Métodos de Agrupamentos para Grande Volume de Dados

Gustavo Rodrigues Lacerda Silva

Programa de Pós-Graduação em Engenharia Elétrica

Universidade Federal de Minas Gerais

Orientador: Prof. Antônio de Pádua Braga

Qualificação de

Doutorado em Engenharia Elétrica

05/2016

Resumo

Este trabalho apresenta uma metodologia direcionada para problemas de agrupamentos com alta dimensão e volume de dados. O objetivo é projetar algoritmos que tenham a capacidade de processar grandes volumes de dados sem a perda de qualidade do agrupamento. Uma das contribuições é o método de agrupamento GPIC, que realiza tanto o cálculo da matriz de afinidades quanto dos autovetores utilizando Unidade de Processamento Gráfico GPU. Resultados preliminares com bases de dados sintéticas demonstram que a abordagem proposta por este trabalho consegue processar uma grande quantidade de dados em tempo menor, mantendo a qualidade do agrupamento.

Sumário

Lista de Símbolos

X	Matriz de dados multivariada
k	Número de agrupamentos
n	Quantidade de características
p	Quantidade de amostras
m	Quantidade de máquinas
A	Matriz de Afinidades

Lista de Abreviaturas

ALU	Arithmetic Logic Unit
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
GLM	Generalized Linear Models
MLE	Maximum-Likelihood Estimation
SIMD	Single Instruction Multiple Data
SM	Streaming multiprocessor
UVA	Unified Virtual Addressing

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A progressiva coleta de dados dos meios de comunicação tem transformado nossa sociedade, afetando as relações de trabalho, além de criar novos hábitos sociais.

Fornecidos com eletrônica avançada, diferentes tipos de dispositivos são cada vez mais dotados de autonomia, flexibilidade, comunicação, interoperabilidade e estão constantemente assumindo funções que agregam valor para os indivíduos e para a sociedade.

As sinergias física e computacional desses dispositivos formam a base de uma profunda transformação da economia global. Esse novo cenário expande as fronteiras da aquisição de dados, da infraestrutura de tecnologia de processamento, além de possibilitar uma significativa melhora na conectividade, comunicação e computação das informações (Bi *et al.*, 2014).

Atualmente, pode-se verificar uma demanda de métodos de aprendizagem de máquina que visam a análise, previsão, adaptação, monitoramento, supervisão e tomada de decisões com base em um grande volume de dados (Wu *et al.*, 2014). Métodos orientados por dados, tais como os métodos de agrupamentos, são considerados um dos caminhos viáveis para enfrentar os desafios nesse novo contexto.

A análise de um grande volume de dados apresenta-se como um dilema em métodos de agrupamentos, uma vez que é de se esperar que em um maior número de amostras exista uma melhor representatividade da informação, e, consequentemente, ocorra uma melhora no desempenho dos métodos. No entanto, a realidade

é que, apesar de uma representatividade superior, a maioria dos métodos de agrupamento de dados atuais não é computacionalmente capaz de lidar com esse alto volume e dimensão dos dados (Fahad *et al.*, 2014).

Até o momento, as principais contribuições deste trabalho são:

- Implementação de um algoritmo que explora o paralelismo de dados na GPU e reduz de forma siginificativa o tempo de execução sem a perda da qualidade do agrupamento (*GPIC - Gpu Power Iteration Cluster*).
- Uma metodologia para amostrar e reduzir o conjunto de dados gerados pela matriz de afinidades resultante do método *GPIC*.

Seguindo essa proposta, o presente trabalho está organizado da seguinte forma: no capítulo 2 é apresentado um conjunto de conceitos básicos que é utilizado ao longo de todo o texto, seguido pela descrição dos principais trabalhos relacionados. Por fim, no capítulo 3 são descritas as constribuições.

Capítulo 2

Referencial Teórico

Neste capítulo serão apresentados os conceitos de *Big Data*, agrupamentos de dados, métodos de agrupamentos tradicionais, distribuídos e em GPU. Por fim, serão apresentadas algumas técnicas de amostragem estatística para *Big Data*.

2.1 *Big Data*

Big Data pode ser definido como volumes de dados disponíveis em diferentes níveis de complexidade, gerado em diferentes velocidades e com diferentes níveis de ambiguidade, que não podem ser processados com a utilização das tecnologias tradicionais, (Chen & Zhang, 2014), (Krishnan, 2013).

Alguns autores utilizam a definição dos três V's (Volume, Variedade e Velocidade) para conceituar o termo *Big Data*, (Hashem *et al.*, 2015), (Davis, 2012), (O Reilly Media, 2015), (Zikopoulos *et al.*, 2011), (Dong & Srivastava, 2013), (Reeve, 2013). Outros autores vão além e adicionam mais três V's (Valor, Veracidade e Visualização) ao conceito de *Big Data* (Hitzler & Janowicz, 2013), (Van Rijmenam, 2014), (Manyika *et al.*, 2011). Tais definições são apresentadas a seguir:

- Volume (dado como um todo): é uma tendência, para muitas empresas, armazenar grande quantidade de vários tipos de dados (redes sociais, saúde, financeiros, bioquímicos, genéticos, astronômicos, etc). Para se ter uma ideia, o mundo produz cerca de 2,5 quintilhão de bytes por dia, e até 2020

serão criados 43 milhões de *gigabytes* (Information, 2015), (O Reilly Media, 2015). A tabela 2.1 define a categorização do volume baseada no tamanho dos dados proposta por Bezdek e Hathaway (Hathaway & Bezdek, 2006).

- Variedade (dados em diversas formas): dados gerados a partir de diversas fontes. Os dados podem ser estruturados, semi-estruturados e não estruturados (O'Leary, 2013).

- Velocidade (dado em movimento): necessidade de receber, processar e analisar o imenso volume de dados quase que em tempo real (Berman, 2013).

- Valor (dado em evidência): define os benefícios econômicos que o dado traz para as empresas, organização e sociedade (Chen *et al.*, 2014).

- Veracidade (precisão do dado): define o que está em conformidade com a verdade ou fato, irá determinar a precisão, certeza e acurácia do dado. A incerteza pode ser causada por inconsistências, aproximações de modelos, ambiguidades, enganos, fraudes, duplicações ou incompletudes. A análise dos dados apresenta inferências com determinados níveis de qualidade, ou seja, os resultados obtidos através de um grande volume de dados não possuem 100% de exatidão, apenas indicam a probabilidade de tais resultados ocorrerem partindo-se de uma determinada previsão (Gandomi & Haider, 2015).

- Visualização (dado compreendido): define as técnicas e as tecnologias utilizadas para a criação de imagens, diagramas, que facilitem a compreensão e a melhoria nas análises dos resultados de *Big Data* (Reeve, 2013), (Keim *et al.*, 2013).

Tabela 2.1: Categorização para *Big Data*

			Big Data		
Bytes	10^6	10^8	10^{10}	10^{12}	$10^{>12}$
Tamanho	Médio	Grande	Enorme	Gigante	Muito Grande

Apesar dos obstáculos e dos desafios envolvidos, o valor potencial de se analisar grande volume de dados é de suma importância, pois esse novo conhecimento adquirido através de técnicas, modelos, teorias e ferramentas permitirá descobertas revolucionárias e inovadoras nas diversas áreas do conhecimento (engenharia, medicina, comércio, educação, segurança pública, privada, etc).

2.2 Agrupamentos de Dados

2.2.1 Análise de Agrupamentos

A maioria dos métodos de análise de agrupamentos trabalha com uma representação da matriz de dados multivariada, X, que contém os valores das variáveis que descrevem cada objeto a ser agrupado.

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{pmatrix}$$

A entrada X_{ij} em X representa o valor da $j$ézima variável no objeto i. As variáveis da matriz X podem possuir os mais diversos tipos de dados (ausentes, repetidos, ruidosos, contínuos, ordinais etc).

A análise de agrupamento de dados pode ser definida pela seguinte proposição: dada uma representação de matriz de dados multivariada X, o objetivo é encontrar k grupos com base em uma métrica estatística de tal forma que as semelhanças entre os objetos do mesmo grupo sejam elevadas, enquanto as semelhanças entre objetos de diferentes grupos sejam baixas (Gan *et al.*, 2007).

Os autores (Han *et al.*, 2011), (Aggarwal & Reddy, 2013) apresentam algumas considerações sobre a análise de agrupamentos, tais como:

- Critérios de Particionamento: nível único *versus* particionamento hierárquico.
- Separação dos Agrupamentos: exclusivo (pertence apenas a uma classe), não exclusivo (pertence a mais de uma classe).

