Paralelização do algoritmo de Método de Estimação Não-Paramétrico por Núcleo Estimador Multivariado (KDE) utilizando GPU/CUDA

Juan Emanuel Hipólito Valenzuela [1]; Raul Sena Ferreira [2]; Marcelo Zamith [3]

1. *Discente do Curso de Ciência da Computação, IM/UFRRJ;*
2. *Discente do Curso de Ciência da Computação, IM/UFRRJ; 3. Professor do DTL/IM/UFRRJ.*

Palavras-chave: KDE Paralelo; KDE Multivariante; CUDA; Computação Paralela.

Introdução

Estimar densidades sobre uma população é um trabalho importante onde o objetivo é pegar uma amostra finita de dados e fazer inferências sobre a função densidade de probabilidade em uma região, inclusive onde não se observam dados.

A estatística possui diversas técnicas que podem ser empregadas para este propósito, porém, para escolher uma delas é necessário saber com que tipo de dados vamos lidar.

Na estatística existem dois tipos de dados:

– Paramétricos: São dados provenientes de um tipo de distribuição de probabilidade onde são feitas inferências sobre os parâmetros da distribuição. Um exemplo de dados paramétricos seria uma distribuição normal onde a mesma possui uma média e uma variância especificados.

– Não-Paramétricos: Este tipo não depende de dados pertencentes a nenhuma distribuição particular. Em particular, eles podem ser aplicados em situações em que menos se sabe sobre o problema em questão. Um exemplo de dados não-paramétricos seria as coordenadas de alunos de uma determinada faculdade, ou seja, a distribuição tem a forma normal, tanto a média quanto a variância não foram especificadas.

Os dados que foram usados neste trabalho foram as coordenadas (latitude e longitude) dos alunos da própria UFRRJ, que cadastraram previamente seus CEPs, no momento da inscrição para ingressar na instituição.

Por causa da natureza não paramétrica dos dados, foi utilizado o método de Estimação Não-Paramétrico por Núcleo Estimador, mais conhecido como KDE.

O KDE possui duas abordagens, univariante e multivariante, o primeiro trabalha em cima de uma população cujo o número de variáveis a considerar seja 1. Já o segundo, trabalha com 2 ou mais variáveis, como no nosso caso a população possui 2 variáveis (latitude e longitude) então foi escolhido o KDE Multivariante para processar os dados.

Na estimativa da densidade kernel, a contribuição de cada ponto de dados é suavizada de um único ponto em uma região do espaço em torno dela. Agregando as contribuições suavizadas individualmente tem-se uma visão geral da estrutura dos dados e sua função densidade.

O custo para calcular o peso de densidade de cada ponto (PDF) pode ser alto dependendo da quantidade de pontos a ser estimado, pois o cálculo do KDE Multivariante leva em consideração o cálculo de um ponto em relação a todos os outros pontos da população, e no nosso caso tínhamos também 2 parâmetros(x e y) associado a cada indivíduo, tornando assim um problema de cálculo de matrizes, mais precisamente um somatório de um produtório.

O objetivo deste trabalho é melhorar o desempenho através da paralelização massiva dos passos da aplicação. Para tanto, foi desenvolvida uma versão paralela do KDE, para aproveitar o paralelismo disponível nas placas gráficas (GPU – *Graphics Processor Unit*) através da linguagem CUDA (*Compute Unified Device Architecture*), que devido a sua natureza SIMD (*Single Instruction Multiple Data*) mostra-se muito adequada a problemas desta classe, onde obtém um ganho significativo em performance frente a processadores convencionais, pois fornece quantidades superiores de unidades de processamento (**EICHENBERGER et al., ____;** SANDERS & KANDROT, 2010).

Posteriormente foi implementada uma versão otimizada deste algoritmo, onde o mesmo fez uso de técnicas de uso de memória compartilhada da própria GPU, obtendo assim resultados ainda melhores.

