UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MATEUS HIRAMATSU FIORI

UTILIZAÇÃO DE COMPUTAÇÃO PARALELA PARA OTIMIZAÇÃO DO ALGORITMO DE CLUSTERIZAÇÃO DIANA

BAURU
2017

MATEUS HIRAMATSU FIORI

UTILIZAÇÃO DE COMPUTAÇÃO PARALELA PARA OTIMIZAÇÃO DO ALGORITMO DE CLUSTERIZAÇÃO DIANA

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Profa. Dra. Roberta Spolon

BAURU
2017

Mateus Hiramatsu Fiori UTILIZAÇÃO DE COMPUTAÇÃO PARALELA PARA OTIMIZAÇÃO DO ALGORITMO DE CLUSTERIZAÇÃO DIANA/ Mateus Hiramatsu Fiori. – Bauru, 2017- 52 p. : il. (algumas color.) ; 30 cm.
Orientador: Profa. Dra. Roberta Spolon
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho" Faculdade de Ciências
Ciência da Computação, 2017.
1. Tags 2. Para 3. A 4. Ficha 5. Catalográfica

Mateus Hiramatsu Fiori

UTILIZAÇÃO DE COMPUTAÇÃO PARALELA PARA OTIMIZAÇÃO DO ALGORITMO DE CLUSTERIZAÇÃO DIANA

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Orientadora: Profa. Dra. Roberta Spolon
Departamento de Computação
Faculdade de Ciências
UNESP - BAURU

Profª. Drª. Simone das Graças Domingues Prado
Departamento de Computação
Faculdade de Ciências
UNESP - BAURU

Profº. Drº. Kelton Augusto Pontara da Costa
Departamento de Computação
Faculdade de Ciências
UNESP - BAURU

Bauru, treze de dezembro de 2017.

Dedico este trabalho, primeiramente, aos meus pais, por todo amor, apoio e confiança depositados. E aos meus amigos, por toda parceria e motivação.

Agradecimentos

Agradeço aos meus pais por proporcionarem toda a estrutura e base necessária para a realização deste trabalho. Agradeço também a Deus por todos os momentos reflexão e confortos psicológicos proporcionados.

However vast the darkness, we must supply our own light.
Stanley Kubrick

Resumo

É inegável o constante crescimento da, já enorme, quantidade de dados que são produzidos pela população mundial hoje em dia. Tais dados são comumente denominados Big Data.

De acordo com as tecnologias e técnicas existem na atualidade, é possível extrair, processar e analisar determinados conjuntos de dados a fim de se obter informações relevantes sobres os mesmos. Tais informação são importantes para os ramos, por exemplo, de marketing e negócios, entre outros.

Então para extrair, manipular e analisar esses dados são utilizadas técnicas de mineração de dados, ou sob uma perspectiva mais generalizada, técnicas de aprendizado de máquina. Como dito anteriormente a quantidade de dados para se obter conhecimentos relevantes a partir destes é muito grande, acarretando numa necessidade de processamento computacional proporcional, portanto o tempo de execução de aplicações, que se utlizem dessas técnicas, tende a aumentar proporcionalmente em relação à quantidade de dados, tudo isso sob uma perspectiva de implementação serial.

A computação paralela então, se apresenta como uma alternativa para otimizar os tempos de execução desses algoritmos, com o intuito de se obter os resultados em um tempo mais baixo, e consequentemente mais viável.

O presente trabalho analisou os tempos de execução de dois método específicos utilizados no algoritmo de clusterização de dados DIANA, algoritmo este pertencente às técnicas não supervisionadas de aprendizado de máquina, tanto com relação à implementação serial quanto a paralela, para tentar explicitar um speed up possível speed up, i.e uma melhora nos tempos de execução, pela utilização de computação paralela.

Palavras-chave: CUDA, DIANA, Computação paralela, Nvidia

Abstract

It is undeniable the constant growth of the already huge amount of data that produced by the population worldwide nowadays. Such data are commonly named Big Data.

According to the techniques and tecnologies that exist nowadays, it's possible to extract, process and analyze datasets in order to obtain relevant information about it. Such information is important to the fields of, for instance, marketing and business, among others.

So to extract, manipulate and analyze these data, it is used data mining techninques, or under a more general perspective, machine learning techniques. As said before, the amount of data to obtain relevant knowledge through these data is too big, resulting in a proportional computational processing need, thus the application execution time, that uses these techniques, tend to increase proportionally regarding the amount of data, all of this under a serial deployment perspective.

So parallel computing presents itself as an alternative to optimize these algorithm's execution times, in order to obtain the results in a lower time, and consequently more viable.

The present work analyzed the execution times of two specific methods used in the DIANA clusterization algorithm that belongs to the machine learning's unsupervised techniques, in both serial and parallel deployment, to try to show a possible speed up, i.e an improvement in the execution time, by the use of parallel computing.

Keywords: CUDA, DIANA, Parallel computing, Nvidia

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CPU	Central Process Unit - Unidade de Processamento Central
GPU	Graphical Process Unit - Unidade de Processamento Gráfico
KDD	Knoledge-Discovery in Database - Descoberta de Conhecimento em Base de Dados
RAM	Random Access Memory - Memória de Acesso Randômico
PCI	Peripheral Component Interconect - Interconexão de Componente Periféricos
ULA	Unidade Lógica Aritmética
SDK	Software Develpment Kit - Kit de Desenvolvimento de Software

Sumário

1 Introdução

É sabido que nas últimas décadas o volume de dados gerado na internet pelos seus usuários vem aumentando substancialmente (MAGOULAS; LORICA, 2009). É interessante que esses dados sejam utilizados de maneira relevante, porém é inviável que tais dados sejam analisados por um especialista, dando lugar para pesquisas e estudos relacionados à Mineração de Dados (Data Mining) e *Big Data*.

Big Data é um termo que representa a grande quantia de dados e informações, estruturadas ou não, existentes em determinada base de dados, podendo ser analisadas para obtenção de conhecimento. (SIVARAJAH et al., 2017).

Tal conhecimento é consequência da Mineração de Dados que pode ser definida como o "processo de descobrir padrões em dados. O padrão descoberto deve ser significativo na medida em que levar a alguma vantagem, geralmente econômica." (WITTEN; FRANK; HALL, 2011). Para que isso seja possível é necessário que estes sejam classificados e agrupados em *clusters* de acordo com suas características. Então são utilizados algoritmos de clusterização, que é uma classificação não-supervisionada da análise de dados. O grande desafio neste tipo de classificação é não ter as classes pré-definidas (JAMES et al., 2014) i.e os dados são agrupados em *clusters* de acordo com características intrínsecas, de diversas maneiras dependendo do método de análise de *clusters* utilizado. A análise de *clusters* é a etapa responsável por dividir os dados em grupos (*clusters*).

Outro fato interessante ressaltar é a necessidade de otimização desses algoritmos que são utilizados no processo de mineração de dados, pois o processo de treinamento destes é demorado, visto que essa atividade pode precisar de várias iterações. (FRANCA, 2015).

O enfoque do trabalho é otimizar dois métodos de análise de *clusters* presentes em algoritmos de clusterização hierárquicos divisivos, *average linkage* e *centroid linkage*, através do uso da programação paralela e compará-los com suas respectivas implementações seriais e entre eles, medindo seus tempos de execução.

A programação paralela é um outro assunto emergente atualmente, necessário para acompanhar as demandas da tecnologia com relação aos processamentos de grandes quantias de dados (NVIDIA, 2017), visto que os processadores atuais tendem a uma constância relacionada à sua capacidade de processamento (MOORE, 1966). Existem muitas pesquisas relacionadas à agrupamento de dados, assim como diversos métodos e algoritmos, porém esses números caem consideravelmente quando combina-se este assunto com a utilização de computação paralela.

