Algoritmo K-Means em GPU com Mitigação de Atômicos

Bruno H. Meyer[1], **Wagner M. Nunan Zola** [1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

{bhm15, wagner}@inf.ufpr.br

Resumo. *Algoritmos de clustering como o K-Means são usados em diversos problemas. Neste trabalho, apresentamos uma implementação do algoritmo K-Means com todas as suas etapas paralelizadas em GPU. Foi possível observar dentre os experimentos realizados que existem vantagens consideráveis na implementação proposta, onde a privatização de variáveis em shared memory diminui a contenção de operações atômicas em memória global, demonstrando reduzir o tempo de execução consideravelmente.*

1. Introdução

O problema de *clustering* ou agrupamento é um desafio na área de aprendizado de máquina. Atualmente existem diversos problemas e bases de dados com grandes volumes de informações, onde se utilizam diversos algoritmos de agrupamento como o K-Means [Berkhin 2006]. Entretanto, a quantidade de dados que precisam ser processados podem necessitar da adaptação de tais algoritmos para que o tempo de execução seja reduzido. O problema de *clustering* consiste na divisão de diversos pontos (em um número qualquer de dimensões) em dois ou mais grupos.

O algoritmo K-Means depende do estabelecimento de um parâmetro K, que representa o número de grupos que serão considerados na solução do problema. O estabelecimento desse parâmetro pode ser automatizado por diversas técnicas que podem demandar a execução do algoritmo várias vezes, apresentando um maior custo computacional e portanto justificando a motivação para implementações otimizadas do K-Means, como versões que utilizam GPU [Zechner and Granitzer 2009]. Neste trabalho é apresentada uma implementação totalmente paralelizada com GPU do K-Means, e os resultados obtidos a partir de sua aplicação em diferentes bases de dados.

2. Fundamentos teóricos

O K-Means tem como entrada vários pontos pertencentes a um espaço multidimensional. O resultado do algoritmo será a atribuição de um valor (rótulo) para cada ponto, onde tal valor representará um agrupamento (*cluster*) de pontos. Além desses rótulos, o algoritmo busca encontrar o ponto médio dos pontos com mesmo rótulo, que são denominados centroides.

A Figura 1 ilustra o funcionamento do K-Means para duas dimensões, que consiste em um algoritmo iterativo onde as iterações possuem duas etapas principais: a rotulação de todos os pontos e a atualização dos centroides. Na etapa de rotulação, é calculada a distância desses pontos para cada um dos centroides e após isso, são escolhidos como rótulos os índices relacionados aos centroides com menores distâncias aos pontos. A

etapa de atualização dos centroides busca encontrar o ponto médio dos pontos que foram rotulados com o mesmo índice e após isso, atualizar os novos valores dos centroides com os pontos médios calculados. Ao final de cada iteração, o algoritmo verifica uma condição que pode ser o número máximo de iterações ou a verificação de convergência que determinará a continuação do algoritmo. O algoritmo é também dependente da inicialização dos centroides, o que é um dos fatores fundamentais para a convergência do mesmo.

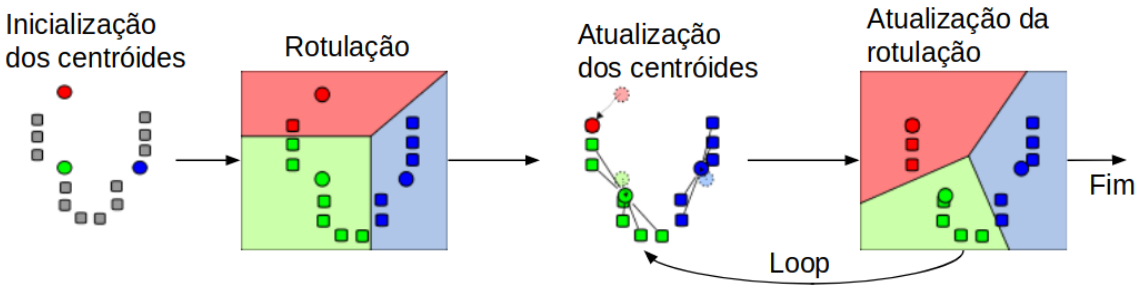


Figura 1. Ilustração do funcionamento do algoritmo K-Means. Fonte: Retirado e adaptado de `https://en.wikipedia.org/wiki/K-means_clustering`.

