Fast CUDA-based Implementations of Automatic Document Classification Algorithms*

Gabriel Ramos[1], **Guilherme Andrade**[2], **Felipe Viegas**[2], **Daniel Madeira**[1], **Leonardo Rocha**[1]

[1] DCOMP/UFSJ - São João del-Rei, MG , Brasil

[2]DCC/UFMG - Belo Horizonte, MG , Brasil

{gramos,madeira,lcrocha}@ufsj.edu.br,{gnandrade,frviegas}@dcc.ufmg.br

Abstract. *With the advent of* Web 2.0*, we see a new scenario: there is more data available than that can be effectively analyzed. Organizing this data is one of the biggest problems in Computer Science. Many algorithms have been proposed for this purpose, highlighting those related to Automatic Document Classification (ADC). There are some proposals to make these algorithms feasible and those related to parallelization on graphics processing units (GPUs) have presented good results. In this work we present two GPU parallel version of ADC algorithms, the GPU-NB, based on Naïve Bayes, and* G-KNN*, based on KNN. We show that, besides achieved the same effectiveness, our approaches were faster than their CPU versions.*

Resumo. *Com a* Web 2.0*, observamos um novo cenário: existe mais dados do que podemos analisar e organizá-los é um dos grandes problemas em Ciência da Computação. Existem muitos algoritmos com este propósito, destacando os de Classificação Automática de Documentos (CAD). Muitas propostas visam tornar esses algoritmos computacionalmente viáveis, sendo os melhores resultados obtidos pela paralelização em GPUs (Graphics Processing Units). Neste trabalho apresentamos duas versões paralelas em GPU de algoritmos de CAD, o GPU-NB, baseado no Naïve Bayes, e o* G-KNN*, baseado no KNN. Mostramos que, além de alcançarem a mesma eficácia, nossas propostas são mais rápidas que suas versões em CPU.*

1. Introdução

Com o surgimento da Web 2.0, observamos uma verdadeira democratização na geração de dados. Esse grande volume de dados gerou um desafiante cenário: há mais dados disponíveis do que efetivamente pode-se analisar. Organizar e encontrar os recursos informacionais apropriados para satisfazer as necessidades dos usuários passou a figurar como um dos problemas mais desafiadores em Ciência da Computação. Dentre as diversas propostas existentes para lidar com esses dados de maneira eficiente (uso adequado dos recursos computacionais e tempo de resposta) e eficaz (qualidade da resposta), destacam-se as da área de Mineração de Dados, principalmente os algoritmos relacionados a Classificação Automática de Documentos (CAD). Por meio de um aprendizado supervisionado, o objetivo de CAD é criar modelos eficazes capazes de associar documentos a categorias bem definidas de forma automatizada.

*Esse trabalho foi parcialmente financiado por CNPq, CAPES, FINEP, Fapemig, e INWEB.

Existe muitos algoritmos propostos para a CAD, e vários desafios continuam recebendo atenção da comunidade científica, tais como lidar com alta dimensionalidade (muitos atributos definindo os documentos) e dados assimétricos (desbalanceamento entre classes). Em cada cenário, as diferentes características dos documentos podem tornar o processo de aprendizado mais simples ou mais trabalhoso, demandando uma maior utilização dos recursos computacionais ou até mesmo inviabilizando a aplicação dessa estratégia. Há várias propostas para tornar essas aplicações computacionalmente viáveis, seja por meio de técnicas de indexação de dados [Christen 2012], seja reduzindo a dimensionalidade dos dados [Zheng et al. 2004] ou por meio de paralelização utilizando diferentes unidades de processamento [Lin and Chien 2010, Grahn et al. 2011, Garcia et al. 2008, Kato and Hosino 2010], com destaque às paralelizações realizadas por meio do uso de unidades de processamento gráfico (GPUs).

