Encontrando a Configuração de Threads por Bloco para os Kernels NPB-CUDA com Q-Learning

Claudio Scheer[1], **Gabriell Araujo**[1], **Dalvan Griebler**[1], **Felipe Meneguzzi**[1], **Luiz G. Fernandes**[1]

[1] Escola Politécnica, Grupo de Modelagem de Aplicações Paralelas (GMAP), Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, Brasil

{claudio.scheer,gabriell.araujo}@edu.pucrs.br
{dalvan.griebler,felipe.meneguzzi,luiz.fernandes}@pucrs.br

Resumo. *Este trabalho apresenta um novo método que utiliza aprendizado de máquina para prever a melhor configuração de threads por bloco para aplicações de GPUs. Os resultados foram similares a estratégias manuais.*

1. Contexto

Unidades de Processamento Gráfico (GPUs) são aceleradores que oferecem grande poder computacional através de seu paralelismo massivo. No entanto, a programação de GPUs é um desafio para os programadores, pois requer o uso de programação paralela, e conhecimento da arquitetura. Outro desafio das GPUs é a portabilidade das aplicações através da escolha de parâmetros adequados de acordo com o *hardware*. Um exemplo de parâmetro que pode afetar significativamente o desempenho da GPU é o número de *threads* por bloco em um *kernel* CUDA. Dado este contexto, este trabalho apresenta (1) um novo método para prever a melhor configuração de *threads* por bloco para uma aplicação de GPU e (2) um novo *dataset*, construído usando o NPB paralelizado com CUDA [Araujo et al. 2021], com atributos correlatos à performance das aplicações na GPU.

2. Implementação e desempenho do método proposto

Este trabalho propõem um método utilizando redes neurais e Q-Learning para encontrar a melhor configuração de *threads* por bloco de acordo com a GPU e aplicação. O modelo de rede neural treinado funciona como um simulador de todas as combinações possíveis de *threads* por bloco para os *kernels* CUDA de uma aplicação. A rede neural fornece uma função que prevê o tempo de execução de uma configuração específica, o que permite explorar o espaço de configurações de *threads* por bloco possíveis usando o Q-Learning. O modelo de regressão foi treinado a partir de um *dataset* composto por atributos da aplicação e arquitetura da GPU, com base no NPB paralelizado com CUDA. O Q-Learning é um algoritmo de aprendizado por reforço que aprende uma política ótima para determinado ambiente simplesmente interagindo com ele enquanto usa uma política gulosa para atualizar os retornos esperados de cada nova política explorada [Sutton and Barto 2018]. A exploração do espaço de busca se dá através de escolhas aleatórias ou da combinação das melhores configurações para cada *kernel* CUDA da aplicação. No método proposto, a probabilidade de explorar novas configurações de *threads* por bloco para determinado *kernel* CUDA é proporcional ao impacto que este *kernel* tem na aplicação.

As configurações de *threads* por bloco geradas com o método proposto foram comparadas com outras quatro estratégias manuais que requerem uma grande quantidade de experimentos e análise do programador, as quais são definidas como: 1) `Max`, cada *kernel* de GPU recebe o número máximo de *threads* por bloco suportado pela GPU. 2) `Warp`, cada *kernel* de GPU recebe o tamanho do *warp* da GPU como número de *threads* por bloco. 3) `Profiling`, são executados experimentos onde varia-se o número de *threads* por bloco de um *kernel* de GPU, e é escolhido o número de *threads* que apresentou o menor tempo de execução. 4) `Exhausting`, são executados experimentos que testam todas as possibilidades possíveis de *threads* por bloco na aplicação. Como esse é um problema exponencial, foram escolhidos apenas os *kernels* que consomem pelo menos 10% do tempo de execução.

Tabela 1. Resultados (segundos)

Benchmark/ Modelos	Classe B				Classe C			
	CG	FT	MG	EP	CG	FT	MG	EP
Max	11.35	2.52	0.20	0.83	22.69	10.93	1.63	2.47
Warp	1.49	2.36	0.21	0.59	3.65	10.16	1.85	2.18
Exhausting	1.40	2.35	0.19	0.52	3.52	10.15	1.78	**2.11**
Profiling	**1.39**	**2.34**	**0.19**	**0.52**	**3.51**	**10.15**	**1.60**	2.12
Modelo 1	1.61	2.40	**0.19**	**0.53**	3.39	10.16	**1.59**	**2.36**
Modelo 2	**1.32**	2.40	0.19	0.53	3.33	10.17	1.91	2.36
Modelo 3	1.33	2.40	0.19	0.53	3.33	**10.15**	1.73	2.36
Modelo 4	1.36	2.40	0.19	0.53	3.33	10.20	1.60	2.36
Modelo 5	1.33	**2.35**	0.19	0.53	**3.32**	10.15	1.59	2.36

Como a exploração do espaço de configurações possíveis de *threads* por bloco não é determinístico, foram gerados cinco configurações para cada aplicação testada, como apresentado na Tabela 1. Cada modelo leva aproximadamente 2 minutos para ser gerado sequencialmente numa CPU AMD Ryzen 9 5900X. Os resultados mais expressivos foram nas aplicações CG (classe C) e EP (classe C). No CG, considerando as cinco configurações de *threads* por bloco previstas pelo Q-Learning, todos os modelos tiveram resultados melhores. Uma das razões para este desempenho é o *kernel* CUDA que consome a maior parte do tempo de execução, onde o método proposto previu 32 *threads* por bloco, enquanto o `Profiling` usou 64 *threads* por bloco. Para o modelo `Exhausting` não foi possível encontrar uma configuração ideal porque limitamos o espaço de configurações testadas. No benchmark EP, o método proposto previu configurações com uma notável degradação de desempenho. A configuração ideal para o caso EP é de 32 *threads* por bloco, enquanto o método proposto previu 64 *threads* por bloco. Isso mostra que os modelos de aprendizado de máquina devem ser aprimorados na análise do consumo de registradores de cada *thread* para fornecer uma configuração ideal.

3. Conclusões

Os resultados parciais deste trabalho demonstraram que o método apresentado é capaz de prever configurações similares e até melhores que estratégias manuais que exigem grande esforço e análise por parte do programador.

Referências

Araujo, G., Griebler, D., Rockenbach, D. A., Danelutto, M., and Fernandes, L. G. (2021). NAS Parallel Benchmarks with CUDA and beyond. *Software: Practice and Experience*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.