UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Paralelização do algoritmo para Agrupamento de Dados MRDCA-RWL usando GPU e CUDA

Trabalho de Conclusão de Curso

Diego Michael Almeida Santana

São Cristóvão – Sergipe

2019

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Diego Michael Almeida Santana

Paralelização do algoritmo para Agrupamento de Dados MRDCA-RWL usando GPU e CUDA

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador(a): Prof. Dr. Renê Pereira de Gusmão

São Cristóvão – Sergipe

2019

Eu dedico esta monografia a todos os cientistas e entusiastas que trouxeram a Computação até aqui.

Agradecimentos

Gostaria de agradecer a Deus pela vida, a Universidade Federal de Sergipe pela oportunidade, ao Departamento de Computação pela estrutura, aos professores pela dedicação e aprendizado, em especial ao Professor Renê pela paciência e orientação, inclusive por ter disponibilizado um código-fonte fundamental. Aos colegas pela troca de experiências e momentos descontraídos, aos amigos pela incentivo, a minha querida família pelo apoio e a minha amada esposa Daniela por seu cuidado ao longo do curso e por me encorajar a iniciá-lo e a concluí-lo. Obrigado a todos, por tudo!

"O modo como você reúne, administra e usa a informação determina se vencerá ou perderá."

(Bill Gates)

Resumo

O aumento crescente do volume de dados disponíveis na Internet (*Big Data*) cria uma necessidade urgente de gerenciamento. Nesse sentido, gerar conhecimento a partir desses dados se torna um desafio computacional ainda maior. Para mitigar, são aplicadas técnicas no campo da Mineração de Dados como os métodos de Agrupamento de Dados (*Data Clustering*). Contudo, o desempenho dessas técnicas frente ao *Big Data* não se mostra satisfatório em relação ao tempo de execução, uma vez que os algoritmos tradicionais são sequenciais ou síncronos, então esta pesquisa procura explorar novas formas de acelerar os algoritmos de Agrupamento de Dados como a implementação de recursos da Computação Paralela. Assim sendo, a tecnologia CUDA foi selecionada após uma revisão das principais técnicas de paralelização e quais resultados foram alcançados. Com efeito de aumentar a escalabilidade para grandes conjuntos de dados e realizar uma comparação de desempenho, então uma versão paralela do Algoritmo Dinâmico de Agrupamento Rígido com Peso de Relevância para cada Matriz de Dissimilaridade Estimada Localmente (MRDCA-RWL) foi implementada. Dessa forma, os experimentos utilizaram dez conjuntos de dados conhecidos e disponíveis no repositório da UC Irvine. Logo, a versão paralela proposta por este trabalho obteve uma aceleração média de 16,7 vezes no tempo de execução, o que representa um salto significativo no desempenho do algoritmo.

Palavras-chave: Agrupamento de Dados, Computação Paralela, CUDA, Big Data.

Abstract

The increasing volume of data available on the Internet (Big Data) creates an urgent need for management. In this sense, generating knowledge from these data becomes an even greater computational challenge. To mitigate, techniques are applied in the Data Mining field as the Data Clustering methods. However, the performance of these techniques against Big Data is not satisfactory in relation to the execution time, since the traditional algorithms are sequential or synchronous, so this research seeks to explore new ways to accelerate the algorithms of Data Clustering as the implementation of Parallel Computing resources. Therefore, CUDA technology was selected after a review of the main parallelization techniques and what results were achieved. With the effect of increasing scalability for large data sets and performing a performance comparison, then a parallel version of the Dynamic Hard Clustering Algorithm with Relevance Weight for each Dissimilarity Matrix Estimated Locally (MRDCA-RWL) has been implemented. In this way, the experiments used ten known data sets and available in the UC Irvine repository. Therefore, the parallel version proposed by this work obtained an average acceleration of 16.7 times in the execution time, which represents a significant leap in the performance of the algorithm.

Keywords: Data Clustering, Parallel Computing, CUDA, Big Data.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Lista de algoritmos

Lista de abreviaturas e siglas

CPU	*Central Processing Unit*
CUDA	*Compute Unified Device Architecture*
DDR	*Double Data Rate*
EEG	Eletroencefalografia
ETL	*Extract Transform Load*
GHz	*Giga Hertz*
GPGPU	*General Purpose Graphics Processing Unit*
GPU	*Graphics Processing Units*
IDE	*Integrated Development Environment*
KDD	*Knowledge Discovery from Data*
MHz	*Mega Hertz*
MPI	*Message Passing Interface*
MRDCA-RWL	*Dynamic Hard Clustering Algorithm with Relevance Weight for each Dissimilarity Matrix Estimated Locally*
RAM	*Random Access Memory*
SDK	*Software Development Kit*
SIMT	*Single-Instruction, Multiple-Thread*
SM	*Streaming Multiprocessor*
SO	Sistema Operacional
UCI	*University of California, Irvine*

Sumário

1 Introdução

Cientistas e pesquisadores acreditam que um dos maiores desafios da Computação no final da década de 2010 é encontrar métodos de como lidar com essa enorme quantidade de dados gerados diariamente. Esse aumento exponencial é mais conhecido como *Big Data*. A exemplo, o YouTube tem mais de 1,9 bilhão de usuários que assistem mais de 1 bilhão de horas de vídeo diariamente; o Google Imagens indexou 10 bilhões de imagens em 2010; E com milhões de usuários ativos, redes sociais como Instagram e Twitter produzem centenas de gigabytes de conteúdo por minuto (JAIN, 2010), (SHIRKHORSHIDI et al., 2014), (GAO et al., 2018), (YOUTUBE, 2019), (GOOGLE, 2010), (INSTAGRAM, 2017).

Esse crescimento explosivo em dados armazenados gerou uma necessidade urgente de novas técnicas e ferramentas automatizadas que podem ajudar de forma inteligente a transformar grandes quantidades de dados em informação e conhecimento úteis (HAN; KAMBER; PEI, 2011). Nesse sentido, para transformar todos esses dados em conhecimento, a mineração de dados (do inglês *data mining*) se torna essencial. Assim Han, Kamber e Pei (2011) definem:

> Data mining, also popularly referred to as knowledge discovery from data (KDD), is the automated or convenient extraction of patterns representing knowledge implicitly stored or captured in large databases, data warehouses, the Web, other massive information repositories, or data streams. [1]

Entre as técnicas para o processamento de dados estão o processamento de linguagem natural, reconhecimento de padrões e aprendizado de máquina. Os algoritmos de aprendizado de máquina aprendem com dados evoluindo a partir do estudo do reconhecimento de padrões e da teoria da aprendizagem computacional. Essas técnicas também fazem parte do campo da Inteligência Artificial (CUOMO et al., 2017).

[1] A mineração de dados, também chamada de descoberta de conhecimento a partir de dados (*knowledge discovery from data - KDD*), é a extração automatizada ou conveniente de padrões representando conhecimento implicitamente armazenado ou capturado em grandes bancos de dados, armazéns de dados, Web, outros repositórios de informações ou fluxos de dados. (tradução nossa).

Dentre as mais conhecidas, temos o Agrupamento de Dados (do inglês *Data Clustering*). Ela é definida como um método, no qual os dados são divididos em grupos de forma que objetos em cada grupo compartilhem mais similaridade do que com outros objetos em outros grupos. Para isso, são realizadas comparações quantitativas de características múltiplas (JAIN, 2010).

Os referidos métodos de agrupamento podem melhorar consideravelmente o desempenho da recuperação de informações da Internet (FAKHI et al., 2017). Porém, o principal problema para processar e minerar informações valiosas nesse contexto de *Big Data* é a questão da escalabilidade, uma vez que as técnicas tradicionais de agrupamento não conseguem lidar com essa enorme quantidade de dados devido à sua alta complexidade computacional, como exemplo o algoritmo K-means que é NP-difícil [2], mesmo quando o número de grupos é igual a dois ($K = 2$) (SHIRKHORSHIDI et al., 2014).

Essa complexidade de tempo pode ser reduzida com a Computação Paralela. E como a execução paralela em Unidades de Processamento Gráfico (GPUs), (do inglês *Graphics Processing Units*), se mostrou uma alternativa promissora em várias pesquisas com algoritmos de Agrupamento de Dados. Há uma motivação para explorar a implementação paralela dos referidos algoritmos a fim de melhorar o tempo de execução para grandes conjuntos de dados (*datasets*) (HONG-TAO et al., 2009), (WU; HONG, 2011), (BHIMANI; LEESER; MI, 2015).

Quanto aos dados, há duas representações: dados de características (*feature data*) e dados relacionais (*relational data*). Nesse sentido, dados de características são mais utilizados, formados por um conjunto de vetores que descrevem os objetos, já dados relacionais são menos comuns e consistem nas relações em pares (similaridade ou não) geralmente armazenada em uma matriz (CARVALHO; LECHEVALLIER; MELO, 2012).

Em Carvalho, Lechevallier e Melo (2012) propuseram o Algoritmo Dinâmico de Agrupamento Rígido com Peso de Relevância para cada Matriz de Dissimilaridade Estimada Localmente (MRDCA-RWL), (do inglês *Dynamic Hard Clustering Algorithm with Relevance Weight for each Dissimilarity Matrix Estimated Locally*). Ele foi publicado em 2012 e utiliza a representação relacional. Nesse contexto, a paralelização de algoritmos com representação relacional foi pouco explorada e os testes com o MRDCA-RWL foram em conjuntos de dados relativamente pequenos, logo este trabalho tem como objetivo implementar uma versão paralela e comparar o desempenho em relação a versão sequencial bem como propor aplicações escaláveis para conjuntos de dados maiores.

1.1 Justificativa

A implementação de métodos de agrupamento para *Big Data* requer plataformas de computação escaláveis capazes de armazenar e processar dados massivos e de alta dimensionalidade (SILVA et al., 2017). Nessa perspectiva, a Computação Paralela se mostra eficiente ao

[2] Classe de complexidade computacional.

diminuir o tempo de execução de algoritmos síncronos ou sequenciais. E conforme observado na revisão bibliográfica, muitos trabalhos obtiveram resultados promissores aplicando técnicas de paralelização.

Como os resultados empíricos do algoritmo MRDCA-RWL mostraram-se superiores em relação a outros algoritmos semelhantes (CARVALHO; LECHEVALLIER; MELO, 2012). Desse modo, adaptar e implementar uma versão paralela dele, poderá potencializar seu desempenho de tempo computacional principalmente para grandes conjuntos de dados.

Diante desse cenário, com a motivação de investigar formas de paralelização, implementar e comparar o desempenho com a versão sequencial, então o algoritmo MRDCA-RWL foi selecionado como objeto desse estudo.

1.2 Objetivos

1.2.1 Geral

Desenvolver uma versão escalável do algoritmo MRDCA-RWL através da paralelização.

1.2.2 Específicos

1. Investigar formas de paralelizar um algoritmo;
2. Selecionar um algoritmo de Agrupamento de Dados;
3. Testar uma proposta de paralelização;
4. Implementar uma versão paralela do algoritmo selecionado;
5. Comparar o desempenho com a versão sequencial desse algoritmo;
6. Coletar os dados obtidos nos experimentos;
7. Apresentar os resultados com uma abordagem quantitativa.

1.3 Estrutura do Documento

Para facilitar a navegação e melhor entendimento, este documento está estruturado em capítulos e seções, que são:

- Capítulo 1 - Introdução: São apresentados os principais conceitos que serão detalhados no decorrer deste trabalho, bem como a justificativa, objetivos e metodologia;
- Capítulo 2 - Trabalhos Relacionados: Contém a base da pesquisa realizada com algoritmos de Agrupamento de Dados e as respectivas técnicas de Computação Paralela utilizadas;

- Capítulo 3 - Tecnologia CUDA: São discutidas as características da tecnologia CUDA;
- Capítulo 4 - Desenvolvimento: É descrito o algoritmo que será objeto da pesquisa, bem como a implementação paralela;
- Capítulo 5 - Experimentos Empíricos: Aqui são apresentados os conjuntos de dados usados e discutidos os resultados dos experimentos;
- Capítulo 6 - Conclusão: As considerações finais deste trabalho são apresentadas.