3. Metodologia

Os experimentos foram realizados em um processador Intel Xeon E5620 com clock 2.40Ghz e com GPU GTX 750 Ti que possui 5 multiprocessadores CUDA e um número máximo de *threads* por bloco igual a 1024. As implementações paralelas foram compiladas com a versão 9.1 de CUDA. Os tempos de execuções das etapas de rotulação e atualização de centroides foram calculados em cada iteração dos experimentos realizados. Por não ser objetivo deste trabalho, a convergência do algoritmo não foi levada em consideração pois a versão paralela e sequencial devem executar o mesmo número de iterações. Foi estabelecido um máximo de 100 iterações e os menores tempos de cada etapa do algoritmo foram salvos. Deve-se considerar que o computador utilizado para a execução dos experimentos realizados é compartilhado entre múltiplos usuários, o que pode causar pequenas variações nas execuções dos algoritmos.

Parte dos dados utilizados foram produzidos artificialmente devido à fácil manipulação das características dos problemas de clusterização como o número de dimensões, número de *clusters* e total de pontos. Também, foi utilizado um conjunto de dados reais, retirados do repositório público UCI [Asuncion and Newman 2007], que contém 5819 pontos com 33 dimensões que representam o desempenho de estudantes da Gazi University in Ankara (Turquia).

3.1. Implementação

O algoritmo K-Means foi implementado em uma versão sequencial na linguagem C++, e após isso houve a construção de duas versões paralelas onde a etapa de rotulação e de atualização dos centroides foram paralelizadas. Uma das versões paralelizadas conta com a execução da etapa de rotulação em GPU e a etapa de atualização de centroides em CPU, enquanto a outra versão realiza ambas as partes das iterações em GPU.

Para as duas versões mencionadas, foram necessárias operações atômicas nas etapas de rotulação e atualização dos centroides. A estratégia utilizada para computar os

dados foi o modelo de *threads* persistentes [Gupta et al. 2012], onde é definido um conjunto fixo de blocos que são responsáveis por subconjuntos dos dados com tamanhos equivalentes. Na versão em que as duas etapas da iteração foram paralelizadas não foi necessário transferir dados entre a CPU e a GPU, o que pôde aumentar a aceleração do algoritmo ao reduzir o *overhead* de comunicação.

3.2. Análises

As duas versões paralelizadas foram comparadas utilizando a base de dados real e uma base artificial composta por 10000 pontos em 21 dimensões e o algoritmo foi executado utilizando o valor de K como 512.

As experiências mostradas na Figura 2 contaram com a avaliação da execução das implementações totalmente paralelizadas em GPU para diferentes números de blocos de *threads*, onde foi utilizada uma base de dados artificial com 50000 pontos e 21 dimensões e $K = 512$, executando o algoritmo 5 vezes variando-se quantidades de blocos entre os valores 1 e 5. Também nesse experimento, foi comparado o uso das operações atômicas em *shared memory* (GPUs) e memória global (GPUg), que acontecem na etapa de rotulação e atualização de centroides. Para realizar a privatização dos dados envolvidos nas operações atômicas em cada bloco de *threads*, foi necessário realizar a sincronização dos blocos, que necessita um número menor de operações atômicas globais em relação à implementação que não realiza a privatização.

Outros experimentos realizados consistem na análise do desempenho das duas versões paralelizadas implementadas com a versão sequencial. Para isso, os algoritmos foram executados diversas vezes modificando o parâmetro K, que variou entre os valores 2, 8, 32, 64, 128 e 256. Foram gerados 2048 pontos com 16 dimensões.

