Phase-Change Memory Aplicada em Memórias Transacionais

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 PIBITI/CNPq

1 INTRODUÇÃO

A programação concorrente é uma área que vem ganhando espaço e importância na Computação devido à popularização das arquiteturas paralelas. Este paradigma de programação permite explorar os múltiplos processadores de forma a melhorar o desempenho de um programa.

Um dos problemas da programação concorrente é a condição de corrida (SILBERSCHATZ, 2000), que consiste na situação em que várias threads ou processos acessam e manipulam os mesmos dados concorrentemente e na qual o resultado da execução depende da ordem específica em que o acesso ocorre.

A parte do programa que contém os dados manipulados concorrentemente é chamada de seção crítica (SILBERSCHATZ, 2000). Para que não ocorra a condição de corrida, são utilizados vários métodos de controle de acesso às mesmas, entre eles as memórias transacionais.

Memórias Transacionais (HERLIHY, 1993) são uma alternativa para a sincronização de *threads*. As memórias transacionais são baseadas em transações de banco de dados, assim tendo como uma vantagem sobre as sincronizações baseadas em *locks* a simplicidade de serem implementadas em *software*. Outra vantagem das memórias transacionais é que como elas executam transações, não existe o problema de *deadlock* que existe na sincronização baseada em *locks* (HARRIS, 2010). A composição de componentes de *softwares* também é mais simples do que em sistemas baseados em *locks*, aumentando a reusabilidade do código (HARRIS, 2010).

Phase Change Memories (PCM) (LEE, 2010) são uma nova alternativa às memórias baseadas em *Dynamic Random Access Memories* (DRAM) utilizadas atualmente como memória principal. Memórias PCM possuem vantagens em termos de consumo de energia, fator cada vez mais importante em grandes *data centers*, e em seu tempo de leitura, que é mais rápido do que uma DDR3. Porém, tempos de escrita são elevados em comparação, visto que o processo de armazenamento de um bit envolve alterar o estado do material da célula de memória em questão. O excesso de escritas em uma posição também pode levar a desgaste prematuro (LEE, 2010).

Nesse trabalho será estudado o impacto do uso de memórias PCM em memórias transacionais, usando a biblioteca *TinySTM* (FELBER, 2008), a ferramenta *PinTools* (LUK, 2005) e o *benchmark* STAMP (CAO MINH, 2008). A principal contribuição do trabalho é avaliar o potencial de impacto das escritas mais lentas de memórias PCM no desempenho de sistemas com memórias transacionais em *software*.

As próximas seções serão apresentadas da seguinte forma: a Seção 2 apresenta a metodologia e as ferramentas utilizadas. A Seção 3 discute os resultados obtidos por meio de simulação. A Seção 4 apresenta as conclusões.

2 METODOLOGIA (MATERIAL E MÉTODOS)

Para a realização do trabalho foi implementado uma biblioteca para simular o tempo de escrita de memórias PCM. Esse simulador foi implementado utilizando a ferramenta *Pintool*s. Para a implementação do simulador foi desenvolvido um programa em C++ que a cada escrita feita à memória era inserido um atraso (*delay*), assim simulando uma memória PCM. O *delay* é inserido devido ao tempo de escrita da memória PCM ser maior que o tempo de leitura.

As memórias DDR3 têm em média 77*ns* de tempo de acesso a memória, para leituras e escritas, enquanto as memórias PCM têm 48*ns* de tempo de leitura e 80*ns* em média de tempo de escrita, isto porque para fazer um *SET* demora mais que fazer um *RESET* (LEE, 2010). Então para calcular o *delay* que iria ser inserido foi feito uma regra de três onde o tempo de leitura da memória PCM foi igualado ao tempo de acesso à memória da DDR3, e foi calculado o tempo de escrita da PCM, o resultado obtido foi de 129*ns*. Para inserir o *delay*, foram inseridas instruções que somam uma variável. Cada instrução inserida tem um tempo aproximadamente de 7*ns*, então foram inseridas 8 instruções, para que o tempo de escrita ficasse o mais aproximado possível do tempo calculado.

