Avaliação dos Padrões de Escrita de *Benchmarks* de STM

Michael A. Costa[1]*, Felipe L. Teixeira[1]†, Mauricio L. Pilla[1], André Rauber Du Bois[1]

{macosta,flteixeira,pilla,dubois}@inf.ufpel.edu.br

[1]Laboratory of Ubiquitous and Parallel Systems– Universidade Federal de Pelotas (UFPEL)
Computação - CDTec
Caixa Postal 354 · CEP 96001-970 · Pelotas, RS - Brasil

Resumo. *Phase Change Memory é uma memória estudada para substituir as atuais DRAMs. Memória Transacional é um mecanismo de sincronização para substituir os mecanismos baseados em locks. Neste artigo, analisamos os padrões de escritas das memórias transacionais em PCM. Os resultados apresentam um aumento do número de escritas em relação ao aumento de threads. Em alguns casos, como Genome 1-2 threads, há um decremento destes valores.*

1. Introdução

A memória compartilhada em processos concorrentes e o consumo de energia são os principais problemas dos computadores hoje em dia.

Phase Change Memory (PCM) [Lee et al. 2010] é uma alternativa para diminuir o consumo de energia. A PCM é estudada para ser a memória principal nas arquiteturas atuais de computadores. Sua eficiência energética é em relação ao armazenamento dos dados, pois após o armazenamento de um bit não há custo para mantê-lo. Outra vantagem é seu tempo de leitura, porém seu tempo de escrita é elevado.

As memórias transacionais ou *Software Transactional Memory* (STM) [Herlihy et al. 1993], são alternativas de sincronizações de *threads*. A memórias transacionais são baseada em transação de banco de dados. Suas vantagens são em relação a facilidade de programação e a inexistência de *deadlock*.

Neste artigo será feita uma analise estendida dos *benchmarks* de STM *STAMP* [Cao Minh et al. 2008] com impacto em memórias PCM. Este trabalho é uma extensão do trabalho [Teixeira et al. 2014], no caso, considerando somente o número de escritas. Para tanto, foram utilizados a biblioteca de STM *TinySTM* [Felber et al. 2008] e a ferramenta de instrumentação *PinTools* [Luk et al. 2005].

O artigo é dividido da seguinte forma: A Seção 2 apresenta os conceitos da *Phase Change Memory*. A Seção 3 apresenta as Memórias Transacionais. A Seção 4 discute o ambiente de simulação e as ferramentas utilizadas no trabalho. A Seção 5 apresenta a metodologia utilizada. A Seção 6 mostra os resultados obtidos. Por fim, a Seção 7 apresenta as conclusões e discute os trabalhos futuros.

2. Phase Change Memory (PCM)

A *Phase Change Memory* (PCM) é uma alternativa como memória principal em uma arquitetura de computadores [Ferreira et al. 2010]. Isto ocorre por sua eficiência em relação ao custo de energia, pois após o armazenamento de um bit não há custo para mantê-lo.

*Bolsista de Iniciação Científica da FAPERGS
†Bolsista de Mestrado da FAPERGS

A PCM utiliza temperatura para induzir mudanças de fase em seu material assim armazenando dados. Para que ocorra um *RESET* é induzido um alto e curto pulso de corrente aumentando a resistividade do material. Este processo gera no material um estado amorfo. Para que ocorra um *SET*, é induzido um moderado e longo pulso de corrente e com a baixa da corrente a resistividade do material é reduzida. Com o resfriamento o material muda para um estado cristalino [Lee et al. 2010]. Os dois estados possuem resistências diferentes, podendo armazenar valores de um bit com precisão.

O problema apresentado na PCM está relacionado ao desgaste de seu material. Com as mudanças de fase do material na hora da escrita, seu tempo de vida é ordens de grandeza menor que outras alternativas. Sua vantagem esta em seu baixo consumo de energia e em seu tempo rápido de leitura.

3. Memórias Transacionais (TM)

Memórias transacionais são mecanismos de sincronização que possuem execuções atômicas e isoladas de partes compartilhadas do código. As TMs são estudadas para no futuro substituírem as sincronizações baseadas em *locks*. As TMs podem ser implementadas em *softwares* (STM), *hardware* (HTM) ou em uma versão híbrida dos dois. Neste artigo, focou-se em STM.

Em uma programação utilizando *Software Transactional Memory* (STM) todo acesso a memória compartilhada é feito dentro de uma transação executada atomicamente. A programação com STM permite que o programador não se preocupe com as aquisições e liberações de *locks*, assim delimitando apenas as seções criticas.

Transação é uma sequência finita de escrita e leitura na memória executada por uma *thread* [Herlihy et al. 1993], e deve satisfazer duas propriedades, as propriedades de isolamento e atomicidade.

4. Ambiente de Simulação

4.1. PinTools

PinTools [Luk et al. 2005] é uma ferramenta de instrumentação dinâmica de programas. Ele foi projetado para fornecer uma funcionalidade onde um código escrito em C ou C++, pode ser inserido em locais específicos de uma execução.

Durante a execução, o código de instrumentação é inserido dinamicamente no executável pelo *PinTools*, diferente de outras ferramentas que inserem código estaticamente. Outra característica é que ele salva e restaura os registradores que são substituídos pelo código inserido para que o aplicativo continue a executar.

4.2. TinySTM

A *TinySTM* [Felber et al. 2008] é uma implementação de memórias transacionais para as linguagens C e C++. Seu algoritmo é baseado em outros algoritmos de TM como o TL2. Ela é uma biblioteca utilizada para escrever aplicativos que usam memórias transacionais para sincronização como substituição aos tradicionais *locks*.