2

Trabalhos Relacionados

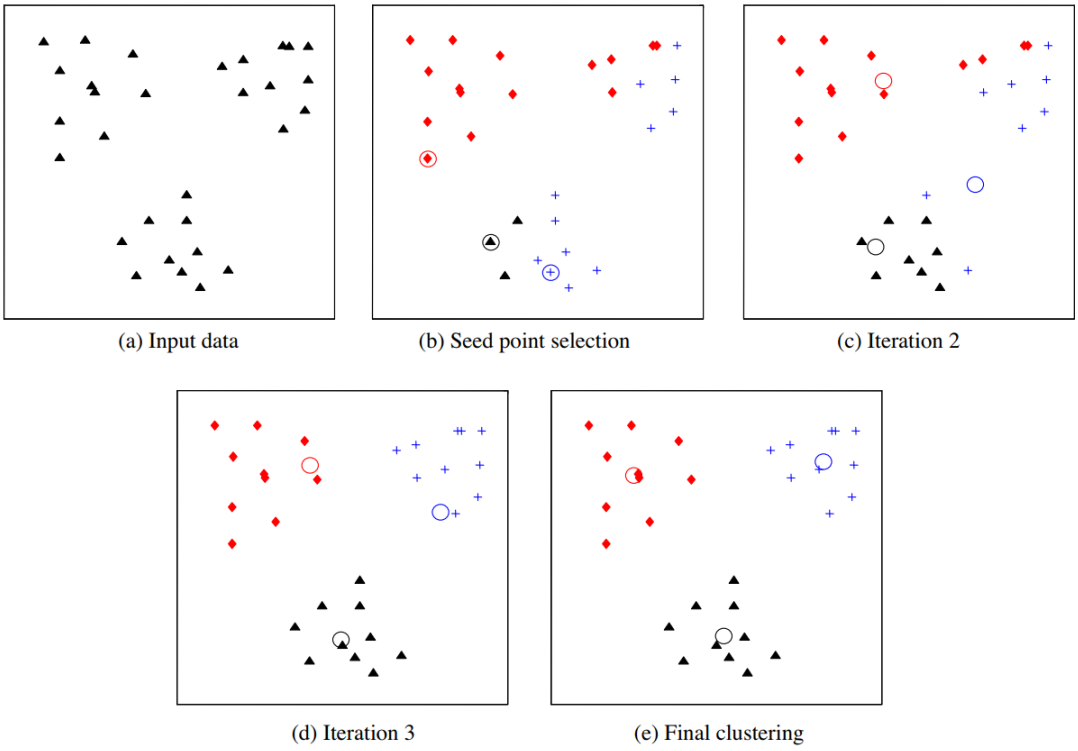
O Agrupamento de Dados é caracterizado como um método de aprendizado não supervisionado que particiona um conjunto de objetos de dados em grupos (do inglês *clusters*) (JAIN; DUBES, 1988). Esses métodos podem ser classificados como: hierárquicos, particionais, baseados em densidade, baseados em grade, baseados em modelos, baseados em redes neurais, baseados em grafos, entre outros. Em especial, os métodos particionais podem ser rígidos (*hard*) e nebulosos (*fuzzy*) (JAIN, 2010), (HAN; KAMBER; PEI, 2011). Nesse contexto, o algoritmo particional mais popular é o K-means. Sendo utilizado em: Hong-tao et al. (2009), Wu e Hong (2011), Yang et al. (2014), Bhimani, Leeser e Mi (2015), Baydoun, Ghaziri e Al-Husseini (2018), entre outros trabalhos. E segundo Jain (2010) essa grande utilização se deve por alguns motivos:

> K-means has a rich and diverse history as it was independently discovered in different scientific fields by Steinhaus (1956), Lloyd (proposed in 1957, published in 1982), Ball and Hall (1965), and MacQueen (1967). Even though K-means was first proposed over 50 years ago, it is still one of the most widely used algorithms for clustering. Ease of implementation, simplicity, efficiency, and empirical success are the main reasons for its popularity. [1]

A Figura 1 apresenta uma ilustração do algoritmo K-means em um conjunto de dados em 2 dimensões com três grupos. Onde: (a) são os dados de entrada bidimensionais com três grupos; (b) indica os três pontos de sementes selecionados como centros de grupo e a atribuição inicial dos pontos de dados por grupo; (c) e (d) mostram as iterações intermediárias atualizando os rótulos de grupos e seus centros; (e) o agrupamento final obtido pelo algoritmo K-means na convergência.

[1] K-means tem uma história rica e diversificada, como foi descoberto independentemente em diferentes campos científicos por Steinhaus (1956), Lloyd (proposto em 1957, publicado em 1982), Ball and Hall (1965) e MacQueen (1967). Embora K-means tenha sido proposto pela primeira vez há mais de 50 anos, ainda é um dos algoritmos mais amplamente usados para agrupamento. Facilidade de implementação, simplicidade, eficiência e sucesso empírico são os principais motivos de sua popularidade. (tradução nossa).

Figura 1 – Ilustração do algoritmo K-means.



Fonte: extraído de Jain (2010).

Por ser tão comum, muitos pesquisadores adaptaram de várias formas o algoritmo K-means para a Computação Paralela. A esse respeito, são muitas as aplicações que utilizam a paralelização, pois existem vários algoritmos paralelizáveis e diversas plataformas paralelas (BAYDOUN; DAWI; GHAZIRI, 2016).

Dessa forma, as arquiteturas paralelas incluem *Field Programmable Gate Arrays* (FPGAs), Unidades de Processamento Centrais Multinúcleos (CPUs), Redes ou Clusters, Supercomputadores e muitos outros. Entre as principais APIs, destacam-se: MPI (Message Passing Interface) e OpenMP (Open Multi Processor) (BAYDOUN; GHAZIRI; AL-HUSSEINI, 2018).

Há também o interesse crescente na implementação com as Unidades de Processamento Gráfico (GPUs), que segundo Hong-tao et al. (2009), se deve ao aumento do desempenho e da melhoria na programação, conhecido como computação de propósito geral (GPGPU). Entre as pricicipais ferramentas estão o Open Computing Language (OpenCL) e Compute Unified Device Architecture (CUDA).

Para um melhor entendimento, as seções a seguir estão separadas por tecnologias que foram utilizadas para implementar algoritmos paralelos. Desse modo, são apresentados conceitos básicos e em seguida os trabalhos relacionados.

2.1 Trabalhos com CUDA

CUDA é uma plataforma de computação paralela e um modelo de programação que permite aumentos drásticos no desempenho de computação, aproveitando a potência da unidade de processamento gráfico (GPU) (NVIDIA, 2018).

Ela foi a mais utilizada nos artigos pesquisados. Em Hong-tao et al. (2009) onde foi realizada a paralelização do K-means, o cálculo complexo e o tempo-custo pôde ser reduzido substancialmente usando a GPU através de CUDA.

Quanto ao desempenho, Wu e Hong (2011) propôs uma forma paralela hibrida do K-means e obteve uma aceleração em CUDA de 75 vezes, em relação ao algoritmo padrão de 21 vezes e ainda salienta que outras etapas do algoritmo podem ser aceleradas com uso de GPUs.

No agrupamento de imagens Bhimani, Leeser e Mi (2015) comparou OpenMP com CUDA e alcançou uma aceleração de cerca 35 vezes na execução. Resalta que o OpenMP funciona melhor em imagens menores, enquanto o CUDA supera para imagens maiores. Sugere trabalhos futuros combinando as plataformas e usando múltiplas GPUs. Já Karbhari e Alawneh (2018) que também pesquisaram a segmentação de imagens, são ilustradas algumas das técnicas de otimização com memória compartilhada e memória constante, e os resultados obtidos mostraram uma aceleração de 9 a 57 vezes em relação a implementação sequencial.

No entanto, também houve resultados inesperados em Sirotković, Dujmić e Papić (2012) onde foi paralelizado o algoritmo K-means para aplicações na segmentação de imagens. Os autores explicam que a largura de banda da memória é um fator limitante, sendo a execução baseada GPU inviável e propõe trabalhos futuros em CPU multi-cores. Já Zhong et al. (2016) não obteve bons resultados para grandes *datasets* e afirma que vai continuar tentando melhorar o desempenho do algoritmo.

De modo geral, a implementação em CUDA produziu os melhores resultados, apesar do tempo requerido para alocação de memória e transferência de dados. Os autores Baydoun, Ghaziri e Al-Husseini (2018) realizaram uma ampla comparação entre o CUDA e o OpenMP com vários *datasets* conhecidos.

2.2 Trabalhos com OpenCL

O OpenCL (Open Computing Language) é o padrão aberto e isento de royalties para programação paralela de plataforma cruzada de diversos processadores encontrados em computadores pessoais, servidores, dispositivos móveis e plataformas incorporadas (KHRONOS, 2019).

É um *framework* para plataformas heterogêneas compostas de CPUs, GPUs, DSPs, FPGAs etc. As GPUs NVIDIA também suportam o OpenCL (KARBHARI; ALAWNEH, 2018).

Ele é similar ao CUDA em vários aspectos, mas uma diferença foi destacada em Torti et al. (2018) que é a ausência de paralelismo dinâmico, portanto não é possível ativar um *kernel* a partir de outro.

Em Zhu et al. (2015) foi implementada a paralelização com OpenCL, esse trabalho abordou a detecção de imagens de radares com o K-means, nele foi relatado que o incremento de velocidade aumenta com o acréscimo do número dos núcleos e a escala do conjunto de dados.

Como o OpenCL tem como objetivo principal a portabilidade entre diferentes dispositivos, ele perde desempenho na execução. Isso foi observado em Torti et al. (2018) em comparação com o CUDA. As versões em CUDA empregaram rotinas altamente otimizadas, que aproveitam todos os recursos de hardware da GPU. Além disso, foram compiladas usando opções de compilação para produzir um código executável que explora completamente a arquitetura de destino específica, nesse caso apenas NVIDIA.

2.3 Trabalhos com OpenMP

OpenMP é uma especificação para um conjunto de diretivas de compilador, rotinas de biblioteca e variáveis de ambiente que podem ser usadas para especificar paralelismo de alto nível em programas Fortran e C/C++ (OPENMP, 2019).

Nos experimentos com OpenMP, Yang et al. (2014) relataram uma boa escalabilidade do algoritmo quando o número de *threads* aumenta. Eles realizaram um comparativo do OpenMP com outras tecnologias como FPGA e MPI. E em relação ao aspecto de dificuldade de programação o OpenMP foi classificado como o mais fácil por causa da paralelização implícita do compilador.

No estudo de Bhimani, Leeser e Mi (2015) em que o K-means foi paralelizado, houve algumas considerações sobre o compartilhamento de memória "The shared memory platform of OpenMP is better suited for K-Means than the distributed platform of MPI as the communication overhead between physical nodes in a distributed setting is expensive."[2]

Em Jaroš et al. (2017) foi estudado a segmentação de imagens, com quantidades que variavam de 1 a 8191 imagens, na qual foi comparada as versões paralela e sequencial do algoritmo, e em alguns casos a diferença foi tão alta que o tempo de execução da versão sequencial ficou estimada em 11 dias, enquanto a paralela cerca de 2 horas.

[2] A plataforma de memória compartilhada do OpenMP é mais adequada para K-Means do que a plataforma distribuída de MPI, já que a sobrecarga de comunicação entre nós físicos em uma configuração distribuída é cara. (tradução nossa).

2.4 Trabalhos com MPI

A Interface de Passagem de Mensagens, ou MPI (Message Passing Interface) é um projeto de código aberto desenvolvido e mantido por um consórcio de parceiros acadêmicos, e do setor de pesquisa (MPI, 2019).

A MPI fornece um modelo de programação para a passagem de mensagem em arquiteturas paralelas. O modelo MPI é muito útil na criação de aplicativos paralelos eficientes e escalonáveis (SARDAR; ANSARI, 2018).

Yang et al. (2014) desenvolveram versões do K-means para 4 arquiteturas e definiram a escalabilidade do MPI como excelente, quando o número de processadores aumenta. Ainda salientaram que o MPI tem uma biblioteca madura que pode ser usada para fazer reduções e muitas outras computações paralelas básicas.

Na comparação realizada por Bhimani, Leeser e Mi (2015), em que foi estudado o agrupamento de imagens, observou-se que a distribuição de processos entre menos nós físicos é mais vantajosa para imagens pequenas. E no caso de imagens maiores, a distribuição de processos entre um número maior de nós é melhor.

2.5 Análise dos Trabalhos

Neste capítulo foi apresentada uma visão geral sobre Agrupamento de Dados, uma revisão das principais tecnologias de paralelização, seus resultados recentes e ainda as abordagens para implementação em algoritmos de agrupamento como o K-means. Dessa forma, a grande maioria obteve uma aceleração significativa no tempo de execução, destacando-se nesse contexto a tecnologia CUDA.

