Universidade Estadual de Campinas
Instituto de Computação

Programação paralela com algoritmo K-means em plataforma CUDA

MO644 - Programação Paralela

Alex Silva Torres, RA 161939
Thiago José Mazarão Maltempi, RA 180070

Campinas, SP - Junho de 2015

1 Problema inicial

K-means é um algoritmo de agrupamento (clustering), que são úteis nos cenários de big data e data mining. Algoritmos de agrupamento são utilizados para construção de grupos de objetos com base nas semelhanças e diferenças entre os mesmos, de tal maneira que os grupos obtidos sejam os mais homogêneos possíveis.

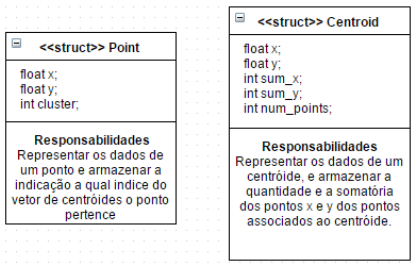
Este algoritmo possui quatro passos principais dependentes entre si. Cada passo possui iterações e a quantidade destas iterações depende diretamente dos dados de entrada do algoritmo, podendo tornar sua execução demorada em casos de muitos dados. Porém as iterações dentro de cada passo não possuem dependências com a iteração anterior, ou seja, este algoritmo apresenta apenas laços do-all, se tornando um ponto muito relevante para a paralelização deste problema, pois abre a possibilidade de distribuir estas iterações em partes para vários processos trabalhando em paralelo, sem se preocupar com a ordem em que estas iterações serão executadas.

O objetivo deste trabalho é criar uma implementação do K-means que torne a execução do algoritmo mais rápida, utilizando técnicas de programação paralela em GPU Nvidia com a plataforma CUDA.

Há um repositório de código fonte livre [1], que possui implementações do K-means em várias linguagens de programação, com o objetivo de medir a eficiência de cada linguagem executando o mesmo algoritmo. O resultado deste trabalho servirá de contribuição para este repositório e também para que qualquer pessoa possa reaproveitar o código bastando apenas algumas modificações no código fonte.