Nesse trabalho apresentamos dois novos algoritmos de CAD paralelizados em GPU: o *GPU-NB*, baseado no algoritmo Naïve Bayes; e o G-KNN baseado no algoritmo KNN. Ambos estão entre os algoritmos mais utilizados em CAD em função da simplicidade, alta eficiência e precisão. A principal diferença de nossa estratégia para as demais propostas está na estrutura de indexação dos dados. A maioria das propostas existentes não consideram aspectos intrínsecos relacionados a CAD, que incluem alta dimensionalidade e heterogeneidade na representação dos documentos (a maioria dos documentos não compartilham os mesmos termos). Considerando a limitação física de memória das GPUs, esses aspectos podem inviabilizar o uso dessas propostas. Nossas estratégias baseiam-se em duas listas compactas, uma contendo a identificação dos documentos e outra contendo a lista de termos de cada documento. Além de simples de ser implementada, a indexação utilizada reduz o consumo de memória e oferece oportunidades de paralelização, proporcionando significativas melhoras de desempenho mesmo com apenas uma GPU.

Avaliamos o desempenho de nossos algoritmos utilizando coleções de documentos distintas, oriundas de cinco bibliotecas digitais comumente utilizadas para avaliar os algoritmos de CAD. Comparamos os resultados obtidos com cada uma das técnicas com suas respectivas implementações originais em CPU, avaliando tanto a eficácia quanto a eficiência. Em todas as classificações realizadas, a eficácia alcançada por nossas propostas foi a mesma que suas implementações originais. Além disso, os resultados evidenciaram que o *GPU-NB* consegue ser até $34x$ mais rápido que a versão em CPU, enquanto o *G-KNN* até $12x$ mais rápido. Por fim, realizamos também um conjunto de experimentos avaliando qual o impacto de algumas características das coleções, tais como esparcidade dos dados e número de classes, no desempenho dos algoritmos propostos.

Este trabalho corresponde a iniciação científica do aluno Gabriel Ramos. Sua participação se deu, em um primeiro momento, auxiliando os alunos de Pós-Graduação do DCC/UFMG Guilherme Andrade e Felipe Viegas e o professor Daniel Madeira na implementação e execução dos experimentos relacionados ao GPU-NB. A partir do aprendizado dessa primeira etapa, Gabriel propôs, implementou e avaliou completamente o G-KNN. *Todo o trabalho foi executado sob a orientação do professor Leonardo Rocha.*

2. Trabalhos Relacionados

Observamos recentemente o surgimento de diversas propostas que visam tornar as aplicações de mineração de dados aplicáveis em cenários reais, onde o volume de dados é tipicamente enorme e as características das coleções variam significativamente [Christen 2012, Zheng et al. 2004]. Dentre as propostas, merecido destaque vem sendo dado para estratégias de paralelização utilizando GPU, unidades processamento gráfico (*Graphics Processing Unit*) [Kumarihamy and Arundhati 2009]. As GPUs são capazes de produzir níveis de paralelismo mais elevados que os obtidos pelos processadores, associado com um menor consumo de energia [University 2010].

Especificamente em relação a CAD, em [Grahn et al. 2011] os autores apresentam uma versão paralela da abordagem de meta-aprendizagem *Random Forest* implementada em CUDA. A abordagem paralela apresentou uma redução significativa no tempo de execução. Em [Lin and Chien 2010], os autores apresentam diversas técnicas para acelerar o SVM em GPUs, incluindo um formato de matriz esparsa para melhorar o desempenho, alcançando excelentes resultados de *speedups*. Com relação aos algoritmos de CAD paralelizados nesse trabalho, não encontramos nenhuma implementação baseada em GPU do algoritmo Naïve Bayes relatados na literatura. Já com relação ao KNN, encontramos algumas propostas, com destaque para as apresentadas em [Garcia et al. 2008, Kato and Hosino 2010]. Em [Garcia et al. 2008] os autores apresentam uma estrutura que representam os dados usando duas matrizes, uma para representar os objetos de teste e a outra para representar os documentos de treinamento, onde as linhas da matriz são os atributos e as colunas são as instâncias. Com base nestas estruturas, que aderem bem ao modelo de programação em GPU, os autores utilizaram a biblioteca CUBLAS (implementação em CUDA da biblioteca BLAS) para paralelizar o cálculo da distância entre os objetos. O resultado deste processo é uma terceira matriz de objetos de treinamento por objetos de teste, em que cada posição representa a distância entre os objetos. Embora esta técnica apresente bons resultados em relação ao tempo de resposta, o consumo de memória tende a ser muito elevado, e nos cenários avaliados em nosso trabalho se tornaram impraticáveis. Em [Kato and Hosino 2010] é apresentada uma versão do KNN utilizando uma paralelização utilizando múltiplas GPUs. Conforme relatado pelos autores, os *speedups* alcançados com essa propostas estão relacionados ao particionamento do processamento independente entre múltiplas GPUs, ignorando as limitações de memória de uma única GPU.

