Padrões e *Frameworks* de Programação Paralela em Ambientes *Multi-Core*

Dalvan Griebler, Mateus Raeder,
Luiz Gustavo Fernandes

Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - Prédio 32 - PPGCC
{dalvan.griebler, mateus.raeder}@acad.pucrs.br, luiz.fernandes@pucrs.br

Introdução

Nos últimos anos, o mercado recente de estações de trabalho e servidores vem aumentando gradativamente a quantidade de núcleos e processadores, inserindo na sua programação o paralelismo, o que resultou no aumento da complexidade em lidar com este tipo de *hardware*. Neste cenário, é necessário a disponibilidade de mecanismos que possam fornecer escalabilidade e permitir a exploração de paralelismo nestas arquiteturas, conhecidas como *multi-core*. Não basta que tais arquiteturas multiprocessadas estejam disponíveis, se estas não são exploradas devidamente.

Depuração, condições de corrida, sincronização de *threads* ou processos e controle de acesso aos dados são exemplos de fatores críticos para a programação destes ambientes paralelos. Novas maneiras de abstrair a complexidade em lidar com estes sistemas estão sendo estudadas, a fim de que a programação paralela seja algo menos complexo para os desenvolvedores de *software*. Padrões paralelos vêm sendo o alvo de constantes estudos com intuito de padronizar a programação [1].

Padrões e *Frameworks* de Programação Paralela

Uma implementação eficiente com padrões paralelos nem sempre é possível. Depende do programador se a modelagem e os cuidados com relação ao acesso de dados foram corretamente tratados. Padrões paralelos determinísticos tornaram-se um atual objeto de pesquisa no cenário de programação paralela (principalmente para arquiteturas *multi-core*). O propósito deles é possibilitar o desenvolvimento de programas eficientes, mantendo uma única ordem na execução, similar ao que existe em algoritmos sequenciais. Além disso, fornecem determinismo na execução dos programas, eliminando a necessidade de condições de corrida [2]. Padrões de programação paralela melhoram a produtividade, pois são otimizados, específicos e podem ser combinados. Além disso, também podem direcionar os desenvolvedores inexperientes e profissionais a desenvolverem *softwares* com maior eficiência e escalabilidade. Três exemplos destes padrões são elencados a seguir:

- *Reduction*: utiliza um operador de associatividade para todos os elementos de uma coleção em um único elemento;
- *Pipeline*: é um conjunto de tarefas executando simultaneamente (estágios) fazendo um relacionamento produtor-consumidor;

- *Map*: utilizado no paralelismo de iterações de laço, onde a computação é realizada independentemente para um único índice.

Estes são apenas alguns exemplos de padrões estruturados determinísticos de programação paralela, que auxiliam o programador na análise de requisitos e nos cuidados em relação à programabilidade dos algoritmos paralelos. Os padrões de programação paralela podem ser divididos em dois grupos: padrões de projeto e padrões de algoritmos paralelos. Padrões de projeto direcionam o desenvolvimento de programas para uma das melhores soluções paralelas, baseados em uma metodologia. Padrões de algoritmos, por sua vez, especificam e definem a estrutura algorítmica, tratando as questões de acesso aos dados e os detalhes do ambiente arquitetural a ser paralelizado, provendo maior eficiência na utilização dos recursos disponíveis. Algumas interfaces de programação implementam alguns padrões de algoritmos, outros padrões são usados para fazer a modelagem do programa, como por exemplo, Mestre-Escravo e Divisão e Conquista [1].

Uma solução para ajudar os programadores na exploração de paralelismo e na facilidade de programação é utilizar *frameworks*[3]. O **Intel Parallel Studio** [2] é um dos principais *frameworks* que possui a combinação de interface de fácil operação para depuração e padrões paralelos. Outros *frameworks* como **FastFlow** [4], **TBB** (*Threading Bulding Blocks*) [5] e **SWARM** (*SoftWare and Algorithms for Running on Multi-core*) [6], também incorporam padrões paralelos, sendo que alguns deles são determinísticos, como por exemplo, *Pipeline*, *Reduce*, *Partition* e *Scan*. Entretanto, mecanismos de depuração não são implementados, os quais tem por finalidade auxiliar no controle de *threads* ou processos para garantir o acesso seguro aos dados.

Proposta de Trabalho

A partir deste estudo, propõe-se a construção de um *framework* para facilitar o desenvolvimento de aplicações em ambientes *multi-core* através da utilização de padrões paralelos. Padrões de programação paralela permitem otimizar a exploração do paralelismo, abstraindo detalhes de programação concorrente para melhor utilizar os recursos de *hardware* e proporcionar maior desempenho.

Referências

[1] MATTSON G. T., SANDERS A. B., and MASSINGILL L. B. *Patterns for Parallel Programming*. Addison-Wesley, Boston, MA, 2005.

[2] INTEL and MCCOOL D. M. Structured Parallel Programming with Deterministic Patterns. In *HotPar-2nd USENIX Workshop on Hot Topics in Parallelism*, pages 1–6, Berkeley, CA, June 2010.

[3] CATANZARO R. and KEUTZER K. Parallel Computing with Patterns and Frameworks. *XRDS: Crossroads, The ACM Magazine for Students*, 17(1):22–27, 2010.

[4] ALDINUCCI M., RUGGIERI S., and TORQUATI M. Porting Decision Tree Algorithms to Multicore Using FastFlow. In *Proc. of European Conference in Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pages 7–23, Barcelona, Spain, September 2010.

[5] INTEL. Intel® Threading Building Blocks (Home Page), Extracted from http://www.threadingbuildingblocks.org. Access in November, 2010.

[6] BADER D. A., KANADE V., and MADDURI K. SWARM: A Parallel Programming Framework for Multicore Processors. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, Long Beach, California USA, March 2007.