GPU-NB: A fast CUDA-based implementation of Näive Bayes

Conference Paper · October 2013

DOI: 10.1109/SBAC-PAD.2013.16

7 authors, including:

Guilherme Andrade
Federal University of Minas Gerais
18 PUBLICATIONS **189** CITATIONS
SEE PROFILE

Felipe Viegas
Federal University of Minas Gerais
16 PUBLICATIONS **106** CITATIONS
SEE PROFILE

Jussara Almeida
Federal University of Minas Gerais
184 PUBLICATIONS **3,108** CITATIONS
SEE PROFILE

Leonardo Rocha
Federal University of São João del-Rei
104 PUBLICATIONS **640** CITATIONS
SEE PROFILE

GPU-NB: A Fast CUDA-based Implementation of Naïve Bayes

Felipe Viegas, Guilherme Andrade, Jussara Almeida,
Renato Ferreira and Marcos Gonçalves
Departament of Computer Science
Universidade Federal de Minas Gerais
{gnandrade,frviegas,jussara,mgoncalv,renato}@dcc.ufmg.br

Gabriel Ramos and Leonardo Rocha
Departament of Computer Science
Universidade Federal de São João Del Rei
{gramos,lcrocha}@ufsj.edu.br

***Abstract*—The advent of the WEB 2.0 has given rise to an interesting phenomenon: there is currently much more data than what can be effectively analyzed without relying on sophisticated automatic tools. Some of these tools, which target the organization and extraction of useful knowledge from this huge amount of data, rely on machine learning and data or text mining techniques, specifically automatic document classification algorithms. However, these algorithms are still a computational challenge because of the volume of data that needs to be processed. Some of the strategies available to address this challenge are based on the parallelization of ADC algorithms. In this work, we present GPU-NB, a parallel version of one of the most widely used document classification algorithms, the Naïve Bayes, that uses graphics processing units (GPUs). In our evaluation using 6 different document collections, we show that the GPU-NB can maintain the same classification effectiveness (in most cases) while increasing the efficiency by up to $34x$ faster than its sequential version using CPU. GPU-NB is also up to $11x$ faster than a CPU-based parallel implementation of Naïve Bayes running with 4 threads. Moreover, assuming an optimistic behavior of the CPU parallelization, GPU-NB should outperform the CPU-based implementation with up to 32 cores, at a small fraction of the cost. We also show that the efficiency of the GPU-NB parallelization is impacted by features of the document collections, particularly the number of classes, although the density of the collection (average number of occurrences of terms per document) has a significant impact as well.**

I. INTRODUCTION

The surge of the WEB 2.0 has democratized not only the access to information but also its creation, in an unprecedented speed and volume. This has given rise to an interesting phenomenon: there is more data nowadays than what can be effectively analyzed without the help of sophisticated automatic tools. Many of these tools are concerned with the organization and extraction of useful knowledge from this enormous amount of information. Among the techniques applied by these tools, some of the most widely used ones include machine learning and data/text mining techniques such as automatic document classification (ADC), the focus of our research in this paper.

Basically the goal of ADC techniques is to create effective models capable of associating documents with well-defined semantic categories in an automated way, thus minimizing human intervention. Many important applications such as spam filtering [1], organization of topic directories and digital libraries [2], identification of writing styles or authorship [3], support to enhanced browsing and searching in large information repositories [4], among many others, rely on ADC techniques.

ADC usually exploits a supervised learning paradigm [5] in which a classification model is built using previously labeled documents (training set). This learned model is then used to classify unseen documents (the test set). There is a large variety of algorithms proposed for supervised ADC, and several challenges continue to receive significant attention from the research community such as high dimensionality (due to large vocabularies) and asymmetric aspects of the data (e.g., skewed class distributions). In each scenario the different characteristics of the collections may make the learning task easier or harder, incurring in larger computational costs and ultimately making the use of such techniques in a particular case infeasible.

In fact, a major challenge for ADC is how to produce models with high effectiveness but also in an efficient way, i.e., with adequate use of the computational resources and low response time due to the volume of information to be processed. Moreover, not only intrinsic characteristics of the ADC algorithm affect the patterns of use of the computational resources, but also characteristics of the collections (e..g., number of documents, number of terms, number of classes) may affect these patterns in different ways.

There are in the literature several strategies designed to deal with such aspects in order to make ADC algorithms more robust and scalable, such as dimensionality reduction [6], indexing strategies [7], [8] or parallelism in multiple processing units [9], [10], [11]. Regarding this last type of solution, several recent studies have focused on parallelization using graphics processing units (GPUs) [12]. GPUs are capable of providing a higher parallelism level than what can be obtained with CPUs, with a lower energy consumption [13].

In this context, we present in the paper a new algorithm for ADC, called GPU-NB, a GPU accelerated algorithm for probabilistic classification. GPU-NB is based on the well-known Naïve Bayes algorithm [14], one of the most widely used classification techniques due to its simplicity and high effectiveness in several tasks. More specifically, our technique is based on Multinomial Naïve Bayes, which is the version

most used for ADC. Our strategy, although simple, is very efficient, as we shall see. It is comprised of two main phases: the generation of the model and the actual classification of the documents. The first phase consists in two steps: the calculation of the document term frequencies and the calculation of the probability of the terms within the classes. In both steps an independent thread is run for each term. In the second phase, a *thread* is run for each document to be classified. The great advantage of our technique is in the simplicity and compactness of the data structures used to represent the document. Such structures made possible a manipulation of a large volume of sparse data, normally found in document collections, even with the constraints of GPU memory space.