- Medidas de Similaridade: todos os métodos de agrupamentos baseiam-se em alguma medida de similaridade ou dissimilaridade entre os dados (Gan *et al.*, 2007). A escolha da função de similaridade depende muito da representação dos dados, bem como qual é o objetivo do método de agrupamento. De uma forma geral, a medida de similaridade irá encapsular muito conhecimento prévio sobre o domínio do problema, e deve ser definida ou aprendida em cooperação com os especialistas do problema (Everitt *et al.*, 2011). Uma boa prática é, geralmente, normalizar os atributos para evitar certas características dominantes dos cálculos de distância. As medidas de distância mais utilizadas são apresentadas na tabela 2.2.

Tabela 2.2: Medidas de Distâncias e Similaridades mais Utilizadas em Métodos de Agrupamentos.

Nome	Fórmula
Euclidiana	$d(x,y) = \left(\sum_{j=1}^{d}(x_j - y_j)^2\right)^{\frac{1}{2}}$
Manhattan	$d(x,y) = \sum_{j=1}^{d}\lvert x_j - y_j\rvert$
Minkowski	$d(x,y) = \left(\sum_{j=1}^{d}\lvert x_j - y_j\rvert^r\right)^{\frac{1}{r}}, r \geq 1$
Máxima	$d(x,y) = \max_{1\leq j\leq k}\lvert x_j - y_j\rvert$
Mahalanobis	$d(x,y) = \sqrt{(x-y)\sum^{-1}(x-y)^T}$

2.2.2 Desafios na Análise de Agrupamentos

A representatividade dos dados é um dos fatores mais importantes e que possui influência direta no desempenho do método de agrupamento. Caso os dados tenham boa representatividade, os agrupamentos tendem a ser compactos e isolados, caso contrário, os métodos de agrupamentos deverão ter a capacidade de trabalhar com os mais variados tipos de dados.

Outro fator desafio importante é a qualidade dos agrupamentos em que a matriz X define o conjunto de dados e a função $d(x_i, x_j)$ representa uma função qualquer de similaridade ou dissimilaridade entre as instâncias do problema x_i e x_j. Depois de se obter os agrupamentos é importante avaliar a validade e a qualidade dos mesmos.

2.2.2.1 Qualidade dos Agrupamentos

Existem diversas métricas encontradas na literatura para avaliar a qualidade dos agrupamentos, e sua escolha depende da aplicação (Halkidi *et al.*, 2001). Os critérios para analisar a qualidade dos agrupamentos, normalmente, são baseados em índices estatísticos que medem, de forma quantitativa, a qualidade de uma partição. Existem três tipos de critérios: externos, internos e relativos.

Nos critérios externos a qualidade do agrupamento é medida de acordo com uma divisão prévia estabelecida a partir do conjunto de dados.

Já os critérios internos, sejam os dados originais ou a própria matriz de similaridade, são utilizados para medir a qualidade do agrupamento.

Por fim, os critérios relativos apresentam uma abordagem em que duas divisões do mesmo conjunto de dados são comparadas para identificar qual divisão representa melhor o agrupamento.

2.2.2.2 Escalabilidade dos Métodos de Agrupamento

A escalabilidade dos métodos é um fator crucial, principalmente quando se trabalha com métodos de agrupamento para *Big Data*. Existem dois fatores extremamente importantes para determinar a escalabilidade de um algoritmo de agrupamento, um é a complexidade do tempo de execução e o outro é o consumo de memória. Algoritmos de agrupamento que possuam a complexidade de tempo linear ou sub-linear e que consumam pouca memória são desejáveis para trabalhar com grande volume de dados.

2.2.2.3 Número de Agrupamentos

A maioria dos métodos de agrupamentos exige a especificação do número de agrupamentos k. Outros métodos aceitam parâmetros que estão diretamente

relacionados ao problema. Determinar automaticamente o número de grupos é um problema difícil e, na prática, o conhecimento do domínio é utilizado para determinar esse parâmetro. Alguns trabalhos propõem alternativas para o número ideal de agrupamentos (Tibshirani *et al.*, 2001), (Celeux & Soromenho, 1996), (Fraley & Raftery, 1998).

2.2.3 Categoria de Métodos de Agrupamentos

Os métodos de agrupamentos podem ser classificados pelos mais diversos tipos. A seguir serão apresentadas as principais características desses algoritmos.

2.2.3.1 Métodos Baseados em Partição

Dividem o espaço de características em áreas de influência baseadas na distância aos centros das partições. Assim, cada agrupamento pode ser representado pelo centro dos seus elementos. Os métodos de partição mais conhecidos são K-médias (MacQueen *et al.*, 1967), Mean Shift (Fukunaga & Hostetler, 1975), K-medóides (Park & Jun, 2009), K-Modas (Huang, 1997), CLARANS (Ng & Han, 2002).

2.2.3.2 Métodos Hierárquicos

Organizam os dados em uma estrutura hierárquica de acordo com as similaridades entre os padrões. As técnicas hierárquicas não necessitam do número de grupos fornecido a priori. Os métodos hierárquicos mais conhecidos são: BIRCH (Zhang *et al.*, 1996), CURE (Guha *et al.*, 1998), ROCK (Guha *et al.*, 1999), CHAMELEON (Karypis *et al.*, 1999).

2.2.3.3 Métodos Baseados em Densidade

Métodos Baseados em Densidade são capazes de encontrar agrupamentos de forma arbitrária, onde os agrupamentos são definidos nas regiões de maior densidade, separados por regiões de menor densidade, que utilizam a conectividade e funções de densidade para realizar os agrupamentos. Os métodos baseados em densidade mais conhecidos são: DBSCAN (Ester *et al.*, 1996), OPTICS (Ankerst *et al.*, 1999), DENGLUE (Hinneburg & Keim, 1998).

2.2.3.4 Métodos Baseados em Grafos

Representam os dados e sua proximidade através de um grafo $G(V, E)$, sendo que $V = \{v_1, ..., v_n\}$ é o conjunto de vértices e E é o conjunto de arestas. Cada vértice representa um elemento do conjunto de dados, e a existência de uma aresta conectando dois vértices é feita com base na proximidade entres os dois dados. Os métodos baseados em grafos mais conhecidos são: *Chameleon* (Karypis *et al.*, 1999), CACTUS (Ganti *et al.*, 1999), ROCK (Guha *et al.*, 1999), *Spectral Cluster* (Ng *et al.*, 2002).

Agrupamentos Espectrais

Colocar aqui a definição de Cluster Espectrais Fast Approximate Spectral Clustering. Colocar aqui a definição do PIC.

Power Iteration Clustering

Power Iteration Clustering é um método de agrupamento espectral proposto por (Lin & Cohen, 2010). Esse método calcula o maior autovetor de uma matriz de afinidades através do método das potências (Lanczos, 1950). A versão original do PIC é apresentada no Algoritmo 2.

Algorithm 1 Power Iteration Cluster

Input:

W	Matriz de afinidades normalizada
k	Quantidade de agrupamentos
v_0	Vetor inicial
ϵ	Precisão desejada

Output:

C	Agrupamentos resultantes $C_1, C_2, ..., C_k$

$\delta_0 \leftarrow v_0$ ▷ Inicializa a variação da variação do autovetor
for $t = 0, 1, 2, ...$ **do**
  $v_{t+1} \leftarrow \frac{Wv_t}{\|Wv_t\|_1}$ ▷ Atualiza a estimativa dos autovetores de W
  $\delta_{t+1} \leftarrow |v_{t+1} - v_t|$ ▷ Atualiza a variação do autovetor anterior e o atual
  if $|\delta_{t+1} - \delta_t| \leq \epsilon$ **then** ▷ Critério de parada: convergência do autovetor
    break ▷ Finaliza a estimativa dos autovetores
  end if
end for
$C \leftarrow k$-médias de v_t ▷ Agrupamento dos autovetores
return C

2.2.3.5 Métodos Baseados em *Kernel*

Permitem o mapeamento não linear do espaço original dos padrões de entrada para um espaço de mais alta dimensão, chamado espaço de características. Os métodos de agrupamentos baseados em Kernel mais conhecidos são: *Mercer kernel-based* (Girolami, 2002), *Kernel K-means* (Dhillon *et al.*, 2004).

2.2.3.6 Métodos Nebulosos

Permitem que uma instância do problema possa pertencer a dois ou mais agrupamentos, mas com diferentes pertinências. Os métodos de agrupamentos nebulosos mais conhecidos são: *Fuzzy c-means* (Bezdek *et al.*, 1984), *Fuzzy k-modes* (Huang & Ng, 1999).

2.2.3.7 Métodos Baseados em *Grid*

Dividem o espaço de variáveis em uma quantidade finita de células formando uma estrutura de grade. Todas as operações de agrupamentos são aplicadas nessa estrutura de grade. Os métodos de agrupamentos baseados em *Grid* mais conhecidos são: *WaveCluster* (Sheikholeslami *et al.*, 1998), STING (Wang *et al.*, 1997), CLIQUE (Agrawal *et al.*, 1998) OptiGrid (Hinneburg & Keim, 1999).

