IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 August 2023, accepted 7 September 2023, date of publication 13 September 2023,
date of current version 19 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3315239

RESEARCH ARTICLE

ezLDA: Efficient and Scalable LDA on GPUs

SHILONG WANG[1], HANG LIU[2], (Senior Member, IEEE), ANIL GAIHRE[2], AND HENGYONG YU[1], (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA 01854, USA
[2]Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ 07030, USA

Corresponding author: Hengyong Yu (hengyong_yu@uml.edu)

ABSTRACT Latent Dirichlet Allocation (LDA) is a statistical approach for topic modeling with a wide range of applications. Attracted by the exceptional computing and memory throughput capabilities, this work introduces ezLDA which achieves efficient and scalable LDA training on GPUs with the following three contributions: First, ezLDA introduces three-branch sampling method which takes advantage of the convergence heterogeneity of various tokens to reduce the redundant sampling task. Second, to enable sparsity-aware format for both **D** and **W** on GPUs with fast sampling and updating, we introduce hybrid format for **W** along with corresponding token partition to T and inverted index designs. Third, we design a hierarchical workload balancing solution to address the extremely skewed workload imbalance problem on GPU and scale ezLDA across multiple GPUs. Taken together, ezLDA achieves superior performance over the state-of-the-art attempts with lower memory consumption.

INDEX TERMS Bayes methods, GPU, high performance computing, latent dirichlet allocation, LDA, parallel algorithms, parallel programming, machine learning, unsupervised learning.

I. INTRODUCTION

Topic modeling is a type of statistical approach that reveals the *latent* (i.e., unobserved) topics for a collection of documents (also referred to as corpus). LDA [1], which chooses the Dirichlet distribution as the statistical model to formulate topic distributions, is one of the most popular topic modeling approaches that find applications in not only text analysis [2], [3], but also computer vision [4], recommendation system [5], [6] and network analysis [7] among many others. Thanks to the practical implications, contemporary search engines rely upon LDA to handle billions of news with 10K of topics and 1000K of words [8].

Recently, we also observe interesting interactions between LDA and popular deep learning models. First, Functional and Contextual attention-based Long Short-Term Memory (FC-LSTM) [9] uses LDA to preprocess the data and feeds the corresponding results into LSTM model to improve the accuracy in a recommendation system. LDA can also cooperate with Convolutional Neural Network (CNN) model as a preprocessing method to deal with automobile insurance fraud problems [10]. Second, logistic LDA [11], which is a modified supervised LDA model, can achieve comparable accuracy with Syntax Aware LSTM (SA-LSTM) [12] on document classification with much shorter training time than SA-LSTM. BPLDA [13] can achieve comparable accuracy on classification and regression as deep learning with much shorter training time. Compared with recent deep learning based natural language processing (NLP) tools, e.g., Embeddings from Language Models (ELMo) [14] and Bidirectional Encoder Representations from Transformers (BERT) [15], [16], LDA also presents a solid theoretical foundation which is absent for deep learning models.

As the size of NLP problems continues to rise, it becomes imperative for us to scale the training of LDA towards more computing resources, as well as accommodating larger corpus with more topics. Graphics Processing Units (GPUs) exhibits remarkable performance over traditional CPU system and are hence widely applied on compute-intensive problems such as deep learning [17], [18], [19], [20], [21] and graph [22]. Towards expediting LDA training, GPUs are a tempting platform for two, if not more, reasons. First, modern GPUs yield extraordinary computing and data delivering capabilities, both of which are crucial for LDA training. Second, GPUs possess a thriving community with steady

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Alawneh.

updates in both hardware and software support, which helps extend the impacts of LDA.

Generally speaking, LDA encompasses three tensors and two tasks. First, the three tensors are: the token list T - an array of <wordId, docId, topicId> triplets, a document-topic matrix (i.e., **D**) and a word-topic matrix (i.e., **W**). Second, the two tasks are sampling and updating. Each token is initialized with a topicId from a random distribution. During sampling, LDA takes as input each specific token, i.e., t, and relies on **D** and **W** to generate a new topic for this token. The intuition of sampling is that *the probability of assigning new topic t is positively correlated to the number of tokens for each topic of the document and word this t belongs to*. During updating stage, topicId in T, **D** and **W** are updated to reflect the new topic generated for each token t.

A. RELATED WORK

As one of the most popular topics in machine learning, LDA [1] has received enormous attentions. This section restricts our discussions to the projects that are closely related to ezLDA, that is, efficiency, scalability and GPU-based LDA.

There mainly exist four directions to make LDA more efficient than the original attempt [1], that is, sparsity-aware, Metropolis-Hasting (MH) and Expectation Maximization (EM) approaches, as well as the hybrid of them. **i)**. Sparsity-aware method utilizes the sparsity of word-topic and document-topic matrix to make the sampling time sub-linear to number of topics K (detailed in Section II-B). SparseLDA [23] pioneers this attempt. **ii)**. MH [24] method generates a complex distribution by constructing a Markov Chain (MC) with a simple easy-to-sample distribution and update the topic with some acceptance rates at each step. Since MH requires frequent random memory address to word-topic and doc-topic matrices, thus is not friendly for sparse matrix. **iii)**. LDA* [8] uses sparsity-aware and MH samplers to deal with short and long documents separately. The follow-up variations are [25], [26], [27], [28], [29], [30], and [31]. **iiii)**. EM [32] divides the LDA training into E-step and M-step while the former is responsible for sampling and the latter for updating. Comparing with standard LDA sampling, EM can enjoy better parallelism because frequent random memory access to update word-topic and document-topic matrices during sampling can be avoided.

Large-scale training is another important field for LDA considering real-world corpus often contains billions of tokens. LightLDA [26] leads this effort. Particularly, it trains LDA model with 1 million topics and 1 million words on eight machines via data parallelism (corpus partition) and model parallelism (word-topic matrix splitting). While LightLDA allows both **D** and **W** to be sparse, it relies upon hash table for fast sampling, which is a hardership for GPU because collision handling in hash table remains elusive on GPUs [33]. This concern is evident by SaberLDA [28] which only stores **D** in sparse format for fast sampling. cuLDA [29] further attempts this challenge on multi-GPU but ends up with identical strategy as SaberLDA except scaling the sampling towards multiple GPUs. As we will evaluate in Table 1, only allowing **D** to be sparse will greatly hinder the scalability of LDA.

Last, for GPU-based LDA, which is the interest of this work, we find very few efforts. *Yan et al.* [34] implements collapsed Gibbs sampling [35] and collapsed variational Bayesian [36] on GPU. BIDMach [30] toolkit implements Monte Carlo Expectation Maximization (MCEM) [37] method on GPU without much GPU specific optimizations thus ends up with moderate performance. SaberLDA [28] proposes the PDOW (partition by document, order by word) strategy to reduce random memory access. Warp-based sampling is also adopted, which means using a warp to process a token and a block to process a word, to avoid thread-divergence and uncoalesced memory access. Further, cuLDA [29] scales LDA to multiple GPUs based on collapsed Gibbs sampling with similar optimizations as SaberLDA. In summary, **the curse of GPU-based LDA is the limited number of topics because they have to store W in dense format** - larger topics will exhaust the limited memory space of GPUs.

B. CONTRIBUTIONS

This paper introduces ezLDA,[1] an efficient and scalable LDA project that trains LDA across multiple GPUs. Notably, ezLDA can train LDA on UMBC dataset within 700 seconds while supporting the unprecedented 32,768 topics on merely one V100 GPU [38]. This achievement is not possible without the following contributions:

First, ezLDA introduces one more direction to make LDA efficient, i.e., the three-branch sampling method which takes advantage of the convergence heterogeneity of various tokens to reduce the redundant sampling task. While the convergence heterogeneity is promising, the caveat is that one cannot simply avoid sampling a token because its topic remains unchanged for a number of iterations, as detailed in Figure 3. Inspired by our key observation that the majority of the tokens often fall in the top popular topics, we single out these popular topics as the third sampling branch in addition to the traditional two branches (detailed in Section II-B). During sampling, we introduce an algorithm to accurately estimate whether this token will remain in the top popular topics thus be skipped or not. Meanwhile, in order to minimize the overhead of three-branch sampling, we introduce processing by both word and document strategy along with inverted index, and top topics pair-storage. Our evaluation shows three-branch sampling can avoid sampling over 70% of the tokens with negligible overhead after 100 iterations on large datasets, PubMed and UMBC.

[1] ezLDA, pronounced as "easy LDA", implies that this project achieves efficiency and scalability without the involvement of users.

Second, to enable sparsity-aware format for both **D** and **W** on GPUs with fast sampling and updating upon this format, we introduce hybrid format for **W** along with corresponding token partition to T and inverted index designs. Particularly, we store **W** in sparse and dense hybrid format and **D** in sparse format. During sampling, we will keep a canonical copy of the dense format of **W**, which accounts for the majority of the tokens but with very few number of words, in GPU memory to cache the updates. After sampling, we will update both the sparse part of **W** and **D**. Since sparse part of **W** holds very few tokens, the update is trivial. Pair-storing the row index and value of **D** is also adopted for fast sampling. For rapid update of **D**, we, again, leverage the inverted index of **D** to navigate through the token list T for tokens of interest. We also notice that SaberLDA [28] has attempted sparsity aware LDA but ends up with only sparse **D** which cannot solve memory exhaustion problem caused by dense **W** when vocabulary and topic size become too large. Consequently, ezLDA, as shown in Table 1, doubles the space saving over SaberLDA thus supports models that SaberLDA cannot.

Third, we improve the scalability of ezLDA across GPU threads and GPUs. For single GPU, we introduce hierarchical workload management to ultimately balance the workload which consists of two optimizations. Specifically, we first use atomic operation to dynamically decide which word should be assigned to which GPU warp thanks to light-weighted GPU atomic operation [39]. Further, we propose to split the extremely large words for better workload balancing. To efficiently combine dynamic inter-word scheduling and large word splitting design, we introduce efficient indexing to achieve light-weighted workload management. Towards multi-GPU support, we propose to cache **W**, partitioned **D** and partitioned T in GPU memory to further boost the performance of ezLDA.

The novelty of this paper is that we introduce the *efficient and scalable techniques* to achieve fast LDA training. Particularly, to the best of our knowledge, our three-branch sampling is the first successful design to exploit the convergence heterogeneity of various tokens for fast LDA sampling. We also shed lights on the possibility of using inverted index to achieve sparsity-aware LDA training where both **W** and **D** are sparse. It is also important to note that this paper strives to make sense of the complicated mathematical designs of LDA with an intuitive example which will also benefit the community. The code is open source in https://github.com/wojiushishen/tree/master/EZLDA.