Foram implementados os métodos de análise de dados já citados, com o uso da programação paralela, utilizando-se da biblioteca CUDA C desenvolvida pela NVIDIA, para paralelizar o algoritmo, com o hardware GeForce GTX 1060 utilizado para o processamento paralelo. Foi medido e comparado o tempo de execução de cada método de análise de *cluster* em sua implementação serial e paralela.

Tal medida é necessária pois algoritmos de clusterização normalmente trabalham com grandes quantidades de dados e possuem um grande número de iterações na etapa de treinamento, portanto demoram para serem processados.

É necessária a otimização de algoritmos de clusterização pois normalmente são manipulados grandes quantias de dados e portanto o tempo de execução e processamento é muito grande. A diminuição desse tempo de execução é interessante para softwares de mineração de dados e consequentemente para as empresas que o utilizem.

A motivação para este trabalho está no fato de se empregar a computação paralela como proposta para otimizar e melhorar o desempenhos dos algoritmos utilizados, e assim obter resultados em um tempo cada vez mais hábil.

2 Objetivos

2.1 Objetivo geral

Otimizar dois métodos de análise de *clusters* presentes em algoritmos de clusterização hierárquicos divisivos utilizando computação paralela, com o intuito de tentar diminuir o tempo de execução deste mesmo.

2.2 Objetivo específico

Implementar e otimizar através do uso da programação paralela os métodos: average linkage e centroid linkage que dividem grupos de dados de acordo com suas características intrínsecas relacionados à mineração de dados.

3 Tecnologias aplicadas

3.1 *Big Data*

Hoje a velocidade com que as transações sociais e econômicas acontecem *online* permitem a captura digital do *Big Data*. A quantidade de dados gerados e compartilhados pelos setores de negócio, científico, administrativo e industrial cresceram exponencialmente nos últimos tempos (AGARWAL; DHAR, 2014). Esses dados podem ser textuais (estruturados, semi-estruturados ou não-estruturados), assim como dados multimídias (áudio, vídeo ou fotos) gerados por uma multiplicidade de plataformas, como por exemplo *sites* de mídia social, redes de sensores, comunicação máquina-à-máquina (SIVARAJAH et al., 2017). Estima-se que, por dia, o mundo produz 2,5 quintilhões de bytes de dados (2,5 exabytes = 2,5 bilhões de gigabytes) (DOBRE; XHAFA, 2014).

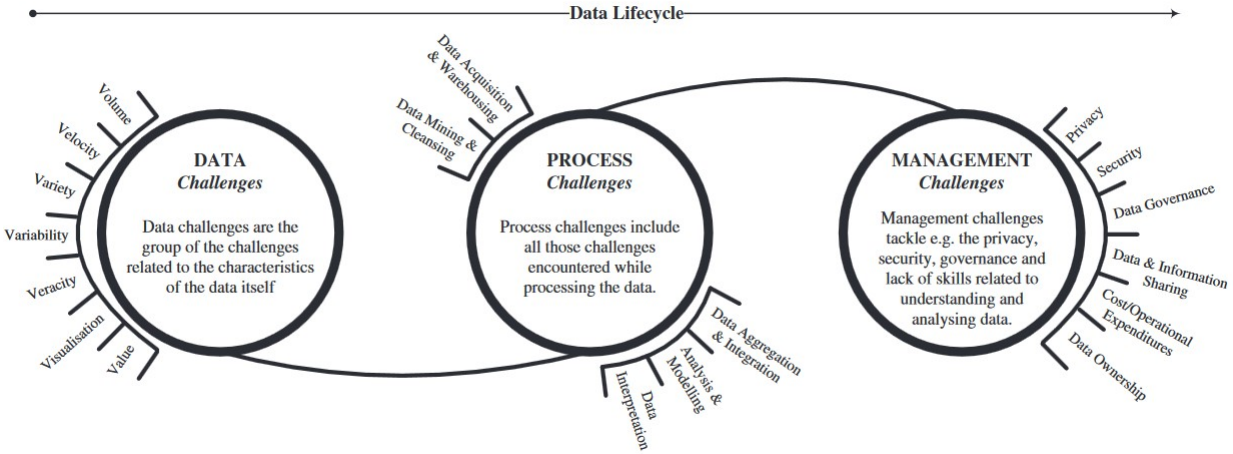
Mas esses dados não possuem valor se não forem analisados apropriadamente. Existem técnicas de análise dados datadas do século XVIII, e elas podem ser utilizadas para retirar conhecimento de grupos de dados (AGARWAL; DHAR, 2014), porém de acordo com a magnitude dos conjuntos de dados gerados hoje em dia, técnicas tradicionais não costumam ser precisas (MAGOULAS; LORICA, 2009). Um processamento e gerenciamento de dados apropriado pode reverter essa situação, podendo assim expor novos padrões, influenciando assim nas tomadas de decisões e facilitar a resposta para desafios e oportunidades emergentes num período de tempo hábil (SIVARAJAH et al., 2017). Tendo como lado positivo a vantagem competitiva gerada no mundo empresarial, algo que está diretamente relacionado com a rapidez que se obtém os conjuntos de dados e a possibilidade de implementação de sistemas que consigam responder a esses dados (MAGOULAS; LORICA, 2009).

Em contra partida, é interessante mencionar que o crescimento do volume no mundo digital não acompanha o avanço das tecnologias relacionadas à infraestrutura. Tecnologias bem estabelecidas no mundo da computação, como por exemplo, banco de dados e *data warehouse*s, ou em português, repositório de dados, estão se tornando inadequadas para a quantidade de dados que são gerados no mundo todos os dias. Uma das soluções para tal problema é a utilização de banco de dados avançados, específicos para a manipulação de *Big Data* (e.g. NoSQL Databases, BigQuery, MapReduce, Hadoop, WibiData e Skytree) (SIVARAJAH et al., 2017).

Para ilustrar tem-se alguns exemplos de experimentos de larga escala da esfera social feitos com *Big Data*. Pesquisadores do Facebook conduziram um estudo para determinar se o humor dos usuários da rede social poderia ser manipulado e descobriram

que isso é possível (não levando em consideração o impacto ético sobre o uso de dados consenso daqueles o geram) (KRAMER; GUILLORY; HANCOCK, 2014). O Google consegue detectar surtos de gripe de sete a dez dias mais rápido que os Centros de Controle e Prevenção de Doenças, apenas monitorando o aumento de buscas com frases associadas com sistemas de gripe [M. Helft, "Google Uses Searches to Track Flu's Spread," New York Times, 11/11/2008

Figura 1: Classificação conceitual dos desafios do *Big Data*



Fonte: (SIVARAJAH et al., 2017)

Tradução: Data challenge são os grupos de desafios relacionados às caracteristicas do próprio dado; Process Challenge desafios de processos incluem todos os desafios encontrados no processamento; Management Challenge são os desafios de gerenciamento relacionados com a privacidade, segurança e governança relacionados ao entendimento e análise dos dados.

Como mostrado na Figura 1, existem 3 fases de desafios relacionados com as características da *Big Data* em questão. A primeira fase é a de exposição dos grupos de desafios que estão relacionados com as características dos dados analisados, i.e desafios que têm como objetivo analisar e assegurar a velocidade, a veracidade, a variedade, o volume, entre outras características do *dataset* em questão. A segunda é a fase de processamento dos dados analisados, é a fase cujo presente trabalho está inserido, tendo como exemplo a fase de mineração dos dados, análise, modelagem e interpretação dos mesmos, e a terceira fase trata do gerenciamento dos dados em questão, mais especificamente tratando-se da parte auditorial dos mesmos, i.e desafios relacionados com a privacidade e segurança do conjunto de dados (SIVARAJAH et al., 2017).

O presente trabalho visa analisar grandes grupos de dados e separá-los em *clusters* (grupos), de acordo com suas características, para então extrair conhecimentos relevantes do grupo de dados em questão.

3.2 Mineração de Dados

Existem muitos nomes que definem o termo *Data Mining*, ou Mineração de Dados. Um dos sinônimos mais utilizados é o KDD (*Knowledge Discovery Database*), como o próprio nome diz, a descoberta de conhecimento em determinada base de dados. Este mesmo termo em outras literaturas é representado como apenas uma parte da Mineração de Dados em si (HAN; PEI; KAMBER, 2011).

