Proposta de Grau de Paralelismo Autoadaptativo com MPI-2 para a DSL SPar

Cassiano Rista[1], **Dalvan Griebler**[1,2], **Luiz Gustavo L. Fernandes**[1]

[1] Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre – RS – Brasil

[2]Laboratório de Pesquisas Avançadas para Computação em Nuvem (LARCC) Faculdade Três de Maio (SETREM), Três de Maio – RS – Brasil

luis.rista@acad.pucrs.br

Resumo. *Este artigo apresenta o projeto de um módulo autoadaptativo para controle do grau de paralelismo à ser integrado a DSL SPar. O módulo para aplicações paralelas distribuídas de stream permite a criação de processos em tempo de execução, seleção da política de escalonamento, balanceamento de carga, ordenamento e serialização, adaptando o grau de paralelismo de forma autônoma sem a necessidade de definição de thresholds por parte do usuário.*

1. Introdução

Os recentes avanços nas tecnologias de microprocessadores e redes locais de computadores de alto desempenho permitiram que os ambientes de *cluster* de computadores se tornassem um recurso computacional usual atualmente, seja através de pesquisas na industria ou na academia [Dantas and Rista 2005]. Através da exploração do paralelismo desses ambientes torna-se possível atingir requisitos de computação de alto desempenho para a execução de diferentes aplicações paralelas distribuídas.

No entanto, a utilidade dos ambientes de *cluster* depende do desenvolvimento de aplicações que executem eficientemente em tal recurso computacional. Tem-se verificado atualmente que a utilização desses ambientes para a execução de aplicações de *stream* têm sido evitada devido a dificuldade de adequar o grau de paralelismo da aplicação em tempo de execução, visto que esses ambientes são usualmente projetados para um número fixo de processos.

Nesse sentido, os desafios impostos para que as linguagens possam expressar as aplicações paralelas distribuídas de *stream* incluem desde a necessidade de modelagem do código e implementação da comunicação de modo explícito, balanceamento de carga, sincronização de processos e serialização de dados. É desejável também aos programadores o conhecimento necessário para entender sobre a arquitetura subjacente de modo a explorar eficientemente o paralelismo disponível.

O desenvolvimento de linguagens nas quais aplicações paralelas distribuídas possam ser expressas se torna um desafio e motivam diferentes projetos de pesquisa, tais como: Charm++ [Charm++ 2019], Chapel [Chapel 2019] e X10 [X10 2019]. Outros exemplos incluem a DSL SPar para ambiente *multicore* [Griebler et al. 2017], ambiente de *cluster* [Griebler and Fernandes 2017] e a variante do ambiente *muticore* com grau adaptativo de paralelismo [Vogel et al. 2018]. Cabe destacar, que os projetos de pesquisa envolvendo a DSL SPar foram desenvolvidos pelo Grupo de Modelagem de Aplicações

Paralelas (GMAP) e compartilham do mesmo objetivo que é fornecer abstrações de alto nível em C++ para aplicações de *stream* para diferentes ambientes computacionais.

Diante desse cenário, aplicações que permitem o processamento de fluxos contínuos de dados, como aplicações de *stream*, vem tornando-se cada vez mais relevantes. A capacidade de adequar o grau de paralelismo da aplicação em tempo de execução possibilita ainda a aquisição de recursos de forma dinâmica com base na variação da carga de trabalho do sistema. Assim, este artigo propõe habilitar o grau de paralelismo de modo autoadaptativo para aplicações paralelas distribuídas de *stream* em ambientes de *cluster* a partir da implementação de um módulo para a DSL SPar [Griebler and Fernandes 2017]. Cabe destacar que o projeto encontra-se em fase final de codificação, paralelamente a montagem e preparação do ambiente para a execução dos experimentos.

2. Trabalhos Relacionados

Charm++ [Charm++ 2019] , Chapel [Chapel 2019] e X10 [X10 2019] são exemplos de *frameworks* encontrados na literatura para aumentar a produtividade em ambientes de computação paralela distribuída. Charm++ [Charm++ 2019] por exemplo, pode ser descrito como uma extensão da linguagem C++ que fornece mecanismos e estratégias de alto nível para o desenvolvimento de aplicações paralelas. A linguagem é baseada na migração de objetos e os programadores interagem através de invocações de objetos assíncronos. Chapel [Chapel 2019] por sua vez é uma linguagem de programação que contém aspectos de orientação a objetos, programação imperativa, além de mecanismos que visam facilitar o desenvolvimento de aplicações paralelas distribuídas para sistemas de larga escala. Por fim, a linguagem X10 [X10 2019] é baseada em Java a partir de um modelo de programação APGAS (Espaço de Endereçamento Global Particionado Assíncrono), que oferece um conjunto de instruções para lidar com um único espaço de memória.