We evaluated the performance of our proposal using six widely used text collections in ADC experiments. Our results show that our technique can speedup the classification process in up $34x$ when compared to a sequential CPU-based implementation, with basically the same effectiveness in most cases. We also compared the efficiency of GPU-NB against parallelized CPU-based implementations of the Naïve Bayes algorithm. We found that our solution runs up to $11x$ faster than the parallel CPU based implementation with 4 threads and, assuming an optimistic quasi-linear behavior of the CPU parallelization, our method would still outperform the CPU-based parallel implementation up to 32 CPU cores, at a small fraction of the cost. Moreover, we also run a set of experiments to evaluate the impact the characteristics of the collections, such as data density and number of classes, on the potential for performance gains. Our results indicate that the number of classes in the collection is the most important factor for speedup, although density and the interaction between both factors are also important and cannot be neglected.

The rest of this paper is organized as follows. Section II covers related work. Section III describes the basics about the Naïve Bayes algorithm. Section IV presents our main contribution: the CUDA-based implementation of Naïve Bayes. Section V discusses our experimental evaluation while Section VI concludes the paper.

II. Related Work

The literature is rich in proposals of ADC algorithms [15], ranging from simple ones, such as k-nearest neighbor or kNN [16] which considers each test document x as an independent instance and uses the k training documents closer to x (the neighborhood) to classify it, to very sophisticated and highly effective, though also complex and computationally costly, techniques based on maximum margin class separation [17]. An example of an ADC technique that combines simplicity, computational efficiency and good effectiveness is Naïve Bayes (NB). NB is a technique that, basically, evaluates the probability that a document belongs to each of the available classes in the document collection. Due to its inherent properties, it is one of the most used techniques in a number of application such as meta-learning [18] and frameworks for information retrieval [19].

In spite of the number of alternatives, a practical challenge common to all these approaches is how to apply them in real scenarios, where the volume of data is typically huge and the characteristics of the collections vary significantly. Some of the most commonly used techniques to deal with such issues are dimensionality reduction by feature selection [6], whose goal is to keep only the features that better "define" the documents, and data indexing [7], [8], whose goal is to represent the data in a more direct and efficient way. Another solution that has been adopted to address these issues is to exploit parallel computation, either by means of distributed memory strategies [20], [9], or by means of massive parallelism through graphic processors [21], [22], [10], [11], [23]. This last group of strategies has recently demonstrated very interesting results, since they are capable of producing parallelism levels much higher than those achieved by CPUs, associated with a smaller energy consumption [13].

In [21], the authors present an interesting study about how several data mining techniques may be implemented using GPUs, more specifically the CUDA architecture [12]. Specifically regarding ADC, the authors of [22] present a parallel implementation in CUDA for the meta-learning approach Random Forest, with a significant reduction in execution time. Parallel versions of the SVM [10], KNN [11] and DBSCAN [23] algorithms are also available. More specifically, in [10], the authors introduce several techniques to accelerate SVM in GPUs, including a spare matrix format to enhance performance, achieving speedups between $55x$ and $134x$. In [11], the authors apply a data segmentation method to distance calculations, adapted to the model of *threads* and hierarchy of memories of the GPU, also achieving interesting results. Finally, in [23] the authors adopt a simple data indexing by graphs that allows various parallelization opportunities to be explored in GPU. This GPU version achieves speedups of up to $100x$. However, to the best of our knowledge, there is no GPU-based implementation of the Naïve Bayes algorithm reported in the literature. Due to widely spread use of this technique, its massive parallelization is by itself a relevant contribution.

Therefore, in our work we propose a parallel version of Naïve Bayes using CUDA [12] called GPU-NB. In our proposal, we exploit a compact data structure indexed by terms, aiming at minimizing memory consumption. Despite its simplicity, as we shall see in Section IV, this structure allowed us to explore several parallelization opportunities using GPUs. In our evaluation, we show that GPU-NB is able to improve the efficiency of the classification task while keeping its effectiveness in most cases, even if facing some loss in numeric precision typical of the GPU architecture (simple precision). In fact, we were able to achieve reductions of up to $34x$ in execution time when compared to the sequential version in CPU. Moreover, we also evaluated how some characteristics of the collections may impact the potential of efficiency improvement, increasing or decreasing the achieved speedup.

