Analise do desempenho de STM em arquiteturas NUMA

MICHAEL ALEXANDRE COSTA[1]; ANDRÉ RAUBER DU BOIS[2]; MAURÍCIO LIMA PILLA[3]

[1]*Universidade Federal de Pelotas – macosta@inf.ufpel.edu.br*
[2]*Universidade Federal de Pelotas – dubois@inf.ufpel.edu.br*
[3]*Universidade Federal de Pelotas – pilla@inf.ufpel.edu.br*

1. INTRODUÇÃO

O ganho no desempenho dos processadores atuais deve-se ao desenvolvimento de arquiteturas paralelas. Porém, para que os programas atuais extraiam total desempenho destas arquiteturas é necessário um grande esforço por parte do programador.

Um dos problemas enfrentados está no acesso à memória compartilhada. Duas ou mais *threads* acessando uma região de memória ao mesmo tempo pode ocasionar condição de corrida e erros na execução do programa. Para que tais erros não aconteçam atualmente são utilizados mecanismos de sincronização conhecidos como *locks*.

Este método de sincronização exige cuidado redobrado do programador, pois seu mau uso pode levar a outros erros como de *deadlock*. Como este modelo é difícil de programar, pesquisadores buscam novas alternativas ao uso de *locks*. Memórias transacionais ou TMs é uma alternativa que reduz a dificuldade de implementação de programas paralelos.

Memórias Transacionais em Software (STM) são mecanismos de sincronização que realizam operações atômicas e isoladas de partes compartilhadas do código. As STMs permitem ao programador não se preocupar com aquisições e liberações de *locks*, demarcando apenas as seções criticas de código.

O uso de STMs para construção de programas paralelos reduz a complexidade de programação, mas apresenta comportamentos distintos conforme a arquitetura utilizada. Arquiteturas UMA (*Uniform Memory Access*) possuem um único barramento entre todos os núcleos do processador e a memória, assim possuindo latência igual para todos núcleos.

Arquiteturas NUMA (*Non-Uniform Memory Access*) possuem um barramento de acesso para cada conjunto de núcleos (nodo) e região memória. Assim, o acesso de um núcleo à sua região de memória possui menor latência que seu acesso à região de memória de outro nodo.

Estas distintas características entre arquiteturas impactam diretamente no desempenho de execução das STMs. Dessa forma, é importante comparar e analisar a diferença de desempenho de aplicações STMs quando executadas em arquiteturas UMA e NUMA com configurações iguais.

A principal diferença deste trabalho para outros relacionados é a utilização de configurações iguais de máquinas UMA e NUMA, sem a utilização de simuladores. Os resultados alcançados demonstram que o uso de STM piora de forma significativa em arquiteturas NUMA.

2. METODOLOGIA

Para compreender e analisar as diferenças de desempenho de STM entre máquinas UMA e NUMA, foi utilizado uma máquina (NUMA) com 4 processadores

AMD *Opteron* 6276 2.3 Ghz, cada processador com 16 núcleos, suportando 64 *threads* no total. Esta máquina possui 8 nodos NUMA, pois o processador físico AMD *Opteron* é dividido internamente em 2 nodos NUMA. Cada nodo possui 16 Gb de memória RAM, o sistema operacional utilizado foi o *Ubuntu Server* 14.04.

Foi utilizada somente uma máquina, configurada de duas maneiras, para realizar a comparação entre as arquiteturas. Para a configuração UMA, foi utilizado um nodo da máquina (nodo zero), deixando 8 núcleos ativos. Para garantir que o nodo somente iria usar a sua região de memória local, foi utilizada a ferramenta *numactl*[1] que permite restringir a quais nodos ou locais de memória que um determinado processo irá utilizar (LAMETER, 2013). Os demais processadores da máquina foram desligados no *kernel*, para garantir que não seriam utilizados, garantindo assim a consistência dos testes.

Para configuração NUMA também foram utilizados apenas 8 núcleos. Porem, a configuração mudou para 4 núcleos do nodo zero mais 4 núcleos do nodo três. Todos os outros foram desligados no *kernel*. Também foi utilizada a ferramenta *numactl* para restringir o acesso as regiões de memória.

Conforme dados obtidos com a ferramenta *numactl*, a latência de acesso à memória local é de 10ns, enquanto que a remota (nodo zero acessando o três) é de 22ns. Sem testes como o proposto neste artigo não é possível mensurar o real impacto desta diferença de latência, principalmente sem utilizar uma mesma configuração de UMA e NUMA. Neste caso, foi utilizado 8 núcleos iguais acessando somente a memória local, ou 4+4 núcleos dispostos em nodos distintos.

A biblioteca de STM utilizado foi a TinySTM (FELBER, 2008). Este é um dos algoritmos estado da arte de STM (PEDRERO, 2017). Foi utilizada a configuração padrão de compilação, com versionamento atrasado e gerenciamento de contenção suicida, no qual a transação que identifica o conflito deve ser cancelada (*abort*) e reinicializada instantaneamente.