A grande diferença da nossa técnica para as acima apresentadas está na simplicidade e compactação das estruturas dos dados utilizadas para representar o documento. Tal estrutura possibilita a manipulação de um grande volume de dados esparsos, uma característica normalmente encontrada em coleções de documentos reais. Além disso, nossa expectativa é que, conforme discutiremos nos trabalhos futuros, adaptar nossa proposta para que a mesma seja capaz de utilizar múltiplas GPUs nos permitirá alcançar um desempenho ainda maior, sendo capazes de processar grandes e reais conjunto de dados. Parte dos resultados apresentados nesse artigo já foram recentemente publicados, mais especificamente a o *GPU-NB* [Viegas et al. 2013]. Além disso, os resultados referentes ao G-KNN foram submetidos a *International Conference on Parallel Processing*.

3. Descrição das Soluções

Nessa seção apresentamos as descrições das estratégias de paralelização utilizadas, tanto para o *GPU-NB* quanto para o G-KNN. Primeiramente descrevemos a estrutura adotada para indexação dos dados utilizada em ambas as estratégias e, em seguida, apresentamos as particularidades na paralelização de cada algoritmo.

3.1. Indexação dos Dados

Conforme mencionamos, nosso foco é na paralelização de algoritmos de CAD em placas aceleradoras, as quais apresentam uma grande limitação de memória. Considerando que coleções de documentos são normalmente esparsas, nas quais os termos tendem a ocorrer em poucos documentos, optamos por utilizar a representação dos documentos por meio de uma estrutura de dados compacta. Para isso utilizamos dois vetores principais: $docIndexVector$ que representa os documentos; e $docVector$ que armazena as informações sobre cada documento. Em $docIndexVector$ o índice representa o documento, e em cada posição desse vetor armazenamos a posição no vetor $docVector$ onde se inicia sua lista de termos com suas respectivas frequências.

Na figura 1 temos um exemplo de ilustração da estrutura de dados utilizada. Podemos perceber, que nesse exemplo, o documento 0 da base de dados, inicia sua lista de termos no índice 0 do vetor $docVector$ e possui dois termos (termos 0 e 1), sendo cada um deles com frequência igual a 1. Por sua vez, o documento 4 inicia sua lista de termos na posição 8 do vetor $docVectors$ e possui apenas o termo 1, com frequência também igual a 1.

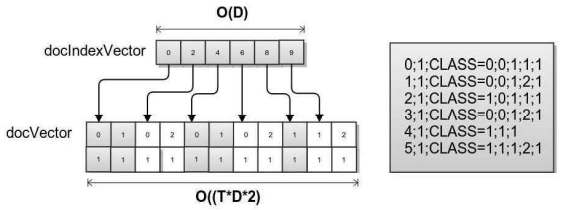


Figura 1. Estrutura de dados usada para representar os Documentos

3.2. GPU-NB

O algoritmo Naïve Bayes é baseado em modelos probabilísticos, baseado no cálculo de $P(c_i|d_t)$ como sendo a probabilidade de um documento de teste $d_t(a_1, a_2, \ldots, a_j)$ pertencer a uma classe c_i, sendo $(a_1, a_2, \ldots, a_j)$ um vetor que representa o conjunto de termos de um documento d_t. O Naïve Bayes define como a classe do documento de teste aquela de maior probabilidade condicional. Essa probabilidade é calculada utilizando o Teorema de Bayes e é definida como:

$$P(c_i|d_t) = \frac{P(c_i) \prod_{\forall j \in d_t} P(a_j|c_i)}{P(d_t)} \quad (1)$$

Observando a Equação 1, temos que a questão fundamental na definição de $P(c_i|d_t)$ está em como estimar $P(a_j|c_i)$, o qual é calculado como segue:

$$P(a_j|c_i) = \frac{Tc_i(a_j) + 1}{\sum_{v \in V} Tc_i(v) + V} \quad (2)$$

Onde $Tc_i(a_j)$ é o número de ocorrências do termo a_j na classe c_i, $V = |V|$ é o tamanho do vocabulário e $\sum_{v \in V} Tc_i(v)$ o somatório do número de ocorrências de todos os termos em documentos da classe c_i. Para evitar inconsistências (i.e., divisão por zero), a suavização de Laplace, que consiste simplesmente em adicionar 1 ao numerador e V ao denominador, é utilizada.