Para executar o *benchmark* STAMP foram utilizadas as seguintes ferramentas, a *TinySTM* e a *PinTools*, estas ferramentas e o *benchmark* utilizado serão descritos a seguir.

2.1 *TinySTM*

A *TinySTM* (FELBER, 2008) é uma implementação memórias transacionais para as linguagens C e C++. Seu algoritmo é baseado em outras bibliotecas de TM como o TL2. Ela é uma biblioteca utilizada para escrever aplicativos que usam memórias transacionais para sincronização em substituição dos tradicionais locks. A versão do *TinySTM* utilizada foi a 1.0.3.

2.2 *Pintools*

Pin (LUK, 2005) é uma ferramenta de instrumentação dinâmica de programas. O Pin foi projetado para fornecer uma funcionalidade onde um código escrito em C ou em C++, pode ser inserido em locais específicos de um executável. A versão do Pin utilizada foi a 2.11.

2.3 STAMP *benchmark*

STAMP (CAO MINH, 2008) é um conjunto de *benchmarks* criado para pesquisa de memórias transacionais. O STAMP é composto por oito *benchmarks*. O STAMP é um conjunto de *benchmarks* para uma implementação do TL2 de memórias transacionais, mas com algumas modificações, que estão disponíveis no site do *TinySTM*, ele também pode ser usado para a mesma. A versão do STAMP utilizada foi a 0.9.10.

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

3 RESULTADOS E DISCUSSÃO

Os *benchmarks* foram executados em uma máquina com dois processadores Intel(R) Xeon(R) CPU E5620 @2.40GHz com 12Mb de cache cada. Cada *benchmark* implementado pelo *benchmark* STAMP, foi executado com suas configurações recomendadas para simulação, os *benchmarks Vacation* e *Kmeans* foram executados com suas configurações para simulação com uma alta contenção do código.

Para cada *benchmark* e configuração, foram medidas dez execuções com 1, 2, 4 e 8 *threads* com o simulador implementado e com somente a instrumentação sem aumentar o tempo de escrita, com a intensão de comparar como as memórias transacionais se comportam quando executadas em uma memória PCM. O coeficiente de variação, ou seja no melhor caso a variação dos resultados definida pelo desvio-padrão divido pela média dos experimentos é de 0,2%, que ocorreu no experimento *Bayes* com 4 *threads* com o simulador da memória PCM, e no pior caso, 6%, que ocorreu no experimento *Yada* com 1 *thread* sem o simulador da memória PCM.

A Figura 1 mostra os resultados de desempenho com PCM simulado, em relação à configuração simulada sem PCM. Os *benchmarks* mostraram diferentes características para cada *benchmark*, como pode ser observado no gráfico acima. Cada coluna mostra quantas vezes mais lento a versão simulando PCM é em relação à versão com instrumentação que não aumenta o tempo de escrita.

