Contribuições ao teste de cobertura de aplicações CUDA

Helder J. F. Luz[1], **Simone R. S. Souza**[1] **and Paulo S. L. Souza**[1]

[1]Instituto de Ciências Matemática e de Computação (ICMC)
Universidade de São Paulo (USP)
Avenida Trabalhador São-carlense, 400 – Centro
CEP: 13566-590 – São Carlos - SP - Brasil

helderfl@icmc.usp.br, srocio@icmc.usp.br, pssouza@icmc.usp.br

Abstract. *CUDA is a programming model for the implementation of GPU general-purpose applications. Although it provides the development of GPU executable codes, programming in this model is not trivial. In addition, developers generally have little experience in developing parallel applications, resulting in different types of errors. Despite the need to mitigate errors in CUDA applications, current testing tools do not provide the necessary support. Seeking to improve the quality of CUDA programs, this work proposes the development of a model and structural test criteria for concurrent programs based on this technology, with the purpose of revealing defects.*

Resumo. *CUDA é um modelo de programação voltado à implementação de aplicações de propósito geral em GPU. Apesar de propiciar a criação de códigos executáveis em GPU, a programação nesse modelo não é trivial. Além disso, os desenvolvedores geralmente possuem pouca experiência na desenvolvimento de aplicações paralelas, ocasionando diversos tipos de defeitos. Apesar da necessidade de mitigar defeitos em aplicações CUDA, as ferramentas de teste atuais não oferecem o suporte necessário. Buscando melhorar a qualidade de programas CUDA, este trabalho propõe o desenvolvimento de um modelo e de critérios de teste estrutural para programas concorrentes baseados nessa tecnologia, com o propósito de revelar defeitos.*

1. Introdução

CUDA (*Compute Unified Device Architecture*), desenvolvido pela Nvidia, é um modelo de programação e plataforma de computação paralela para propósitos gerais voltado para GPUs (*Graphics Processing Unit*) da Nvidia. Ele facilita a implementação e solução de problemas computacionais complexos em GPU [Cheng et al. 2014], possuindo bibliotecas e primitivas para facilitar o uso eficiente da arquitetura.

Os dois principais apelos para o uso de CUDA são: (i) a possibilidade de se obter ganhos significativos de desempenho; (ii) a possibilidade de desenvolvimento de código concorrente muito próximo ao sequencial, abstraindo-se vários dos detalhes da geração de processos/*threads* e da IPC (*Interprocess Communication*). Os ganhos expressivos de desempenho são obtidos com o uso de GPUs, as quais são compostas de várias unidades de processamento replicadas. Estas são alimentadas com instruções e dados vindos de memórias existentes na GPU. A replicação das unidades de processamento permite a execução simultânea de várias *threads*; a organização de memória da GPU fornece uma

vazão de instruções e dados adequada, pois é baseada fortemente nos princípios de localidade espacial e temporal de *caches* [Patterson and Hennessy 2013]; evitando assim concorrer com o barramento e bancos de memória no computador hospedeiro da GPU.

Entretanto, o desenvolvimento de programas concorrentes CUDA envolve a implementação de código fonte para ser executado na CPU (*Central Processing Unit*) e outra parte na GPU. A GPU funciona como um co-processador e depende da CPU para fazer a execução de sua parcela de código, uma vez que essa última informa os dados e as instruções a serem executadas. Além disso, a arquitetura SIMD (*Single Instruction; Multiple Data*) e os diversos tipos de memória implicam em variados desafios na implementação para alcançar um bom desempenho em uma aplicação semanticamente correta [Cheng et al. 2014].

Dado esse cenário, desenvolver aplicações com CUDA não é, de fato, uma atividade trivial. Isso ocorre principalmente porque os desenvolvedores estão acostumados a implementar programas sequenciais, onde suas execuções podem ser entendidas ao observar o fluxo sequencial do código fonte [Cheng et al. 2014]. Códigos em CUDA necessitam que o desenvolvedor tenha uma abstração maior e consiga visualizar os fluxos paralelos de execução, somados às comunicações e sincronizações.