III. NAÏVE BAYES

The Naïve Bayes algorithm [14] is based on probabilistic models that calculate the score of a class c_i as the probability of a document d_t to be assigned to c_i. Based on this score, a ranking of classes is created, and Naïve Bayes classifies d_t in the class in the first position of the rank. More formally,let $P(c_i|d_t)$ be the probability that a test document $d_t(a_1, a_2, \ldots, a_j)$ belongs to class c_i, where $(a_1, a_2, \ldots, a_j)$ is a (binary or weighted) vector representing the terms in document d_t. This probability is calculated using the Bayes Theorem, and is defined as:

$$P(c_i|d_t) = \frac{P(c_i)P(d_t|c_i)}{P(d_t)} \quad (1)$$

The calculation of $P(c_i|d_t)$ has a high computational cost given that the number of possible vectors d_t is also very high. As a consequence, the Naïve-Bayes algorithm attenuates the problems by assuming that the occurrence of all terms are independent. This allows one to calculate the probability of each term a_j independently from the others, making the classification process more viable. Thus, $P(c_i|d_t)$ can be defined as:

$$P(c_i|d_t) = \frac{P(c_i)\prod_{\forall j \in d_t} P(a_j|c_i)}{P(d_t)} \quad (2)$$

There exists in the literature two main approaches for implementing the algorithm. The first, called Bernoulli [24] is a traditional approach in the area of Bayesian Networks, being more appropriate for a fixed number of attributes. The second, called Multinomial [24], is more traditional in statistical language modeling for speech recognition and text classification (our present goal), and thus is the approach adopted in this work.

Observing 2, we can notice that a fundamental question in the definition of $P(c_i|d_t)$ is how to estimate $P(a_j|c_i)$. Therefore, according to [14], we define:

$$P(a_j|c_i) = \frac{Tc_i(a_j)}{\sum_{v \in V} Tc_i(v)} \quad (3)$$

where $Tc_i(a_j)$ is the number of occurrences of term a_j in class c_i and $\sum_{v \in V} Tc_i(v)$ is the summation of the numbers of occurrences of all terms in documents of class c_i. To avoid inconsistencies (e.g.., division by zero), a Laplace smoothing consisting in simply adding 1 to both, the numerator and the denominator, is commonly used:

$$P(a_j|c_i) = \frac{Tc_i(a_j) + 1}{\sum_{v \in V} Tc_i(v) + 1} \quad (4)$$

where V is the term vocabulary. Thus, we can conclude that $P(a_j|c_i)$ defines the representativity of a term a_j in class c_i as being the ratio of the frequency of term a_j in class c_i and the total frequency of all terms in class c_i.

In ADC, our goal is to find the "most suitable" class for a document, which in Naïve Bayes is given by the highest conditional class probability c_i for a given document d_t. As is commonly adopted, we use the sum of the logarithms instead of a simple multiplication of probabilities in order to avoid that probabilities of documents with a large number of terms become quickly close to zero. Therefore, the actual maximization performed in most NB implementations is:

$$c_{map} = argmax_{c_i \in C}[logP(c_i) + \sum_{1 \leq j \leq |V|} logP(a_j|c_i)] \quad (5)$$

The Naïve Bayes training and classification phases are encoded in Algorithm 1.

Algorithm 1: Multinomial Naïve Bayes

```
TrainMultinomialNB(C,D)
    V ⟵ ExtractVocabulary(D);
    N ⟵ CountDocs(D);
    for each c ∈ C
    do
        N_c ⟵ CountDocsInClass(D, c);
        prior[c] ⟵ N_c / N;
        text_c ⟵ ConcatenateTextOfAllDocsInClass(D, c);

        for each t ∈ V
        do
            T_ct ⟵ CountTokensOfTerm(text_c, t);
        for each t ∈ V
        do
            condprob[t][c] ⟵ (T_ct + 1) / (Σ_t' (T_ct' + 1));
    return V, prior, condprob
ApplyMultinomialNB(C,V,prior,condprob,d)
    W ⟵ ExtractTokensFromDoc(V, d);
    for each c ∈ C
    do
        score[c] ⟵ logprior[c];
        for each t ∈ W
        do
            score[c] += logcondprob[t][c];
    return argmax_c ∈ Cscore[c]
```

IV. PARALLELIZATION OF NAÏVE BAYES

In this section, we present our strategy to parallelize the Naïve Bayes algorithm. We start by presenting GPU-NB, our new GPU-based implementation using CUDA (Section IV-A). Next, we discuss the CPU-based implementation of the algorithm used here as baseline for comparison (Section IV-B).

A. GPU-NB

The selection of a convenient data representation is very important in obtaining an efficient implementation of algorithms. Because our main goal is to obtain good performance on current graphics processing hardware, which are afflicted by

severe memory limitations, we opted for a dense data structure, indexed by the terms, to represent the documents of the collection. We utilized two vectors, namely $TermIndexVector$, which represents the terms, and $DocTermVector$, which stores lists of documents in which each term appears. In $TermIndexVector$, the index represents the term, and each position of the vector contains the position in $DocTermVector$ where the list of terms for that particular document begins. Figure 1 depicts the data organization.

Both stages, training and test, were parallelized in our proposal. Figure 2 shows the steps of the main execution of GPU-NB, highlighting the parts of the algorithm that run on the Host CPU as well as the ones that execute on the GPU device. It also depicts the data transfers between host and device.

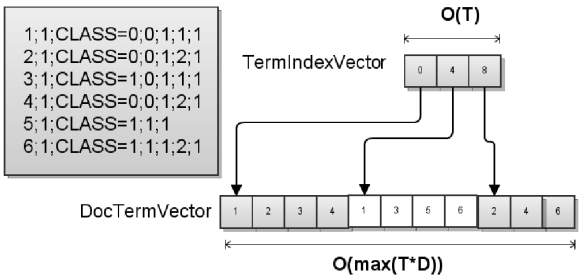

Fig. 1. Data structure used to represent documents in a collection.