Extração de conhecimento de determinado banco de dados é o processo que identifica padrões dos dados que o compõe. Para que isso seja feito são utilizadas diversas técnicas de diversas áreas do conhecimento, tais como estatística e matemática, não se restringindo apenas à computação (HAN; PEI; KAMBER, 2011).

Existem alguns termos específicos da Mineração de dados que se faz necessária a explicação para um melhor entendimento do trabalho.

Padrões, são abstrações do conjunto de dados, podendo ser referenciado como pontos de um conjunto de dados.

Processo, é a busca pelos padrões e a avaliação dos mesmos, sendo este dividido em várias etapas.

Informação, são os dados que já passaram por um processo e são compreensíveis, normalmente é explicitada em gráficos.

Conhecimento, são as informações que foram analisadas, sendo estas confiáveis e relevantes. É o resultado da análise dos responsáveis pela tomada de decisões com o objetivo de buscar uma melhor compreensão sobre o problema.

Inteligência, é o conhecimento que foi construído e aplicado a uma determinada situação para entendê-la melhor. (FROZZA, 2013)

Dito isso, a mineração de dados atua especificamente nas bases de dados em busca de padrões para que o conhecimento seja gerado.

A primeira etapa é a de limpeza dos dados (*data cleaning*), é nesta etapa que serão removidos ruído e dados inconsistentes. Isso é feito através de rotinas de limpeza de dados que preenchem valores vazios, e suavizam ruídos no conjunto de dados, tornando assim todo o conjunto mais consistente.

A segunda etapa é a de integração dos dados (*data integration*), é nesta etapa que todas as fontes de dados são agrupadas de maneira padronizada e inteligível. Algumas literaturas consideram a primeira e a segunda etapa como um passo de pré-processamento, onde o dataset gerado será armazenado em uma *data warehouse*, ou depósito de dados digitais.

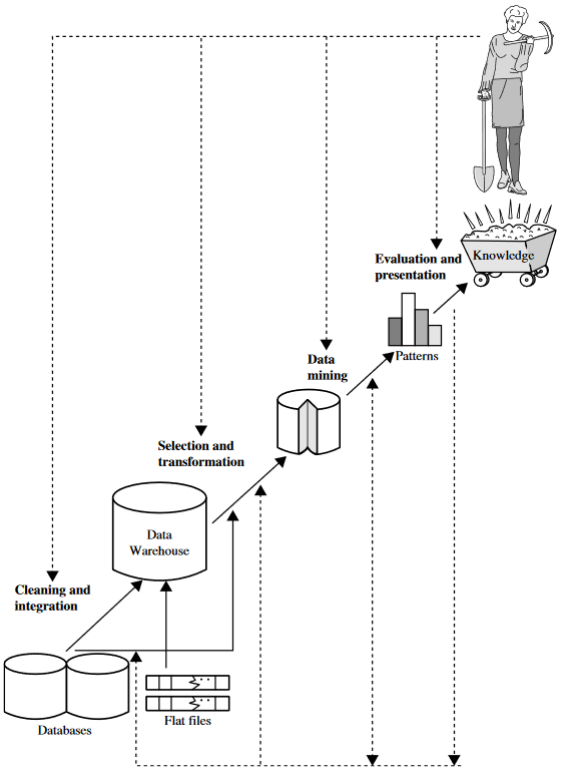
A terceira etapa é a de seleção dos dados (*data selection*), onde serão levantados

apenas os dados que serão relevantes ao serem analisados. A quarta etapa é a de transformação dos dados (*data transformation*), onde os dados provenientes da etapa anterior são consolidados e modificados em um formato apropriado para a mineração.

A quinta etapa que é chamada de mineração de dados (*data mining*). Esta é uma etapa essencial, é nela que são aplicados métodos inteligentes para a extração de padrões, sendo assim a possível utilização de técnicas de aprendizado de máquina.

A sexta e a sétima etapa estão respectivamente relacionadas com a parte de avaliação dos padrões encontrados, i.e a identificação dos padrões verdadeiramente relevantes, e com parte da visualização do conhecimento adquirido, utilizando-se de técnicas específicas para a representação do conhecimento ao usuário (HAN; PEI; KAMBER, 2011)

Figura 2: Passos do processo de extração de conhecimento de uma base de dados



Fonte: (HAN; PEI; KAMBER, 2011)

Na figura 2 estão descritos os passos da mineração de dados, sendo eles, de baixo para cima: limpeza e integração, seleção e transformação, mineração de dados e avaliação e apresentação.

3.2.1 Etapas da mineração de dados

A fase de mineração de dados pode ser então dividida em 6 passos distintos mais importantes, sendo estes: Classificação, Regressão, Regras de Associação, Agrupamento,

Estimativa e Desvio; sendo estes passos correlacionados com algumas técnicas de aprendizado de máquina, como por exemplo a utilização de redes neurais para a classificação dos dados pelo uso de técnicas de aprendizado não-supervisionado com algum algoritmo de clusterização.

Classificação. É nessa fase que algoritmos de clusterização são utilizados, é nesta fase que os dados são agrupados de acordo com suas características em comum. Classificação é também dita como função de aprendizado.

O objetivo dos algoritmos de clusterização é encontrar algum atributo em comum entre os dados e as classes (ou *clusters*), de modo que este processo possa predizer a classe de um dado novo, desconhecido e portanto ainda não *cluster*izado.

É possível ainda dividir a classificação em 2 tipos: a classificação/aprendizado supervisionada e não-supervisionada. Na classificação supervisionada as classes são definidas explicitamente, i.e já existem rótulos para as classes. No aprendizado não-supervisionado os *clusters* são criados a partir das características dos próprios dados, de acordo com suas similaridades (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012).

Exemplos práticos de classificação de dados são: divisão e-mails spam e não-spam e identificação da forma de tratamento mais adequada para determinado paciente.

A fase de regressão está fortemente atrelada com a estatística e tem conceitualmente a mesma função da fase de classificação, tendo como única diferença o tipo do atributo analisado, ao invés de discreto o atributo é contínuo.

O objetivo da regressão é encontrar em um conjunto de dados X, sendo X = x1, x2, . . . , xn uma função y que os mapeie e os represente geometricamente (y = x1,x2, . . . , xn (ZYTKOW; RAUCH, 2004).

Posteriormente existe é feita a fase de regras de associação. O objetivo principal dessa etapa é identificar tendências que possam posteriormente serem usadas para entender e explorar os padrões e comportamentos dos dados. Isso é muito utilizado em cadeias de varejo para a distribuição de produtos em suas prateleiras. Um exemplo de regra de associação é, ao se observar os dados de venda de um determinado supermercado, verifica-se que 90% dos clientes que comprar o produto A, também compram o produto B, portanto tem-se uma regra com 90% de confiabilidade do produto B.

A etapa de agrupamento também pode ser vista como um complemento da fase de classificação, no sentido de particionar o um grupo de dados muito heterogêneo em subgrupos mais homogêneos. Um exemplo de aplicação é em determinada segmentação de mercado, pode-se dividir os clientes em grupo que tenham comportamento similar, ou que pertençam à um mesmo país, antes de se aplicar a classificação.

A etapa de estimativa é utilizada para o preenchimento de algum atributo desco-

nhecido de determinado dado, utilizando uma aproximação baseada no padrão de recorrência do atributo em questão, de acordo com os demais dados.

"Estimativa é aprender uma função que mapeia um item dado para uma variável de predição real estimada" (FREITAS; LAVINGTON, 2012)

Como exemplo de tarefas de estimativa tem-se: estimar o número de filhos em uma família, ou estimar a probabilidade de um paciente morrer baseando-se num conjunto de resultados de diagnósticos médicos.

