IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Digital Object Identifier

GHLDA: GPU based Hierarchy Latent Dirichlet Allocation

Shilong Wang[1], Hengyong Yu[1] (FELLOW, IEEE)
[1]Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA 01854 USA
Corresponding author: Hengyong Yu (e-mail: Hengyong-yu@ieee.org).

ABSTRACT Hierarchical Latent Dirichlet Allocation (HLDA) is an enhanced version of the conventional Latent Dirichlet Allocation (LDA) text model, designed to uncover underlying semantic patterns in text corpora. The HLDA introduces a hierarchical structure that captures topic hierarchies across various levels of detail, resulting in more precise outcomes. Graphic Processing Units (GPUs) offer notable benefits over Central Processing Units (CPUs) in terms of extensive parallel processing and high memory throughput. However, the complexity of the tree data structure and the dynamically growing memory requirements present obstacles to the scalability of HLDA when implemented on GPUs. This paper introduces a GPU-based HLDA (GHLDA), first known-to-all GPU-based implementation of HLDA. Our innovations are twofold: we implement the HLDA on GPU at a systematic level; and we handle the complex tree structures by matrix on GPU.

INDEX TERMS Hierarchical Latent Dirichlet Allocation, GPU, Bayes Methods, Parallel Algorithms, Parallel Programming, Machine Learning, Unsupervised learning

I. INTRODUCTION

LATENT Dirichlet Allocation (LDA) [1], a topic model based on Bayesian theory and the Dirichlet distribution, is widely applied in various fields such as text classification [2], [3], recommendation systems [4], [5], online advertisement, deep learning [6], [7], information retrieval [8], and so on. With an increasing demand of handling big data, it becomes an important research topic to train a large-scale LDA. Different parallel algorithms have been proposed for scalable training, including Metropolis-Hasting [9], Expectation Maximization [10], and sparsity-aware [11] methods. As a powerful tool for parallel computing, graphic processing unit (GPU) has become a hot research direction for implementing large-scale training of LDA [12], [13], primarily due to its abundant cores and high memory throughput compared to central processing units (CPUs).

However, the traditional LDA models assume that all topics are independent and parallel, failing to capture the intricate relationships among topics effectively. To address this limitation, some more accurate LDA models are proposed, such as correlated topic model (CTM) [14] and Hierarchical LDA (HLDA) [15]. CTM assumes that topics are correlated and a covariance is introduced to measure the closeness of different topics. For example, topic "software" is more relevant to topic "hardware" than topic "house". Besides, intuitively speaking, topics have some hierarchical relationships, and some topics are subtopics of other topics. For example, topic "travel location" is a sub-topic of topic "travel". Thus, the HLDA refines the traditional LDA model by introducing a tree structure to the LDA model to reflect hierarchical relationships among topics. Such a hierarchical model can mine real relationship of topics hidden in the corpus better than the traditional models, and thus being widely investigated.

Fig. 1 shows an example of HLDA model. Unlike the traditional LDA model, different topics have hierarchy relationships. Take the very left path as an example, the very top topic contains some normal words "data", "input", "learning", "network", and "function", indicating this topic is related to machine learning. It's subtopic contains "Bayesian", "Gaussian", "chain", "model", and "mixture", which reflects that this subtopic is related to the traditional machine learning methods: Gaussian and Bayesian methods. Then, it comes to the final level, which contains "parameter", "Gaussian", "distribution", "probability", and "prior". Obviously, these words are related to implementations of Gaussian and Bayesian methods. Intuitively speaking, these topics have hierarchical relationships.

Despite that the HLDA can extract more subtle latent information from text corpus than the traditional LDA model, several tough issues should be tackled for the scalable training of HLDA. First, the HLDA needs to store and handle

complex tree structures. Second, unlike the traditional LDA model that has a fixed number of topics, the HLDA has a dynamically increased topic size. That means the sizes of matrices will change during the training process, and it is not friendly to handle memory. Third, by a Collapsed Gibbs Sampling, the HLDA alternatively sample topic levels and tree paths. Hence, both the tree and the matrices need to be updated in real-time, which greatly increases the difficulty of training. Therefore, the previous HLDA approaches are often limited to small corpus or large-scale training with massive CPUs and machines. This will significantly increase the training cost. Unlike CPU, GPU is more suitable for simple-logic and repeated tasks, and it is frustrated on complex logic and data structures. As a result, GPU-based HLDA has not been developed due to the aforementioned limitations.

A. RELATED WORK

Because the traditional flatten LDA model has been well investigated in both theory and application fields, we will limit our discussion on the HLDA in this section. There are mainly two directions for the HLDA learning. Li and McCallum introduced a model called Pachinko Allocation [16], which utilized a directed acyclic graph to capture the hierarchy relationship among topics. Thus, a topic in high level will be a mixture of it's sub-topics. The most famous HLDA model is based on nested Chinese Restaurant Process (nCRP) [15], [17], [18]. A topic is assumed to have several sub-topics, and nCRP will be used to generate a tree structure.

As to the large-scale training of HLDA, Purjara and Skomoroch obtained the hierarchy by splitting the corpus and training topics from top-to-down recursively [19]. Wang *et. al.* trained the hierarchy by a scalable tensor orthogonal decomposition algorithm from top-to-down recursively [20]. Chen *et. al.* implemented the HLDA based on the nCRP model and applied the Partially Collapsed Gibbs Sampling to alternatively sample topic level and document path [21]. For GPU-based HLDA, as far as we know, there is no related publications. The complicated tree structure and dynamically increased memory consuming are great challenges for GPU-based HLDA.

B. CONTRIBUTIONS

This paper follows the nCRP model and systematically introduces the first know-to-all GPU-based HLDA (GHLDA) method. With notable design and optimization, the GHLDA can train HLDA on the Pubmed dataset within 30 minutes while supporting 800 topics on merely one A100 GPU [22]. To achieve this achievement, we have the following major contributions.

