http://journal.unoeste.br/index.php/ca/index
DOI: 10.5747/ce.2019.v11.n2.e274
ISSN on-line 2178-8332

Colloquium Exactarum

Submetido: 10/11/2018 Revisado: 08/02/2019 Aceito: 03/06/2019

GANHO DE DESEMPENHO DO FEMA UTILIZANDO PROGRAMAÇÃO PARALELA E ÁRVORES DE PARTICIONAMENTO ESPACIAL

A PERFORMANCE IMPROVEMENT TO FEMA APPLYING PARALLEL PROGRAMMING AND BINARY PARTITION SPACE TREE

Carlos Adriano Miranda; Silvio Carro; Danillo Roberto Pereira

Universidade do Oeste Paulista - UNOESTE
Faculdade de Informática de Presidente Prudente – FIPP
E-mail: adriadrianooo@gmail.com; silvio@unoeste.br; danilopereira@unoeste.br

RESUMO – O presente estudo apresenta a utilização de estruturas de dados e GPU como uma melhoria de desempenho do algoritmo de classificação FEMa. Primeiramente, à partir de um *datasets* é criada uma árvore de partição binária do tipo *Kd-Tree* e após sua construção, aplicado o algoritmo de busca dos K vizinhos mais próximos (K-NN) na Kd-Tree para cada amostra de teste apresentada na fase de classificação. Após ter o resultado da busca das amostras mais próximas, é feita a etapa de classificação do FEMa aplicando uma base dos Métodos dos Elementos Finitos (FEM), para trazer o resultado. Outra abordagem é utilizar códigos CUDA no algoritmo do FEMa, para que o mesmo seja paralelizado e executado em GPU's, para obter um ganho de desempenho no tempo de execução.
Palavras-chave: FEMa, GPU, Kd-Tree, K-NN.

ABSTRACT – This paper presents an application with data structures and GPU to get better performances in FEMa algorithm. At first, a binary partition Kd-Tree is constructed from a dataset, after his building, the search algorithm of the K nearest neighbours (K-NN) is applied in the Kd-Tree to all sample in the test dataset. After get the result of nearest samples search, the step of classification begin applying the Finite Element Method basis to get the result. Another approach is to utilize cuda codes in algorithm, so that it can be parallelized and run in GPU to obtain a gain of performance in the code runtime.
Keywords: FEMa, GPU, Kd-Tree, K-NN.

1. INTRODUÇÃO

O FEMa (*Finite Element Machine*) é um framework utilizado para fazer classificações por meio de métodos matemáticos, baseado no Método de Elementos Finitos (FEM - *Finite Element Method*), que consiste em dividir o problema em pequenos pedaços, e para cada pedaço (base) escrever uma simples equação, que juntos descrevem o problema por completo.

Para fazer essa classificação, o FEMa trabalha com os métodos de interpolação de bases. Abordando superficialmente o assunto, essas bases formadas são interpoladas (função que irá descrever o problema) e para cada caso de teste, é obtido um vetor de probabilidade, em que cada posição representa uma classe. Dependendo de como foi feito na etapa de treinamento, essa classificação pode assumir possibilidades variadas para qual classe a amostra se encontra e isso possibilita ter ideias diferentes sobre o mesmo problema, sendo útil principalmente para diagnósticos médicos que podem ter opiniões divergentes [PEREIRA, D. R. et al].

Quando utilizado em grandes base de dados (big datas), exibe um grande diferencial, pois dependendo da base de funções em que irá trabalhar e analisar os dados, pode não ser necessário usar parâmetros, assumindo que todas as bases já são interpoladoras.

Possui características que permitem ser utilizado em *big datas*, portanto é essencial que tenha uma organização eficiente das amostras. Estruturas de dados do tipo árvores de particionamento espacial podem solucionar esse problema de uma maneira muito eficiente.

O tipo de árvore que será utilizado no aprimoramento do FEMa é a *Kd-Tree* (multidimensional binary search tree) uma generalização da *Binary Space Partitioning Trees* (BSP).

Essa estrutura apresentada trabalha em um ambiente espacial, então assim chamada de Spatial Data Structures (SAMET, 1994), que consiste em guardar os dados como se fossem objetos que estão flutuando em um plano. Para que isso ocorra e esses objetos possam ser encontrados, o plano vai ser sempre dividido, até que para cada plano tenha apenas um objeto. A KD-Tree subdivide o espaço utilizando somente de divisões horizontais e verticais, não importando o tamanho dimensional do espaço (NAYLOR, 2005). Para uma otimização da busca de dados, pode ser implementado na mesma o algoritmo dos vizinhos mais próximos percorrendo o menor número de nós possíveis (MUJA; LOWE, 2009).

