DSL-POPP: Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões

Dalvan Griebler, Luiz Gustavo Fernandes

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - Prédio 32 - PPGCC

dalvan.griebler@acad.pucrs.br, luiz.fernandes@pucrs.br

Resumo. *A proposta deste trabalho é induzir o programador a desenvolver programas orientados a padrões paralelos, que implementados na interface de uma linguagem específica de domínio ajudam a reduzir o esforço de programação sem comprometer o desempenho de uma aplicação. Resultados experimentais com o padrão mestre/escravo mostraram um bom desempenho nos algoritmos paralelizados.*

Introdução

Este trabalho apresenta uma Linguagem Específica de Domínio (DSL) para Programação Paralela Orientada a Padrões (POPP) [1], denominada de DSL-POPP [2]. A principal motivação é diminuir o esforço de programação necessário para exploração do paralelismo em programas de arquiteturas Multi-core. Pretende-se fazer isto através do uso de padrões de programação, os quais são geralmente mais fáceis de entender para programadores inexperientes [3]. O desafio é implementar esta abordagem sem comprometer o desempenho que pode ser obtido na paralelização de aplicações. Com o intuito de averiguar este hipótese, realizamos um estudo experimental com o padrão mestre/escravo via DSL-POPP.

DSL-POPP

A estrutura do modelo POPP é genérica o suficiente para suportar diferentes padrões paralelos [2]. É importante realçar que a DSL-POPP pode ser estendida para prover outros padrões paralelos (para cada padrão novo, um novo conjunto de primitivas deve ser definido). Para usar a DSL, desenvolvedores deverão incluir a biblioteca da linguagem (`poppLinux.h`) no código fonte. Esta biblioteca inclui todas as definições de núcleos, blocos de código e primitivas.

O processo de compilação é demonstrado na Figura 1. A transformação do código do modelo POPP é automaticamente realizada pelo pré-compilador que é também responsável por analisar erros de sintaxe e semântica. Assim, o pré-compilador gera código paralelo em C baseando-se no padrão utilizado por meio da biblioteca Pthreads. Então, finalmente, o compilador GCC gera o código binário.

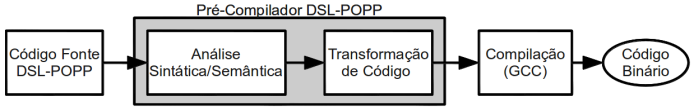

Figura 1. Visão global do processo de compilação.

A especificação da interface do padrão mestre/escravo é resumida na Figura 2. Baseando-se no modelo POPP [2], os núcleos começam com "$", os blocos de código com "@" e as primitivas com "_&". Os blocos e as primitivas pertencentes ao núcleo principal contém "_" ao final de seus nomes.

Na versão para o padrão mestre/escravo da DSL-POPP as funções são compostas por blocos mestre e escravo. Os desenvolvedores também podem especificar quantos escravos serão criados a partir de um bloco escravo (`$mainMasterSlavePat-`

tern(int n)). A primitiva _&SynchronizeSlaves_ é responsável por criar os escravos a partir dos blocos escravos e esperar até que eles terminem seu trabalho. Além disso, para garantir o acesso seguro aos dados compartilhados é provida a primitiva _&ProtectData(MUTEX, var){} que implementa o mecanismo *mutex*.

Uma simples multiplicação de matrizes utilizando a interface é demonstrada na Figura 3. Cada escravo executa o mesmo código especificado dentro do bloco escravo. Entretanto, construções de laços for são automaticamente paralelizadas pelo bloco escravo. No código exemplo, DSL-POPP divide de forma transparente as interações do laço mais externo for(i=0; i<SIZE; i++) entre os escravos. E o bloco mestre apenas está inicializando os blocos escravos.

Núcleos: $

Interface	*Descrição*
$mainMasterSlavePattern(n){}	Núcleo principal do padrão mestre/escravo
$MasterSlavePattern(n){}	Sub-núcleo do padrão mestre/escravo

Blocos de código: @

Interface	*Descrição*
@Master_{}	Bloco Mestre do núcleo principal
@Slave_{}	Bloco Escravo do núcleo principal
@Master{}	Bloco Mestre do sub-núcleo
@Slave{}	Bloco Escravo do sub-núcleo

Primitives: _&

Interface	*Descrição*
&SynchronizeSlaves	Sincroniza todos os blocos Slave_{}
_&SynchronizeSlaves	Sincroniza todos os blocos Slave{}
_&ProtectData(type, var){}	Protege dados compartilhados

Figura 2. Interface do padrão mestre/escravo.

```
1 #include<poppLinux.h>
2 #define SIZE 2000
3 long int m1[SIZE][SIZE], m2[SIZE][SIZE],
4          mult[SIZE][SIZE];
5 $mainMasterSlavePattern(16){ //$Nucleo
6   @Master_{ //@Bloco
7     _&SynchronizeSlaves_; //_&Primitiva
8   }
9   @Slave_{ //@Bloco
10    long int i, j, k;
11    for(i=0; i<SIZE; i++)
12      for(j=0; j<SIZE; j++)
13        for(k=0; k<SIZE; k++)
14          mult[i][j] += (m1[i][k] * m2[k][j])
15  }
16 }
```

Figura 3. Multiplicaçao de Matrizes com DSL-POPP (16 escravos).

Resultados e Conclusões

Os experimentos para medição do desempenho foram realizados em uma máquina composta por dois processadores Intel Xeon Quad-Core E5520 de 2.27GHz e 16GB de memória. Para cada situação experimentada foram calculados os *speed-ups* a partir da média de 40 execuções. As aplicações paralelizadas foram as seguintes:

- MD: efetua uma simulação de dinâmica molecular com *4000* partículas em *4* etapas.
- EI: estima uma integral sobre uma área retangular 2D de *91768* por *91768*.
- MM: resolve uma multiplicação de matrizes densas de ordem *1800*.

Tabela 1. *Speed-ups das aplicações com DSL-POPP.*

Alg.	**Threads**							
	2	4	6	8	10	12	14	16
MD	2.00	3.98	5.90	7.72	8.08	8.96	9.94	10.93
EI	1.94	3.62	5.30	7.28	7.78	8.97	10.15	11.19
MM	1.91	3.50	5.11	6.75	7.07	8.08	8.81	8.78

A avaliação do desempenho das aplicações está demonstrada na Tabela 1. Todas as aplicações apresentaram um bom desempenho, mas uma certa perda ocorreu quando executadas a partir de 10 *threads*. Isso foi devido ao uso da tecnologia *hyper-threading*, pois quando usado nestas configurações haverá mais *threads* do que *cores* físicos. Diante dos resultados, pretende-se no futuro disponibilizar outros padrões através da DSL-POPP e fornecer suporte para explorar o paralelismo de arquiteturas paralelas híbridas compostas de CPUs e GPUs.

Referências

[1] Raeder M., Griebler D., Baldo L., and Fernandes L. G. Performance Prediction of Parallel Applications with Parallel Patterns Using Stochastic Methods. In *Sistemas Computacionais (WSCAD-SSC), XII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 1–13, Vitória, Espírito Santo, Brasil, October 2011.

[2] Griebler D. J. Proposta de uma Linguagem Específica de Domínio de Programação Paralela Orientada a Padrões Paralelos: um Estudo de Caso Baseado no Padrão Mestre/Escravo para Arquiteturas Multi-Core. Master's thesis, PUCRS, 2012.

[3] Mattson G. T., Sanders A. B., and Massingill L. B. *Patterns for Parallel Programming*. Addison-Wesley, Boston, USA, 2005.