First, unlike the CPU-based HLDA method, which holds the complicated tree structure in CPU memory, the GHLDA represents the tree by two sets, node ID set R and the corresponding count set Q. These two sets will be reconstructed from a set of document path D, instead of real-time update. This will avoid complex pruning and insertion, and details are discussed in Section III-A.

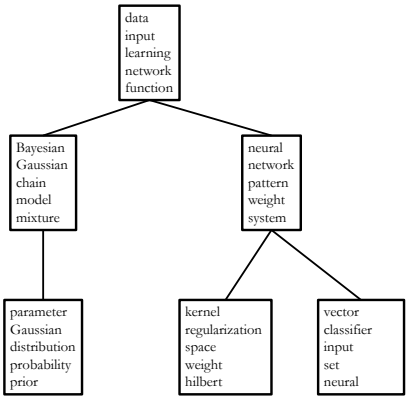


FIGURE 1. Example of training results on corpus NIPS by the GHLDA.

Second, we takes full efforts to make the GHLDA sampling suitable for computing on GPU. The sampling process are mainly divided into four parts: token-level sampling, token list reorder, document-path sampling, and tree update. For the token-level sampling, each token is handled by a thread and assigned to a new level after the sampling. Then, the token order within the same document will be reordered to obtain count vectors $\mathbf{m}$ and $\mathbf{n}$, which are needed for the document-path sampling. The details are discussed in Section III-B. For the document-path sampling, we conduct sampling of a document in block level and calculate probability of a possible path of this document in a thread level. Each document will get a new path after this step, and the details are discussed in Section III-C. After the document-path sampling, tree-related sets R and Q will be updated from the new document-path set P, and a new iteration will start. The details are discussed in Section III-A.

Third, considering that the number of topics will change after every iteration, we have to frequently use cudaMalloc and cudaFree during the sampling, which are time-consuming APIs. We use memory pool to avoid such expensive operations. The GPU memory is not released instantly after the sampling and will be used by other pointers.

The novelty of this paper is that we make the complicated HLDA feasible to scalable and efficient training on GPU by design in system level. Particularly, to the best of our knowledge, the GHLDA is the first GPU-based HLDA training method.

C. ORGANIZATION

The rest of the paper is organized as follows. Section II introduces the background. Section III presents detail implementation of the GHLDA. Section IV presents the experiment results. We conclude our work in Section V.

II. BACKGROUND AND CHALLENGES

A. NAMING PRINCIPLES OF FORMULAS

Before introducing the HLDA algorithms, we need to introduce the naming principles of formulas in this paper.

1) A bold uppercase letter denotes a matrix. For example, $\mathbf{A}$ is a matrix. $\mathbf{A}_d$ is the d^{th} row vector of this matrix.

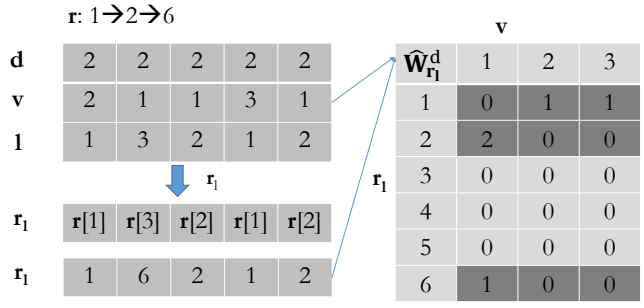


FIGURE 2. Explanation of $\widehat{\mathbf{W}}^{d}_{\boldsymbol{r}_l}$.

$\mathbf{A}_{dv}$ or $\mathbf{A}_{d,v}$ is the element of d^{th} row and v^{th} column in $\mathbf{A}$.

2) A bold lowercase letter denotes a vector. For example, $\mathbf{a}$ represents a vector, and $\mathbf{a}_d$ is the d^{th} element of this vector.
3) A normal uppercase letter except N and B denote a set of vectors. For example, P is a set. $\mathbf{P}_d$ is d^{th} vector in this set. P_{dv} or $P_{d,v}$ is the v^{th} element of d^{th} vector in P.
4) Normal uppercase letter N with a subscript denotes a count. For example, N_l represents number of levels.
5) Superscript $\neg$ means excluding and superscript without $\neg$ means including. For example, $\mathbf{W}^{\neg dl}$ means a matrix after excluding tokens whose document ID is d and level ID is l from $\mathbf{W}$.
6) Hat appears with a superscript d and subscript $\boldsymbol{r}_l$. For example, $\widehat{\mathbf{W}}^{d}_{\boldsymbol{r}_l}$ means a topic-word matrix. As is shown in Fig.2, it consists of tokens from document d and corresponding topics are determined by path $\mathbf{r}$.

For other special cases, we have explanations in corresponding chapters.

B. HLDA ALGORITHM AND THEORY

Before introducing the HLDA, we first explain some basic concepts in LDA. A corpus is the training dataset of LDA, which is actually a collection of documents. Each occurrence of a single word in a document is called a token. Number of unique words in the corpus is called vocabulary size. Usually before a LDA is trained, a corpus needs to be translated into a token list, *i.e.*, a corpus consists of a list of tokens, and each token is represented by a pair of <document_id, word_id>. The nCRP-based HLDA actually combines the nCRP model with the traditional LDA to extract more accurate information from a corpus. We will call the nCRP-based HLDA as HLDA for short in the following part of this paper.

Algorithm. The traditional LDA model assumes that a corpus is generated from two distributions: a document generated from document-topic distribution and a topic generated from topic-word distribution, both of which satisfy the Dirichlet distribution. Each document is modeled by a unique mixture of N topics. The total number of topics N is fixed and defined based on demand before the training.

The HLDA refines the traditional LDA by introducing a tree structure. Instead of modeling by a fixed number N for topics, all the potential topics form a tree based on the nCRP model, and each node of this tree represents a topic. Each document samples a path from the tree with a distribution. The document-level and word-level both satisfy Dirichlet distribution. Hence, the generation of a corpus includes the following steps:

1) Draw a path from the nCRP model for a document;
2) Draw a topic mixing potion from document-level distribution;
3) Draw a word mixing potion from word-level distribution.

