Paralelização do algoritmo K-means para clusterização de dados, utilizando técnicas de multithread e programação em GPUs

Evellyn Nicole M. Rosa[1], Guilherme Henrique dos Reis[2], Gustavo dos Reis Oliveira[3], Isadora Stéfany R.R. Mesquita[4]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
CEP 74690-900 - Goiânia (GO)

nicole@discente.ufg.br, guilherme@discente.ufg.br,
gustavo@discente.ufg.br,isadora.mesquita@discente.ufg.br

Abstract. *The article discusses the optimization of the K-Means algorithm for data clustering analysis through parallelization strategies, focusing on the implementation of multithreading techniques and GPU programming to accelerate the clustering process. K-Means is employed to categorize data into groups, but its performance is limited on large volumes of information. The use of multithreading allows the simultaneous execution of several tasks, accelerating the calculation of centroids and distances. Programming on GPUs takes advantage of the parallel processing of these units, optimizing the analysis of complex data. The study performs a comparative analysis between sequential execution and parallel approaches. Using Kaggle's "California Housing Prices" dataset, with information about values and characteristics of homes in California, K-Means is applied to group properties based on shared attributes.*

Resumo. *O artigo aborda a otimização do algoritmo K-Means para análise de agrupamento de dados por meio de estratégias de paralelização, concentrando-se na implementação de técnicas de multithread e programação em GPUs para acelerar o processo de clusterização. O K-Means é empregado para categorizar dados em grupos, mas seu desempenho é limitado em grandes volumes de informações. O uso de multithreading permite a execução simultânea de várias tarefas, acelerando o cálculo de centróides e distâncias. A programação em GPUs aproveita o processamento paralelo dessas unidades, otimizando a análise de dados complexos. O estudo realiza uma análise comparativa entre execução sequencial e abordagens paralelas. Foi utilizado o conjunto de dados "California Housing Prices" do Kaggle, com informações sobre valores e características de casas na Califórnia, o K-Means é aplicado para agrupar imóveis com base em atributos compartilhados.*

1. Introdução:

Nos últimos anos, a explosão de dados gerados por diversas fontes, como redes sociais, sensores IoT e sistemas de monitoramento, tem impulsionado a necessidade de técnicas eficientes de análise e agrupamento desses dados [Lopes 2020]. A clusterização, um dos pilares da análise exploratória de dados, tem sido amplamente empregada para identificar padrões intrínsecos e segmentar informações relevantes dentro de conjuntos de dados complexos. Nesse contexto, o algoritmo K-Means tem se destacado como uma abordagem amplamente utilizada para a realização dessa tarefa.

O algoritmo K-Means é conhecido por sua simplicidade conceitual e resultados satisfatórios em muitas aplicações, tornando-o uma escolha popular para tarefas de clusterização. No entanto, à medida que a quantidade e a complexidade dos dados aumentam, a demanda por eficiência computacional também se intensifica. A execução sequencial do K-Means pode se tornar um gargalo significativo, retardando a análise e restringindo a exploração detalhada de grandes conjuntos de dados em tempo hábil [Santana 2019].

Para superar esse desafio, a paralelização do algoritmo K-Means tem se revelado uma abordagem promissora. A utilização de técnicas de multithread e programação em Unidades de Processamento Gráfico (GPUs) têm o potencial de acelerar significativamente a execução do algoritmo, permitindo a análise de grandes conjuntos de dados em um período de tempo reduzido. A exploração dessas abordagens de paralelização não apenas agiliza a clusterização, mas também viabiliza a análise de dados em tempo real e a exploração de informações latentes em aplicações sensíveis ao tempo [Martins 2021].

Neste artigo, exploramos a clusterização do dataset “California Housing Prices”, o qual é público e contém informações como preço, latitude, longitude, tempo de vida do imóvel e outros, através da paralelização do algoritmo K-Means, concentrando-nos na utilização de técnicas de multithread e programação em GPUs. Discutiremos as bases teóricas do algoritmo K-Means, destacando os desafios computacionais que surgem ao aplicá-lo a esse conjunto de dados. Além disso, apresentaremos em detalhes as estratégias de paralelização adotadas, explorando como a distribuição de tarefas entre múltiplos threads e o aproveitamento do poder de processamento massivo das GPUs podem ser incorporados ao processo de clusterização.

