Michael Alexandre Costa

Implementação de um Escalonador Baseado em Heurísticas para Transações em NUMA

Proposta de Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação

Orientador: Prof. Dr. André Rauber Du Bois
Coorientador: Prof. Dr. Mauricio Lima Pilla

Pelotas, 2020

RESUMO

Memórias Transacionais (MT) são apresentadas como alternativa à sincronização com *locks* e monitores. MTs utilizam conceitos de transações semelhantes as existentes em bancos de dados. Estes conceitos permitem ao programador escrever programas paralelos em mais alto nível, reduzindo a complexidade da sincronização.

Ambientes altamente paralelos possuem potencial para um alto número de conflitos entre as transações executadas. Para solucionar estes conflitos e manter consistente a execução de aplicações, sistemas com *Software Transactional Memory* (STM) cancelam uma das transações conflitantes e executam ela novamente após o término da outra transação.

Cancelar as transações conflitantes é eficiente para manter a consistência da aplicação, mas geram um custo de processamento muito elevado. Para reduzir este custo, estudos atuais utilizam escalonadores de transações em STM. Os escalonadores atuam identificando transações conflitantes e manipulando as *threads* em execução para minimizar o número de conflitos futuros.

Estes estudos mostram-se promissores, permitindo reduzir consideravelmente o *overhead* de execução existente em ambientes de alta contenção. Ajustando estas políticas de escalonamento para avaliar a arquitetura utilizada pela aplicação, as aplicações de STM podem ter um ganho de desempenho ainda maior.

Arquiteturas do tipo *Non Uniform Memory Access (NUMA)* oferecem maior capacidade de paralelismo quando comparadas as arquiteturas do tipo *Uniform Memory Access (UMA)*. Porém, os escalonadores e bibliotecas de STM possuem suas aplicações e seu foco de estudo em arquiteturas UMA.

Um escalonador de STM que considera as diferenças entre as arquiteturas existentes pode otimizar sua execução. Esta avaliação da arquitetura deve ser feita pelo escalonador, assim, a biblioteca de STM se mantém simples para o programador, o que facilita o desenvolvimento de programas paralelos complexos.

O objetivo deste trabalho é propor um método de escalonamento baseado em heurísticas para transações em arquiteturas NUMA. Com o conhecimento sobre a arquitetura o escalonador pode otimizar o tempo de execução, executando as transações no processador que obtenha a menor latência de acesso à memória.

Palavras-Chave: escalonador; memórias transacionais; arquitetura numa

1 MOTIVAÇÃO

Memórias Transacionais (MT) são mecanismos de sincronização que realizam execuções atômicas e isoladas de partes compartilhadas do código. Na programação utilizando *Software Transactional Memory* (STM), o acesso à memória compartilhada é realizado dentro de uma transação executada atomicamente (TEIXEIRA, 2015). As Memórias transacionais propiciam aos programadores maior facilidade de desenvolver programas paralelos, onde os programadores não precisam se preocupar com aquisições e liberações de *locks*, assim, evitando problemas como o *deadlock*.

Para garantir a sincronização das transações, STMs utilizam um sistema com detecção de conflitos e gerenciadores de contenção. Os gerenciadores de contenção auxiliam na sincronização e consistência do sistema. Assim, se duas transações manipulam o mesmo dado, o ambiente de execução da STM detecta um conflito, e toma decisão de cancelar uma das transações e reinicia a execução desta de acordo com seu gerenciador de contenção.

Os gerenciadores de contenção garantem a reexecução da transação após observar a ocorrência de um conflito, não sendo sua função impedir que conflitos ocorram. Para obter melhor desempenho em tempo de execução os escalonadores podem usufruir de políticas que evitam a recorrência dos mesmos conflitos. Estas políticas devem ser consideradas, mas nem sempre estão presentes nas bibliotecas de STM.

A utilização de escalonadores de STM permitem maior controle sobre as transações em execução, possibilitando a serialização de parte do código para evitar conflitos, ou até mesmo controlar o fluxo de transações em relação ao número de *cores* de uma máquina. Algumas políticas de escalonamento necessitam de uma base de dados. Estes dados podem ser passados previamente para o escalonador ou coletados em tempo de execução. Este tratamento e análise ocorre de acordo com a heurística utilizada junto ao escalonador.

As heurísticas de escalonadores existentes trazem benefícios para STM, estas heurísticas podem ser melhoradas para um escalonamento voltado a arquitetura NUMA. O trabalho (SANZO, 2017), apresenta uma categorização dos escalonadores existentes na bibliografia, este trabalho analisa as principais heurísticas utilizadas

e suas configurações. Entre os trabalhos avaliados está o escalonador denominado Shrink.

O escalonador Shrink, apresentado em (DRAGOJEVIĆ et al., 2009) e abordado no trabalho de (SANZO, 2017), possui heurística baseada em previsão. Este utiliza um conjuntos de dados previstos, dados lidos por um conjunto de transações executadas em um *thread*, para prever que um conflito pode ocorrer. Os conjuntos de dados que indicam a possibilidade de conflito tem suas transações serializadas pelo escalonador.

