Adaptando o Paralelismo em Aplicações de *Stream* Conforme Objetivos de *Throughput*

Adriano Vogel[1], **Dalvan Griebler**[1], **Luiz G. Fernandes**[1]

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

adriano.vogel@edu.pucrs.br

Abstract. *As aplicações de processamento de streams possuem características de execuções dinâmicas com variações na carga e na demanda por recursos. Adaptar o grau de paralelismo é uma alternativa para responder a variação durante a execução. Nesse trabalho é apresentada uma abstração de paralelismo para a DSL SPar através de uma estratégia que autonomicamente adapta o grau de paralelismo de acordo com objetivos de desempenho.*

1. Introdução

O recente aumento no número de dispositivos (ex: sensores, câmeras, radares) conectados e produzindo dados demanda técnicas inovadoras de processamento. O paradigma de *stream processing* [Andrade et al. 2014] representa uma classe de aplicações voltadas para processar e filtrar em tempo real um fluxo contínuo de dados. Porém, processar em tempo real sob variações no fluxo, velocidade e volume dos dados se tornou um desafio para os sistemas de processamento de *stream*. Uma abordagem para tentar acelerar as execuções foi introduzir o paralelismo, realizando simultaneamente diferentes operações sob dados/tarefas.

Um fator desafiador é que introduzir paralelismo não é uma tarefa trivial para a maioria dos programadores de aplicações de *stream*, pois demanda um conhecimento aprofundado de arquitetura de computadores e otimizações de baixo nível, como *cache*, acesso à memória, particionamento de dados, condições de corrida, etc. No cenário de processamento paralelo de *stream* existe a SPar [Griebler et al. 2017, Griebler et al. 2018], que busca facilitar a introdução de paralelismo em aplicações de *stream* em C++ usando atributos. Tais atributos são anotados em trechos paralelizáveis do código sequencial. Conforme as anotações no código, o compilador da SPar gera código paralelo.

O problema tratado nesse trabalho está relacionado com a definição e adaptabilidade do grau de paralelismo, que por padrão na SPar é definido manualmente pelo programador, sendo fixo durante toda a execução e não respondendo a flutuações que ocorram durante a execução. No estudo anterior [Vogel and Fernandes 2018] foi demonstrado como gerenciar autonomicamente o grau de paralelismo baseado no monitoramento das filas da *runtime*, enquanto que em [Vogel et al. 2018] foi evidenciado o impacto do grau de paralelismo na latência dos itens de *stream*, e como a latência pode ser gerenciada através de otimizações no grau de paralelismo.

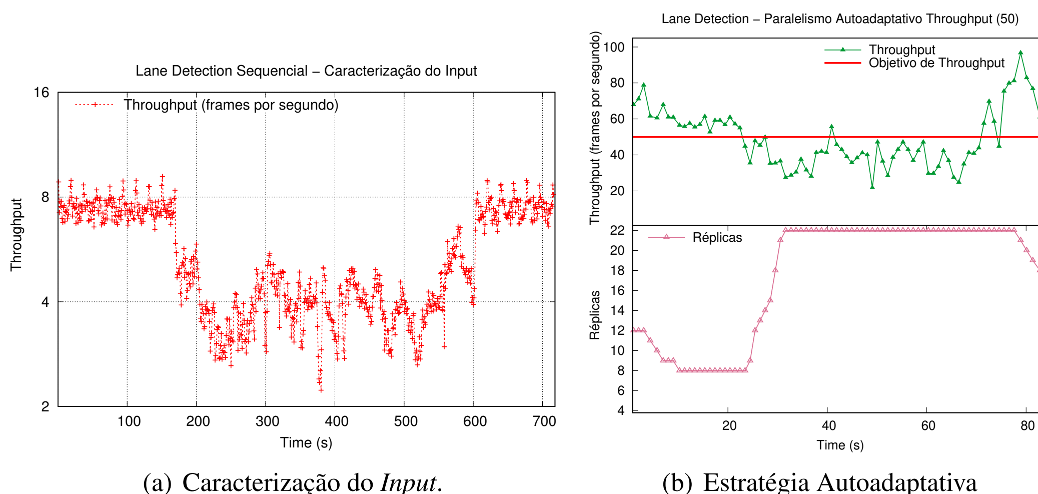
Existem trabalhos relacionados que implementam algoritmos para execuções adaptativas, como em [Gedik et al. 2014] e [Sensi et al. 2016]. Por outro lado, o objetivo desse trabalho é oferecer abstrações de paralelismo de alto nível e reduzir complexidades para

programadores de aplicação. Para reduzir complexidades, conceitos de computação autônoma [Hellerstein et al. 2004] e *self-adaptive systems* [Macías-Escrivá et al. 2013] podem ser usados para gerenciar as execuções sem interferência manual do programador. A estrutura desse artigo é a seguinte: a Seção 2 apresenta a implementação e caracterização da estratégia, na Seção 3 são apresentados os resultados dos experimentos realizados, e a conclusão é evidenciada na Seção 4.