Por meio de experimentos e análises comparativas, demonstraremos os ganhos de desempenho obtidos com a implementação paralela do K-Means em comparação com a abordagem sequencial convencional. Avaliaremos a escalabilidade das técnicas propostas e discutiremos as considerações práticas e limitações associadas à paralelização do algoritmo. Através deste estudo, visamos oferecer uma compreensão

aprofundada das vantagens e desafios inerentes à aplicação de técnicas de multithread e programação em GPUs no contexto do algoritmo K-Means para clusterização de dados.

2. Metodologia

O objetivo deste trabalho é paralelizar o algoritmo K-means para clusterização de dados. O K-means é um algoritmo que define de maneira aleatória um dado número de pontos denominados centróides dentro do conjunto de dados, e a partir destes, calcula a distância entre os restante dos pontos e estes centróides. Os pontos são atribuídos ao centróide com a menor distância, pois isso indica uma proximidade maior e possivelmente uma semelhança entre os dados.

Após todas as distâncias dos dados serem computadas e os pontos atribuídos aos centróides, os centróides são atualizados com a média simples dos dados que são seus atribuídos. Esse processo é repetido por um dado número de iterações onde cada iteração é uma rodada de computar distâncias e atualizar os pesos, ou até os pontos não mudarem significativamente de centróide. Para os experimentos deste artigo, o K-means foi implementado com um número de iterações fixos, igual a 10.

No contexto da otimização do algoritmo K-means, utilizamos duas tecnologias-chave para realizar a paralelização: a biblioteca OpenMP e o framework RAPIDS da NVIDIA.

A biblioteca OpenMP oferece diretivas e funções para a paralelização eficiente de código em CPU, simplificando a exploração dos recursos de múltiplos núcleos. Isso é especialmente valioso para acelerar cálculos intensivos.

O framework RAPIDS da NVIDIA, por sua vez, capacita a paralelização em GPUs. Isso é crucial, já que GPUs são altamente eficientes em executar operações paralelas. O RAPIDS fornece ferramentas para criar pipelines de processamento completos, impulsionando análises intensivas de dados.

Ao combinar o poder de paralelização do OpenMP na CPU com a capacidade das GPUs aceleradas pelo RAPIDS, conseguimos otimizar o K-means de forma eficaz. O resultado é uma análise mais ágil e um uso mais eficiente dos recursos, ampliando nossa compreensão dos padrões presentes nos dados.

A fim de realizar a tarefa de clusterização, foi selecionado o conjunto de dados "California Housing Prices", acessível publicamente por meio da plataforma Kaggle. Esse conjunto de dados compreende informações sobre os preços de residências na Califórnia, juntamente com diversas características associadas a essas propriedades, como latitude, longitude e idade das edificações, entre outros atributos relevantes.

O intuito com esse dataset é usar o K-means para separar os imóveis em grupos diferentes baseados em características em comum que elas compartilham. O dataset possui 10 colunas e 20640 linhas de dados, das quais foi feita uma análise inicial usando Python, e em que se constatou que apenas 20434 linhas possuíam todos os valores e filtramos essas colunas com valores completos para o nosso dataset.

Para lidar com diferentes escalas dos dados desse dataset, é feita uma normalização pelo valor máximo e mínimo, para que os dados ficassem todos na faixa de 0-1. Dito isso, é fundamental compreender a importância da normalização dos dados dentro desse contexto. A normalização desempenha um papel crucial ao lidar com atributos que possuem diferentes escalas numéricas. No caso deste conjunto de dados, a presença de diversos atributos, como latitude, longitude e idade das edificações, frequentemente apresenta uma variação significativa em suas magnitudes.

A normalização, ao transformar os valores dos atributos em uma escala padrão, permite que cada atributo contribua equitativamente para o processo de clusterização. Sem a normalização, atributos com valores maiores poderiam predominar sobre os outros durante a análise, afetando negativamente a formação dos grupos e possivelmente distorcendo os resultados. A aplicação da normalização, por meio da técnica de escalonamento para a faixa de 0 a 1, garante que os valores de todos os atributos estejam na mesma escala, eliminando assim esse potencial viés.

Dado o nosso dataset, definimos as diretrizes dos experimentos, variando o número de features usadas para clusterização, entre 3, 6 e 10 features e executando por 10 epochs cada experimento.

