Análise de Padrões de Escrita das Memorias Transacionais em Software

TEIXEIRA, Felipe Leivas[1*]; PILLA, Maurício Lima[1]; DU BOIS, André Rauber[1]

[1]Universidade Federal de Pelotas (UFPEL)
Computação - CDTec
Caixa Postal 354 · CEP 96001-970 · Pelotas, RS - Brasil
{flteixeira, pilla, dubois}@inf.ufpel.edu.br

*Bolsista GREEN-GRID
FAPERGS/PqG (11/1065-1), PRONEX/FAPERGS/CNPq (10/0042-8)

1. INTRODUÇÃO

O consumo de energia é um fator cada vez mais importante nos grandes *data centers*. A *Phase Change Memory* (PCM) (LEE et al., 2010) surgiu como uma alternativa para o consumo proibitivo de energia em memórias tradicionais, pois após o armazenamento de um bit não há consumo de energia para mantê-lo como em memórias RAM estáticas ou dinâmicas.

O problema das memórias PCM está em suas escritas, que são lentas, visto que o processo de armazenamento de um bit envolve alterar o estado do material da célula de memória em questão (LEE et al., 2010). Além das escritas serem lentas, elas desgastam o material devido à mudança de fase que fazem para armazenar os dados, diminuindo sua vida útil.

As Memórias Transacionais (HERLIHY et al., 1993) ou *Software Transactional Memories* (STM) são uma alternativa para sincronização de *threads*. O conceito por trás de STM é baseado em transações de banco de dados, assim tendo como uma vantagem sobre as sincronizações baseadas em *locks* a simplicidade de escrita de código.

Outra vantagem das STMs é que não existe o problema de *deadlock* que ocorre na sincronização baseada em *locks* (HARRIS et al., 2010). A composição de componentes de softwares também é mais simples do que em sistemas baseados em *locks*, aumentando a reusabilidade do código (HARRIS et al., 2010). Memórias Transacionais são uma alternativa promissora para a escrita de código portável e escalável em memória compartilhada.

Nesse trabalho, será analisado o impacto de escritas de *benchmarks* de memórias transacionais em software. Para tanto, a biblioteca de STM TinySTM (FELBER et al., 2008), a ferramenta PinTools (LUK et al., 2005) e o *benchmark* STAMP (CAO MINH et al., 2008) foram usados para gerar um ambiente de avaliação.

O artigo é dividido da seguinte forma: a Seção 2 apresenta a metodologia. A Seção 3 mostra os resultados obtidos. A Seção 4 discute as conclusões.

2. METODOLOGIA

Para a execução deste trabalho, foi implementada uma instrumentação de código utilizando a ferramenta Pintools. Para a implementação da instrumentação foi desenvolvido um programa em C++ que conta cada escrita feita à memória e também faz uma avaliação de quantos bits foram trocados na escrita.

A implementação foi feita da seguinte forma: a cada escrita na memória é incrementado um contador, para contar o número de escritas. Também é armazenado o endereço de memória que está sendo escrita e o valor que será

escrito nela. Com o endereço de memória é obtido o valor atual que está armazenado em um vetor e é feita uma comparação com o valor que será escrito naquele endereço, de modo a verificar quantos bits foram modificados. Essa comparação é feita em tempo de execução. Logo depois de fazer essa comparação, o vetor é atualizado com o valor escrito.

3. RESULTADOS E DISCUSSÃO

Os *benchmarks* foram executados em uma máquina com processador Core i7 2600 4 cores físicos e 8 cores lógicos, com memória RAM de 8GiB. Para cada *benchmark* do STAMP e configuração foram medidas dez execuções para cada cenário com 1, 2, 4 e 8 *threads* com a instrumentação implementada.

Tabela 1: Tabela do número de escritas (em milhões)

	1 Thread	2 Threads	4 Threads	8 Threads
Bayes	883	883	883	883
Genome	382	351	347	376
Intruder	1233	1312	1468	1793
Kmeans	1738	1732	1745	1841
Labyrinth	2318	2572	3054	3702
SSCA2	429	430	430	430
Vacation	2407	2417	2423	2430
Yada	2244	2621	2994	3297

Os *benchmarks* mostraram diferentes padrões de números de escritas de acordo com o número de *threads*, como pode ser visto na Tabela 1. Alguns *benchmark* mostraram um aumento do número de escritas com o aumento do número de *threads*: *Intruder*, *Labyrinth*, *SSCA2*, *Vacation* e *Yada*. Este resultado é esperado, pois aumentando o número de *threads* concorrentes, o número de escritas deve aumentar se não houver contenção.

Os outros *benchmarks* mostraram uma variação de acordo o número de *threads*. O *benchmark Genome* foi o único que mostrou um número menor de escritas com 2, 4 e 8 *threads* em relação a quando executado com uma *thread*. Ele mostrou uma queda de 8% no número de escritas quando o número de *threads* foi aumentado de 1 para 2 *threads*. Este resultado pode estar relacionado com um aumento na contenção e serialização das *threads* ou com a implementação do *benchmark*.