Nossa estratégia de paralelização desse algoritmo utilizando GPUs se baseia em dois kernels principais implementados utilizando a linguagem C em CUDA. Esses kernels exploram o máximo de paralelismo possível, minimizando a dependência dos dados entre as threads. Também se concentram em maximizar a quantidade de computação por acesso à memória.Aproveitando que o método Naïve Bayes assume independência entre os termos, o problema se torna embaraçosamente paralelo para os kernels propostos.A seguir descrevemos cada um deles:

- **Computando a probabilidade dos termos dado as classes (Kernel: Aprendizado):** Durante a leitura dos dados, uma matriz de termos por classes é construída, onde cada posição representa a frequência de ocorrência de cada termo em cada classe. Com base nessa matriz, cada thread em CUDA fica responsável por calcular a probabilidade de seu respectivo termo em cada uma das classes. Como não existe dependência entre os termos, este kernel é embaraçosamente paralelo e cada thread é responsável por uma linha da matriz.
- **Classificação dos documentos de teste (Kernel: Classificação):** Finalmente, dado a matriz de probabilidade construída no kernel anterior, a classificação dos documentos de teste pode ser executada em paralelo. Cada thread fica responsável por um documento de teste, que calcula a probabilidade deste documento em cada classe. O documento é classificado na classe com a maior probabilidade final.

Na figura 2 apresentamos os detalhes dos passos de execução do GPU-NB, enfatizando quais partes são executadas na CPU (*Host*) e quais são executadas na GPU (*Device*), bem como as transferências de dados entre CPU e GPU.

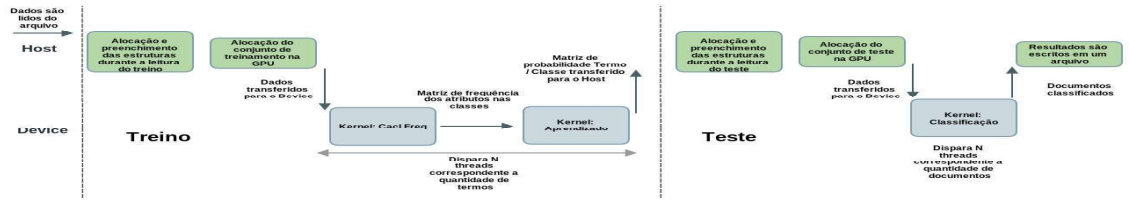


Figura 2. Diagrama de operações e transferência de dados do GPU-NB

3.3. G-KNN

O KNN é um método de classificação em um espaço vetorial não linear que consiste em selecionar os k documentos mais próximos do documento de teste d_t, baseado em algum métrica de similaridade (adotamos a similaridade de cosseno). Posteriormente, d_t é atribuído a classe majoritária dentre os k documentos mais próximos. O número de vizinhos k é um parâmetro para o algoritmo e possui grande impacto na qualidade da classificação gerada pelo kNN. Neste trabalho, consideramos $k = 30$,

visto que é o valor mais utilizado na literatura para CAD [Garcia et al. 2008]. A implementação do G-KNN a partir da paralelização do KNN é feita em C para CUDA e é dividida em dois kernels principais, conforme descritos a seguir:

- **Cálculo das distâncias:** Este *kernel* executa o cálculo das distâncias entre os documentos de teste a serem classificados e todos os documentos de treino. A cada documento de treino, associamos uma *thread*. Portanto, cada *thread* é responsável pelo cálculo da distância entre seu respectivo documento de treino e o documento de teste a ser classificado. Como o cálculo entre cada par de documentos é totalmente independente, o problema se torna embaraçosamente paralelo. No fim da execução do *kernel*, temos um vetor com todas as distâncias calculadas.
- **Ordenação das distâncias:** Após o passo de cálculo das distâncias, ordenamos essas distâncias pelo seu valor de similaridade. Para realizar esse passo, utilizamos a biblioteca *thrust*, distribuída como parte do CUDA SDK. Esta biblioteca provê um conjunto de algoritmos e estruturas de dados paralelos em GPU, os quais são flexíveis e transparentes para o programador. Em particular nesse caso, usamos a função *sort_by_index*, que retorna um segundo vetor de índices ordenados. Portanto é possível encontrar os k documentos mais próximos do documento teste a ser classificado.