C. ORGANIZATION

The rest of the paper is organized as follows: Section II explains the background. Section III presents the novel three-branch sampling design. Section IV discusses sparsity-aware LDA and scalable LDA is introduced in Section V. Section VI evaluates this work and we conclude in Section VII.

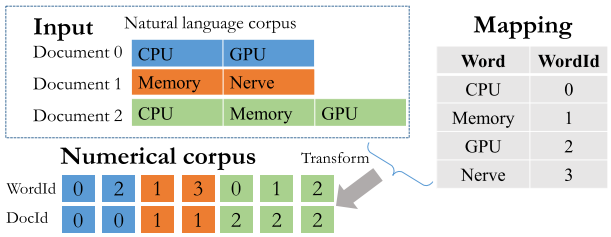


FIGURE 1. The running example used in this paper.

II. BACKGROUND AND CHALLENGES

A. GENERAL PURPOSE GPUS

Without loss of generality, this section uses Volta GPUs as an example to illustrate the essential backgrounds about modern GPUs, mainly from three aspects, that is, processor, memory and programming primitives. For more details about GPUs, we refer the readers to [40] and [41].

The streaming multiprocessor (SMX), which consists of several CUDA cores, is a basic processing chip for GPUs. For instance, Nvidia Tesla V100 GPUs [38] contain 80 SMXs, each of which has 64 single-precision CUDA cores and 32 double-precision units and a 96 KB shared memory/L1 cache and 65,536 registers. V100 also features 6MB L2 cache and 16 GB global memory, which is shared by all SMXs. Similar to CPU, the memory access latency increases from register to shared memory, further to L2 cache and global memory.

With massive CUDA cores, GPUs can run a large number of threads. A thread block is a programming abstraction that represents a group of threads that can be executed serially or in parallel, a thread block contains up to 1024 threads. A warp is a set of 32 threads within a thread block. It is important to mention that a warp of threads is executed in Single Instruction Multiple Thread (SIMD) fashion, which means all the threads in a warp execute the same instruction. In terms of programming primitive, recent GPUs provide several warp-level primitives such as *__shfl_sync()* and *__ballot_sync()* for fast intra-warp communication.

B. LDA ALGORITHM AND THEORY

Before explaining LDA designs, Figure 1 describes how to transform a real world natural language corpus into numerical corpus which can be used by LDA. Particularly, for a natural language corpus which consists of three documents with 2, 2 and 3 tokens, respectively, the preprocessing step will extract the unique words and assign each of them a specific wordId in mapping. This step is necessary because identical word might appear repeatedly, where each occurrence is called a token, e.g., memory appears in both documents 1 and 2. After transformation, we arrive at the numerical corpus.

1) ALGORITHM

LDA is a three-layer Bayesian model, that is, each document is viewed as a distribution of topics and each topic is further deemed as a distribution of vocabulary. For a given token,

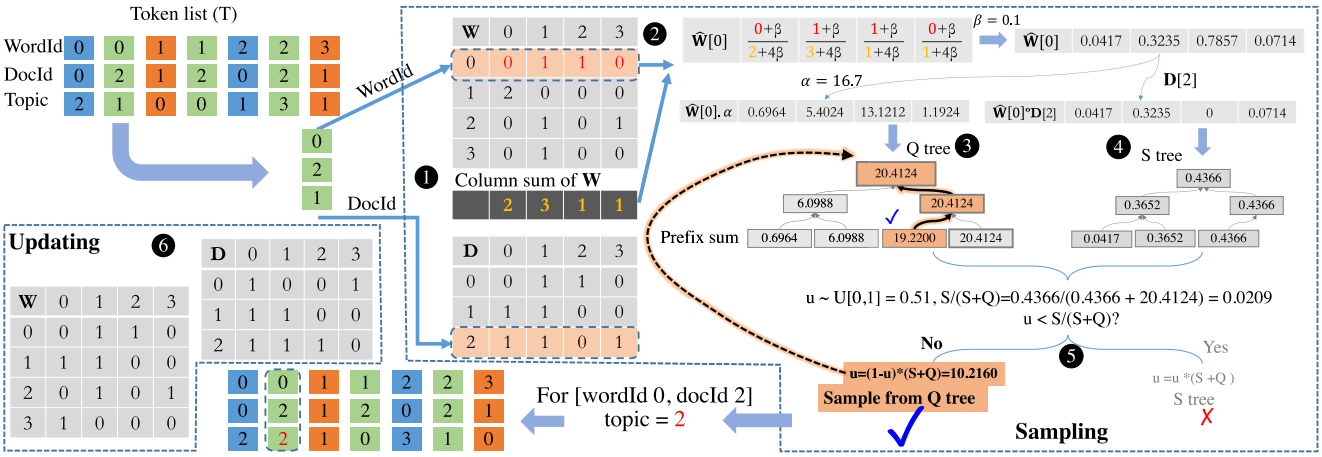


FIGURE 2. Two-branch sampling of one token for the corpus in Figure 1. Better viewed in color.

a new topic can be generated based on these two distributions. So, during training, two matrices are involved, i.e., **D** and **W**. While detailed theory behind why LDA would work can be found in [1], this paper focuses on the algorithm.

For each token, ESCA [42] - one of the popular LDA version - assigns this token to topic k, i.e., $p(k)$ through the following equation:

$$p(k) \propto \underbrace{(D[d][k]+\alpha)}_{part\,1} \cdot \underbrace{\frac{W[v][k]+\beta}{(\sum_{v=1}^{V} W[v][k]+V\beta)}}_{part\,2,\ \widehat{W}[v][k]}, \quad (1)$$

where α and β are two constant hyper parameters. Similarly to [28] and [29], we adopt $\alpha = 50/K$ and $\beta = 0.01$ for ezLDA, where K is the total number of topics. $D[d][k]$ is the number of tokens in document d that belongs to topic k in **D**. Similarly, $W[v][k]$ is the number of tokens of word v that belongs to topic k in **W**.

The intuition of Equation (1) is that, for token t that belongs to word v and document d, if more tokens from document d and word v fall in topic k, LDA will be more likely to assign topic k to this token t, that is, $D[d][k]+\alpha$ and $W[d][k]+\beta$ will be larger. Further, the total number of tokens in v - $\sum_{v=1}^{V} W[v][k]+V\beta$ - is negatively correlated.

Defining part 2 of Equation (1) as $\widehat{W}[v][k]$, which can be regarded as the normalized version of **W** matrix, we get:

$$p(k) \propto (D[d][k]+\alpha) \cdot \widehat{W}[v][k], \quad (2)$$

It is important to note that we choose to extend ESCA [42] because ESCA is sparsity-aware, which means the time complexity is sub-linear with respect to the number of topics. ESCA achieves this sparsity-aware goal by decomposing the part 1 of Equation (1) into two separate terms. So Equation (1) can be rewritten in the following format:

$$p(k) \propto= \underbrace{D[d][k] \cdot \widehat{W}[v][k]}_{p_s(k)} + \underbrace{\alpha \cdot \widehat{W}[v][k]}_{p_q(k)}, \quad (3)$$

Equation (3) can be further written into vector format:

$$p \propto (D[d]+\boldsymbol{\alpha}) \circ \widehat{W}[v] = \underbrace{D[d] \circ \widehat{W}[v]}_{p_s,\ \text{or S tree}} + \underbrace{\boldsymbol{\alpha} \circ \widehat{W}[v]}_{p_q,\ \text{or Q tree}}, \quad (4)$$

where $\circ$ is the Hadamard Product (HP) operator. $\widehat{W}[v]$ is the normalized v-th row of **W**. $D[d]$ is d-th row of **D**. $\boldsymbol{\alpha}$ is a vector with all elements to be α.

Finally, ESCA defines S and Q as the sum of p_s and p_q, respectively, sampling process of LDA becomes as follow. Note, we term this sampling method as two-branch because it has S and Q two branches.

- Generating a random number $u \sim U[0, 1]$.
- Generating the new topic by $\begin{cases} p_s, & \text{if } u \le \dfrac{S}{S+Q}; \\ p_q, & \text{otherwise.} \end{cases}$

2) EXAMPLE

Figure 2 presents one iteration of LDA on the same corpus as shown Figure 1 with randomly assigned topics. During initialization (❶), one will generate the **W** and **D** matrices from the token list T, where **W** and **D** are document-topic and word-topic matrices, respectively. Particularly, the dotted box in **W** means the document 0 has 0, 1, 1 and 0 tokens for topics 0, 1, 2 and 3, respectively. Similarly, the dotted box in **D** means that word 2 has 1, 1, 0 and 1 tokens for topics 0, 1, 2 and 3, respectively. Note, the column sum of **W** is also computed, as shown below **W**, which means, in total, we have 2, 3, 1 and 1 tokens for topics 0, 1, 2 and 3, respectively.

LDA training encompasses two steps, i.e., sampling and update. Further for sampling, LDA uses either Q or S tree to conduct sampling thus is called two-branch sampling. For the second token of the first word from the token list T - {0, 2, 1}, we follow Equation (1) to compute the $\widehat{W}[0]$ as $\{\frac{0+\beta}{2+4\beta}, \frac{1+\beta}{3+4\beta}, \frac{1+\beta}{1+4\beta}, \frac{0+\beta}{1+4\beta}\}$ (❷). Since $\alpha = 50/3 = 16.7$ and $\beta = 0.01$, we obtain $\widehat{W}[0] \circ \boldsymbol{\alpha}$ as {0.0818, 5.5477, 16.2190, 0.1603} and $\widehat{W}[0] \circ D[2]$ as {0.0049, 0.3322, 0, 0.0096}. Conducting prefix-sum [43] of $\widehat{W}[0] \circ \boldsymbol{\alpha}$ and $\widehat{W}[0] \circ D[2]$,

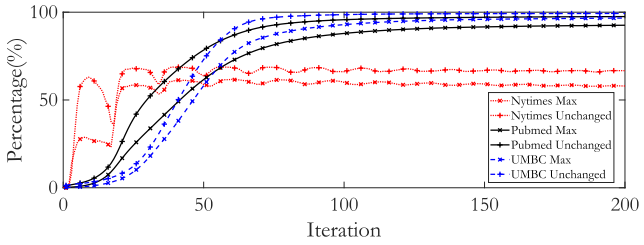


FIGURE 3. Percentage of tokens with unchanged topic and tokens corresponding to the max topic.

we arrive at the ranges of {0.0818, 5.6295, 21.8485, 22.0088} and {0.0049, 0.3371, 0.3467, 0.3467} for the Q and S tree, respectively.