The three CUDA kernels implemented extract as much parallelism as possible, by minimizing data dependencies across the different CUDA threads. It also focuses on maximizing the amount of computation per memory access. Taking advantage of an important characteristic of the Naïve Bayes method, namely the assumption of term independency, the problem becomes embarrassingly parallel for the proposed kernels. The following items detail the strategies adopted for each kernel:

- **Computing the frequency of the terms (CalcKernel):** This kernel issues CUDA threads for each term in the data collection. Together they fill up a matrix of size $Term \times Class$ with the number of times each term occurs on each class. Since there is no dependency across terms, this kernel is completely parallel, with each thread being responsible for a line in the matrix.
- **Computing the probabilities of terms in classes (LearningKernel):** Given the matrix generated in the previous kernel, the probability is computed in a similar fashion. The CUDA threads are issued for each term in the collection, and compute the probability of their respective term in each of the classes, dividing the frequency of the term in the class by the total number of terms in that class. Once again, the algorithm is embarrassingly parallel, given the term independency assumption of the Naïve Bayes method.
- **Classification of test documents (TestingKernel):** Finally, given the probability matrix generated in the previous kernel, the classification of the test documents can be performed for each document in parallel. There is one thread for each document in the test collection, which is responsible for computing the probability of the document on each class. The document is classified on that class with highest final probability.

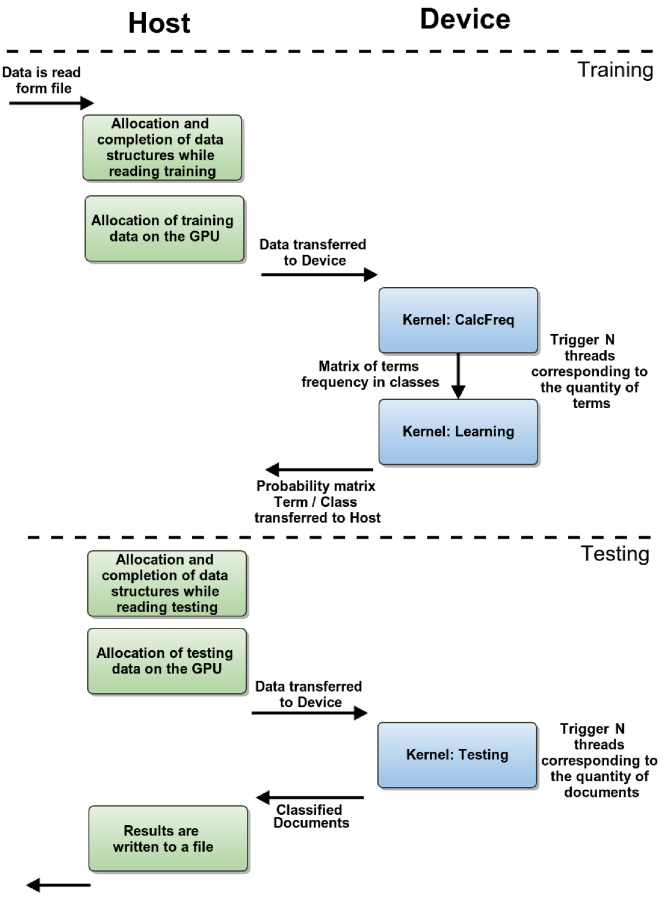

Fig. 2. Computation and data transfers in GPU-NB.

B. CPU Implementation

Even though the CPU implementation is not the main focus of our work, we here use it as baseline for comparison in terms of both efficiency and classification efficacy. Such implementation, was, therefore, carefully made in C++ with shared memory parallelization using OpenMP. This was done so that we can get a fair comparison for our GPU implementation of the method. The code was organized in the same steps as shown in Figure 2, with extra optimizations in order to achieve high efficiency on CPU, as discussed below.

The operations corresponding to the first kernel, which computes the frequencies for each term, is performed while the training data is read from the disk. While this approach allows the overlapping of I/O with computation, it also introduces data dependencies. So, there is no shared memory parallelization for that part of the algorithm. We have conducted preliminary experiments and realized that the overlapping of I/O and computation actually offsets the loss of parallelism leading to better performance on the CPU.

The parallel versions of the other two stages of the computation were implemented using a bag of tasks approach. As in the GPU implementation, we have exploited the characteristics of the Naïve Bayes method to achieve independent tasks. Therefore, the CPU threads compute the probabilities for each term, as well as the classification of each document, independently of the remaining threads. Moreover, the classification of the test documents also exploited the overlapping of I/O and computation, as each thread would compute the probabilities of each document as it reads the document from the disk.

In the following section we present our experimental evaluation of the proposed method. For that, we have selected some document collections with varying characteristics. Our experiments focused on comparing the CPU and the GPU implementations in terms of efficiency (speedup) as well as efficacy (the quality of the classifications).

V. Experimental Evaluation

In this section we present experimental results conducted in order to evaluate the effectiveness and the efficiency of our GPU-NB algorithm. The results were compared to those of (sequential and parallel) CPU implementations of the Naïve Bayes algorithm.

