Aprimorando a Análise de Desempenho de Aplicações Baseadas em Tarefas Irregulares e Árvores de Eliminação

Marcelo Cogo Miletto[1]*, **Claudio Schepke**[2], **Lucas Mello Schnorr**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
[2]Universidade Federal do Pampa (UNIPAMPA)

{marcelo.miletto, schnorr}@inf.ufrgs.br, claudioschepke@unipampa.edu.br

Resumo. *Este trabalho apresenta os resultados de estratégias de aprimoramento da análise de desempenho de aplicações baseadas em tarefas com carga de trabalho irregular, através da automação de detecção de tarefas anômalas usando modelos de regressão. Também apresentamos técnicas de visualização de desempenho de aplicações baseadas na estrutura da árvore de eliminação, utilizada na paralelização de algoritmos a de fatoração de matrizes esparsas.*

Contextualização e Motivação: A análise de desempenho de aplicações trata-se de um passo fundamental para a otimização de aplicações paralelas e distribuídas. Somada com as atuais arquiteturas heterogêneas, a complexidade de desenvolvimento de aplicações paralelas eficientes também aumenta. Alguns paradigmas de programação como a programação baseada em tarefas simplificam a codificação de tais aplicações através de camadas de abstração, como a representação da aplicação como um grafo acíclico dirigido e do sistema de *runtime*, responsável pelo gerenciamento de dados e escalonamento das tarefas, provendo desempenho e portabilidade. Tais abstrações, comportamentos específico de tarefas, e outras estruturas utilizadas para a representação de um algoritmo, podem complicar a análise de desempenho. Sendo assim, devemos considerar tais particularidades para o estudo do desempenho de uma aplicação. Como estudo de caso, usamos o *solver* esparso baseado em tarefas `qr_mumps` [Agullo et al. 2016], construído usando a biblioteca StarPU, e o método *multifrontal*, baseado na estrutura da árvore de eliminação.

Metodologia e Resultados: As estratégias propostas foram desenvolvidas no contexto da ferramenta StarVZ [Schnorr et al. 2020]. Para a detecção automática de tarefas anômalas, consideramos a adoção de modelos de regressão usados para prever a duração esperada de uma determinada tarefa de acordo com o seu custo computacional em operações de ponto flutuante, seu tipo, e o tipo de recurso em que foi executada (ex: CPU ou GPU). A classificação de uma anomalia depende se a observação de uma tarefa encontra-se acima do valor previsto de acordo com o seu custo computacional, ilustrado pela Figura 1 (A) onde a linha vermelha representa o limite superior de predição considerando um intervalo de confiança de 95%. Também consideramos o uso da técnica de mistura finita de modelos de regressão em casos onde apenas um modelo era insuficiente para descrever o comportamento das tarefas, agrupando as tarefas de acordo com a sua probabilidade de fazer parte de um determinado modelo, como apresentado pela Figura 1 (B), onde claramente temos dois comportamentos entre o mesmo tipo de tarefa, sendo as tarefas do *cluster* vermelho mais lentas. Os painéis de visualização voltados para a aplicação também foram adicionados na ferramenta, representando o desempenho da aplicação relacionado com características específicas de suas tarefas, estrutura de dados, e a formulação do método

*Bolsa do Conselho Nacional de Desenvolvimento Científico (CNPq) processo número 131347/2019-5.

multifrontal. A Figura 1 (C) representa a estrutura da árvore de eliminação ao longo do tempo e onde ocorreram as computações de acordo com o gradiente de cores que representa a taxa de utilização de recursos em um dado momento e local da árvore. Através da técnica de detecção de anomalias foram detectados cinco diferentes fontes de anomalias dentre as tarefas: (1) pico de submissões de tarefas, (2) sobrecarga do sistema de rastreamento, (3) conteúdo numérico dos blocos de dados de uma tarefa, (4) compartilhamento de *streamming multiprocessors* de uma GPU entre mais de um trabalhador StarPU, e (5), o aumento do número de cache *misses* no nível L3 devido à interferência de tarefas concorrentes, representado pela Figura 1 (B). Neste último caso, exploramos uma abordagem para reduzir a interferência entre tarefas restrigindo a execução de tarefas sensíveis à estes *misses* na L3 a um número menor de CPUs. Com isto, foi possível alcançar reduções de até 24% no tempo total de execução para casos onde havia interferência significativa. Com a análise relacionada a estrutura da árvore, foi possível observar como os diversos parâmetros tanto a nível de aplicação quanto do sistema de *runtime* afetam o comportamento da aplicação. Exploramos aspectos como prioridades de tarefas, restrição no uso de memória e escalonadores, alinhando as abstrações envolvidas no método *multifrontal* ao desempenho da aplicação, aproximando a análise do ponto de vista de um desenvolvedor.

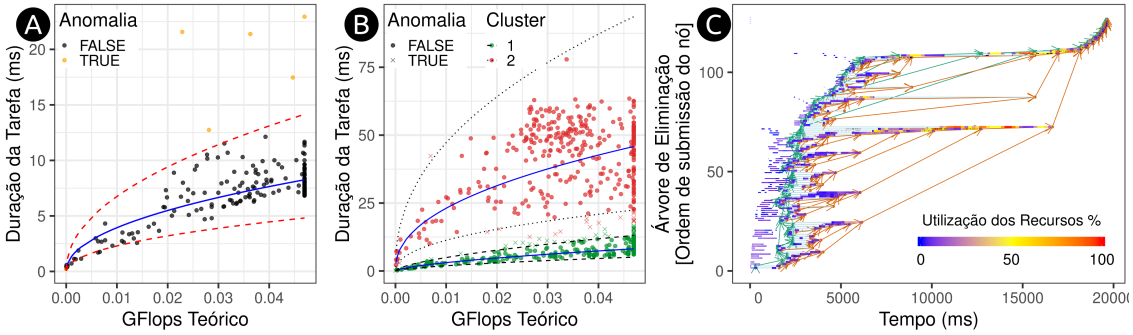

Figura 1. Modelos de regressão e visualização da árvore de eliminação.

Conclusão e Trabalhos Futuros: A série de experimentos realizados durante este trabalho permitiu investigar e identificar configurações da aplicação e *runtime* em diferentes plataformas, encontrando cenários onde o desempenho é prejudicado, guiando a análise a casos onde podemos melhorar o desempenho da aplicação como no cenário (5). Direções futuras envolvem soluções mais sofisticadas para lidar com a interferência entre tarefas, como a detecção *online* de cenários de interferência, ou o particionamento e escalonamento de tarefas considerando a interferência na memória cache [Guo et al. 2020].

Referências

Agullo, E., Buttari, A., Guermouche, A., and Lopez, F. (2016). Implementing multifrontal sparse solvers for multicore architectures with sequential task flow runtime systems. *Acm transactions on mathematical software (toms)*, 43(2):1–22.

Guo, Z., Yang, K., Yao, F., and Awad, A. (2020). Inter-task cache interference aware partitioned real-time scheduling. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 218–226.

Schnorr, L. M., Pinto, V. G., Nesi, L. L., and Miletto, M. C. (2020). *starvz: R-Based Visualization Techniques for Task-Based Applications*. R package version 0.4.1.