2.2.3.8 Métodos Baseados em Modelos

Assumem que os dados provêm de uma mistura de distribuições de probabilidade, cada um dos quais representa um grupo diferente. Os métodos de agrupamentos baseados em Modelos mais conhecidos são: EM (Dempster *et al.*, 1977) e SOMS (Kohonen, 1998).

2.2.4 Análise de Métodos de Agrupamentos Tradicionais para *Big Data*

Em (Fahad *et al.*, 2014) é realizada uma análise detalhada de alguns métodos de agrupamentos com o objetivo de avaliar a aderência de cada um deles quando aplicados a um problema de *Big Data*. Os critérios adotados pelo autor foram:

- Capacidade de se trabalhar com diversos tipos de dados.
- Tamanho do conjunto de dados.
- Quantidade de parâmetros de entrada.
- Capacidade de se trabalhar com *outliers* e dados ruidosos.
- Tempo de Execução.
- Estabilidade: capacidade de gerar a mesma partição do dados independentemente da ordem pela qual os padrões são apresentados ao algoritmo.
- Capacidade de se trabalhar com problemas de alta dimensão.
- Formato do agrupamento.

Baseado nos critérios citados acima, os métodos de agrupamentos indicados pelo autor (Fahad *et al.*, 2014) para trabalhar com problemas de *Big Data* foram: *Fuzzy C-Means* (Bezdek *et al.*, 1984), BIRCH (Zhang *et al.*, 1996), DENGLUE (Hinneburg & Keim, 1998), OptiGird (Hinneburg & Keim, 1999) e EM (Dempster *et al.*, 1977). Os resultados encontrados em (Fahad *et al.*, 2014) são apresentados na tabela 2.3. O autor aponta, ainda, algumas conclusões sobre a avaliação realizada:

- Nenhum algoritmo conseguiu respeitar todos os critérios propostos.
- Os algoritmos EM e *Fuzzy C-Means* apresentaram um excelente desempenho com relação a qualidade do agrupamento, exceto quando submetidos a dados com alta dimensão.
- Nenhum dos algoritmos conseguiu respeitar o critério de estabilidade.
- DENGLUE, OptiGrid e BIRCH são métodos indicados para grande volume de dados, especialmente os dois primeiros que possuem a capacidade de trabalhar com problemas de alta dimensão.

Tabela 2.3: Melhores Métodos de Agrupamento para *Big Data* (Fahad *et al.*, 2014).

Algoritmos	Validação Externa	Validação Interna	Estabilidade	Tempo de Execução	Escalabilidade
EM	Sim	Parcial	Não	Alto	Baixa
Fuzzy C-Means	Sim	Parcial	Não	Alto	Baixa
DENGLUE	Não	Sim	Não	Baixo	Alta
OptiGrid	Não	Sim	Não	Baixo	Alta
BIRCH	Não	Não	Não	Baixo	Alta

2.3 Métodos de Agrupamentos Paralelos e Distribuídos para Grande Volume de Dados

A implementação de algoritmos de agrupamentos paralelos não é algo tão recente (Li & Fang, 1989). Com o crescimento do volume de dados a escalabilidade de um método de agrupamento se torna praticamente obrigatória, visto que uma única máquina não é capaz de lidar com *terabytes* e *petabytes* de dados.

Existem diversas formas para dividir o volume de dados a ser processado. Uma abordagem é dividir o trabalho entre os vários núcleos dentro de uma mesma CPU ou GPU ou dividir o trabalho entre várias máquinas.

A abordagem *multi-core* possui suas limitações relacionadas a processador, memória e disco. Uma alternativa natural a abordagem *multi-core* é a utilização do conceito de processamento em múltiplas máquinas, o que permite uma escalabilidade de processamento, memória e disco.

No entanto, em contrapartida, a adoação dessa prática acarreta um custo alto de comunicação entre as máquinas. Essa técnica permite que os dados sejam distribuídos em pedaços menores, em diferentes máquinas e, em seguida, tais máquinas utilizam o seu poder computacional para solucionar o problema proposto (Shirkhorshidi *et al.*, 2014).

Existem duas grandes direções em que a paralelização dos métodos de agrupamentos se apresenta na literatura o paralelismo de dados e o paralelismo de tarefas. O paralelismo de dados permite que cada tarefa execute uma mesma série de cálculos sobre diferentes dados. O foco é distribuir o dado por diferentes nós (servidores) para serem processados em paralelo. Ao passo que o paralelismo de tarefas é alcançado quando a execução do algoritmo pode ser dividida em segmentos, alguns dos quais são independentes e, por conseguinte, podem ser executados de forma concorrentemente.

2.3.1 Grande Volume de Dados com *MapReduce*

A computação distribuída agrega complexidade aos sistemas que necessitam oferecer tolerância a falhas, paralelização e comunicação em níveis aceitáveis. Nesse aspecto, um modelo que ofereça simplicidade e transparência do ponto de vista de

desenvolvimento de aplicações e, ao mesmo tempo, seja robusto e representativo na especificação de algoritmos que lidam com grandes volumes de dados é um grande desafio.

Em (Dean & Ghemawat, 2008) é apresentado a descrição do modelo *MapReduce* na perspectiva de computação distribuída e sua implementação em um *framework*. *MapReduce* combina conceitos de programação funcional com a estratégia de enviar a computação ao dado para garantir eficiência e, principalmente, a escalabilidade no paralelismo de dados. O modelo de programação proposto é expresso em duas funções:

- *Map(chave, valores)*: recebe como entrada arquivos a serem processados e produz um conjunto de pares $\langle$*chave*, *valor*$\rangle$, que representa a saída intermediária. O conjunto é agrupado pela chave produzindo pares $\langle$*chave*, *lista de valores associados com a chave*$\rangle$, que servem como entrada para a função de redução.

- *Reduce(chave, lista(valores))*: a lista de valores associada a uma chave gerada pela função de mapeamento é processada e a saída final é gerada.

A independência do processo possibilita a existência de diversas instâncias das funções de mapeamento e redução (também chamadas de tarefas), resultando em paralelismo de granularidade ideal para *clusters* formado por servidores com um custo mais acessível.

Na implementação *MapReduce* as tarefas de mapeamento e redução são atribuídas a *Workers*, que são escalonados por um *Master*. Além de realizar o escalonamento de tarefas aos *Workers* em diferentes máquinas (levando em conta fatores como localidade de dados e balanceamento de carga), o *Master* é responsável por receber requisições dos clientes, monitorar e reportar estado.

O *Master* verifica periodicamente se cada *Worker* está ativo. Em caso de falha, ou seja, um *Worker* não responder à mensagem do *Master* dentro de um intervalo de tempo, as tarefas de mapeamento não completadas devem ser reiniciadas em outro *Worker*, pois suas saídas não são armazenadas em meio persistente (disco rígido), ao contrário das tarefas de redução que já tenham terminado. A alta disponibilidade é garantida através de *checkpoints* de estado periódicos do

Master. Por fim, o problema de tarefas retardatárias é resolvido através da redundância, ou seja, quando uma tarefa estiver próxima o bastante do fim, o *Master* dispara uma tarefa clone e considera a que terminar primeiro.

A figura 2.1 apresenta o funcionamento do fluxo de execução *MapReduce*.

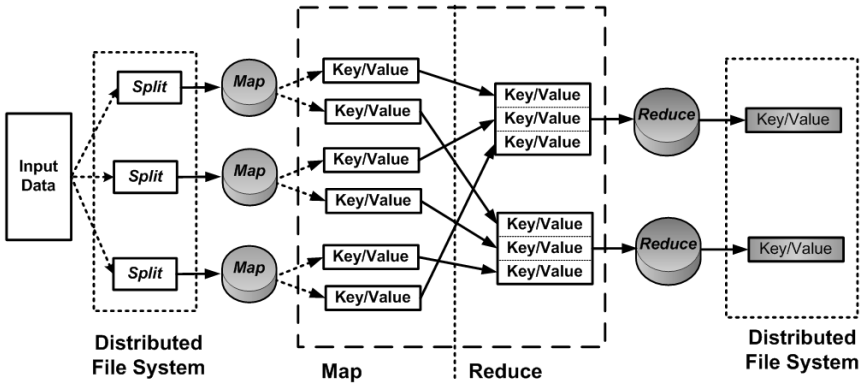

Figura 2.1: Fluxo de Execução *MapReduce* (Dean & Ghemawat, 2008).