Na figura 3 apresentamos o fluxograma do algoritmo, ressaltando as tarefas executadas em CPU (como *Host*) e as tarefas executadas em GPU (*Device*). Esta figura também ilustra o fluxo de dados entre o *host* e o *device*.

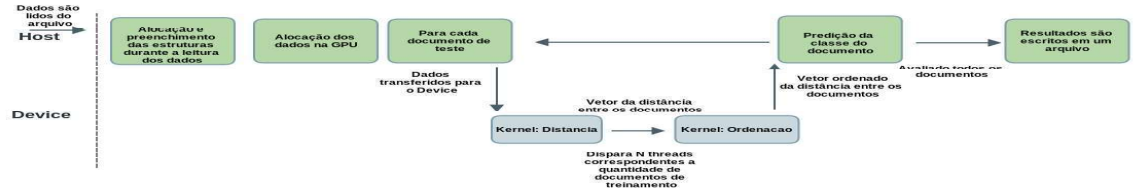


Figura 3. Fluxograma de operações e transferência de dados do G-KNN.

3.4. Implementações em CPU

Apesar de implementações paralelas em CPU não serem nosso foco, elas se tornam necessárias para serem utilizadas como linha de base na avaliação da eficácia e da eficiência de nossa proposta. Nossas implementações das versões em CPU foram feitas na linguagem C e paralelizadas em memória compartilhada por meio da biblioteca *pthreads*. Para ambos os algoritmos, utilizamos os mesmos passos descritos em cada um deles na versão em GPU. No caso da versão em CPU do Naïve Bayes, aproveitamos o tempo de leitura e saída para montar a matriz de probabilidades. No caso da versão em CPU do KNN, utilizamos como algoritmo de ordenação uma versão paralela do *quicksort*.

4. Avaliação Experimental

Nessa seção apresentamos os experimentos realizados para avaliar a eficácia e a eficiência de nossas técnicas. Todos os experimentos foram executados em um computador equipado com Intel Core i7-3930K CPU de 3,20GHz, 32GB memória

e uma GeForce GTX670 2GB e todos os resultados apresentados são o referentes à média de 10 execuções independentes, referentes a uma validação cruzada de 10 partes. Testes de significância (teste-t de dupla calda com 95% confiança) foram executados para comparar os métodos.

4.1. Coleções de Documento

Utilizamos cinco coleções de documento que são comumente utilizadas na área de recuperação de informação. As coleções referenciadas como MedLine e Reuters contêm artigos da área de medicina e notícias da Reuters, respectivamente. A coleção ACM contém documentos da biblioteca digital da ACM. A coleção 20ng contém documentos relacionados a grupos de discussão e a Webkb contém páginas Web coletadas pela CMU World Wide Knowledge Base. A tabela 1 apresenta algumas das características de cada coleção: número de classes, número de documentos, esparcidade (porcentagem média de ausência de termo, considerando todos os termos) e o desvio padrão do número de termos por documento (S.D.), o qual captura a variabilidade do total de termos por documento. Todas elas apresentam um esparcidade acima de 99%, ou seja, cada documento possui, em média, apenas 1% do número total de termos que aparecem na coleção, o que demonstra a importância de uma estrutura de dados compacta.

Coleção	Classes	# de Termos	# de Docs.	Esparcidade	Termos por doc. (S.D)
MedLine	7	268,783	861,454	99.99%	63.61 ± 44.19
ACM	11	56,449	24,897	99.95%	57.37 ± 43.93
20ng	20	61,050	18,805	99.79%	261.02 ± 186.17
Reuters	8	24,985	8,184	99.83%	86.46 ± 58.69
Webkb	7	40,195	8,277	99.65%	280.55 ± 21.56

Tabela 1. Descrição das Coleções.

