2017 IEEE 24th International Conference on High Performance Computing Workshops (HiPCW)

Parallel LDA with Over-Decomposition

Gordon E. Moon, Aravind Sukumaran-Rajam and P. Sadayappan
Department of Computer Science and Engineering
The Ohio State University
Columbus, Ohio 43210, U.S.A
Email: moon.310@osu.edu, sukumaranrajam.1@osu.edu, sadayappan.1@osu.edu

***Abstract*—Latent Dirichlet Allocation (LDA) is a statistical technique for topic modeling. Prior efforts to parallelize LDA have either used expensive atomic operations or weakened the sampling model to enable parallelization without heavy use of atomics. In this paper, we present a parallel LDA implementation that uses an over-decomposed 2D tiling strategy to overcome the limitations of previous parallelization schemes. An alternate implementation that uses some approximation is also presented that fully avoids use of atomic operations. Experimental results with the NIPS and NYTimes datasets demonstrate the effectiveness of the new implementations.**

***Keywords*-Parallel Topic Modeling; Parallel Latent Dirichlet Allocation; Parallel Machine Learning;**

I. Introduction

Latent Dirichlet Allocation (LDA) is a statistical technique for topic modeling. Given a set of D documents, each represented as a collection of words, LDA determines a latent topic distribution for each document and each word in the vocabulary for the given corpus of documents. The original implementation of the algorithm proposed by Blei et al. [1] was very compute intensive and therefore parallel implementations [2]–[7] have been sought.

The sequential LDA algorithm (described in detail in the next section) performs a number of passes through the collection of documents. In each pass, each word of each document is processed. Three key data structures are read and written as each word is processed: a 2D array N representing the current word-to-topic distribution, a 2D array M maintaining the current document-to-topic distribution, and a 1D array NT holding the current topic-count distribution. Each word in each document is initially assigned randomly to one of the K latent topics, and as the iterations proceed, the word-to-topic and document-to-topic distributions in arrays N and M and the topic distribution array NT converge. When a word in a document is processed, some elements of all three arrays are read and one element of each array is modified. Thus, as a later word is processed, the modified distribution reflecting the processing of all previous words is used. This process of using the updated distributions is called collapsed Gibbs sampling (CGS) [16].

Many previous efforts at parallelizing LDA have sought to relax CGS and use approximations because strict CGS imposes constraints on efficient parallelization. One approach that has been explored [2] is to divide up the documents among parallel threads/processes, so that the document-to-topic array M is disjointly partitioned and exclusively read/written by only the owning thread/process. However, since common words are present in documents processed by different threads/processes, it is impossible to also disjointly partition and access the word-to-topic distribution matrix N across threads/processes. In a shared-memory parallel environment, a solution is to use atomic update operations on N, but the overheads can be high. A second alternative that has been considered is to perform uncollapsed Gibbs sampling, by using an independent copy of M, N and NT for each thread/process and only updating the local copy, thereby avoiding the need to use atomic operations. At the end of an iteration, the local copies of M, N and NT are *merged* to sample new parameters θ and ϕ, and then redistributed to the threads/processes before the start of the next iteration. However, the use of uncollapsed sampling leads to slower convergence [2].

In this paper, we develop a 2D tiled approach to perform efficient parallel LDA with CGS. We first develop a parallel algorithm that avoids the need for any atomic operations on N or M, but still requires atomic operations for the topic-count matrix NT. Load-balancing is a significant issue since different documents can have widely varying word counts. We use over-decomposition as a means of addressing the problem of load imbalance. We also devise an alternate atomics-free approach that introduces an approximation by maintaining local copies for NT. The performance and quality of solution for the two approaches are compared. Experiments on an 8-core multi-processor show that the new parallel implementation converges 4.9× faster than sequential CGS LDA, while maintaining the same quality.