The rule of tree construction is that the parent node should be the larger one of the two child nodes. Using the first two pairs of Q tree as an example, 5.6295 is the parent node of {0.0818, 5.6295} (❸). Similarly for the rest of Q tree and S tree construction (❹). The ❺ step draws a random number from uniform distribution $U[0, 1]$, u = 0.51 in this case, and compares it against $\frac{S}{S+Q}$ to decide which tree to sample in order to derive a new topic for this token. Since 0.51 is not smaller than $\frac{S}{S+Q}$, we use Q tree to conduct the sampling by adjusting $u = (1 - u) \cdot (S + Q) = 10.2160$ and descending the Q tree to arrive at new topic. u falls into interval (6.0988, 20.4124) and (6.0988, 19.2200) in level 1 and level 2 of Q tree, so the new topic will be 2. Following this way, LDA will update the topics for all the tokens T, subsequently the **D** and **W** matrices (❻).

Since T is sorted by wordId, we only need to construct Q tree once for all the tokens of the same word. S tree construction, in contrast, needs to be done more frequently because adjacent tokens often come from different documents, as shown in Figure 2.

3) EVALUATION METRIC

We use log-likelihood per token (LLPT), also known as negative logarithm of perplexity, as the parameter to evaluate the convergence of LDA.

$$LLPT = \frac{1}{N}\sum_{n=1}^{N}(log_2 \sum_{k=1}^{K}(\frac{D[d][k]+\alpha}{\sum_{k=1}^{K} D[d][k]+K\alpha} \cdot \frac{W[v][k]+\beta}{\sum_{v=1}^{V} W[v][k]+V\beta})), \quad (5)$$

where N is the total number of tokens in this corpus. The rule is that LLPT should increase and gradually become stable when computation proceeds iteratively.

III. THREE-BRANCH FAST SAMPLING

This section discusses two important observations, our three-branch sampling and implementation optimizations that lower the overhead of three-branch sampling.

A. OBSERVATIONS

This section makes the following two important observations that inspire our three-branch sampling.

First, different tokens converge at dissimilar speeds. As shown in Figure 3, when iteration proceeds, more and more tokens experience unchanged topics. In other words, some tokens converge earlier and some later. For instance, at 50-th iteration, over 70% of the tokens keep their topic unchanged in PubMed dataset.

Second, the majority of the tokens tend to converge to the most popular topic. This observation is self-explanatory - because a topic contains more tokens, and becomes the most popular topic. In fact, Figure 3 quantitatively showcases this observation. In particular, more than 60% of the tokens converge to the most popular topic in PubMed dataset at 50-th iteration. Note, we also have tested and verified the EZLDA convergence on other datasets such as Enron, NIPS and KOS datasets, it shows similar convergence trend as PubMed dataset.

The first observation implies that we can reduce the sampling workload for early converged tokens. However, since reducing the sampling task needs extra checking operations, this might incur significant overhead. Fortunately, our second observation further suggests that we may lower the overhead by only focusing on the most popular topic.

B. THREE-BRANCH SAMPLING

Since traditional two-branch sampling cannot leverage our observations in Section III-A for workload reduction, we introduce three-branch sampling which singles out the most popular topic as one more branch. Below we discuss the theoretical soundness and implementation details of this design. Note, we cannot simply avoid sampling all the tokens from the most popular topic because, as discussed in Section IV-D, very few tokens from the most popular topic might change their topic, though more tokens will converge to the most popular topic.

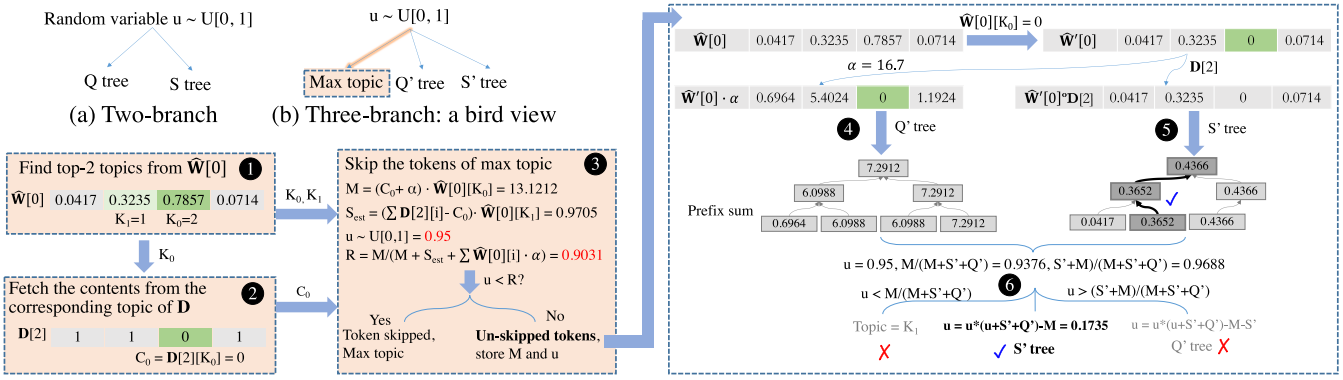
1) THEORETICAL SOUNDNESS

Our three-branch sampling rewrites Equation (3) into the following format:

$$p \propto \underbrace{D[d] \circ \widehat{W}'[v]}_{p_s} + \underbrace{\boldsymbol{\alpha} \circ \widehat{W}'[v]}_{p_q} + \underbrace{(D[d]+\boldsymbol{\alpha}) \circ \widehat{W}[v]^m}_{p_m}, \quad (6)$$

where $\widehat{W}'[v]$ is derived by setting the maximum entry of $\widehat{W}[v]$ as 0. On the contrary, $\widehat{W}[v]^m$ is achieved by setting all except the maximum entry of $\widehat{W}[v]$ as 0.

Consequently, p_m has only one non-zero entry which corresponds to the most popular topic. As shown in the left of Figure 4(a), traditional two-branch sampling approach conducts sampling from two branches – either S or Q tree. Particularly, S and Q trees are constructed from p_s and p_q in Equation (4), respectively. The proposed three-branch sampling, as shown in the right of Figure 4(a), consists of three branches. That is, S' and Q' trees which are constructed from



(c) Detailed workflow of three-branch sampling.

FIGURE 4. ezLDA three-branch sampling: (a) Two-branch vs (b) ezLDA three-branch sampling, a bird view and (c) Detailed workflow of three-branch sampling.

p_s and p_q in Equation (6), respectively, and the max topic branch which is the p_m in Equation (6).

Three-branch sampling is exemplified by Figure 4(b). For each unique word, ezLDA first finds the top-2 topics in $\widehat{W}[v]$, which are $K_0 = 2$ and $K_1 = 1$ (❶). Here "top-2 topics" means these topics correspond to top-2 largest values, 0.7857 and 0.3235, in $\widehat{W}[v]$. Then given the third topic is the most popular one, we will extract the number of tokens from the same index in $D[d]$, that is, 2 (❷). Consequently, $K_0 = 2, K_1 = 1$ and $C_1 = 2$. Afterwards, as shown in ❸ of Figure 4(b), we can calculate $M = 13.1212$ and $S_{est} = 0.9705$ and generate $u \sim U[0, 1]$. Compare u against $\frac{M}{M+S_{est}+Q'}$ to determine whether this token remains in the most popular topic. If yes, this token will not involve in the following steps and corresponding topic will be updated to be K_0. Otherwise, store M and u for this un-skipped token. Finally, we will execute the remaining steps (❹, ❺ and ❻), which are similar to two-branch sampling except following two differences: First, these steps only need to be done for the remaining un-skipped tokens. As training goes, more and more tokens are skipped and linear time decrease will be introduced. Second, max topic is singled out and considered separately. So $\widehat{W}[K_0]$ should be set to be zero and final sampling will include an additional M branch, as shown in the bottom right of Figure 4(b), even after construction of S' tree, we can still avoid the sampling if $u < \frac{M}{M+S'+Q'}$. Besides, the Q' tree and S' tree (❹ and❺) constructions are the same as two-branch sampling method.

2) S_{est} COMPUTATION

In order to skip as many tokens as possible, ezLDA needs to make S_{est} as close to S' as possible. Meanwhile, to ensure theoretical soundness, we must also make sure tokens that go to 'Yes' branch in step ❸ must belong to the left branch in step ❻, which requires S' not greater than S_{est}. We use the following inequality to extract the S_{est} and calculate M.

Assuming $\widehat{W}[v]$ is sorted in descending order:

$$\begin{aligned} \widehat{W}[v] &= [a_1, a_2, \ldots, a_n], \\ D[d] &= [b_1, b_2, \ldots, b_n]. \end{aligned} \tag{7}$$

This means $a_i > a_j$ if $i > j$. Thus, maximum topic branch is:

$$M = a_1 \cdot (b_1 + \alpha). \tag{8}$$

Given

$$\begin{aligned} S' &= \widehat{W}[v] \cdot D[d] - M + a_1 \cdot \alpha \\ &= (a_2 \cdot b_2 + a_3 \cdot b_3 + \cdots + a_n \cdot b_n) \\ &< \sum_{2 \le i \le g} a_i \cdot b_i + a_{g+1} \cdot \sum_{g < i \le n} b_i, \end{aligned} \tag{9}$$

we hence propose:

$$S_{est} = \sum_{2 \le i \le g} a_i \cdot b_i + a_{g+1} \sum_{g < i \le n} b_i. \tag{10}$$

where $g \ge 1$ controls the **accuracy and cost** of the estimation. That is, the larger the value of g, the higher the cost, as well as the accuracy between S' and S_{est}.

3) PARAMETER TUNING

First, the choice of g is a trade-off between benefit and overhead. ezLDA uses $g = 2$ because we can achieve significant better performance than g=1 with similar overhead after our optimization in Section III-C.

C. OPTIMIZATIONS FOR THREE-BRANCH SAMPLING

While three-branch sampling can avoid expensive S' tree constructions and sampling for all the skipped tokens, it also introduces three more steps, i.e., ❶, ❷ and ❸ as shown in Figure 4(b).

Across all the steps, the cost for steps ❶ and ❸ is negligible. For step ❸, the workload is simple. For step ❶, the reason lies in that the token list (i.e., T) is sorted by wordId, as shown in Figure 5(a), three-branch sampling only needs to find the top

FIGURE 5. Inverted index for document.

topics, that is, K_1 and K_2 pair in Figure 4(b) - once for all the tokens falling to the same word v. But also because T is sorted by wordId, step ❷ would take significant amount of time if we want to find the corresponding C_1 and C_2 pair across all documents for each v right after we find K_1 and K_2 pair.

