AVALIAÇÃO DO CONSUMO DE ENERGIA E DESEMPENHO DE MEMÓRIAS TRANSACIONAIS EM SOFTWARE

TIMÓTEO MATTHIES RICO[1]; RODRIGO MEDEIROS DUARTE[1]; MAURÍCIO LIMA PILLA[1]; ANDRÉ RAUBER DU BOIS[1]

[1]*Universidade Federal de Pelotas- {tmrico, rmduarte, pilla, dubois}@inf.ufpel.edu.br*

1. INTRODUÇÃO

Com o advento de processadores *multicore*, a utilização de programação paralela no desenvolvimento de *software* é fundamental na melhoria da eficiência, por fazer melhor proveito dos recursos de *hardwares* disponíveis e assim aumentar o desempenho das aplicações (MÓR, ALVES, *et al.*, 2010). Um modelo de programação paralela muito utilizado é o baseado em memória compartilhada, a qual é usada como meio de comunicação entre as execuções concorrentes. Neste modelo, os acessos concorrentes aos recursos compartilhados devem ser sincronizados de modo a garantir a consistência e integridade do sistema (ANDREWS, 2000).

Os mecanismos de sincronização têm sido tradicionalmente implementados através de métodos baseados em *locks* explícitos no código fonte dos programas. Embora muito utilizada, esta abordagem é propensa a erros, e pode apresentar uma série de complicações tais como instabilidade no sistema, dificuldade de composição e baixo desempenho (MCKENNEY, MICHAEL e WALPOLE, 2007).

Um novo mecanismo de sincronização, denominado Memória Transacional (TM), foi desenvolvido com objetivos de reduzir as dificuldades e limitações encontradas em tradicionais métodos de sincronização (BALDASSIN, 2009). Este mecanismo permite uma programação mais simples, pois o programador apenas precisa especificar quais serão os blocos de código que devem ser executados de forma atômica e isolada (formando uma transação), deixando para o sistema de execução transacional a obrigação de implementar eficientemente a sincronização em baixo nível.

O sistema transacional garante a atomicidade e isolamento entre as execuções concorrentes baseado em dois conceitos chaves: *(i)* detecção/resolução de conflitos e *(ii)* versionamento de dados (BALDASSIN, 2009). Resumidamente, há ocorrência de conflitos entre transações concorrentes quando no mesmo intervalo de tempo duas ou mais transações acessam o mesmo dado compartilhado e ao menos um dos acessos é de escrita. Em caso de conflito, um componente do sistema transacional, denominado gerenciador de contenção, é invocado para resolver os conflitos entre as transações e assim garantir o progresso do sistema. Este componente implementa uma ou mais políticas de resolução de conflitos, as quais ditam caso deva-se abortar a transação que detectou o conflito, as outras transações conflitantes, e caso atrasar ou não a reexecução das transações abortadas. O versionamento de dados lida com o gerenciamento das diferentes versões dos dados (originais e especulativas) manipulados por uma transação. Em casos de efetivação da transação, os dados especulativos são armazenados na memória, caso contrário permanecem os originais e a transação é reexecutada.

É inegável a importância de técnicas para redução do consumo de energia em sistemas embarcados e, atualmente, esta tendência se espalhou também para *data centers* e sistemas *desktops*. Porém, a maioria das avaliações de

Memórias Transacionais são guiadas exclusivamente pelo desempenho: quanto mais transações processadas por unidade de tempo, melhor é o sistema (BALDASSIN, 2009).

Nesse contexto, este trabalho tem como foco analisar o consumo de energia e desempenho de diferentes implementações de Memórias Transacionais em *Software* (STM). Tem-se como principal objetivo identificar a implementação mais eficiente de STM em termos de consumo de energia e desempenho para diferentes cenários de execução.

2. MATERIAL E MÉTODOS

Para identificar a configuração de STM mais eficiente, três implementações foram utilizadas: TL2 (versão 0.9.6) (DICE, SHALEV e SHAVIT, 2006), TinySTM (versão 1.0.3) (FELBER, FETZER e RIEGEL, 2008) e SwissTM (versão 2011-08-15) (DRAGOJEVIC, GUERRAOUI e KAPALKA, 2009). Estas implementações diferem-se principalmente pelas estratégias de versionamento de dados, detecção e resolução de conflitos empregadas.

