Estimando Densidades via Kernels: Uma Abordagem Paralela

Renan Procópio Duarte [1] & Carlos Eduardo Ribeiro de Mello [2]

1. Bolsista CNPq/PIBIC, Discente do Curso de Ciência da Computação, DCC/IM/UFRRJ; 2. Professor do DCC/IM/UFRRJ.

Palavras-chave: KDE; paralelo; CUDA, GPU.

Introdução

O projeto *"Análise de acesso a recursos compartilhados em ambientes multicore com GPUs"* trata o problema de Estimativa de Densidade via Kernels (*Kernel Density Estimation* - KDE). Esta técnica consiste em estimador probabilístico não-paramétrico, *i.e.,* não assume distribuições paramétricas, para estimar a densidade de observações a partir de uma amostra conhecida. O KDE se baseia basicamente na suavização da curva da função de densidade de probabilidade (pdf), onde a partir de uma amostra finita pode-se fazer diversas inferências sobre a população. A implementação do algoritmo do KDE pode ser separada em duas vertentes: Unidimensional e Multidimensional.

Há uma diversidade de aplicações do KDE, que vão desde econometria computacional e análise de mercado, até compactação de imagens, extração de bordas de imagens, linguística computacional e aprendizado de máquina. Todavia, o custo computacional ao executar o KDE, na notação *big O*, é da ordem $O(n^2k)$, onde *n* é o tamanho da amostra e *k,* o número de dimensões do problema em abordado, o que o torna extremamente custoso quando o tamanho da amostra é demasiadamente grande.

Metodologia

A metodologia de pesquisa no projeto tem como objetivo utilizar os paradigmas de programação paralela e distribuída para aumentar o desempenho computacional. A ideia é dividir o problema em pequenos problemas que sejam capazes de executar em unidades de processamento independentes. As GPUs (*Graphics Processing Unit – Placas Gráficas*) tem sido uma alternativa interessante para melhorar o desempenho de aplicações, principalmente daquelas que possuem operações de cálculo como da álgebra linear, pelo fato que no interior de cada placa pode ter mais de 300 núcleos (denominados *CUDA™ Cores*) atuando em paralelo. O objetivo, portanto, é utilizar a GPU para obter ganho no desempenho da aplicação, na qual o KDE será empregado.

Para mapear problemas de grande escala em uma arquitetura paralela utilizando o Modelo de programação CUDA™ (*Compute Unified Device Architecture*), da *NVIDIA©*, tem de seguir os dois passos do esquema de decomposição dos dados: O problema é particionado em blocos que podem ser computados independentemente em paralelo e; os blocos são particionados em sub-tarefas que podem ser computadas cooperativamente em paralelo.

Em CUDA™, a função a ser executada na GPU recebe o nome de *kernel*. Essa função é responsável por acessar o hardware onde ela é executada *N* vezes em paralelo em *N* diferentes CUDA™ *threads*. Essa função pode receber argumentos como valores ou ponteiros para memórias globais, locais e também possui uma série de constantes definidas que permite um *thread* identificar qual elemento deve ser processado por ela. Uma função *kernel* é definida utilizando a declaração __global__ e o número de *threads* que serão executadas.

Uma vez que o pré-processamento e o pós-processamento dos dados encontram-se em uma

ferramenta (*MatLab©*) teria de haver a migração da implementação para a mesma ferramenta a fim de evitar um *overhead* na troca de dados. A estratégia abordada foi utilizar o arquivo pré-compilado .ptx, o qual é referenciado dentro do *MatLab©* pela linha de comando do mesmo através de uma função pré-definida. Em conjunto ao arquivo .ptx, é definido na função o arquivo da implementação (nome_programa.cu), o número de Blocos (conjunto de *warps* que compartilham a memória cache L1), o número de *threads* por Bloco e uma variável, para o retorno do KDE da amostra.

Resultados e Discussão

No total constatou-se um *speed up*, ganho no desempenho computacional, extremamente elevado que escala juntamente com o tamanho da amostra, ou seja, quanto maior a amostra de dados maior é o ganho obtido com a utilização das placas gráficas (GPUs) através plataforma CUDA™.

Tendo como base uma amostra de 4 mil indivíduos, o ganho da implementação em paralelo executado na ferramenta *MatLab* já é considerável com seus 0,01ms (milissegundos) contrastando com os 1,3 segundos obtidos em uma implementação sequencial; porém o real ganho se mostra quando há mais de 40 mil indivíduos, onde a implementação sequencial requer 57,768 segundos (em média) para calcular o KDE da amostra e a implementação realizada neste trabalho requer apenas 2ms (milissegundos) para chegar ao mesmo resultado, mostrando-se mais rápida do que a própria implementação em CUDA™, na linguagem de programação *C* a qual leva 0,13 segundos.

Todavia ainda existe um problema que ocorre quando a amostra não pode ser representada como um todo na memória de uma única vez devido ao seu tamanho, o que pode acontecer quando a amostra vem de uma análise de *Big Data*.

Conclusão

A partir da análise das implementações feitas e dos resultados obtidos neste trabalho, verifica-se que o uso da programação paralela, através das placas gráficas, mostra-se extremamente vantajoso tendo em vista o ganho computacional que as mesmas proporcionam a aplicação. Este trabalho também apresenta a importância da estimativa de densidade kernel e suas aplicações nas mais diversas áreas.

Para o futuro, espera-se melhorar ainda mais o nível de paralelismo dentro da placa assim como também distribuir a aplicação em uma rede de computadores a fim de utilizar vários níveis de paralelização esperando obter-se uma melhora na execução do KDE.

Referências Bibliográficas

MICHAILIDIS, P.D., MAGARITIS, K.G. Accelerating Kernel Density Estimation on the GPU Using the CUDA Framework. Applied Mathematical Sciences, Vol. 7, n. 30, HIKARI Ltd., 2013.

Karunadasa, N.P., Ranashinghe, Accelerating high performance applications with CUDA and mpi. International Conference on Industrial and Information System, p 331-336, dezembro 2009.

KIRK, D.B., HWU, W.W. Programming Massively Parallel Processors. Massachusetts: Elsevier Inc., 2010.

_____. CUDA Programming Guide Version 4.2. NVIDIA Corporation, California, 2010.