In the following subsections we present the data collections used in our experiments, as well as the evaluation metrics and techniques applied to the experimental results. All experiments were performed on a computer with Intel Core 17-2600 CPU at 3,40GHz, with HyperThreading, 16GB of main memory, and equipped with a GeForce GT520 2GB graphics card. All results presented are averages of 10 independent executions of the algorithms. Moreover significance tests (paired t-tests) were run to test for significant differences, with 95% confidence, across different methods.

A. Document Collections

Six real digital libraries were used in our experiments. The collections referred to as Medline and Reuters_ny contain medical documents and a set of news from Reuters[1], respectively. The ACM collection contains documents from the ACM digital, whereas the acl_bin collection contains product reviews from Amazon [25]. Our set also includes a collection of documents from newsgroups (the 20ng collection[2]), and a collection related to webpages collected by the World Wide Knowledge Base Project of the CMU text learning group (the Webkb collection). Table I presents basic characteristics of each collection: number of classes, number of attributes, number of documents and density (average number of occurrences of terms per document).

B. Effectiveness of the proposed algorithm

In order to assess the quality of the classification achieved by our proposed GPU-NB algorithm we used standard metrics adopted in the Data Mining and Information Retrieval communities: the microF_1 (Mic.F_1) and the macroF_1 (Mac.F_1).

[1]www.reuters.com

[2]http://qwone.com/ jason/20Newsgroups/

Collection	Classes	Number of Attributes	Number of Documents	Density
MedLine	7	268,783	861,454	30.805
ACM	11	56,449	24,897	27.687
20ng	20	61,050	18,805	129.511
Reuters_ny	8	24,985	8,184	42.2302
Webkb	7	40,195	8,277	139.275
acl_bin	2	1,110,351	27,677	181.509

TABLE I
General information on the data collections.

Mic.F_1 measures the global effectiveness in terms of the decisions made by the classifier (the inverse of the error rate). Mac.F_1, on the other hand, measures the effectiveness of the classification within each class independently, computing the harmonic average of precision and recall for each class [26].

Like in most of the related work on evaluation of classification algorithms, a k-fold cross-validation [27] technique is used in the experiments with $k = 10$ for each combination of collection and algorithm. This technique consists in dividing the input collection into k subcollections of the same size, preserving the class distribution. Then the experiments are run k times, where in each time one of the k subcollections is used as test set, and the remaining $k - 1$ are used as training set. The metrics Mic.F_1 and Mac.F_1 are computed for each of the k test sets and, after all executions, the independently computed metrics are averaged for a more reliable measurement of the quality of the classification.

Our results are presented in Table II. The "diff" lines indicate the percentage difference of our proposed GPU-NB and a baseline (sequential) CPU implementation of Naïve Bayes. The symbols proceeding the values in that line gives a pictorial indication of whether the variation is statistically significant, according to a paired t-test with 95% confidence. When the difference is not significant, the symbol is a •. However, if it is significant, a ▼ represents a negative variation (CPU-based Naïve Bayes beats GPU-NB).

Our goal with these experiments was to evaluate how the main weakness of current GPU architectures, the lack of double precision floating point operations, impacts the quality of the classification. This limitation may affect the precision of the mathematical operations performed by GPU-NB during the probability calculations. As we can see in Table II, the effectiveness of the classification produced by GPU-NB was, in most cases, equivalent to that obtained by the CPU implementation. The exceptions are Mac.$F1$ in Webkb, Mic.$F1$ and Mac.$F1$ in 20ng and Mic.$F1$ in MedLine. Observing Table I, we note that these exceptions can be explained by the density and number of classes in the collections. The denser the collection, the higher the number of terms and thus the higher the number of calculations per document, implying in a larger numerical error propagation and less precise probabilities (Equation 2). This may explain the losses that occur in 20ng and Webkb (denser collections), as well as the statistical ties that happened in ACM and Reuters_ny (sparser collections). Regarding MedLine, although this is a sparser collection, we observe

Metric		Mac.F_1(%)	Mic.F_1(%)
Medline	CPU	63.8816	82.8773
	GPU	62.7653	79.1672
	diff	-1.20 ●	-4.50 ▼
ACM	CPU	56.6745	73.6262
	GPU	57.4675	74.2601
	diff	+1.20 ●	+0.90 ●
20ng	CPU	86.7831	87.4586
	GPU	81.3838	83.7411
	diff	-6.20 ▼	-4.25 ▼
Reuters_ny	CPU	81.8947	92.991
	GPU	83.5371	93.1405
	diff	+1.75 ●	+0.15 ●
Webkb	CPU	52.3967	59.3847
	GPU	47.2047	57.9202
	diff	-9.35 ▼	-2.10 ●
acl_bin	CPU	88.7706	88.7765
	GPU	88.4972	88.5034
	diff	-0.27 ●	-0.27 ●

TABLE II
EFFECTIVENESS OF THE NAÏVE BAYES IMPLEMENTATIONS.

a small loss in Mic.$F1$. This may be explained by the large total number of terms. This could affect the calculations of individual probabilities of individual terms (Equation 4). Regarding the acl_bin collection, although this is the densest collection, it is also the one with the smallest number of classes. Thus, for this collection, the loss in precision due to the calculation of the probabilities is compensated by the calculation of the conditional probabilities of only two classes (Equation 5).