A TL2 utiliza versionamento de dados e detecção de conflitos em modo atrasado, e gerenciador de contenção tímido, que aborta prontamente a transação que detectou o conflito. A TinySTM emprega versionamento de dados de forma atrasada, detecção de conflitos adiantado, e gerenciador de contenção tímido. A SwissTM utiliza versionamento de dados atrasado, e emprega uma abordagem mista de detecção de conflitos e gerenciador de contenção. A detecção de conflitos em transações de escrita/escrita é feita de modo adiantado. Em transações de leitura/escrita o sistema transacional detecta conflitos de forma atrasada. O gerenciador de contenção usa o esquema tímido para transações curtas ou somente leitura. Em transações complexas, o gerenciador de contenção dá prioridade para a transação mais antiga (guloso).

Escolheu-se estas três implementações por comporem o estado-da-arte em STM e apresentarem características diferentes no projeto do sistema transacional (DRAGOJEVIC, GUERRAOUI e KAPALKA, 2009).

Para avaliar as três implementações de STM foram utilizadas duas aplicações (*Intruder* e *Vacation*) do *benchmark* STAMP (versão 0.9.10) (CAO MINH, CHUNG, *et al.*, 2008). As características das aplicações do STAMP, no que influencia a execução de aplicações utilizando Memórias Transacionais são: tamanho da transação, conjunto de leitura e escrita, tempo em transação e a taxa de contenção. Especificamente, a aplicação *Intruder* apresenta um cenário com transações curtas, conjunto de leitura e escrita médio, tempo em transação médio e alta contenção. A aplicação *Vacation* representa um cenário de execução com transações médias, conjunto de leitura e escrita médio, tempo em transação alto e taxas de contenção baixa ou média/alta, de acordo com a configuração.

A medição do consumo de energia (Joule) baseou-se nos dados coletados em um microcontrolador especializado embutido na placa-mãe, presente na maioria dos servidores, denominado *Baseboard Management Controller* (BMC). O desempenho (tempo de execução em segundos) foi mensurado pelo resultado de saída de execução do *benchmark* STAMP.

Os testes foram realizados no sistema computacional Intel Xeon E5620, sistema operacional Suse Linux SP11 e G++ 4.5.2, utilizando-se de 1 a 8 *threads*. Cada configuração de execução (STM, aplicação e quantidade de *threads*) foi executada 10 vezes, e as médias foram calculadas.

3. RESULTADOS E DISCUSSÕES

Com base nos resultados obtidos, mostrados nas Tabelas 1, 2 e 3, as seguintes constatações podem ser feitas.

Observa-se que a TL2 na aplicação *Vacation* sob baixa contenção, conforme incrementa-se o número de *threads*, o desempenho e consumo de energia tornam-se mais eficientes, quase assemelhando-se aos resultados apresentados pela TinySTM e SwissTM. No entanto, nas aplicações *Vacation* sob média/alta contenção e *Intruder*, a TL2 não apresentou bons resultados, no pior caso (*Intruder* 8 *threads*) sendo cerca de 3x menos eficiente em relação ao desempenho e consumo de energia da SwissTM.

Tabela 1 - Aplicação *Vacation* baixa contenção

STM	1 *thread*		2 *threads*		4 *threads*		8 *threads*	
	tempo	energia	tempo	energia	tempo	energia	tempo	energia
TL2	39,785	5331,239	23,074	3262,206	11,804	1784,375	6,452	1057,488
TinySTM	27,630	3754,407	17,570	2532,985	**9,108**	1385,683	5,025	807,814
SwissTM	**26,674**	**3621,224**	**17,559**	**2483,895**	9,127	**1373,259**	**4,915**	**789,989**

Tabela 2 - Aplicação *Vacation* média/alta contenção

STM	1 *thread*		2 *threads*		4 *threads*		8 *threads*	
	tempo	energia	tempo	energia	tempo	energia	tempo	energia
TL2	72,124	9825,466	40,423	5836,170	20,741	3203,228	11,828	1977,967
TinySTM	37,820	**5055,174**	24,044	**3423,377**	**12,464**	**1912,906**	6,862	**1130,528**
SwissTM	**37,005**	5079,920	**24,039**	3463,403	12,578	1921,598	**6,837**	1140,439