Para ter um ganho de desempenho, foi aplicado a paralelização do código através do uso de GPU (Graphics Processing Unit), com a biblioteca CUDA (Compute Unified Device Architecture), pois o FEMa apresenta ótimos resultados em big datas, podendo assim melhorar seu desempenho.

2. TRABALHOS RELACIONADOS

Existem vários autores que já aplicaram o uso de *Kd-Trees, Nearest Neighbours* e GPU em *machine learning* (ML) para melhoria de desempenho ou para estudo de comparações.

No experimento de Jiang e Hallstrom (2013), foi usado a técnica de *Kd-Tree* com busca de *Nearest Neighbours* comparada com a técnica *Bayesian Classifier* para classificação do movimento humano com sensores. Como resultado apresentado, as duas técnicas tiveram acurácia entre 80% e 85%, porém, com um dos testes a *Kd-Tree* conseguiu superar o outro método e obter 90%.

De acordo com os experimentos realizados por Garcia, Debreuve e Barlaud (2008), na tentativa de acelerar o algoritmo de *K-NN* com o uso de GPU, obtiveram sucesso, pois seu algoritmo conseguiu ser 120 vezes mais rápido que o algoritmo *BF-Matlab* e um mínimo de 40 vezes do algoritmo *KDT-C.* Shamonin et al. (2013) diz que com seus experimentos em obter um

speed-up do algoritmo *Gaussian Pyramid* para geração de imagens, obtendo sucesso com um acréscimo de duas vezes na velocidade de processamento. Com esses experimentos, podem ser visto que o poder computacional proporcionado com o uso de GPU nos algoritmos.

A principal ideia para esse artigo é utilizar essas tecnologias que já demonstraram ótimos resultados e aplicar no algoritmo do FEMa, para que o mesmo possa ter um ganho em desempenho na execução com a GPU, também provar ser possível a melhoria da acurácia e desempenho com a aplicação dos algoritmos denominados *Kd-Tree* e *K-NN.*

3. CONCEITOS FUNDAMENTAIS

Nesta seção é apresentada a fundamentação teórica sobre os métodos utilizados para o desenvolvimento deste trabalho.

3.1 Método dos Elementos Finitos

Para um melhor entendimento e esclarecimento de como o FEMa funciona, será explicado de forma breve o algoritmo FEM.

Considere D e V como um conjunto infinito, e F : D → V é uma função que contém todo o mapeamento do espaço. Esse espaço F não pode ser representado por um elemento genérico nos computadores, então deve - se achar uma função genérica Γ por meio de aproximação. A qualidade da aproximação pode ser definida por $|\Gamma$ - F|, onde |.| é o erro de aproximação (PEREIRA, 2009).

3.1.1 Base de aproximação

Uma base de aproximação de um espaço vetorial pode ser dada através de uma lista $\varphi = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$que contém funções linearmente independentes do espaço V, onde todo $v \in V$e pode ser obtido por uma combinação linear, expressa por $v = \Sigma_{i=1}^{n} ai\varphi i$, com os coeficientes $a = \{a_1, a_2, \ldots, a_n\}$ pertencendo aos $\Re$. Então, Γpode ser representado nos computadores pelos seus n coeficientes.

3.1.2 Interpolação

Com um conjunto de pontos $X = \{x_1, x_2, \ldots, x_n\}$tal que $X \subset D$e seus valores associados $Y = \{y_1, y_2, \ldots, y_n\}$, sendo eles $Y \subset V$, devemos então achar uma função aproximada Γ que interpole todos os pares $\{x_i, y_1\}$.

Para poder descrever a função Γ pela base φ, temos então que achar os coeficientes a_i tal que $\Gamma(x_i) = \Sigma_{j=1}^{n} a_j\varphi_j(x_i) = y_i, \forall i \in \{1,2,\ldots,n\}$.Isso quer dizer que todo $y_i \in Y$é gerado de uma combinação linear com todas as funções das bases junto com seus coeficientes. É equivalente resolver o sistema linear $Za = y$, sendo Z uma matriz $n\,x\,n$ que contém a influência de cada elemento φ da base no seu ponto x_j, sendo assim $Z_{ij} = \varphi_i(x_j)$.