Em (Chen *et al.*, 2011) é apresentada um método de agrupamento espectral em conjunto com *MapReduce* com o objetivo de calcular a matriz de similaridade e computar de forma eficiente os autovetores para grandes volumes de dados. O método distribui os a matriz de dados X em m nós(máquinas). Em cada nó o método calcula a matriz de afinidades A entre o subconjunto de dados distribuído. Assim, todo o conjunto de dados é processado de uma forma que o uso de disco e memória em cada nó sejam reduzidos.

2.4 Métodos de Agrupamentos para Unidades de Processamento Gráfico

As soluções de *software* que fazem uso efetivo das Unidades de Processamento Gráfico (GPU) apresentam ganhos significativos de desempenho (Owens *et al.*, 2008). Conforme mencionado anteriormente, o paralelismo de dados refere-se a

um programa que tenha a capacidade de realizar muitas operações aritméticas em estruturas de dados de forma simultânea. Neste trabalho o interesse é utilizar o paralelismo a nível de dados em GPUs para processar um grande volume de informações simultaneamente.

Em (Fatica *et al.*, 2008), os autores apresentam um estudo sobre como métodos de mineração de dados podem ser implementados com o auxílio de GPUs.

Uma versão do método de agrupamento denominado K-médias é implementada em (Farivar *et al.*, 2008) utilizando GPUs. Outra abordagem que utiliza o K-médias com GPUs é apresentada em (Wu *et al.*, 2009), onde os autores aplicam o K-médias em um conjunto de dados com um bilhão de pontos. Os resultados encontrados apontaram que a versão em GPU é até 10 vezes mais rápida que em CPU de 8 núcleos.

Outros algoritmos de agrupamentos que se beneficiam da utilização de GPUs para processamento de grande volume de dados são encontrados na literatura. Jianqiang et al. (Dong *et al.*, 2013) apresentam uma versão do método de agrupamento BIRCH em GPU. Os resultados encontrados apontam que a abordagem proposta é até 154 vezes mais rápida do que a versão original do método em CPU.

Em (Andrade *et al.*, 2013) os autores apresentam o G-DBSCAN, uma versão em GPU do algoritmo de agrupamento DBSCAN. A versão proposta em GPU apresentou resultados até 100 vezes mais rápida do que sua versão sequencial em CPU.

2.4.1 Arquitetura de uma GPU Cuda

A arquitetura de uma GPU pode variar de uma geração para outra, mas, normalmente, é composta de Multiprocessadores de *Streamming* (SMs), em que cada Multiprocessador de *Streamming* possui inúmeros Processadores de *Streaming* (SP) que compartilham unidades de lógica aritmética (ALUs) e *cache* de instruções (Kirk & Wen-mei, 2012). Além disso, cada SP possui a capacidade de executar milhares de *threads* por aplicação.

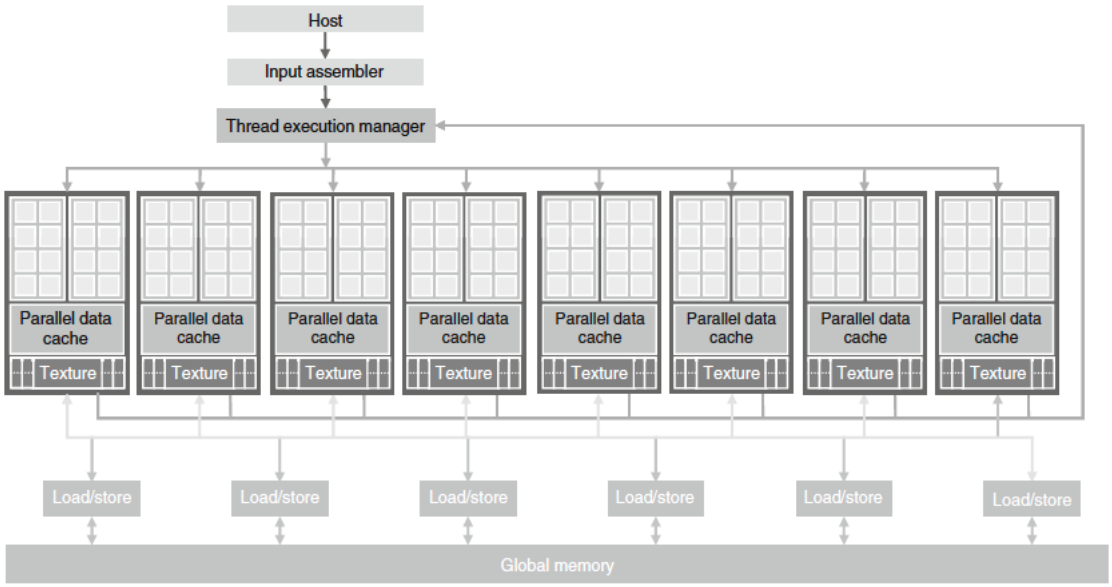

Figura 2.2: Arquitetura de uma GPU CUDA (Kirk & Wen-mei, 2012).

Com o objetivo de maximizar o desempenho, a arquitetura de uma GPU CUDA possui hierarquias de memória diferenciadas, conforme ilustrado na figura 2.3. As principais hierarquias são:

- Registrador: é o espaço de memória mais rápido de uma GPU.
- Memória compartilhada: serve basicamenente para comunicação intra-*thread*.
- Memória global: pode ser alocada de forma estática ou dinâmica através de ponteiros dentro dos *kernels*.
- Memória constante: acesso somente para leitura. As solicitações de acesso são fornecidas através de uma hierarquia de *cache* otimizada para realizar a transmissão para diversas *threads*.
- Memória local: variáveis locais que não podem ser manipuladas nos registradores, parâmetros e endereços de retorno para sub-rotinas.

- Memória de textura: assim como a memória constante, as solicitações de leitura desse tipo são fornecidas por um *cache* específico, otimizado para acesso somente de leitura.

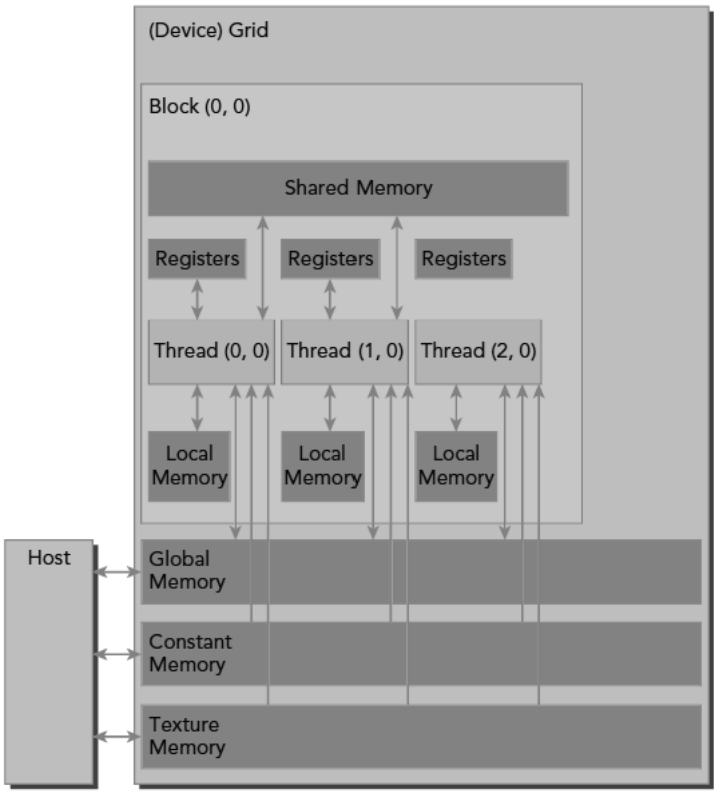

Figura 2.3: Hierarquias de Memória em GPU (Cheng *et al.*, 2014).

2.4.2 Modelo de Programação em GPU

CUDA é uma arquitetura paralela de propósito geral destinada a utilizar o poder computacional das GPUs NVIDIA. Implementanda através de uma extensão da linguagem C, permite controlar a execução de *threads* na GPU e gerenciar sua memória (Nickolls *et al.*, 2008).

No modelo de programação CUDA, a GPU é considerada um co-processador capaz de executar um grande número de *threads* em paralelo. Além disso, permite ao programador o desenvolvimento de um código para uma única *thread* (*single program multiple data*) com a capacidade de execução em múltiplas *threads* e em múltiplos *cores* (Kirk & Wen-mei, 2012).

Um dos conceitos importantes no modelo de programação em GPUs é o de *kernel*, que pode ser definido como um trecho de código executado em paralelo por múltiplas *threads* na GPU. Os *kernels* geram um grande número de *threads* com o objetivo de explorar o paralelismo a nível de dados. A figura 2.5 ilustra cada componente do fluxo de execução de um *kernel*. Tais componentes são descritos abaixo (Wilt, 2013):

- *Grid*: apresenta uma estrutura em forma de matriz, em que cada posição da matriz contém um bloco. O *grid* é o nível mais alto da hierarquia de paralelismo de uma GPU e, além disso, é dividido em blocos de *threads*, limitado ao tamanho máximo de 65535 x 65535 x 1 blocos.

