DESEMPENHO COMPUTACIONAL E ENERGÉTICO DAS MEMÓRIAS TRANSACIONAIS

DUARTE, Rodrigo M.[1]; DU BOIS, André R.[2]; CAVALHEIRO, Gerson G. H.[2]; PILLA, Maurício Lima.[2]

[1]Engenharia de Computação-Universidade Federal de Pelotas; [2]Universidade Federal de Pelotas, CDTec.
<rmduarte, dubois, gerson.cavalheiro, pilla>@inf.ufpel.edu.br.

1 INTRODUÇÃO

Processadores de arquitetura *multi-core* hoje são uma realidade. Se antes eram usados apenas em servidores de grande porte, agora estão presentes na grande maioria dos computadores, sejam eles desktops ou laptops. Para se conseguir extrair o máximo desempenho desta nova arquitetura, torna-se necessário que as novas aplicações desenvolvidas agora sejam concorrentes ou paralelas, para permitir que as tarefas sejam distribuídas entre os *cores* existentes.

Maquinas *multi-core* são programadas geralmente utilizando o modelo de memória compartilhada, onde varias *threads* utilizam uma área de memória para comunicação. Para evitar que *threads* interfiram de forma errada no trabalho uma das outras, torna-se necessário uma forma de proteger ou sincronizar o acesso a uma região crítica. Tradicionalmente as linguagens de programação fornecem mecanismo como *locks* ou *mutexs*, porém este modelo de programação é difícil e propenso a erros (RAJWAR. 2008, LEE. 2006, JONES.2007, RIGO. 2007).

Devido a problemas apresentados na utilização de *locks,* novos modelos de programação paralela tem surgido, com o objetivo de facilitar a programação.

Este trabalho de pesquisa tem como objetivo verificar o desempenho computacional do modelo de programação paralela conhecido como memória transacional, comparando-o com o tradicional usando *locks*. Também é analisado o consumo de energia no uso destes recursos frente aos métodos tradicionais.

2 METODOLOGIA (MATERIAL E MÉTODOS)

2.1 Os problemas do modelo de programação usando locks:

Locks como método de sincronismo entre *threads* deixa a programação difícil e propensa a erros (RAJWAR. 2008, LEE. 2006, JONES.2007, RIGO. 2007). Entre os problemas apresentados pelos *locks*, são mais comuns a ocorrência de gargalos seriais, *deadlocks* e a difícil reaproveitamento de código.

Um problema comum de programação utilizando *locks* e proteger grandes blocos de código por *locks*. Uma região crítica protegida por um bloqueio permite o acesso à somente uma *thread* por vez, o que acaba gerando ocorrência de gargalos seriais. Este tipo de problema acaba reduzindo o nível de paralelismo.

Na tentativa de evitar o gargalo serial, tende-se a usar um maior número de bloqueios, ou seja, fatorar ainda mais o código. Porem está técnica deixa o código ainda mais complexo e propenso a instabilidades como a ocorrência de *deadlocks.* Ou ainda, pode ocorrer do programador adquirir um número menor do que o necessário de *locks*, o que pode levar a erros de condição de corrida, que são de ocorrência imprevisível e de difícil depuração (TENENBAU. 2006).

2.2 Memória Transacional:

O modelo de programação paralela usando memória transacional é baseado no mesmo conceito de transação presente em banco de dados (RIGO. 2007). Essas operações com a memória são atômicas, ou seja, são executadas por completo ou não são executadas. Nessas transações, os acessos a memória são armazenados em um *log* que é verificado quando o bloco de transação finaliza. Se neste *log* há uma visão consistente da memória, então as alterações são efetivadas, caso contrário, são abortadas.

Memórias transacionais proporcionam que se explore mais paralelismo, aumentando o desempenho e escalabilidade. O uso deste também facilita a programação *multithreading* porque o programador não precisa mais se preocupar com problemas de sincronização, e.g., *deadlocks*, pois todo o controle de acesso a memória compartilhada é feito automaticamente pelo sistema transacional.