3.1.3 Bases interpoladoras

Uma base φ só pode ser interpoladora para o conjunto de pontos x, se ocorrer a seguinte situação:

$$\varphi_i(x_j) = \begin{cases} i\ se\ i = j \\ 0\ caso\ contrário \end{cases} \quad (1)$$

Então, a matriz Z se torna identidade, quando ocorre que $a_i = y_i, \forall_i \in \{1,2,\ldots,n\}$.

Portanto, com uma base de elementos finitos φ e um conjunto de elementos $X \in D$ podemos gerar uma base interpoladora ρ, sendo ela uma combinação linear de todos os elementos φ_i, dada por $\rho_i(x) = \sum_j U_{ij}\varphi_j(x)$(2) utilizando U como a matriz inversa de Z.

3.1.4 Bases partição da unidade

A base φ é somente uma partição da unidade, se e somente se $\varphi_i(x) \geq 0, \forall i\ e\ \forall x \in D$e $\Sigma_{i=1}^{n}\varphi_i(x) = 1, \forall x \in D$. Essa base oferece uma propriedade chamada de suavização, que consiste em corrigir suavidades. Essa propriedade pode ser obtida por:

$$a_{min} \geq \sum_{i=1}^{n} a_i\varphi_i(x) \geq a_{max} \quad (3)$$

sendo que a_{min} e a_{max}são respectivamente, o menor e o maior valor do conjunto de coeficientes de a. Então, para uma base φ com $\varphi_i(x) \geq 0$ obtemos sua base de partição através da seguinte fórmula:

$$\rho_i(x) = \frac{\varphi_i(x)}{\Sigma_{j=1}^{n} \varphi_j(x)} \quad (4)$$

3.1.6 *Shepard Basis*

Cada elemento φ_i é definido por uma fórmula geral

$$\varphi_i(x) = \frac{\omega(x,x_i)}{\Sigma_{j=1}^{n} \omega(x,x_i)} \quad (5)$$

em que ωé uma função não negativa, sendo que ela tende ao infinito quando o elemento x tende ao elemento x_i, isso demonstra que essa base é interpoladora e partição de unidade. A função ω é escolhida normalmente com uma potência de $k \geq 0$ da distância Euclidiano ao inverso

$$\omega(x, x_i) = \frac{1}{|x - x_i|^k} \quad (6)$$

3.2 Finite Element Machine for Fast Learning (FEMa)

O FEMa é um *framework*, para classificar *big datas*, e não necessitando de parâmetros e etapas de treinamentos que utilizam de *feedback*.(PEREIRA, D.R).

Seja T um conjunto de dados, primeiro o mesmo é particionado em outros dois T_1e T_2, sendo que $T = T_1 \cup T_2$, onde o conjunto de teste e treinamento será feito em T_1e o resto da análise em T_2. Assim sendo, o par $(x_i, y_i) \in T$ é o vetor de característica $x_i \in \Re^m$da amostra i, e $y_i$como rótulo.

Isso quer dizer que o FEMa aprende um conjunto de funções probabilísticas, com c representando a quantidade de classes obtidas e $P_i(x)$ a probabilidade de um determinado elemento pertencer a classe em questão.

O FEMa pode pular a etapa de treinamento dependendo da função de bases usadas. Quando as bases não são interpoladoras é necessário calcular a equação 2, entretanto, se essas bases não forem unidades de partição, então é a equação 4. Isso torna-se interessante, pois o FEMa pode trabalhar com qualquer tipo de base. Essas equações são consideradas como etapa de treinamento.

Assumindo que estamos trabalhando com bases interpoladoras e partição de unidade, dado um conjunto $x \in T_2$temos que computar a probabilidade para cada classe obtida, a partir da equação

$$P_i(x) = \sum_{j=1}^{|T_1|} p_i^j \varphi_j(x) \quad (7),$$

onde $p_i^j \in [0, 1]$contém a possibilidade da amostra de treino $j$estar contida na classe em questão, no caso, classe i. Essa propriedade é muito interessante, pois pode ser adotado um valor de probabilidade para alguma amostra, assim esse cálculo pode ser feito considerando um valor já estipulado.

