Análise Ortogonal dos Versionamentos Implementados pela TinySTM

Felipe Leivas Teixeira[1*]; André Rauber Du Bois[1]; Maurício Lima Pilla[1]

[1]*Universidade Federal de Pelotas – {flteixiera, dubois, pilla}@inf.ufpel.edu.br*

1. INTRODUÇÃO

A programação concorrente é uma área que vem ganhando espaço e importância na computação devido à evolução dos computadores que hoje têm vários processadores. Os vários processadores favorecem a programação concorrente, uma vez que ela pode explorar os processadores para melhorar o desempenho de um programa.

Um dos problemas de programação concorrente é a condição de corrida. Para não ocorrer uma condição de corrida, são utilizados vários métodos de controle de acesso de memória, incluindo Memórias Transacionais (HERLIHY et al., 1993).

Memórias Transacionais (TMs) são uma alternativa para sincronização de *threads*. Elas fornecem uma maior abstração. Todos os acessos a memória compartilhada são executados dentro de transações, que são baseadas nas transações de banco de dados. As TMs oferecem muitas vantagens, tais como prevenção de *deadlocks* e modularidade, aumentando a reúso de código (HARRIS et al., 2010).

TMs implementam versionamento de dados para fazer o gerenciamento das versões dos dados. Isso garante a Atomicidade das transações. Existem dois tipos de versionamento de dados: Versionamento Adiantado e Versionamento Atrasado. Em Memórias Transacionais em Software (STM), o versionamento de dados necessita do *write acquisiton*, que é quando uma transação deve adquiri privilégio de escrita para as posições de memória, e o *write versioning*, que faz o gerenciamento das versões dos dados. O *write acquisiton* e o *write versioning* também tem dois tipos: Adiantado e Atrasado.

TinySTM (FELBER et al., 2008) é uma biblioteca STM. Ela implementa três tipos de versionamento de dados, são eles: *WRITE_THROUGH, WRITE_BACK_ETL e WRITE_BACK_CTL*. Os dois primeiros utilizam o *write acquisition* adiantado e o último utiliza o *write acquisition* atrasado.

Neste trabalho, foram avaliados os versionamentos de dados implementados pela TinySTM. A avaliação consiste em uma comparação das características ortogonais utilizando Eigenbench (MARATHE et al., 2007). O diferencial deste trabalho é que ao contrário de outros trabalhos, é feita a análise e avaliação de desempenho ortogonal dos diferentes versionamentos de dados implementados pela TinySTM.

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

O principal objetivo deste trabalho é caracterizar os diferentes versionamentos de dados implementados pela TinySTM. Para tanto, o *benchmark* Eigenbench foi utilizado. Ele foi utilizado porque ele é um *benchmark* sintético que pode simular as características dos sistemas de transacionais.

*Bolsista de Mestrado da FAPERGS

O Eigenbench faz uma simulação de cenários que podem acontecer com sistemas de transacionais. Para isso, utiliza três *arrays* separadas:

- *Array1*: é chamado de *hot array*, ele é compartilhado por todas *threads* causando conflitos;
- *Array2*: é chamado de *mild array*, ele também é acessado por meio de transações. Cada *thread* acessa sua própria partição de *Array2*, assim, os acessos não causam conflitos;
- *Array3*: é chamado de *cold array*, ele é dividido igual ao Array2 mas não é acessado por transações, ele pode ser acessado dentro de uma transação ou fora.

Os experimentos foram executados em uma máquina com um processador Intel Core i7-2600 com 3.4GHz e três níveis de cache (L3 com 8MB, L2 com 1MB e L1 com 256KB). O processador possui 4 núcleos e 8 *threads* com *HyperThreading*. Foi usado o sistema de operação GNU/Linux Ubuntu 12.04 LTS 64-bits. Todos os experimentos com Eigenbench foram executados 30 vezes para cada implementação de versionamento de dados. A maioria deles foram executados com 8 *threads* e sequencial. O experimento que medida que a escalabilidade foi executado com 1, 2, 4 e 8 *threads* e sequencial. Os parâmetros utilizados podem ser visto na Tabela 1.