The paper is organized as follows. In the next section, we present some background information on LDA and related prior work. In Section III, we present the parallel LDA algorithm that uses 2D tiling and over-decomposition, as well as the alternative fully atomics-free scheme. Experimental results are presented in Section IV using two widely used data sets: the NIPS and NYTimes datasets.

0-7695-6331-7/17/$31.00 ©2017 IEEE
DOI 10.1109/HiPCW.2017.00013

II. Background and Related Work

A. Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a topic modeling technique based on a hierarchical Bayesian model. It is used for identifying latent topics distributions for collections of text documents [1]. In LDA, each document *j* of *D* documents is modeled as a random mixture over *K* latent topics, denoted by θ_j. Each topic *k* is associated with a multinomial distribution over a vocabulary of *V* distinct words denoted by ϕ_k. LDA assumes that θ and ϕ are drawn from Dirichlet priors with fixed hyper-parameters α and β, respectively. For the i^{th} word token in the document *j*, a topic-assignment variable z_{ij} is sampled according to the topic distribution of the document $\theta_{j|k}$, and the word x_{ij} is drawn from the topic-specific distribution of the word $\phi_{w|z_{ij}}$. To draw the latent variable **z**, CGS is primarily used to infer the posterior distribution over latent variable **z**. A number of studies have shown that CGS is more effective than other sampling approaches since it reduces the variance considerably through marginalizing out all prior distributions of $\theta_{j|k}$ and $\phi_{w|k}$ during the sampling procedure [2], [4], [8]. Accordingly, given the parameters and all words except for the topic-assignment variable z_{ij}, the conditional distribution of z_{ij} can be calculated as:

$$P(z_{ij} = k|\mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta) \propto \frac{N^{\neg ij}_{x_{ij}|k} + \beta}{NT^{\neg ij}_{k} + V\beta}(M^{\neg ij}_{j|k} + \alpha) \quad (1)$$

where $M_{j|k} = \sum_w S_{w|j|k}$ is the document-to-topic count matrix, the number of word tokens in document *j* assigned to topic *k*; $N_{w|k} = \sum_j S_{w|j|k}$ is the word-to-topic count matrix, the number of occurrences of word *w* assigned to topic *k*; $NT_k = \sum_w N_{w|k}$ is the topic-count vector. The superscript $\neg ij$ denotes that the previously assigned topic of corresponding word token x_{ij} is excluded from the counts. Algorithm 1 shows the standard sequential LDA algorithm using CGS.

B. Related Work

Previous efforts at parallelizing LDA can be categorized into two broad groups, based on implementation for multicore CPUs [2], [3], [9], [10] versus GPU [4], [5], [11], [12]. Newman et al. [2] introduced a parallel MPI implementation of AD-LDA (Approximate Distribution LDA). They partitioned the documents over multiple processors and many other studies have used such a data partitioning scheme to parallelize CGS on CPUs [3], [9], [10]. The key global data structures, the word-to-topic count matrix and topic-count matrix are copied to each processor, and a reduction operation is used for updating local counts after each iteration. Several recently studies have developed GPU-based AD-LDA. However, maintaining a copy of global parameters on each processor is challenging for large-scale datasets due to the limited size of GPU memeory [4], [5].

Algorithm 1 Sequential LDA using CGS

Input: *DATA* containing *D* documents, **x** word tokens in each document and *V* distinct words, *K* topics and hyper-parameters α, β

Output: document-topic count matrix M, word-topic count matrix N and latent variable *Z*

```
1: repeat
2:   for doc = 0 to D − 1 do
3:     L ← doc_length
4:     for word = 0 to L − 1 do
5:       cur_word ← DATA[doc][word]
6:       old_topic ← Z[doc][word]
7:       decrement M[doc][old_topic]
8:       decrement N[cur_word][old_topic]
9:       decrement NT[old_topic]
10:      sum ← 0
11:      for k = 0 to K − 1 do
12:        sum ← sum + (N[cur_word][k]+β)/(NT[k]+Vβ) (M[doc][k] + α)
13:        p[k] ← sum
14:      end for
15:      U ← rand_uniform() × sum
16:      for k = 0 to K − 1 do
17:        if U < p[k] then
18:          break
19:        end if
20:      end for
21:      increment M[doc][k]
22:      increment N[cur_word][k]
23:      increment NT[k]
24:      Z[doc][word] ← k
25:    end for
26:  end for
27: until convergence
```

Even though an atomic operation guarantees the correctness of update, it negatively affects performance.