It's not hard to find that the traditional LDA model with N topics is actually a special case of the HLDA, *i.e.*, all documents are modeled by the same path, and the level of this path will be N. Since a tree can grow or shrink during the training, the number of topics will change during the course of the training process.

The HLDA process satisfies the following jointed distribution:

$$p(L,P,U)=\prod_{d=1}^{N_d}[p(P_d|R^{\neg d})\frac{B(\mathbf{D}_d+\boldsymbol{\alpha})}{B(\boldsymbol{\alpha})}]\prod_{k=1}^{\infty}\frac{B(\mathbf{W}_k+\boldsymbol{\beta}_{\mathbf{l}_k})}{B(\boldsymbol{\beta}_{\mathbf{l}_k}\mathbf{1})}, \tag{1}$$

where L is a set of token levels, P is a set of document paths, U is a given corpus, and R is a set of all available paths for documents. $\mathbf{D}$ is the document-level matrix, $\mathbf{W}$ is the topic-word matrix, N_d is the number of documents in corpus U, $\mathbf{1}$ is an all-one vector, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are hyper-parameters related to Dirichlet distribution, $B(\cdot)$ is beta function, and $\Gamma(\cdot)$ is gamma function. $B(\boldsymbol{\alpha})=\prod_k\Gamma(\alpha_k)/\Gamma(\sum_k\alpha_k)$. Regarding Eq.(1), we are more interested in getting the posterior distribution of L and P for a given corpus U. However, this equation does not have a close-form solution.

To solve Eq.(1), the Collapsed Gibbs Sampling (CGS) can be adopted. By applying the CGS, the HLDA can alternatively sample from the following two distributions [21]:

1) Sample token level $\mathbf{l}_k$: draw a level for each token from

$$p(L_{dk}=\mathbf{l}_k)\propto(\mathbf{D}^{\neg dk}_{d\mathbf{l}_k}+\boldsymbol{\alpha}_{\mathbf{l}_k})\times\frac{\mathbf{W}^{\neg dk}_{P_{d\mathbf{l}_k},v}+\boldsymbol{\beta}_{\mathbf{l}_k}}{\mathbf{s}^{\neg dk}_{P_{d\mathbf{l}_k}}+N_v\boldsymbol{\beta}_{\mathbf{l}_k}}, \tag{2}$$

2) Sample document path $\mathbf{r}$:

$$p(P_d=\mathbf{r})\propto\underbrace{p(\mathbf{r}|R^{\neg d})}_{p_a(\boldsymbol{r})}\times\underbrace{\prod_{l=1}^{N_l}\frac{B(\mathbf{W}^{\neg d}_{\mathbf{r}_l}+\widehat{\mathbf{W}}^{d}_{\mathbf{r}_l}+\boldsymbol{\beta}_l)}{B(\mathbf{W}^{\neg d}_{\mathbf{r}_l}+\boldsymbol{\beta}_l)}}_{p_b(\boldsymbol{r})}, \tag{3}$$

where N_l is the depth of the tree, and N_v is the vocabulary size.

In Eq.(3), $p_a(\mathbf{r})$ is the probability generated by nCRP model and related to the tree structure. It has following format:

$$p_a(\boldsymbol{r})=\prod_{l=2}^{N_l}\frac{x}{y_{\boldsymbol{r}_{l-1}}+\boldsymbol{\gamma}_l},\ x=\begin{cases}y_{\boldsymbol{r}_l} & \text{if } \boldsymbol{r}_l \text{ is an existing node}\\ \boldsymbol{\gamma}_l & \text{if } \boldsymbol{r}_l \text{ is a new node}\end{cases}, \tag{4}$$

where y_i is the visit count of node i (Note: path of document d should be excluded from the tree first), and $\boldsymbol{\gamma}$ is a hyper-parameter to control the probability to generate a new node. For $p_b(\mathbf{r})$, we can further write it into the following log format:

$$log(p_b(\mathbf{r})) = \sum_{k=1}^{N_k} log(\mathbf{W}^{\neg dk}_{\mathbf{r}_{\mathbf{l}_k},\mathbf{v}_k} + \mathbf{m}_k + \boldsymbol{\beta}_{\mathbf{l}_k}) + h_{\mathbf{r}}, \quad (5)$$

and

$$h_{\mathbf{r}} = \sum_{l=1}^{N_l} [log\Gamma(\mathbf{s}^{\neg d}_{\mathbf{r}_l} + N_v\boldsymbol{\beta}_l) - log\Gamma(\mathbf{s}^{\neg d}_{\mathbf{r}_l} + \hat{\mathbf{s}}^{d}_{\mathbf{r}_l} + N_v\boldsymbol{\beta}_l)], \quad (6)$$

where N_k is number of tokens in document d. We can call tokens of identical document ID, word ID and level ID as "repeated" tokens. $\mathbf{m}$ reflects the ranks of these "repeated" tokens. $\boldsymbol{s}$ is total number of tokens for each topic.

Evaluation Metric. To evaluate the performance of convergence, we use a log-likelihood per token (LLPT), which is a negative logarithm of perplexity. The formula is shown below:

$$LLPT = -\frac{1}{N_t}\sum_{d=1}^{N_d}\sum_{k=1}^{N_k} log(\frac{\mathbf{D}_{d\mathbf{l}_k} + \boldsymbol{\alpha}_{\mathbf{l}_k}}{N_k + \sum_{l=1}^{N_l}\boldsymbol{\alpha}_l} \times \frac{\mathbf{W}_{P_{dl_k},v_k} + \boldsymbol{\beta}_{\mathbf{l}_k}}{\mathbf{s}_{P_{dl_k}} + N_v\boldsymbol{\beta}_{\mathbf{l}_k}}), \quad (7)$$

where N_t is total number of tokens in the corpus. After training, the tokens will converge to some topics which correspond to large probability in Eq.(2). Obviously, the LLPT can reflect the trend of the convergence, and larger LLPT represents better convergence.

C. CHALLENGES

From Section II-B, we can see that the HLDA is much more complicated than the traditional LDA, and we have to deal with following challenges.

