Universidade Federal de Pernambuco
Graduação em Ciência da Computação
Centro de Informática
2012.2

UM ESTUDO SOBRE A TRANSIÇÃO PARA ARQUITETURAS MULTICORE EM APLICAÇÕES DE CÓDIGO ABERTO

Proposta de Trabalho de Graduação

Aluno – Rafael Brandão Lôbo, rbl@cin.ufpe.br
Orientador – Fernando José Castor de Lima Filho, fjclf@cin.ufpe.br

Recife, 20 de janeiro de 2013

1. Contexto

Microprocessadores baseados em apenas uma unidade de processamento seguiram aumentos de performance e reduções de custo nas últimas décadas até alcançarem limites impostos pela tecnologia, como problemas de dissipação de calor e consumo de energia, o que provocou uma desaceleração no crescimento [7]. Como melhorar a velocidade de um único núcleo do processador tornou-se uma tarefa cada vez mais complicada, novos processadores passaram a vir com uma maior quantidade de núcleos que funcionam paralelamente, conhecidos como processadores *multicore*, e acredita-se que essa tendência seja irreversivel [1]. Apesar de sistemas operacionais modernos darem suporte a esses processadores ao permitir a execução de diversos processos em paralelo nestes núcleos adicionais [2], são poucas as aplicações que também fazem proveito dessa arquitetura, desperdiçando um grande potencial oferecido pela tecnologia. Estudiosos acreditam que o futuro bem sucedido dos processadores *multicore* depende da capacidade dos desenvolvedores de utilizarem ao máximo o seu potencial em suas aplicações [3].

No entanto, projetar uma aplicação que leve em consideração processadores *multicore* significa inserir tarefas paralelizáveis em sua arquitetura, o que historicamente não tem sido uma tarefa de fácil absorção entre engenheiros de software [8]. Problemas de concorrência entre threads, como *deadlocks*, *starvation*, e mal uso de memória compartilhada são apenas alguns dos vários problemas enfrentados no cotidiano ao lidar com esse tipo de arquitetura. Além disso, potenciais bugs tornam-se complexos de serem identificados ou resolvidos já que exigem um conhecimento amplo de como as threads e/ou processos envolvidos na aplicação se intercalam para gerar o resultado esperado.

Seguir várias linhas de execução paralelas é sem dúvidas uma tarefa complexa. Existem diversas formas de minimizar essa complexidade. Alguns optam por adotar linguagens que facilitem o desenvolvimento desse tipo de aplicação através de construções da própria linguagem, como Java, Erlang e Go, ou até mesmo criar linguagens especificas para o domínio que já tratem desses problemas eficientemente, como sugerido em [4]. Outra opção é ditar regras de como essa paralelização deve ser tratada minimizando as dificuldades: evitando-se construções da linguagem que dificultem a manutenção do código a longo prazo, proibindo uso de certas estruturas da linguagem, e assim por diante, usados por [5] e [6].

É notório que o uso de construções para programação concorrente e paralela é difícil; existem vários estudos sobre a ocorrência de bugs quando se usa construções para programação concorrente [11, 12, 13]; existem estudos que mostram que construções para programação concorrente são amplamente utilizados [9, 10]; entretanto, ainda não há nenhum estudo que avalia os efeitos de modificar sistemas para torná-los capazes de tirar proveito de arquiteturas *multicore*, em particular, se modificações com este fim podem levar a ocorrência de bugs ou não.

2. Objetivos

O principal objetivo deste estudo é entender o impacto causado pelas mudanças em cada projeto e identificar as principais motivações que levaram a tais mudanças. Em seguida, concluir se elas foram satisfatórias a ponto de cumprir com suas expectativas originais.

3. Metodologia

Serão escolhidos três grandes projetos de código aberto que tenham passado por modificações com o objetivo de tirar proveito do desempenho extra dos processadores *multicore*. Para cada projeto, será feito:

- Estudo introdutório sobre sua funcionalidade e características gerais;
- Análise dos impactos que as mudanças causaram no projeto como um todo, includindo identificação de áreas mais afetadas pelas mudanças e frequência de bugs relacionados.
- Avaliação do projeto em seu estado final para verificar se as expectativas que justificaram as modificações foram alcançadas.

E, depois, uma comparação será realizada entre os projetos, esperando encontrar pontos em comum e resultando na conclusão deste estudo.

4. Cronograma

O Quadro 1 exibe o cronograma listando os dias em que cada atividade será realizada durante o processo de desenvolvimento do trabalho de graduação. As atividades de documentação serão realizadas durante todo o período para evitar o acúmulo excessivo de documentação no final das análises dos projetos. O prazo reduzido deste trabalho é devido à necessidade de conclusão do curso antes do início da matrícula do mestrado que já se iniciará na primeira semana de março.

	Janeiro		Fevereiro							
Atividades / Dias	28-30	31-02	03-05	06-08	09-11	12-14	15-17	18-20	21-23	25
1. Revisão bibliográfica										
2. Análise do projeto 1										
3. Análise do projeto 2										
4. Análise do projeto 3										
5. Escrita da monografia										
6. Preparação da apresentação										
7. Apresentação do projeto										

Quadro 1 - Cronograma

5. Referências

[1] Technology Quarterly. “Parallel bars”. The Economist, June 2nd 2011. Disponível em: http://www.economist.com/node/18750706.

[2] Betti, E.; Bovet, D. P.; Cesati, M.; Gioiosa, R., “Operating System's Support for Multicore Systems: Current and Future Trends”, Technical Report SPRGTV-TR001 SPRG - TOR VERGATA, System Programming Research Group, Dept. of Computer Science, System and Industrial Engineering, University of Rome "Tor Vergata", April, 2007..

[3] Krste Asanovic et al. “A view of the parallel computing landscape”, Communications of the ACM, vol. 52, núm. 10, outubro de 2009. Páginas 56-67.

[4] Catanzaro, B.; Fox, A.; Keutzer, K.; Patterson, D.; Bor-Yiing Su; Snir, M.; Olukotun, K.; Hanrahan, P.; Chafi, H., "Ubiquitous Parallel Computing from Berkeley, Illinois, and Stanford”, *Micro, IEEE*, vol. 30, núm. 2, March-April 2010. Páginas 41-55.

[5] The Chromium Project Team, “Chromium’s Design Documents: Threading”. Disponível em: http://dev.chromium.org/developers/design-documents/threading

[6] “WebKit2 - High Level Document”. Disponível em: http://trac.webkit.org/wiki/WebKit2

[7] Diaz, J.; Munoz-Caro, C.; Nino, A. "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era”, IEEE Transactions on Parallel and Distributed Systems, vol. 23, núm. 8, Agosto 2012. Páginas 1369-1386.

[8] Paul McKenney, “Is Parallel Programming Hard, And, If So, What Can You Do About It?”. Disponível em: http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html

[9] Weslley Torres, Gustavo Pinto, Benito Fernandes, João Paulo Oliveira, Filipe Alencar Ximenes, “Fernando Castor: Are Java programmers transitioning to multicore?: a large scale study of java FLOSS”. SPLASH Workshops 2011: 123-128

[10] Semih Okur, Danny Dig, “How do developers use parallel libraries?” SIGSOFT FSE 2012: 54

[11] Lu S, Park S, Seo E, Zhou Y. “Learning from mistakes: a comprehensive study on real world concurrency bug characteristics”. SIGOPS Oper. Syst. Rev. March 2008; 42(2):329–339.

[12] Sadowski C, Yi J, Kin S. “The evolution of data races. Proceedings of the The 9th Working Conference on Mining Software Repositories”, ICSE ’12, IEEE: Zurich, Switzerland, 2012.

[13] Pedro Fonseca, Cheng Li, Vishal Singhal, Rodrigo Rodrigues: “A study of the internal and external effects of concurrency bugs”. DSN 2010: 221-230

6. Assinaturas

Abaixo estão as assinaturas dos responsáveis que se comprometeram com o desenvolvimento deste trabalho.

_____

Rafael Brandão Lôbo
Aluno

_____

Fernando José Castor de Lima Filho
Orientador