3

Tecnologia CUDA

A tecnologia CUDA foi escolhida como proposta de paralelização deste trabalho, pois observou-se na revisão bibliográfica resultados promissores além de ser a mais utilizada pelos autores. Percebe-se também que ela está em constante evolução, possui uma boa documentação e disponibilidade para a comunidade através das GPUs fabricadas pela NVIDIA (NVIDIA, 2019b).

As seções a seguir detalham as principais características da tecnologia CUDA.

3.1 Sobre a Tecnologia CUDA

Lançada em novembro de 2006 pela NVIDIA, CUDA é uma das mais recentes arquiteturas de software e hardware para computação paralela de propósito geral em GPUs. Com essa tecnologia a GPU é vista como um conjunto de Multiprocessadores de *Streaming* (SMs) que executam um alto número de *threads* em paralelo, ou seja, é uma plataforma e um modelo de programação que aproveita o mecanismo de computação paralela nas GPUs NVIDIA, abstraindo-as como dispositivo de processamento de dados paralelo (NVIDIA, 2019b).

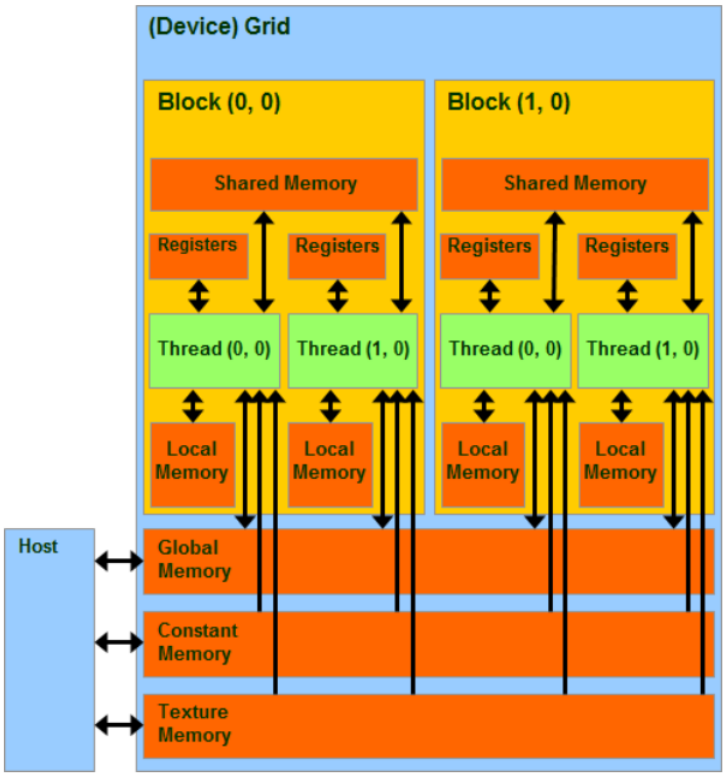
A Figura 2 apresenta uma visão geral de um dispositivo CUDA, a relação do *host* e o *device*, a organização da grade (*grid*), *threads* e blocos, bem como o acesso aos diferentes tipos de memória.

3.2 Organização das *Threads*

Uma definição de *thread* é feita por Stallings (2018):

> In essence, a single process, which is assigned certain resources, can be broken up into multiple, concurrent threads that execute cooperatively to perform the

Figura 2 – Visão geral de um dispositivo CUDA.



Fonte: extraído de Baydoun, Dawi e Ghaziri (2016).

> work of the process. This introduces a new level of parallel activity to be managed by the hardware and software. [1]

As *threads* em CUDA são organizadas por grupos de *threads* chamados de blocos. Além disso, os blocos são agrupados em uma grade, conforme a Figura 3. A ordem de execução dos blocos dentro da grade é indefinida (CANO, 2018), (NVIDIA, 2019b).

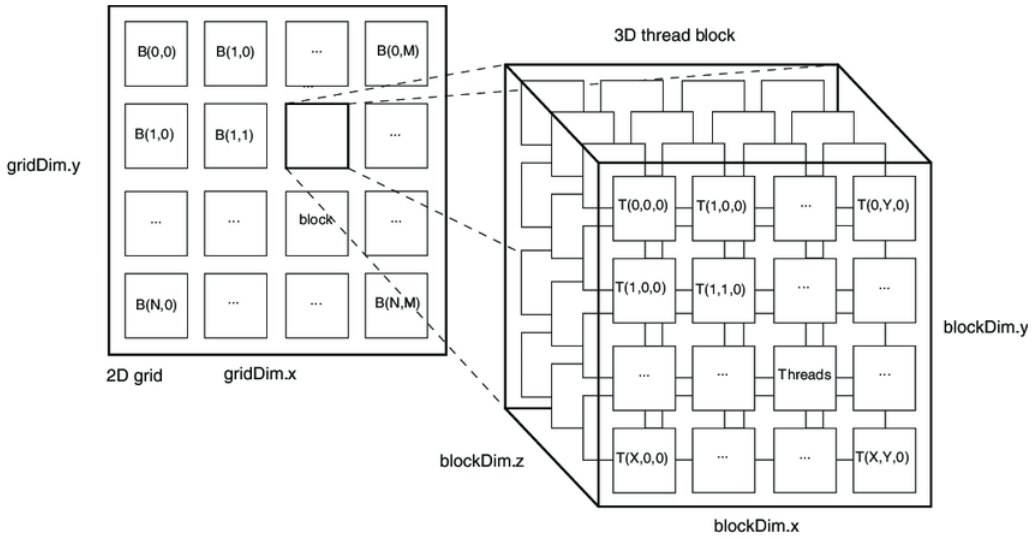
Para gerenciar uma quantidade tão grande de *threads*, CUDA emprega uma arquitetura chamada SIMT (*Single-Instruction, Multiple-Thread*), que executa o escalonamento em grupos de 32 *threads* para cada multiprocessador. (NVIDIA, 2019b).

3.3 Modelo de Memória

As *threads* CUDA podem acessar dados de vários espaços de memória durante sua execução. Cada *thread* possui memória local privada. Nesse sentido, cada bloco de *thread* tem memória compartilhada visível para todos os *threads* do bloco e com o mesmo tempo de vida

[1] Em essência, um único processo, que é atribuído a determinados recursos, pode ser dividido em várias *threads* simultâneos que executam cooperativamente para realizar o trabalho do processo. Isso introduz um novo nível de atividade paralela a ser gerenciado pelo hardware e software. (tradução nossa).

Figura 3 – Modelo de programação multidimensional de *threads* e blocos em CUDA.

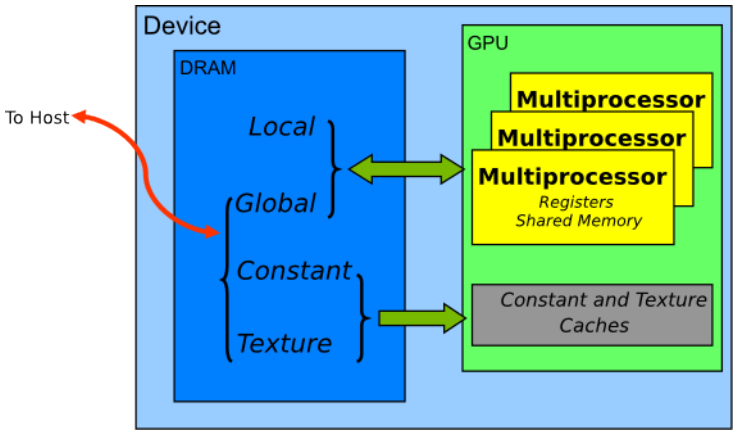


Fonte: extraído de Cano (2018).

do bloco. Todas as *threads* têm acesso à mesma memória global. Há também dois espaços de memória adicionais de somente leitura, acessíveis por todas as *threads*: os espaços de memória de constante e textura (NVIDIA, 2019b).

A Figura 4 mostra mais detalhes sobre os espaços de memória.

Figura 4 – Espaços de memória em um dispositivo CUDA.



Fonte: extraído de NVIDIA (2019a).

3.4 Modelo de Programação

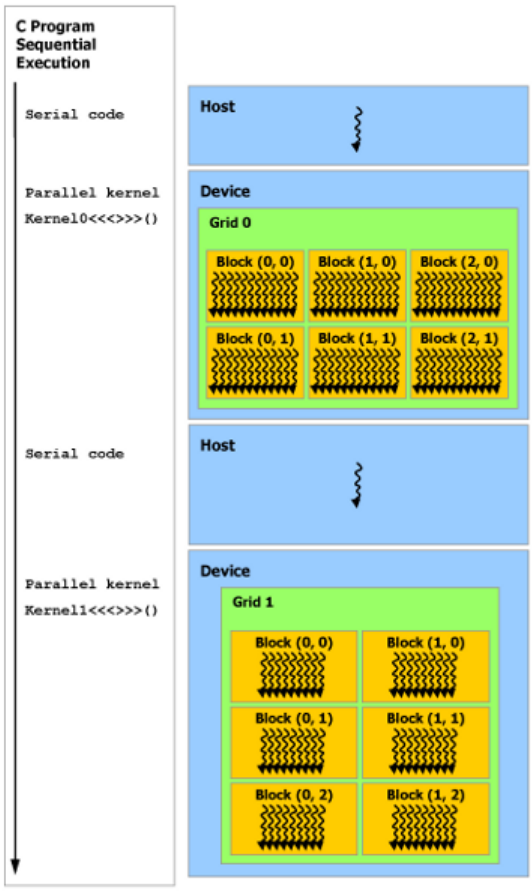
Um aplicativo típico em CUDA consiste em duas partes. A primeira parte é executada na CPU também chamada de *host*, em que é responsável pela transferência de dados de entrada e

saída entre a GPU e a CPU. Além disso, é responsável também por configurar a execução do código do dispositivo especificando a quantidade e a organização lógica das *threads* a serem geradas na GPU. A segunda parte é completamente executada na GPU e é conhecida como *kernel* paralelo (SIROTKOVIć; DUJMIć; PAPIć, 2012).

Desse modo, cada bloco de *threads* pode ser programado em qualquer um dos multiprocessadores disponíveis dentro de uma GPU, em qualquer ordem concorrente ou sequencialmente, de forma que um programa CUDA compilado possa ser executado em qualquer número de multiprocessadores (NVIDIA, 2019b).

Essa organização constitui um modelo de programação heterogêneo, conforme é mostrado na Figura 5 em que é possível distinguir as partes que são executadas sequencialmente na CPU (em branco) e paralelamente na GPU (em azul).

Figura 5 – Programação heterogênea CUDA.

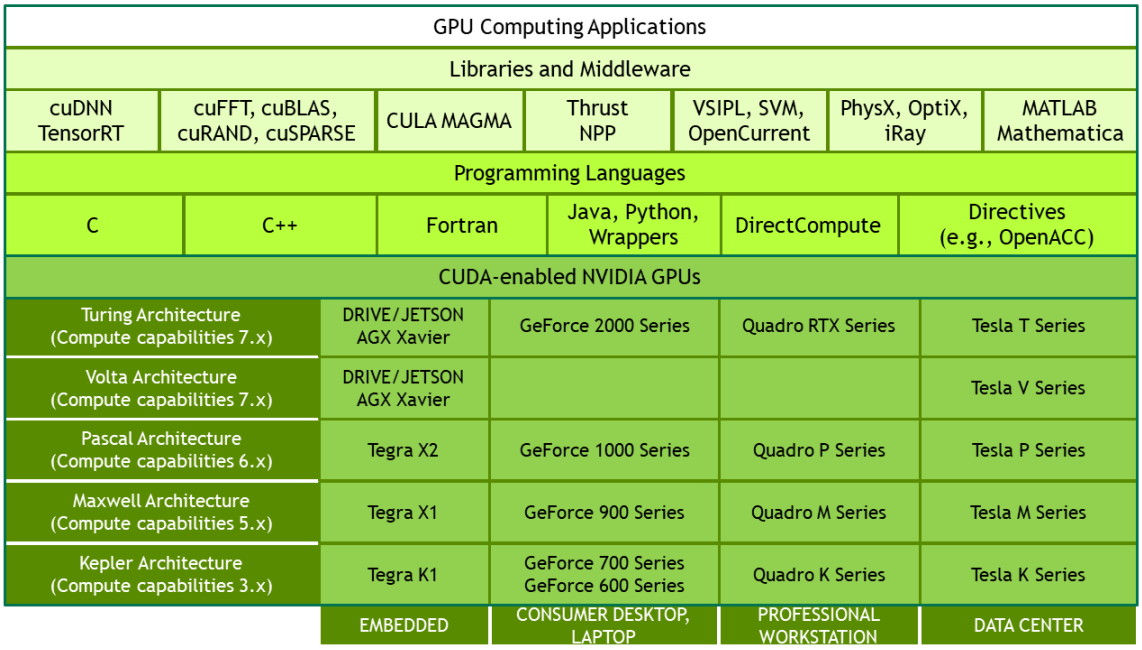


Fonte: extraído de NVIDIA (2019b).