4.2. Avaliação da Eficácia

Para avaliar a eficácia de nossas técnicas, utilizamos duas métricas comumente utilizadas na área de recuperação de informação: a microF_1 (Mic.F_1) e a macroF_1 (Mac.F_1). A Mic.F_1 mede a eficácia global em termos de todas as decisões feitas pelos classificador e a Mac.F_1 mede a eficácia de classificação em relação a cada classe de forma independente. Comparamos os resultados obtidos por nossas técnicas com os obtidos por suas respectivas implementações em CPU (serial). Apresentamos os resultados na tabela 2. As linhas "diff" indicam a diferença de porcentagem entre nossas técnicas e suas versões em CPU. Os símbolos próximos aos valores indicam se a diferença representa uma variação significativa, de acordo com teste-t de dupla calda com 95% de confiança. Para uma diferença não significativa, o símbolo é •. Nosso objetivo com esse experimento é avaliar se o principal ponto fraco das arquiteturas GPU atuais, a ausência de ponto flutuante de dupla precisão, pode impactar na qualidade das classificações. Essa limitação poderia afetar a precisão das operações matemáticas realizadas pelo *GPU-NB* e G-KNN nos cálculos de probabilidade e de distância, respectivamente. Como podemos perceber, tanto a eficácia do *GPU-NB* (tabela 2 (a)) quanto do G-KNN (tabela 2 (b)) foram, em todos os casos, equivalentes a aquelas obtidas pelas implementações em CPU.

4.3. Avaliação da Eficiência

Nossa primeira avaliação foi relacionada ao consumo de memória de nossas estratégias. Medimos o consumo médio de memória para cada combinação entre uma de nossas técnicas e cada coleção, considerando 10 execuções. Consideramos

Métrica		Mac.F_1(%)	Mic.F_1(%)
Medline	CPU	62.7579	79.1406
	GPU	62.0236	78.6421
	diff(%)	-1.17 •	-0.63 •
ACM	CPU	57.5706	74.2748
	GPU	57.1607	73.3628
	diff(%)	-0.71 •	-1.24 •
20ng	CPU	86.688	87.3617
	GPU	86.7227	87.3936
	diff(%)	+0.04 •	+0.04 •
Reuters_ny	CPU	83.1034	93.1296
	GPU	83.9205	93.1051
	diff(%)	+0.98 •	-0.03 •
Webkb	CPU	52.425	59.3349
	GPU	51.9763	58.7545
	diff(%)	-0.86 •	-0.98 •

(a) GPU-NB vs. Naïve Bayes

Métrica		Mac.F_1(%)	Mic.F_1(%)
Medline	CPU	67.21	80.74
	GPU	68.09	80.72
	diff	+0.88 •	-0.02 •
ACM	CPU	52.77	69.04
	GPU	52.70	69.03
	diff	-0.07 •	-0.01 •
20ng	CPU	39.21	38.59
	GPU	39.04	38.86
	diff	-0.17 •	+0.27 •
Reuters_ny	CPU	84.99	93.58
	GPU	84.88	93.57
	diff	-0.11 •	-0.01 •
Webkb	CPU	49.82	68.15
	GPU	49.65	67.55
	diff	-0.17 •	-0.60 •

(b) G-KNN vs. KNN

Tabela 2. Análise da Eficácia das Paralelizações em GPU

também uma estimativa de consumo de memória por meio de uma representação tradicional utilizando uma matriz de termos por documentos. Esses resultados são apresentados na tabela 3. Podemos observar que o consumo de memória utilizando a representação tradicional torna a execução dos algoritmos computacionalmente inviáveis para a maioria das coleções utilizadas, considerando a limitação de memória de GPUs. Por outro lado, o consumo de memória de nossas propostas foi baixo, com um pico de consumo 237 MB do *GPU-NB* para a MedLine, demonstrando a eficiência da estrutura de dados utilizada em relação a utilização dos recursos computacionais. No geral, o consumo de memória do *GPU-NB* é um pouco maior em função da necessidade de representar a matriz de probabilidades.