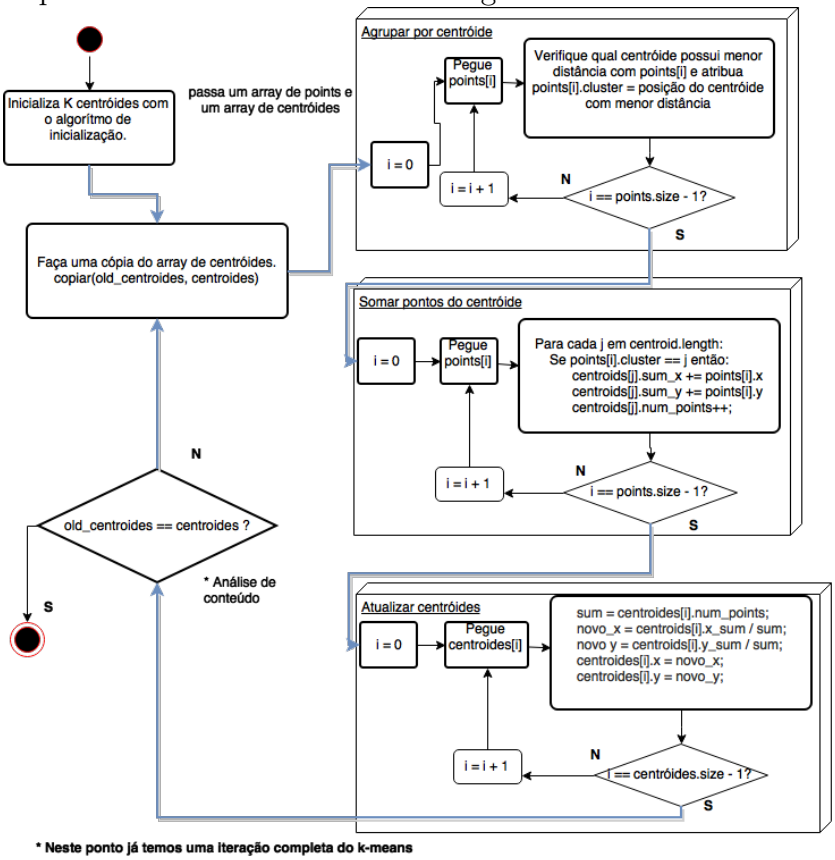
2 Solução sequencial

A solução sequencial foi desenvolvida utilizando a linguagem C, seguindo duas estruturas básicas: points e centroids, conforme segue o diagrama a seguir:



[1] Disponível em github.com/andreaferretti/kmeans

O algoritmo K-means possui quatro passos principais que são dependentes entre si, sendo eles: “agrupar por centróide”, “somar pontos dos centróides”, “atualizar centróides” e “comparar com última iteração”, respectivamente. Estes passos estão detalhados no fluxograma abaixo.



3 Profiling

Foi analisado o algoritmo implementado sequencialmente em CPU, através da ferramenta Intel Vtune, com a massa de dados de 100.000 pontos iniciais e 10 centróides em busca dos principais hotspots do programa, sendo estes:

1. “Agrupar por centroide” - 84,33% do tempo de execução;
2. “Somar pontos dos centróides” - 11,86% do tempo de execução;
3. Outras funções - 3,81% do tempo de execução.

As duas principais funções que juntas somam 96,19% do algoritmo, deve-se ao grande número de iterações, que é dada pela quantidade de pontos

multiplicado pela quantidade de centróides, para todos os casos. E em especial o passo "agrupar por centróide", precisa executar a fórmula da distância euclidiana para todos os casos, o que faz aumentar consideravelmente o tempo de execução desta função.

4 Solução paralela

Foi considerado como ganho utilizar programação paralela em GPU e plataforma CUDA, pelo grande número de threads que a GPU pode oferecer, e as APIs da plataforma CUDA como por exemplo, math, operações atômicas etc. Por outro lado, foi pensado nos contras da plataforma, que exige cópias de memória RAM (host) para memória da GPU (device).

4.1 Paralelizando "agrupar por centroid"

Foi criada uma thread para cada ponto, e cada thread possui a responsabilidade de executar o passo "agrupar por centróide" inteiro. Para se fazer a distância euclidiana foram utilizadas operações matemáticas da API math oferecida pela plataforma CUDA.

4.2 Paralelizando "Somar pontos dos centróides"

A função "somar pontos dos centróides", assim como "agrupar por centróide", itera todos os pontos, e é responsável por somar todos os pontos referentes a cada centróide e atribuir estes valores a eles.

Esta função possui o problema de concorrência das threads, que precisam escrever nas mesmas posições de um vetor. Para resolver este problema, foi utilizada a função atomicAdd() da plataforma CUDA.

4.3 Paralelizando "atualizar centróides"

Esta função tende a ter um baixo número de iterações e suas operações são de baixa complexidade. Mesmo assim vale a pena mantê-la em GPU, para que evite uma transferência de dados da memória entre device e host. Também foi pensado em um cenário em que possa haver muitos centróides, então, a paralelização desta função pode ser considerada relevante.