Some studies have used uncollapsed Gibbs sampling on GPUs, which does not marginalize out the prior distribution of θ and ϕ in each sampling step [11], [12]. Different from CGS, an uncollapsed Gibbs sampler maintains local copies of M, N and NT at each processor, thereby avoiding atomic operations. Although this approach is easier for parallelization of LDA, the uncollapsed Gibbs sampler requires more iterations to converge, compared to the standard collapsed Gibbs sampler [2], [12].

Thus, all prior parallel implementations of LDA have either used expensive atomic operations to maintain collapsed sampling [5], [12] or have deviated from collapsed sampling by creating local copies of the word-to-topic and/or document-to-topic matrices to avoid using atomics for updates [2]–[4], [13]. Unlike these prior efforts, the parallel LDA algorithm we describe in this paper maintains collapsed

sampling without atomic operations to update the word-to-topic or document-to-topic counts. This is achieved by partitioning the set of documents as well as the vocabulary into disjoint subsets, as explained in the next section. We show how CGS can be parallelized efficiently on multicores, thus maintaining the higher convergence rate of collapsed Gibbs sampling over uncollapsed Gibbs sampling.

III. Parallel LDA with Over-Decomposition

Our Parallel LDA is based on the CGS approach. As with prior approaches, the set of documents is disjointly partitioned for processing by different processors. The main difference is in the approach to handling conflicts in processing the word-to-topic count matrix. Since multiple documents can have the same words, two threads could simultaneously try to read/update the word-to-topic matrix, leading to a race condition. While atomic operation can be used to overcome the race condition, they can be expensive. Alternatively, if the parallelization was across the words, then the read/update to the document-to-topic matrix would lead to race conditions. In either case, there is an additional atomic update required on the topic-count vector.

Our parallel implementation is based on 2D tiling of the document-word matrix and is designed to considerably reduce the required number of atomic operations. The main idea is to partition both the collection of documents as well as the active vocabulary in the corpus into P disjoint sets, and reorganize the sampling over words in documents along diagonals in a 2D tiling of the document-word space, so as to avoid any conflicts between processors in modifying either the word-to-topic or document-to-topic count arrays.

A. Data Partitioning

Figure 1 illustrates the data/work partitioning scheme. The entire vocabulary (set of all words across all documents) is divided into P disjoint partitions. Similarly, the documents are also divided into P partitions. Thus the entire document-word matrix is divided into $P \times P$ partitions. For example, in Figure 1, words 0 to 9 in documents 0 to 3 form partition 0. Similarly, words 10 to 19 in documents 0 to 3 form partition 1. Assume that document 0 contain words {1, 2, 30, 44}, document 1 contains words {4, 9, 99}, document 2 contains words {50, 60} and document 3 contains words {5, 24, 30}. Partition 0 will consist of words {1, 2} in document 0, words {4, 9} in document 1, and word {5} in document 3.