In order to combat this overhead, ezLDA designs processing by word and document for K_1 and K_2 pair, and C_1 and C_2 pair, respectively. While processing by word is straightforward because T is sorted by word, processing by document turns out to be challenging. In this context, we introduce an inverted index for each document which stores the indices of tokens belonging to each document. This inverted index idea adopts Compressed Sparse Row (CSR) format [44], [45] to store the indices of the tokens for each document. As shown in Figure 5(b), indices {0, 4}, {2, 6} and {1, 3, 5} are from documents 0, 1 and 2, respectively.

Scanning through the inverted index, we can fetch the corresponding row of **D**, as well as all the tokens of the same document easily. Note, in this processing by document design for C_1 and C_2 pair, we need to first write all the K_1 and K_2 pairs for all tokens into global memory and load them back for computation. However, this cost is way lower than we conduct processing by word for both K_1 and K_2 pair and C_1 and C_2 pair. Particularly, processing by word for C_1 and C_2 pair needs to repeatedly scan through the token list and search for C_1 and C_2 for each token because tokens of the same document are not stored together. It is also worthy to note that we combine K_1 and K_2, C_1 and C_2 pairs into a single value K_{12} and C_{12}, where the higher half bits of them store K_1 and C_1, and the lower half bits store K_2 and C_2 respectively, in order to further reduce the overhead.

IV. SPARSITY-AWARE OPTIMIZATION

Reducing the sampling time is important for LDA, so does the space consumption. This section introduces the sparsity-aware storage format for both **D** and **W**, as well as our new mechanisms to facilitate rapid sampling and updating dwelling on these sparse formats.

A. OBSERVATIONS

The space problem faced by ezLDA appears for two types of data, that is, corpus data and algorithmic data. Corpus data is concerned with the gigantic corpus size while algorithmic data is related to both corpus size and number of topics. Below, we discuss the details surrounding these two challenges.

The space consumption incurred by the large corpus can be tackled by simply partitioning the corpus into multiple chunks. This way, each GPU will need much less memory

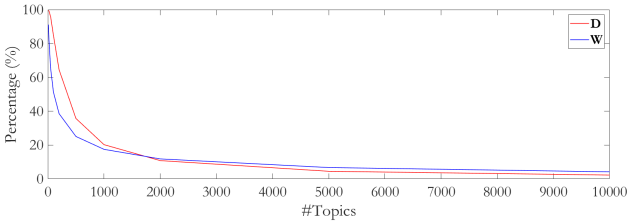


FIGURE 6. Density of W and D on NYTimes dataset.

for corpus. Doing so, however, comes with one obvious drawback - one needs to repeatedly load each chunk in and out the GPU, which could introduce overhead. ezLDA uses asynchronous kernel launching and memory transfer to hide this cost, similar to existing work [46], [47].

In fact, curbing the space consumption of algorithmic data (e.g. **D** and **W**) is even more imperative. Below, we unveil the reason from column and row perspective of a matrix. First, with the climbing of the corpus size, the diversity of the tokens will also increase, indicating the need of a larger number of topics (i.e., number of columns in **D** and **W**). Second, for a corpus with abundant documents or unique words, the number of rows in both **D** and **W** will also soar.

The good news is that both **D** and **W** are often very sparse because very few, if any, of the words or documents will occupy all the topics. As shown in Figure 6, the density of **D** and **W** decreases rapidly along with the increase of number of topics for the NYTimes dataset.

B. SPARSITY-AWARE REPRESENTATION

We introduce compressed CSR format to store the sparse **W** and **D** matrices. While the traditional CSR contains three major components: *row offset, column indices* and *values*, we further compress *column indices* along with *values* in order to save space and improve performance. Inspired by the pair-storage in Section III-C, we compress the column indices and values of CSR into and a single integer, where the higher and lower half bits are for column index and corresponding value, respectively.

Storing the entire **W** in sparse format might not always save space. Particularly, despite sparse format will save memory space for sparse rows in **W**, words with large number of topics (i.e. dense rows) will, unfortunately, suffer from extra space consumption because CSR requires to store the column indices. In contrast, dense format only needs to store the values since the position of the value can automatically indicate its column location.

We thus advocate to store **W** in hybrid format. That is, the rows with large volume of nonzero columns (i.e., topics) will be stored in dense format while the remaining rows in sparse format. ezLDA comes up with a light-weighted heuristic to estimates the upper bound of **W** in order to decide whether we store a corresponding row in sparse or dense format. That is, the maximum number of topics one word can possess will not go beyond the number of tokens this word has in the

entire corpus. With this rule, one can assign the words with tokens that is larger than the assumed number of topics (i.e. K) as dense row and the remaining rows to be the same as the number of tokens. For example, if K = 1000, then a word which has a total of 999 tokens in the corpus will be assigned to at most 999 topics and thus can be stored in sparse format. Similarly, a word which has a total of 1001 tokens will be stored in dense format.

To enjoy the space saving from the hybrid format, we propose to group dense words together in token list T. Toward that end, the word identities (IDs) are relabeled based upon their token counts. Basically, words with larger number of tokens hold smaller IDs. Further, words with token count more than the topic number are stored in dense format in **W**. Subsequently, in each chunk from the token list, we remap the word IDs from the token list into T_{dense} and T_{sparse}, respectively, which represent the dense and sparse parts of T, respectively.

In summary, this hybrid approach comes with the following two advantages. First, comparing to the dense or sparse alone approach, the proposed method will yield the most space saving. Second, storing dense rows into dense format explicitly will reduce the overhead of $\widehat{W}[v] \circ D[d]$.

C. SPARSITY-AWARE COMPUTATION

Once storing **W** and **D** in sparse format is resolved, conducting updating and sampling atop the sparse **W** and **D** become a ground challenge for two reasons. First, during sampling, we need to do element-wise product of $\widehat{W}[v] \circ D[d]$. Given the elements from the same storage index of sparse $\widehat{W}[v]$ and **D**[d] are most likely not from the same column, we will need to match their columns. Second, to update an element in sparse matrix, we must first find the correct row and column to write the update. So unlike dense matrix, it is impossible to update **W** once after a new topic is known in sampling kernel. The update of **W** can only be done by reconstructing from T after sampling, which will consume much more time.

A naive design could easily combine sampling with updating via keeping a canonical copy of **W**. During sampling, this design will compute the S' and Q', thus the ratio $t_1 = \frac{S'}{S'+Q'+M}$ and $t_2 = \frac{Q'}{S'+Q'+M}$. Based upon the generated random number u, this design could determine to sample from either S' or Q' tree. After arriving at the updated topic for a token, we can immediately update the canonical copy.

However, this naive design also faces two challenges. First, keeping extra canonical copies for **W** will consume more space. Second, it is hard to predict, for a random token, where to update the canonical copies of **W** providing they are in the sparse format. Third, given LDA is memory intensive, reading **W** twice (one for sampling, the other for updating) will hamper the performance.

We only keep a canonical copy for $\mathbf{W}_{dense}$ and reconstruct $\mathbf{W}_{sparse}$ as well as the entire **D** from T after sampling. Below, we discuss the details. ezLDA keeps a canonical copy of $\mathbf{W}_{dense}$ for update because the words in the dense rows often contain exceeding number of tokens which span across multiple chunks. In that context, we would need to transfer a large number of chunks if we choose to reconstruct $\mathbf{W}_{dense}$ from T. In contrast, with a canonical copy of $\mathbf{W}_{dense}$, the update to $\mathbf{W}_{dense}$ can be done quickly because $\mathbf{W}_{dense}$ is in dense format, as well as in memory during sampling.

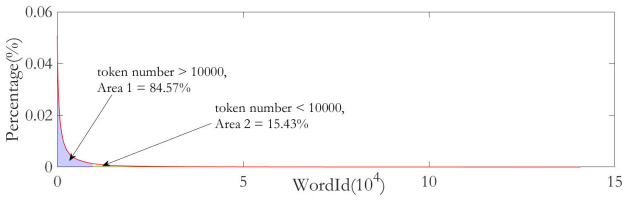


FIGURE 7. Token distribution of PubMed dataset.

For $\mathbf{W}_{sparse}$, we only need to read in the $\mathbf{T}_{sparse}$ part of the token list. Since more than 80% tokens contribute to $\mathbf{W}_{dense}$, $\mathbf{T}_{sparse}$ will be relatively small. Thus reconstructing $\mathbf{W}_{sparse}$ will be very fast.

Since a majority of the tokens belong to $\mathbf{W}_{dense}$ which is updated during sampling, the update of the entire **W** usually consumes very short time.

The update of **D** is aided by the inverted index which is discussed in Figure 5(b). In order to discuss the update to **D**, we need to understand how ezLDA partitions and preprocesses the corpus. Particularly, each document is solely assigned to one chunk, and all the tokens in each chunk are sorted by wordId. Here, assuming this chunk contains three documents of a corpus. This way, we can reconstruct three rows of **D** with this chunk. Inside of each chunk, the tokens are sorted by wordId for ease of update of $\mathbf{W}_{sparse}$. Towards updating **D**, we resort to the inverted index in Figure 5(b). Particularly, one can scan through the CSR to decide which tokens are needed to update rows 0, 1 and 2 of **D**.

With the updating process being taken care of, sampling becomes the immediate bottleneck. To facilitate a fast S' tree construction, which involves the HP between two CSR rows of **W** and **D**, we reconstruct the entire row of **W** into dense format in shared memory. Afterwards, HP is done by scanning through the specific row of **D** and use the column index to access that of the dense **W**. Note, this shared memory will be repeatedly used for all rows of **W**.

D. DISCUSSION

This section shares a failed trial. Inspired by the traditional iterative graph computing algorithms, such as, delta-step Pagerank [48] and Single Source Shortest Path [49], [50], we falsely assume the tokens that already converged will no longer change their topics. Therefore, our naive design introduces a tracker array to indicate whether the topic of a token remains unchanged for several iterations. If so, this naive dropping method will not sample this token in the following iterations.

However, the naive dropping strategy fails to work mainly because it betrays the nature of LDA. Particularly, the core of

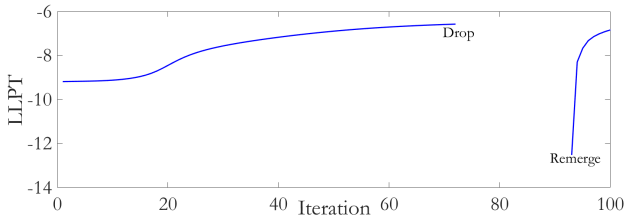


FIGURE 8. Perplexity of naive dropping strategy on PubMed.