Essa propriedade pode ser descrita usando a seguinte formulação:

$$p_i^j = \begin{cases} 1 \; se \; y_i = i \\ 0 \; caso \; contrário \end{cases} \quad (8)$$

desde que os *datasets* estejam rotulados podemos usar $p_i^j \in \{0, 1\}$, então, é gerado um conjunto de probabilidades $P(x)$ para cada amostra $x \in T_2$. Esses exemplos são atribuídos a classe μ que atendem a seguinte equação:

$$\mu = arg\, max_i P_i(x). \quad (9)$$

É dado a possibilidade de inferir uma certeza $C(x)$ na seguinte formulação:

$$C(x) = \frac{P_\mu(x)}{\sum_{j=1}^{c} P_j(x)}. \quad (10)$$

3.3 Kd-Tree

A Kd-Tree é uma generalização de árvores de particionamento espacial, com diferença que ela oferece cortes livres no plano, mas que são alinhados aos objetos e suas coordenadas, sendo cada corte em uma de suas dimensões alternadamente (FARIAS, 2006). É muito utilizada também em computação gráfica para *raytracing* (FOLEY; SUGERMAN, 2005) (WALD; HAVRAN, 2006) e para visão computacional junto com o algoritmo de *nearest neighbour* (MUJA; LOWE, 2009).

A partir da raiz, o algoritmo cria hiperplanos em cada dimensão, para poder efetuar os cortes e criar outros dois subplanos, a direita e à esquerda. Cada um

desses dois subplanos são recursivamente subdivididos até em casa plano tiver apenas uma amostra do conjunto (NAYLOR, 2005).

3.4 GPU (Graphics Processing Unit)

Aplicações que necessitam de muitos processamentos para chegar a um resultado final, podem ser implementadas utilizando tecnologias de GPU. Fazer uma aplicação em GPU consiste em dividir o código em várias *threads*, ou subprogramas, para que cada uma delas possa ser executada em paralelo, possibilitando que haja uma comunicação entre as partes, e que a memória utilizada seja compartilhada entre ambas (Vasconcellos et al, 2017).

A estrutura física de uma GPU consiste em vários multiprocessadores, que por sua vez contém vários processadores SIMD (Single Instruction Multiple Data) e cada processo irá executar a mesma operação (função) mas em um conjunto de dados diferentes para cada ciclo de *clock* (ZHANG; CHEN; WANG, 2010).

A biblioteca que possui recursos para esse tipo de programação é distribuída pela NVIDIA chamada CUDA (Compute Unified Device Architecture), pois é possível programar em linguagens de baixo nível com ela, como o C e C++ (AMARIS, et al., 2015), disponibilizada em uma biblioteca. Por permitir que o programador utiliza de linguagens baixo nível, é possível acessar e manipular o hardware da GPU sem a necessidade de bibliotecas externas (PRATX; XING, 2011).

3.5 K-NEAREST NEIGHBOURS

Esse algoritmo tem como objetivo buscar a partir de uma amostra, calcular a distância a partir de um raio *x* de busca, procurar os vizinhos mais próximos. Essa técnica é utilizada para classificação com base na quantidade de vizinhos de cada classe encontrados. A classe que tiver a maior quantidade de amostras é classificada (GARCIA, DEBREUVE e BARLAUD, 2008).

Para o uso na *Kd-Tree,* o algoritmo trabalha com um raio de busca na estrutura da árvore. Como a árvore tem uma organização espacial dos dados e com a ajuda do raio de busca, o algoritmo do KNN não precisa percorrer toda a estrutura e assim não percorre todos os dados presentes no *dataset*. Com essa abordagem a busca sempre converge para os dados que estão espacialmente mais próximos da amostra de teste.

4. METODOLOGIA

A ferramenta proposta foi desenvolvida em duas partes, na primeira foi incorporado no FEMa a estrutura de árvores espaciais do tipo *Kd-Tree*, utilizando a linguagem de programação C, junto do algoritmo de *K-NN*. Já na segunda parte, o FEMa foi modificado para poder ser paralelizado, compilado e executado na GPU, utilizando da linguagem de programação estendida derivada do C++, com a biblioteca CUDA, proveniente da NVIDIA, que já implementa algumas funcionalidades importantes para manipular a GPU.

A *Kd-Tree* foi desenvolvida de duas formas diferentes para efeito comparativo. Ambas as estruturas aqui apresentadas são construídas a partir da repartição de treino do *dataset* original. O primeiro algoritmo faz os cortes de hiperplano em cima das amostras no espaço, sendo assim, todos os nós que estão presentes na estrutura carregam amostras, não somente os nós folhas como o projeto original apresenta. O segundo algoritmo seguiu a maneira padrão de implementação da *Kd-Tree* proposto por Bentley (1975), o algoritmo faz cortes de hiperplano entre as amostras quando o plano contém um número par de amostras, caso seja ímpar, o corte é feito na coordenada da própria amostra, mas a mesma não é inserida na árvore, mas sim distribuída em uma das metades do plano dividido. Uma amostra só é inserida quando o plano contém a contagem de somente uma amostragem, sendo assim as amostras ficam somente nos nós folhas. Para os dois

algoritmos desenvolvidos, é utilizado a mediana das amostras analisadas para cada plano a ser dividido.