Challenge #1. The algorithm of HLDA is complicated. Unlike the tradition LDA, which only needs to sample a topic for a token, the HLDA needs to alternatively sample a path from a tree for a document and a level for a token. The sampling becomes very complicated since more steps and parameters are involved.

Challenge #2. The HLDA needs to handle complicated tree structure. From Eq.(4), one can see that the tree structure is complicated, and it also needs to be updated during course of the training process. There are great challenges for implementation on a GPU.

Challenge #3. The HLDA needs to handle dynamically increased memory requirement. Since the number of topics is not fixed, memory requirements for R, Q, $\mathbf{D}$, $\mathbf{W}$ and $\mathbf{s}$ are all dynamic, which increases the difficulty of memory management on GPU.

III. GHLDA SAMPLING

In this part, we will introduce the detailed implementation of the GHLDA.

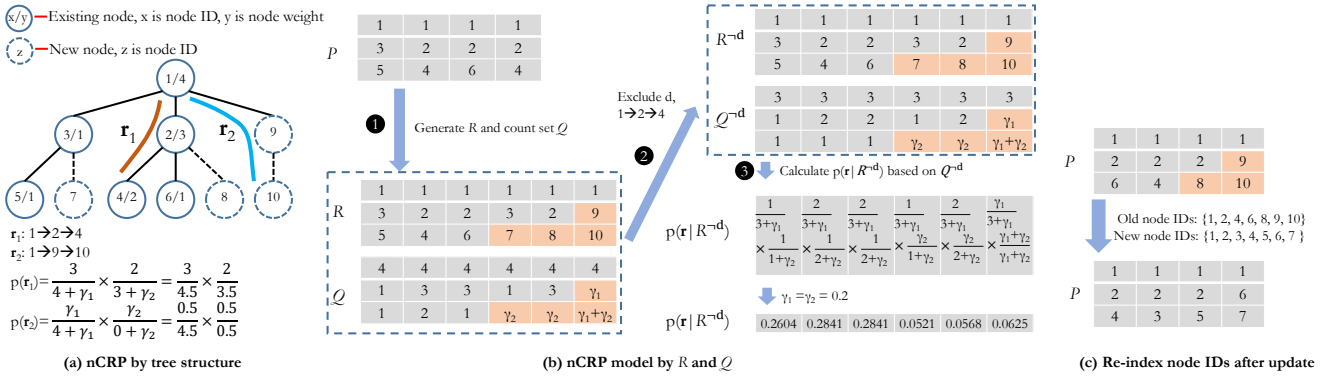
A. TREE IMPLEMENTATION

For each document, based on Eq.(3), we need to traverse the tree to get all available paths. Besides, the calculation of p_a in Eq.(4) will diverge for an existing node and a new node. Fig.3(a) presents an implementation of the tree-based nCRP model. For paths $\mathbf{r}_1$ and $\mathbf{r}_2$, we need to traverse the paths to calculate the probabilities by Eq.(4). The calculation of the probabilities for $\mathbf{r}_1$ and $\mathbf{r}_2$ will differ because $\mathbf{r}_2$ has new nodes 9 and 10, while $\mathbf{r}_1$ contains only the existing nodes. Maintaining a tree structure and traversing this tree are simple for a CPU system but not the case for a GPU system. Besides, the divergence in calculating probabilities betrays the pattern of the Single Instruction Multiple Thread (SIMD) on a GPU.

Fig.3(b) presents the detailed implementation of our proposed nCRP method on a GPU. P is a multi-set to store paths of all documents in the corpus. We use a multi-set R to represent all possible paths in the tree and a multi-set Q to represent the corresponding weights. First, we generate two multi-set R and Q from P (❶). The paths in R, containing only the existing nodes, can be obtained by extracting all unique paths from P. From Fig.3(a), it is easy to find that each none-leaf existing node corresponds to a unique new path. For example, node 1 corresponds to $1 \rightarrow 9 \rightarrow 10$, and node 2 corresponds to $1 \rightarrow 2 \rightarrow 8$. Hence, we can obtain the remaining new paths that contain new nodes by traversing the none-leaf nodes in P. The total paths in R will be the number of existing nodes. For Q, we can obtain it by traversing P and replacing the node IDs by the node weights. For a position that corresponds to a new node, we will assign a value related to $\boldsymbol{\gamma}$. The rule for the value assignment will be as follow. For a new path in R, tracing from the root until it encounters a new node in level l, assign $\sum_{n=l}^{i}\boldsymbol{\gamma}_{n-1}$ to the new nodes in level i. By applying this strategy, $x = m_{\mathbf{r}_l}$ in Eq.(4) will work for all available paths, and there will be no branch. Then, the calculation of probability will follow the SIMD pattern, which is more suitable for GPU. Now we have R and Q, and we need to exclude path of the document d from R (❷). In the final step, we can calculate the probability based on the first branch in Eq.(4) (❸).

Regarding the update, since some nodes may be removed after each iteration, we need to re-index the node IDs to avoid the explosion of node IDs. As is shown in Fig.3(c), node IDs $\{1, 2, 4, 6, 8, 9, 10\}$ are mapped to $\{1, 2, 3, 4, 5, 6, 7\}$. Then we can follow step ❶ in Fig.3(b) to generate R and Q by P after every iteration, which avoids the heavy cost of inserting and pruning operations.

For the sampling of path in Section III-C, a block and a thread will be used to process a document and a path, respectively. The number of nodes, also the number of unique paths, is usually in $10^2 - 10^3$ level. Then, R and Q can be fetched into the shared memory and reused by all the documents for better performance.



(a) nCRP by tree structure (b) nCRP model by R and Q (c) Re-index node IDs after update

FIGURE 3. Implementation of a tree structure.