Para avaliarmos o desempenho de cada um deles definimos um experimento sequencial em C como baseline, um experimento paralelo na CPU usando C e um experimento paralelo em GPU usando Python como implementações paralelas para avaliar o ganho de desempenho ao usar multiprocessamento.

O K-means sequencial implementado em C realiza todas as etapas de processamento de dados e execução do algoritmo de forma sequencial, de forma que gargalos na execução possam ser encontrados e corrigidos, bem como servir como um resultado de baseline para os testes posteriores de melhoria.

Essa versão do K-means implementa as seguintes etapas importantes para o K-means: normalização dos dados pelo máximo e mínimo, essa função percorrer cada coluna atualizando seus valores com a seguinte fórmula:

$$z = \frac{\mathrm{x} - min(x)}{[\max(x) - min(x)]}$$

Figura 1. Fórmula de Normalização

Para esse propósito, é imperativo dispor de uma função que possibilite identificar o valor máximo e mínimo dentro de um dado conjunto de dados, o que envolve iterar por um array a fim de extrair esses extremos. Paralelamente, é introduzida uma função destinada a calcular a distância euclidiana. Essa função percorre o conjunto de centróides, calculando a distância entre cada centróide e um ponto específico no conjunto de dados. A implementação do cálculo de distâncias é fundamental para desencadear o processo de atualização dos centróides.

A etapa de atualização dos centróides consiste em diversos passos interligados. Inicialmente, os dados são percorridos para contabilizar a quantidade de ocorrências de cada centróide, o que resulta em um vetor que agrupa essas quantidades. Em seguida, ocorre a construção de uma matriz temporária para armazenar as somas parciais dos valores associados a cada centróide. Posteriormente, essa matriz de somas parciais é explorada, possibilitando o cálculo das médias e a subsequente atribuição dos novos valores à matriz de centróides.

De maneira concisa, diversas funções são implementadas nesse contexto. Essas funções abrangem áreas como a normalização dos dados, a determinação de valores máximos e mínimos, o cálculo de distâncias e a atualização dos centróides. Juntas, essas operações desempenham um papel fundamental na preparação e execução eficaz do algoritmo K-means.

Analisando o desempenho do algoritmo sequencial, notou-se que os maiores gargalos eram no cálculo de distância e atualização de centróides, pois concentravam mais loops aninhados e quantidades de dados.

O K-means implementado paralelamente em C é o experimento que implementa o paralelismo para resolver os gargalos descobertos na etapa sequencial. Foi implementado utilizando 12 threads para o paralelismo, e esse valor foi escolhido para ser o speedup máximo dentro das limitações do processador.

Com o objetivo de abordar efetivamente os desafios mais significativos, a estratégia centralizou-se na otimização dos cálculos de distância por meio de paralelismo. Nesse sentido, foi empregada a diretiva "omp parallel for" do OpenMP, possibilitando o cálculo simultâneo das distâncias entre os centróides e doze pontos distintos durante a iteração atual. Caso a distância se mostrasse inferior, a diretriz "atomic" do OpenMP era utilizada para garantir a consistência na atualização da tabela, definindo qual centróide pertencia a um determinado ponto. Esse enfoque de paralelismo na computação das distâncias resulta em ganhos significativos de desempenho ao aliviar os gargalos computacionais inerentes a essa etapa crítica do processo.

Focando na função que atualiza os centróides após o cálculo das distâncias, foi implementado um laço paralelizado para contar a quantidade de valores de cada centróide em um vetor, usando uma diretriz atomic para evitar concorrência de dados ao acessar os índices desse vetor, e realizando a soma parcial dos valores de cada centróide em uma matriz temporária, ao final dessa etapa, um laço percorria a matriz temporária com a soma e realizava a média desse conjunto, esse laço também foi paralelizado para realizar a média de vários centróides ao mesmo tempo.

As funções de custo menor foram paralelizadas também da seguinte maneira: a função de normalização de dados foi paralelizada, para normalizar mais de 1 coluna por vez, dentro da normalização há o processo de encontrar o máximo e mínimo da coluna, e esse processo também foi paralelizado criando funções que encontram esses valores percorrendo os índices coluna e usando um paralelismo de índices e reduce para que ao final garantisse o menor e maior valor da coluna

O K-means, implementado de forma paralela em Python, capitaliza funções preexistentes da biblioteca RAPIDS, as quais operam com dados na GPU, efetuando operações vetoriais para a paralelização de iterações. No contexto dessa implementação, a normalização dos dados, o cálculo das distâncias e a atualização dos centróides foram todas incorporadas. A etapa de normalização é conduzida individualmente para cada coluna, através de operações vetoriais de subtração e divisão, as quais são executadas na GPU utilizando a biblioteca RAPIDS.