2.3 STM-Haskell:

Para a realização deste trabalho, foi utilizado um benchmark escrito usando a linguagem funcional Haskell com uma extensão da linguagem para fornecer a abstração das memórias transacionais, conhecida como STM-Haskell. Haskell é ideal para implementar memórias transacionais por dois principais motivos (HARRIS. 2008): o sistema de tipos consegue separar as ações que possuem efeito colateral das que não tem e grande parte das computações são puras, ou seja, não apresentam efeitos colaterais. Como esse tipo de ação pura não modifica a memória, ela não precisa ser tratada pelo sistema transacional. Logo estas ações nunca precisam ser desfeitas, elas simplesmente podem ser repetidas, caso uma a transação seja abortada.

O sistema de tipos também garante que ações que realizam efeito colateral não sejam realizadas dentro de transações, ou seja, somente operações que realizam alteração na memória serão executadas dentro de uma transação.

2.4 Green Computing:

Computação Verde (Green computing) refere-se a um conjunto de práticas adotadas pelo setor tecnológico para tornar o meio ambiente mais sustentável. A programação paralela tem certa influencia nesta área (VYKOUKAL. 2009). Segundo (Stéfano 2010), o objetivo da programação paralela para a economia de energia é obter uma melhor forma de aproveitamento dos recursos disponíveis, utilizando de técnicas de escalonamento de tarefas nos *cores* e do controle de granularidade. Em seu artigo sobre o papel da programação paralela no aproveitamento dos recursos energéticos, demostra que um algoritmo paralelo bem programado é capaz de utilizar os recursos à disposição de maneira mais eficiente, aumentando ao máximo o desempenho e diminuindo sua ociosidade.

2.5 Método:

As versões dos benchmarks usando *locks* foram obtidas através da substituição dos blocos atômicos por *locks* globais, tendo assim as aplicações do benchmark com os dois modelos de sincronismo.

Na realização dos testes, foram feitas 10 medidas de tempo de execução de cada programa nos dois modelos, bem como a mensuração do consumo de energia. Para se medir o consumo de energia foram utilizados multímetros, ligados a

entrada de força do computador de teste, onde através de medidas de tensão e corrente, foram feitas as relações de potência. Logo após, estes dados foram integrados em relação ao tempo de execução, onde se conseguiu extrair o consumo de energia de cada aplicação.

O computador onde foram realizados os testes foi um core2quad, 2Mb L2 com 4Gb de memoria RAM, rodando Linux Ubuntu 11.04. A versão do compilador Haskell utilizada foi o ghc 6.12.3 e a biblioteca com a implementação da STM foi a 2.1.1.2.

Também foi utilizado o programa *Threadscope*. (DONNIE 2011), no qual se pode verificar como estava sendo escalonadas as tarefas entre os *cores* existentes na maquina de teste.

3 RESULTADOS E DISCUSSÃO

Os resultados parciais dos testes realizados foram os seguintes conforme tabelas abaixo, onde o tempo está em segundos e a energia consumida em joules:

cores	BT				HT				LL			
	LOCK		STM		LOCK		STM		LOCK		STM	
	Tempo	Energia	Tempo	Energia	Tempo	Energia	Tempo	Energia	Tempo	Energia	Tempo	Energia
1	2,2671	162,35	11,4860	780,07	10,1746	785,67	11,1417	848,71	4,4554	318,35	13,6702	947,07
2	13,5590	828,75	7,0405	573,99	35,4861	2507,09	9,7026	797,31	35,8291	2414,97	10,1328	783,31
4	13,5517	917,22	4,0813	534,25	34,9160	2421,02	7,6604	670,22	35,4544	2453,85	7,0602	523,52