Tem-se também a etapa de desvio, com o objetivo de descobrir um conjunto de valores que não seguem os padrões que foram encontrados nas etapas anteriores. Essa etapa pode ser utilizada para identificação de fraudes em elementos que estão fora do padrão, ou que estão muito longe da curva de aproximação traçada na etapa de análise dos dados ou regressão (HAN; PEI; KAMBER, 2011).

3.2.2 Aplicações da mineração de dados

A mineração de dados tem dado muitos resultados positivos em diversas áreas, sendo assim impossível nomeá-las todas. Mas é possível citar duas grandes áreas nas quais os resultados são críticos: inteligência de negócios e motores de busca na web.

A primeira delas sendo a área de *Business Intelligence*, ou inteligência de negócios. É indispensável para o mundo dos negócios a aquisição de informações que ajudem a se obter um melhor entendimento do contexto comercial de alguma organização em questão, e.g seus clientes, o mercado em si, suprimentos, recursos e competidores.

O *Business Intelligence* fornece informações preditivas, históricas, ou atuais de operações de negócio. Tendo como exemplo o processamento analítico online, gerenciamento em performance de negócio e análise preditiva.

Claramente, a mineração de dados é uma das partes mais importantes para que isso seja possível. Sem esta não seria possível a realização de uma análise de mercado efetiva, i.e o descobrimento das forças e das dores de competidores de mercado e decisões inteligentes relacionadas a estratégia de mercado. Mais especificamente a clusterização de dados ocupa um papel muito importante no gerenciamento de relações com o cliente. que agrupa tais clientes em *clusters* baseando-se em suas similaridades, com isso é possível entender melhor as características específicas de cada segmento de clientes, e assim desenvolver estratégias cada vez mais customizáveis.

A segunda são os motores de busca na Web, que nada mais são do que servidores especializados em buscas na internet. (e.g Google). São essencialmente aplicações de mineração de dados muito grandes, assim muitas técnicas de mineração de dados são aplicadas em todos os aspectos do motor de busca, desde decidir quais páginas que deverão ser

analisadas, até como os resultados retornados podem ser personalizados de acordo com o contexto (FROZZA, 2013).

Existem grandes desafios relacionados com motores de busca para a mineração de dados. É necessário a manipulação de quantidade grande de dados e que não para de crescer, portanto são necessárias técnicas de otimização e a utilização de múltiplos computadores. Outro fator de desafio é fato desses motores terem de lidar com dados online, pois apesar dos modelos que são utilizados terem sido treinados offline, quando onlIine estes precisam retornar uma resposta em um tempo real viável. E um último fator de desafio é a manutenção e a atualização incremental do modelo que é alimentado com fluxos dados que estão em constante crescimento (FROZZA, 2013).

3.3 Computação Paralela e computação heterogênea

Antes de começar a explicação do que realmente é o CUDA, é necessário estabelecer alguns conceitos básicos que são pré-requisitos para a existência do CUDA.

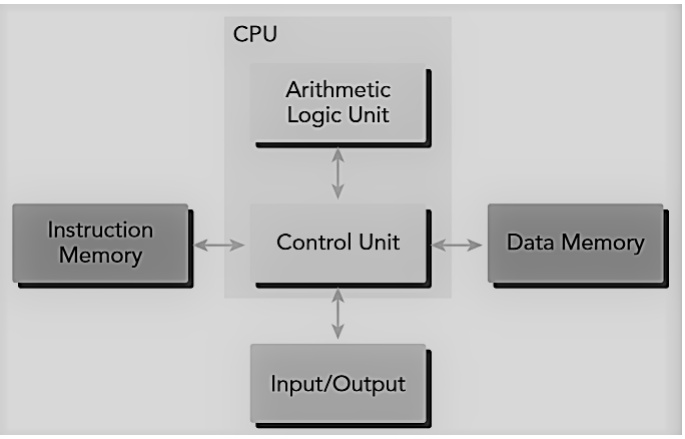
O primeiro deles é a computação paralela, tendo esta como primeiro objetivo o aumento da velocidade de computação. Computação pode ser definida como uma forma de computação na qual muitos cálculos acontecem simultaneamente, operando a partir do princípio que grandes problemas podem ser divididos em muitos pequenos problemas, que então são solucionados concorrentemente.

Normalmente isso envolve duas áreas distintas da computação: a arquitetura de computação (no âmbito dos hardwares) e programação paralela (no âmbito do software). A arquitetura de computadores foca em dar suporte ao paralelismo em um nível arquitetural, enquanto a programação paralela se foca em solucionar o problema concorrentemente, utlizando-se do total poder computacional da arquitetura (COOK, 2013).

Atualmente a maioria dos computadores utilizam a arquitetura mostrada na Figura 3.

O componente principal para a alta performance de computação é a CPU (Central Processing Unit), ou núcleo. Atualmente existem computadores com mais de um núcleo que suportam paralelismo em nível arquitetural. Porém dependendo do problema que será analisado uma maior quantidade de núcleos podem ser necessários, para que o paralelismo realmente aumente o poder computacional, nesse caso são utlizadas as GPUs (Graphical Processing Unit) como complemento. GPUs representam uma arquitetura de muitos núcleos). GPUs e CPUs não compartilham de um ancestral em comum. Historicamente GPUs são acelerados gráficos, muito utilizada para jogos ou renderização de vídeos. Apenas recentemente que as GPUs começaram a ser utilizadas para um propósito mais geral, sendo totalmente programáveis. Em suma, núcleos de CPU são desenvolvidos para

Figura 3: Arquitetura de Von Neumann

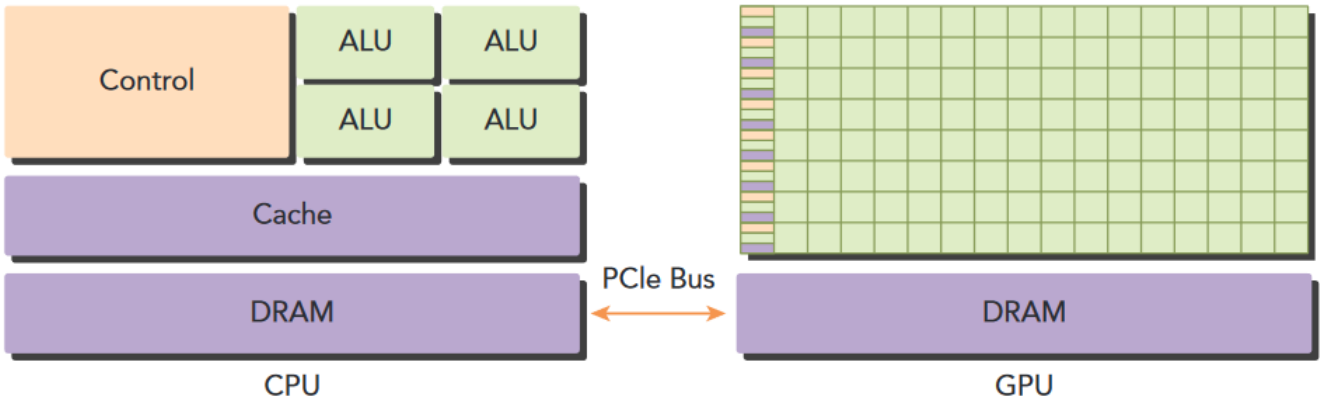


Fonte: (COOK, 2013)

lógicas de controle complexas, buscando otimizar a execução de programas sequenciais. Enquanto o núcleo da GPU é otimizado para tarefas com dados paralelos com uma lógica de controle mais simples.

A CPU e a GPU trabalham de maneira complementar, este tipo de computação é denominada computação heterogênea, como mostrado na figura 4.

Figura 4: Arquitetura Heterogênea



Fonte: (COOK, 2013)

Na figura 4, são explcitadas as diferentes arquiteturas da CPU e da GPU e sua comunicação através de um barramento PCIe. Ambas possuem memórias DRAM e memórias cache que são do tipo voláteis e são utilizadas no processamento. Ambas também possuem unidades de controle (Control), onde na CPU sua proporção é maior indicando uma abstração mais generalizada para o processamento utilizando relativamente poucas unidades lógicas aritméticas (ALU - *arithmetic logic unit*), enquanto na GPU existem diversas unidades de controle e diversas ULAs, indicando um processamento mais específico para cada processado nas unidades lógicas.