O algoritmo de *K-NN* foi implementado para poder fazer a busca nos dois algoritmos da *Kd-Tree*. A partir de uma amostra inserida no algoritmo e a respectiva árvore, o algoritmo faz a busca em profundidade fazendo a comparação a partir de um dado raio em volta da amostra de busca, retornando os *k* vizinhos mais próximos em uma lista encadeada.

O algoritmo do FEMa também teve que ser adaptado para permitir o uso dessas estruturas, retirando algumas funções do código em que a *Kd-Tree* e o *K-NN* substituem. A ideia com essas alterações foi também tentar diminuir o custo computacional do algoritmo e sua complexidade.

A partir de um raio de busca, o K-NN busca os vizinhos mais próximos da amostra na árvore. O algoritmo durante a busca guarda a maior e menor distância que obteve entre todas as percorridas e caso com o raio de busca não resulte em nenhum vizinho, o raio é redefinido pela média dos dois valores guardados.

A segunda parte da proposta foi desenvolvida para que o FEMa funcionasse em hardwares gráficos, especificamente na GPU. Para que isso fosse possível, todo o algoritmo teve que ser reformulado e algumas partes modificadas, utilizando a biblioteca CUDA.

Como a GPU é um hardware específico, foi necessário implementar de uma forma que o mesmo suportasse todos os processamento. Várias abordagens foram descartadas por trazer sobrecarga ao processamento ou ao uso de memória.

A introdução de novas estruturas no código como a *Linked List* (lista encadeada), foi uma possível solução, mas que com os testes e uso do algoritmo se tornou lenta para o propósito e foi descartada.

Partindo da ideia de Shamonin et al. (2013) em seu trabalho, o algoritmo trabalha com um quantidade menor de *threads*. Elas são iniciadas pela CPU e controladas na GPU em um *loop* principal, para que todas as amostras de teste sejam executadas. A quantidade de *threads* e blocos são informadas via parâmetro na execução, pois depende do *hardware*. As *threads* calculam a quantidade de vezes que a classificação vai ser executada, a partir da quantidade informada das mesmas e o número total de amostras de testes. Uma *thread* executa uma amostra em um intervalo de x *threads* sem ultrapassar o limite total.

Todo o processamento está contido em funções executadas pela GPU, menos o momento da leitura dos *datasets*, que são lidos e carregados para a memória da CPU, e somente após isso os dados são carregados para a memória compartilhada da placa gráfica, pois a biblioteca CUDA não oferece acesso para leitura de disco.

5 RESULTADOS

Os dois novos algoritmos construídos nesse projeto, o FEMa com uso de dois tipos de *Kd-Tree* e o FEMa com GPU, foram submetidos a testes em sete *datasets*, descrito abaixo na Tabela 1. Os testes foram executados no sistema operacional Linux, na distribuição Ubuntu 18.04 LTS, com um processador core i5 8ª geração, 8Gb de memória RAM equipado com uma placa gráfica NVIDIA Geforce MX150.

Tabela 1. *Datasets* utilizados para fazer testes com os algoritmos construídos.

Nome	Amostras	Características
Iris	145	3
Fem Dataset	214	9
Abalone	4177	8
Satélites	6435	28
Avila	20867	10
Chess (King-Rook vs. King)	28056	6
Bank Marketing	45.211	17

Fonte: Próprio autor

Os *datasets* foram particionados aleatoriamente entre treino e testes, com a parte de testes podendo ter entre 10% e 30% do conjunto total. Alguns *datasets* já estavam divididos quando obtidos. Todos os dados estão presentes na UCI Machine Learning (disponível com acesso livre no endereço: http://archive.ics.uci.edu/ml/index.php), somente o Fem Dataset foi obtido junto do algoritmo do FEMa, como sendo um conjunto de testes e parâmetro. A disposição dos dados está descrito na Tabela 2.

Tabela 2. Divisão dos *datasets* entre treino e teste.

Nome	Treino	Testes
Iris	100	45
Fem Dataset	105	109
Abalone	2923	1254
Satélites	4435	684
Avila	10430	10437
Chess (King-Rook vs. King)	19368	8417
Bank Marketing	37949	7263

Fonte: Próprio autor

Os resultados aqui obtidos, não se dizem respeito a acurácia do FEMa a outros métodos, mas somente na tentativa de melhorar a acurácia do mesmo, ou seu desempenho.