2. Implementação do Grau de Paralelismo Autoadaptativo

O grau de paralelismo na DSL SPar corresponde ao número de réplicas em um determinado estágio paralelo. Considerando a demanda por abstrair a definição do número de réplicas e possibilitar que tal valor possa ser ajustado durante a execução, foi implementado uma estratégia que monitora a *runtime* e usa um regulador de paralelismo para decidir se é necessário adaptar a execução, aumentando ou diminuindo o número de réplicas.

O regulador de paralelismo demanda que um objetivo de desempenho seja definido pelo programador, para assim decidir se é necessário adaptar o número de réplicas. No contexto de processamento *stream*, um exemplo de objetivo pode ser o *throughput* (tarefas por segundo). Usando esse objetivo de desempenho, o regulador monitora periodicamente o *throughput* atual da aplicação e o compara com o objetivo de desempenho. Se o *throughput* for inferior, réplicas são adicionadas na tentativa de melhorar o desempenho. O regulador decide quantas réplicas devem ser adicionadas calculando o percentual da diferença entre o desempenho atual e o objetivo, comparando tal percentual com a capacidade de processamento disponível no ambiente de execução. O número de réplicas é diminuído pelo regulador se o *throughput* for significativamente superior ao objetivo.



(a) Caracterização do *Input*. (b) Estratégia Autoadaptativa

Figura 1. Execução Sequencial do Input (a) e paralela (b).

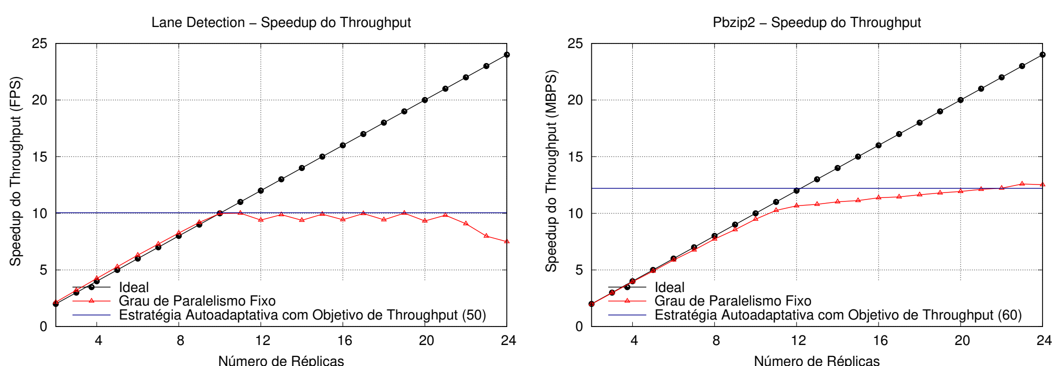
O funcionamento da estratégia autoadaptativa é demonstrado em uma implementação na aplicação Lane Detection, que é uma aplicação de vídeo usada por carros autônomos para a detecção de pistas em rodovias. A Figura 1(a) mostra a caracterização da carga de trabalho em uma execução sequencial do vídeo utilizado como *input*. Na Figura 1(b) é mostrada a execução paralela autoadaptativa da aplicação com um objetivo de desempenho de 50 (*frames*) por segundo, nessa aplicação cada *frame* equivale a uma tarefa. O número de réplicas foi estabilizado em 8 réplicas no início da execução, após o vigésimo

segundo da execução o *throughput* diminuiu devido a variação do *input*, evidenciado na Figura 1(a). Consequentemente, a estratégia adicionou novas réplicas buscando aumentar o *throughput* para alcançar o objetivo, assim atingindo a utilização máxima dos recursos na máquina utilizada. Quando o *throughput* aumentou em uma nova tendência de desempenho da carga de trabalho no *input* utilizado, o grau de paralelismo foi diminuído pela estratégia para utilizar os recursos de forma compatível com o objetivo de desempenho.

