Paralelização do Algoritmo Evolução Diferencial em GPU com Uso de Gerador Cíclico de Índices

Mateus Boiani[1], **Rafael Stubs Parpinelli**[1]

[1] Programa de Pós-Graduação em Computação Aplicada – PPGCA
Universidade do Estado de Santa Catarina (UDESC) – Joinville – SC – Brasil

mateus.boiani@edu.udesc.br, rafael.parpinelli@udesc.br

Resumo. *A Evolução Diferencial é um algoritmo de otimização para problemas complexos de domínio contínuo. Apresenta-se aqui uma versão deste algoritmo utilizando a plataforma de computação paralela NVIDIA CUDA, chamado gDE. O gDE utiliza um gerador cíclico de índices que o diferencia das abordagens tradicionais e, quando comparado com sua versão em CPU (cDE), um speedup médio de 5× foi obtido.*

1. Introdução

A Evolução Diferencial (*Differential Evolution* - DE) foi proposta por Storn e Price [Das and Suganthan 2011] e é um dos algoritmos populacionais mais utilizados em problemas complexos de domínio contínuo. Sendo um algoritmo evolutivo, este possui como característica dois níveis de paralelismo, implícito e explícito. O primeiro é resultado da utilização de múltiplas soluções durante o processo de otimização e o segundo permitindo que a execução do algoritmo seja modelada em arquiteturas paralelas. Outros trabalhos já exploraram o uso da DE em GPU (*Graphics Processing Unit*) [Wong et al. 2015]. Um dos desafios ao executar este algoritmo em GPU é a gerar os índices aleatórios necessários nas etapas de *crossover* e mutação. Alguns autores geram todos os índices em CPU e os copiam para GPU utilizando técnicas de tentativa-e-erro para garantir que sejam diferentes entre si, e outros executam um *kernel* de tentativa-e-erro em GPU. Porém, é sabido que geração de números aleatórios em GPU possuem limitações quanto a quantidade e qualidade de amostragem. Desta forma, este trabalho apresenta uma versão do DE em GPU que utiliza uma rotina otimizada para geração de índices aleatórios, chamada de gDE. Esta rotina consiste de um vetor circular de índices, onde são sorteados K deslocamentos, o primeiro é um valor entre 0 e o tamanho da população e para os demais um valor entre 1 e o tamanho da população divido por K. Para determinar os índices é feita a soma parcial dos deslocamentos (ex: para $K = 3$, o segundo índice é obtido através da soma do 1º deslocamento com o 2º e assim sucessivamente). O algoritmo foi construído utilizando a plataforma de computação paralela NVIDIA CUDA e a versão implementada é a DE/rand/1/bin.

A primeira etapa da execução efetua a alocação da memória, inicializa os vetores de solução e os parâmetros de controle. Posteriormente, as informações são copiadas para memória global da GPU e é dado início ao processo de otimização. O gDE divide-se em 3 *kernels*, o primeiro é o gerador de índices aleatórios, que deve garantir índices diferentes entre si e diferentes do indivíduos que os usará. O segundo *kernel* do algoritmo DE efetua a avaliação dos vetores *trial* gerados. Por último, o *kernel* de substituição das melhores proles, responsável por comparar e substituir indivíduos *trial* melhor adaptados.

A Equação 1 descreve a função *Shifted Griewank* que foi escolhida para realizar a otimização. Os vetores iniciais de solução possuem tamanho D de acordo com a quantidade de dimensões a serem otimizadas. Cada valor do vetor está distribuído no intervalo de $[-600.0, +600.0]$. Ao calcular o valor da otimização é necessário aplicar o deslocamento, de modo a obter $z = x - o$, onde o é o vetor de deslocamento. $f\_bias = -180.0$ é o ponto ótimo deslocado. Ambas abordagens (cDE e gDE) utilizam taxa de 30.0% para o *crossover*, fator de mutação 0,5 e 5000 gerações. Ambas utilizam o gerador de índices proposto neste trabalho, porém, o algoritmo em DE em CPU (cDE) não é paralelizado.

$$F_1(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} cos(\frac{z_i}{\sqrt{i}}) + 1 + f\_bias \quad (1)$$

A Tabela 1 apresenta os resultados obtidos no processo de otimização para o tempo de processamento em segundos. Os testes foram executados em uma máquina com processador Intel i7-4790K, GeForce GTX 970 utilizando Linux Ubuntu 16.04. O tamanho da população utilizado é descrito por η, a quantidade de dimensões está representada por D, $\bar{x}$ é a média entre 10 execuções e σ é o desvio-padrão. Para $\eta = 1000$ o algoritmo gDE utiliza 25 blocos com 25 *threads* cada. O teste de Wilcoxon foi utilizado para analisar se existe diferença estatística entre os tempos de processamento considerando 5% de significância (*p-value* indicado). Ambas abordagens atingiram o valor ótimo da função em todos os casos mostrando a eficiência do algoritmo DE em domínios contínuos.

		cDE		**gDE**		
η	D	Tempo (s)		Tempo (s)		*p-value*
		$\bar{x}$	σ	$\bar{x}$	σ	
1000	10	35,864	0,632	7,234	0,016	0,001
	50	43,591	0,368	8,409	0,058	0,001
	100	54,240	0,617	9,962	0,023	0,001

Tabela 1. Comparação dos resultados obtidos pelo DE em CPU e em GPU.

2. Conclusão

Dos resultados obtidos é possível observar que o gDE obteve tempos computacionais inferiores quando comparado com a abordagem em CPU, mantendo a qualidade da otimização (*speedup* médio de 5×). Um dos grandes diferenciais do gDE está na forma como o gerador de índices aleatórios funciona, não existindo a necessidade de utilizar métodos de tentativa-e-erro, como é feito tradicionalmente. Em trabalhos futuros, pretende-se explorar outras funções *benchmark* com altas dimensionalidades, incluir rotinas de auto-ajuste de parâmetros e implementar o gDE em um modelo co-evolutivo.

Referências

Das, S. and Suganthan, P. N. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31.

Wong, T. H., Qin, A. K., Wang, S., and Shi, Y. (2015). *cuSaDE: A CUDA-Based Parallel Self-adaptive Differential Evolution Algorithm*, pages 375–388. Springer International Publishing, Cham.