Despite the small losses in classification effectiveness (under 10%) for a few collections, methodologically we can say that our proposal is basically equivalent to the CPU implementation. In any case, in the near future we can exploit feature selection techniques [6] in order to keep only the most discriminative features, thus reducing the aforementioned effects. Moreover, double precision floating points are becoming available on the latest GPUs, due to its high demand for general purpose computations. Next we evaluate the gains in efficiency of our proposed approach.

C. *Efficiency of the proposed algorithm*

In order to evaluate the efficiency of the proposed algorithm, we measured the execution times for all iterations of the cross-validation described earlier, both for GPU-NB as well as for the CPU implementation, and averaged all the times for each configuration. In the case of the CPU implementation, we measured the average execution time of 1 (sequential), 2 and 4 threads, as this is the number of cores in the processor used for experimentation. For completeness, we added an 8 thread version, as the processor also supports hyperthreading technology. Based on these measurements, we computed speedups in relation to the single CPU thread version. The obtained speedups are shown in Figure 3. We omit confidence intervals for the sake of clarity of the graph. We note however that, for all collections, the speedups achieved with GPU-NB over the sequential implementation was statistically larger, with 95% confidence, than the speedups of all parallel CPU-based implementations considered. Thus, in the following, we focus our discussion on average results.

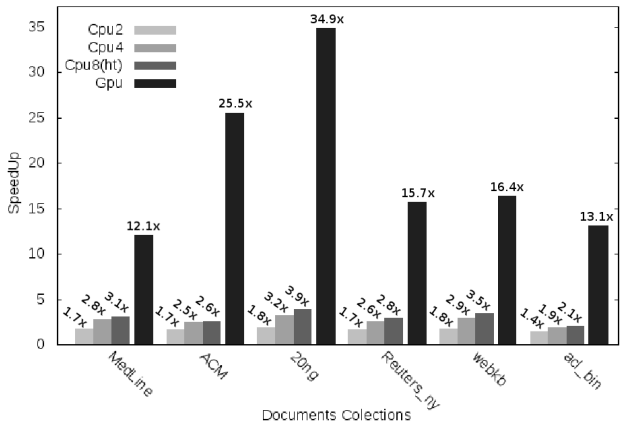

Fig. 3. Speedup of Parallel Implementations for Various Collections.

Analysing the results, we can observe that for parallelizations in CPU, the speedup obtained by utilizing 4 threads (effectively 4 cores), reaches a maximum of $3.2x$ (for the 20ng collection). On the other hand, evaluating GPU-NB, we observed speedups of up to $34,9x$ (also for the 20ng collection), meaning that our proposed algorithm lead to significant reduction on the execution time of method. By comparing these two results, we can safely claim that our proposed algorithm is $11x$ faster on the GPU we used. If we maintain a quasi-linear behavior of the CPU parallelization we have obtained up to 4 cores, GPU-NB would continue to be faster up to 32 CPU cores. This is very optimistic as the more cores we add to the architecture, the more contention we get on the inner buses. Observing the 8 thread execution of the CPU based implementation we notice that it does not get much better than the execution with 4 threads, because these are exploiting the HyperThreading capability of the CPU, and not really running 8 parallel and independent threads. In that case, GPU-NB is still almost $9x$ faster.

If we take cost into consideration, our results are even more impressive. While a GPU like the one we used in our experiments costs about US$60.00, obtaining a configuration with 32 CPU cores would cost around US$4,000.00. Another important observation is that the proposed algorithm is typically executed multiple times as part of larger applications. Thus, the greater reduction on execution time achieved by GPU-NB is of great benefit, for practical purposes. Finally, we can increase significantly the performance by using better GPUs, with more cores in its architecture.

Another interesting aspect to be noticed from our experiments is that there is a significant variation of the speedup with different collections, which leads us to conjecture that

there are characteristics of the collections that greatly impact the proposed parallelization strategy, leading to an increase or decrease on the speedup. We proceeded to conduct further experimentation, detailed in the following subsection, in order to determine how the different characteristics of the data collections impact the performance of our proposed parallelization.

D. Impact of Characteristics of the Collection on the Efficiency of GPU-NB

The goal of an adequate experimental design is to obtain the maximum information with the minimum number of experiments. In particular, it is important to be able to separate out the effects of various factors (parameters of interest) that might affect the performance of the system under study, and to determine whether the observed difference is due to significant effects or simply to random variations caused by measurement errors and/or the inherent variability of the process being analyzed [28]. There are various types of experimental designs. A $2^k r$ factorial design allows the experimenter to quantify the relative importance (i.e., impact) of a number k of factors, each with two alternative values or levels (lower and upper), on the system performance measured by a given response variable [28]. It also allows one to quantify the importance of all possible inter-factor interactions. Two factors have a significant interaction if the effect of one depends on the specific level of the other, in which case they cannot be evaluated separately and independently. A $2^k r$ design consists of 2^k experiments defined by all combinations of factor levels, each of which is replicated r times to allow us to measure the inherent variability in the data (also called experimental errors). Such design is appropriate to provide some initial insights into the relative importance of different factors (and factor interactions) on the response variable. This importance is expressed as the fraction of the total variation observed in the measurements that can be explained by the changes in the levels of each factor (and interaction). More important factors explain a larger fraction of the total variation. The fraction of variation that cannot be explained is credited to experimental errors.