Em contraste aos projetos relacionados, optou-se pelo MPI-2 [Gropp et al. 2014] para habilitar o grau de paralelismo autoadaptativo para a DSL SPar em ambientes de *cluster* [Griebler and Fernandes 2017]. É importante ressaltar que a DSL SPar gera transformações resultantes *source-to-source* em FastFlow [FastFlow 2019] para ambientes *multicore* e MPI padrão (estático) para *cluster*. O MPI (*Message Passing Interface*) por sua vez pode ser visto atualmente como o principal padrão para a transmissão de mensagens em ambientes de *cluster*. Sendo que a partir da versão denominada usualmente de MPI-2 tornou-se possível a criação de processos em tempo de execução. Além desse aspecto, levou-se em consideração também o fato da linguagem FastFlow (quando associada a versão distribuída) não possuir nenhum mecanismo de ativação de processos, o que implica ao programador ter que iniciar manualmente o processo em cada nó do *cluster*.

3. Fase Atual de Desenvolvimento do Projeto

O desenvolvimento do projeto visa a implementação de um módulo capaz de habilitar o grau de paralelismo autoadaptativo para aplicações paralelas distribuídas de *stream* para a DSL SPar [Griebler and Fernandes 2017], sem a necessidade de definição de *thresholds* ou intervenção por parte do usuário, adaptando o grau de paralelismo da aplicação no ambiente de *cluster* de forma autônoma.

A DSL SPar foi estendida baseando-se no conceito de computação autônoma a partir do modelo de controle de referencia MAPE-K [Lalanda et al. 2013], demonstrado

na Figura 1, tendo como objetivo principal o gerenciamento do recurso computacional de modo automático e dinâmico visando minimizar dessa forma a necessidade de intervenção humana. O modelo utiliza parâmetros de desempenho como *Throughput* e Latência obtidos através de sensores de *software* que permitem a DSL o suporte de decisão necessário para adaptar o grau de paralelismo da aplicação. Para isso, utiliza-se a função *Spawn* do MPI-2 que permite a criação dinâmica de processos em tempo de execução.

```
1  [[spar::ToStream]] while(1){
2    frame f = read_frame();
3    if(f.empty()) break;
4    [[spar::Stage,spar::Input(f),spar::Output(f),
       spar::Replicate(n)]]
5    for (int i=0; i<f.length(); i++) {
6      f[i] = convert(f[i]);
7    }
8    [[spar::Stage,spar::Input(f)]]{
9      write_frame(f);
10   }
11 }
```

Listagem 1. Exemplo de código SPar.

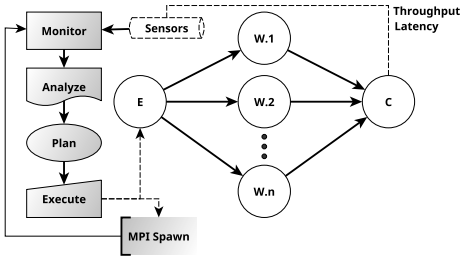

Figura 1. Proposta MAPE-K na SPar.

O desenvolvimento do projeto visa simplificar a geração de código paralelo para o ambiente de *cluster*, suportando modelos como *Farm* e *Pileline* [McCool et al. 2012] apresentados na Figura 2, garantindo que os recursos do ambiente de computação paralela distribuída sejam capazes de manter a execução da aplicação de *stream* dentro de um determinado objetivo de nível de serviço (SLO). Cabe destacar, que esse nível é obtido em tempo de execução através da estratégia de adaptação autônoma que atua no grau de paralelismo da aplicação através do uso de *thresholds* dinâmicos atualizados em intervalos de 1 segundo, sem a necessidade de intervenção humana.