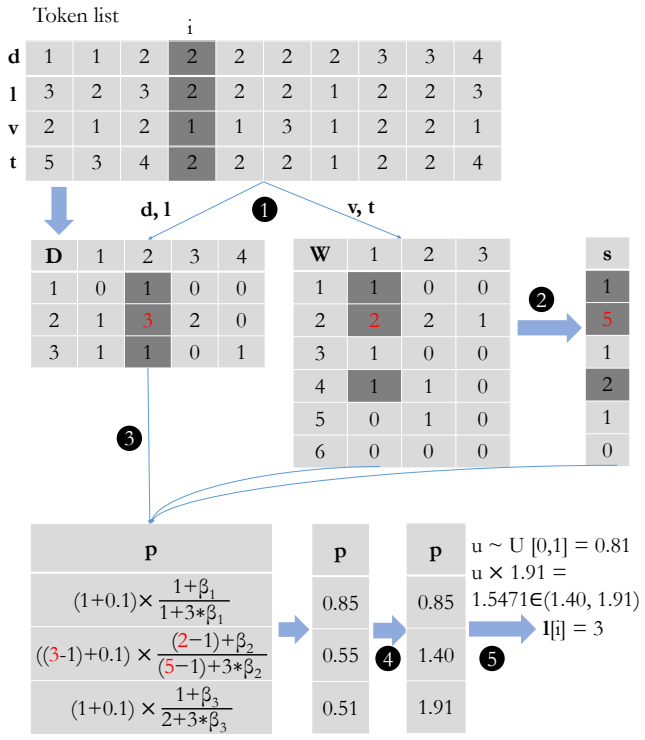


FIGURE 4. Sampling in the token level.

B. SAMPLING IN THE TOKEN LEVEL

Fig.4 presents the detailed implementation of token level sampling. The token list comprises four arrays, **d**, **l**, **v**, **t**, which means document ID, level ID, word ID and topic ID, respectively. The sampling process can be summarized as follows:

1) Generating document-level matrix **D** from **d** and **l**, and topic-word matrix **W** from **v** and **t** (❶);
2) Obtaining the array of topic-sum **s** by summing up rows of matrix **W** (❷);
3) Applying Eq.(2) to get probability **p** for each token (❸);
4) Calculating the normalized prefix sum of **p** (❹);
5) Generating a number u from the uniform distribution [0, 1] and obtaining a token's new level by judging which interval of **p** that u falls into (❺).

Taking the fourth token from the token list, $\{2, 2, 1, 2\}$, as an example, the document belongs to a path $1 \rightarrow 2 \rightarrow 4$, *i.e.*, P_d in Eq.(2) is (1, 2, 4). First, we can obtain $\mathbf{D}_d = (1, 3, 1)$ by referring the token's document ID. Since we know the token's word ID is 1, we can obtain $\mathbf{W}_{P_d,v} = (1, 2, 1)$ by extracting the elements whose labels match P_d from the first column of **W**. Similarly, we can obtain $\mathbf{s}_{P_d} = (1, 5, 2)$. Then, we can exclude this token from $\mathbf{D}_d$, $\mathbf{W}_{P_d,v}$ and $\mathbf{s}_{P_d}$. Since the document ID and level ID are $(2, 2)$, $\mathbf{D}_d^{\neg dk}$ will be $(1, 3-1, 1) = (1, 2, 1)$. Similarly, we can obtain $\mathbf{W}_{P_d,v}^{\neg dk} = (1, 2-1, 1) = (1, 1, 1)$, and $\mathbf{s}_{P_d}^{\neg dk} = (1, 5-1, 2) = (1, 4, 2)$. We can apply Eq.(2) to get the probability of each level $\mathbf{p}(L_{dk} = \mathbf{l}) = (0.85, 0.55, 0.51)$ (❸). We can further obtain the prefix sum and normalization $(0.85, 1.40, 1.91)$ (❹). In the final step, we generate a value $u = 0.81$ from a Uniform [0,1] distribution and find it falls into the interval (1.40, 1.91), which is the third interval of the prefix sum. Finally, we will update the token level as 3 (❺).

C. SAMPLING THE DOCUMENT PATH

After sampling in the token level in Section III-B, we will have the updated **l** and **t** in the token list. Then, we conduct the next step: sampling a new path for each document based on Eq.(3). Fig.5 presents the detailed implementation of path sampling. The sampling can be summarized as the following steps.

1) Updating topic-word matrix **W** from **v** and **t** (❶);
2) Obtaining the array of topic-sum **s** by summing up rows of the matrix **W** (❷);
3) Within each document, sorting the token list first by **v** and then by **l** (❸);
4) Within each document, doing exclusive scan by **v** to obtain **m** (❹);
5) Within each document, doing inclusive scan by **m** to obtain **n** (❺);
6) Within each document, calculating $\hat{\mathbf{n}}_{\mathbf{r}_l}^d$ based on path **r** and document d (❻);
7) Calculating p_1 in Eq.(5) based on **W**, **m** and $\hat{\mathbf{n}}_{\mathbf{r}_l}^d$ (❼);
8) Calculating p_2 in Eq.(5) based on $\hat{\mathbf{s}}_{\mathbf{r}_l}^d$ and $\hat{\mathbf{n}}_{\mathbf{r}_l}^d$ (❽);
9) Summing up p_1 and p_2 to obtain p_b (❾);
10) Following the steps ❷ and ❸ in Fig.3(b) to obtain