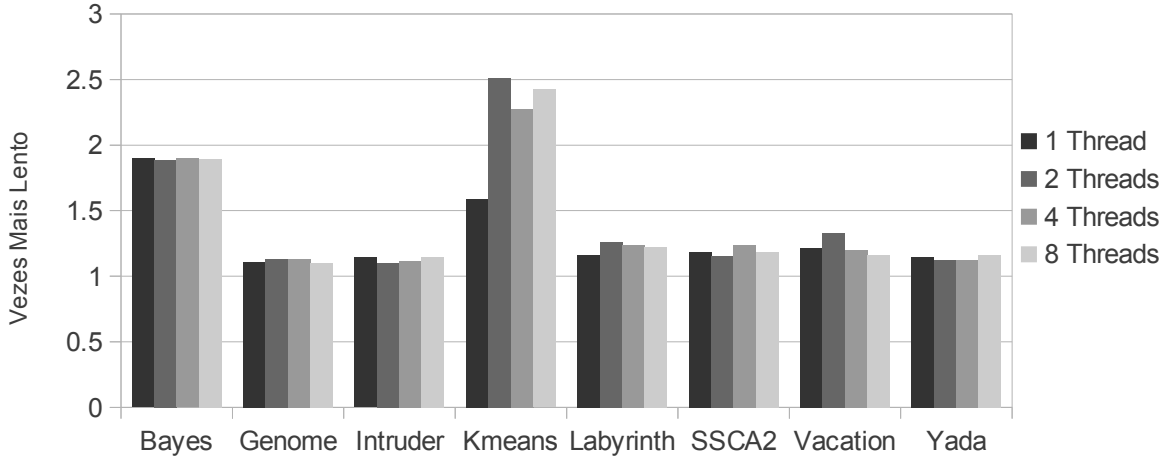

Figura 1: desempenho com PCM relativo à configuração base.

Enquanto os *benchmarks Bayes*, *Genome*, *Intruder* e *Yada* mostraram pouca variação do desempenho em relação ao número de *threads*, outros mostraram uma variação maior. Os *benchmarks* que apresentaram uma maior variação foram o *Kmeans*, *Labyrinth*, *SSCA2* e o *Vacation*. Entre estes se destacou o *benchmark Kmeans*, que apresentou uma variação muito grande no desempenho com os diferentes números de *threads*, tendo com duas *threads* seu pior desempenho.

4 CONCLUSÃO

Neste artigo, o impacto da memória PCM em *benchmarks* de memórias transacionais em *software* foi estudado. Memórias PCM e a sincronização por STMs são tendências futuras e a compreensão das interações entre elas é importante para obter bom desempenho.

Como *benchmark* foi usado o STAMP. Como ele implementa vários *benchmarks*, uma ampla área de aplicação das memórias transacionais é coberta. Com isso foi possível analisar as memórias transacionais em várias situações diferentes, tendo então uma melhor verificação de como uma memória PCM influência às memórias transacionais.

O uso da memória PCM em memórias transacionais mostrou diferentes impactos, tendo mostrado uma regularidade em alguns dos *benchmarks*, enquanto em outros mostrou uma maior variação no desempenho, que pode ser justificada pelo grande tempo de escrita das memórias PCMs e por uma quantidade grande de *aborts* nas transações.

Para os trabalhos futuros, pretende-se verificar o impacto da memória PCM em memórias transacionais levando em conta a probabilidade de dados estarem ou não em determinados pontos da hierarquia de memória. Também pretende se medir o consumo de energia e desenvolver modificações nas bibliotecas de STM para adaptá-las às hierarquias de memória com PCM.

5 REFERÊNCIAS

HARRIS, T.; LARUS, J.; RAJWAR, R. **Transactional memory, 2nd edition**. Synthesis Lectures on Computer Architecture, 5(1):1–263, 2010.

LEE, B. C.; ZHOU, P.; YANG, J.; ZHANG, Y.; ZHAO B.; IPEK, E.; MUTLU O.; BURGER, D. Phase- change technology and the future of main memory, 2010.

HERLIHY, M.; ELIOT, J.; MOSS, B. Transactional memory: Architectural support for lock-free data structures. In in **Proc. of the 20th ISCA**, p. 289–300, 1993.

LUK, C.; COHN, R.; MUTH, R.; PATIL, H.; KLAUSER, A.; LOWNEY, G.; WALLACE, S,; JAPANA, V.; HAZELWOOD, R. Pin: Building customized program analysis tools with dynamic instrumentation. In **Prog. Lang. Design and Impl.**, p. 190–200, 2005.

CAO MINH, C.; CHUNG, J.; KOZYRAKIS, C.; OLUKOTUN, K. STAMP: Stanford transactional applications for multiprocessing. In **IISWC ’08: Proce. of The IEEE Intl. Symposium on Workload Characterization**, p. 35 – 46, Sept. 2008.

SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G. **Sistemas Operacionas: Conceitos e Aplicações.** Rio de Janeiro: Campus, 2000.

FELBER, Pascal; FETZER, Christof; RIEGEL, Torvald. Dynamic performance tuning of word-based software transactional memory. In **PPoPP ’08,** p**.** 237–246, 2008.