Metodologia

A principal característica da GPU é processar a mesma instrução sobre diferentes dados, logo, cada core da GPU pode executar uma iteração do KDE paralelamente, onde cada thread ficou responsável pelo cálculo de cada PDF. Utilizando o modelo de programação unificado CUDA (Compute Unified Device Architecture), criamos um programa Host, que gerenciou a troca de informações com a GPU e execução do algoritmo em GPU (chamada de Kernel).

A programação de um kernel exige 4 passos: Primeiro reservamos espaço na memória da placa gráfica, em seguida os dados são copiados da memória para a placa gráfica, depois o código é executado na GPU e o resultado é então copiado de volta para a CPU.

A capacidade máxima da placa é definida pela quantidade de blocos e quantidade máxima de threads por bloco disponíveis na GPU.

Visando a utilização máxima de memória da placa o número de blocos e threads foram divididos em 28 e 512 respectivamente.

Resultados e Discussão

Usamos o KDE paralelizado do Matlab como parâmetro de bom desempenho, o motivo é que o Matlab é uma ferramenta consagrada e amplamente utilizada para cálculos numéricos diversos. Vários algoritmos utilizados pela ferramenta conseguem obter resultados com desempenho melhor até do que códigos tradicionais implementados em C/C++ e Java.

A máquina utilizada no experimento tem as seguintes configurações:

Processador: Intel I5 2500; Placa gráfica: GeForce GTX 650 Ti Boost;

Nº de cores: 4 (Hyperthread)

Neste trabalho, utilizamos 4 abordagens do KDE Multivariante de 2 dimensões:

(KDE sequencial, KDE paralelizado do Matlab, KDE com CUDA e KDE com CUDA otimizado), com uma instância de mais de 14.000 pontos(x,y), cujos resultados são mostrados na **Tabela 1.**

Os algoritmos foram executados 3 vezes e posteriormente foi tirado a média. Os tempos das implementações foram obtidos utilizando a função clock() da linguagem C e na implementação paralela foi usada a função cudaEventRecord().

Tabela 1. Tempos medidos.

KDE Sequencial	KDE Paralelizado Matlab	KDE c/ CUDA	KDE c/ CUDA otimizado	Speed-Up (GPU x Serial)	Speed-Up (GPU x Matlab)
31.088s	**6.355s**	**1.680s**	**1.028s**	**30,241**	**6,181**

O *speed-up* foi utilizado como medida de comparação entre os tempos de execução sequencial e paralelo (**HENNESSY & PATTERSON,2011**).

Houve um ganho de 3024% da versão de GPU-Otimizada sobre a sequencial, esta também se mostrou 6 vezes mais rápido do que o KDE paralelo do Matlab.

Conclusão

Através deste trabalho foi possível concluir que a utilização da programação paralela em GPU melhorou o processamento dos cálculos do KDE drasticamente e funcionou perfeitamente com este algoritmo, já que é um algoritmo paralelizável e pôde fazer uso do que a GPU tem de melhor, o cálculo de grandes quantidades de dados utilizando o mesmo código, além disso, esta técnica pode ser usada em diversas aplicações com abordagem semelhante ao utilizado neste trabalho.

Referências Bibliográficas

KIRK D.B., HWU W.W.. Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufman, 2nd ed., 2012

FERREIRA M.R.P.. Método Kernel: Estimação de densidades e classificação de padrões. Departamento de estatística, UFPB. Disponível em <http://www.cin.ufpe.br/~fatc/AM/kernel.pdf>. Acesso em Junho de 2014.

EPANECHNIKOV, V.A. (1969). "Non-parametric estimation of a multivariate probability density". Theory of Probability and its Applications 14: 153–158.

WAND, M.P; JONES, M.C. (1995). Kernel Smoothing. London: Chapman & Hall/CRC.

SANDERS, J. and KANDROT, E.. CUDA by Example: an introduction to general-purpose GPU programming. Addison-Wesley Professional. 2010.