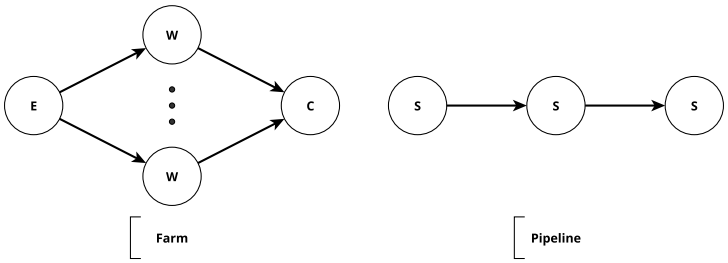

Figura 2. Visão conceitual de modelos tradicionais de *Farm* e *Pipeline*.

Para atingir os níveis de SLO através do grau de paralelismo autoadaptativo, foram utilizadas técnicas de programação para a criação e/ou remoção de processos em tempo de execução, serialização de dados, ordenamento, escalonamento e balanceamento de carga, preservando a semântica original conforme apresentado na Listagem 1 e gerando transformações de código *source-to-source* resultantes em MPI-2.

Em relação ao escalonamento de processos, foram implementadas duas politicas usualmente utilizadas. A primeira politica de escalonamento se baseou no tradicional algoritmo *Round Robin*, atribuindo recursos aos processos a partir de uma fila circular simples. A segunda politica de escalonamento implementada fez uso do algoritmo *On-Demand*, onde os processos realizam novas requisições a medida que finalizam suas ta-

refas. Outra característica desse algoritmo é o balanceamento de carga intrinsecamente inerente a politica de escalonamento.

Por fim, a técnica de ordenação foi implementada com base em uma árvore binaria do tipo *Red-Black* que apresenta uma complexidade de tempo para pesquisa, inserção e remoção de ordem O (log n) [Cormen et al. 2001]. Associado a técnica de serialização de dados, torna-se possível reduzir a sobrecarga do ordenamento e comunicação respectivamente. Como resultado, pretende-se analisar o impacto que o suporte dinâmico em tempo de execução pode trazer sobre métricas como desempenho, portabilidade e eficiência, quando do processamento paralelo distribuído de aplicações de *stream* na DSL SPar. É importante ressaltar que o projeto encontra-se em fase final de codificação, paralelo a configuração do ambiente para a execução dos experimentos preliminares: conjunto de *Mandelbrot* e bzip2.

Referências

[Chapel 2019] Chapel (2019). The Chapel Parallel Programming Language.

[Charm++ 2019] Charm++ (2019). Parallel Programming with Migratable Objects.

[Cormen et al. 2001] Cormen, T., Cormen, T., Leiserson, C., Rivest, R., of Technology, M. I., Stein, C., Books24x7, I., Press, M., and Company, M.-H. P. (2001). *Introduction To Algorithms*. Introduction to Algorithms. MIT Press.

[Dantas and Rista 2005] Dantas, M. A. R. and Rista, C. (2005). A wireless monitoring approach for a ha-oscar cluster environment. In *19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, pages 302–306.

[FastFlow 2019] FastFlow (2019). Parallel Programming Framework.

[Griebler et al. 2017] Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2017). SPar: A DSL for High-Level and Productive Stream Parallelism. *Parallel Processing Letters*, 27(01):1740005.

[Griebler and Fernandes 2017] Griebler, D. and Fernandes, L. G. (2017). Towards Distributed Parallel Programming Support for the SPar DSL. In *Parallel Computing is Everywhere, Proceedings of the International Conference on Parallel Computing*, ParCo'17, pages 563–572, Bologna, Italy. IOS Press.

[Gropp et al. 2014] Gropp, W., Hoefler, T., Thakur, R., and Lusk, E. (2014). *Using Advanced MPI: Modern Features of the Message-Passing Interface*. Computer science & intelligent systems. MIT Press.

[Lalanda et al. 2013] Lalanda, P., McCann, J. A., and Diaconescu, A. (2013). *Autonomic Computing - Principles, Design and Implementation.* Undergraduate Topics in Computer Science. Springer.

[McCool et al. 2012] McCool, M., Robison, A. D., and Reinders, J. (2012). *Structured Parallel Programming: Patterns for Efficient Computation*. Morgan Kaufmann.

[Vogel et al. 2018] Vogel, A., Griebler, D., Sensi, D. D., Danelutto, M., and Fernandes, L. G. (2018). Autonomic and Latency-Aware Degree of Parallelism Management in SPar. In *Euro-Par 2018: Parallel Processing Workshops*, pages 28–39, Turin, Italy. Springer.

[X10 2019] X10 (2019). X10: Performance and Productivity at Scale.