Atrelado a isso, os programadores possuem pouca experiência na implementação de aplicações paralelas [Tanenbaum and Bos 2014]. Frequentemente profissionais que utilizam esses modelos de programação concorrente não são da área da computação, tendo pouca experiência com o desenvolvimento correto de soluções computacionais. Isso dificulta o desenvolvimento de aplicações de qualidade que apresentem ganhos de desempenho expressivos e, ao mesmo tempo, tenham poucos defeitos. No caso do modelo de programação CUDA, isso pode acarretar em defeitos de natureza variada. Segundo [Cook 2013], alguns defeitos comuns são:

- Acesso inválido de um vetor/matriz: acontece quando não se limita o acesso das *threads* aos limites das variáveis.
- Relacionados à sincronia das *threads*, sendo este um defeito não determinístico difícil de ser revelado e reproduzido [de Souza et al. 2007];
- Erros que afetam qualquer modelo ou linguagem de programação (sequencial ou concorrente).

Revelar defeitos causados pela inexperiência dos desenvolvedores e dificuldades relacionadas à arquitetura e modelo de programação podem se apresentar como uma atividade custosa. Em programas mais complexos, o espaço de busca pode tornar inviável a procura por defeitos pelo método da força bruta [Myers et al. 2011]. Para viabilizar e auxiliar essa busca, existem as técnicas da atividade de teste de *software*.

2. Teste de software

A atividade de Teste tem por objetivo revelar defeitos no programa ou modelo por meio da sua execução com base em técnicas de teste; se a aplicação não funcionar dentro de suas especificações, diz-se que foi revelado um defeito na mesma [Myers et al. 2011].

Dentre as técnicas de teste de software, há o teste estrutural, que utiliza a implementação do programa para realizar a atividade de teste, com base em um modelo

que exprime o comportamento do modelo de programação CUDA e critérios que definem os elementos requeridos, responsáveis por guiar a atividade de teste.

Há diversos trabalhos que utilizam teste estrutural, mas em geral os modelos de teste já desenvolvidos não se aplicam ao CUDA devido ao modelo de programação diferente. Eles são voltados para a arquitetura MIMD (*Multiple Instructions;Multiple Data*), encontrada nas CPUs, enquanto a GPU utiliza arquitetura conceitualmente similar a SIMD. Os poucos trabalhos voltados ao teste de aplicações focam em problemas específicos, como *deadlock*, condições de disputa e defeitos de desempenho [Li et al. 2014].

Neste cenário, o teste de aplicações paralelas de alto desempenho, incluindo aquelas desenvolvidas sob o modelo de programação CUDA, é um desafio [Myers et al. 2011]. O teste de tais aplicações paralelas ainda é pouco desenvolvido e utilizado. Há a falta de modelos de teste específicos para extrair informações relevantes para testar tais aplicações, principalmente aquelas relacionadas à comunicação, sincronização e ao não determinismo existentes, usualmente em função do comportamento dinâmico das mesmas. Atrelados aos modelos, são necessários novos critérios e ferramentas de teste que deem o suporte adequado à atividade, de maneira a viabilizá-la na prática.

3. Objetivos

Com base nos desafios apresentados, o principal objetivo deste trabalho é investigar como melhorar a qualidade de programas concorrentes CUDA, por meio da definição de critérios estruturais de teste que auxiliem na revelação de defeitos.

Os objetivos específicos são: definir classes de defeitos para programas concorrentes CUDA; desenvolver um modelo de teste estrutural que considere elementos que necessitam ser testados; desenvolver critérios de teste estrutural; desenvolver um protótipo de ferramenta de teste que implemente o modelo e os critérios propostos; disponibilizar os programas concorrentes CUDA usados como *benchmarks* neste projeto.

