Estudo de Instruções e Desempenho para Implementação de Memória Transacional em Hardware

FREDERICO PEIXOTO ANTUNES[1]; FERNANDO EMILIO PUNTEL[2]; GERSON GERALDO H. CAVALHEIRO[3], ANDRE RAUBER DU BOIS[4]

[1]Universidade Federal de Pelotas – fpantunes@inf.ufpel.edu.br
[2]Universidade Federal de Pelotas – fepuntel@inf.ufpel.edu.br
[3]Universidade Federal de Pelotas – gerson.cavalheiro@inf.ufpel.edu.br
[4]Universidade Federal de Pelotas – dubois@inf.ufpel.edu.br

1. INTRODUÇÃO

O modelo de Memória Transacional (TM) (HARRIS; LARUS; RAJWAR, 2010) foi proposto na década de 1990 como um modelo para suportar acessos concorrentes a dados por threads independentes em um programa concorrente. Este modelo, em alternativa aos mecanismos clássicos de sincronização baseados em seção crítica, caracteriza-se por ser não bloqueante, ou seja, a execução não é interrompida esperando o resultado de um espaço de memória compartilhado.

Memórias Transacionais funcionam tendo um trecho de código atômico, chamado de transação, que ao ser executado armazena localmente todas modificações que fez na memória e verifica se houve conflitos. Assim, caso seja verificado algum acesso concorrente aos dados manipulados por alguma transação, ou seja, seja constatada a ocorrência de conflitos de acesso, a transação sofre uma operação de *abort*, sendo computação já realizada descartada e a transação refeita. Quando nenhum conflito é identificado, a transação é aceita, sendo realizada uma operação denominada *commit*. Com o *commit* realizado, as modificações armazenadas localmente à transação são tornadas públicas e adicionadas na memória compartilhada.

Memórias Transacionais podem ser implementadas em software (Memórias Transacionais em Software - STM) ou em hardware (Memórias Transacionais em Hardware - HTM). A STM é instanciada por um mecanismo de controle de concorrência que permite a criação de transações e que mantém estruturas de controle que permitem, por instrumentação das operações leitura e escrita de dados em memória para verificar conflitos e tomar medidas para aceitar ou abortar a execução de transações. A HTM, por sua vez, é um mecanismo implementado diretamente sobre recursos de hardware disponibilizados para este fim específico. Nem toda a arquitetura de processador disponibiliza o conjunto de instruções necessárias para implementação de uma HTM, no entanto, uma vez disponível, é possível realizar a implementação de uma infraestrutura para Memória Transacional com menor sobrecusto que aquela oferecida por um suporte em software.

O presente trabalho tem como objetivo identificar o potencial das instruções em hardware TSX (Transactional Synchronization Extension) para construção de uma biblioteca com suporte à HTM. Dentro dos objetivos, busca-se obter uma análise da diferença de desempenho entre uma implementação de HTM e uma implementação de STM amplamente aceita pela comunidade da área, a TinySTM. Os resultados obtidos permitirão aprofundar o conhecimento do grupo de pesquisa sobre o potencial do uso de Memória Transacional em hardware e oferecer novas possibilidades de investigações sobre o seu funcionamento.

2. METODOLOGIA

Para construir a comparação entre os métodos de HTM e STM foram utilizados a ferramenta TinySTM (FELBER; FETZER; RIEGEL, 2008), que fornece as instruções de TM já implementadas em software, e uma implementação própria de HTM desenvolvida a partir de uma implementação disponibilizada em um repositório público (https://github.com/riclas/htm-stamp). O benchmark utilizado para testar ambas implementações foi a benchmark STAMP (MINH et al., 2008), e os resultados finais obtidos passaram por um programa para fazer a análise estatística dos mesmos.

A ferramenta TinySTM é uma das duas implementações de STM mais populares, ao lado da SwissTM (DRAGOJEVIĆ; GUERRAOUI; KAPALKA, 2009), sendo ela menor e mais simples que a segunda, com a desvantagem de não possuir o mesmo nível de robustez. Já o modelo HTM não possui implementações vastamente populares, e portanto uma versão modificada de uma implementação existente foi feita. A implementação de HTM é feita utilizando o conjunto de instruções fornecidas pelo processador, sendo neste caso as instruções TSX (Transactional Synchronization Extension) da Intel.

Com a necessidade de se ter um ambiente o mais consistente possível para a comparação entre os modelos, optou-se por utilizar um benchmark padrão para aplicações com Memória Transacional: o benchmark STAMP (Stanford Transactional Applications for Multi-Processing). Este benchmark consiste em uma coleção de aplicações selecionadas por serem apropriadas para testes com memória transacional e que permite utilizar o mesmo conjunto de algoritmos sem diferenças na sua implementação para avaliar diferentes aspectos das TMs. Também para se manter essa homogeneidade é necessário que os modelos sejam executados no mesmo ambiente de execução, porém o conjunto de instruções TSX só são encontrados em alguns poucos processadores da Intel, e portanto para ter acesso a ele foi necessário utilizar as máquinas na nuvem Intel Devcloud para a execução dos programas, possuindo o grande ponto negativo de que nestes servidores cloud não há como garantir que os mesmos recursos são fornecidos consistentemente durante toda a execução.

Tendo a implementação dos benchmarks adaptada com os dois modelos de TM foram feitas, então, 30 execuções de cada programa em cada um dos modelos, e os resultados obtidos foram compilados e tratados para serem usados em um programa de análise estatística desenvolvido por um dos membros do grupos de pesquisa do LUPS (Laboratory of Ubiquitous and Parallel Systems). Essa análise estatística consiste da verificação se os dados obedecem a distribuição normal, e a testagem com os testes de Kolmogorov-Smirnov (MASSEY JR, 1951) e T de Student (STUDENT, 1908).

3. RESULTADOS E DISCUSSÃO

Para se chegar aos resultados obtidos e aqui apresentados foi implementada uma primeira versão de uma biblioteca de serviços de HTM e uma adaptação do benchmark STAMP para executar com ela. Foi-se, então, automatizada a execução de todos os programas do benchmark múltiplas vezes para se obter e armazenar os resultados de uma forma que possibilitasse ser feito um tratamento desses dados para serem usados no algoritmo de análise

estatística. Este tratamento de dados foi implementado em Python, utilizando a plataforma Jupyter Notebook.

O tratamento dos resultados de desempenho aplica diferentes métodos estatísticos, incluindo o teste Kolmogorov-Smirnov, Teste T de Student, Teste U de Mann-Whitney e o Teste de Shapiro-Wilk para validar os dados coletados. Dentre todos os programas do benchmark, Yada e Intruder foram os únicos que não apresentaram resultados que obedeciam a distribuição normal e, portanto, para os intuitos deste trabalho, foram removidos da análise para que fosse possível fazer a comparação dos resultados com um nível de 95% de confiança. Os resultados ainda foram divididos em resultados de baixa contenção e alta contenção, sendo essa divisão referente aos casos dos benchmarks Kmeans e Vacation que possuíam valores de parâmetros recomendados nas suas instruções tanto para baixa contenção (valores para execução menos exigente) e para alta contenção (para execução mais exigente).

Na Tabela 1 é apresentado o relatório gerado no programa de análise estatística. Nele é possível ver que todas as amostras foram consideradas boas pelo teste de Kolmogorov-Smirnov, mas principalmente, que as médias de tempo de execução em STM foram 1,99 e 1,93 vezes mais demoradas que em HTM, para baixa contenção e alta contenção respectivamente, demonstrando uma superioridade da memória transacional em hardware nos testes feitos com um ganho de desempenho de quase 100%.

Tabela 1 - Relatório das análises estatísticas dos dados de desempenho.

	HLE Low Contention	TinySTM Low Contention	HLE High Contention	TinySTM High Contention
count	30,00000000	30,00000000	30,00000000	30,00000000
mean	6,64593523	13,2423894	9,18916263	17,8161061
std	0,30887221	0,51445374	0,57926916	0,60152534
error (95%)	0,09581733	0,15959216	0,17969899	0,18660323
min	6,07867100	12,54297500	8,36114000	16,68868100
25%	6,44376700	12,86142500	8,71698200	17,32480400
50%	6,59554100	13,14351400	9,02933400	17,75844900
75%	6,75660600	13,42185800	9,34438400	18,10825200
max	7,36308700	15,07882200	11,02221500	19,45616900
KS stat	0,09953153	0,10741320	0,15620677	0,07885333
KS p	0,89908089	0,84319068	0,41444587	0,98484831
KS p>0,05	Sample OK	Sample OK	Sample OK	Sample OK
SW stat	0,97141069	0,88969457	0,90882558	0,97171658
SW p	0,57845724	0,00474438	0,01390105	0,58713734
SW p>0,05	Sample OK	Sample Bad	Sample Bad	Sample OK

Portanto no presente estado do trabalho possui-se uma implementação de HTM que ainda deve ser revisada para verificar possíveis melhorias, além de também existir a possibilidade de se testar a implementação de HTM com o conjunto de intrínsecas RTM (Restricted Transactional Memory). Ademais, o

trabalho no momento somente possui testes de execução em servidores na nuvem, sendo necessário em uma próxima etapa serem feitos testes em uma máquina específica onde poderá se ter um maior conhecimento dos recursos disponíveis e sendo utilizados durante a execução.

A implementação feita para o trabalho e os resultados obtidos e descritos aqui podem ser encontrados no github (https://github.com/hpfred/HTM-STamp_Modified).

4. CONCLUSÕES

Com o que foi levantado neste trabalho foi possível trazer novos conhecimentos sobre a área de memória transacional para o grupo de pesquisa do LUPS e com os resultados obtidos também foi possível se ter algum indício otimista para a possibilidade das memórias transacionais em hardware, em especial tendo o conhecimento que TMs são ideais para alguns usos específicos e já são usadas nessas áreas, como é o caso de ordenação de bancos de dados (LEIS; KEMPER; NEUMANN, 2014).

5. REFERÊNCIAS BIBLIOGRÁFICAS

HARRIS, T.; LARUS, J.; RAJWAR, R. Transactional memory. **Synthesis Lectures on Computer Architecture**, v.5, n.1, p.1–263, 2010.

RIGO, S.; CENTODUCATTE, P.; BALDASSIN, A. Memórias transacionais: Uma nova alternativa para programação concorrente. In: **MINICURSOS DO VIII WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO**, WSCAD, 2007. **Anais...** Gramado, 2007.

FELBER, P.; FETZER, C.; RIEGEL, T. Dynamic performance tuning of word-based software transactional memory. In: **ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING**, 13., 2008. **Proceedings...**, 2008. p.237–246.

DRAGOJEVIĆ, A.; GUERRAOUI, R.; KAPALKA, M. Stretching transactional memory. **ACM sigplan notices**, v.44, n.6, p.155–165, 2009.

MINH, C. C.; CHUNG, J.; KOZYRAKIS, C.; OLUKOTUN, K. STAMP: Stanford transactional applications for multi-processing. In: **IEEE INTERNATIONAL SYMPOSIUM ON WORKLOAD CHARACTERIZATION**, 2008. **Proceedings...** Seattle, 2008. p.35–46.

MASSEY JR, F. J. The Kolmogorov-Smirnov test for goodness of fit. **Journal of the American statistical Association**, v.46, n.253, p.68–78, 1951.

STUDENT. The probable error of a mean. **Biometrika**, p.1–25, 1908.

LEIS, V.; KEMPER, A.; NEUMANN, T. Exploiting hardware transactional memory in main-memory databases. In: **IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING**, 30., 2014. **Proceedings…** Chicago, 2014. p.580-591.