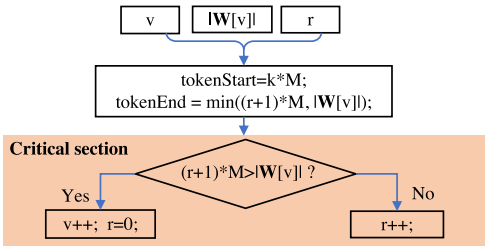


FIGURE 9. Hierarchical workload balancing. Note, v, $|W[v]|$, r and M are the word index, number of tokens for word v, region index, and maximum number of token processed by each index increment, respectively.

LDA, i.e., Bayes model, is that the sampling process of LDA is a non-deterministic process. That is, even if the topic of a token remains unchanged for several iterations, which means the probability of assigning this specific token to the same topic is very close to 1, this token still has a chance to change topics because the random number generated from $U[0, 1]$ might fall in other topics whose probabilities are low.

Figure 8 shows the failure of the naive dropping strategy. In this test, the dropping starts at iteration 72. At iteration 90, all dropped tokens are re-included in the training to check whether this strategy works. Clearly, the results are not good. At the point of re-including, perplexity becomes even smaller than the value before dropping and severely deviates from the correct convergence point.

V. SCALABLE ezLDA ON GPUs

This section discusses novel techniques we exploit to better scale ezLDA across GPU threads, as well as GPUs.

A. INTRA-GPU WORKLOAD BALANCING

For a corpus, the number of tokens per word often follows power law, that is, a few high frequent words occupy majority of tokens, as shown in Figure 7. The workloads associated with various words are hence largely unbalanced. However, the contemporary LDA projects [28] typically assign a block to a word, regardless of the associated workload, leading to severe workload imbalance issue in LDA training. This section thus introduces two methods to overcome the workload imbalance problem, that is, *dynamic workload balancing* for small words and *workload splitting* for large words.

1) SMALL WORD

Given various words come with different number of tokens, we adopt the dynamic workload balance strategy from a recent work [39] to address the inter-word workload imbalance issue. Note, instead of processing the words by each block in a pre-determined manner, this approach will use **atomicAdd**() to, on-the-fly, determine which word will be processed by the available thread block.

2) LARGE WORD

While applying the dynamic workload balance strategy can largely address the inter-word workload imbalance problem faced by *small words*, it will not work for *large words* which govern too many tokens. In this context, the block that processes the extremely large words will become the straggler. For instance, assuming one corpus has 128 tokens and the most frequent word holds 50 tokens, we use 8 blocks for training. Then, each block should process 16 tokens in workload balanced setting. However, with the dynamic workload balance strategy, the block that processes the word with 50 tokens will be responsible for this entire word alone, leading to workload imbalance.

In order to solve this problem, we introduce *large word dissection*, i.e, very high frequency words are partitioned and processed by multiple thread blocks. Particularly, we can quickly derive the maximum number of tokens a block can process through dividing the total amount of workloads by the number of thread blocks. If the token number of a specific word is larger than this maximum value, we will partition this word into several parts and assign them to multiple blocks. In this work, we use 10,000 as the threshold for ezLDA.

It is important to note that applying dynamic small word workload balancing and large word dissection together will pose challenges for word assignment. For instance, we need to decide which word and what portion of that word are the next workload. ezLDA introduces a two-level index strategy to deal with this challenge.

Figure 9 shows the design of our two-level index strategy. Word index v determines the word to be processed and region index r determines which region of tokens in that word should be processed. Apparently, the increment of v and r are correlated and must be executed atomically. Considering an atomic function can only be used for a single operation, we propose to use critical section to fulfill that goal. To remedy the absent of critical section support on GPU, ezLDA relies on atomic operations to build a critical section [51].

B. MULTI-GPU ezLDA

As the size of corpus and number of topics continue to grow, the training time of LDA also prolongs, which leads to our support of multi-GPU ezLDA. When extending to multiple GPUs, ezLDA is concerned with two essential data structures, that is, data (i.e., T) and algorithmic data (i.e., **D** and **W**), and the correlated workload partition, and communication.

The good news is that T and **D** are well partitioned in the single GPU-based design, as discussed in Section IV. Particularly, each chunk is responsible for similar number

TABLE 1. ezLDA vs cuLDA/SaberLDA memory consumption on PubMed dataset. The corpus is partitioned into 8 chunks during computation.

Method	K	W	D	T	Total
cuLDA/SaberLDA	1,000	1.08 GB	1.45 GB	2.16 GB	4.69 GB
	10,000	10.8 GB	1.45 GB	2.16 GB	14.41 GB
	32,768	35.4 GB	1.45 GB	2.16 GB	39.01 GB
EZLDA	1,000	0.31 GB	0.98 GB	4.97 GB	6.26 GB
	10,000	1.63 GB	0.98 GB	4.97 GB	7.58 GB
	32,768	2.5 GB	0.98 GB	4.97 GB	8.44 GB

of documents. This partition of T leads to evenly partitioned **D** across GPUs. And each chunk actually contains a similar number of tokens. Using UMBC dataset on four GPUs as an example, the maximum and minimum workload chunks only have a difference of 5% in terms of the number of tokens.

For word topic matrix, i.e., **W**, unlike the single GPU version, we keep an in-memory canonical copy for both W_{dense} and W_{sparse}. After all chunks are processed, we can update both W_{dense} and W_{sparse} by summing up the canonical copies across all GPUs and broadcasting the result back to all of them.

VI. EXPERIMENTS

We implement ezLDA with ∼4,000 lines of C++/CUDA code and compile the source code with Nvidia CUDA 9.2 toolkit and -O3 optimization compilation flag. We use two platforms to evaluate ezLDA. For comparison with state-of-the-art SaberLDA, we use an Nvidia GTX 1080 GPU - identical platform used in SaberLDA - on an Alienware with 24 GB memory and Intel(R) Core(TM) i7-8700 (3.20Hz) CPU. For ezLDA internal study, we use a customized server which installs a dual-socket Xeon processor with 24 cores, and four Nvidia V100 GPUs. Note, each reported result is an average of five runs, where the differences across various executions are very small (< 1%).

Dataset: We evaluate ezLDA with two popular datasets that are also studied by cuLDA [29] and SaberLDA [28]:

- PubMed [52]: 8,200,000 documents, 141,043 unique words and 738M tokens.
- NYTimes [52]: 299,752 documents, 101,636 unique words and 100M tokens.

To better study the scalability and real-world impacts, we further prepare the following dataset by text splitting, stop words removing and non-frequent words stemming:

- UMBC: 40,000,000 documents, 200,000 unique words and 1.33 billion tokens. This dataset is obtained from UMBC webbase corpus [53].

A. ezLDA *vs. STATE-OF-THE-ART*

Table 1, Figures 10 and 11 compare ezLDA against the state-of-the-art, i.e., SaberLDA and cuLDA for space complexity, convergence speed and throughput (number of tokens per second), respectively.

1) SPACE

As shown in Table 1, ezLDA consumes 33% more space for small K = 1,000 compared with SaberLDA and cuLDA. But

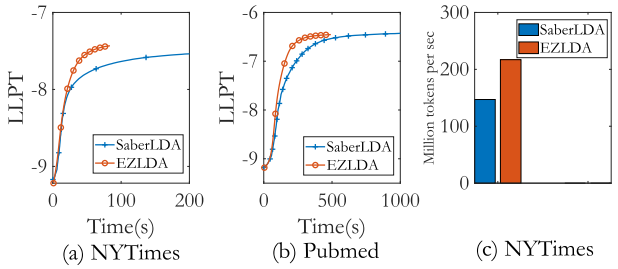


FIGURE 10. (a) The convergence of ezLDA vs SaberLDA with 1,000 topics on NYTimes dataset on GTX 1080. (b) The convergence of ezLDA vs SaberLDA with 1,000 topics on Pubmed dataset on GTX 1080. (c) Throughput of ezLDA vs SaberLDA for first 100 iterations on NYTimes on GTX 1080.

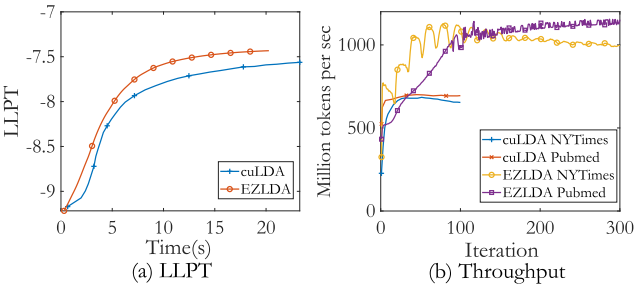


FIGURE 11. number of tokens per second for ezLDA vs. cuLDA on V100:(a) NYTimes, (b) Pubmed, (c)LLPT on Nytimes.

we save 47% and 78% space when K is large, e.g., 10,000 and 32,768. Particularly, ezLDA require more space than SaberLDA and cuLDA for T because we need to allocate space in T for K_1/K_2 pair (Section III-C) and M (Equation 8). However, ezLDA manages to save much more memory on **W** thanks to sparsity aware representation.

2) ezLDA vs SaberLDA

Figure 10 compares ezLDA against SaberLDA on convergence speed and throughput, i.e, number of tokens per second. Since SaberLDA is not open source, we cite the performance numbers from their manuscript and run ezLDA on identical GPU for fair comparison. As shown in Figures 10(a) and 10(b), ezLDA climbs to higher perplexity with shorter training time. For throughput, as shown in Figure 10(c), ezLDA achieves 1.5× speedup, on average, for the first 100 iterations on NYTimes. Note, since SaberLDA does not include the number of tokens per second statistics, we follow cuLDA to derive this number for NYTimes according to Figure 9 in SaberLDA [28].

3) ezLDA vs cuLDA

Though cuLDA outperforms SaberLDA, ezLDA, as shown in Figure 11(a), still manages to convergence faster than cuLDA on NYTimes (cuLDA does not include LLPT for PubMed). Thanks to cuLDA which includes number of tokens per second for both datasets, we report the comparison of this metric in Figure 11(b). Particularly, ezLDA achieves an average throughput of 905 and 770 million tokens/second for the

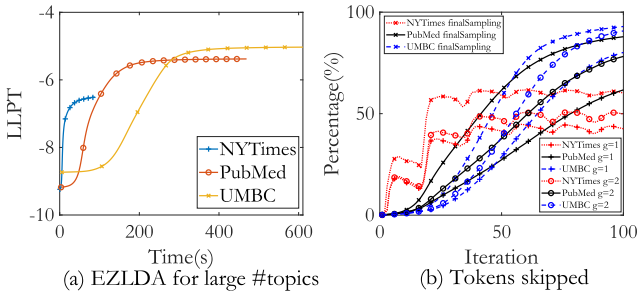


FIGURE 12. (a) ezLDA for large K, i.e., 32,768. (b) The percentage of tokens skipped by three-branch sampling for K = 1,000, where *g* is the parameter from Equation 10.