We here apply a $2^k r$ design to analyze the impact of characteristics of the collection on the efficiency of GPU-NB, aiming at drawing fundamental knowledge about its performance on different collections. Thus, the response variable in our design is speedup. We consider $k = 2$ factors, namely, density and total number of classes. The density of a collection captures both the size of the vocabulary and the total number of documents in the collection. Both aspects vary significantly across collections. However, their impact on the speedup of GPU-NB is directly reflected in the impact of the density, as a normalized metric. Thus, taking only density as a varying factor is enough to study their impact. As in Section V-B, we apply a 10-fold cross-validation in our experiments. Thus, our design consists of $r = 10$ replications of each configuration.

Having defined the $k = 2$ factors of our design, the next step is to select their two levels. For the number of classes, we selected the lower level as 7 classes and the upper level as 11 or more classes. For density, we selected the lower level as values below 31 and the upper level as values above 129. The choice of levels was driven by the distributions of the analyzed factors across our collections (see Table I) and the need to have at least one collection representing each configuration of our design so as to capture their relative impact on the speedup. The collections selected for the factorial design are MedLine, Webkb, ACM and 20ng, representing, respectively, the following configurations: low density and small number of classes, high density and small number of classes, low density and large number of classes, high density and large number of classes.

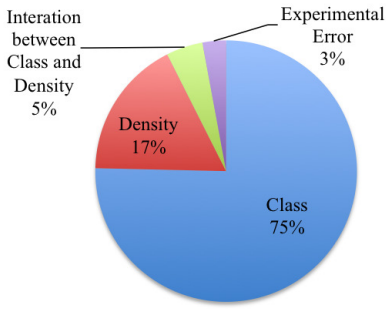

Fig. 4. Variation Explained by Factors in System Performance

Figure 4 shows the main results of the $2^k r$ experimental design, summarized as the fractions of variation explained by each factor, interaction, and by experimental errors. Note that the number of classes in the dataset is responsible for 75% of the total variation, being thus the most important factor that impacts the speedup of GPU-NB. This is because GPU-NB employs a parallelism of threads at the level of terms, and thus, the larger the number of classes, the larger the number of arithmetic operations performed by each thread (e.g., computation of $P(t|c)$) per memory access. This explains why the ACM and 20nb collections, with 11 and 20 classes respectively, achieve the highest speedups of all datasets ($25.5x$ and $34.9x$, respectively).

Note also that the density factor and its interaction with the number of classes cannot be disregarded either, as both are responsible for non-negligible fractions of the variation, and thus also significantly impact the final speedup of our method. Indeed, we find that that both effects are statistically significant (non-zero), with 95% confidence. The importance of density, for instance, explains why the results for the acl_bin collection (speedup of $13x$) are slightly superior to those of Medline (speedup of $12.1x$): even though the former has fewer classes (2 versus 7), it has a much higher density (181.51 versus 30.81). The fraction of variation left unexplained is only 3%, and we believe that this can be further reduced by increasing the number of replications.

VI. Conclusions and Future Works

In this work, we proposed GPU-NB, a parallel implementation of the widely used Naïve Bayes algorithm for ADC. In our proposal, we exploited a very compact data structure to index the documents per term aiming at minimizing memory

consumption. as well as maximizing the opportunities for massive parallelization of the algorithm in GPUs.

We experimentally evaluated our proposed approach using six widely used datasets in ADC, considering both effectiveness and efficiency. Our results show that we can produce a speedup of up to $34x$ in execution time when compared to a sequential CPU-based implementation while keeping the same quality of the classification in most cases (with only a few cases with small losses due to loss in numeric precision of the GPU). We should stress that in situations like meta-learning, in which a NB is called many times, such reductions are very important. In a factorial project designed to evaluate the impact of the characteristics of the collections in the potential of speedup, we found that the number of classes is the main factor (explaining 75% in the variation of performance). However, the density of the collections also explain about 17% of the variation, being also an important factor.

As future work, we intend to evaluate GPU-NB in other textual collections, study other aspects that can affect the potential for speedup, and apply our algorithm in meta-learning frameworks to evaluate the total reduction in time. We also intend to relax issues related to the independence of terms in the NB algorithm given the possibilities that the GPU provide, aiming at improving the effectiveness of the algorithm.

ACKNOWLEDGEMENTS

This work was partially supported by the Brazilian National Institute of Science and Technology for Web Research (MCT/CNPq/INCT Web Grant Number 573871/2008-6), and by the authors' individual grants from CNPq, CAPES, FINEP and FAPEMIG.

REFERENCES

[1] J. Hovold, "Naive bayes spam filtering using word-position-based attributes," in *CEAS 2005 - Second Conference on Email and Anti-Spam, July 21-22, 2005, Stanford University, California, USA*, 2005.

[2] I. Fahmi, "Examining learning algorithms for text classification in digital libraries," in *Master's thesis*, University of Groninge, Netherland, 2004.