3.5 SDK

A NVIDIA disponibiliza gratuitamente um SDK [2], também conhecido como CUDA Toolkit, ele inclui ferramentas de depuração e otimização, um compilador C/C++ e muitos exemplos de implementação. Dessa forma, conforme ilustrado na Figura 6 ainda possui várias bibliotecas especializadas para diversas áreas, tais como: álgebra linear, geração de números aleatórios, Transformada Rápida de Fourier, inferência de aprendizagem profunda de alto desempenho (*deep learning*) e entre outras. Tais pacotes podem ser programados com diversas linguagens, entre elas: C, C++, Fortran, Java e Python (NVIDIA, 2018).

Figura 6 – Aplicações de CUDA e compatibilidade.



Fonte: extraído de NVIDIA (2019b).

[2] https://developer.nvidia.com/cuda-toolkit

4

Desenvolvimento

A partir da escolha de CUDA como tecnologia de paralelização, o algoritmo de agrupamento de dados MRDCA-RWL foi selecionado para a implementação paralela. Além disso, um comparativo de desempenho entre a versão sequencial e paralela será realizado para conjuntos de dados com tamanhos variados. A motivação para escolha desse algoritmo foi apresentada na Seção 1.1.

Neste capítulo serão abordados os aspectos do algoritmo, bem como os detalhes da implementação em CUDA.

4.1 Algoritmo MRDCA-RWL

Proposto em 2012, é uma variação do algoritmo MRDCA (2010) com peso de relevância estimado localmente e está classificado como um método *hard* (rígido) que utiliza múltiplas matrizes de dissimilaridade para agrupar dados relacionais.

Segue uma descrição do algoritmo MRDCA-RWL por Carvalho, Lechevallier e Melo (2012):

(1) Inicialização.

Fixar o número K de grupos;
Fixar a cardinalidade $1 \leq q << n$ de protótipos $G_k(k = 1,...,K)$;
Definir $t = 0$;
Definir $\lambda_k^{(0)} = (\lambda_{k1}^{(0)},...,\lambda_{kp}^{(0)}) = (1,...,1)(k = 1,...,K)$;
Selecionar aleatoriamente K protótipos distintos $G_k^{(0)} \in E^{(q)}(k = 1,...,K)$;
Atribuir cada objeto e_i ao protótipo mais próximo, para obter a partição
$P^{(0)} = (C_1^{(0)},...,C_K^{(0)})$ com $C_k^{(0)} = \{e_i \in E : \sum_{j=1}^{p} \lambda_{kj}^{(0)} D_j(e_i, G_k^{(0)}) \leq$

$\sum_{j=1}^{p} \lambda_{hj}^{(0)} D_j(e_i, G_h^{(0)}), (h = 1, ..., K)\}$

(2) Etapa 1: cálculo dos melhores protótipos.

Definir $t = t + 1$;
A partição $P^{(t-1)} = (C_1^{(t-1)}, ..., C_K^{(t-1)})$ e os vetores de peso de relevância $\lambda_k^{(t-1)} = (\lambda_{k1}^{(t-1)}, ..., \lambda_{kp}^{(t-1)}), k = 1, ..., K$ estão fixados.
Computar o protótipo $G_k^{(t)} = G^* \in E^{(q)}$ do grupo $C_k^{(t-1)} (k = 1, ..., K)$ de acordo com: $G^* = argmin_{G \in E^{(q)}} \sum_{e_i \in C_k^{(t-1)}} \sum_{j=1}^{p} \lambda_{kj}^{(t-1)} D_j(e_i, G)$
$= argmin_{G \in E^{(q)}} \sum_{e_i \in C_k^{(t-1)}} \sum_{j=1}^{p} \lambda_{kj}^{(t-1)} \sum_{e \in G} d_j(e_i, e)$

(3) Etapa 2: cálculo do vetor de peso de melhor relevância.

Os protótipos $G_k^{(t)} \in E^{(q)} (k = 1, ..., K)$ e a partição $P^{(t-1)} = (C_1^{(t-1)}, ..., C_K^{(t-1)})$ estão fixados.
Computar os componentes $\lambda_{kj}^{(t)} (j = 1, ..., p)$ do vetor de peso de relevância $\lambda_k^{(t)} (k = 1..., K)$ de acordo com

$$\lambda_{kj}^{(t)} = \frac{\{\prod_{h=1}^{p} [\sum_{e_i \in C_k^{(t-1)}} D_h(e_i, G_k^{(t)})]\}^{\frac{1}{p}}}{[\sum_{e_i \in C_k^{(t-1)}} D_j(e_i, G_k^{(t)})]} = \frac{\{\prod_{h=1}^{p} [\sum_{e_i \in C_k^{(t-1)}} \sum_{e \in G_k^{(t)}} d_h(ei, e)]\}^{\frac{1}{p}}}{[\sum_{e_i \in C_k^{(t-1)}} \sum_{e \in G_k^{(t)}} d_j(e_i, e)]}$$

(4) Etapa 3: definição da melhor partição.

Os protótipos $G_k^{(t)} \in E^{(q)} (k = 1, ..., K)$ e os vetores de peso de relevância $\lambda_k^{(t)} = (\lambda_{k1}^{(t)}, ..., \lambda_{kp}^{(t)}), k = 1, ..., K$, estão fixados.
$test \leftarrow 0$
$P^{(t)} \leftarrow P^{(t-1)}$
para $i = 1$ até n faça
    encontrar o grupo $C_m^{(t)}$ ao qual e_i pertence
    encontrar o grupo vencedor $C_k^{(t)}$ tal que
        $k = argmin_{1 \le h \le K} \sum_{j=1}^{p} \lambda_{hj}^{(t)} D_j(e_i, G_h^{(t)})$
        $= argmin_{1 \le h \le K} \sum_{j=1}^{p} \lambda_{hj}^{(t)} \sum_{e \in G_h^{(t)}} d_j(e_i, e)$
    se $k \neq m$
        $test \leftarrow 1$
        $C_k^{(t)} \leftarrow C_k^{(t)} \cup \{e_i\}$
        $C_m^{(t)} \leftarrow C_m^{(t)} \backslash \{e_i\}$

(5) Critério de parada.

se $test = 0$ então PARE; caso contrário, vá para 2 (Etapa 1).

4.1.1 Partes Paralelizáveis

No algoritmo MRDCA-RWL há algumas partes suscetíveis para a Computação Paralela. Nessa perspectiva, foi observado por exemplo laços de repetições (ou *loops*) que são blocos comumente paralelizáveis, mas cabe salientar que há muitos cálculos envolvidos, então deve-se testar diversas possibilidades para evitar erros nos resultados.

Sobre isso Stallings (2018) descreve como uma dificuldade que o desenvolvedor enfrenta, visto que só cabe a ele decidir como dividir o trabalho em tarefas executáveis independentemente, ou seja, de forma assíncrona ou paralela.

E como foi discutido no Capítulo 2, existem várias abordagens para implementação de algoritmos paralelos. Nesse sentido, a implementação deverá ser projetada de forma que tarefas maiores sejam divididas em tarefas menores, para favorecer os processos paralelos.

4.2 Implementação Paralela em CUDA

Com uma análise detalhada do algoritmo MRDCA-RWL, a Etapa 1 foi selecionada para adaptação em CUDA. Ela é responsável pela realização do cálculo dos melhores protótipos. Tal escolha é justificada, pois essa etapa apresenta o maior custo computacional com ordem de complexidade $O(n^2 \times p)$, conforme a descrição dos autores Carvalho, Lechevallier e Melo (2012):

> *Step 1: computation of the best prototypes.* For each cluster using each table of dissimilarity the authors test each individual as a candidate prototype. This needs the computation of the distance between an individual $i(i = 1,...n)$ and all elements of each cluster using all p dissimilarity matrices and it costs $O(n^2 * p)$. The selection of the prototype of cardinality q for each cluster needs to sort the vector of individual for each cluster (it costs $O(K \times n \times logn)$) and to select the best q individuals as the prototype (it costs $O(K \times q)$). Thus, the step 1 costs $O(n^2 * p)$. [1]

Para execução dos experimentos, o algoritmo MRDCA-RWL foi implementado com a linguagem de programação C++, embora CUDA suporte outras linguagens, ela é baseada em C, possuindo mais exemplos e uma maior documentação nessa linguagem, então esses fatores determinaram a escolha. Na implementação deste trabalho se fez necessário aplicar o paradigma da Programação Imperativa, uma vez que CUDA não suporta todos os conceitos de Orientação a Objetos (NVIDIA, 2019a). Outras informações sobre a configuração de software estão na Seção 5.2.

[1] *Etapa 1: cálculo dos melhores protótipos.* Para cada grupo usando cada tabela de dissimilaridade, os autores testam cada indivíduo como um protótipo candidato. Isso requer o cálculo da distância entre um indivíduo $i(i = 1,...n)$ e todos os elementos de cada grupo usando todas as p matrizes de dissimilaridade e custa $O(n^2 * p)$. A seleção do protótipo de cardinalidade q para cada cluster precisa ordenar o vetor de indivíduo para cada grupo (isso custa $O(K \times n \times logn)$) e selecionar os melhores q indivíduos como o protótipo (isso custa $O(n^2 * p)$). Assim, a etapa 1 custa $O(n^2 * p)$. (tradução nossa).

Todavia, os algoritmos apresentados nesta monografia estão descritos em pseudocódigo, a fim de melhorar o entendimento. O Algoritmo 1 mostra a versão sequencial do MRDCA-RWL conforme descrito na Seção 4.1. Nele é possível observar a Etapa de Inicialização através dos métodos *alocaMemoria*, *selecionaKPrototipos* e *atribuiObjetoPrototipo*. Em seguida a Etapa 1 é enfatizada, depois a Etapa 2 com o método *calculaPesoRelevanciaLocal*, e por fim a Etapa 3 com o método *selecionaMelhorParticao*.

Ainda no Algoritmo 1 foi dado ênfase na Etapa 1 com o intuito de mostrar o alto nível de aninhamento dos laços de repetições (*loops*). Além do somatório central que é usado para a seleção dos melhores protótipos. Uma vez que os dois últimos laços representam o maior número de iterações e consequentemente influencia o tempo de execução, por isso a Etapa 1 foi paralelizada.

No Algoritmo 2 está a versão paralela proposta por este trabalho, além dos métodos já citados no Algoritmo 1, foi incluído o método *alocaMemoriaCUDA*, responsável por alocar os espaços de memória na GPU. E a chamada ao método *somaPrototiposCUDA* em que foi executado o somatório paralelizado na GPU. Antes disso, foi definida a variável *threads* com o valor 1.024, isso é o número máximo de *threads* por bloco que CUDA permite (NVIDIA, 2019b). E a variável *blocos* com a expressão $(n + threads - 1) \div threads$, para distribuir a execução através dos blocos e *threads* em função do número de objetos (N). Desse modo, essas duas variáveis são parâmetros especiais dos *kernels* CUDA (NVIDIA, 2019a). A cada iteração do segundo laço é executada a função nativa *cudaDeviceSynchronize* que aguarda todas as *threads* serem concluídas para sincronizar o fluxo com o *host*.

O método *somaPrototiposCUDA* descrito no Algoritmo 3 é executado totalmente na GPU, também chamada de *Device*. Nele a variável h armazena um índice único de identificação entre as *threads*, então a primeira condição substitui o primeiro laço, um vez que a execução paralela das *threads* equivale a todas iterações desse laço. E assim como foi descrito na Seção 3.2, CUDA organiza as *threads* em 3 dimensões, para essa implementação foi usada a dimensão *blockDim.x*.

A alocação de memória e a condição de corrida foram dois desafios enfrentados na implementação da versão paralelizada em CUDA. Mais detalhes são descritos nas subseções a seguir.

4.2.1 Alocação de Memória

A alocação de memória é um fator preponderante para o desempenho da paralelização e seu custo deve ser bem avaliado. Sendo assim, a transferência de memória é considerada um custo extra (HONG-TAO et al., 2009).