Portanto uma aplicação heterogênea consiste em duas partes: a parte em que o código executa na CPU ou no “Host”, e a parte que o código executa na GPU ou no “Device”. O código no host é responsável por gerenciar o ambiente, os dados e a lógica de programação, antes que os dados sejam carregados no device onde tarefas de computação intensiva são executadas.

A GPU utilizada nesse trabalho é uma NVIDIA GeForce GTX 1060, que possui uma arquitetura que permite a utilização do SDK CUDA.

	GeForce GTX 1060
CUDA Cores	1280
Memória	6G
Performance de pico	3.85 Tflops
Banda de memória	3.85 GB/s

Tabela 1: Características da placa de vídeo

Fonte: https://www.pcper.com/reviews/Graphics-Cards/NVIDIA-GeForce-GTX-1060-Preview-Pascal-GP106

Na tabela 1, Cuda Cores é a quantidade de núcleos que existem no acelerador gráfico, performance de pico é a medida da capacidade computacional, definida em quantos cálculos de pontos flutuantes (tanto float quando double) podem ser processados dos segundo, e banda de memória é a medida da razão da quantidade de dados que podem ser lidos ou escritos na memória por segundo.

3.3.1 CUDA

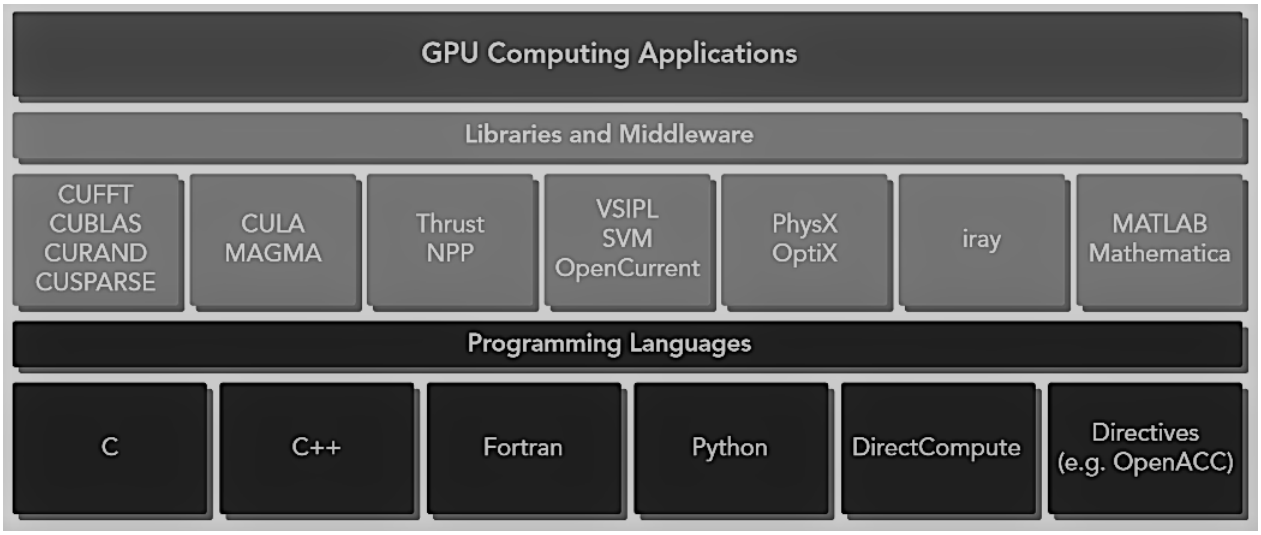
CUDA é uma plataforma de computação paralela com propósitos gerais, e modelos de programação que podem ser utilizados junto a aceleradores gráficos compatíveis, para resolver problemas computacionais complexos de maneira eficiente. Com a utilização do CUDA é possível acessar a GPU para computação, o que tradicionalmente só era feito na CPU.

A plataforma pode ser utilizaca?? através de bibliotecas do CUDA, diretivas do compilador e extensões de linguagens de programação padrão, e.g C, C++, Python. A linguagem escolhida para o desenvolvimento do trabalho é C.

Na figura 5 são apresentadas as linguagem que são suportadas pelo kit de desenvolvimento CUDA, sendo elas C, C++, Python e etc. E as bibliotecas que podem ser utilizadas e invocadas no código fonte sendo elas: MATLAB Mathematica, CULA MAGMA CUFFT e etc. Basicamente essas bibliotecas que conectam as linguagens de programas até a aplicação CUDA paralela.

CUDA C é uma extensão da linguagem padrão ANSI C que permite a computação

Figura 5: Bibliotecas utilizadas pelo CUDA SDK



Fonte: (COOK, 2013)

heterogênea e ainda possibilita o gerenciamento de dispositivos, memória e outras tarefas. CUDA é também escalável possibilitando que os programas escale seu paralelismo à GPU de maneira transparente, variando o número de núcleos, enquanto mantém uma curva de aprendizado suave para programadores familiarizados com a linguagem de programação C.

O CUDA fornece dois níveis de API para o gerenciamento da GPU e organização de threads, sendo estas: CUDA Driver API, e CUDA Runtime API.

O driver API é uma API de mais baixo nível, sendo relativamente mais difícil de se programar, porém fornece mais controle ao programador em como este irá utilizar a GPU. A API de runtime é implementada acima da API de driver, onde cada função da API de runtime é "quebrada" em operações mais básicas da API de driver(COOK, 2013).

A figura 6 ilustra a hierarquia de utilização das bibliotecas do kit de desenvolvimento CUDA em uma determinada aplicação.

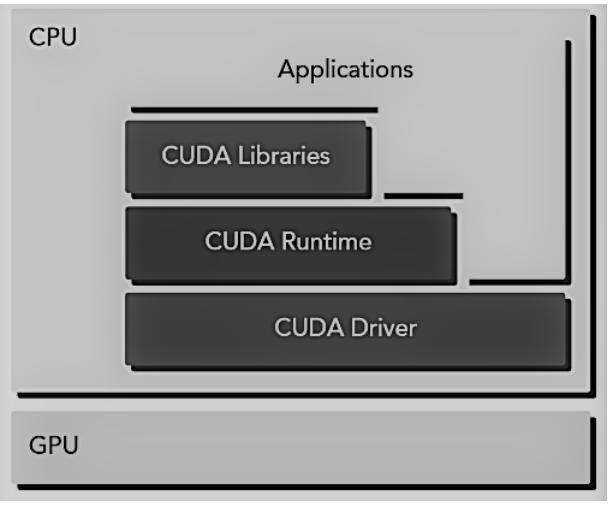
Não existe diferença de desempenho notável entre as duas APIs. Como o problema é organizado produz mais efeito sobre a o desempenho.

Um programa CUDA consiste basicamente de duas partes: o código que será executado na CPU e o código que será executado da GPU.

A figura 7 apresenta um exemplo de código que será executado na CPU e a figura 8 apresenta um exemplo de código que será executado na GPU.

Para que o código seja executado de fato na GPU, é necessária uma chamada da função "__global__' definida, na areá de código que será executada na CPU, como mostrado na figura 9.

Figura 6: Abstração do uso das APIs do CUDA



Fonte: (COOK, 2013)

Figura 7: Código executado na CPU

```
#include <stdio.h>
int main(void)
{
   printf("Hello World from CPU!\n");
}
```

Fonte: Feito pelo autor

Figura 8: Código executado na GPU

```
__global__ void helloFromGPU(void)
{
    printf("Hello World from GPU!\n");
}
```

Fonte: Feito pelo autor

Um componente chave para a programação em CUDA é o kernel (ou núcleo), i.e o código que rodará na GPU. Como um programador, é possível expressar o kernel como um programa sequencial, e o CUDA cuida de gerenciar o que foi escrito pelo programador em termos de organização de thread. A intenção é permitir que o foco seja na lógica da programação tirando a responsabilidade do programador de criar e gerenciar milhares de threads na GPU.