Tabela 1: Tabela dos parâmetros de cada experimento

	N	**A1**	**A2**	**A3**	**R1**	**W1**	**R2**	**W2**	**R3i**	**R3o**	**W3(i/o)**	**Ict**
Default	8	0	32k	0	0	0	90	10	0	0	0	0
Contetion	-	Var	Var	-	45	5	45	5	-	-	-	-
Scalability	Var	-	-	-	-	-	-	-	-	-	-	-
Density	-	512	31k	512	8	2	82	8	Var	Var	-	-
T. Lenght	-	-	-	-	-	-	Var	Var	-	-	-	-
T. Locality	-	-	-	-	-	-	-	-	-	-	-	Var
Pollution	-	-	-	-	-	-	Var	-	-	-	-	-
Predom.	-	-	16k	16k	-	-	-	-	-	Var	-	-
Work-set	-	-	Var	-	-	-	-	-	-	-	-	-

Var = Variável;
R/W = Leitura/Escrita;
A = *Array;*

3. RESULTADOS E DISCUSSÃO

As siglas utilizadas para identificar os diferentes versionamentos é a seguinte:

- **Tiny-B_ETL**: Indica o versionamento de dados *WRITE_BACK_ELT* versionamento com *write acquisition* adiantado e *write versioning* atrasado;
- **Tiny-B_CTL**: Indica o versionamento de dados *WRITE_BACK_CLT* versionamento com *write acquisition* atrasado e *write versioning* atrasado;
- **Tiny-T**: Indica o versionamento de dados *WRITE_THROUGH* versionamento com *write acquisition* adiantado e *write versioning* adiantado;

- **Unp**: Indica a execução sem uso de nenhuma proteção à memória.