Além disso, os padrões de acesso à memória devem ser cuidadosamente projetados para evitar gargalos, caso contrário, o desempenho da GPU pode ser significativamente degradado

Algoritmo 1: MRDCA-RWL - Etapa 1 - Cálculo dos melhores protótipos - Sequencial

Entrada:

$dados$	Vetor de dados
n	Número de objetos
k	Número de grupos
p	Número de atributos
q	Número de medoids
$matrizes$	Matrizes multidimensionais

```
1  alocaMemoria(lambda, g, conta, k, q)
2  selecionaKPrototipos(dados, g, k, q)
3  atribuiObjetoPrototipo(dados, conta, n, k, p, q, matrizes)
4  repita
5      para i ← 0 até i < k faça
6          para j ← 0 até j < n faça
7              c_j ← 0
8          fim
9          para m ← 0 até m < p faça
10             para h ← 0 até h < n faça
11                 v ← 0
12                 para l ← 0 até l < n faça
13                     se dados_l.c = i então
14                         v ← v + matrizes_mhl
15                     fim
16                 fim
17                 c_h ← lambda_im · v)
18             fim
19         fim
20         calculaMin(c, g, n, q)
21     fim
22     calculaPesoRelevanciaLocal(dados, p, k, g, q, n, matrizes)
23     selecionaMelhorParticao(teste, dados, conta, p, k, g, q, n, matrizes)
24 até teste = 1
```

(SIROTKOVIć; DUJMIć; PAPIć, 2012). Como ocorreu com Zechner e Granitzer (2009), em que problemas na alocação e limite de memória prejudicaram a aceleração na versão paralela em GPU quando comparada com a versão sequencial em CPU.

Nesse sentido, a alocação de memória deste trabalho foi planejada levando-se em conta o hardware disponível nos experimentos (apresentados no Capítulo 5). Com isso, foram selecionados os conjuntos de dados com até 21 mil objetos, pois acima desse limite haveria um estouro de memória ou *out of memory*.

Sendo assim, antes de executar o algoritmo MRDCA-RWL as matrizes de dissimilaridades devem ser alocadas na memória principal (RAM), essas matrizes são *arrays* multidimensionais. Dessa forma, para a paralelização em CUDA também é necessário efetuar uma cópia para a memória global da GPU. CUDA possui funções nativas para realizar essas operações, como a

Algoritmo 2: MRDCA-RWL - Etapa 1 - Cálculo dos melhores protótipos - Paralelo

Entrada:

$dados$	Vetor de dados
n	Número de objetos
k	Número de grupos
p	Número de atributos
q	Número de medoids
$matrizes$	Matrizes multidimensionais
$matriz1d$	Matriz unidimensional CUDA

```
1  alocaMemoriaCUDA(dcgrupo, dmatriz1d, dc, n, p)
2  alocaMemoria(lambda, g, conta, k, q)
3  selecionaKPrototipos(dados, g, k, q)
4  atribuiObjetoPrototipo(dados, conta, n, k, p, q, matrizes)
5  cudaMemcpy(dcgrupo, dados.c, n, cudaMemcpyHostToDevice)
6  cudaMemcpy(dmatriz1d, matriz1d, p·n·n, cudaMemcpyHostToDevice)
7  repita
8      para i ← 0 até i < k faça
9          para j ← 0 até j < n faça
10             c_j ← 0
11         fim
12         cudaMemset(dc, 0, n)
13         para m ← 0 até m < p faça
14             threads ← 1024
15             blocos ← (n + threads − 1) ÷ threads
16             somaPrototiposCUDA <<< blocos, threads >>>
                   (n, i, m, dcgrupo, dmatriz1d, lambda_im, dc)
17         fim
18         cudaDeviceSynchronize
19         cudaMemcpy(c, dc, n, cudaMemcpyDeviceToHost)
20         calculaMin(c, g, n, q)
21     fim
22     calculaPesoRelevanciaLocal(dados, p, k, g, q, n, matrizes)
23     selecionaMelhorParticao(teste, dados, conta, p, k, g, q, n, matrizes)
24 até teste = 1
```

função *cudaMalloc* para alocação e a função *cudaMemcpy* para a cópia. Tal cópia pode ser bidirecional, dependendo do parâmetro *cudaMemcpyHostToDevice* ou *cudaMemcpyDeviceToHost*. Essas funções foram implementadas no Algoritmo 2.

No entanto, como esses dois métodos são restritos apenas para *arrays* unidimensionais, então foi necessário converter as matrizes multidimensionais para unidimensionais antes de serem processadas na GPU (NVIDIA, 2019b). Desse modo, foi necessário criar um mapeamento dos elementos da matriz unidimensional através do índice *ix* com a expressão $(l + n \cdot (h + n \cdot m))$, conforme mostra o Algoritmo 3.

Algoritmo 3: somaPrototiposCUDA - Somatório dos melhores protótipos - *Kernel* CUDA

Entrada:

n	Número de objetos
i	Índice do grupo atual
m	Índice do atributo atual
dcg	Vetor do índice do grupo
$dm1d$	Matriz unidimensional
dlb	Lambda

Saída:

dc	Vetor com somatório

```
1  h ← blockIdx.x · blockDim.x + threadIdx.x
2  se h < n então
3      v ← 0
4      para l ← 0 até l < n faça
5          se dcg_l = i então
6              ix ← (l + n · (h + n · m))
7              v ← v + dm1d_ix
8          fim
9      fim
10     atomicAdd(dc_h, dlb · v)
11 fim
```

4.2.2 Condição de Corrida

Lidar com a Concorrência é um grande desafio na Computação Paralela. Nesse sentido deve-se evitar a condição de corrida. Esse conceito é definido por Stallings (2018) como "A race condition occurs when multiple processes or threads read and write data items so that the final result depends on the order of execution of instructions in the multiple processes."[2]

O Algoritmo 3 descreve o método *somaPrototiposCUDA* e por ser um método *kernel*, ou seja, paralelizado na GPU. Ele será distribuído e executado em várias *threads*. Nele é realizado um somatório, contudo esse cálculo ficaria incorreto se a condição de corrida não fosse evitada. Para isso, CUDA possui funções atômicas como o *atomicAdd*, ele garante que apenas uma *thread* por vez tenha acesso a variável global que está sendo somada. Tal função foi utilizada nesse algoritmo.

[2] Uma condição de corrida ocorre quando vários processos ou threads leem e gravam itens de dados para que o resultado final dependa da ordem de execução das instruções nos vários processos. (tradução nossa).

5 Experimentos Empíricos

Após a implementação, execução dos algoritmos e coleta de dados neste capítulo serão apresentados os conjuntos de dados selecionados, a configuração utilizada, os resultados obtidos e a análise dos mesmos.

5.1 Conjuntos de Dados

O UCI Machine Learning Repository [1] é uma coleção de bancos de dados, teorias de domínio e geradores de dados que são usados pela comunidade de aprendizado de máquina para a análise empírica de algoritmos de aprendizado de máquina. O arquivo foi criado em 1987 por David Aha e seus colegas de pós-graduação na UC Irvine (DUA; GRAFF, 2017).

Tabela 1 – Conjunto de Dados

Conjunto de Dados	**Objetos**	**Atributos**	**Grupos**
Image Segmentation	2.310	19	7
Abalone	4.177	9	3
Statlog (Landsat Satellite)	6.435	36	7
Facebook Live Sellers in Thailand	7.051	12	4
Electrical Grid Stability Simulated Data	10.000	14	2
Online Shoppers Purchasing Intention	12.330	18	8
EEG Eye State	14.980	15	2
HTRU2	17.898	9	2
MAGIC Gamma Telescope	19.020	11	2
Avila	20.867	10	12

Fonte: o autor (2019).

Desse repositório foram selecionados os conjuntos de dados para os experimentos empíricos. Sendo assim, a Tabela 1 relaciona cada conjunto de dados com seu respectivo número

[1] https://archive.ics.uci.edu/ml/index.php

de objetos, atributos e grupos (classes). Nesta monografia os dois critérios utilizados para selecionar o conjunto de dados foram o número de objetos e atributos, pois quanto maior o conjunto de dados maior é o tempo de execução dos algoritmos de agrupamento. Com isso, a comparação de desempenho com a versão paralela do algoritmo MRDCA-RWL poderá ser mais relevante. Tais conjuntos de dados são descritos nas próximas subseções.

5.1.1 *Image Segmentation*

O *Image Segmentation* é formado por dados de imagem descritos por atributos de valor numérico de alto nível, com 7 classes (ou grupos). As instâncias foram sorteadas aleatoriamente a partir de um banco de dados de 7 imagens externas. As imagens foram segmentadas manualmente para criar uma classificação para cada pixel. Cada instância é uma região 3x3 (DUA; GRAFF, 2017).

5.1.2 *Abalone*

O conjunto de dados *Abalone* contém a previsão da idade do abalone (molusco marinho) de medições físicas. Essa idade é determinada cortando a concha através do cone, manchando-a e contando o número de anéis através de um microscópio. Outras medidas, mais fáceis de obter são usadas para prever a idade. Outras informações, como padrões climáticos e localização podem ser necessárias para solucionar o problema (DUA; GRAFF, 2017).

5.1.3 *Statlog (Landsat Satellite)*

O *Statlog (Landsat Satellite)* consiste nos valores multi-espectrais de pixels em vizinhanças 3x3 em uma imagem de satélite e na classificação associada ao pixel central em cada bairro. O objetivo é prever esta classificação, dados os valores multi-espectrais. No banco de dados de amostra, a classe de um pixel é codificada como um número (DUA; GRAFF, 2017).

5.1.4 *Facebook Live Sellers in Thailand*

O conjunto de dados *Facebook Live Sellers in Thailand* contém páginas do Facebook de 10 vendedores de varejo de moda e cosméticos tailandeses. Postagens de natureza diferente (vídeo, fotos, status e links). As métricas de engajamento consistem em comentários, compartilhamentos e reações. A variabilidade do engajamento do consumidor é analisada através de uma Análise de Componentes Principais, destacando as mudanças induzidas pelo uso do Facebook Live. O componente sazonal é analisado por meio de um estudo das médias das diferentes métricas de engajamento para diferentes períodos de tempo (horário, diário e mensal) (DUA; GRAFF, 2017).

5.1.5 *Electrical Grid Stability Simulated Data*

O *Electrical Grid Stability Simulated Data* contém uma análise de estabilidade local do sistema estelar de 4 nós (o produtor de eletricidade está no centro) implementando o conceito *Decentral Smart Grid Control*. A análise é realizada para diferentes conjuntos de valores de entrada (DUA; GRAFF, 2017).

5.1.6 *Online Shoppers Purchasing Intention*

O conjunto de dados *Online Shoppers Purchasing Intention* foi formado para que cada sessão pertencesse a um usuário diferente em um período de 1 ano, para evitar qualquer tendência para uma campanha específica, dia especial, usuário perfil ou período. Das 12.330 sessões no conjunto de dados, 84,5% (10.422) foram amostras de classe negativa que não terminaram com compras, e o resto (1.908) foram amostras de classe positivas que terminaram com compras (DUA; GRAFF, 2017).

5.1.7 *EEG Eye State*

O *EEG Eye State* consiste em 14 valores de Eletroencefalografia (EEG) e um valor indicando o estado do olho. Todos os dados são de uma medição EEG contínua com o Emotiv EEG Neuroheadset. A duração da medição foi de 117 segundos. O estado do olho foi detectado através de uma câmera durante a medição do EEG e adicionado posteriormente manualmente ao arquivo após a análise dos quadros de vídeo. Dessa maneira, '1' indica o olho fechado e '0' o estado de olho aberto. Todos os valores estão em ordem cronológica com o primeiro valor medido no topo dos dados (DUA; GRAFF, 2017).

5.1.8 *HTRU2*

O *HTRU2* é um conjunto de dados que descreve uma amostra de candidatos a pulsares coletados durante o Levantamento do Universo de Resolução em Tempo Alto (Sul). Os pulsares são um tipo raro de estrela de nêutrons que produzem emissões de rádio detectáveis aqui na Terra (DUA; GRAFF, 2017).