Tabela 1

cores	SI				SUD				TC			
	LOCK		STM		LOCK		STM		LOCK		STM	
	Tempo	Energia	Tempo	Energia	Tempo	Energia	Tempo	Energia	Tempo	Energia	Tempo	Energia
1	10,8038	889,32	13,0002	755,54	3,3517	257,05	10,0678	731,89	10,7602	781,11	12,7154	966,32
2	259,6410	3602,64	54,6676	20080,89	3,5389	271,50	6,0653	527,58	7,5306	522,61	7,0956	542,86
4	268,6886	7991,34	115,2498	21012,94	4,3206	322,47	4,2913	386,25	3,9964	276,80	5,0094	421,21

Tabela 2

Os resultados obtidos mostraram que, aumentando o número de *threads*, aumenta-se o desempenho usando STM, como em BT, HT, LL, SUD e TC (PERFUMO. 2007). Porém, em aplicações onde a grande contenção da memória, como o SI, o desempenho dos programas tende a ser ruim comparado a utilização de *locks.*

Verifica-se também que o consumo de energia não está, diretamente ligado ao desempenho computacional. Como exemplo, na aplicação SI, pode-se notar que o tempo da STM em relação ao modelo tradicional usando *locks*, foi menor, entretanto, o consumo de energia foi sete vezes maior. Para uma verificação mais profunda deste caso, foi utilizado o software *Threadscope*, que acabou revelando que, o baixo consumo presente na versão usando *locks*, na verdade era pelo motivo de que quase nunca a aplicação usava todos os cores a 100% (ocorrência de gargalo serial), e que o alto consumo por parte da STM, ocorreu devido a um grande número de abortos realizados nas transações devido a alta contenção ocorrida em específico nesta aplicação.

4 CONCLUSÃO

Através dos resultados parciais obtidos, pode-se concluir que apesar de em alguns casos as memórias transacionais apresentarem baixo desempenho, na maioria das aplicações apresentam melhor rendimento computacional e menor consumo de energia. Conclui-se então que, memórias transacionais são um modelo promissor para utilização como modelo de programação paralela e são uma importante área de pesquisa.

5 AGRADECIMENTOS

O primeiro autor gostaria de agradecer ao CNPq pela bolsa PIBIC, concedida ao projeto.

Este trabalho foi financiado pelos projetos FAPERGS/PRONEX/CNPq GREEN-GRID e FAPERGS/PESQUISADOR GAÙCHO CMTJava.

6 REFERÊNCIAS

TANENBAU, A. S. and Woodhull. **Operating Systems Desing and Implementation.** - local-:Prentice Hall, 2006.

HARRIS, T., Marlow, S., Jones, S. P., and Herlihy, M. **Composable memory transactions**. Commun, ACM 51:91-100, 2008

RAJWAR, R. and Goodman, J. **Transactional execution**: Toward reliable, high-performance multithreading. IEEE Micro, 23: 117-125, 2003

LEE, E. A. **The problem with threads**. IEEE Computer, 39(5): 33-42, 2006

JONES, S. P. Beautiful concurrency. **Beautiful Code:** Leading Programmers Explain How They Think, O´Reuilly Media. Inc, , 1: 1-24, 2007

RIGO S., C. P. and A., B. Memórias Transacionais, uma nova alternativa para programação concorrente, **WSCAD**, Anais Eletrônicos do WSCAD,2007.

DONNIE Jones, Simom Marlow, S. S. **Threadscope**, http://researche.microsoft.com/threadscope/. 2011

PERFUMO, C. and Sönmez, N. and Cristal, A. and Unsal, O. and Valero, M. and Harris, T. Dissecting transactional executions in Haskell, **TRANSACT**, Second ACM SIGPLAN Workshop on Transactional Computing, 2007

VYKOUKAL, J., Wolf, M., and Beck, R. (2009). Does green it matter? analysis of the relati-onship between green it and grid technology from a resource-based view perspective.In **PACIS**, Proceedings, page paper 51. 2009

STÉFANO D. K., M. A. Z. A. e J. V. L. Eficiência energética em computação de alto desempenho: Uma abordagem em arquitetura e programação para green computing. **SEMISH – XXXXVII,** Seminário Integrado de Software e Hardware. 2010