O escalonador Shrink foi desenvolvido junto com a biblioteca de STM *tinySTM* (FELBER; FETZER; RIEGEL, 2008), e assim como os demais escalonares estudados na bibliografia foram desenhados para utilizar arquiteturas UMA *(Uniform Memory Access)*, assim sendo, consideram as informações da execução do algoritmo e abstraem as características da arquitetura utilizada, assumindo o uso de arquiteturas UMA.

A utilização de arquiteturas NUMA *(Non Uniform Memory Access)* trazem a vantagem de agregar maior paralelismo ao adicionar mais processadores sem aumentar o gargalo de acesso ao barramento.

As arquiteturas *NUMA* possuem múltiplos núcleos dispostos em conjuntos de processadores (Nodos) e a memória é fisicamente composta por vários bancos de memória, podendo estar cada um deles vinculados a um Nodo e a um espaço de endereçamento compartilhado. Assim, quando um processador acessa a memória que está vinculada a si, acesso local, possuímos um custo de latência. Se o acesso for à memória de outro processador, acesso remoto, o custo de latência é maior que o acesso local.

Os uso de escalonadores otimizam o desempenho das bibliotecas de STM, porém, os trabalhos não consideram em sua tomada de decisão as arquiteturas NUMA. Em arquiteturas NUMA deve ser considerado que as diferentes latências de acesso à memória podem causar *overhead* acima do esperado em uma execução. Esta característica se explorada no momento da tomada de decisão pode ajudar o escalonador a manter a menor latência possível durante a execução de um programa.

Esta proposta de dissertação tem como objetivo principal propor um método de escalonamento baseado em heurísticas para transações em arquiteturas NUMA. Este escalonador buscará reduzir o custo de latência no acesso à memória, otimizando o desempenho de STM em arquiteturas com maior poder de paralelismo.

Como base para o trabalho, será utilizado o escalonador Shrink, que é baseado em previsão. Um escalonador baseado em previsão possui a característica de mensurar e prever a ocorrência de alguma informação relevante ao escalonador. Esta característica se mostrou promissora no Shrink para avaliar o índice de conflito de um conjunto de transações. A heurística baseada em previsão permite estender o escalonador para coletar dados relacionados às arquiteturas NUMA em tempo de execução.

A proposta acima implica em um método de escalonamento baseado em heurísticas para transações em arquiteturas NUMA, que utilizam o escalonador Shrink para implementação como prova de conceito. A heurística de escalonamento deve avaliar a latência de acesso e tomar a decisão de, com base nos demais dados coletados pelo escalonador, serializar as transações, e evitar o maior custo de latência. Com isto, será fornecido um escalonador que em tempo de execução terá conhecimento sobre a arquitetura na qual está sendo utilizado.

2 OBJETIVOS E RESULTADOS

O trabalho proposto tem como objetivo melhorar o desempenho de programas paralelos que usam STM em arquiteturas NUMA. Para isto será modificado um escalonador STM, onde será inserindo premissas que avaliam as características da arquitetura utilizada em tempo de execução. O escalonador avaliará o custo da latência de acesso à memória, com isto tomará decisões sobre as *threads* em execução, podendo serializar as transações ou executar em outros *cores* para reduzir o custo de latência.

O escalonador escolhido para desenvolver este trabalho foi o Shrink, visto em (DRAGOJEVIĆ et al., 2009). Este foi escolhido por sua implementação utilizar a biblioteca *TinySTM*, e suas características proverem a estrutura necessária para aplicar as heurísticas que vão avaliar as arquitetura NUMA.

Shrink é um escalonador baseado em previsão, este avalia a intensidade de conflitos de um conjunto de transações, e sobre estes dados toma ou não a decisão de serializar estas transações. O Shrink também tem uma política de utilizar a coleta de dados apenas para aplicações com alta carga de conflitos, assim evita *overhead* desnecessário e otimiza seus resultados.

O trabalho terá a revisão bibliográfica da área, e a avaliação da diferença entre os tempos de execução dos escalonadores de STM executados em arquiteturas UMA e NUMA. Será realizado testes na arquitetura NUMA com o escalonador modificado, a fim de avaliar o impacto de uma heurística de escalonamento de transações para arquiteturas NUMA.

O principal resultado esperado com o trabalho é melhorar o desempenho de tempo de execução dos escalonadores de STM em arquiteturas NUMA, ao utilizar com maior eficiência e consciência os recursos providos em NUMA. Também, manter um baixo *overhead* para o uso de STM, utilizando o escalonador apenas com alta contenção e utilizando as heurísticas NUMA apenas para estas arquiteturas. Por fim, pretende-se fornecer um escalonador STM que em tempo de execução avalie e considere as características da arquitetura utilizada para otimizar seu desempenho.

3 METODOLOGIA

Para a realização deste trabalho, será realizado o estudo do código fonte do escalonador Shrink, disponível em (DRAGOJEVIĆ, 2010). O estudo será focado no entendimento da biblioteca *TinySTM*, onde está implementado o código do escalonador Shrink. Neste cenário de estudo da biblioteca, destina-se tempo para utilizar junto com escalonador o *benchmark STAMP* apresentado em (MINH et al., 2008). O conjunto de *benchmark STAMP* é um dos conjuntos de *benchmarks* mais utilizados na bibliografia.