Na parte de gerenciamento de memória a lógica segue a mesma da linguagem C padrão. A figura 10 compara função utilizadas para gerenciamento de memória tanto na linguagem C padrão e C CUDA.

Uma das características do modelo de programação CUDA é a hierarquia de memória e sua abstração para melhor entendimento no momento de se alocar mais memória. Existem dois tipos principais de memória na GPU, são esses memória global e memória

Figura 9: Chamada da função que será executada na GPU

```
helloFromGPU <<<1,10>>>();
```

Fonte: Feito pelo autor

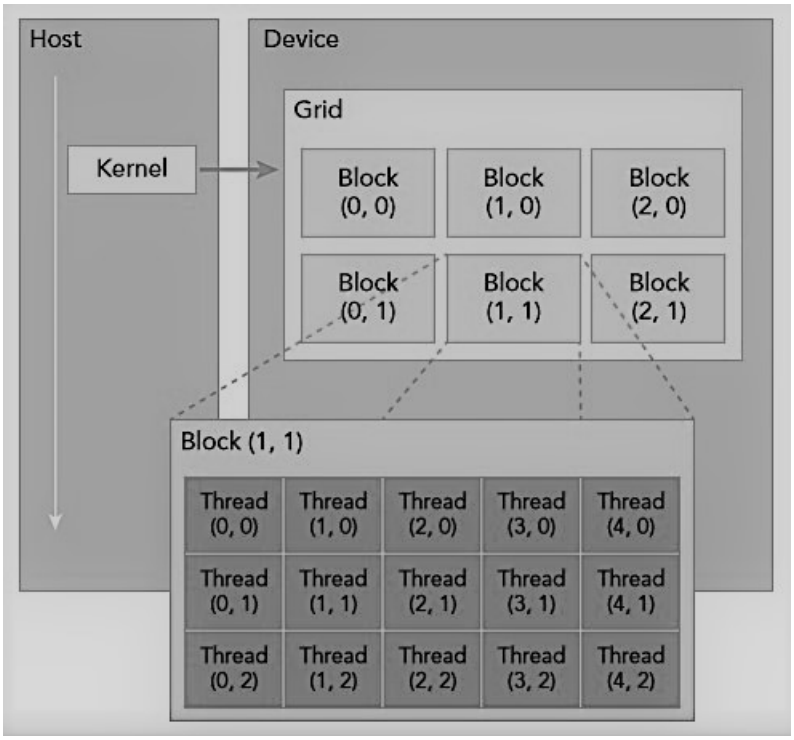
Figura 10: Comparação de funções C e CUDA C

Funções C padrão	Funções CUDA C
malloc	cudaMalloc
memcpy	cudaMemcpy
memset	cudaMemset
free	cudaFree

Fonte: Feito pelo autor

compartilhada. A memória global é equivalente à memória RAM da CPU, enquanto a memória compartilhada é equivalente à memória Cache da CPU, com a diferença que a memória compartilhada pode ser diretamente controlada no kernel do CUDA C (COOK, 2013).

Figura 11: Abstração da memória na GPU



Fonte: (COOK, 2013)

A figura 11 ilustra uma abstração hierárquica de como a memória na GPU deve ser tratada, afim de facilitar o trabalho do desenvolvedor.

4 Algoritmos de clusterização

Para um melhor entendimento sobre qual algoritmo será utilizado, é necessário estabelecer alguns conceitos. A mais importante definição é a de *cluster*. Um *cluster* nada mais é do que um grupo ou conjunto de dados, no qual os dados pertencentes a ele apresentam características com valores próximos (ou similares).

Então um *cluster* agrupa objetos de dados baseando-se apenas nas informações encontradas no dado e em suas relações. Quanto melhor a diferença entre os *clusters*, melhor e mais distinta é a clusterização (TAN; STEINBACH; KUMAR, 2005).

Assim, algoritmos de clusterização em geral, têm como objetivo analisar determinado conjunto de dados e agrupar estes dados em um número k de *clusters* de acordo com suas características.

É importante ressaltar também que algoritmos de clusterização são muito utilizados no ramo do aprendizado de máquina, mais especificamente fazendo parte do aprendizado não-supervisionado, se contrapondo ao aprendizado supervisionado, onde cada dado é também classificado como sendo parte de algum grupo, porém com a diferença de que previamente são definidos rótulos para a identificação destes grupos e consequentemente de cada dado.

Na fase de treinamento então, o input é exclusivamente composto de dados não rotulados. Em geral é difícil avaliar quantitativamente a performance do algoritmo (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012).

4.1 Algoritmo DIANA

Para um melhor entendimento do algoritmo, é necessário definir o termo dissimilaridade. Dissimilaridade está relacionada com distância. Todos os dados possuem um lugar específico em um determinado plano, de acordo com suas características. Por exemplo, um determinado dado D com características (x1 = 2 e x2 = 3), está localizado num plano cartesiano com eixos x1 e x2, na posição (2, 3). Portanto é possível calcular as distâncias entre os dados, utilizando por exemplo, distância Euclidiana. E quando uma determinada distância é máxima, diz-se que os dados em questão são dissimilares (KAUFMAN; ROUSSEEUW, 1990).

O algoritmo DIANA foi primeiramente implementado por Kauffman & Rousseeuw em 1990. A abordagem deste algoritmo é ser hierárquico e divisivo, portanto tem-se inicialmente um grupo de dados, e conforme as iterações passam tais dados são divididos em outros grupos de acordo com suas dissimilaridades, até que o número k de grupos seja

atingido, ou cada dado seja único no grupo (KAUFMAN; ROUSSEEUW, 1990).

Considerando então um grupo G contendo todos os dados do conjunto de dados, tem-se como objetivo dividí-lo em dois grupos de tal maneira que a dissimilaridade entre esses grupos seja máxima.

Num primeiro momento o algoritmo busca em G o dado (ou padrão) que é o mais dissimilar, então este dado é retirado de G e posto em um grupo temporário tempG. Em seguida, para cada dado x pertencente a G é calculado a média de dissimilaridade com relação a todos os dados pertencentes ao grupo G e também é calculada a média de dissimilaridade de x com relação a todos os elementos do grupo tempG. Então se o dado x está mais perto do grupo G, ele permanece no grupo, caso contrário ele é posto no grupo tempG. O procedimento é repetido até que cada dado seja único em um grupo ou se um critério de parada foi satisfeito (KAUFMAN; ROUSSEEUW, 1990).

A figura 12 descreve o pseudo-código do algoritmo DIANA.

Figura 12: Pseudo-código do algoritmo DIANA

```
procedure DIANA (X, MaxGrupos)
Input: X = {P1, P2,..., PN}        %N padrões a serem agrupados
       MaxGrupos                   %Quantidade máxima de grupos a serem criados
Output: AGt
1. begin
2. t ← 1
3. AG1 ← {X}                       % agrupamento inicial
4. G ← X
5. repeat
6.   maxDiss ← maiorDissimilaridade(G)   % encontrar elemento mais dissimilar
7.   G ← G – {maxDiss}
8.   tempG ←{maxDiss}
9.   repeat
10.    Para cada padrão x pertencente a G encontre:
12.      D(x) = (média diss(x,y), x ∈ G e x≠y] – (média diss(x,z) z ∈ tempG)
11.      Encontre o padrão x para qual a diferença D(x) é a maior.
12.      if D(x) > 0 then
13.         begin
14.           tempG ← tempG ∪ {x}
15.           G ← (G – {x})
16.         end
17.  until todos valores D(x) sejam negativos
18.  t ← t+1
19. Gt ← tempG
20. AGt ← (AGt-1 – {G}) ∪ {G, Gt}
21. G ← grupoMaiorDiametro(AGt)          % encontrar grupo com maior diâmetro
22. until t = MaxGrupos
23. return AGt
24. end_procedure
```

Fonte: (KAUFMAN; ROUSSEEUW, 1990)

4.2 Métodos hierárquicos divisivos

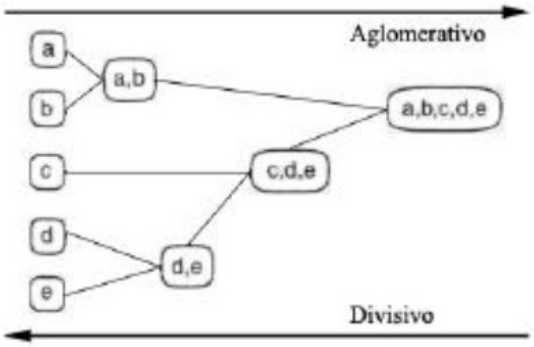
Existem alguns métodos para que um determinado conjunto de dados então seja dividido em um número k de *clusters*. A abordagem hierárquica divisiva é a que será

utilizada no presente trabalho.