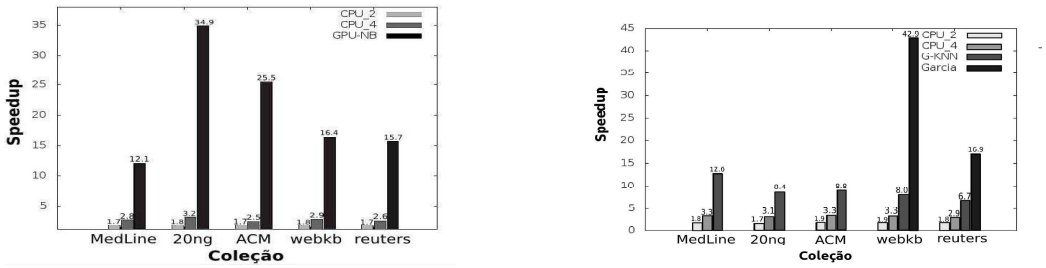
Coleção	Reuters	Webkb	ACM	20ng	MedLine
G-KNN	7,73 MB	8,88 MB	8,53 MB	18,78 MB	211,97 MB
GPU-NB	10,13 MB	13,10 MB	13,26 MB	20,91 MB	237,06 MB
Tradicional	779,26 MB	1,23 GB	5,24 GB	4,28 GB	862,57 GB

Tabela 3. Consumo de Memória.

A segunda etapa de avaliação da eficiência consiste na comparação dos tempos de execução de nossas técnicas paralelizadas em GPU e suas respectivas versões implementadas em CPU. Para essa avaliação, consideramos o tempo médio das 10 execuções do processo de validação cruzada de 10 partes descrita anteriormente. No caso das implementações em CPU, consideramos os tempos relacionados a 1 (sequencial), 2 e 4 *threads*, uma vez que o processador utilizado contém 4 núcleos. Baseado nas medições desses tempos, calculamos os *speedups* em relação ao uso de apenas um núcleo da versão em CPU (sequencial). No caso da avaliação do G-KNN, consideramos também os tempos relacionados ao KNN paralelizado em GPU apresentado por Garcia [Garcia et al. 2008]. No caso do *GPU-NB* não identificamos na literatura nenhuma outra implementação em GPU do Naïve Bayes. Os resultados são apresentado nos gráficos da figura 4.

Avaliando os *speedups* relacionados ao *GPU-NB*, temos que a versão paralelizada utilizando 4 threads alcançou um speedup máximo de $3,2x$ para 20ng. Por outro lado, o *GPU-NB* alcançou um *speedup* de $34,9x$, também para 20ng. Comparando esses resultados, temos que nossa versão paralelizada em GPU é quase $11x$ mais rápida que a versão em CPU.Considerando um comportamento quase linear da versão paralela em CPU (o que é improvável em função da contenção gerada com a adição de novos núcleos.), precisaríamos de 34 núcleos para obter o mesmo *speedup* da GPU. Considerando a avaliação do G-KNN, temos que o

algoritmo proposto por Garcia alcançou os maiores speedup: $42,9x$ para Webkb e $16,9x$ para Reuters. Entretanto, para as demais coleções, em função da estrutura de dados adotada pelo mesmo (matriz de termos por documento) não foi capaz de executar para as demais coleções avaliadas. Por outro lado, o *G-KNN* foi capaz de executar para todas as coleções e alcançou um *speedup* máximo de $12,6x$ para MedLine. A versão CPU com 4 threads alcançou um *speedup* máximo de $3,3x$. Comparando esses resultados, temos que nossa versão em GPU é $3,8x$ mais rápida que a versão paralela em CPU. Considerando um comportamento quase linear da versão paralela em CPU, precisaríamos de 12 núcleos para obter o mesmo *speedup* da GPU. Por fim, se compararmos os custos para aquisição de GPUs e com o custo de novos núcleos, nossos resultados são ainda mais expressivos. Enquanto uma GPU como a utilizada nesse trabalho custa em torno de US$300.00, adquirir uma máquina com 32 núcleos teria um custo por volta de US$1,500.00.