O raio base de busca do K-NN foram escolhidos arbitrariamente a partir de testes nos *datasets* para escolher o melhor parâmetro possível antes comparar os resultados, como também o número de vizinhos mais próximos. Ambos os parâmetros foram escolhidos a partir do momento que o aumento dos mesmo não surtia efeito, ou se fossem diminuídos o resultado era piorado. A quantidade de *threads* e blocos foram escolhidos para poderem ser executados na GPU de testes.

Os *datasets* estão dispostos na tabela 3 à 10, na mesma ordem presente na tabela 1 e 2. O algoritmo Kd-Tree 1 se refere a implementação com cortes de hiperplanos em cima das amostras, e o algoritmo Kd-Tree 2 com hiperplanos entre as amostras.

Tabela 3. Comparativo entre os algoritmos aplicados no Iris Dataset.

Algoritmo	Raio	Tempo /s	Acertos	Erros
FEMa	-	0,0134	44	1
FEMa + Kd-Tree 1	5	0,0091	44	1
FEMa + Kd-Tree 2	5	0,0124	44	1
FEMa + GPU	-	0,164	44	1

Fonte: Próprio autor

Tabela 4. Comparativo entre os algoritmos aplicados no Fem Dataset.

Algoritmo	Raio	Tempo /s	Acertos	Erros
FEMa	x	0,0208	70	39
FEMa + Kd-Tree 1	1	0,0104	70	39
FEMa + Kd-Tree 2	1	0,0066	70	39
FEMa + GPU	x	0,166	70	39

Fonte: Próprio autor

Para os *datasets* Íris e Fem, na Tabela 3 e 4, não foram utilizados nenhum parâmetros para nenhum algoritmo, somente um raio de busca previamente definido. A GPU foi utilizada com um total de 50 *threads*.

Tabela 5. Comparativo entre os algoritmos aplicados no dataset de Abalone.

Algoritmo	Raio	Tempo /s	Acertos	Erros
FEMa	-	0,769	612	642
FEMa + Kd-Tree 1	700	0,255	619	635
FEMa + Kd-Tree 2	10	0,160	642	612
FEMa + GPU	-	0,328	612	642

Fonte: Próprio autor

No dataset Abalone apresentado na Tabela 5, os parâmetros utilizados no FEMa, foram de 20 amostras por classe. Na primeira *Kd-Tree* foram utilizados no máximo 20 vizinhos e no segundo algoritmo no máximo 30. A GPU foi utilizada 40 *threads*.

Tabela 6. Comparativo entre os algoritmos aplicados no dataset Satélites.

Algoritmo	Raio	Tempo /s	Acertos	Erros
FEMa	-	1,244	189	111
FEMa + Kd-Tree 1	10500	0,363	139	161
FEMa + Kd-Tree 2	1	0,985	176	124
FEMa + GPU	-	7,805	172	129

Fonte: Próprio autor

Na Tabela 6, apresenta o dataset Satélites. Os parâmetros utilizados no FEMa, foi de 20 amostras por classe, e no primeiro algoritmo da *Kd-Tree* não foi utilizado parâmetros e no segundo algoritmo foram escolhidos no máximo 30 vizinhos. A GPU foi utilizada 40 *threads*.

Tabela 7. Comparativo entre os algoritmos aplicados no dataset de Ávila.

Algoritmo	Raio	Tempo /s	Acertos	Erros
FEMa	-	27,579	8136	2302
FEMa + Kd-Tree 1	1	7,566	7819	2619
FEMa + Kd-Tree 2	1	3,881	7599	2839
FEMa + GPU	-	-	-	-

Fonte: Próprio autor

Apresentado na Tabela 7, contém os resultados obtidos no dataset Ávila. Os parâmetros utilizados no FEMa, foi de 10 amostras por classe, e no primeiro algoritmo da *Kd-Tree* foi utilizado no máximo 20 vizinhos e no segundo algoritmo foram

escolhidos no máximo 30 vizinhos. A GPU foi utilizada 40 *threads*.

Tabela 7. Comparativo entre os algoritmos aplicados no dataset Chess (King-Rook vs. King).

Algoritmo	Raio	Tempo /s	Acertos	Erros
FEMa	-	43,379	6300	2117
FEMa + Kd-Tree 1	1	9,410	3930	4487
FEMa + Kd-Tree 2	1	1,918	6561	1856
FEMa + GPU	-	-	-	-

Fonte: Próprio autor

O *dataset* Chess (King-Rook vs. King), com os resultados na Tabela 7, usou como parâmetros para o algoritmo do FEMa, um número de 10 amostras por classe, para a Kd-Tree 1 e 2 um máximo de 50 e 25 vizinhos, respectivamente.