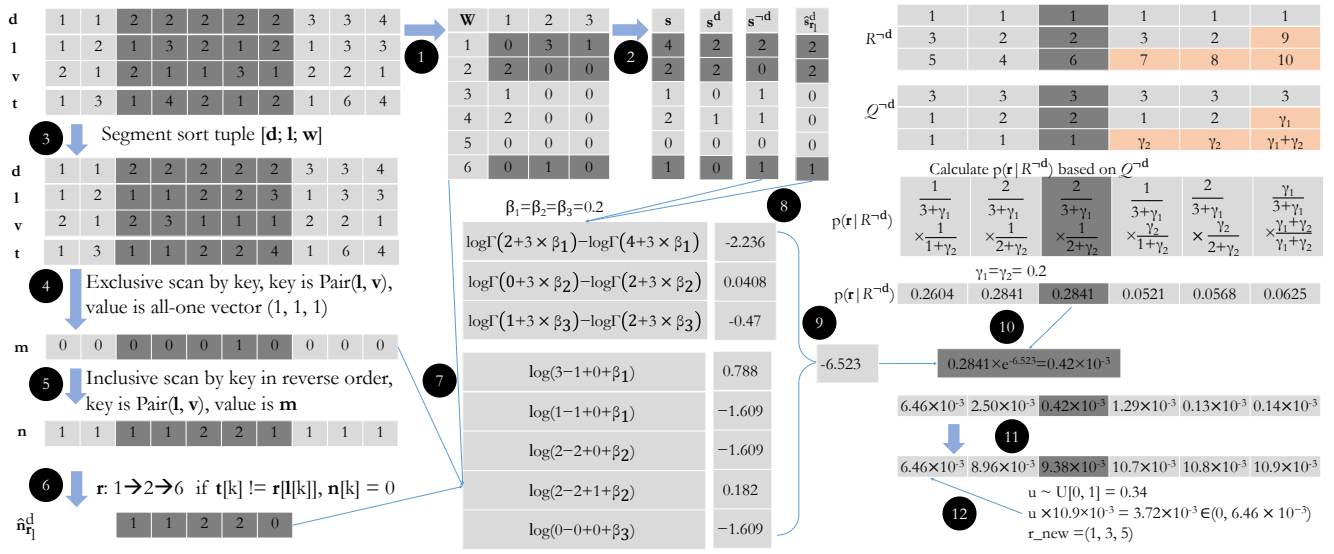


FIGURE 5. Sampling in the document path.

$p(\boldsymbol{r}|R^{\neg d})$;

11) Multiplying $p(\boldsymbol{r}|R^{\neg d})$ with p_b to obtain $p(P_d = \mathbf{r})$ (❿);
12) Calculating the normalized prefix sum of $p(P_d = \mathbf{r})$ (⓫);
13) Generating a number u from uniform distribution [0, 1] to obtaina document's new level by judging which interval of $\mathbf{p}_u$ falls into (⓬);
14) Updating P, **W** and token list.

Taking document 2 as an example, its current path is $1 \rightarrow 2 \rightarrow 4$. To calculate $\boldsymbol{m}_k$ in Eq.(5), we need to sort the tokens within this document first by **v** and then by **l** (❸). This will ensure not only "repeated" tokens are gathered together but also tokens in the same document are sorted by level ID. We can see that the fifth and seventh tokens (2, 2, 1, 2) are "repeated" tokens, and they are gathered together after the sort. Then, for the tokens in the same document, we can do exclusive scan to capture the ranks of "repeated" tokens, *i.e.*, $\boldsymbol{m}_k$ in Eq.(5) (❹). For the "repeated" tokens (2, 2, 1, 2), the corresponding $\boldsymbol{m}_k$ is assigned to 0 and 1, respectively. Then, for tokens in the same document, we can conduct inclusive scan on $\boldsymbol{m}$ in a reverse order to obtain $\boldsymbol{n}$. This reflects the frequency of the "repeated" tokens (❺). We can obtain $\boldsymbol{n}_k = 2$ for "repeated" tokens (2, 2, 1, 2), and $\boldsymbol{n}_k = 1$ for all the remaining "none-repeated" tokens. For a given path **r** ($1 \rightarrow 2 \rightarrow 6$), we can get $\hat{\mathbf{n}}^d_{\mathbf{r}_l}$ (❻).we can apply p_1 in Eq.(5) based on **W**, **m** and $\hat{\mathbf{n}}^d_{\mathbf{r}_l}$ (❼). For the sixth token (2, 2, 1, 2) in the token list, its $\mathbf{l}_k = 2$, $\mathbf{v}_k = 1$, $\mathbf{t}_k = 2$, $\mathbf{m}_k = 1$, and $\hat{\mathbf{n}}^{dk}_{\mathbf{r}_l} = 2$. We can get $\mathbf{r}_{\mathbf{l}_k} = 2$, and $\mathbf{W}_{\mathbf{r}_{\mathbf{l}_k},\mathbf{v}_k} = 2$. It's not hard to find that when $\mathbf{r}_{\mathbf{l}_k} = \mathbf{t}_k$, we have $\mathbf{W}^{\neg d}_{\mathbf{r}_{\mathbf{l}_k},\mathbf{v}_k} = \mathbf{W}_{\mathbf{r}_{\mathbf{l}_k},\mathbf{v}_k} - \mathbf{n}_k$, otherwise, $\mathbf{W}^{\neg d}_{\mathbf{r}_{\mathbf{l}_k},\mathbf{v}_k} = \mathbf{W}_{\mathbf{r}_{\mathbf{l}_k},\mathbf{v}_k}$. That means $\mathbf{W}^{\neg d}_{\mathbf{r}_{\mathbf{l}_k},\mathbf{v}_k} = \mathbf{W}_{\mathbf{r}_{\mathbf{l}_k},\mathbf{v}_k} - \hat{\mathbf{n}}^{dk}_{\mathbf{r}_l}$. For this token, $\mathbf{r}_{\mathbf{l}_k} = \mathbf{t}_k = 2$. Hence, $\mathbf{W}^{\neg d}_{\mathbf{r}_{\mathbf{l}_k},\mathbf{v}_k} = 2 - 2 = 0$. We can obtain p_2 in Eq.5 based on **s** (❽). Similar to step ❻, for each level, we need to exclude tokens of this document from $\mathbf{s}_{\mathbf{r}_l}$ to get $\mathbf{s}^{\neg d}_{\mathbf{r}_l}$. For a given level l, when $\mathbf{r}_l = \mathbf{v}_l$, we have $\mathbf{s}^{\neg d}_{\mathbf{r}_l} = \mathbf{s}_{\mathbf{r}_l} - \mathbf{x}_l$, where $\mathbf{x}_l$ is total number of tokens in this level. Otherwise, $\mathbf{s}^{\neg d}_{\mathbf{r}_l} = \mathbf{s}_{\mathbf{r}_l}$. For $l = 2$, we have 3 tokens in level 2 and we know $\mathbf{r}_l = \mathbf{t}_l = 2$. Therefore, we can get $\mathbf{s}^{\neg d}_{\mathbf{r}_l} = 2-2 = 0$. By summing up p_1 and p_2, we can obtain $p_b = -6.523$ for the path **r** ($1 \rightarrow 2 \rightarrow 6$) (❾). We can obtain $p(r|R^{\neg d}) = 0.2841$ by applying steps ❷ and ❸ in Fig.3(b). We can further obtain $p(P_d = \mathbf{r}) = 0.42 \times 10^{-3}$ by multiplying $p(r|R^{\neg d})$ and p_b (❿). We can get $p(P_d = \mathbf{r}) = \{6.46 \times 10^{-3}, 2.50 \times 10^{-3}, 0.42{\times}10^{-3}, 1.29{\times}10^{-3}, 0.13{\times}10^{-3}, 0.14{\times}10^{-3}\}$ for all potential paths in R by applying step ❻, ❼, ❽, ❾ and ❿ for all potential paths from R. Then, just like what we have done in Section III-B, we can calculate the normalized prefix sum $p(P_d = \mathbf{r})$ (⓫) and generate a value $u = 0.34$ from uniform distribution [0, 1] and find it falls into the first interval of the prefix sum. The path $1 \rightarrow 3 \rightarrow 5$, which corresponds to the first element in $p(P_d = \mathbf{r})$ will be assigned as the document's new path (⓬). After the path sampling for all the documents are completed, P is updated. Then, we can update R and **t** in token list, and **W** based on P.