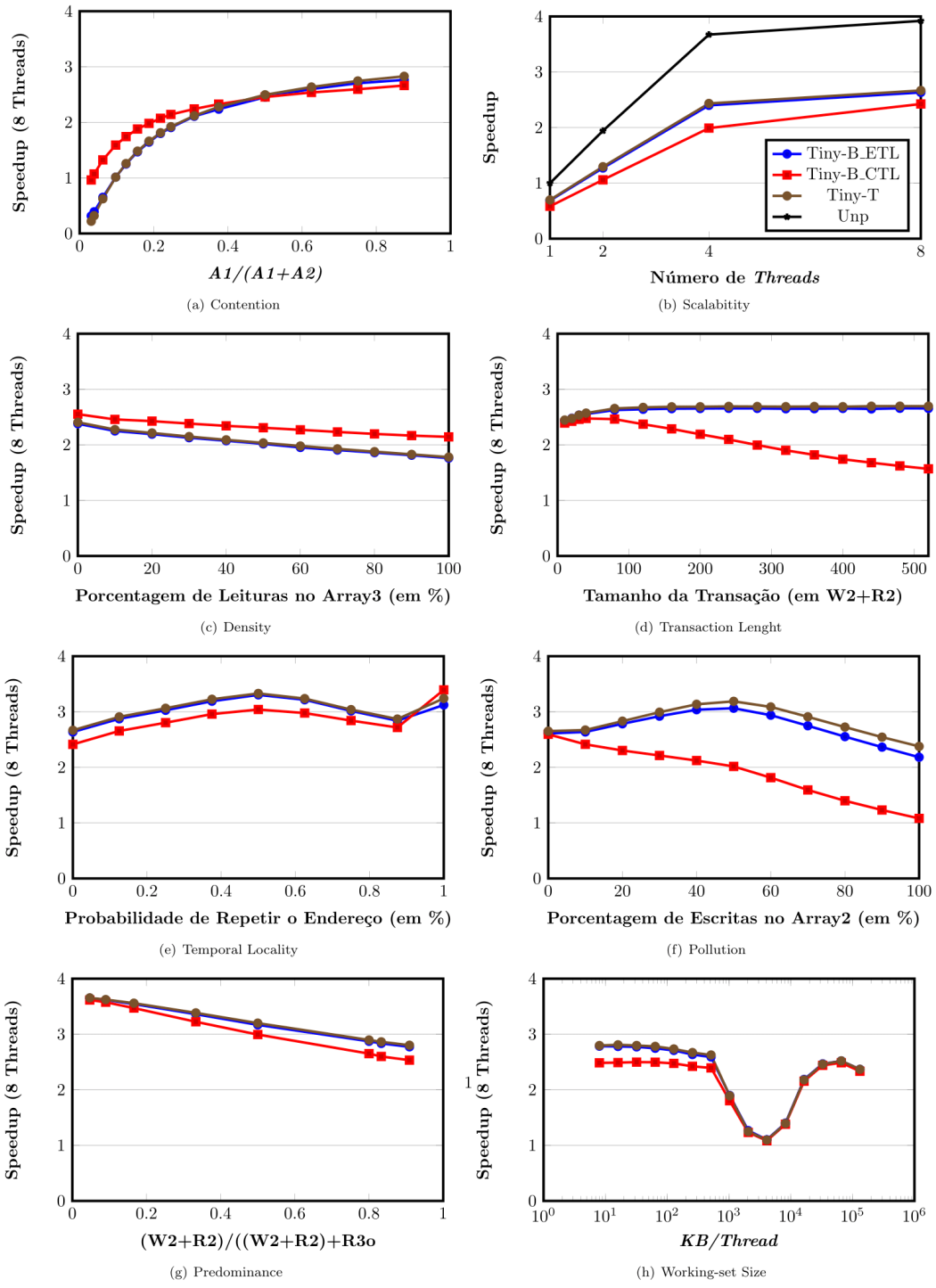

Figura 1: Resultados dos experimentos com *Benchmark* Eigenbench, para os três Versionamentos

A Figura 1 apresenta os resultados dos experimentos. Os resultados mostraram que o versionamento *WRITE_THROUGH* mostrou ser o melhor na maioria dos experimentos, isto porque ele escrever diretamente na memória, fazendo com que o *commit-time* seja rápido em ambientes com pouca contenção que é o cenário da maioria dos experimentos.

O versionamento *WRITE_BACK_ETL* teve um comportamento muito semelhante com ao versionamento *WRITE_THROUGH*, mas com um desempenho um pouco inferior, devido a sobrecarga no *commit-time*. Visto que ele utiliza *write acquisition* atrasado.

Por outro lado, o versionamento *WRITE_BACK_CTL* apresentou o pior desempenho na maioria dos experimentos, devido ao alto custo de abortar transações que já fizeram muito trabalho. Mas, em cenários com alta contenção ele se mostrou o melhor, sendo até 4,3 vezes mais rápido do que o versionamento *WRITE_THROUGH* como pode ser visto na Figura 1(a).

4. CONCLUSÕES

Neste trabalho nos avaliamos os versionamentos de dados implementados pela TinySTM. A avaliação consistiu em uma comparação das características ortogonais utilizando o *benchmark* Eigenbench.

Os resultados mostraram que o versionamento *WRITE_THROUGH* apresentou o melhor resultado na maioria dos experimentos e que *WRITE_BACK_ETL* teve um comportamento semelhante, mas perdendo em algumas características devido a sobrecarga no *commit-time*. O versionamento *WRITE_BACK_CTL* mostrou ser o melhor para cenários com alta contenção devido aos versionamentos de dados que utilizam o *write acquisition* adiantado são abortados mais vezes, causando uma perda de desempenho. Também pode ser visto que um *abort* com o versionamento *WRITE_BACK_CTL* é mais custoso devido o *abort* acontecer apenas no *commit-time*. Em geral a análise ortogonal mostrou que o maior diferencial sobre os resultados foi o *write acquisition* onde os versionamentos que utilizam o *write acquisition* adiantado tem melhor desempenho em cenários com baixa contenção enquanto o versionamento que utiliza o *write acquisition* atrasado tem melhor desempenho em cenários com alta contenção.

5. REFERÊNCIAS BIBLIOGRÁFICAS

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.

MARATHE, V.J., MOIR, M.: Efficient nonblocking software transactional memory. In: **Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP ’07**, pp. 136–137. ACM, New York, NY, USA (2007). DOI 10.1145/1229428.1229454.