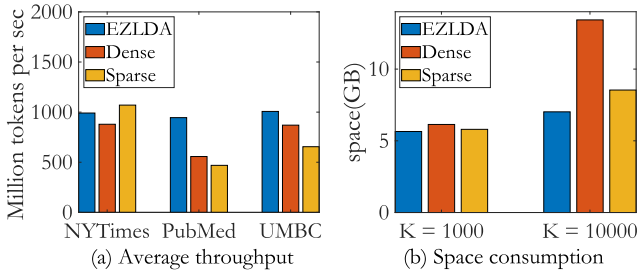


FIGURE 13. ezLDA hybrid representation vs. dense-only and sparse-only representations for topic = 1,000: (a) throughput and (b) space complexity for UMBC dataset.

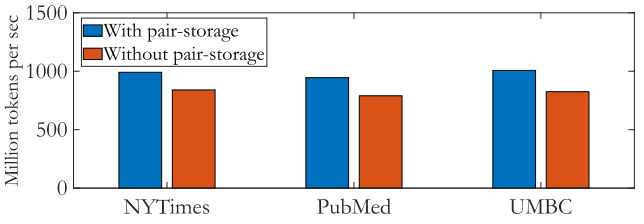


FIGURE 14. The performance impacts of pairStorage on V100 GPU for K = 1000.

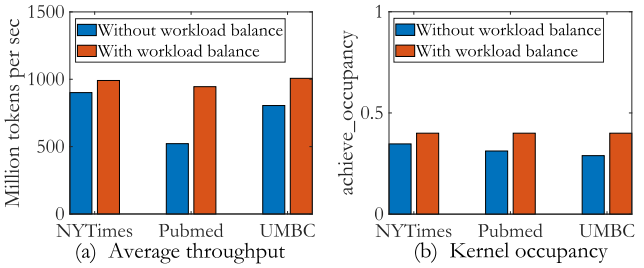


FIGURE 15. Profiling the impacts of intra-GPU workload balancing: (a) number of tokens per second for first 300 iteration (b) achieve_occupancy.

first 100 iterations and retains over 1000 million tokens/sec after 100 iterations on NYTimes and PubMed, respectively. This outperforms cuLDA that retains 633 and 686 million tokens/second, on average, for the first 100 iterations on NYTimes and PubMed, respectively.

B. ezLDA *PERFORMANCE STUDY*

1) LARGE NUMBER OF TOPICS

As shown in Figure 12(a), ezLDA can handle all the datasets with 32,768 topics on a single GPU. This is an important capability for real-world scale corpus [8].

2) THREE-BRANCH SAMPLING

Figure 12(b) profiles the impact of three-branch sampling. In general, we find this method is more effective when dealing with larger dataset. Particularly, NYTimes enjoys skipping 60% of the tokens during the final sampling and nearly 50% tokens skip the S construction at iteration 100. For PubMed, 87% tokens skip the final sampling and nearly 74% tokens skip the S construction at iteration 100. For UMBC, 93% tokens skip final sampling and nearly 89% tokens skip the S' construction at iteration 100. We also study different g in Equation 10. As expected, more tokens are skipping S' construction for larger g, because larger g makes S_{est} closer to S'.

3) HYBRID STORAGE OF W

Figure 13 profiles the impact of dense/sparse hybrid representations. The key conclusion is that our hyrbid optimization can both improve performance and save space (at least for the large K case). As shown in Figure 13(a), on average, hybrid format is 1.34× and 1.47× faster than the dense and sparse only formats, respectively. Compared with ezLDA, sparse format needs to update **W** after all chunks are processed, which means all chunks need to be transferred back to GPU a second time to finish the update. For dense format, much time will be wasted on updating rows of **W** corresponding to small words. Further, the hybrid format consumes 17.8% and 47.8% less space than sparse format and dense format for K = 10,000, respectively.

Pair K1/K2, C1/C2 and D storage, as shown in Figure 14, yields 1.12×, 1.19× and 1.22× speedup on NYTimes, PubMed and UMBC datasets, respectively. The speedup is achieved because LDA training is memory-bound [54] and pair-storage significantly reduces the global memory traffic in three-branch sampling.

C. *SCALABLE* ezLDA

1) HIERARCHICAL WORKLOAD BALANCING

Using three-branch sampling without workload balance as baseline, as shown in Figure 15(a), on average, our hierarchical workload balancing technique yields 1.1×, 1.7× and 1.2× speedup on NYTimes, PubMed and UMBC, respectively, for number of tokens per second. The speedup on PubMed dataset is higher because this dataset presents higher workload imbalance. The speedup is resulted from that workload balancing can improve the GPU occupancy [55]. As shown in Figure 15(b), we improve the achieved_occupancy ratio by 27% across the datasets. Note, achieved_occupancy means the ratio of active warps over maximum number of supported warps on the multiprocessor.

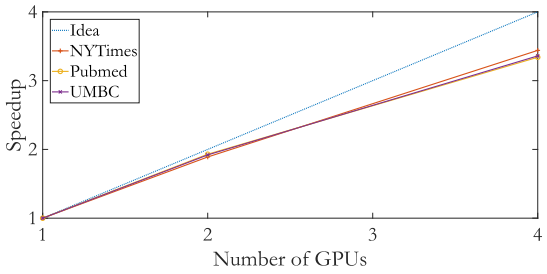


FIGURE 16. The performance impacts of scalable ezLDA.

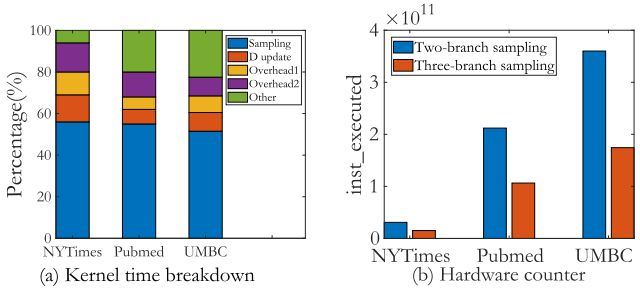


FIGURE 17. Profiling ezLDA: (a) kernel time breakdown (b) Hardware counter.

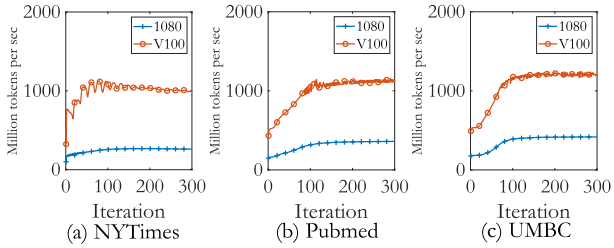


FIGURE 18. Throughput impacts of GTX 1080 vs. Volta V100. (a) NYTimes (b) Pubmed (c) UMBC.

2) MULTI-GPU SCALABILITY

Figure 16 shows that ezLDA can scale to four V100 GPUs with 3.44×, 3.34× and 3.36× speedup on NYTimes, PubMed and UMBC dataset, respectively. While this result indicates that ezLDA is scalable, we also notice that ezLDA cannot achieve linear scalability. The reason lies in the need of communicating **W** and the slight workload imbalance across partitions.

D. PERFORMANCE COUNTER AND GPU GENERATION IMPACTS

Figure 17(a) studies the time consumption breakdown in three-branch optimization. Though three-branch can skip tremendous tokens, it also introduces two noticeable overheads, i.e., steps ❷ and ❸ in Figure 4(c). On average, these two steps consume 8% and 12% of the total runtime, respectively. Figure 17(b) further profiles the microarchitectural impacts of three-branch optimization. Particularly, we profile the inst_executed [55] and find that three-branch optimization can reduce the executed instructions by 49%, on average, across the three datasets.

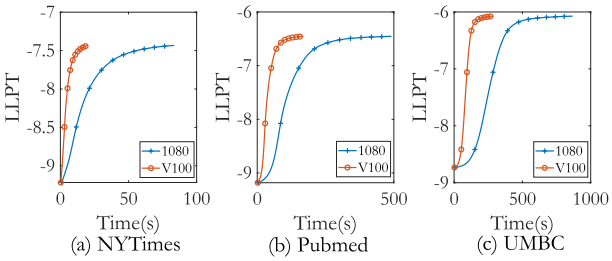


FIGURE 19. Convergence speed impacts of GTX 1080 vs. Volta V100. (a) NYTimes (b) Pubmed (c) UMBC.

Figures 18 and 19 study the GPU generation impacts on number of tokens per second and convergence speed for ezLDA, respectively. Since the bandwidths of GTX 1080 [56] and V100 [38] are 320 GB/s and 900 GB/s, respectively, we expect the performance impacts would also be around 3×. As shown in Figures 18, ezLDA can achieve an average of 991, 945 and 1007 million tokens/sec on V100 GPU and 250, 311 and 363 million tokens/sec on GTX 1080 GPU on NYTimes, Pubmed and UMBC datasets respectively. As shown in Figure 19, ezLDA also converges significantly faster on V100 than GTX 1080.

VII. CONCLUSION

In this paper, we present ezLDA that achieves superior performance over the state-of-the-art attempts with lower memory consumption. Particularly, we find that majority of tokens tend to converge to the most popular topic with training process and thus introduce a novel three-branch sampling method which takes advantage of this convergence heterogeneity of various tokens to reduce the redundant sampling task. Further, to enable sparsity-aware format for both **D** and **W** on GPUs with fast sampling and updating, we introduce hybrid format for **W** along with corresponding token partition to T and inverted index designs. Last but not the least, we design strategies to balance workload across GPU threads and scale ezLDA across multiple GPUs.

In the future, we will utilize the convergence heterogeneity of LDA and apply it on some advanced LDA models such as correlated topic model [57] and hierarchical LDA model [58].