3. Avaliação de Desempenho

A estratégia autoadaptativa impacta na execução de aplicações, pois usa um número variante de réplicas de acordo com as decisões tomadas pelo regulador de paralelismo. O impacto foi avaliado comparando o desempenho de execuções autoadaptativas com versões paralelas usando um grau de paralelismo fixo (o mesmo número de réplicas durante a execução). Os testes foram executados em uma máquina multi-core equipada com 2 Intel(R) Xeon(R) CPU 2.40 GHz (12 cores- 24 *Hyperthreads*), 32 GB - 2133 MHz de memória usando o sistema operacional Ubuntu Server.

A Figura 2(a) apresenta os resultados da comparação de desempenho na aplicação Lane Detection, usando como *input* o vídeo caracterizado na Figura 1(a). Nesse experimento a estratégia autoadaptativa atingiu um desempenho similar as melhores execuções com paralelismo fixo, evidenciando que o modo autoadaptativo não prejudica o desempenho da aplicação. O desempenho competitivo da versão autoadaptativa é justificado pela estratégia responder rapidamente a flutuações, adicionando em poucos segundos várias novas réplicas, como mostrado na Figura 1(b). Nas execuções com grau de paralelismo fixo fica evidente que o uso de *Hyperthreads* não trouxe ganho de desempenho na máquina testada, pois o limite de escalabilidade da aplicação foi atingido com 10 réplicas. Nota-se também que ocorreram oscilações no desempenho com mais de 12 réplicas, o que é resultado da combinação do uso de *Hyperthreads*, desbalanceamento de carga e ordenamento dos itens de *stream*.



(a) Aplicação Lane Detection. (b) Aplicação Pbzip2.

Figura 2. *Speedup* do *Throughput*.

Na aplicação Pbzip2, que é uma aplicação do domínio de compressão de dados, foi utilizado um *input* de 6.3 GB. Nesse experimento da Figura 2(b) é notável uma diferente tendência de desempenho. Nas execuções com paralelismo fixo, o desempenho melhora com até 23 réplicas. A execução autoadaptativa atingiu um desempenho proporcional a 22 réplicas do paralelismo fixo (próximo ao desempenho máximo).

4. Conclusão

Nesse estudo foi apresentada e validada uma nova abstração de paralelismo para a DSL SPar. A estratégia autoadaptativa monitora e otimiza a aplicação durante a execução. Comparando os resultados nas duas aplicações (com diferentes comportamentos) fica evidente que o desempenho atingido com a estratégia autoadaptativa, apesar de ter uma execução mais elaborada, foi próximo aos melhores casos do paralelismo fixo.

No futuro se pretende implementar o paralelismo autoadaptativo em ambientes de memória distribuída. Ainda, as estratégias autoadaptativas poderiam ser portadas para outras classes de aplicações com padrões/composições mais complexas.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal Nível Superior – Brasil (CAPES) – Código de Financiamento 001 e FAPERGS.

Referências

[Andrade et al. 2014] Andrade, H., Gedik, B., and Turaga, D. (2014). *Fundamentals of Stream Processing: Application Design, Systems, and Analytics*. Cambridge University Press.

[Gedik et al. 2014] Gedik, B., Schneider, S., Hirzel, M., and Wu, K.-L. (2014). Elastic Scaling for Data Stream Processing. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1447–1463.

[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 et al. 2018] Griebler, D., Hoffmann, R. B., Danelutto, M., and Fernandes, L. G. (2018). High-Level and Productive Stream Parallelism for Dedup, Ferret, and Bzip2. *International Journal of Parallel Programming*, pages 1–19.

[Hellerstein et al. 2004] Hellerstein, J. L., Diao, Y., Parekh, S., and Tilbury, D. M. (2004). *Feedback Control of Computing Systems*. John Wiley & Sons.

[Macías-Escrivá et al. 2013] Macías-Escrivá, F. D., Haber, R., Del Toro, R., and Hernandez, V. (2013). Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267–7279.

[Sensi et al. 2016] Sensi, D. D., Torquati, M., and Danelutto, M. (2016). A Reconfiguration Algorithm for Power-Aware Parallel Applications. *ACM Transactions on Architecture and Code Optimization*, 13(4):43:1–43:25.

[Vogel and Fernandes 2018] Vogel, A. and Fernandes, L. G. (2018). Grau de Paralelismo Adaptativo na DSL SPar. In *Escola Regional de Alto Desempenho (ERAD)*, page 2, Porto Alegre, BR. Sociedade Brasileira de Computação (SBC).

[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*, page 12, Turin, Italy. Springer.