4.3. STAMP

STAMP [Cao Minh et al. 2008] é um conjunto composto por oito *benchmarks* criados para pesquisa de memórias transacionais. Apesar de ter sido desenvolvido para a biblioteca de STM TL2, com algumas modificações disponíveis pode ser utilizado na *TinySTM*.

Os *benchmarks* implementados pelo *STAMP* são: *Bayes, Genome, Intruder, Kmeans, Labyrinth, SSCA2, Vacation e Yada*.

5. Metodologia

Como este artigo é uma extensão de parte do que foi feito em [Teixeira et al. 2014], a metodologia utilizada foi a mesma. O código foi instrumentado utilizando a ferramenta *Pintools*. Para a implementação da instrumentação, foi desenvolvido um programa em C++ que conta cada escrita feita à memória.

6. Resultados

Para cada *benchmark* foram medidos dez execuções em cada cenário de 1, 2, 4, 8, 16, 32, 64 *threads*. No melhor caso a variação dos resultados definida pelo desvio-padrão dividido pela média dos experimentos foi aproximadamente 0%, que ocorreu no experimento *Kmeans* com 1 *thread*, e no pior caso, 6%, que ocorreu no experimento *Kmeans* com 64 *threads*.

Os *benchmarks* mostraram diferentes valores no número de escritas conforme o aumento do número de *threads*. A Tabela 1 apresenta a média dos resultados obtidos nas execuções dos *benchmarks* para cada cenário de *thread*.

Tabela 1. Média de escritas (em milhões)

Benchmarks/Threads	1	2	4	8	16	32	64
Bayes	374	375	375	377	380	383	387
Genome	335	282	274	323	442	675	1.058
Intruder	1.068	1.142	1.292	1.603	2.190	3.234	5.485
Kmeans	1.259	1.282	1.324	1.473	1.782	2.333	3.862
Labyrinth	706	779	896	1.073	1.367	1.774	2.024
SSCA2	254	254	254	254	255	255	255
Vacation	2.184	2.185	2.185	2.188	2.192	2.199	2.222
Yada	1.526	1.793	2.005	2.176	2.369	2.502	3.150

O *benchmark Genome* apresentou uma queda de aproximadamente 16% para o aumento de 1-2 *threads*. Essa diminuição acontece devido ao *benchmark* ser dividido em duas fases, com a primeira sendo composto principalmente de transações que fazem somente leituras. Em contra mão no aumento de 32-64 *threads* houve um aumento de aproximadamente 57%. Assim como o *benchmark Intruder* que apresentou um aumento de aproximadamente 69,6% para o aumento de 32-64 *threads*. Esses aumentos são explicados devido ao alto número de *aborts*, gerados devido ao grande número de *threads* que acabam causando muitos conflitos.

Os *benchmarks Bayes* e *SSCA2* apresentar as menores taxas de crescimento com aumento do número de *threads*. Este comportamento pode ser explicado devido a

implementação dos *benchmarks*, assim como os parâmetros utilizados para os testes. Estes *benchmarks* não são totalmente paralelos, o que justifica menores taxas de crecimento.

Os demais *benchmarks* apresentam uma contínua taxa de crescimento no número de escritas em relação ao aumento de *threads*. Este resultado é esperado, pois aumentando o número de *threads* concorrentes, o número de escritas devem aumentar proporcionalmente, se não houver contenção.

7. Conclusão e Trabalhos Futuros

Neste trabalho foi feito uma análise estendida do comportamento das escritas apresentado em [Teixeira et al. 2014]. Os resultados mostram padrões nas escritas das STMs. Um dos padrões analisados é que, na maioria dos *benchmarks*, com o aumento de *threads* temos um aumento no número de escritas. Entre 32-64 *threads*, notamos na maioria dos *benchmarks*, um aumento maior no número de escritas em relação as outras *threads*.

Para trabalhos futuros pretende-se analisar os padrões de escritas utilizando outras bibliotecas de STM, assim como estender outras abordagens feitas no artigo [Teixeira et al. 2014].

Referências

Cao Minh, C., Chung, J., Kozyrakis, C., and Olukotun, K. (2008). STAMP: Stanford transactional applications for multi-processing. In *IISWC '08: Proceedings of The IEEE International Symposium on Workload Characterization*.

Felber, P., Fetzer, C., and Riegel, T. (2008). 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*, pages 237–246, New York, NY, USA. ACM.

Ferreira, A. P., Zhou, M., Bock, S., Childers, B., Melhem, R., and Mossé, D. (2010). Increasing pcm main memory lifetime. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 914–919, 3001 Leuven, Belgium, Belgium. European Design and Automation Association.

Herlihy, M., Eliot, J., and Moss, B. (1993). Transactional memory: Architectural support for lock-free data structures. In *in Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 289–300.

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. (2010). Phase-change technology and the future of main memory. *IEEE Micro*, 30(1):131–141.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Janapa, V., and Hazelwood, R. K. (2005). Pin: Building customized program analysis tools with dynamic instrumentation. In *In Programming Language Design and Implementation*, pages 190–200. ACM Press.

Teixeira, F., Pilla, M., Du Bois, A., and Mossé, D. (2014). Profiling patterns of bit flipping for software transactional memories. In *26th International Symposium on Computer Architecture and High Performance Computing - SBAC-PAD*.