Tabela 8. Comparativo entre os algoritmos aplicados no dataset Bank Marketing

Algoritmo	Raio	Tempo /s	Acertos	Erros
FEMa	-	84,561	6678	585
FEMa + Kd-Tree 1	1000	19,361	6682	581
FEMa + Kd-Tree 2	800	76,204	6464	799
FEMa + GPU	-	-	-	-

Fonte: Próprio autor

O *dataset* Bank Marketing usou como parâmetros para o algoritmo do FEMa, um número de 20 amostras por classe, para a Kd-Tree 1 e 2 um máximo de 20 vizinhos, respectivamente. Os resultados podem ser vistos na Tabela 8.

Os *datasets* Avila, Chess (King-Rook vs. King) e Bank Marketing não tiveram seus resultados em GPU, pois as configurações da máquina e como é estruturado o algoritmo do FEMa, não permitiram que o mesmo fosse executado. Como dito na metodologia, o uso de *Linked List* que foi testado e solucionou alguns problemas de memória e concorrência, tornou o algoritmo lento e ineficiente, portanto, a versão de GPU apresentada tem pouca diferença da versão CPU, apenas mudando no fato de ser possível o teste de mais de uma amostra por vez.

Pode-se se notar que nos conjuntos de dados em que apresenta-se uma maior concentração de dados, o uso de *Kd-Tree* tornou o algoritmo mais rápido na sua construção e execução. Uma outra observação, se diz respeito ao tamanho do raio que foram utilizados na busca do KNN nas novas estruturas inseridas. Esse parâmetro que se torna essencial, se torna difícil de prever e o tornar padrão para todos os conjuntos de dados, pois depende da distribuição espacial dos dados no conjunto total de dados. O segundo algoritmo da *Kd-Tree* apresentou uma variância menor nesse quesito em comparação com o primeiro.

Apesar de os dois novos algoritmos tornarem o FEMa mais rápido e conseguir manter uma quantidade próxima de acertos/erros, o primeiro algoritmo se tornou mais eficiente. Porém, para os *datasets* Chess e Satélites, se demonstrou com uma acurácia muito pequena, se distanciando do ideal aqui proposto.

Uma observação que vale ressaltar, é que o FEMa também necessita de um parâmetro muito importante, como a quantidade de amostras por classe, que pode variar e trazer classificações diferentes.

Portanto, pode-se concluir que as modificações do FEMa com o uso das *Kd-Trees* podem ser utilizadas como um acréscimo ao algoritmo original, mas não como uma substituição, pois nem em todos os casos de testes apresentados conseguiram se apresentar melhores que o FEMa no principal quesito, a classificação. Como o algoritmo depende muito de um raio ideal para ter um ótimo resultado, a partir de um trabalho que utilizam de

datasets padronizados e que o parâmetro não se altera, ou muito pouco, a utilização da *Kd-Tree* pode se tornar viável.

6. CONSIDERAÇÕES FINAIS

O trabalho aqui apresentado teve duas abordagens comparativas entre as tecnologias. Apesar dos bons resultados, pode-se ainda complementar e/ou melhorar os algoritmos existentes. Deixo como trabalhos futuros, um estudo mais aprofundado de aplicabilidade de GPU no FEMa, com uso de linguagem de baixo nível, para poder trabalhar diretamente no hardware da placa gráfica.

Outro ponto é encontrar um padrão para o uso do FEMa com as *Kd-Trees*, onde a partir de um total de amostras essas novas estruturas se tornam viáveis, ou um total máximo, bem como pode ser influenciada pela quantidade de características.

Mesmo o K-NN mostrando divergência nas buscas de vizinhos mais próximos, dependendo muito do melhor raio de busca e máximo de vizinhos, provou ser um método eficiente junto da *Kd-Tree* que conseguiu diminuir o acesso aos dados e assim melhorar o tempo de execução do algoritmo.

REFERÊNCIAS

AMARIS, M. et al. **A Simple BSP-based Model to Predict Execution Time in GPU Applications.** 2015 Ieee 22nd International Conference On High Performance Computing (hipc), [s.l.], p.285-294, dez. 2015. IEEE. http://dx.doi.org/10.1109/hipc.2015.34.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. **Communications Of The ACM,** [s.l.], v. 18, n. 9, p.509-517, set. 1975. http://dx.doi.org/10.1145/361002.361007.