- Blocos: cada bloco é alocado a um multiprocessador da GPU, que pode executar um ou mais blocos simultaneamente. O bloco é formado por um número pré-definido de threads (máximo de 512 *threads*). Todos os blocos criados possuem o mesmo número de *threads*.

- *Threads*: as *threads* de cada bloco são divididas em *Warps*, contendo 32 *threads* cada. As GPUs permitem a execução simultânea de todas as *threads* do *Warp*, desde que todas executem o mesmo código.

- *Warp*: é a menor unidade de escalonamento da GPU e é composta por um grupo de 32 *threads* que executam em modo SIMD. A taxa de ocupação de um *kernel* é a razão entre o número de *Warps* ativos sobre o número máximo de *Warps* suportados pelo dispositivo.

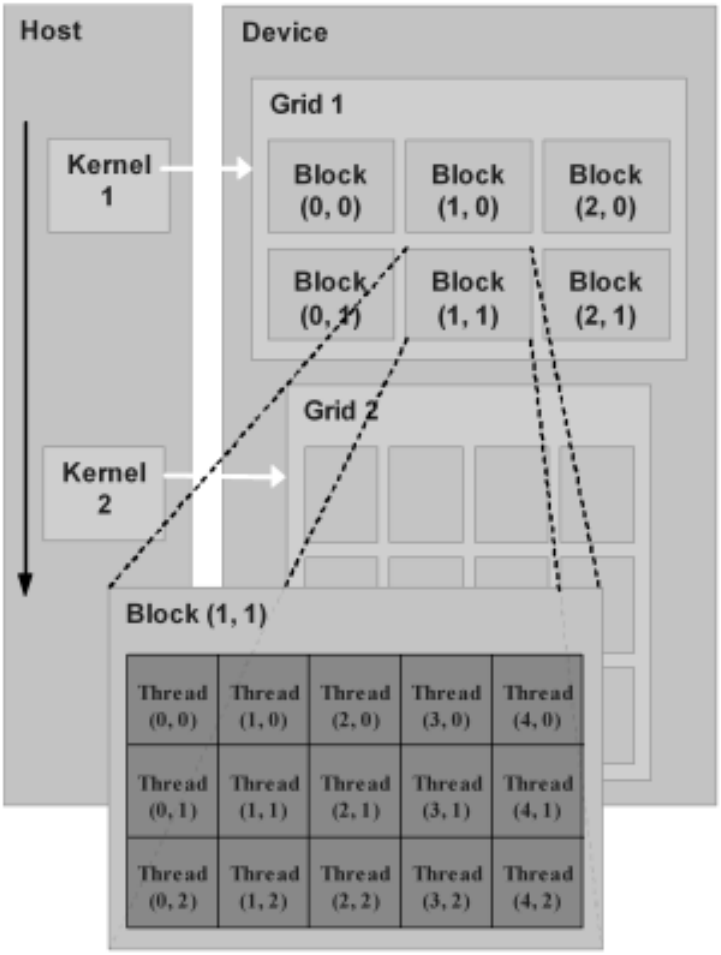

Figura 2.4: Grid, Bloco e Thread CUDA (Kirk & Wen-mei, 2012)

.

Fica a cargo do desenvolvedor especificar o número de *threads* por blocos e o número de blocos de um *grid*. É importante salientar que o número de *threads* por bloco também depende do modelo da placa gráfica.

A figura 2.5 apresenta o fluxo de execução de um programa CUDA. O início do programa se dá com a cópia dos dados a serem processados na memória do

host para a memória principal do *device* GPU. Logo após, um *grid* é configurado e é realizada uma chamada a um kernel que será executado na GPU. Após a execução no *device*, o programador deve transferir os dados da memória da GPU para a CPU e, finalmente, liberar a memória do *device*. O fluxo de execução pode ou não ser encerrado, o que depende da implementação realizada.

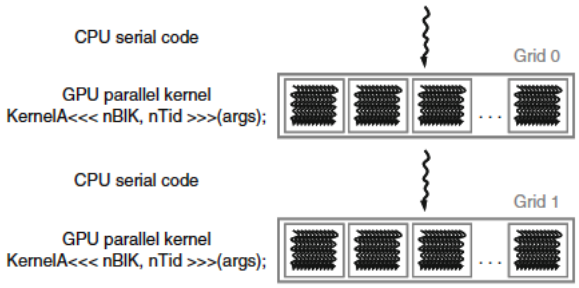

Figura 2.5: Fluxo de Execução de um programa CUDA (Kirk & Wen-mei, 2012)

.

2.4.3 Transferência de Dados em GPU

A computação de grande volume de dados em GPU possui algumas limitações, sendo a principal a transferência de dados entre GPUs e CPUs. CPUs e GPUs possuem tamanhos e velocidades de memória diferenciados. Portanto, entender qual é a melhor abordagem para se trabalhar com a alocação de memória em GPUs é de suma importância. A seguir são apresentadas as abordagens existentes (Cheng *et al.*, 2014):

- *Pinned memory*: representa uma porção de memória não paginável na memória principal. Os dados são alocados diretamente na área de memória do *host*, o que permite uma cópia direta da memória não paginada do *host* para a memória do *device*. A figura 2.6 apresenta o funcionamento da *Pinned Memory*.
- *Unified virtual addressing*: permite que a memória do *host* e a memória do *device* possam ser compartilhadas em um mesmo espaço de memória, conforme ilustrado na figura 2.7.

- *Zero-copy memory*: permite que através de uma função *kernel* seja possível acessar a região de memória do *host*. Assim, o programador não necessita de realizar a cópia de dados do *host* para o *device*.

- *Unified memory*: a principal diferença da *Unified Memory* para *Unified Virtual Addressing* é que a UVA não migra os dados fisicamente de um dispositivo para o outro, essa capacidade é exclusiva da *Unified Memory*. Esse conceito permite desacoplar a memória da execução de modo que os dados possam ser migrados sob demanda para o *host* ou *device* com o objetivo de melhorar a localidade e o desempenho da execução.

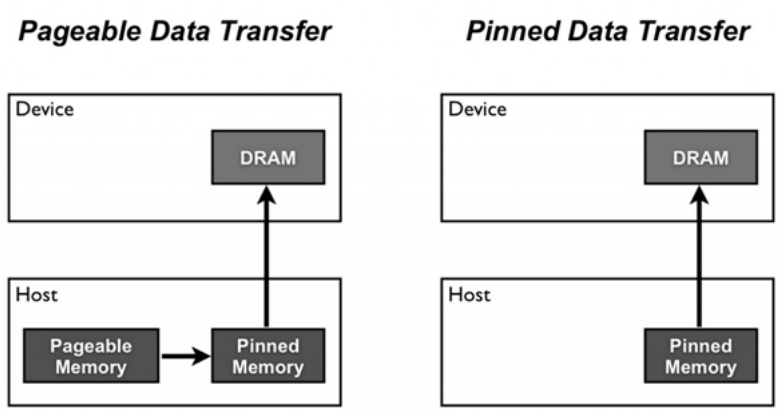

Figura 2.6: Pinned Memory (Cheng *et al.*, 2014)

.

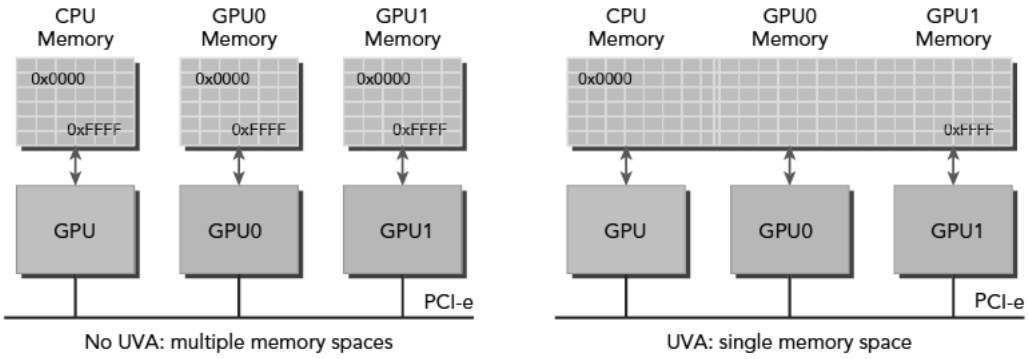

Figura 2.7: Unified Virtual Addressing (Cheng *et al.*, 2014)

.

2.5 Técnicas de Amostragem para Grande Volume de Dados

A última e não menos importante abordagem investigada está relacionada a utilização de técnicas de amostragem em dados com alto volume e dimensão.