5.1.9 *MAGIC Gamma Telescope*

O *MAGIC Gamma Telescope* é um *dataset* gerado para simular o registro de partículas gama de alta energia em um telescópio gama atmosférico de Cherenkov. O telescópio gama Cherenkov observa os raios gama de alta energia, aproveitando a radiação emitida pelas partículas carregadas produzidas dentro dos chuveiros eletromagnéticos iniciados pelas gamas e se desenvolvendo na atmosfera. Milhares de fótons de Cherenkov foram coletados em padrões (chamados de imagem do chuveiro), permitindo discriminar estatisticamente aqueles causados

por gamas primários (sinal) das imagens de chuveiros hadrônicos iniciados por raios cósmicos na atmosfera superior (fundo) (DUA; GRAFF, 2017).

5.1.10 *Avila*

O conjunto de dados de *Avila* foi extraído de 800 imagens da "Bíblia Ávila", uma cópia gigante da Bíblia em latim do século XII. A tarefa de previsão consiste em associar cada padrão a um copista. Os dados foram normalizados usando o método de normalização Z (DUA; GRAFF, 2017).

5.2 Configuração

Para efeito de replicação dos experimentos deste trabalho, o Quadro 1 especifica as configurações de hardware e software utilizadas pelo autor.

Quadro 1 – Configuração dos Experimentos.

Hardware	Notebook Dell® Inspiron™7559
CPU Clock Núcleos	Intel® Core™i7-6700HQ 2.60 GHz - 3.50 GHz 4 (8 lógicos)
Memória RAM	16 GB DDR3 1600 MHz
Armazenamento	1 TB SSD Samsung® 850 EVO
GPU Arquitetura Clock Núcleos CUDA Memória	NVIDIA® GeForce™GTX 960M Maxwell™ 1096 MHz 640 4 GB GDDR5 2500 MHz 128 bits 80 GB/s
SO Versão do drive da GPU Versão do CUDA Toolkit Versão do DirectX IDE Linguagem	Microsoft® Windows™10 Pro 64 bits (versão 1809) 26.21.14.3064 10.1.105 12 Microsoft® Visual Studio Community 2015 C++

Fonte: o autor (2019).

5.3 Resultados

Os experimentos empíricos foram realizados através da execução da versão sequencial e paralela do algoritmo MRDCA-RWL em cada conjunto de dados. Todas as execuções tiveram seus tempos medidos para cada cenário e seus dados foram armazenados, coletados e sumarizados através de técnicas de ETL (do inglês *Extract Transform Load*).

Desse modo, todas as métricas de tempos de execução são médias, obtidas após 15 (quinze) execuções em cada conjunto de dados. E são referente a execução da Etapa 1 (cálculo

dos melhores protótipos) do algoritmo, visto que essa foi a parte do algoritmo paralelizada em CUDA. O motivo e demais detalhes estão mencionados no Capítulo 4.

Em seguida, o tempo de execução da versão paralela implementada foi comparado com o tempo da versão sequencial equivalente, a fim de mensurar a aceleração. Assim sendo, as subseções a seguir vão apresentar e discutir os resultados.

5.3.1 Comparativo por Número de Objetos

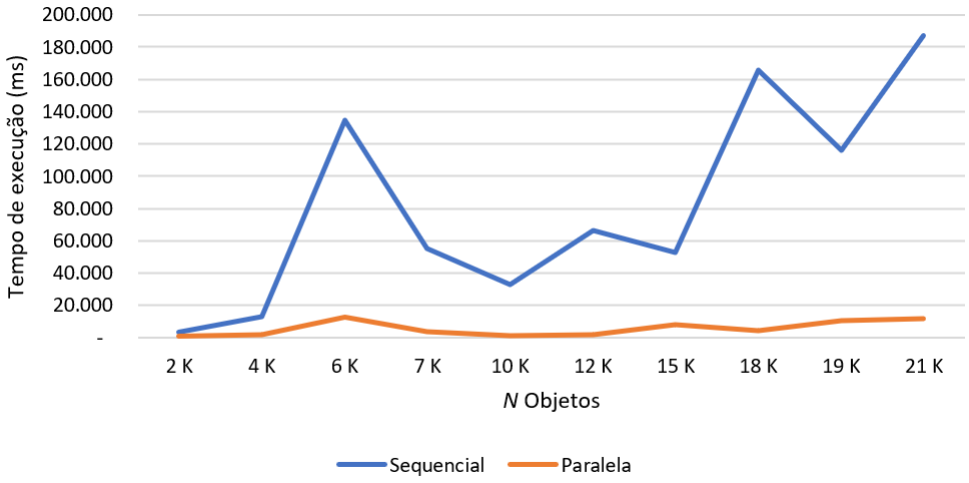
Como cada conjunto de dados possui características distintas, os experimentos e a análise foram elaboradas focando nessas diferenças. O primeiro experimento foi executado com o intuito de medir a aceleração obtida em função do número de objetos (N) que o conjunto de dados contém.

Então como se pode observar na Figura 7 o tempo de execução em milissegundos (ms) da versão sequencial varia bastante e segue uma trajetória crescente em função do N Objetos passando dos 180.000 ms. No entanto, a versão paralela segue uma trajetória com pouco crescimento com tempos de execução abaixo dos 20.000 ms, mesmo com o aumento significativo do N Objetos de 2 K para 21 K.

Nesse sentido, é notória a diferença no comparativo. Quanto a aceleração obtida na versão paralela em CUDA, ela foi maior em conjuntos de dados com grande número de objetos, como mostra a Figura 8, variando de 4 a 37,9 vezes mais rápido que a versão sequencial.

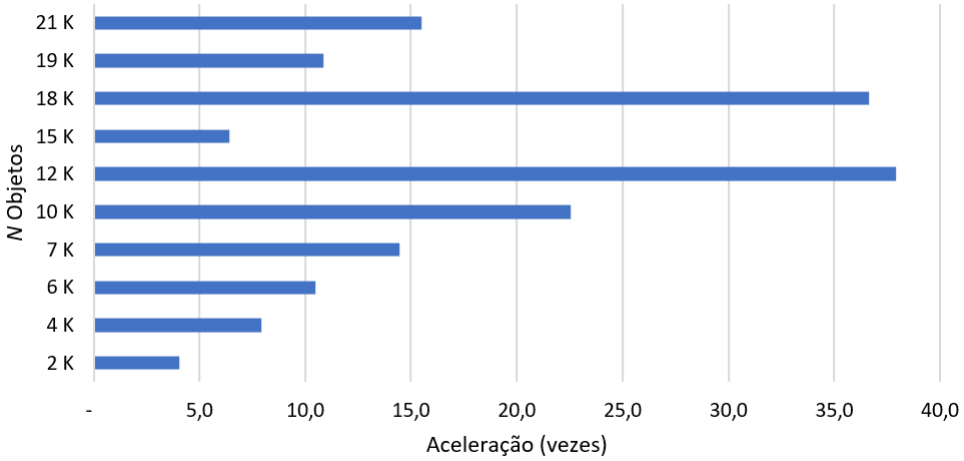
Os dados desses comparativos estão mais detalhados na Tabela 2, em que são apresentados os tempos de execução por conjunto de dados e sua respectiva aceleração.

Figura 7 – Comparativo de tempo de execução em milissegundos (ms) da versão Sequencial e Paralela (CUDA) do algoritmo MRDCA-RWL por número de objetos (N) em cada conjunto de dados.



Fonte: o autor (2019).

Figura 8 – Fator de aceleração (em vezes) da versão e Paralela (CUDA) do algoritmo MRDCA-RWL em relação a Sequencial por número de objetos (N) em cada conjunto de dados.



Fonte: o autor (2019).

Tabela 2 – Comparativo de tempo de execução em milissegundos (ms) da versão Sequencial e Paralela do algoritmo MRDCA-RWL e fator de aceleração (em vezes) em cada conjunto de dados por ordem do número de objetos (N).

Conjunto de Dados	**N Objetos**	**Sequencial**	**Paralela**	**Aceleração**
Image Segmentation	2 K	3.646	905	4,0
Abalone	4 K	12.919	1.634	7,9
Statlog (Landsat Satellite)	6 K	134.898	12.845	10,5
Facebook Live Sellers in Thailand	7 K	55.057	3.805	14,5
Electrical Grid Stability Simulated Data	10 K	32.945	1.461	22,5
Online Shoppers Purchasing Intention	12 K	66.339	1.748	37,9
EEG Eye State	15 K	52.800	8.254	6,4
HTRU2	18 K	165.586	4.519	36,6
MAGIC Gamma Telescope	19 K	116.186	10.711	10,8
Avila	21 K	186.917	12.076	15,5
		827.294	**57.960**	**14,3**

Fonte: o autor (2019).

5.3.2 Comparativo por Número de Iterações

Outra forma de análise foi a respeito do número de iterações que são realizadas na Etapa 1 do algoritmo (conforme descrito na Seção 4.2), uma vez que para calcular os melhores protótipos os laços de repetições aninhados são iterados uma grande quantidade de vezes.

Essa quantidade de iterações da Etapa 1 é calculada com a expressão N^2KP, onde N é o número de objetos, K o número de grupos que cada conjunto de dados possui e P é o número de atributos selecionados para o experimento, nesse caso representam a quantidade de matrizes de

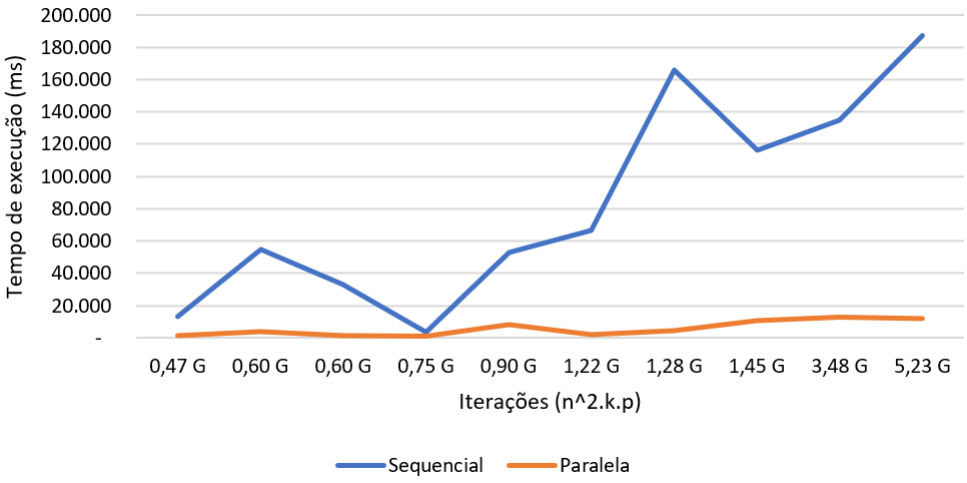
dissimilaridade. São essas as três principais propriedades do conjunto de dados e foram reunidas para equilibrar a comparação.

A Figura 9 apresenta a quantidade de iterações na ordem de 10^9 (G) variando de 0,47 G até 5,23 G. É possível perceber que a versão sequencial aumenta o tempo de execução de forma ainda mais acentuada quando o número de iterações cresce. Contudo, a versão paralela cresce de forma discreta até o limite superior.

A aceleração foi heterogênea, conforme mostra a Figura 10, pois a seleção dos protótipos iniciais é aleatória, o que acaba alterando o tempo de convergência do algoritmo. Outro fator que também influencia a execução de forma significativa é o número de objetos N. Na Tabela 3 estão os dados desses comparativos por iterações.

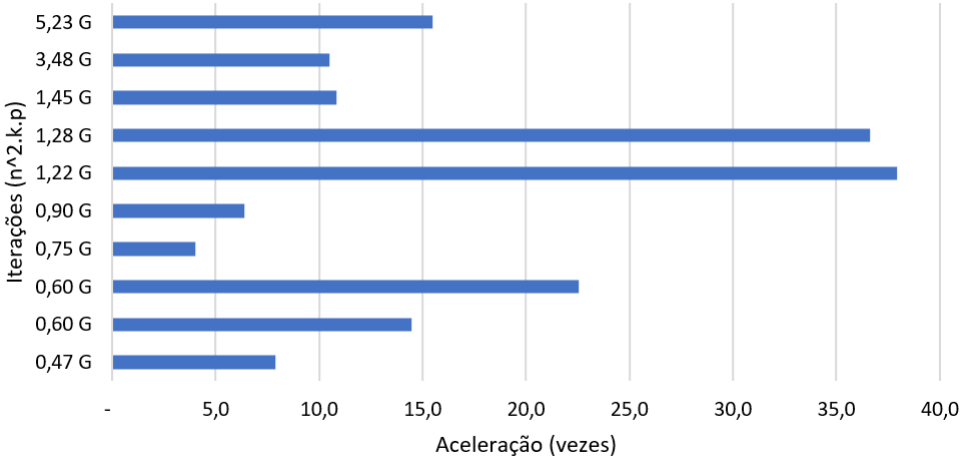
Quando todos os tempos são somados, a versão sequencial atinge 827.294 ms e a versão paralela 57.960 ms, com uma redução total de 769.334 ms, ou seja, a versão paralela foi 93% mais rápida que a versão sequencial.