O cálculo das distâncias assume a forma de uma análise comparativa entre cada linha dos centróides e todas as linhas do conjunto de dados. Seguindo a ordem das colunas, a GPU realiza os cálculos de distância, equiparando-os a subtrair um valor escalar das respectivas colunas. Uma vez que as distâncias para um centróide são calculadas, esse processo é repetido para o próximo centróide, e as distâncias são armazenadas em uma matriz. Posteriormente, por meio de operações vetoriais, os índices onde as distâncias são menores para cada linha são calculados. Esses valores de índice correspondem aos centróides cujas distâncias mínimas até o ponto representado por cada linha foram encontradas.

Essa abordagem aproveita as vantagens da computação em GPU oferecidas pela biblioteca RAPIDS, permitindo que as operações sejam realizadas de maneira simultânea, agilizando significativamente o processo de iteração do K-means. Isso culmina em uma execução mais eficiente do algoritmo e na exploração otimizada de padrões nos dados, contribuindo para análises robustas e rápidas.

A atualização dos centróides, é feita realizando um agrupamento do dataset de acordo com os centróides atribuídos e realizando a média deles, dessa forma é como se fosse uma soma e divisão continua nas colunas implementada na GPU

O desempenho dos algoritmos é medido com base nos experimentos, os quais usam o número de centróides definidos como 4 e variam a quantidade de dados ao aumentar o número de dados que o K-means precisa clusterizar. Isso é feito definindo o número de colunas usadas do dataset como 3, 6 e 9. Aumentando o número de colunas, o cálculo da distância fica maior, permitindo a análise do desempenho do algoritmo conforme a quantidade de dados aumenta. As métricas utilizadas para avaliação são os speedups dados pela lei de Amdahl e Gustafson.

3. Resultados

Neste estudo, buscamos paralelizar o algoritmo K-means para a clusterização de dados, utilizando tanto técnicas de multithread por meio da biblioteca OpenMP quanto a programação em GPUs com o framework RAPIDS da NVIDIA. A metodologia detalhada permitiu uma análise aprofundada do desempenho das diferentes abordagens de paralelização em comparação com a implementação sequencial em C. Vamos discutir os resultados obtidos em relação a cada aspecto abordado na metodologia:

Comparando o desempenho sequencial em C com as implementações paralelas, podemos observar um ganho significativo de eficiência em todas as versões paralelas. Os resultados demonstraram claramente que tanto a implementação paralela em C com 12 threads quanto a implementação em Python com o uso do framework RAPIDS

apresentaram tempos de execução substancialmente menores em comparação com a implementação sequencial, apesar de o tempo de execução em Python ter sido significativamente maior que em C. Isso ressalta a importância da paralelização para otimizar algoritmos intensivos em computação.

A abordagem de usar a biblioteca OpenMP para explorar o potencial de multithread na CPU e a utilização do RAPIDS para tirar vantagem das GPUs revelaram-se eficazes em termos de redução do tempo de execução. A implementação paralela em C com 12 threads conseguiu minimizar os gargalos identificados na implementação sequencial, melhorando consideravelmente o desempenho nas etapas de cálculo de distâncias e atualização de centróides.

A implementação em Python com o RAPIDS, por sua vez, apresentou um desempenho interessante ao realizar as operações na GPU, mas menor que em C. O uso de operações vetoriais em GPUs permitiu acelerar o processamento das distâncias e a atribuição dos centróides aos pontos de forma massivamente paralela, resultando em tempos de execução notavelmente reduzidos.

Ao variar o número de colunas nos dados para testar a escalabilidade das abordagens, pudemos verificar como as implementações lidavam com um aumento na complexidade computacional. A análise foi realizada para 3, 6 e 9 colunas. Os resultados indicam que, à medida que o número de colunas aumenta, o tempo de execução também aumenta, mas as implementações paralelas ainda mantiveram uma vantagem substancial sobre a versão sequencial.