Uma coleção de grande volume de dados é criada através da agregação de uma série de conjunto de dados com diferentes subpopulações denominada heterogeneidade de dados (Fan *et al.*, 2014). Cada subpopulação exibe uma característica que não é compartilhada entre as demais. Quando se avalia conjuntos moderados de dados, a heterogeneidade presente na subpopulação, normalmente, é caracterizada como *outliers* e a modelagem do comportamento dessa informação é dificultada em função da escassez de exemplos (Fan *et al.*, 2014).

No entanto, quando se trabalha com uma enorme quantidade de dados, tal heterogeneidade pode ser melhor compreendida através da exploração de algumas análises que anteriormente não eram possíveis.

Dados com alta dimensão e volume também apresentam algumas dificuldades de interpretação. A primeira está relacionada a acumulação de ruídos. Uma

vez que os modelos de predição para *Big Data* envolvem o ajuste simultâneo de diversos parâmetros, a acumulação de ruídos em cada um desses parâmetros pode acarretar a geração de um modelo incorreto (Fan *et al.*, 2014).

Outro ponto a ser analisado é a correlação espúria entre os dados. O fato de uma ou mais variáveis oscilarem de forma parecida não significa que uma cause a outra. Nesse caso, a correlação entre as variáveis envolvidas é espúria, devido apenas a coincidência (Jackson & Somers, 1991). Dados originados de *Big Data* indicam alta probabilidade de ocorrência de correlações espúrias devido ao enorme tamanho do conjunto de dados (Gandomi & Haider, 2015).

2.5.1 *Chunking Average*

Norman Matloff apresenta uma técnica denominada *Chunking Average* (Matloff, 2015). Referida técnica consiste em particionar o dado em r *chunks*. A primeira observação ($k = \frac{n}{r}$), corresponde ao primeiro *chunk*, e as próximas k observações formam o segundo *chunk* e assim sucessivamente.

Após o particionamento aplica-se uma função de máxima verossimilhança para cada *chunk*. Por fim, calcula-se a média dos r resultados obtidos no passo anterior para produzir o estimador($\hat{\theta}$) de θ.

Os resultados encontrados pelo autor apontam que o método apresenta boa acurácia e uma alta escalabilidade, uma vez que seu processamento pode ser distribuído entre várias máquinas com diversos *cores*.

2.5.2 *Bag of Little Bootstraps*

Outra técnica estatística utilizada em problemas de *Big Data* é a *Bag of Little Bootstraps* (Kleiner *et al.*, 2014). O método funciona da seguinte forma: seleciona-se os s subconjuntos de tamanho b, e de forma aleatória é realizada uma amostragem do conjunto de dados ($\mathcal{I}_j = \{i_1, \ldots, i_b\}$ de $\{1, 2, \ldots, n\}$), sem substituição, por s vezes. Tal amostragem é baseada na distribuição multinomial, ou seja, na frequência do padrão.

Para cada s subconjuntos de dados, deve-se repetir o procedimento de amostragem r vezes ($k = 1, \ldots, r$) e calcular o estimador $\hat{\theta}_{j,k}$ *(MLE)*. Após o cálculo do estimador é checado o erro padrão ($\frac{\sigma}{\sqrt{n}}$) de $\hat{\theta}$ da iteração r do conjunto de dados

s_j $\xi_j^* = \mathrm{SD}\left\{\hat{\theta}_{j,1}^*, \ldots, \hat{\theta}_{j,r}^*\right\}$. Por fim, calcula-se a média baseada no erro padrão de cada subconjunto s, $\xi_1^*, \ldots, \xi_s^*$, com o objetivo de obter o desvio padrão do estimador $\hat{SD}(\hat{\theta}) = \frac{1}{s}\sum_{j=1}^{s} \xi_j^*$.

Assim como no *Chunking Average*, a técnica *Bag of Little Bootstraps* permite que múltiplas amostras do dado sejam processadas e transferidas entre os nós de computação. Além disso, o autor prova que para um dado independente e identicamente distribuído é possível produzir um estimador $\hat{\theta}$ que é assintoticamente equivalente a θ.

Capítulo 3

Abordagens Propostas

3.1 *GPU Power Iteration Clustering*

O autor do método PIC (Lin & Cohen, 2010) disponibilizou a implementação original do PIC na linguagem de programação MATLAB (Lin & Cohen., 2015). De posse dessa implementação foi realizado um experimento para avaliar o comportamento do PIC para uma quantidade moderada de dados. Foram utilizados dois conjuntos de dados sintéticos que representam o problema das duas luas e dos três círculos com o seguinte tamanho n (15.0000, 30.000 e 45.000).

Os resultados encontrados no experimento indicam que o principal gargalo do PIC (Lin & Cohen, 2010) é o cálculo da matriz de afinidades para os dados de entrada (O(n^2) no pior caso). Os resultados encontrados são apresentados na tabela 3.1.

Base de Dados	n	Matriz de Afinidades	PIC Total
Duas Luas	15.000	2.644,52	2.644,52
Três Círculos	15.000	2.194,27	2.363,64
Duas Luas	30.000	21.924,33	23.173,62
Três Círculos	30.000	18.064,53	21.091,17
Duas Luas	45.000	99.857,83	91.239,62
Três Círculos	45.000	62.228,62	84.977,15

Tabela 3.1: Tempo de execução do método PIC em segundos

Para solucionar o gargalo identificado foi desenvolvido o método GPIC (*GPU Power Iteration Clustering*) Algoritmo **??**, uma implementação baseada em GPU do método PIC. O propósito do GPIC é criar um conjunto de *Kernels CUDA* que são lançados um após o outro com o objetivo de executar diferentes etapas do PIC original, reduzindo o tempo de execução e mantendo a mesma qualidade obtida pelo método original. A implementação proposta do GPIC é apresentada a seguir.

Algorithm 2 GPU Power Iteration Cluster

Input:

A	Matriz de afinidades
k	Quantidade de agrupamentos
v_0	Vetor inicial
nr	Número de linhas de A
nc	Número de colunas de A
ϵ	Precisão desejada

Output:

C	Agrupamentos resultantes $C_1, C_2, ..., C_k$

$d \leftarrow h(data)$ ▷ Copia os dados do *host* para o *device*
$A \leftarrow$ **AffinityMatrix**$(data, nr, nc)$ ▷ Calcula a matriz de afinidades A
$D \leftarrow$ **RowSum**(A, nr) ▷ Soma as linhas da matriz A
$W \leftarrow$ **NormMat**(A, D, nr) ▷ Normaliza a matriz A
$v_0 \leftarrow$ **Reduction**(D, nr) ▷ Realiza a soma o vetor D
$v_t \leftarrow$ **Norm**(D, v_0, nr) ▷ Normaliza o vetor D
$\delta_0 \leftarrow v_0$ ▷ Inicializa a variação da variação do autovetor
for $t = 0, 1, 2, ...$ **do**
    $v_{t+1} \leftarrow$ **Multiply**(v_t, W, nr) ▷ Multiplica a matriz W pelo vetor v_t
    $\tau \leftarrow$ **Reduction**(v_{t+1}, nr) ▷ Realiza a soma do vetor v_{t+1}
    $v_{t+1} \leftarrow$ **Norm**(v_{t+1}, τ, nr) ▷ Atualiza a estimativa dos autovetores de W
    $\delta^{t+1} \leftarrow |v_{t+1} - v_t|$ ▷ Atualiza a variação do autovetor anterior e o atual
    if $|\delta_{t+1} - \delta_t| \leq \epsilon$ **then** ▷ Critério de parada: convergência do autovetor
        break ▷ Finaliza a estimativa dos autovetores
    end if
end for
$C \leftarrow k$-médias de v_t ▷ Agrupamento dos autovetores
return C

3.1.1 *AffinityMatrix Kernel*

O primeiro *kernel* ***AffinityMatrix***(X, nr, nc) 3 é lançado com o objetivo de calcular a matriz de afinidades A através dos dados de entrada. O cálculo da matriz de afinidades é realizado através da função $s(x_i, x_j) = s(x_j, x_i)$, onde $i! = j$ e $s = 0$ quando $i = j$. A função de distância utilizada é a distância euclidiana apresentada na tabela 2.2.

A matriz A é uma matriz quadrada simétrica de tamanho n, sendo que n é o tamanho da entrada. O *kernel* lançado possui n *threads* e cada *thread* fica responsável por calcular uma linha da matriz A. Dessa forma, cada elemento da matriz A é indexado a partir do *id* de cada *thread* e do índice corrente i.