5 Resultados

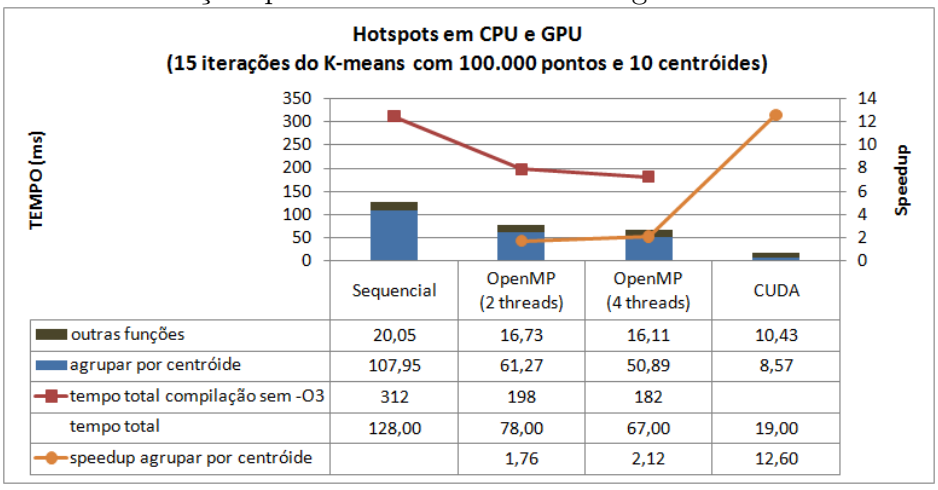
Foram medidos os tempos de execução utilizando massas de 10.000, 100.000 e 1.000.000 de pontos, utilizando 10, 100 e 1.000 centróides cada e 15 iterações do k-means. Para que seja possível obter a média do tempo gasto, foram feitas 100 execuções de cada cenário e tirado a média aritmética de cada um deles. Todos os testes foram executados em uma CPU Intel® Core™ i7-4500U CPU @ 1.80GHz 4 e em uma GPU Nvidia GeForce GT 740M com 2048 MB e 384 CUDA cores.

É notável o ganho da implementação em CUDA em relação ao sequencial. Especialmente para execuções que houveram maior número de pontos e/ou centróides. Graças a esta implementação, foi possível chegar a um speedup de 8. Na tabela 1 pode ser acompanhado com mais detalhes cada um dos cenários.

Tabela 1: Comparativo de 15 entradas do k-means com diferente entradas

	10.000 Pontos			100.000 Pontos			1.000.000 Pontos		
Quantidade de centroides	Sequencial(ms)	Cuda(ms)	SpeedUP	Sequencial(ms)	Cuda(ms)	SpeedUP	Sequencial(ms)	Cuda(ms)	SpeedUP
10	13,19	5,05	2,61	128,67	18,96	6,78	1348,29	165,94	8,12
100	88,54	12,18	7,26	886,19	98,55	8,99	8883,27	960,49	9,24
1000	774,17	104,51	7,40	7790,62	936,73	8,31	77478,00	9310,90	8,32

Para que fosse possível comparar a capacidade de paralelismo entre uma GPU e uma CPU para este problema, foi implementado o principal hotspot do algoritmo ("agrupar por cluster") em OpenMP. Foi medido o speedup considerando o apenas os tempos gastos da função "agrupar por cluster". Os detalhes das medições podem ser visualizados no gráfico abaixo.



6 Considerações finais

O modelamento de um algoritmo sequencial ajudou a entender o problema, isso pode ser visto, quando comparado aos algoritmos do repositório que nós decidimos contribuir, o algoritmo serial proposto é quase duas vezes mais rápido do que o atual do repositório.

A implementação em C CUDA obteve um resultado esperado com um speedup maior que 8, enquanto o mesmo código foi testado em uma grid NVidia K520, com 3072 CUDA cores, obteve speedup de aproximadamente 25 vezes (utilizando o cenário de 1.000.000 de pontos e 1.000 centróides). Sendo assim, consideramos o objetivo proposto por este projeto atingido.

Como proposta futura, planejamos estender a dimensão dos pontos de 2 para N, tornando esta implementação uma API open source. Além disso, o projeto possui encorajamento da Fundação CPqD, que planeja utilizar processamento do K-means em GPU em suas bibliotecas de reconhecimento biométrico de voz e face.