[3] R. Zheng, J. Li, H. Chen, and Z. Huang, "A framework for authorship identification of online messages: Writing-style features and classification techniques," *J. Am. Soc. Inf. Sci. Technol.*, vol. 57, no. 3, pp. 378–393, Feb. 2006.

[4] L. Terveen, W. Hill, and B. Amento, "Constructing, organizing, and visualizing collections of topically related web resources," *ACM Trans. Comput.-Hum. Interact.*, vol. 6, no. 1, pp. 67–94, 1999.

[5] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002.

[6] Z. Zheng, X. Wu, and R. Srihari, "Feature selection for text categorization on imbalanced data," vol. 6, pp. 80–89, 2004.

[7] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, pp. 1537–1555, 2012.

[8] G.-H. Cha and Y.-I. Yoon, "Clustered indexing technique for multidimensional index structures," in *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, ser. DEXA '02. London, UK, UK: Springer-Verlag, 2002, pp. 935–944.

[9] C. Kruengkrai and C. Jaruskulchai, "A parallel learning algorithm for text classification," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '02. New York, NY, USA: ACM, 2002, pp. 201–206. [Online]. Available: http://doi.acm.org/10.1145/775047.775077

[10] T.-K. Lin and S.-Y. Chien, "Support vector machines on gpu with sparse matrix format," in *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, 2010, pp. 313–318.

[11] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using gpu," in *CVPRW '08. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops.*, 2008, pp. 1–6.

[12] M. Fatica and D. Luebke, "High performance computing with CUDA," Supercomputing 2007 tutorial. In Supercomputing 2007 tutorial notes, November 2007.

[13] *Reducing the Energy Consumption of Embedded Systems by Integrating General Purpose GPUs*, ser. Technical reports in computer science. TU, Department of Computer Science, 2010. [Online]. Available: http://books.google.com.br/books?id=uBfUYgEACAAJ

[14] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008, ch. 13, pp. 234–265.

[15] H. Borko and M. Bernick, "Automatic document classification," *Journal of the ACM (JACM)*, vol. 10, no. 2, pp. 151–162, 1963.

[16] M.-L. Zhang and Z.-H. Zhou, "A k-nearest neighbor based algorithm for multi-label classification," in *Granular Computing, 2005 IEEE International Conference on*, vol. 2. IEEE, 2005, pp. 718–721.

[17] T. Joachims, "Making large scale svm learning practical," 1999.

[18] C. Elkan, "Boosting and naive bayesian learning," Tech. Rep., 1997.

[19] I. Sandin, G. Andrade, F. Viegas, D. Madeira, L. C. da Rocha, T. Salles, and M. A. Gonçalves, "Aggressive and effective feature selection using genetic programming." in *IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–8.

[20] A. S. Ruocco and O. Frieder, "Clustering and classification of large document bases in a parallel environment," 1997.

[21] D. Kumarihamy and L. Arundhati, "Implementing data mining algorithms using NVIDIA CUDA," 2009. [Online]. Available: http://hdl.handle.net/1900.100/3013

[22] H. Grahn, N. Lavesson, M. H. Lapajne, and D. Slat, "Cudarf: A cuda-based implementation of random forests," in *Proceedings of the 2011 9th IEEE/ACS International Conference on Computer Systems and Applications*, ser. AICCSA '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 95–101.

[23] G. Andrade, G. Ramos, D. Madeira, R. S. Oliveira, R. Ferreira, and L. C. da Rocha, "G-dbscan: A gpu accelerated algorithm for density-based clustering," in *ICCS*, 2013, pp. 369–378.

[24] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *Workshop on Learning for Text Categorization at the 15th Conf. of the American Association for Artificial Intelligence*. Madison, Wisconsin: AAAI Press., 1998, pp. 41–48.

[25] X. Glorot, "Domain adaptation for large-scale sentiment classification : A deep learning approach," *Learning*, no. 1, pp. 513–520, 2011.

[26] D. D. Lewis, "Evaluating and optimizing autonomous text classification systems," in *Eighteenth Annual, International ACM-SIGIR Conference*, 1995, pp. 264–254.

[27] L. Breiman and P. Spector, "Submodel Selection and Evaluation in Regression. The X-Random Case," *International Statistical Review*, vol. 60, no. 3, 1992.

[28] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York, NY: John Wiley, 1991.

[29] A. Freitas, "A survey of parallel data mining," in *Proc 2nd Int Conf on the Practical Applications of Knowledge Discovery and Data Mining*, H. Arner and N. Mackin, Eds. London: The Practical Application Company, 1998, pp. 287–300.

[30] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers Inc., 2000.

[31] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[32] J. Platos, V. Snasel, T. Jezowicz, P. Kromer, and A. Abraham, "A pso-based document classification algorithm accelerated by the cuda platform," in *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, 2012, pp. 1936–1941.

[33] G. Forman, "An extensive empirical study of feature selection metrics for text classification," vol. 3, pp. 1289–1305, 2003.

[34] D. Lowd and P. Domingos, "Naive bayes models for probability estimation," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 529–536.

[35] W. Lam and C. Y. Ho, "Using a generalized instance set for automatic text categorization," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: ACM, 1998, pp. 81–89.