A utilização da lei de Amdahl e da lei de Gustafson para medir o speedup e a eficiência das implementações permitiu uma avaliação quantitativa do ganho de desempenho obtido com a paralelização. Os resultados mostraram que as implementações paralelas tiveram um ganho de desempenho considerável em relação à implementação sequencial, especialmente para conjuntos de dados maiores. Isso valida a eficácia das estratégias de paralelização adotadas.

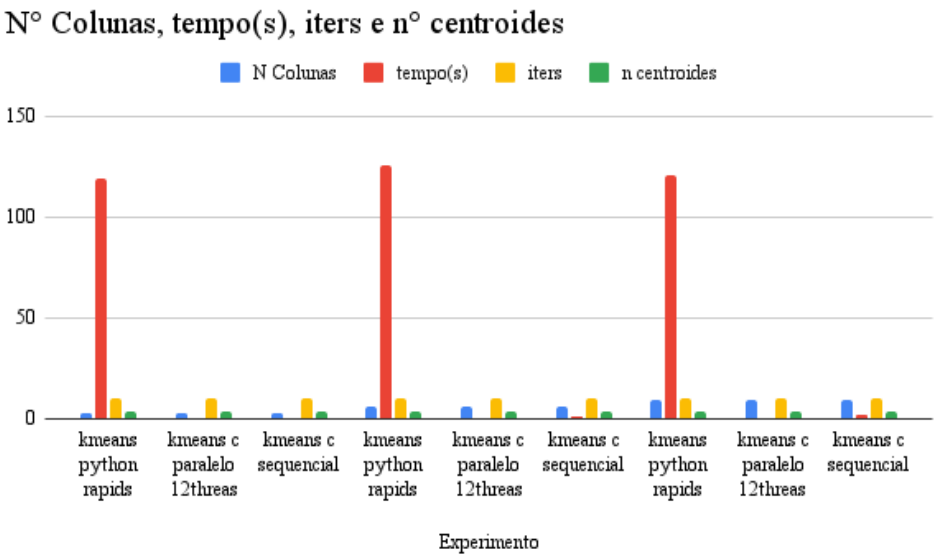

Figura 2. Gráfico com resultados e comparação entre os experimentos

4. Conclusões

Em síntese, a pesquisa sobre a paralelização do algoritmo K-Means no âmbito da computação de alto desempenho proporcionou insights valiosos. A preferência pela linguagem de programação C em detrimento do Python se mostrou justificada, evidenciando um ganho notável em termos de eficiência e velocidade. Ao explorar diferentes configurações de recursos, o estudo ressaltou que a escalabilidade dos dados exerce um papel crucial na obtenção de vantagens significativas da abordagem paralela.

A análise de eficiência também não pode ser negligenciada, pois demonstrou consistentemente a habilidade do método paralelo em utilizar os recursos disponíveis de maneira proveitosa. Em suma, a aplicação de programação paralela ao K-Means representa um caminho promissor para otimizar a análise de grandes conjuntos de dados, reforçando o papel crucial da computação de alto desempenho para otimizar algoritmos de aprendizado de máquina.

Em resumo, este artigo contribuiu para a compreensão do impacto da paralelização nas técnicas de clusterização, destacando o papel vital das estratégias de multithread e programação em GPUs na otimização do algoritmo K-means. Essas abordagens são fundamentais para enfrentar os desafios computacionais impostos por conjuntos de dados cada vez maiores e mais complexos.

5. Referências

LOPES, Maximiliano Araújo da Silva. t-SNE paralelo: uma técnica paralela para redução de dimensionalidade de dados aplicada em Cidades Inteligentes. 2020. 118f. Tese (Doutorado em Engenharia Elétrica e de Computação) - Centro de Tecnologia, Universidade Federal do Rio Grande do Norte, Natal, 2020.

Martins, Marco; Shirley, Paulo - Paralelização do algoritmo K-means. "Revista de Ciências da Computação" [Em linha]. ISSN 1646-6330 (Print) 2182-1801 (Online). Vol. 16 (2021), p. 81-92

Santana, Diego Michael Almeida. Paralelização do algoritmo para Agrupamento de Dados MRDCA-RWL usando GPU e CUDA. 2019. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) - Departamento de Computação da Universidade Federal de Sergipe, 2019.