Foram executados os benchmarks Intruder e SSCA2 do conjunto de *benchmarks* STAMP (MINH, 2008). O objetivo do Intruder é detectar a invasão de uma rede buscando por padrões conhecidos de ataques, sendo o nível de contenção deste teste alto. Já o SSCA2 constrói uma estrutura de dados em grafos usando vetores adjacentes e auxiliares, sendo o nível de contenção baixo. Apesar do STAMP possuir mais *benchmarks*, foram escolhidos estes dois por possuirem os maiores tempos de execução, e possuirem quantidade de contenção e total de *aborts* bastante diferentes, sendo possível assim efetuar uma analise de diferentes perspectivas sobre os resultados.

A versão utilizada do STAMP foi a 0.9.10, juntamente com a TinySTM 1.0.5, ambos disponíveis em *TMWare*[2]. Por fim, foram obtidas 30 execuções de cada *benchmark*, em cenários de 1, 2, 4 e 8 *threads*, para cada configuração de máquina descrita acima.

3. RESULTADOS E DISCUSSÃO

Para cada *benchmark* testado, foram obtidos o tempo de execução e o número de *aborts* em cada cenário de *thread*. A partir destes dados amostrados foram calculados a média e desvio padrão, assim como o coeficiente de variação dos testes. A figura 1 apresenta os gráficos das médias e desvio padrão do tempo de execução e número de aborts coletados.

---

[1] http://man7.org/linux/man-pages/man8/numactl.8.html

[2] http://tmware.org/tinystm.