The GHLDA implementation consists of approximately 2,000 lines of C++/CUDA codes. The codes are compiled by using CUDA 12.0 toolkits with -o3 optimization. The performance evaluation of the GHLDA is conducted on an NVIDIA A100 GPU, notable for its 48GB global memory capacity and a remarkable memory throughput of 2TB/s.

IV. EXPERIMENTS AND RESULTS

Since the GHLDA is the first GPU-based HLDA method, we can only compare our results with the state-of-the-art CPU-based HLDA : PCGS-HLDA [21]. We evaluate the GHLDA with two popular datasets that are also used by PCGS-HLDA:

- NIPS [23]: 1,500 documents, 12,419 unique words, and 1.9M tokens.
- NYTimes [23]: 299,752 documents, 101,636 unique words, and about 100M tokens.

A. GHLDA *VS. THE STATE-OF-THE-ART*

For a fair comparison, we choose identical hyper-parameters for the PCGS-HLDA, *i.e.*, $\boldsymbol{\beta} = (\beta_0, 0.5\beta_0, 0.25\beta_0, 0.25\beta_0)$, where β_0 is chosen from $\{e^{-4.0}, e^{-3.5}, ..., e^{2.0}\}$. γ is chosen from $\{e^{-6.0}, e^{-5.5}, ..., e^{0.0}\}$, and $\boldsymbol{\alpha} = 0.2 \times \mathbf{1}$. Since the codes of PCGS-HLDA are not open source and the detailed run time is not reflected in the PCGS-HLDA paper, we can only compare the quality of inference for GHLDA vs. PCGS-HLDA, *i.e*, the perplexity. Here, $perplexity = e^{-LLPT}$ and a smaller perplexity usually means better convergence. As shown in Fig.6, the GHLDA can achieve comparable quality of inference as the PCGS-HLDA.

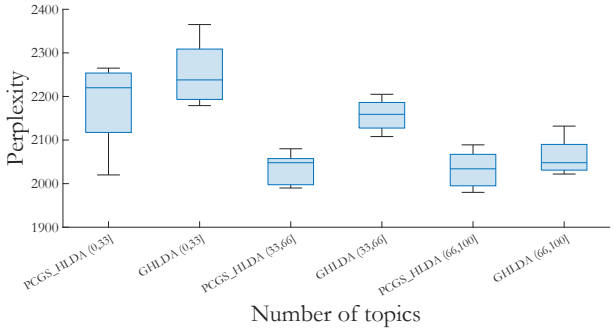


FIGURE 6. The convergence of the GHLDA vs. PCGS-HLDA on the NIPS dataset.

B. GHLDA *PERFORMANCE STUDY*

To study the performance of the GHLDA on different situations, we choose two cases, which correspond to different hyper-parameters.

- Case 1: $\gamma = e^{-10}$, $\boldsymbol{\alpha} = 0.1 \times \mathbf{1}$, $\boldsymbol{\beta} = (e^1, e^1, e^1, 0.25e^1)$
- Case 2: $\gamma = e^{-10}$, $\boldsymbol{\alpha} = 0.1 \times \mathbf{1}$, $\boldsymbol{\beta} = (e^1, e^1, e^1, 0.07e^1)$

These two cases will generate different number of topics, and Case 1 and Case 2 correspond to (0, 100] and (200, 300] respectively.

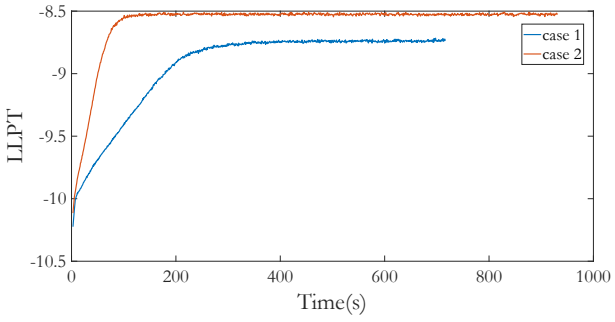


FIGURE 7. Convergence of the GHLDA on the NYTimes dataset.

Convergence speed. Fig.7 shows the training speed of the GHLDA. Case 1 and Case 2 can converge in about 300s and 100s, respectively.

Training time vs. number of topics. We investigate the relations between training time and number of topics in Fig.8.