Tal método é definido por uma série de sucessivas divisões dos elementos do conjunto de dados, onde tais elementos são divididos de acordo com suas distâncias relativas aos demais elementos do conjunto (DONI, 2004).

A figura 13 ilustra a maneira como os dados são clusterizados de acordo com as respectivas abordagens, aglomerativa e divisiva.

Figura 13: Ilustração do fluxo de divisão e aglomeração de dados



Fonte: (DONI, 2004)

Os métodos divisivos são os mais incomuns entre os métodos hierárquicos, devido a sua exigência de uma capacidade computacional mais alta (Costa, 1990), motivo este que justifica também a utilização de computação paralela em conjunto com algoritmos de clusterização divisivos.

Para se medir a qualidade de um agrupamento divisivo de dados é utilizado o coeficiente divisivo, ou CD (Kauffman, 1990). Assim, para cada dado i, tem-se d(i) que é o diâmetro do último agrupamento ao qual i pertenceu, dividido pelo diâmetro de todo conjunto de dados. O coeficiente então é definido como:

$$\frac{1}{n}\sum d(i) \tag{4.1}$$

Os valores dos coeficientes variam de 0 a 1, onde valores baixos correspondem a estruturas ruins, i.e onde nenhum poucos ou nenhum agrupamento foi encontrado, enquanto valores próximos a 1 representam que estruturas muito claras foram identificadas (DONI, 2004).

Existem diversos métodos específicos para analisar as distâncias relativas entre os dados e então decidir para qual *cluster* determinado dado deve ir, a partir de uma análise da matriz de similaridades que será criada ao longo do algoritmo. Essa matriz armazena todas as distâncias de todos para todos os dados(DONI, 2004).

O presente trabalho utilizará dois métodos específicos para a clusterização, sendo eles o *Average Linkage* e o *Centroid Linkage*.

4.2.1 Método *average linkage*

Neste método é levada em consideração todas as distâncias dos dados presentes em um determinado grupo G, e um determinado grupo H. Assim, para se determinar se um dado i pertence ao grupo G ou ao grupo H, é feita a média de similaridade de i com relação a todos os elementos do grupo H e a média de similaridade com relação a todos os elementos do grupo G, com isso é possível determinar qual grupo é mais similar ao dado i.

Matematicamente o método é definido por:

$$d(i, G) = \frac{1}{nG} \sum d_{i,j} \ , j \epsilon G \tag{4.2}$$

O foco deste método está no equilíbrio dos dados, então os *clusters* tendem a ser relativamente compactos e relativamente distantes uns dos outros (DONI, 2004).

Existem algumas características relevantes sobre este método (Kauffman, 1990): primeiramente é que existe uma menor sensibilidade à ruídos, o método apresenta também bons resultados quando utilizados com distância Euclidiana, e existe também uma tendência de se formar grupos com um número de elementos similar.

4.2.2 Método *centroid linkage*

Neste método é levado em consideração os centroids de cada grupo para o cálculo das distâncias, seja entre um dado e um grupo ou as distâncias entre dois grupos.

Um centroid pode ser definido como o centro geométrico de um grupo, i.e a média aritmética das características de todos os dados pertencentes a um determinado grupo, pode ser também chamado de baricentro ou centro de gravidade (ALTSHILLER-COURT, 1925).

O centroid de um grupo G é representado por C_G, assim a distância entre um grupo G e um dado i é dada por:

$$d(i, G) = \|i - C_G\| \tag{4.3}$$

O método tem como característica a robustez na presença de ruídos e devido ao fenômeno da reversão o método não é muito utilizado.

O fenômeno da reversão ocorre quando a distância entre centróides é menor que a distância entre grupos já formados, isso fará com que os novos grupos se formem a um nível inferior aos grupos existentes (DONI, 2004).

5 Metodologia

Para a realização deste trabalho, inicialmente foi necessário um período para o estudo sobre utilização de computação paralela aplicada a algoritmos de clusterização. Em meio a isto, foram pesquisados artigos que relacionassem essas tecnologias para também os utilizar como base tanto para a fundamentação teórica, quanto para questões de comparação de resultados.

Para fins de testes foi utilizada uma base de dados teórica chamada DUNN, coletada a partir de um período de tempo ao qual ocorreram diversos nascimentos de bebês, o conjunto de dados contém 90 pontos com 4 caracteristicas, sendo elas o horário de nascimento, o sexo, o peso, e tempo decorrido após a meia noite que cada nascimento ocorreu. Para fins de visualização foi-se utilizado apenas duas características, sendo elas o peso e tempo decorrido após a meia-noite de cada nascimento, assim como a normalização destes dados.

Com relação ao algoritmo implementado, foi-se utilizada a linguagem C para a implementação serial e C CUDA para a implementação paralela. Ambas foram implementadas desde o início, tendo como base o pseudo-código descrito em (KAUFMAN; ROUSSEEUW, 1990). Já existem implementações seriais prontas do algoritmo DIANA, porém com a implementação desde o início teve como vantagem o entendimento de trechos específicos do código que foram muito úteis no momento da implementação paralela do algoritmo.

A estrutura de dados utilizada tanto para armazenar quanto para manipular os dados do conjunto foi uma matriz estática 3D, matriz[i][j][z], onde "i"representa o grupo que o dado pertence, "j"identifica o(s) elemento(s) presente(s) no grupo "i", e "z"identifica as caracteristicas e o indice único de cada elemento "j"do grupo "i". Tal estrutura foi utilizada tanto na implementação serial, quanto na paralela.

É importante ressaltar um momento específico do algoritmo, na qual é preenchida uma estrutura denominada matriz quadrada de dissimilaridade, tendo como dimensão a quantidade absoluta de elementos presentes no conjunto de dados, sendo preenchida com todas as distâncias euclidianas de todos os elementos com relação à todos os demais elementos. É exatamente essa parte do código que foi utilizada para a paralelização.

Para que a paralelização seja possível é necessária uma alocação de threads da GPU específica para o conjunto de dados utilizado. Por exemplo, o conjunto de dados DUNN possui 90 elementos, portanto a matriz de dissimilaridade terá dimensão de 90x90, e no caso da paralelização será utilizada uma thread específica para o preenchimento de cada posição da matriz em questão. Em poucas palavras, cada cálculo de distância eu-

clidiana acontece em uma thread na GPU. Para que isso fosse possível, foi alocado na GPU 90 blocos de threads, onde cada bloco possuia 90 threads. Com isso foi possível mapear as posições da matriz com relação a cada thread. Em termos de CUDA C cada linha da matriz de dissimilaridade tem como índice a identificação do bloco (blockIdx.x) e cada coluna tem como identificação o índice de cada thread (threadIdx.x), portanto a cada elemento da matriz de dissimilaridade pode ser descrito como matrizDissimilaridade[blockIdx.x][threadIdx.x].

