detectando os acessos à memória e efetivando (*commit)* apenas transações não conflitantes. Um conflito ocorre quando diferentes transações acessam o mesmo local de memória e pelo menos um destes acessos é uma escrita. Um sistema TM em Hardware (HTM) possui três funcionalidades críticas: Detecção de conflitos, gerenciamento de versões e resolução de conflitos.

Na execução deste trabalho, três HTMs foram testados. O primeiro (LL) possui detecção de conflitos e gerenciamento de versões do tipo *lazy* (funcionalidades executadas durante o *commit*) e política de resolução de conflitos *Committer Wins*. O segundo (EL) apresenta detecção de conflitos *eager* (funcionalidade executada durante a transação), gerenciamento de versões *lazy* e resolução de conflitos com política *Requester Wins*. O terceiro (EE) apresenta detecção de conflitos e gerenciamento de versão do tipo *eager* e resolução de conflitos com política *Requester Stalls*.

As patologias de eficiência são intrínsecas ao HTM e as transações, elas descrevem problemas que surgem em tempo de execução. A patologia *FriendlyFire* ocorre quando uma transação entra em conflito abortando outra e em seguida é abortada sem efetivar nenhum trabalho. Esta patologia é identificada em TMs do tipo EL. A patologia *StarvingWriter* ocorre quando uma transação que tenta escrever aguarda que outras transações de leitura terminem. Se novas transações de leitura surgem antes que as anteriores finalizem, a transação de escrita estará paralisada por séries continuas de transações de escrita. Esta patologia ocorre em TMs do tipo EE. A patologia *SerializedCommit* ocorre quando uma HTM do tipo LL serializa transações durante sua efetivação para garantir uma ordem global serial, desta forma, transações em processo de *commit* são paralisadas aguardando que outras transações sejam efetivadas. A patologia *FutileStall* ocorre quando a detecção de um conflito faz que uma transação aguarde a conclusão de outra transação que acaba sendo abortada. Esta patologia ocorre em HTMs do tipo EE. A patologia *StarvingElder* ocorre quando transações pequenas abortam transações maiores, pois atingem a fase de *commit* antes e são privilegiadas pela política *Committer Wins*. Esta patologia ocorre em TMs do tipo LL. A patologia *RestartConvoy* ocorre quando uma transação em fase de *commit* entra em conflito com múltiplas instancias da mesma transação. As transações abortadas reiniciam simultaneamente e, por serem semelhantes, competem pelo *commit*. Esta patologia ocorre em HTMs do tipo LL. A patologia *DuelingUpgrades* ocorre quando duas transações lêem e em seguida tentam modificar o mesmo bloco de cache. Este comportamento é comum em sistemas TM, porém se torna problemático em HTMs do tipo EE, já que abortar transações custa caro nestes HTMs.

Buscando evitar as patologias, propõe-se quatro políticas de resolução de conflitos. A política *Predictor* busca solucionar a patologia *DuelingUpgrades,* selecionando permissões exclusivas e adicionando o bloco de cache ao conjunto de escrita da transação. Assim, elimina-se *upgrades* de coerência resultantes de leituras e modificações de uma transação. Sem este recurso, uma das transações que modificam um bloco sempre irá abortar. A política *Hybrid* estende a *Predictor* para reduzir o problema *StarvingWritter,* permitindo que uma transação de escrita mais antiga aborte transações de leitura mais recentes. A política *Timestamp* busca resolver a patologia *FriendlyFire* resolvendo conflitos de acordo com a idade lógica das transações, em vez de aplicar a política *Requester Wins*. Esta política soluciona o *FriendlyFire* garantindo que pelo menos uma transação realize trabalho útil. A política *Backoff* soluciona o *RestartConvoy* aplicando um atraso aleatório no reinício de transações abortadas.

Os testes foram realizados em um sistema de 32 núcleos executado no simulador *Simics*, utilizando tarefas do *benchmark* SPLASH. Os resultados obtidos demonstram que nos três HTMs básicos a performance varia bastante e nenhum deles é o melhor em todos os casos. Observa-se ainda que o HTM EL consome bastante tempo executando tarefas intrínsecas a TM. Os sistemas desenvolvidos com as novas políticas de resolução de conflito apresentaram melhor desempenho. A política *Predictor* produziu *speedups* de até 2.3 em alguns testes. A política *Timestamp* praticamente eliminou problemas de *FriendlyFire* para todos os testes em HTMs EL. A política *Backoff* reduziu drasticamente as patologias *SerializedCommit, StarvingElder* e *RestartConvoy.*

O trabalho realizado demonstra que identificar as patologias e aplicar mecanismos para evita-las e resolve-las adequadamente pode melhorar significantemente a performance de HTMs.