Tabela 2: Tabela do número total de bits alterados nas escritas (em milhões)

	1 Thread	2 Threads	4 Threads	8 Threads
Bayes	2863	2873	28632	2850
Genome	1135	1032	1038	1108
Intruder	5561	6046	6603	7907
Kmeans	9278	9225	9294	9575
Labyrinth	9767	10480	12239	14776
SSCA2	1307	1315	1326	1308
Vacation	9172	9244	9232	9284
Yada	12780	14279	15949	17739

Na Tabela 2 é analisado o número total de bits alterados em todas escritas que ocorreram na execução dos testes. Ela mostra um padrão parecido com o da Tabela 1, ou seja, o número de bits alterados é quase sempre uma função do número de escritas, mas com algumas diferenças interessantes. Por exemplo o *benchmark Vacation* que com 4 *threads* e no *benchmark* SSCA2 com 8 *threads* tiveram um aumento do número de escritas mas uma diminuição do número de bits alterados. Já no *benchmark Genome* com 4 *threads* mostrou o contrário, mostrou um aumento do no número de bits alterados mesmo com a diminuição do número de escritas. Estes resultados podem estar relacionados com a implementações dos *benchmarks*.

A ideia geral dos resultados é encontrar padrões nas escritas nos *benchmarks* de memória transacionais. Isso para permitir uma melhor compreensão dos padrões de acesso que as STM devem gerar. A partir destes resultados será possível desenvolver adaptações nas bibliotecas de STM para tirar melhor proveito do conhecimento da aplicação, diminuindo desgaste, consumo e tempo de escrita em uma memória PCM.

4. CONCLUSÕES

Neste artigo, foi analisado o impacto de escritas de *benchmarks* de memórias transacionais em software. Como *benchmark* foi usado o STAMP, e ele implementa vários *benchmarks,* assim cobrindo uma ampla área de aplicação das STMs é coberta. Com isso foi possível analisar as STMs em várias situações diferentes, tendo então uma melhor verificação do impacto de suas escritas.

Os resultados mostraram alguns padrões nas escritas das STMs. Um destes foi que na maioria dos *benchmarks* quanto maior era o número de *threads* maior era o número de escritas, o número de bits alterados.

Alguns *benchmarks* mostraram um comportamento diferente. Conforme aumentava o número de *threads*, o número de escritas e o número de bits alterados variavam, o que pode ser justificado em parte por suas implementações, parâmetros escolhidos entre outras razões, que serão detalhadas em trabalhos futuros.

Para trabalhos futuros, pretende-se fazer uma análise da concentração das escritas das STMs para observar em quais posições de memória concentra-se o maior número de escritas e quais bits são mais modificados no decorrer da execução. Também pretende-se avaliar qual o comportamento das STMs ao longo do tempo de execução. Além disso, pretende-se fazer uma adaptação de uma biblioteca de STM, analisando os resultados deste artigo, para que diminua o número de escritas em uma memória PCM, reduzindo o tempo de execução, o consumo de energia, e aumente o tempo de vida de uma memória PCM.

5. REFERÊNCIAS BIBLIOGRÁFICAS

LEE, B. C.; ZHOU, P. Z. P.; YANG, J. Y. J.; ZHANG, Y. Z. Y.; ZHAO, B. Z. B.; IPEK, E.; MUTLU, O. and BURGER, D., "Phase-change technology and the future of main memory," **IEEE Micro, vol. 30**, no. 1, pp. 131–141, 2010.

HERLIHY, M.; ELIOT, J. and MOSS, B., "Transactional memory: Architectural support for lock-free data structures," in **Proceedings of the 20th Annual International Symposium on Computer Architecture**, 1993, pp. 289–300.

HARRIS, T.; LARUS, J. and RAJWAR, R., "Transactional memory, 2nd edition," **Synthesis Lectures on Computer Architecture, vol. 5**, no. 1, pp. 1–263, 2010.

FELBER, P.; FETZER, C. and RIEGEL, T., "Dynamic performance tuning of word-based software transactional memory," in **PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming**. New York, NY, USA: ACM, 2008, pp. 237–246.

LUK, C. K.; COHN, R.; MUTH, R.; PATIL, H.; KLAUSER, A.; LOWNEY, G.; WALLACE, S.; JANAPA; V. and HAZELWOOD, R. K., "Pin: Building customized program analysis tools with dynamic instrumentation," In **Programming Language Design and Implementation. ACM Press**, 2005, pp. 190–200.

CAO MINH, C.; CHUNG, J.; KOZYRAKIS, C. and OLUKOTUN, K., "STAMP: Stanford transactional applications for multiprocessing," in **IISWC '08: Proceedings of The IEEE International Symposium on Workload Characterization**, September 2008.