Figura 9 – Comparativo de tempo de execução em milissegundos (ms) da versão Sequencial e Paralela (CUDA) do algoritmo MRDCA-RWL por número de iterações (N^2KP) em cada conjunto de dados.



Fonte: o autor (2019).

Figura 10 – Fator de aceleração (em vezes) da versão e Paralela (CUDA) do algoritmo MRDCA-RWL em relação a Sequencial por número de iterações (N^2KP) em cada conjunto de dados.



Fonte: o autor (2019).

Tabela 3 – Comparativo de tempo de execução em milissegundos (ms) da versão Sequencial e Paralela do algoritmo MRDCA-RWL e fator de aceleração (em vezes) por ordem do número de iterações (N^2KP).

N **Objetos**	K **Grupos**	P **Atributos**	N^2KP	**Sequencial**	**Paralela**	**Aceleração**
4.177	3	9	0,47 G	12.919	1.634	7,9
7.050	4	3	0,60 G	55.057	3.805	14,5
10.000	2	3	0,60 G	32.945	1.461	22,5
2.310	7	20	0,75 G	3.646	905	4,0
14.980	2	2	0,90 G	52.800	8.254	6,4
12.330	8	1	1,22 G	66.339	1.748	37,9
17.898	2	2	1,28 G	165.586	4.519	36,6
19.020	2	2	1,45 G	116.186	10.711	10,8
6.435	7	12	3,48 G	134.898	12.845	10,5
20.867	12	1	5,23 G	186.917	12.076	15,5
				827.294	**57.960**	**14,3**

Fonte: o autor (2019).

5.3.3 Comparativo por Número de Atributos

Para a comparação por número de atributos (P), foram selecionados dois conjuntos de dados que permitiam variar os atributos: *Electrical Grid Stability Simulated Data* e o *Statlog (Landsat Satellite)*. Essa análise é importante pois cada atributo representa uma matriz de dissimilaridade.

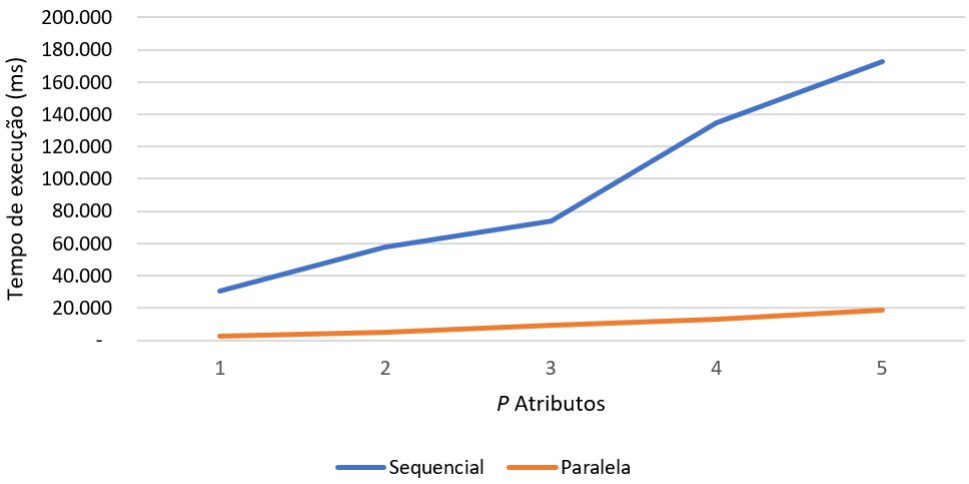
No primeiro conjunto de dados *Statlog (Landsat Satellite)* foi executado variando de 3 a 15 atributos (P), com intervalo de 3. E nesse experimento houve uma diferença significativa entre as duas versões, como mostra a Figura 11. A versão sequencial levou para 3 atributos 30.746 ms

até 172.542 ms para 15 atributos. E a versão paralela variou o tempo de execução de 2.944 ms até 18.970 ms. Dessa forma, mostrando uma aceleração máxima de 11,3 vezes para 6 atributos e mínima de 8,0 vezes para 9 atributos, conforme a Figura 12.

E no segundo conjunto de dados *Electrical Grid Stability Simulated Data* foi executado de 1 a 7 atributos (*P*) com acentuada diferença entre as duas versões, como mostra a Figura 13. A versão sequencial levou para 1 atributo 20.972 ms até 88.415 ms para 7 atributos. Já a versão paralela variou o tempo de execução de 187 ms até 4.177 ms. Mostrando uma aceleração máxima de 111,9 vezes para 1 atributo e 21,2 vezes para até 7 atributos, conforme a Figura 14.

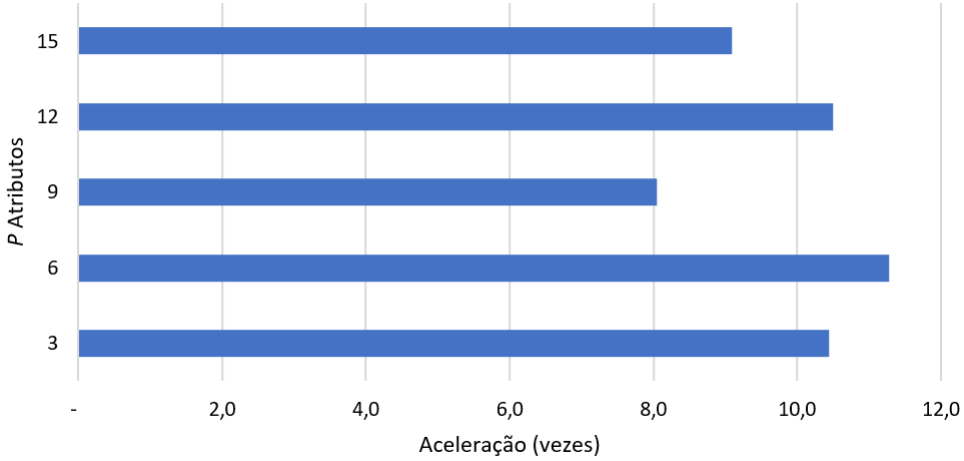
Nesse experimento observou-se que quanto maior o número de atributos, maior é o tempo de execução de forma quase linear. Contudo, esse crescimento proporcional é em média 92% menor na versão paralela, se comparado a versão sequencial. As Tabelas 5 e 4 detalham todas as métricas por atributos.

Figura 11 – Comparativo de tempo de execução em milissegundos (ms) da versão Sequencial e Paralela do algoritmo MRDCA-RWL com o conjunto de dados *Statlog (Landsat Satellite)* variando o número de atributos (*P*).



Fonte: o autor (2019).

Figura 12 – Fator de aceleração (em vezes) da versão e Paralela (CUDA) do algoritmo MRDCA-RWL com o conjunto de dados *Statlog (Landsat Satellite)* variando o número de atributos (*P*).



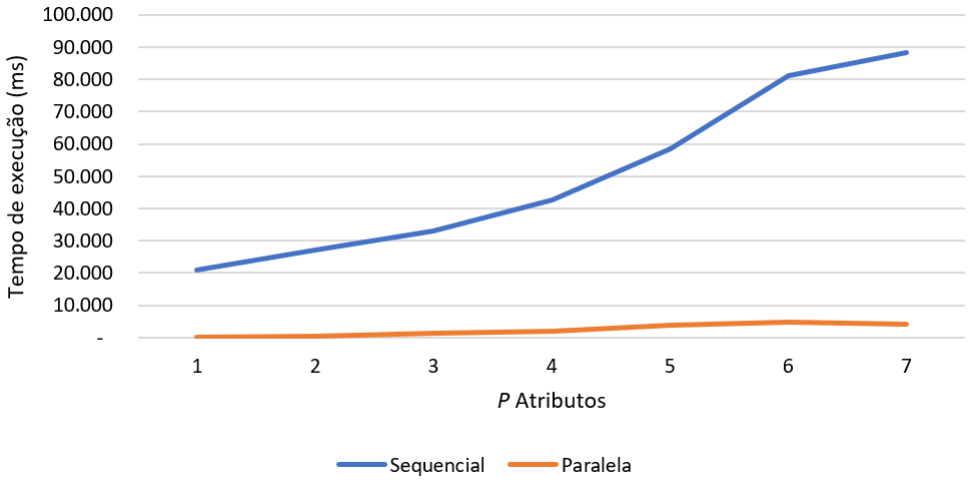
Fonte: o autor (2019).

Tabela 4 – Comparativo de tempo de execução em milissegundos (ms) da versão Sequencial e Paralela do algoritmo MRDCA-RWL com o conjunto de dados *Statlog (Landsat Satellite)* variando o número de atributos (*P*).

***P* Atributos**	**Sequencial**	**Paralela**	**Aceleração**
3	30.746	2.944	10,4
6	57.768	5.121	11,3
9	74.080	9.203	8,0
12	134.898	12.845	10,5
15	172.542	18.970	9,1
	470.034	**49.083**	**9,6**

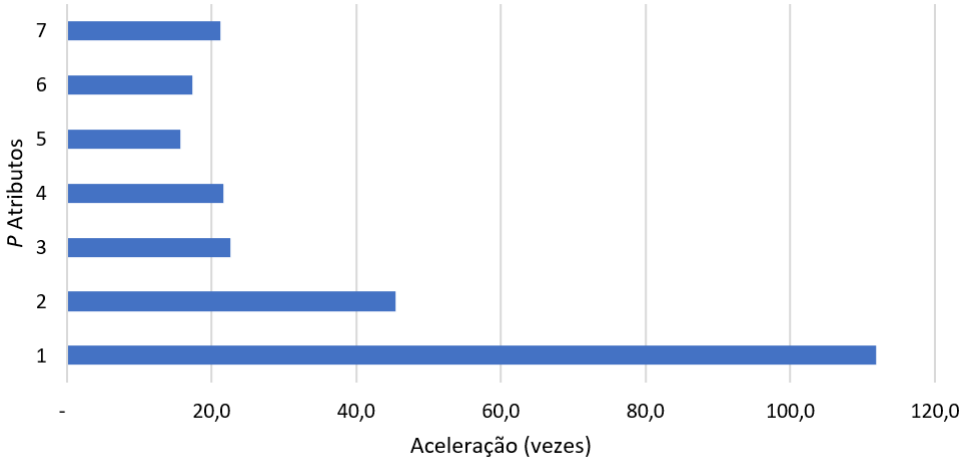
Fonte: o autor (2019).

Figura 13 – Comparativo de tempo de execução em milissegundos (ms) da versão Sequencial e Paralela do algoritmo MRDCA-RWL com o conjunto de dados *Electrical Grid Stability Simulated Data* variando o número de atributos (*P*).



Fonte: o autor (2019).

Figura 14 – Fator de aceleração (em vezes) da versão e Paralela (CUDA) do algoritmo MRDCA-RWL com o conjunto de dados *Electrical Grid Stability Simulated Data* variando o número de atributos (*P*).



Fonte: o autor (2019).

Tabela 5 – Comparativo de tempo de execução em milissegundos (ms) da versão Sequencial e Paralela do algoritmo MRDCA-RWL com o conjunto de dados *Electrical Grid Stability Simulated Data* variando o número de atributos (*P*).

***P* Atributos**	**Sequencial**	**Paralela**	**Aceleração**
1	20.972	187	111,9
2	27.236	598	45,5
3	32.945	1.461	22,5
4	42.754	1.977	21,6
5	58.565	3.719	15,7
6	81.161	4.695	17,3
7	88.415	4.177	21,2
	352.048	**16.815**	**20,9**

Fonte: o autor (2019).

5.3.4 Análise do Desempenho

Após a apresentação dos comparativos por número de objetos, iterações e atributos ficou perceptível a redução do tempo de execução em todos os experimentos. Dessa forma, a versão paralela da etapa 1 do algoritmo MRDCA-RWL apresentou um desempenho superior chegando a uma aceleração média de 16,7 vezes, se comparada a sua versão sequencial.

Com essa média de tempo de execução, um experimento que antes duraria 1 hora será executado em cerca de 4 minutos. Alcançando desse modo uma otimização de tempo significativa.