Algorithm 3 *AffinityMatrix Kernel*

Input:

X	Matriz de dados multivariada
nr	Número de linhas
nc	Número de colunas

Output:

A	Matriz de afinidades

1: $tid \leftarrow blockIdx * blockDim + threadIdx$
2: $v \leftarrow 0$
3: $A \leftarrow \{\}$
4: **for** $i < nr$ **do**
5: **if** $tid == i$ **then**
6: $A[(i * nr) + tid] = 0;$
7: **end if**
8: **if** $i > tid$ **then**
9: **for** $j < nc$ **do**
10: $v+ \leftarrow (X[(tid * nc) + j])$ - $(X[(i * nc) + j])^2$
11: **end for**
12: $v \leftarrow \sqrt{v};$
13: $A[(tid * nr) + i] \leftarrow v;$
14: $A[(i * nr) + tid] \leftarrow v;$
15: $v \leftarrow 0;$
16: **end if**
17: **end for**
18: **return** A

3.1.2 *RowSum Kernel*

O segundo *kernel* ***RowSum****(*A, nr*)* 4 é lançado com o objetivo de somar as linhas da matriz de afinidades. O resultado dessa operação é armazenado no vetor D.

Algorithm 4 *RowSum Kernel*

Input:

A Matriz de afinidades

nr Número de linhas

Output:

D Vetor com a soma das linhas de A

1: $tid \leftarrow blockId * blockDim + threadId$
2: $v \leftarrow 0$
3: $D \leftarrow \{\}$
4: **if** $tid < nr$ **then**
5: **for** $i < nr$ **do**
6: $v \mathrel{+}\leftarrow A[(tid * nr) + i];$ ▷ Soma linhas de A
7: **end for**
8: $D[tid] \leftarrow v;$
9: **end if**
10: **return** D

3.1.3 *NormalizeMatrix Kernel*

Um terceiro *kernel* ***NormalizeMatrix(****A, D, nr****)* 5 é lançado com o objetivo de normalizar a matriz de afinidades A. Os resultados obtidos são armazenados na matriz W.

Algorithm 5 *NormalizeMatrix Kernel*

Input:

A	Matriz de afinidades
D	Vetor com a soma das linhas de A
nr	Número de linhas de A

Output:

W	Matriz de afinidades normalizada

1: $tid \leftarrow blockId * blockDim + threadId$
2: $W \leftarrow \{\}$
3: **if** $tid < nr$ **then**
4: **for** $i < nr$ **do**
5: $W[(tid * nr) + i] \leftarrow \frac{A[(tid*nr)+i]}{D[tid]}$ ▷ Normaliza A
6: **end for**
7: **end if**
8: **return** W

3.1.4 *Reduction Kernel*

O quarto *kernel* ***Reduction(****D****,*** *nr****)*** 6 é lançado para calcular o volume. A operação de cálculo do volume é descrita através da soma do vetor D. No passo 10 do GPIC, esse mesmo *kernel* é lançado novamente e o resultado é armazenado na variável τ.

Algorithm 6 *Reduction Kernel*

Input:

D Vetor com a soma das linhas de A ou v_t vetor com a soma das linhas de v_{t+1}

nr Número de linhas de A

Output:

v_0 Vetor inicial

1: $tid \leftarrow threadId$
2: $i \leftarrow blockId * (blockDim * 2) + threadId$
3: $sData \leftarrow SharedMemory$
4: **if** $i < nr$ **then**
5: $\quad \tau \leftarrow D[i]$
6: **else**
7: $\quad \tau \leftarrow 0$
8: **end if**
9: **if** $i + blockDim < n$ **then**
10: $\quad \tau+ \leftarrow D[i + blockDim]$
11: **end if**
12: $sData[tid] \leftarrow \tau$
13: $syncthreads()$
14: **for** $s = \frac{blockDim}{2}; s > 0; s >>= 1$ **do**
15: $\quad$ **if** $tid < s$ **then**
16: $\quad\quad \tau \leftarrow sData[tid + s]$
17: $\quad\quad sData[tid] \leftarrow \tau$
18: $\quad$ **end if**
19: $\quad syncthreads()$
20: **end for**
21: **if** $tid == 0$ **then**
22: $\quad D[blockId] = \tau$
23: **end if**
24: **return** v_0

3.1.5 *Norm Kernel*

O quinto *kernel* ***Norm***(v_{t+1}, τ, nr) 7 é lançado com o objetivo de normalizar o vetor D, e os resultados encontrados são armazenados no vetor v_t. No passo 11 do GPIC, esse mesmo *kernel* é lançado novamente com o objetivo de normalizar os novos valores de v_t.

Algorithm 7 *Norm Kernel*

Input:

v_{t+1}	Vetor com a soma das linhas de A
τ	Soma do vetor v_{t+1}
nr	Número de linhas de v_{t+1}

Output:

v_{t+1}	Vetor inicial

1: $tid \leftarrow blockId * blockDim + threadIdx$
2: **if** $tid < nr$ **then**
3: $\quad v_{t+1}[tid] \leftarrow \frac{v_{t+1}[tid]}{\tau}$ ▷ Normaliza v_t
4: **end if**
5: **return** v_{t+1}

3.1.6 *Multiply Kernel*

O sexto e último *kernel* ***Multiply***(v_t, W, nr) 8 é lançado para multiplicar a matriz W pelo vetor vt.

Algorithm 8 *Multiply Kernel*

Input:

v_t Vetor normalizado D

W Matriz normalizada A

nr Número de linhas de v_{t+1}

Output:

v_{t+1} Vetor inicial

1: $tid \leftarrow blockId * blockDim + threadId$
2: $a \leftarrow 0$
3: **if** $tid < nr$ **then**
4: **for** $i < rows$ **do**
5: $a \leftarrow a + W[(tid * nr) + i] * v_t[i]$ ▷ Multiplica W por v_t
6: **end for**
7: $v_{t+1}[tid] \leftarrow a$
8: **end if**
9: **return** v_{t+1}

Capítulo 4

Experimentos e Resultados

4.1 Problemas Artificiais

Neste capítulo serão apresentados os resultados para diversos experimentos. Com o objetivo de verificar o método de agrupamento proposto, novos experimentos foram conduzidos com as duas bases de dados já mencionadas anteriomente. Os resultados encontrados são apresentados na tabela 4.1 e indicam um ganho de desempenho muito significativo para o algoritmo GPIC que, em média, foi 1471 vezes mais rápido que o PIC original.

Base de Dados	n	G-PIC	PIC
Duas Luas	15.000	3,99	2.644,52
Três Círculos	15.000	4,03	2.644,52
Duas Luas	30.000	18,00	23.173,62
Três Círculos	30.000	18,03	24.226,98
Duas Luas	45.000	45,07	109.644,50
Três Círculos	45.000	45,90	112.247,64

Tabela 4.1: Comparação entre os métodos de agrupamentos G-PIC e PIC em segundos

4.2 Problemas Reais

Capítulo 5

Conclusão e Trabalhos Futuros

5.1 Conclusão

Dessa forma, este trabalho teve por objetivo

5.2 Trabalhos Futuros

Desta forma, pode-se elencar os seguintes passos em trabalhos futuros:

Referências

Aggarwal, C.C. & Reddy, C.K. (2013). *Data clustering: algorithms and applications*. CRC Press. 5

Agrawal, R., Gehrke, J., Gunopulos, D. & Raghavan, P. (1998). *Automatic subspace clustering of high dimensional data for data mining applications*, vol. 27. ACM. 11

Andrade, G., Ramos, G., Madeira, D., Sachetto, R., Ferreira, R. & Rocha, L. (2013). G-dbscan: A gpu accelerated algorithm for density-based clustering. *Procedia Computer Science*, **18**, 369–378. 16

Ankerst, M., Breunig, M.M., Kriegel, H.P. & Sander, J. (1999). Optics: ordering points to identify the clustering structure. In *ACM Sigmod Record*, vol. 28, 49–60, ACM. 8

Berman, J.J. (2013). Introduction. In J.J. Berman, ed., *Principles of Big Data*, xix – xxvi, Morgan Kaufmann, Boston. 4

Bezdek, J.C., Ehrlich, R. & Full, W. (1984). Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, **10**, 191–203. 10, 12

Bi, Z., Da Xu, L. & Wang, C. (2014). Internet of things for enterprise systems of modern manufacturing. *Industrial Informatics, IEEE Transactions on*, **10**, 1537–1546. 1

Celeux, G. & Soromenho, G. (1996). An entropy criterion for assessing the number of clusters in a mixture model. *Journal of classification*, **13**, 195 212. 8

Chen, C.P. & Zhang, C.Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, **275**, 314–347. 3

Chen, M., Mao, S. & Liu, Y. (2014). Big data: A survey. *Mobile Networks and Applications*, **19**, 171–209. 4

Chen, W.Y., Song, Y., Bai, H., en Lin, C.J. & Chang, E.Y. (2011). Large-scale spectral clustering with mapreduce and mp1. *Scaling up Machine Learning: Parallel and Distributed Approaches*. 15