(a) GPU-NB (b) G-KNN

Figura 4. Avaliação dos Speedups de nossas Técnicas

Como os valores de *speedups* alcançados por nossas técnicas variam muito entre as coleções, realizamos dois experimentos fatorais $2^k r$ para identificar quais fatores das coleções mais influenciam no *speedup* de cada um de nossas técnicas. Em ambos os experimentos a variável alvo foi o speedup. Para o experimento do *GPU-NB* consideramos como fatores o número de classes e o desvio padrão do número de termos por documento. Para o G-KNN consideramos o total de documentos e o desvio padrão do número de termos por documento. A escolha dessas variáveis foi feita de acordo com as características das paralelizações realizadas em cada algoritmo. Os resultados desses experimentos podem ser observados na figura 5

(a) GPU-NB (b) G-KNN

Figura 5. Resultado do Experimento Fatorial.

Para o *GPU-NB* temos que o fator número de classes é responsável por 75% do *speedup*. Isso acontece porque o *GPU-NB* utiliza um paralelismo a nível de termos e, quanto maior o número de classes, maior o total de operações aritméticas executadas por acesso a memória. Esse fato explica porque a 20ng foi a coleção de

maior *speedup* para esse algoritmo. Por outro lado, para o G-KNN o fator mais importante é o número de documentos. Isso acontece porque o G-KNN aplica um paralelismo a nível de documentos, por isso o *speedup* alcançado na MedLine foi o maior. Assim, não apenas encontramos excelentes resultados de eficiência para nossas técnicas, como também identificamos a explicação para tais resultados.

5. Conclusões e Trabalhos Futuros

Neste artigo propomos dois algoritmos, o *GPU-NB* e o G-KNN, versões paralelizadas em placas aceleradoras gráficas (GPU) de dois dos algoritmos mais tradicionais de classificação automática de documento (CAD), o Naïve Bayes e o KNN. Diferente de outras abordagens, nossas técnicas utilizam uma estrutura de indexação simples, capaz de manipular grandes volumes de dados esparsos, uma característica comum em coleções de documentos reais, além de possibilitar diversas oportunidades de paralelização em GPU. Avaliamos nossas técnicas considerando cinco coleções de documentos reais, comumente utilizadas na área de recuperação de informação em trabalhos de CAD, contrastando com suas respectivas versões implementadas em CPU. Sob o aspecto da eficácia, demonstramos que a qualidade das classificações realizadas por nossas estratégias foram as mesmas daquelas obtidas por suas versões implementadas em CPU. Por outro lado, sob o aspecto eficência, nossas propostas foram bem mais rápidas que as versões em CPU, $34x$ no caso do *GPU-NB* e $12x$ no caso do *G-KNN*. Como trabalhos futuros, pretendemos paralelizar em GPU outras técnicas de mineração de dados no intuito de se criar uma nova biblioteca de algoritmos. Além disso, estamos trabalhando na adaptação de nossas técnicas para que as mesmas sejam capazes de utilizar múltiplas unidades de GPU.

Referências

Christen, P. (2012). A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*.

Garcia, V., Debreuve, E., and Barlaud, M. (2008). Fast k nearest neighbor search using gpu. In *IEEE CVPRW*.

Grahn, H., Lavesson, N., Lapajne, M. H., and Slat, D. (2011). Cudarf: A cuda-based implementation of random forests. In *AICCSA*, pages 95–101.

Kato, K. and Hosino, T. (2010). Solving k-nearest neighbor problem on multiple graphics processors. In *IEEE/ACM CCGrid*.

Kumarihamy, D. and Arundhati, L. (2009). Implementing data mining algorithms using NVIDIA CUDA.

Lin, T.-K. and Chien, S.-Y. (2010). Support vector machines on gpu with sparse matrix format. In *Ninth ICML*, pages 313–318.

University, T. D., editor (2010). *Reducing the Energy Consumption of Embedded Systems by Integrating General Purpose GPUs*. Technical report.

Viegas, F., Andrade, G., Almeida, J., Gonçalves, M., Ferreira, R., Ramos, G., and da Rocha, L. C. (2013). Gpu-nb: A fast cuda-based implementation of naïve bayes. In *SBAC-PAD*, pages 369–378.

Zheng, Z., Wu, X., and Srihari, R. (2004). Feature selection for text categorization on imbalanced data. *ACM SIGKDD Explorations Newsletter*, 6:80–89.