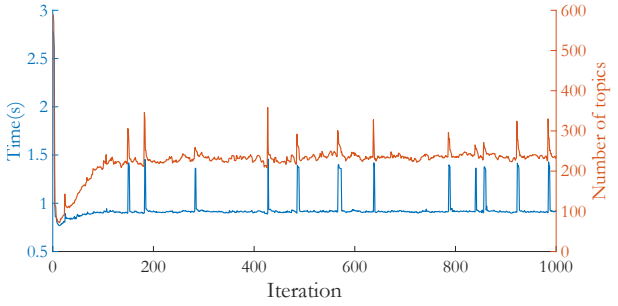


FIGURE 8. Training time and Number of topics VS. iteration on the NYTimes dataset for Case 2.

Obviously, when the number of topics increases, training time also increase, *i.e.*, training time is proportional to the number of topics.

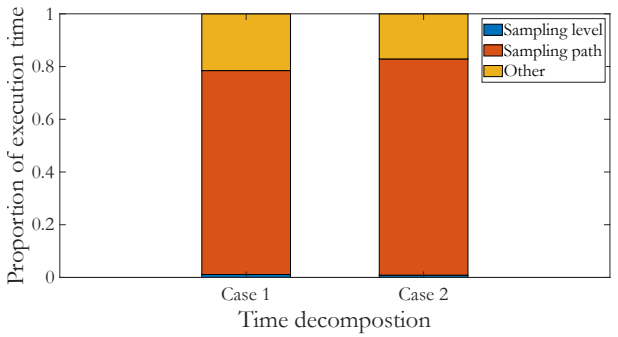


FIGURE 9. Time decomposition of different part in GHLDA

Time decomposition. From Fig.9, we can see that the most time consuming part for both Case 1 and Case 2 are the sampling the path. Time for sampling level is trivial compared with that of path sampling. Other time are spent on token list reordering, tree update, *etc*.

V. CONCLUSION

In this paper, we present the GHLDA, which is the first GPU-based HLDA known-to-all. We implement HLDA on GPU from systematic level and propose some innovative methods to accommodate GHLDA for GPU. We try to represent the complex tree structure by matrix and use memory pool to handle the dynamically changing memory usage. The proposed GHLDA can handle HLDA training on large corpus with merely one GPU. In the future, we plan to extend our method to multiple GPUs and other type of HLDA method.

REFERENCES

[1] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. Journal of machine Learning research, 3(Jan):993–1022, 2003.
[2] Jordan L. Boyd-Graber, David M. Blei, and Xiaojin Zhu. A topic model for word sense disambiguation. In EMNLP-CoNLL, 2007.
[3] Feng Zhang, Jidong Zhai, Xipeng Shen, Onur Mutlu, and Wenguang Chen. Zwift: A programming framework for high performance text analytics on compressed data. In Proceedings of the 2018 International Conference on Supercomputing, pages 195–206. ACM, 2018.

[4] Wen-Yen Chen, Jon-Chyuan Chu, Junyi Luan, Hongjie Bai, Yi Wang, and Edward Y Chang. Collaborative filtering for orkut communities: Discovery of user latent behavior. WWW, pages 681–690, 2009.
[5] Ralf Krestel, Peter Fankhauser, and Wolfgang Nejdl. Latent dirichlet allocation for tag recommendation. In Proceedings of the third ACM conference on Recommender systems, pages 61–68. ACM, 2009.
[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.
[7] Feng Qian, Lei Sha, Baobao Chang, Lu-chen Liu, and Ming Zhang. Syntax aware lstm model for chinese semantic role labeling. arXiv preprint arXiv:1704.00405, 2017.
[8] Yibo Wang and Wei Xu. Leveraging deep learning with lda-based text analytics to detect automobile insurance fraud. Decision Support Systems, 105:87–95, 2018.
[9] Luke Tierney. Markov chains for exploring posterior distributions. the Annals of Statistics, pages 1701–1728, 1994.
[10] Yuan Yang, Jianfei Chen, and Jun Zhu. Distributing the stochastic gradient sampler for large-scale lda. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1975–1984, 2016.
[11] Limin Yao, David Mimno, and Andrew McCallum. Efficient methods for topic model inference on streaming document collections. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 937–946, 2009.
[12] Kaiwei Li, Jianfei Chen, Wenguang Chen, and Jun Zhu. Saberlda: Sparsity-aware learning of topic models on gpus. ACM SIGOPS Operating Systems Review, 51(2):497–509, 2017.
[13] Xiaolong Xie, Yun Liang, Xiuhong Li, and Wei Tan. Culda_cgs: Solving large-scale lda problems on gpus. arXiv preprint arXiv:1803.04631, 2018.
[14] David Blei and John Lafferty. Correlated topic models. Advances in neural information processing systems, 18:147, 2006.
[15] Yee Teh, Michael Jordan, Matthew Beal, and David Blei. Sharing clusters among related groups: Hierarchical dirichlet processes. Advances in neural information processing systems, 17, 2004.
[16] Wei Li and Andrew McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In Proceedings of the 23rd international conference on Machine learning, pages 577–584, 2006.
[17] Amr Ahmed, Liangjie Hong, and Alexander Smola. Nested chinese restaurant franchise process: Applications to user tracking and document modeling. In International Conference on Machine Learning, pages 1426–1434. PMLR, 2013.
[18] John Paisley, Chong Wang, David M Blei, and Michael I Jordan. Nested hierarchical dirichlet processes. IEEE transactions on pattern analysis and machine intelligence, 37(2):256–270, 2014.
[19] Jay Pujara and Peter Skomoroch. Large-scale hierarchical topic models. In NIPS Workshop on Big Learning, volume 128, 2012.
[20] Chi Wang, Xueqing Liu, Yanglei Song, and Jiawei Han. Scalable and robust construction of topical hierarchies. arXiv preprint arXiv:1403.3460, 2014.
[21] Jianfei Chen, Jun Zhu, Jie Lu, and Shixia Liu. Scalable training of hierarchical topic models. Proceedings of the VLDB Endowment, 11(7):826–839, 2018.
[22] Nvidia. V100 GPU. Retrived from https://www.nvidia.com/en-us/data-center/v100/. Accessed: 2019, August 5.
[23] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.