Cheng, J., Grossman, M. & McKercher, T. (2014). *Professional Cuda C Programming*. John Wiley & Sons. vi, 18, 21, 22, 23

Davis, K. (2012). *Ethics of Big Data: Balancing risk and innovation*. "O'Reilly Media, Inc.". 3

Dean, J. & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, **51**, 107–113. vi, 14, 15

Dempster, A.P., Laird, N.M. & Rubin, D.B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1–38. 11, 12

Dhillon, I.S., Guan, Y. & Kulis, B. (2004). Kernel k-means: Spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, 551–556, ACM, New York, NY, USA. 10

Dong, J., Wang, F. & Yuan, B. (2013). Accelerating birch for clustering large scale streaming data using cuda dynamic parallelism. In *Intelligent Data Engineering and Automated Learning–IDEAL 2013*, 409–416, Springer. 16

Dong, X.L. & Srivastava, D. (2013). Big data integration. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, 1245 1248, IEEE. 3

Ester, M., Kriegel, H.P., Sander, J. & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, vol. 96, 226–231. 8

Everitt, B.S., Landau, S., Leese, M. & Stahl, D. (2011). *Cluster Analysis, 5th Edition*, i xv. John Wiley Sons Ltd. 6

Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A.Y., Foufou, S. & Bouras, A. (2014). A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *Emerging Topics in Computing, IEEE Transactions on*, **2**, 267–279. vii, 2, 11, 12

Fan, J., Han, F. & Liu, H. (2014). Challenges of big data analysis. *National science review*, **1**, 293–314. 23, 24

Farivar, R., Rebolledo, D., Chan, E. & Campbell, R.H. (2008). A parallel implementation of k-means clustering on gpus. In *PDPTA*, vol. 13, 212–312. 16

Fatica, M., LeGresley, P., Buck, I., Stone, J., Phillips, J., Morton, S. & Micikevicius, P. (2008). High performance computing with cuda. *SC08*. 16

Fraley, C. & Raftery, A.E. (1998). How many clusters which clustering method answers via model based cluster analysis. *The computer journal*, **41**, 578 588. 8

Fukunaga, K. & Hostetler, L.D. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, **21**, 32–40. 8

Gan, G., Ma, C. & Wu, J. (2007). *Data clustering: theory, algorithms, and applications*, vol. 20. Siam. 5, 6

Gandomi, A. & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, **35**, 137–144. 4, 24

Ganti, V., Gehrke, J. & Ramakrishnan, R. (1999). Cactus clustering categorical data using summaries. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 73–83, ACM. 9

Girolami, M. (2002). Mercer kernel-based clustering in feature space. *Neural Networks, IEEE Transactions on*, **13**, 780–784. 10

Guha, S., Rastogi, R. & Shim, K. (1998). Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD Record*, vol. 27, 73–84, ACM. 8

Guha, S., Rastogi, R. & Shim, K. (1999). Rock: A robust clustering algorithm for categorical attributes. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, 512–521, IEEE. 8, 9

Halkidi, M., Batistakis, Y. & Vazirgiannis, M. (2001). On clustering validation techniques. *Journal of Intelligent Information Systems*, **17**, 107–145. 7

Han, J., Kamber, M. & Pei, J. (2011). *Data mining: concepts and techniques: concepts and techniques*. Elsevier. 5

Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A. & Khan, S.U. (2015). The rise of big data on cloud computing: review and open research issues. *Information Systems*, **47**, 98–115. 3

Hathaway, R.J. & Bezdek, J.C. (2006). Extending fuzzy and probabilistic clustering to very large data sets. *Computational Statistics and Data Analysis*, **51**, 215 – 234, the Fuzzy Approach to Statistical Analysis. 4

Hinneburg, A. & Keim, D.A. (1998). An efficient approach to clustering in large multimedia databases with noise. In *KDD*, vol. 98, 58–65. 8, 12

Hinneburg, A. & Keim, D.A. (1999). Optimal grid clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. 11, 12

Hitzler, P. & Janowicz, K. (2013). Linked data, big data, and the 4th paradigm. *Semantic Web*, **4**, 233 235. 3

Huang, Z. (1997). A fast clustering algorithm to cluster very large categorical data sets in data mining. In *DMKD*, 0, Citeseer. 8

Huang, Z. & Ng, M.K. (1999). A fuzzy k-modes algorithm for clustering categorical data. *Fuzzy Systems, IEEE Transactions on*, **7**, 446–452. 10

Information, O. (2015). Understanding the 7 vs of big data. 4

Jackson, D. & Somers, K. (1991). The spectre of spurious correlations. *Oecologia*, **86**, 147–151. 24

Karypis, G., Han, E.H. & Kumar, V. (1999). Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, **32**, 68–75. 8, 9

Keim, D., Qu, H. & Ma, K.L. (2013). Big data visualization. *Computer Graphics and Applications, IEEE*, **33**, 20 21. 4

Kirk, D.B. & Wen-mei, W.H. (2012). *Programming massively parallel processors: a hands-on approach*. Newnes. vi, 16, 17, 19, 20, 21

Kleiner, A., Talwalkar, A., Sarkar, P. & Jordan, M.I. (2014). A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **76**, 795–816. 24

Kohonen, T. (1998). The self organizing map. *Neurocomputing*, **21**, 1–6. 11

Krishnan, K. (2013). *Data warehousing in the age of big data*. Newnes. 3

Lanczos, C. (1950). *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office. 9

Li, X. & Fang, Z. (1989). Parallel clustering algorithms. *Parallel Computing*, **11**, 275–290. 13

Lin, F. & Cohen, W.W. (2010). Power iteration clustering. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, 655–662. 9, 26

Lin, F. & Cohen., W.W. (2015). Power iteration clustering. 26

MacQueen, J. *et al.* (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, 281–297, Oakland, CA, USA. 8

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. & Byers, A.H. (2011). Big data: The next frontier for innovation, competition, and productivity. 3

Matloff, N. (2015). *Parallel Computing for Data Science: With Examples in R, C++ and CUDA*. Chapman & Hall/CRC The R Series, CRC Press. 24

Ng, A.Y., Jordan, M.I., Weiss, Y. *et al.* (2002). On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, **2**, 849–856. 9

Ng, R.T. & Han, J. (2002). Clarans: A method for clustering objects for spatial data mining. *Knowledge and Data Engineering, IEEE Transactions on*, **14**, 1003–1016. 8

Nickolls, J., Buck, I., Garland, M. & Skadron, K. (2008). Scalable parallel programming with cuda. *Queue*, **6**, 40–53. 18

O Reilly Media, I. (2015). *Big Data Now: 2014 Edition*, vol. 1. O Reilly Media. 3, 4

O'Leary, D. (2013). Artificial intelligence and big data. *Intelligent Systems, IEEE*, **28**, 96–99. 4

Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E. & Phillips, J.C. (2008). Gpu computing. *Proceedings of the IEEE*, **96**, 879–899. 15

Park, H.S. & Jun, C.H. (2009). A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, **36**, 3336–3341. 8

Reeve, A. (2013). *Managing Data in Motion: Data Integration Best Practice Techniques and Technologies*. Newnes. 3, 4

Sheikholeslami, G., Chatterjee, S. & Zhang, A. (1998). Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB*, vol. 98, 428–439. 11

Shirkhorshidi, A., Aghabozorgi, S., Wah, T. & Herawan, T. (2014). Big data clustering: A review. In B. Murgante, S. Misra, A. Rocha, C. Torre, J. Rocha, M. FalcÃ£o, D. Taniar, B. Apduhan & O. Gervasi, eds., *Computational Science and Its Applications ICCSA 2014*, vol. 8583 of *Lecture Notes in Computer Science*, 707–720, Springer International Publishing. 13

Tibshirani, R., Walther, G. & Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **63**, 411 423. 8

Van Rijmenam, M. (2014). *Think Bigger: Developing a Successful Big Data Strategy for Your Business*. AMACOM Div American Mgmt Assn. 3

Wang, W., Yang, J., Muntz, R. *et al.* (1997). Sting: A statistical information grid approach to spatial data mining. In *VLDB*, vol. 97, 186–195. 11

Wilt, N. (2013). *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley. 19

Wu, R., Zhang, B. & Hsu, M. (2009). Clustering billions of data points using gpus. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, 1–6, ACM. 16

Wu, X., Zhu, X., Wu, G.Q. & Ding, W. (2014). Data mining with big data. *Knowledge and Data Engineering, IEEE Transactions on*, **26**, 97–107. 1

Zhang, T., Ramakrishnan, R. & Livny, M. (1996). Birch: Anecient data-clusteringmethod forvery large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada*. 8, 12

Zikopoulos, P., Eaton, C. *et al.* (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media. 3