É necessário ressaltar também que o CUDA C não possui ainda estruturas simples de dados para a abstração de uma matriz em termos de linhas e colunas, sendo possível apenas a manipulação de vetores. Portanto, foi necessária a linearização da matriz de dissimilaridade no momento da paralelização e consequentemente o uso da fórmula:

$$i = blockDim.x * blockIdx.x + threadIdx.x \tag{5.1}$$

Onde "i"é a posição absoluta de cada elemento da matriz de dissimilaridade linearizada, para identificação cada elemento.

Feito isso, foi calculado os tempos de execução de cada implementação e calculado os *speed ups* dos métodos *average linkage* e *centroid linkage*, através da fórmula:

$$S = \frac{Ts}{Tp} \tag{5.2}$$

Onde Ts é o tempo execução da implementação serial e Tp é o tempo de execução da implementação paralela para os respectivos métodos.

6 Resultados

Para a obtenção dos resultados foram realizados testes de execução para a medição dos tempos médios de execução da aplicação, seja ela tanto serial, quanto paralela, do ponto de vista do seu desenvolvimento.

Portanto então foram medidos 5 vezes os tempos de execução de cada aplicação, conforme mostra a figura 14, abaixo;

Figura 14: Tempos de execução.

Average Linkage

	Tempo de execução (s)				
SERIAL	3,43	3,41	3,38	3,40	3,49
PARALELO	3,49	3,39	3,42	3,36	3,42

Centroid Linkage

	Tempo de execução (s)				
SERIAL	3,80	3,32	3,44	3,57	3,34
PARALELO	3,57	3,23	3,17	3,30	3,17

Fonte: Feito pelo autor

Analisando a figura 14, o tempo de execução médio da implementação serial com o método average linkage foi de 3,42s, enquanto o tempo médio de execução da implementação paralela deste mesmo método foi de 3,41s, tendo como *speed up* o valor de 1,0029 não sendo significativo o sufiente para caracterizar-se como *speed up*. E na implementação serial do método centroid linkage obteve-se um tempo médio de execução de 3,49s, e tempo médio de execução para a versão paralela foi de 3,28s com *speed up* de 1,064 , também não sendo significativo o suficiente para se caracterizar como *speed up*, de acordo com o artigo de (PILLA, 2009), onde os *speed ups* obtidos foram na ordem de 40 com a utilização da GPU.

7 Conclusão

De acordo com o tempo levado para o obtenção de conhecimento sobre as tecnologias existentes sobre a computação paralela é suficiente dizer que tal abordagem de programação e implementação de aplicações que utilizem tais conceitos, é diferentes do estilo de programação tradicional sequencial, levando a um nível de abstração nunca visto pelo autor. Porém é necessário dizer também que tais abordagens de programação são importantes para a resolução de problemas que exigem um processamento relativamente maior devido a quantidade de dados envolvida. Quantidades estas que tendem a aumentar com o tempo, partindo de um princípio de resoluções de problemas relacionados com a realidade.

O presente trabalho analisou o implementou, tanto serial quanto paralelamente dois métodos específicos para medição das distâncias dos elementos aos grupos (*clusters*), sendo eles o average linkage e o centroid linkage, que são métodos hierárquicos divisivos para clusterização de dados, analisando o conjunto de dados DUNN.

Com relação à isso, conclui-se que não houve speed up no tempo de processamento dos mesmos, não obtendo resultados numericamente significativos para a afirmação do speed up. Tal fato se da pela natureza do algoritmo envolvido e de como os dados são tratados, possibilitando paralelizar apenas algumas partes do algoritmo, as quais os cálculos que envolvem os dados, são independentes entre si. Seria interessante, em trabalhos futuros, a utilização de computação paralela para análise de outros algoritmos de clusterização, que tenham uma outra abordagem de manipulação de dados, e.g o algoritmo AGNES, também proposto por (KAUFMAN; ROUSSEEUW, 2008), assim como a utilização de outras bases de dados maiores para a obtenção de resultados mais significativos.

Referências

AGARWAL, R.; DHAR, V. Editorial-big data, data science, and analytics: The opportunity and challenge for is research. *Info. Sys. Research*, INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 25, n. 3, p. 443–448, set. 2014. ISSN 1526-5536. Disponível em: <http://dx.doi.org/10.1287/isre.2014.0546>.

ALTSHILLER-COURT, N. *College Geometry: An Introduction to the Modern Geometry of the Triangle and the Circle*. [S.l.]: New York: Barnes & Noble, 1925.

COOK, S. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. 1st. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN 9780124159334, 9780124159884.

DOBRE, C.; XHAFA, F. Intelligent services for big data science. *Future Generation Computer Systems*, v. 37, n. Supplement C, p. 267 – 281, 2014. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X13001593>.

DONI, M. V. AnÁlise de cluster: MÉtodos hierÁrquicos e de particionamento. 2004. ISSN 0148-2963.

FRANCA, R. dos S. Otimização e estimação de paramêtros para processamento de padrões interferométricos por computação evolucionária. 2015.

FREITAS, A.; LAVINGTON, S. *Mining Very Large Databases with Parallel Processing*. Springer US, 2012. (Advances in Database Systems). ISBN 9781461555216. Disponível em: <https://books.google.com.br/books?id=xqbfBwAAQBAJ>.

FROZZA, C. G. e R. Sistema de recomendação de produtos utilizando mineração de dados. *Tecno-Lógica*, v. 17, n. 1, 2013.

HAN, J.; PEI, J.; KAMBER, M. *Data Mining: Concepts and Techniques*. Elsevier Science, 2011. (The Morgan Kaufmann Series in Data Management Systems). ISBN 9780123814807. Disponível em: <https://books.google.com.br-/books?id=pQws07tdpjoC>.

JAMES, G. et al. *An Introduction to Statistical Learning: With Applications in R*. [S.l.]: Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370.

KAUFMAN, L.; ROUSSEEUW, P. *Finding Groups in Data: An Introduction to Cluster Analysis*. [S.l.]: Wiley, 1990. (Wiley Series in Probability and Statistics). ISBN 9780471878766.

KAUFMAN, L.; ROUSSEEUW, P. J. *Divisive Analysis (Program DIANA)*. John Wiley Sons, Inc., 2008. 253–279 p. ISBN 9780470316801. Disponível em: <http://dx.doi.org-/10.1002/9780470316801.ch6>.

KRAMER, A. D. I.; GUILLORY, J. E.; HANCOCK, J. T. Experimental evidence of massive-scale emotional contagion through social networks. *Proceedings of the*

National Academy of Sciences, v. 111, n. 24, p. 8788–8790, 2014. Disponível em: <http://www.pnas.org/content/111/24/8788.abstract>.

MAGOULAS, R.; LORICA, B. *Big Data Technologies and Techniques for Large-Scale Data*. [S.l.: s.n.], 2009. (Release 2.0.1).

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. *Foundations of Machine Learning*. [S.l.]: The MIT Press, 2012. ISBN 026201825X, 9780262018258.

MOORE, G. E. Cramming more components onto integrated circuits. 1966.

NVIDIA. *CUDA Documentation*. 2017. http://docs.nvidia.com/cuda/. Acesso em: 29 mar 2017.

PILLA, P. O. A. N. L. L. Uso da classficação dwarf mine para a avaliação comparativa entre a arquitetura cuda e multicores. 2009. ISSN 0148-2963.

SIVARAJAH, U. et al. Critical analysis of big data challenges and analytical methods. *Journal of Business Research*, v. 70, n. Supplement C, 2017. ISSN 0148-2963.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN 0321321367.

WITTEN, I. H.; FRANK, E.; HALL, M. A. *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN 0123748569, 9780123748560.

ZYTKOW, J.; RAUCH, J. *Principles of Data Mining and Knowledge Discovery: Third European Conference, PKDD'99 Prague, Czech Republic, September 15-18, 1999 Proceedings*. Springer Berlin Heidelberg, 2004. (Lecture Notes in Computer Science). ISBN 9783540482475. Disponível em: <https://books.google.com.br-/books?id=2Q1qCQAAQBAJ>.