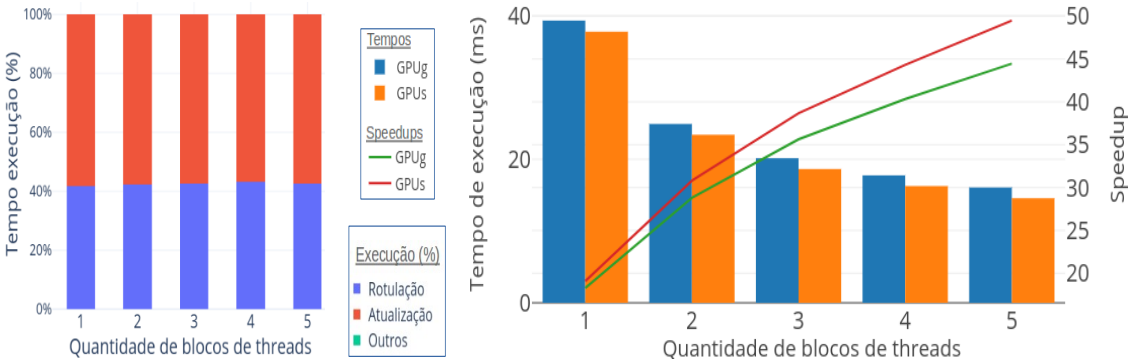
4. Resultados e discussão

A Figura 2 ilustra os resultados obtidos em relação ao ganho de desempenho entre a versão sequencial e a paralelizada utilizando diferentes quantidades de blocos (eixo das abscissas). Percebe-se uma variação considerável no tempo de execução em relação à quantidade de blocos utilizados. Na Figura, é possível observar a escalabilidade da implementação, que tem seu *speedup* máximo quando a quantidade de blocos é igual ao número de multiprocessadores da GPU utilizada.

Foi observado que a aceleração máxima obtida na etapa de rotulação foi aproximadamente 113, enquanto a etapa de atualização do centroide teve o *speedup* máximo próximo a 3. No total, como indicado na Figura, a versão GPU, comparada à versão CPU, chega a ser 50 mais rápida como mostra a Figura 2b.

Vemos também nessa figura um resultado considerável, onde a implementação que utiliza *shared memory* para efetuar operações atômicas (GPUs) tem uma aceleração maior do que a implementação onde se utiliza diretamente a memória global (GPUg).

Quando a implementação proposta foi aplicada à base de dados reais, percebeu-se que assim como no experimento descrito anteriormente, a atualização dos centroides é um fator limitante para a aceleração do algoritmo como mostrado na Figura 2a, que apresenta a etapa de atualização como a maior parte do tempo de execução nas iterações do algoritmo. Considerando que o *speedup* máximo observado na etapa de rotulação



(a) Execução de kernels (%)

(b) Tempos totais e Aceleração em relação à versão sequencial

Figura 2. Análise da variação dos blocos na execução do algoritmo implementado

foi 70, enquanto o *speedup* máximo das iterações completas (que incluem a etapa de rotulação) foi 15. Também é interessante mencionar que a versão onde a atualização dos centroides é feita em CPU apresentou *speedup* inferior aos casos onde esta etapa do algoritmo é realizada em GPU, o que pode ser explicado pelo fato de que quando ambas etapas são feitas em GPU, não há necessidade de transferência de dados entre CPU e GPU.

Para a base de dados sintética, em relação à implementação híbrida (CPU+GPU) que atualiza os centroides em CPU, foi possível observar que no intervalo entre 8 e 64 do parâmetro K houve um decaimento considerável no *speedup*. Isso pode ser explicado pelo fato da implementação híbrida depender da transferência de dados entre GPU e CPU, sendo que a quantidade de dados transferida varia com o parâmetro K.

5. Conclusões

Este trabalho apresenta uma implementação do algoritmo K-Means com a variação de duas versões: uma versão que paraleliza somente a etapa de rotulação e outra versão onde a etapa de rotulação e atualização dos centroides são paralelizadas. Assim como os diferentes trabalhos relacionados, a etapa de rotulação parece bem escalável, sendo um desafio otimizar a etapa de atualização dos centroides. Foram utilizadas operações atômicas para parte das implementações utilizando a *shared memory* da GPU, característica que apresentou resultados melhores em relação ao acesso direto na memória global.

Referências

Asuncion, A. and Newman, D. (2007). Uci machine learning repository. `https://archive.ics.uci.edu/ml/datasets/Turkiye+Student+Evaluation`.

Berkhin, P. (2006). A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer.

Gupta, K., Stuart, J. A., and Owens, J. D. (2012). A study of persistent threads style gpu programming for gpgpu workloads. In *2012 Innovative Parallel Computing (InPar)*.

Zechner, M. and Granitzer, M. (2009). Accelerating k-means on the graphics processor via cuda. In *Intensive Applications and Services, 2009. INTENSIVE'09. First International Conference on*, pages 7–15. IEEE.