Tabela 3 - Aplicação *Intruder*

STM	1 *thread*		2 *threads*		4 *threads*		8 *threads*	
	tempo	energia	tempo	energia	tempo	energia	tempo	energia
TL2	47,106	6465,350	36,326	5248,230	26,622	4164,365	23,942	4295,360
TinySTM	31,638	4306,716	21,154	2964,572	12,259	1850,793	8,434	**1354,250**
SwissTM	**25,627**	**3484,790**	**17,781**	**2558,977**	**10,705**	**1645,275**	**8,075**	1358,816

Em relação à aplicação *Intruder*, observou-se que as STMs obtiveram pouca melhora no desempenho na passagem de 4 para 8 *threads*. Esse comportamento é devido à alta contenção presente nesta aplicação. Conforme aumenta-se o número de *threads*, maior será a taxa de conflitos entre as transações, degradando consequentemente o desempenho e o consumo de energia, pois maior será o desperdício de processamento devido a reexecução das transações canceladas.

De modo geral, a SwissTM foi a implementação que apresentou melhores resultados, tanto em termos de consumo de energia quanto desempenho. Na aplicação *Vacation* (baixa e média/alta contenção), a TinySTM apresentou resultados similares a SwissTM. No entanto, na aplicação *Intruder* para até 4 *threads*, a TinySTM não apresentou bons resultados em relação à SwissTM, sendo cerca de 20% menos eficiente no consumo de energia e desempenho.

4. CONCLUSÕES

A análise do consumo de energia de implementações de STM tem sido pouco explorada. E, quando realizadas, são baseadas em ambientes computacionais simulados. Assim, este trabalho buscou suprir a deficiência na literatura, apresentando em especial a análise do consumo de energia de três

implementações que compõem o estado-da-arte em STM, realizando-a em um ambiente computacional não-simulado.

As implementações TL2, TinySTM e SwissTM, foram utilizadas neste trabalho em seu modo de operação padrão. No entanto, estas permitem em sua instalação configurar alguns parâmetros, como por exemplo, as estratégias adotadas para versionamento de dados e detecção/resolução de conflitos. Nesta perspectiva, tem-se como trabalhos futuros, analisar as mesmas implementações de STM utilizadas neste trabalho variando os parâmetros de instalação mencionados acima, calcular o desvio padrão em relação à média, e também utilizar mais aplicações do *benchmark* STAMP, contemplando assim um maior número de cenários de execução e configurações do sistema transacional.

5. AGRADECIMENTOS

Agradecemos aos projetos PRONEX/FAPERGS/CNPq GREEN-GRID Computação de Alto Desempenho Sustentável e Composição de Ações Transacionais (Pesquisador Gaúcho/FAPERGS) pelo apoio nessa pesquisa.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ANDREWS, G. R. **Foundations of Multithreaded, Parallel, and Distributed Programming**. Boston, MA, USA: Addison Wesley, 2000.

BALDASSIN, A. J. **Explorando Memória Transacional em Software nos Contextos de Arquiteturas Assimétricas, Jogos Computacionais e Consumo de Energia**. Tese (Doutorado em Ciência da Computação), Universidade Estadual de Campinas, Campinas, SP, Brasil, 2009.

CAO MINH, C. et al. STAMP: Stanford Transactional Applications for Multi-Processing. **In: IISWC '08: PROCEEDINGS OF THE IEEE INTERNATIONAL SYMPOSIUM ON WORKLOAD CHARACTERIZATION**, 2008.

DICE, D.; SHALEV, O.; SHAVIT, N. Transactional Locking II. **In: INTERNATIONAL SYMPOSIUM ON DISTRIBUTED COMPUTING**, 2006.

DRAGOJEVIC, A.; GUERRAOUI, R.; KAPALKA, M. Stretching transactional memory. **In: ACM SIGPLAN CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION**, New York, NY, USA, 2009.

FELBER, P.; FETZER, C.; RIEGEL, T. Dynamic performance tuning of wordbased software transactional memory. **In: ACM SIGPLAN SYMP. ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING**, New York, NY, USA, 2008.

MCKENNEY, P. E.; MICHAEL, M. M.; WALPOLE., J. Why The Grass May Not Be Greener On The Other Side: A Comparison of Locking vs. Transactional Memory. **4th ACM SIGOPS W. on Prog. Languages and Operating Systems**, 2007.

MÓR, S. D. K. et al. Eficiência Energética em Computação de Alto Desempenho: uma abordagem em arquitetura e programação para green computing. **In: XXXVII SEMISH**, Belo Horizonte, MG, Brasil, 2010.