APPENDIX

Algorithm 1 shows pseudocode of ezLDA. Before training, T_{dense} and T_{sparse} are transferred from CPU to GPU (line 3) to generate W_{dense} (line 4) and W_{sparse} (line 5), *MPT_Generate* kernel (line 6) is run to get most and second most popular topic K_0, K_1 and Q for each word. Once the training started, ezLDA transfers T_{dense} and T_{sparse} from CPU to GPU (line 10). Then **D** are updated based on the token list and C_0, which is $K_1's$ value in row of **D**, is also acquired for each token (line 11). After that, we have C_0, K_0, K_1, W_{dense} and Q, run *MPT_Calculate* kernel to filter the skipped tokens and assign most popular topic to these tokens. Meanwhile, for every un-skipped token, get the most popular topic $K_0's$ estimated probability M and random float u(line 12). Now we have

T_{dense}, W_{dense}, M, u and **D** in GPUs, we will run *Sampling*(D) kernel to update T_{dense} and W'_{dense} (line 13). When T_{sparse}, W_{sparse}, M, S_{est} and **D** are ready, we opt to *Sampling*(S) kernel to only update T_{sparse} (line 14). Upon finishing of this iteration, we copy T_{dense} and T_{sparse} back to GPU (line 15). After all chunks are done, update W_{dense} via copying W'_{dense} to W_{dense} (line 22). It is important to note that we will copy T_{sparse} from CPU to GPU (line 18) to update W_{sparse} (line19) because T_{sparse} is very small compared with T_{dense} and the time cost for transferring T_{sparse} from CPU to GPU is negligible. Now W_{sparse} and W_{dense} are updated, *MPT_Generate* kernel (line 23) will be run to get K_0, K_1 and Q for next iteration. ezLDA will repeat this process for next iteration.

Algorithm 1 ezLDA

1: $GPUMemSet(W_{dense} and W_{sparse}, 0)$
2: **for** chunkId =1 to num_chunks **do**
3: $DataTransfer(T_{sparse} and T_{dense}, CPU \rightarrow GPU)$
4: $W_{sparse} = W_{sparse}\_Update(T_{sparse})$
5: $W_{dense} = W_{dense}\_Generate(T_{dense})$
6: **end for**
7: **for** iter=1 to num_iterations **do**
8: $MPT\_Generate(W^{iter}_{dense}, W^{iter}_{sparse}, K^{iter}_1, K^{iter}_2)$
9: **for** chunkId =1 to num_chunks **do**
10: $DataTransfer(T^{iter}_{sparse} and T^{iter}_{dense}, CPU \rightarrow GPU)$
11: $C^{iter}_0 = D\_Update(T^{iter}_{sparse}, T^{iter}_{dense}, D^{iter})$
12: $M^{iter} = MPT\_Calculate(T^{iter+1}_{dense}, T^{iter+1}_{sparse}, K^{iter}_0, K^{iter}_1, C^{iter}_0, u^{iter})$
13: **if** $unskippedTokenFlag = true$ **then**
14: $T^{iter+1}_{dense} = Sampling(D)(T^{iter}_{dense}, W^{iter}_{dense}, D^{iter}, u^{iter}, M^{iter}, W'^{iter+1}_{dense})$
15: $T^{iter+1}_{sparse} = Sampling(S)(T^{iter}_{sparse}, W^{iter}_{sparse}, D^{iter}, u^{iter}, M^{iter})$
16: **end if**
17: $DataTransfer(T^{iter+1}_{sparse} and T^{iter+1}_{dense}, GPU \rightarrow CPU)$
18: **end for**
19: **for** chunkId =1 to num_chunks **do**
20: $DataTransfer(T^{iter+1}_{sparse}, CPU \rightarrow GPU)$
21: $W^{iter+1}_{sparse} = W_{sparse}\_Update(T^{iter}_{sparse})$
22: **end for**
23: $GPUMemSet(W^{iter+1}_{sparse}, 0)$
24: $Copy(W^{iter+1}_{dense}, W'^{iter+1}_{dense})$
25: $MPT\_Generate(W^{iter+1}_{dense}, W^{iter+1}_{sparse}, K^{iter+1}_0, K^{iter+1}_1)$
26: **end for**

Algorithm 2 shows the pseudo-code of our three-branch sampling. First, dense vector $\widehat{W}[v]$ needs to be generated by *SparseToDense* kernel if **W** is W_{sparse} (line 2). Set $\widehat{W}[v][K_1]$ to be 0 (line 3) to exclude the most popular topic K_1 (line 3). Q tree needs to be generated and then shared by tokens of same word (line 4). Second, for each un-skipped token, three-branch sampling will generate a new topic by following steps: 1) Build S' tree to get $S[i]$ (line 6). 2) Fetch M[i] from memory for each token and calculate corresponding thresholds, t1 and t2 (line 6-7). 3) Fetch u[i] and compare it with t1 and t2. If u[i] falls into the range of most popular topic K_1, new topic will be K_1 (line 9-10). Otherwise, sampling from S' or Q' tree (line 11-15) to get a new topic.

Algorithm 3 shows the pseudo-code of MPT Generate Kernel. For each word, we get most and second most popular

Algorithm 2 Sampling

1: **function** SamplingKernel($W, D, T, M, u, K_1, \alpha, \beta$)
2: $\widehat{W}[v] = SparseToDense(W[v])$
3: $\widehat{W}[v][K_1] = 0$
4: $Q' = BuildTreeQ(\widehat{W}[v], \alpha, Tree(p_q))$
5: **for** i = 0 to num_unskippedtokens **do**
6: $S'[i] = BuildTreeS(\widehat{W}[v], D[d], \beta, Tree(p_s[i]))$
7: $t1 = M[i]/(S'[i] + Q' + M[i])$
8: $t2 = (M[i] + Q')/(S[i] + Q' + M[i])$
9: **if** $(u[i] < t1)$ **then**
10: $newtopic = K_1$
11: **else if** $(t1 < u[i] < t2)$ **then**
12: $newtopic = Sampling(Tree(p_s[i]), (u[i] - t1)/(t2 - t1))$
13: **else**
14: $newtopic = Sampling(Tree(p_q), (u[i] - t1 - t2)/(1 - t1 - t2))$
15: **end if**
16: $T[i] = newtopic$
17: **end for**
18: **end function**

topic K_1 and K_2 in vector $\widehat{W}_v$. The cost for MPT Generate Kernel is trivial because we only need N_W iterations, where N_W is vocabulary size.

Algorithm 3 MPT Generate Kernel

1: **function** MPTGKernel($K_1, K_2, \widehat{W}_v$)
2: $K_1 = argMax(\widehat{W}_v)$
3: $K_2 = argSecMax(\widehat{W}_v)$
4: **end function**

Algorithm 4 shows the pseudo-code of **D** update kernel, as shown in Algorithm 4. First, D_v is got by **AtomicAdd** 1 in corresponding topic position for each token from same document. Then we convert D_v into sparse vector D^{sparse}_v and update **D**. We can reuse D_v to get C_1. We use one warp to process one document and one thread to process one token for this kernel.

Algorithm 4 Update Kernel(**D**)

1: **function** UpdateKernel($T, D^{sparse}_v, C_1, K_1, W$)
2: **for** i = 0 to num_tokens **do**
3: $AtomicAdd(D_v[T[i]], 1)$
4: **end for**
5: $Block\_sync$
6: **for** i = 0 to num_tokens **do**
7: $D^{sparse}_v = DenseToSparse(D_v)$
8: $C_1[i] = GetMaxParameter(D_v, K_1)$
9: **end for**
10: **end function**

Algorithm 5 shows the pseudo-code of MPT Calculate Kernel. M is calculated with corresponding formula. Here D_{rowsum} is the row sum of **D** and easy to be calculated from row offset. We use one warp to process one word and one thread to process one token for this kernel.

We apply hierarchical workload balancing strategy on all kernels. As illustrated above, we also design different workload distribution strategies for different kernels, some are warp-thread based, while some are block-warp based. For three-branch sampling, D_v is reused in **D** update kernel for

Algorithm 5 MPT Calculate Kernel

```
function MPTCKernel(C_1, K_1, K_2, D, Q', α, Ŵ_v)
    for i = 0 to num_tokens do
        m = (C_1[i] + α) * (Ŵ_v[K_1])
        S_est = (D_rowsum − C_1[i]) * (Ŵ_v[K_1])
        t = RandomGenerate(0, 1)
        if (t < m/(m + S_est + Q')) then
            newtopic = K_1
        else
            num_unskippedtokens + +
            u[i] = t
            M[i] = m
        end if
    end for
    returnM, u
end function
```

speedup. Memory of C_1 is reused by M to reduce GPU memory usage.

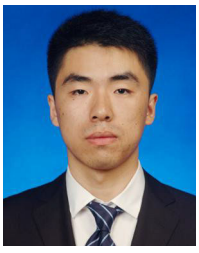
ACKNOWLEDGMENT

(Shilong Wang and Hang Liu contributed equally to this work.)