4. Metodologia

Inicialmente está sendo feita a revisão sistemática sobre o estado da arte em teste de *software* e programação concorrente voltados ao modelo de programação CUDA.

Posteriormente programas concorrentes CUDA desenvolvidos por terceiros serão analisados buscando identificar padrões de defeitos como: aplicações de código aberto disponibilizadas na *web*, códigos fornecidos por empresas, algoritmos utilizados para a análise de eficiência de ferramentas correlatas e exemplos de implementações fornecidos pela Nvidia.

Com base na análise feita, será desenvolvida e especificada uma taxonomia de defeitos, buscando organizar e classificar os defeitos identificados. Por meio da taxonomia, um conjunto de *benchmarks*, que apresentem ou não os defeitos especificados, será desenvolvido, buscando exercitar diversos casos diferentes que possibilitem o teste e a comparação do protótipo com ferramentas correlatas

Utilizando como referencial modelos de teste existentes aplicados a diversas linguagens e por meio do estudo do modelo de programação CUDA e das arquiteturas de GPUs Nvidia, será desenvolvido e especificado um modelo de teste que exprima o comportamento de aplicações CUDA. De posse da taxonomia de defeitos e do modelo, serão

especificados os critérios de teste que buscam revelar defeitos em programas que utilizem esse modelo de programação.

Por fim será implementado um protótipo para dar suporte ao modelo e aos critérios propostos. Por meio do protótipo e *benchmarks* desenvolvidos, a eficiência e eficácia do modelo e critérios serão avaliados e comparados com ferramentas correlatas.

5. Resultados esperados

Ao final deste trabalho, espera-se fazer algumas contribuições na área de teste de *software* e desenvolvimento de aplicações CUDA com qualidade. As contribuições esperadas são:

- Taxonomia de defeitos: disponibilização da taxonomia dos defeitos observados em programas desenvolvidos com CUDA.
- *Benchmarks*: os algoritmos que implementam a taxonomia serão disponibilizados, permitindo que futuros trabalhos utilizem para comparação e avaliação.
- Critérios de teste para programas concorrentes CUDA: permitindo que eles sejam aplicados na melhora da qualidade de programas desenvolvidos e possibilitando novas pesquisas com base neles.
- Ferramenta de teste: o protótipo a ser desenvolvido será disponibilizado para permitir comparações e o desenvolvimento de novas ferramentas utilizando-o integralmente ou parcialmente.
- Aplicações no ensino de programação concorrente CUDA: durante o processo de desenvolvimento deste projeto, serão ministrados cursos de programação, sendo que parte do material desenvolvido poderá ser utilizado para o ensino de aplicações CUDA de qualidade.
- Aplicação no ensino de Teste de *Software*: os artefatos gerados poderão ser usados no ensino de teste de programas concorrentes CUDA.
- Aplicação na melhoria de qualidade das aplicações: o modelo, critérios e ferramenta permitirão a melhoria da qualidade de programas concorrentes que utilizam esse modelo de programação.

Referências

Cheng, J., Grossman, M., and McKercher, T. (2014). *Professional CUDA C Programming*. Wrox : Programmer to Programmer. Wiley.

Cook, S. (2013). *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., 1st edition.

de Souza, S. R. S., Vergilio, R. S., and de Souza, P. S. L. (2007). Teste de programas concorrentes. In Delamaro, M. E., Maldonado, J. C., and Jino, M., editors, *Introdução ao Teste de Software*, chapter 9. Elsevier, 1st edition.

Li, P., Li, G., and Gopalakrishnan, G. (2014). Practical symbolic race checking of gpu programs. In *High Performance Computing, Networking, Storage and Analysis*.

Myers, G. J., Sandler, C., and Badgett, T. (2011). *The Art of Software Testing*. Wiley Publishing, 3rd edition.

Patterson, D. and Hennessy, J. (2013). *Computer Organization and Design: The Hardware/Software Interface*. Elsevier Science.

Tanenbaum, A. S. and Bos, H. (2014). *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition.