Vinícius Henrique Almeida Praxedes

Paralelização de Algoritmos Notórios de Clustering em GPUs NVIDIA

Brasil

2019, Novembro

Vinícius Henrique Almeida Praxedes

Paralelização de Algoritmos Notórios de Clustering em GPUs NVIDIA

Monografia acadêmica apresentada à Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos necessários à obtenção do diploma do curso de Ciência da Computação

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Programa de Graduação

Orientador: Daniel Duarte Abdala

Brasil

2019, Novembro

Vinícius Henrique Almeida Praxedes
Paralelização de Algoritmos Notórios de Clustering em GPUs NVIDIA/ Vinícius Henrique Almeida Praxedes. – Brasil, 2019, Novembro-
0p. : il. (algumas color.) ; 30 cm.

Orientador: Daniel Duarte Abdala

Monografia – Universidade Federal de Uberlândia – UFU
Faculdade de Computação
Programa de Graduação, 2019, Novembro.

1. Palavra-chave1. 2. Palavra-chave2. 2. Palavra-chave3. I. Orientador. II. Universidade Federal de Uberlândia. III. Faculdade de Computação. IV. Título

Vinícius Henrique Almeida Praxedes

Paralelização de Algoritmos Notórios de Clustering em GPUs NVIDIA

Monografia acadêmica apresentada à Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos necessários à obtenção do diploma do curso de Ciência da Computação

Trabalho aprovado. Brasil, 24 de novembro de 2012:

Daniel Duarte Abdala
Orientador

Professor
Convidado 1

Professor
Convidado 2

Brasil
2019, Novembro

Sumário

1 Introdução

A busca pelo menor tempo de execução é uma diretriz ubíqua na computação. Desde os primórdios da área buscamos algoritmos e procedimentos que, dados os mesmos parâmetros de entrada, executem a mesma tarefa na menor quantidade de tempo possível. Outros recursos como espaço de memória utilizado, eficiência energética ou uso da rede em muitos cenários são mais importantes que o tempo de execução, mas ainda sim a mesma continua sendo um dos mais estudados parâmetros para categorização e avaliação de algoritmos e procedimentos na computação. De fato o tempo de execução – em ciclos, ou passos, de processamento – é a métrica utilizada na análise de uma das maiores incógnitas da computação, o problema P versus NP.

Um grande avanço na quantidade de poder de processamento dos computadores e, portanto, diminuição do tempo de execução de algoritmos, foi a paralelização dos processadores. A habilidade de poder executar duas ou mais ações simultaneamente possibilitou muitos ganhos palpáveis na velocidade de execução de algoritmos e procedimentos, porém introduziu uma necessidade de mudança na forma de se pensar em resoluções de problemas computacionalmente: paralelizar um algoritmo antes serial não é uma tarefa trivial, e requer cuidados especiais com concorrência no acesso a recursos da máquina, dependência de dados e cálculos, sincronização, entre outros dilemas.

Um dos componentes que mais utilizam da paralelização num computador moderno são as GPUs, unidades de processamento gráfico, ou placas de vídeo, que são basicamente processadores especializados em operações vetoriais, altamente paralelizadas, usualmente dedicados a operações gráficas, e com sua própria memória dedicada, a VRAM. Enquanto processadores de uso geral, CPUs, costumam ter no máximo dezenas de núcleos para processamento paralelo, GPUs possuem milhares ou até milhões de núcleos para operações vetoriais.

No entanto, cada vez mais está sendo descoberto e aproveitado o potencial de uso das GPUs em atividades não apenas voltadas para renderização, interfaces e outras operações gráficas, mas para a computação de propósito geral. Diversos algoritmos modernos e antigos beneficiam-se imensamente da paralelização enorme proporcionada por GPUs, e com ferramentas como a biblioteca e linguagem CUDA criada pela NVIDIA, está cada vez mais fácil implementar o uso de placas de vídeo em conjunto com CPUs convencionais nos mais variados algoritmos.

Nem todo algoritmo pode ser paralelizado, no entanto. Existem procedimentos e algoritmos que são inerentemente seriais, como o cálculo do n-ésimo número da sequência de *Fibonacci*, que requer que dois números prévios da sequência tenham sido calculados

para obtermos o atual – salvo, é claro, alguma descoberta teórica matemática do comportamento da sequência que nos permitisse uma nova maneira de calcular o n-ésimo elemento sem essa necessidade.

É importante entender também que nenhum algoritmo é paralelizável por completo. Sempre existem partes de algoritmos que devem ser executadas serialmente para funcionar corretamente. O ganho máximo da paralelização de um algoritmo qualquer é uma função da parcela do seu código que pode ser paralelizada, e é definida pela Lei de Amdahl (**??**): $\frac{1}{1-p}$, onde p é a razão entre código paralelizável e código não-paralelizável.

E é a paralelização de uma classe de algoritmos em especial que é o foco desta pesquisa: algoritmos de *clustering*, ou clusterização. Tais algoritmos, de forma sucinta, agrupam objetos de maneira que os objetos no mesmo grupo, ou *cluster*, sejam mais parecidos entre si, de acordo com alguma métrica, do que com objetos de outros grupos. A análise de clusters é essencial em diversas áreas da computação, como mineração de dados, aprendizado de máquina, compressão de dados, entre outras.

A hipótese principal é a de que algoritmos de clustering, em geral, são altamente paralelizáveis, e apresentam um ganho considerável de desempenho (menor tempo de execução) quando implementados para utilizar o poder de paralelismo vetorial de placas de vídeo NVIDIA, através da linguagem CUDA (**??**). Mais que isso, através de uma análise sistemática de estudos prévios e implementações de tais algoritmos em CUDA, visa-se generalizar o processo de paralelização destes. Isto é, identificar quais partes são necessariamente seriais, quais são paralelizáveis, e que sequência de passos gerais deve ser seguida para se conseguir paralelizar com sucesso e ganho significativo de desempenho um algoritmo de clustering qualquer.

O foco de pesquisa são quatro algoritmos de clustering especialmente notórios: *K-means* (**??**), *Clustering Hierárquico*, *DBSCAN* (**??**), ou *Clusterização Espacial Baseada em Densidade de Aplicações com Ruído*, e *Random Forests* – que mesmo não sendo exclusivamente um algoritmo de clusterização, pode ser utilizado justamente para tal. Implementações e estudos realizados sobre estes serão analisados, e implementações novas terão desempenho comparado com estas.

1.1 Objetivos

1.1.1 Objetivo Geral

Esse trabalho tem como objetivo principal testar a validez de sua hipótese (discutida mais à fundo na seção 1.2) de que algoritmos de clusterização são intrinsecamente paralelizáveis, e que o ganho de velocidade ao serem paralelizados é altamente significativo.

Além disso, deseja-se compilar aqui um vasto conhecimento de como paralelizar esses algoritmos em geral, analisando principalmente os quatro aqui estudados a fundo (k-means, clustering hierárquico, DBSCAN e Random Forests) e usando este aprendizado para criar um passo-a-passo genérico de como realizar tal modificação de código em um algoritmo de clusterização qualquer.

1.1.2 Objetivos Específicos

Para atingir o objetivo geral, será necessário completar diversos objetivos menores, ou *milestones*, antes, criando um caminho de pesquisa a ser seguido — não necessariamente na ordem apresentada. São estes:

- Pesquisar extensamente a bibliografia da área, realizando assim um levantamento do estado da arte de algoritmos paralelos de clusterização;
- Estudar implementações já realizadas dos quatro algoritmos aqui estudados, a fim de adquirir conhecimento de como a paralelização em CUDA deve ser realizada;
- Paralelizar um algoritmo de clustering "novo", isto é, nunca antes paralelizado e exibido em trabalho científico, a fim de solidificar o conhecimento e prática de programação em CUDA;
- Quantificar o ganho de desempenho das implementações paralelas, realizando diversos experimentos de *speedup*;
- Comparar código monolítico (sem paralelização) com código paralelo dos quatro algoritmos analisados, extraindo assim um conhecimento de como paralelizar um algoritmo de clusterização genérico;

1.2 Hipótese

A hipótese que esta pesquisa procura testar é a de que algoritmos de clusterização em geral são inerentemente vetoriais e, consequentemente, se beneficiariam significativamente de arquiteturas de processamento vetoriais, como uma unidade de processamento gráfico, ou GPU.

Um problema ser vetorial diz respeito ao escopo de itens de dados relevantes ao problema. Grande parte dos problemas da computação são escalares, o que significa que eles lidam com itens de dados unitários, como por exemplo *integers* ou *floats*, um de cada vez. Já um problema vetorial lida com itens de dados que são conjuntos unidimensionais, chamados vetores, que são formados por vários itens unitários de dados agrupados.

Um algoritmo que tente resolver um problema vetorial terá desempenho maior quando executado num processador vetorial, isto é, um processador que possui um conjunto de instruções capaz de manipular vetores. Apesar de um algoritmo de um problema vetorial ainda puder ser implementado e executado com sucesso num processador escalar, o desempenho será menor pois os dados vetoriais do problema terão que ter seus elementos processados um a um pelo processador, já que ele não trabalha com vetores propriamente ditos em seu conjunto de instruções.

Grande parte do ganho de desempenho supracitado vem do paralelismo proporcionado pelos processadores vetoriais, como GPUs, ao manipular conjuntos maiores de dados de uma só vez, e em vários núcleos simultaneamente, além de economizar traduções de endereço de memória e operações de obtenção (*fetch*) e decodificação (*decode*) de instruções, pois haverá um número muito menor de instruções e endereços de memória quando os dados estão agrupados em vetores, que podem ser manipulados e usados em operações como se fossem apenas um item de dados.

Este trabalho, então, visa demonstrar que algoritmos de clusterização, em geral, são intrinsecamente vetoriais. Isto é, qualquer algoritmo de clustering concebível será de natureza vetorial, pois estes analisam dados e tentam agrupá-los de a acordo com algum grau de semelhança entre eles, análise essa que pode ser feita com conjuntos dos itens de dados (vetores), ao invés de individualmente. Logo, qualquer algoritmo de clusterização teria uma parcela de seu código que seria paralelizável, e assim ganhariam desempenho significativo com uma execução numa GPU.

1.3 Justificativa

A pesquisa feita aqui pode ser de grande utilidade para a área de análise de clusters, e impulsionar a implementação de mais algoritmos paralelos de clusterização.

Com a compilação de conhecimento realizada aqui a intenção é facilitar pesquisas posteriores na área de paralelização de algoritmos de clustering e motivar novas implementações paralelas de outros algoritmos dessa classe com os experimentos de ganho de desempenho, ilustrando o quão importante é o uso de processadores vetoriais como GPUs para tornar o uso de alguns destes algoritmos prático.

Além disso, a apresentação nesse estudo de um procedimento genérico para paralelizar qualquer algoritmo de clustering será de extrema utilidade para qualquer desenvolvedor ou pesquisador que desejar implementar uma versão acelerada em GPU de um algoritmo de clusterização, mesmo este sendo totalmente novo. No mínimo servirá de ponto de partida para o entendimento e aprendizado de como realizar tal modificação no código do algoritmo, e renderá uma implementação que serve de base para estudos e otimizações, até se atingir uma implementação digna para uso prático.

2 Fundamentação Teórica

Para compreender a pesquisa científica aqui realizada, é necessário primeiro entender o que são algoritmos de clusterização, tanto de maneira geral quanto específica, explorando os fundamentos e funcionamento dos quatro algoritmos pesquisados. Veremos que a complexidade de tempo destes algoritmos tendem a ser inconvenientemente altas ($O(n \cdot \log n)$, $O(n^2)$ ou até $O(n^3)$ sendo complexidades comuns) e por isso qualquer ganho de velocidade significativo obtido valerá muito a pena.

Também é imprescindível explorar o funcionamento dos processadores vetoriais — sendo as *Unidades de Processamento Gráfico* (GPUs) o principal exemplo destes e sobre o qual essa pesquisa irá focar — e entender por quê usá-los para paralelizar algoritmos de clusterização proporcionará, em tese, um ganho de velocidade expressivo na execução destes, abrandando o peso de suas complexidades de tempo. Além de tudo isso, será apresentada brevemente a arquitetura que será utilizada para paralelizar os algoritmos estudados: a plataforma e modelo de programação CUDA, da NVIDIA, que permitirá extrair o poder de paralelização das placas de vídeo NVIDIA.

2.1 Clusterização

A clusterização, ou clustering, é a tarefa de agrupar um conjunto de elementos de modo que cada elemento de um grupo se "pareça" mais com outros elementos do grupo (cluster) que pertence do que com elementos dos grupos que não pertence — para algum significado bem definido de semelhança. É um processo muito comum e virtualmente necessário nas áreas de mineração de dados, análise estatística, análise de imagem, aprendizado de máquina, reconhecimento de padrões, e muitas outras.

O significado de um cluster não pode ser bem definido e vai depender do conjunto de dados a ser analisado e a forma que os resultados obtidos serão utilizados — de fato, esse é o principal motivo pelo qual tantos algoritmos diferentes de clustering existem (**??**). O comum em todas definições usadas é que um cluster é um conjunto de objetos de dados. Esses objetos são representados num espaço de dimensões iguais ao número de dados relevantes para cada objeto, e os algoritmos tentam criar grupos (clusters) nesse espaço que agrupem os objetos de uma maneira significativa, ou útil, para o estudo sendo feito e a definição de cluster sendo utilizada.

Diversos modelos de cluster podem ser usados para definir o que é um cluster: **modelos de centroide**, onde cada cluster possui um centro e os objetos pertencerão aos clusters com centros mais próximos deles, dada uma definição de distância no espaço de

dados; **modelos de densidade**, que definem clusters como regiões densas e conexas no espaço de dados, contrastando com regiões menos densas que separam os clusters; **modelos de conectividade**, que constroem clusters a partir de conexões de objetos por distância; **modelos de distribuição**, que utilizam de distribuições estatísticas, como a distribuição normal ou exponencial, para modelar agrupamentos dos objetos; entre dezenas de outros modelos. Entender o modelo de cluster utilizado é essencial para compreender um algoritmo de clusterização e as diferenças entre a multitude destes.

O resultado, ou saída, de um algoritmo de clusterização é comumente um rotulamento dos objetos de dados passados na entrada, o que indicará a divisão em clusters feita por ele. Classificações podem ser feitas quanto à natureza da clusterização obtida pelos algoritmos: ***hard clustering***, onde cada objeto pertence ou não a um cluster; ***fuzzy clustering***, onde cada objeto pertence uma certa porcentagem a cada cluster, o que pode significar, por exemplo, a chance do objeto pertencer àquele cluster (**??**). E subclassificações mais granulares ainda podem ser definidas, como: **clusterização de particionamento estrito**, onde cada objeto pertence a exatamente um cluster; **clusterização de particionamento estrito com *outliers***, onde objetos pertencem a exatamente um cluster, ou nenhum cluster, assim sendo considerados *outliers*.

3 Metodologia de Desenvolvimento e Pesquisa

A pesquisa realizada neste trabalho consiste de estudos e análises de trabalhos prévios, desenvolvimento de uma versão paralelizada de um algoritmo de clustering e experimentos sobre essa implementação. Pode-se dividir tal metodologia em um conjunto de etapas.