Nesse contexto, assim como foi mencionado em Wu e Hong (2011), Bhimani, Leeser e Mi (2015) e Karbhari e Alawneh (2018), os experimentos empíricos deste trabalho também obtiveram êxito ao paralelizar um algoritmo de Agrupamento de Dados com a tecnologia CUDA através da GPGPU.

6

Conclusão

Como foi apresentado ao longo desta monografia existem vários algoritmos de Agrupamentos de Dados e tecnologias para Computação Paralela. Muitos desses são aplicados para a organização do crescente volume de informação trafegado na Internet. Contudo, o algoritmo MRDCA-RWL foi executado em conjuntos de dados relativamente pequenos e por ser um algoritmo recente (2012) que utiliza dados relacionais, há publicações apenas de versões sequenciais. Logo, esses fatos motivaram a elaboração deste trabalho, que propôs uma versão paralela desse algoritmo com o objetivo de comparar o desempenho e obter um tempo de execução menor mesmo em grandes conjuntos de dados.

Assim como houve vários resultados positivos na revisão bibliográfica para outros algoritmos, os experimentos realizados neste estudo mostraram que essa versão paralela implementada com a tecnologia CUDA possui vantagens significativas em relação a versão sequencial, pois obteve-se uma aceleração média de 16,7 vezes no tempo de execução da etapa 1 do algoritmo, e em número relativo é 93% mais rápido. É importante salientar que há uma grande probabilidade de incrementar essa aceleração considerando-se *datasets* maiores e um hardware avançado.

Nessa perspectiva, com esses resultados a versão paralela alcançou uma boa escalabilidade mesmo quando o volume de dados aumentou, ou seja, isso abre inúmeras possibilidades para aplicações em diversas áreas como na Computação Científica, Inteligência Artificial, Mineração de Dados, entre outras.

Portanto, o objetivo deste trabalho foi alcançado, uma vez que a implementação da versão paralela do algoritmo MRDCA-RWL obteve um desempenho superior em relação a versão sequencial conhecida. Então, ao aplicar a Computação Paralela em algoritmos de Agrupamento de Dados favorece a implementações altamente escaláveis, e isso é um requisito cada vez mais necessário principalmente em um cenário de *Big Data*.

6.1 Limitações do Trabalho

Pelo tempo disponível na elaboração deste trabalho, só foi possível implementar e executar os experimentos em apenas uma tecnologia paralela (CUDA), uma vez que outras tecnologias de paralelização poderiam ter sido usadas para efeito de comparação. Um outro ponto foi em relação ao tamanho da memória disponível no hardware, que acabou definindo o limite máximo dos conjuntos de dados usados nos experimentos.

6.2 Trabalhos Futuros

Para trabalhos futuros, outras partes do algoritmo MRDCA-RWL como menor custo computacional também podem ser paralelizadas, a fim de melhorar ainda mais os resultados. Também é recomendado executar a versão paralela proposta neste trabalho em uma configuração de hardware superior. Nesse sentido, seria importante observar se a média de aceleração se manterá para conjuntos de dados com mais atributos e objetos.

Referências

BAYDOUN, M.; DAWI, M.; GHAZIRI, H. Enhanced parallel implementation of the k-means clustering algorithm. In: *2016 3rd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA)*. [S.l.: s.n.], 2016. p. 7–11. Citado 2 vezes nas páginas 19 e 24.

BAYDOUN, M.; GHAZIRI, H.; AL-HUSSEINI, M. Cpu and gpu parallelized kernel k-means. *J. Supercomput.*, Kluwer Academic Publishers, Norwell, MA, USA, v. 74, n. 8, p. 3975–3998, ago. 2018. ISSN 0920-8542. Disponível em: <https://doi.org/10.1007/s11227-018-2405-7>. Citado 3 vezes nas páginas 18, 19 e 20.

BHIMANI, J.; LEESER, M.; MI, N. Accelerating k-means clustering with parallel implementations and gpu computing. In: *2015 IEEE High Performance Extreme Computing Conference (HPEC)*. [S.l.: s.n.], 2015. p. 1–6. Citado 6 vezes nas páginas 15, 18, 20, 21, 22 e 46.

CANO, A. A survey on graphic processing unit computing for large-scale data mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, v. 8, n. 1, p. e1232, 2018. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1232>. Citado 2 vezes nas páginas 24 e 25.

CARVALHO, F. de A.T. de; LECHEVALLIER, Y.; MELO, F. M. de. Partitioning hard clustering algorithms based on multiple dissimilarity matrices. *Pattern Recognition*, v. 45, n. 1, p. 447–464, 2012. ISSN 0031-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0031320311002640>. Citado 4 vezes nas páginas 15, 16, 28 e 30.

CUOMO, S. et al. A gpu-accelerated parallel k-means algorithm. *Computers & Electrical Engineering*, 2017. ISSN 0045-7906. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0045790617327994>. Citado na página 14.

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>. Acesso em: 10 abr. 2019. Citado 4 vezes nas páginas 35, 36, 37 e 38.

FAKHI, H. et al. New optimized gpu version of the k-means algorithm for large-sized image segmentation. In: *2017 Intelligent Systems and Computer Vision (ISCV)*. [s.n.], 2017. p. 1–6. Disponível em: <https://ieeexplore.ieee.org/document/8054924>. Citado na página 15.

GAO, B. et al. A method to accelerate k-means and gmm computation with gpu and multi-core cpu. In: *2018 IEEE Fourth International Conference on Multimedia Big Data (BigMM)*. [S.l.: s.n.], 2018. p. 1–5. Citado na página 14.

GOOGLE. *Ooh! Ahh! Google Images presents a nicer way to surf the visual web*. 2010. Disponível em: <https://googleblog.blogspot.com/2010/07/ooh-ahh-google-images-presents-nicer.html>. Acesso em: 11 fev. 2019. Citado na página 14.

HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques*. 3. ed. São Francisco, CA, EUA: Morgan Kaufmann Publishers Inc., 2011. ISBN 0123814790, 9780123814791. Citado 2 vezes nas páginas 14 e 18.

HONG-TAO, B. et al. K-means on commodity gpus with cuda. In: *2009 WRI World Congress on Computer Science and Information Engineering*. [S.l.: s.n.], 2009. v. 3, p. 651–655. Citado 5 vezes nas páginas 15, 18, 19, 20 e 31.

INSTAGRAM. *700 million*. 2017. Disponível em: <https://instagram-press.com/blog/2017/04/26/700-million/>. Acesso em: 15 mar. 2019. Citado na página 14.

JAIN, A. K. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, v. 31, n. 8, p. 651 – 666, 2010. ISSN 0167-8655. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR). Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167865509002323>. Citado 4 vezes nas páginas 14, 15, 18 e 19.

JAIN, A. K.; DUBES, R. C. *Algorithms for clustering data*. Upper Saddle River, NJ, EUA: Prentice-Hall, Inc., 1988. Disponível em: <http://portal.acm.org/citation.cfm?id=46712>. Citado na página 18.

JAROš, M. et al. Implementation of k-means segmentation algorithm on intel xeon phi and gpu: Application in medical imaging. *Advances in Engineering Software*, v. 103, p. 21–28, 2017. ISSN 0965-9978. Special Issue Civil-Comp Parallel, Distributed and Cloud Computing. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0965997816301041>. Citado na página 21.

KARBHARI, S.; ALAWNEH, S. Gpu-based parallel implementation of k-means clustering algorithm for image segmentation. In: *2018 IEEE International Conference on Electro/Information Technology (EIT)*. [S.l.: s.n.], 2018. p. 0052–0057. ISSN 2154-0373. Citado 2 vezes nas páginas 20 e 46.

KHRONOS. *OpenCL Overview*. 2019. Disponível em: <https://www.khronos.org/opencl/>. Acesso em: 14 jan. 2019. Citado na página 20.

MPI. *A High Performance Message Passing Library*. 2019. Disponível em: <https://www.open-mpi.org/>. Acesso em: 21 jan. 2019. Citado na página 22.

NVIDIA. *CUDA FAQ*. 2018. Disponível em: <https://developer.nvidia.com/cuda-faq>. Acesso em: 20 dez. 2018. Citado 2 vezes nas páginas 20 e 27.

NVIDIA. *CUDA C Best Practices Guide*. 2019. Disponível em: <https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html>. Acesso em: 04 jun. 2019. Citado 3 vezes nas páginas 25, 30 e 31.

NVIDIA. *CUDA C Programming Guide*. 2019. Disponível em: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>. Acesso em: 23 mar. 2019. Citado 7 vezes nas páginas 23, 24, 25, 26, 27, 31 e 33.

OPENMP. *OpenMP FAQ*. 2019. Disponível em: <https://www.openmp.org/about/openmp-faq/>. Acesso em: 19 jan. 2019. Citado na página 21.

SARDAR, T. H.; ANSARI, Z. An analysis of mapreduce efficiency in document clustering using parallel k-means algorithm. *Future Computing and Informatics Journal*, v. 3, n. 2, p. 200–209, 2018. ISSN 2314-7288. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2314728817300661>. Citado na página 22.

SHIRKHORSHIDI, A. S. et al. Big data clustering: A review. In: MURGANTE, B. et al. (Ed.). *Computational Science and Its Applications - ICCSA 2014*. Cham: Springer International Publishing, 2014. p. 707–720. ISBN 978-3-319-09156-3. Citado 2 vezes nas páginas 14 e 15.

SILVA, G. R. L. et al. Cuda-based parallelization of power iteration clustering for large datasets. *IEEE Access*, v. 5, p. 27263–27271, 2017. ISSN 2169-3536. Citado na página 15.

SIROTKOVIć, J.; DUJMIć, H.; PAPIć, V. K-means image segmentation on massively parallel gpu architecture. In: *2012 Proceedings of the 35th International Convention MIPRO*. [S.l.: s.n.], 2012. p. 489–494. Citado 3 vezes nas páginas 20, 26 e 32.

STALLINGS, W. *Operating Systems: Internals and Design Principles*. 9. ed. EUA: Pearson IT Certification, 2018. ISBN 9352866711, 9789352866717. Citado 3 vezes nas páginas 23, 30 e 34.

TORTI, E. et al. Parallel k-means clustering for brain cancer detection using hyperspectral images. *Electronics*, v. 7, p. 283, 10 2018. Citado na página 21.

WU, J.; HONG, B. An efficient k-means algorithm on cuda. In: *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. [S.l.: s.n.], 2011. p. 1740–1749. ISSN 1530-2075. Citado 4 vezes nas páginas 15, 18, 20 e 46.

YANG, L. et al. High performance data clustering: A comparative analysis of performance for gpu, rasc, mpi, and openmp implementations. *J. Supercomput.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 70, n. 1, p. 284–300, out. 2014. ISSN 0920-8542. Disponível em: <http://dx.doi.org/10.1007/s11227-013-0906-y>. Citado 3 vezes nas páginas 18, 21 e 22.

YOUTUBE. *YouTube para a imprensa*. 2019. Disponível em: <https://www.youtube.com/intl/pt-BR/yt/about/press/>. Acesso em: 6 mar. 2019. Citado na página 14.

ZECHNER, M.; GRANITZER, M. Accelerating k-means on the graphics processor via cuda. In: *2009 First International Conference on Intensive Applications and Services*. [S.l.: s.n.], 2009. p. 7–15. Citado na página 32.

ZHONG, S. et al. The expansibility research of k-means algorithm under the gpu. In: *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. [S.l.: s.n.], 2016. p. 734–737. ISSN 2327-0594. Citado na página 20.

ZHU, H. et al. Parallel fast global k-means algorithm for synthetic aperture radar image change detection using opencl. In: *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. [S.l.: s.n.], 2015. p. 322–325. ISSN 2153-6996. Citado na página 21.

Apêndices

APÊNDICE A – Cronograma

Quadro 2 – Cronograma do Trabalho de Conclusão de Curso (TCC).

	2018			**2019**								
Atividade / Período	**Out**	**Nov**	**Dez**	**Jan**	**Fev**	**Mar**	**Abr**	**Mai**	**Jun**	**Jul**	**Ago**	**Set**
Escolha do tema	X											
Revisão bibliográfica		X	X	X	X							
Elaboração do projeto				X	X	X						
Entrega do projeto						X						
Implementação							X	X	X			
Coleta de dados									X			
Análise de resultados									X	X		
Elaboração da monografia								X	X	X	X	
Entrega da monografia												X
Defesa da monografia												X

Fonte: o autor (2019).