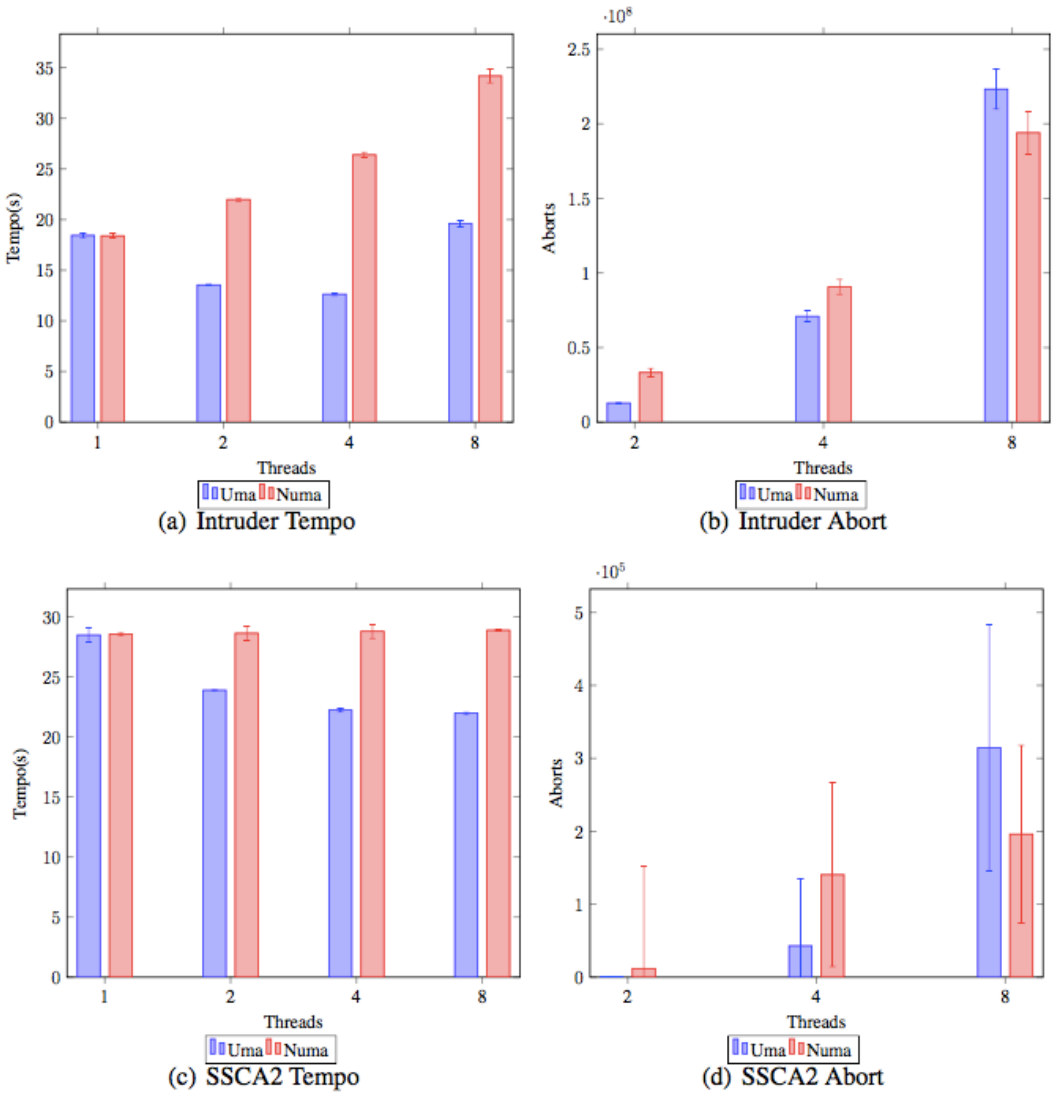

Figura 1: Médias dos números de aborts e tempos de execução usando UMA e NUMA

O tempo de execução com 1 *thread* para os dois *benchmarks* apresentam resultados estatisticamente iguais, dado um p-valor de 0.533 para os testes do Intruder e 0.602 para os testes do SSCA2. Este resultado é apresentado devido a execução com 1 *thread* ser serial e não apresentar conflitos. Por ser uma execução serial, os testes com 1 *thread* também não apresenta *aborts*, assim, os dados de *aborts* com 1 *thread* foram ocultados dos gráficos (b) e (d).

A partir de 2 *threads,* os testes apresentam diferença comportamental entre as arquiteturas. Onde, o *benchmark* Intruder em UMA escala até 4 *threads* e apresenta uma queda de desempenho com 8 *threads*. Esta queda no desempenho está associada ao gargalo de acesso à memória por meio de um único barramento. Com uso da arquitetura NUMA, conforme aumenta o número de *threads* há uma piora no desempenho. Isto está relacionado ao alto nível de contenção do *benchmark* atrelado as diferentes latências de acesso a memória local e remota.

O *benchmark* SSCA2 escala em arquitetura UMA, sendo que os resultados apresentados em 2 *threads* são próximos, porém, estatisticamente distintos com p-valor de $1.644*10^{-28}$. Os testes apresentaram em NUMA pior resultados quando

comparados a UMA. Os resultados em NUMA também se mostraram constantes com o aumento do número de *threads*. Isto ocorre devido os tipos de estruturas de dados manipuladas pelo *benchmark* atrelada as diferentes latências de acesso à memória.

Os dois *benchmarks* apresentaram um crescimento no número de *aborts* conforme aumenta o número de *threads*. Isto ocorre devido ao aumento de contenção gerado pelo maior paralelismo.

O uso de STM mostrou-se promissor em arquiteturas UMA, porém, em aplicativos com alto índice de conflitos, conforme aumenta o paralelismo o gargalo imposto pela arquitetura gera uma queda de desempenho. Em arquiteturas NUMA, que permitem maior paralelismo, o uso de STM apresenta menor desempenho. Este menor desempenho está diretamente correlacionado as diferentes latências de acesso à memória, visto que a biblioteca de STM utiliza um relógio global, apresentando como problema a centralização de dados.

4. CONCLUSÕES

Máquinas com arquiteturas NUMA estão se tornando dominantes em configurações com mais de um processador, sobretudo em servidores. Apesar da principal vantagem da arquitetura ser o fato de possui bancos de memórias separados por processadores, diminuindo a concorrência de acesso, isso pode ser um problema para aplicações que utilizam estrutura de dados centralizadas. O objetivo deste artigo foi analisar o comportamento do algoritmo TinySTM, em 2 aplicações com níveis de contenção diferentes, comparando o resultado de execução em arquiteturas UMA e NUMA, com configurações iguais.

Os resultados demonstraram que o uso de STM apresentam perda significativa em máquinas NUMA conforme aumenta o número de *threads*. Isto está relacionado com o uso de estruturas de dados centralizadas como o relógio global da TinySTM. Outro fator visível nos resultados é que as distintas latências de acesso à memória atrelado ao custo de refazer operações abortadas oneram o desempenho.

Como trabalhos futuros pretende-se expandir os testes, utilizando mais aplicações do *benchmark* STAMP, aumentando o número de *threads*, variando o tipo de versionamento e gerenciador de contenção da TinySTM, para verificar se algum resultado positivo é apresentado em arquiteturas NUMA. Também pretende-se efetuar alterações na TinySTM, obtendo uma versão NUMA *aware*, visto que os algoritmos de STM são desenvolvidos para arquiteturas UMA.

5. REFERÊNCIAS BIBLIOGRÁFICAS

FELBER, P. FETZER, C. RIEGEL, T. Dynamic Performance Tuning of Word-based Software Transactional Memory. **ACM**, New York, n.10, p.237 - 246, 2008.

LAMETER, C. NUMA (Non-Uniform Memory Access): Ann overview. **Queue**, New York, v.11, n.7, p.40 - 51, 2013.

MINH, C. CHUNG, J. KOZYRAKIS, C. OLUKOTUN, K. (STAMP): Stanford Transactional Applications for Multi-Processing. **2008 IEEE International Symposium on Workload Characterization**, p. 35 - 46, 2008.

PEDRER, M. GUITIERREZ, E. ROMERO, S. PLATA, O. (ReduxSTM): Optimizing (STM) designs for Irregular Applications. **Journal Of Parallel and Distributed Computing**, v.107, p. 114 - 133, 2017.