FARIAS, M. A. **Operações booleanas entre objetos delimitados por surfels usando constrained BSP-trees.** 2006. Dissertação (Mestrado) - Curso de Computação, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Rio Grande do Sul, 2006. Cap. 2. Disponível em: http://hdl.handle.net/10183/7081. Acesso em: 20 set. 2017.

FOLEY, T.; SUGERMAN, J. KD-tree acceleration structures for a GPU raytracer**.** In: ACM SIGGRAPH/ EUROGRAPHICS CONFERENCE ON GRAPHICS HARDWARE, 5., 2005, Los Angeles, Ca, Usa. **Proceeding.** Los Angeles, California: Acm, 2005. p. 15 - 22. Disponível em: https://graphics.stanford.edu/papers/gpu_kdtree/kdtree.pdf. Acesso em: 13 set. 2017. https://doi.org/10.1145/1071866.1071869

GARCIA, V.; DEBREUVE, E.; BARLAUD, M.. **Fast k nearest neighbor search using GPU.** 2008 Ieee Computer Society Conference On Computer Vision And Pattern Recognition Workshops, [s.l.], p.1-6, jun. 2008. IEEE. http://dx.doi.org/10.1109/cvprw.2008.4563100.

IZE, T.; WALD, I.; PARKER, S. G.. **Ray tracing with the BSP tree.** 2008 Ieee Symposium On Interactive Ray Tracing, [s.l.], p.159-166, ago. 2008. IEEE. https://doi.org/10.1109/RT.2008.4634637.

JIANG, H.; HALLSTROM, J. O.. Fast, Accurate Event Classification on Resource-Lean Embedded Sensors. **Acm Transactions On Autonomous And Adaptive Systems**, [s.l.], v. 8, n. 2, p.1-22, 1 jul. 2013. http://dx.doi.org/10.1145/2491465.2491470

MUJA, Marius; LOWE, David. **Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration.** Proceedings Of The Fourth International Conference On Computer Vision Theory And Applications, Lisboa, Portugal, p.5-8, fev. 2009. VISAPP 2009. Disponível em:

http://www.cs.ubc.ca/~lowe/papers/09mujа.pdf. Acesso em: 13 set. 2017.

NAYLOR, B. F.. A tutorial on Binary Space Partitioning Trees. In: MEHTA, D. P.; SAHNI, S. (Ed.). **Handbook of data structures and applications.** [S. l.]: Chapman & Hall/crc, 2005. https://doi.org/10.1201/9781420035179.ch20

PEREIRA, Danillo Roberto. **Representação e Cálculo Eficiente da Iluminação Global na Síntese de Imagem.** 2009. Dissertação (Mestrado) - Curso de Ciência da Computação, Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2009. Cap. 4. Disponível em: http://www.liv.ic. unicamp.br/~danillorp/Links/dissertacao.pdf. Acesso em: 08 set. 2017.

PEREIRA, D. R. et al. **FEMa: A Finite Element Machine for Fast Learning** - SUBMITTED - UNDER REVIEW. 1

PRATX, G.; XING, L. GPU computing in medical physics: A review. **Medical Physics,** [S.l.], v. 38, n. 5, p.2685-2697, 9 maio 2011. http://dx.doi.org/10.1118/1.3578605.

SAMET, H. **The Design and Analysis of Spatial Data Structures.** [S. l.]: Addison-wesley Publishing Company, Inc, 1994. 493 p. Disponível em: https://cdn.preterhuman.net/texts/math/Data_Structure_And_Algorithms/The Design and Analysis of Spatial Data Structures - Hanan Samet.pdf. Acesso em: 13 set. 2017.

SHAMONIN, D. et al. Fast parallel image registration on CPU and GPU for diagnostic classification of Alzheimer's disease**.** **Frontiers In Neuroinformatics**, [s.l.], v. 7, p.237-345, 2013. http://dx.doi.org/10.3389/fninf.2013.00050.

SUTHAHARAN, Shan. Big Data Classification: Problems and Challenges in Network Intrusion Prediction with Machine Learning. **Acm Sigmetrics Performance Evaluation Review,** [s.l.], v. 41, n. 4, p.70-73, 17 abr. 2014. https://doi.org/10.1145/2627534.2627557.

VASCONCELLOS, J. F. A. *et al.* Algoritmo Paralelo para Árvore Geradora usando GPU. WSCAD 2017 . *In*: SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, 18., 2017, Campinas. **Anais** [...]. Campinas: SBC,2017.

ZHANG, N.; CHEN, Y.-S.; WANG, J.-li. **Image parallel processing based on GPU.** 2nd International Conference on Advanced Computer Control, [s.l.], 2010. IEEE.