A primeira etapa consiste de uma extensa pesquisa bibliográfica. O intuito é levantar o estado da arte na área de algoritmos de clusterização acelerados em GPU usando a linguagem e livraria CUDA da NVIDIA. Entender quais algoritmos já foram implementados com sucesso em CUDA, e como foi feita tal implementação, além dos resultados (ganhos em desempenho) das mesmas. Essa etapa agregará conhecimento sobre como utilizar CUDA para acelerar algoritmos de clustering, além de dar uma ideia do tipo de ganho de desempenho esperado de uma paralelização média desse tipo de algoritmo.

A segunda etapa consiste da implementação de uma versão paralela "inédita" de algum algum algoritmo de clusterização, ou seja, paralelizar um algoritmo nunca antes paralelizado e exibido em alguma pesquisa. Usando o aprendizado adquirido na primeira etapa de pesquisa, um algoritmo será paralelizado em CUDA, e seus resultados comparados com os resultados da versão serial (rodando somente numa CPU) para garantir corretude. Os ganhos de velocidade da versão acelerada em CUDA será então comparada também com os ganhos obtidos nos trabalhos analisados na primeira etapa. Isso servirá como uma medição da efetividade da implementação feita.

A terceira etapa consiste da busca de um procedimento geral para paralelizar um algoritmo de clustering genérico. Ou seja, encontrar um passo-a-passo de modificações ao código de um algoritmo serial que, ao fim, o transforme numa versão acelerada usando CUDA desse mesmo algoritmo, ainda mantendo sua corretude e proporcionando algum ganho de desempenho.

A quarta etapa, por fim, se trata de diversos experimentos de ganho de velocidade, ou *speedup*, do algoritmo que teve aqui sua versão acelerada em GPU implementada e apresentada. Esses experimentos irão dar uma ideia do quão significativo foi o ganho de desempenho ao paralelizar o algoritmo usando CUDA, se há um teto ou chão para tais ganhos, como o *speedup* aumenta ou diminui com o aumento do dataset a ser analisado, assim como realizar uma análise de outros parâmetros importantes que não sejam velocidade, como uso de memória — afinal, as VRAMs das placas de vídeo são comumente mais limitadas em tamanho que as RAMs utilizadas pelas CPUs.

4 Cronograma

A tabela abaixo representa o cronograma de pesquisa e desenvolvimento deste trabalho. Cada célula da mesma representa uma semana de tempo (sete dias corridos), considerando cada mês como tendo exatamente quatro semanas. O período coberto vai de dezembro de 2019 à julho de 2020, data estimada de término da conclusão desta monografia.

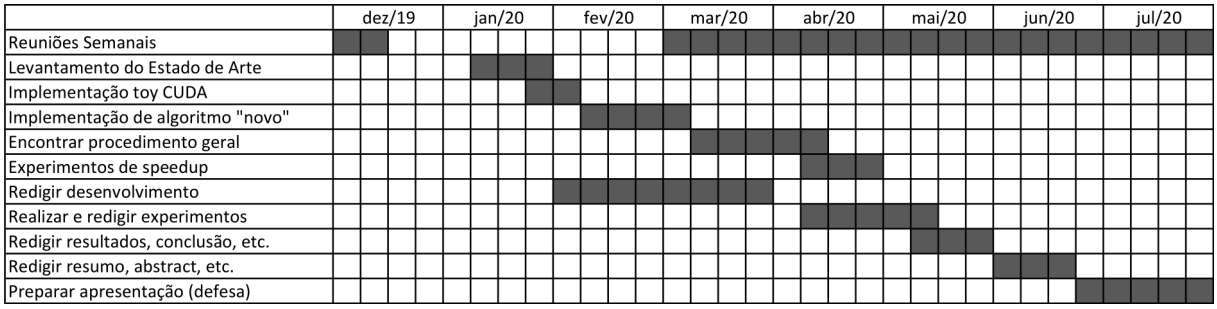

Figura 1 – Cronograma de pesquisa e desenvolvimento