Um escalonador baseado em heurísticas para transações em arquiteturas NUMA, que avalie as diferentes latências de acesso à memória, será a contribuição principal deste trabalho.

O custo de latência é a principal característica NUMA a ser avaliada pelo escalonador. Onde, de forma simples o escalonador busca reduzir a latência migrando as transações com maiores custos de acesso à memória e serializando as transações conflitantes. Para evitar *overhead* de execução será considerado, para habilitar o escalonador, o nível de contenção do sistema e a arquitetura na qual está sendo executada.

Após a realização das modificações serão realizados testes utilizando as duas arquiteturas, estes testes têm a finalidade de avaliar estatisticamente o desempenho fornecido pelo escalonador modificado. Também serão realizados testes nas duas arquiteturas com escalonador Shrink sem modificação, este tem como objetivo comparar estatisticamente o desempenho entre o Shrink e sua modificação fornecida neste trabalho.

Para demais comparações e avaliações estatísticas, pretende-se executar testes com outros escalonadores disponíveis na bibliografia e a biblioteca *tinySTM* sem o uso de escalonadores. Assim, permitindo examinar a influência de escalonadores de STM em arquitetura NUMA.

O escalonador Shrink possui duas premissas importantes de escalonamento. A primeira, é utilizar o escalonador apenas para ambientes de alta contenção, assim, o escalonador só é utilizado quando um determinado limiar é ultrapassado por um contador de conflitos. Isto mantém um baixo custo de execução em ambientes com baixa

contenção. A segunda, é serializar todo conjunto de transação com alta possibilidade de conflito, por meio da verificação do número de conflitos existentes em um conjunto de transações. Isto evita que conflitos já existentes voltem a ocorrer.

Estas característica são importantes para reduzir o tempo de execução. Neste trabalho pretende-se estender estas características e inserir a migração das transações com alta latência de acesso. Um dos desafios deste trabalho será manter o desempenho em ambientes de baixa contenção, e manter o escalonador original em arquiteturas UMA. Para arquiteturas NUMA, o desafio torna-se prover a migração das transações. Este último tem o objetivo fornecer o menor *overhead* possível ao sistema.

Para os testes realizados nas duas arquiteturas, será utilizado junto com biblioteca de STM *TinySTM* e escalonadores o conjunto de *benchmarks STAMP*. Este fornece uma gama distinta de *benchmarks*, onde podemos testar ambientes de alta e baixa contenção.

4 CRONOGRAMA

1. Revisar a bibliografia relacionada, estudando os principais escalonadores implementados e suas característica;
2. Estudar o código fonte do escalonador Shrink como base para implementação da heurística NUMA proposta;
3. Realizar as primeiras avaliações das heurísticas de escalonamento;
4. Escrever a fundamentação teórica da dissertação;
5. Apresentar o seminário de andamento;
6. Executar os testes de validação do escalonador com as mudanças bases previstas;
7. Implementar o escalonador utilizando as heurísticas NUMA;
8. Realizar e verificar os testes finais para avaliação dos dados coletados;
9. Escrever a dissertação final;
10. Entrega da dissertação final.

Tabela 1 – Cronograma de atividades do mestrado

	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												

REFERÊNCIAS

DRAGOJEVIĆ, A. **Shrink for TinySTM**. Disponível em: <http://lpd.epfl.ch/transactions/wiki/doku.php?id=scheduling>.

DRAGOJEVIĆ, A.; GUERRAOUI, R.; SINGH, A. V.; SINGH, V. Preventing versus curing: avoiding conflicts in transactional memories. In: ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 28., 2009. **Proceedings...** [S.l.: s.n.], 2009. p.7–16.

FELBER, P.; FETZER, C.; RIEGEL, T. Dynamic Performance Tuning of Word-Based Software Transactional Memory. In: PPOPP '08: PROC. OF THE 13TH ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING, 2008, New York, NY, USA. **Anais...** ACM, 2008. p.237–246.

MINH, C. C.; CHUNG, J.; KOZYRAKIS, C.; OLUKOTUN, K. STAMP: Stanford Transactional Applications for Multi-Processing. In: WORKLOAD CHARACTERIZATION, 2008. IISWC 2008. IEEE INTERNATIONAL SYMPOSIUM ON, 2008. **Anais...** [S.l.: s.n.], 2008. p.35–46.

SANZO, P. D. Analysis, Classification and Comparison of Scheduling Techniques for Software Transactional Memories. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.], v.28, n.12, p.3356–3373, Dec 2017.

TEIXEIRA, F. L. **Análise do Impacto de Diferentes Versionamentos de Dados das Memórias Transacionais sobre Memórias Phase-Change**. 2015. Dissertação de Mestrado — PPGC/UFPEL, Pelotas/RS.

5 ASSINATURAS

Michael Alexandre Costa
Proponente

André Rauber Du Bois
Prof. Orientador