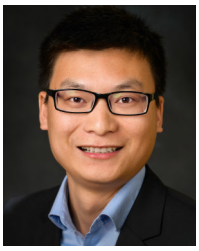
REFERENCES

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.
[2] J. L. Boyd-Graber, D. M. Blei, and X. Zhu, "A topic model for word sense disambiguation," in *Proc. EMNLP-CoNLL*, 2007, pp. 1–12.
[3] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and W. Chen, "Zwift: A programming framework for high performance text analytics on compressed data," in *Proc. Int. Conf. Supercomputing*, 2018, pp. 195–206.
[4] L. Cao and L. Fei-Fei, "Spatially coherent latent topic model for concurrent segmentation and classification of objects and scenes," in *Proc. IEEE 11th Int. Conf. Comput. Vis.*, 2007, pp. 1–8.
[5] W.-Y. Chen, J.-C. Chu, J. Luan, H. Bai, Y. Wang, and E. Y. Chang, "Collaborative filtering for Orkut communities: Discovery of user latent behavior," in *Proc. WWW*, 2009, pp. 681–690.
[6] R. Krestel, P. Fankhauser, and W. Nejdl, "Latent Dirichlet allocation for tag recommendation," in *Proc. 3rd ACM Conf. Recommender Syst.*, Oct. 2009, pp. 61–68.
[7] J. Chang and D. M. Blei, "Relational topic models for document networks," in *Proc. Artif. Intell. Statist.*, 2009, pp. 81–88.
[8] L. Yut, C. Zhang, Y. Shao, and B. Cui, "LDA*: A robust and large-scale topic modeling system," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1406–1417, Aug. 2017.
[9] M. Shi, Y. Tang, and J. Liu, "Functional and contextual attention-based LSTM for service recommendation in mashup creation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1077–1090, May 2019.
[10] Y. Wang and W. Xu, "Leveraging deep learning with LDA-based text analytics to detect automobile insurance fraud," *Decis. Support Syst.*, vol. 105, pp. 87–95, Jan. 2018.
[11] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and W. Chen, "Efficient document analytics on compressed data: Method, challenges, algorithms, insights," *Proc. VLDB Endowment*, vol. 11, no. 11, pp. 1522–1535, Jul. 2018.
[12] F. Qian, L. Sha, B. Chang, L.-C. Liu, and M. Zhang, "Syntax aware LSTM model for Chinese semantic role labeling," 2017, *arXiv:1704.00405*.
[13] J. Chen, J. He, Y. Shen, L. Xiao, X. He, J. Gao, X. Song, and L. Deng, "End-to-end learning of LDA by mirror-descent back propagation over a deep architecture," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1765–1773.
[14] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018, *arXiv:1802.05365*.
[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[16] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-BERT: Hessian based ultra low precision quantization of BERT," 2019, *arXiv:1909.05840*.
[17] Y. Ueno and R. Yokota, "Exhaustive study of hierarchical AllReduce patterns for large messages between GPUs," in *Proc. 19th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2019, pp. 430–439.
[18] A. Svyatkovskiy, J. Kates-Harbeck, and W. Tang, "Training distributed deep recurrent neural networks with mixed precision on GPU clusters," in *Proc. Mach. Learn. HPC Environments*. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 1–8.
[19] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," 2014, *arXiv:1412.5567*.
[20] T. Akiba, K. Fukuda, and S. Suzuki, "ChainerMN: Scalable distributed deep learning framework," 2017, *arXiv:1710.11351*.
[21] N. Dryden, N. Maruyama, T. Moon, T. Benson, A. Yoo, M. Snir, and B. Van Essen, "Aluminum: An asynchronous, GPU-aware communication library optimized for large-scale training of deep neural networks on HPC systems," in *Proc. IEEE/ACM Mach. Learn. HPC Environments (MLHPC)*, Nov. 2018, pp. 1–13.
[22] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the GPU," in *Proc. 20th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, Jan. 2015, pp. 1–12.
[23] L. Yao, D. Mimno, and A. McCallum, "Efficient methods for topic model inference on streaming document collections," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jun. 2009, pp. 937–946.
[24] L. Tierney, "Markov chains for exploring posterior distributions," *Ann. Statist.*, vol. 22, no. 4, pp. 1701–1728, Dec. 1994.
[25] A. Q. Li, A. Ahmed, S. Ravi, and A. J. Smola, "Reducing the sampling complexity of topic models," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 891–900.
[26] J. Yuan, F. Gao, Q. Ho, W. Dai, J. Wei, X. Zheng, E. P. Xing, T.-Y. Liu, and W.-Y. Ma, "LightLDA: Big topic models on modest computer clusters," in *Proc. 24th Int. Conf. World Wide Web*, May 2015, pp. 1351–1361.
[27] J. Chen, K. Li, J. Zhu, and W. Chen, "WarpLDA: A cache efficient O(1) algorithm for latent Dirichlet allocation," *Proc. VLDB Endowment*, vol. 9, no. 10, pp. 744–755, Jun. 2016.
[28] K. Li, J. Chen, W. Chen, and J. Zhu, "SaberLDA: Sparsity-aware learning of topic models on GPUs," *ACM SIGOPS Operating Syst. Rev.*, vol. 51, no. 2, pp. 497–509, 2017.
[29] X. Xie, Y. Liang, X. Li, and W. Tan, "CuLDA_CGS: Solving large-scale LDA problems on GPUs," 2018, *arXiv:1803.04631*.
[30] H. Zhao, B. Jiang, J. F. Canny, and B. Jaros, "SAME but different: Fast and high quality Gibbs parameter estimation," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2015, pp. 1495–1502.
[31] H.-F. Yu, C.-J. Hsieh, H. Yun, S. V. N. Vishwanathan, and I. S. Dhillon, "A scalable asynchronous distributed algorithm for topic modeling," in *Proc. 24th Int. Conf. World Wide Web*, May 2015, pp. 1340–1350.
[32] Y. Yang, J. Chen, and J. Zhu, "Distributing the stochastic gradient sampler for large-scale LDA," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 1975–1984.
[33] S. Ashkiani, M. Farach-Colton, and J. D. Owens, "A dynamic hash table for the GPU," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 419–429.
[34] F. Yan, N. Xu, and Y. Qi, "Parallel inference for latent Dirichlet allocation on graphics processing units," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 2134–2142.
[35] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling, "Fast collapsed Gibbs sampling for latent Dirichlet allocation," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2008, pp. 569–577.
[36] Y. W. Teh, D. Newman, and M. Welling, "A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 1353–1360.
[37] G. C. G. Wei and M. A. Tanner, "A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms," *J. Amer. Stat. Assoc.*, vol. 85, no. 411, pp. 699–704, Sep. 1990.
[38] Nvidia. *V100 GPU*. Accessed: Aug. 5, 2019. [Online]. Available: https://www.nvidia.com/en-us/data-center/v100/

[39] A. Gaihre, Z. Wu, F. Yao, and H. Liu, "XBFS: Exploring runtime optimizations for breadth-first search on GPUs," in *Proc. 28th Int. Symp. High-Performance Parallel Distrib. Comput.*, 2019, pp. 121–131.
[40] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.
[41] A. Cano, "A survey on graphic processing unit computing for large-scale data mining," *Wiley Interdiscipl. Reviews, Data Mining Knowl. Discovery*, vol. 8, no. 1, p. e1232, Jan. 2018.
[42] M. Zaheer, M. Wick, J.-B. Tristan, A. Smola, and G. L. Steele Jr., "Exponential stochastic cellular automata for massively parallel inference," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2016, pp. 966–975.
[43] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens, "Scan primitives for GPU computing," in *Proc. Graph. Hardw.*, 2007, pp. 97–106.
[44] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The art of Scientific Computing*. Cambridge, U.K.: Cambridge Univ. Press, 2007.
[45] Y. Chen, A. B. Hayes, C. Zhang, T. Salmon, and E. Z. Zhang, "Locality-aware software throttling for sparse matrix operation on GPUs," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.* Berkeley, CA, USA: USENIX Association, 2018, pp. 413–425.
[46] W. Han, D. Mawhirter, B. Wu, and M. Buland, "Graphie: Large-scale asynchronous graph traversals on just a GPU," in *Proc. 26th Int. Conf. Parallel Architectures Compilation Techn. (PACT)*, Sep. 2017, pp. 233–245.
[47] C. Holmes, D. Mawhirter, Y. He, F. Yan, and B. Wu, "GRNN: Low-latency and scalable RNN inference on GPUs," in *Proc. 14th EuroSys Conf.*, Mar. 2019, p. 41.
[48] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2091–2100, Aug. 2013.
[49] U. Meyer and P. Sanders, "δ-stepping: A parallel single source shortest path algorithm," in *Proc. Eur. Symp. Algorithms*. Cham, Switzerland: Springer, 1998, pp. 393–404.
[50] S. Song, X. Liu, Q. Wu, A. Gerstlauer, T. Li, and L. K. John, "Start late or finish early: A distributed graph processing system with redundancy reduction," *Proc. VLDB Endowment*, vol. 12, no. 2, pp. 154–168, 2018.
[51] R. Crovella. *Cuda Atomics Change Flag*. Accessed: Jan. 2018. [Online]. Available: https://stackoverflow.com/questions/18963293/cuda-atomics-change-flag
[52] D. Newman, "Bag of Words," UCI Mach. Learn. Repository, 2008, doi: 10.24432/C5ZG6P.
[53] L. Han, A. L. Kashyap, and J. Weese, "UMBC-EBIQUITY-CORE: Semantic textual similarity systems," in *Proc. 2nd Joint Conf. Lexical Comput. Semantics*. Stroudsburg, PA, USA: Association for Computational Linguistics, Jun. 2013, pp. 1–14.
[54] M. Wahib and N. Maruyama, "Scalable kernel fusion for memory-bound GPU applications," in *Proc. SC Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2014, pp. 191–202.
[55] Nvidia Inc. *Nvidia NVPROF*. Accessed: Jun. 3, 2020. [Online]. Available: https://docs.nvidia.com/cuda/profiler-users-guide/index.html
[56] Nvidia Inc. *Nvidia Titan 1080*. Accessed: Jun. 3, 2020. [Online]. Available: https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf
[57] D. Blei and J. Lafferty, "Correlated topic models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 18, 2006, p. 147.
[58] Y. Teh, M. Jordan, M. Beal, and D. Blei, "Sharing clusters among related groups: Hierarchical Dirichlet processes," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 17, 2004, pp. 1–8.



SHILONG WANG received the B.E. and M.S. degrees in electromagnetic field and microwave technology from the Harbin Institute of Technology, Harbin, China, in 2011 and 2014, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA, USA. His research interests include high-performance computing, machine learning, and deep learning.



HANG LIU (Senior Member, IEEE) received the B.E. degree from the Huazhong University of Science and Technology, in 2011, and the Ph.D. degree from the George Washington University, in 2017. He is currently an Assistant Professor of electrical and computer engineering with the Stevens Institute of Technology, where he leads the HPDA Laboratory. Prior to joining the Stevens Institute of Technology, he was an Assistant Professor with the University of Massachusetts Lowell. His research exploits emerging hardware—such as graphics processing unit (GPU), field-programmable gate array (FPGA), high-end CPU, and solid-state drive (SSD)—to build high-performance systems for graph computing, machine learning, computational omics, numerical simulation, and cloud computing.

ANIL GAIHRE received the B.E. degree in electronics and communication from the Institute of Engineering, Pulchowk Campus, Nepal, in 2014. He is currently pursuing the Ph.D. degree in computer engineering with the Department of Electronics and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ, USA. From 2014 to 2017, he was a Software Engineer and the Project Leader with E&T Nepal Pvt. Ltd. His research interests include high-performance computing, sparse linear algebra, and blockchain.

HENGYONG YU (Senior Member, IEEE) received the bachelor's degree in information science and technology and computational mathematics and the Ph.D. degree in information and communication engineering from Xi'an Jiaotong University, in 1998 and 2003, respectively. He is currently a Full Professor and the Director of the Imaging and Informatics Laboratory, Department of Electrical and Computer Engineering, University of Massachusetts Lowell. He has authored/coauthored more than 200 peer-reviewed journal articles and more than 140 conference proceedings/abstracts. According to Google Scholar Citation, the H-index is 45 and i10-index is 144. His research interests include medical imaging with an emphasis on computed tomography and medical image processing and analysis. In January 2012, he received the NSF CAREER Award for the development of CS-based interior tomography. He serves as an Editorial Board Member for IEEE Access, *Signal Processing*, and *CT Theory and Applications*. He is the founding Editor-in-Chief of *JSM Biomedical Imaging Data Papers*.