B. Algorithm

In Figure 1, partitions which have mutually exclusive words and documents are marked with the same color and shading pattern. For example, consider partitions 0, 5, 10 and 15 which are along the diagonal of the partitioned document-word matrix. The set of words in each of these partitions is mutually exclusive. The documents in these partitions

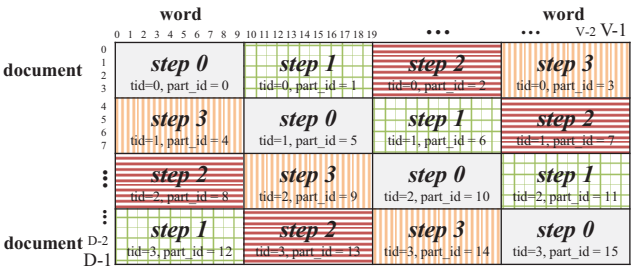

Figure 1. Data partitioning, if $P = 4$

are also mutually exclusive. Hence these partitions can be processed in parallel in a race-free manner without the need for any expensive atomic operations on the word-to-topic and document-to-topic arrays. Our parallelization approach is based on this scheme. All the partitions processed in parallel in the same step are distinct. If there are $P \times P$ partitions, then there will be P steps, each processing partitions along a wrap-around diagonal in the set of 2D data-tiles.

Algorithm 2 shows the parallel LDA algorithm. In Line 1, the number of documents in each partition is computed and in Line 2, the number of words in each partition is computed. Line 4 iterates over the number of steps. Each step corresponds to processing mutually exclusive partitions in parallel (Line 5). In Line 6 to Line 9, each thread determines the set of documents and words that it will process in the current step. In Lines 10 to 33, we compute and update the word-to-topic, document-to-topic, and topic-count vector. The only atomic operations needed are in processing the topic-count vector.

C. Load Balancing

An important issue to be addressed is load imbalance. Each document only uses a subset of words in the vocabulary. Hence, in each 2D partition of the document-word matrix, the number of words can be quite non-uniform. It is even possible that some partitions are empty. Note that the amount of work in each partition is proportional to the number of words.

Consider Figure 1, where the document-word matrix is divided into 4×4 partitions and the number of threads is 4. Consider step 0, in which partitions 0, 5, 10, and 15 are processed by threads 0, 1, 2, and 3, respectively. Assume that the number of words in partition 0, 5, 10, and 15 are 100, 0, 10, 15, respectively. Thread 0 has much more work when compared to other threads. Hence threads 1, 2, and 3 will be idle as they wait for thread 0 to complete. Note there is an implicit barrier synchronization at the end of each step.

In order to achieve load balancing, we use over-decomposition. Instead of dividing the document-word matrix into $P \times P$ partitions, where P is the number of

Algorithm 2 Parallel LDA

Input: *DATA_PREP* containing D documents, **x** word tokens in each document and V distinct words, P partitions, K topics and hyper-parameters α, β

Output: document-topic count matrix M, word-topic count matrix N and latent variable Z

```
1: doc_part_size ← ceil(D / P)
2: voc_part_size ← ceil(V / P)
3: repeat
4:   for step = 0 to P − 1 do
5:     for part_id = 0 to P − 1 in parallel do
6:       r_start ← part_id × doc_part_size
7:       r_end ← min(r_start + doc_part_size, |D|)
8:       c_start ← (part_id + step) % (P × voc_part_size)
9:       c_end ← min(c_start + voc_part_size, |V|)
10:      for doc = r_start to r_end − 1 do
11:        for word = c_start to c_end − 1 do
12:          cur_word ← DATA_PREP[doc][word]
13:          old_topic ← Z[doc][word]
14:          decrement M[doc][old_topic]
15:          decrement N[cur_word][old_topic]
16:          atomic decrement NT[old_topic]
17:          sum ← 0
18:          for k = 0 to K − 1 do
19:            sum ← sum + (N[cur_word][k]+β) / (NT[k]+Vβ) (M[doc][k]+α)
20:            p[k] ← sum
21:          end for
22:          U ← rand_uniform() × sum
23:          for k = 0 to K − 1 do
24:            if U < p[k] then
25:              break
26:            end if
27:          end for
28:          increment M[doc][k]
29:          increment N[cur_word][k]
30:          atomic increment NT[k]
31:          Z[doc][word] ← k
32:        end for
33:      end for
34:    end for
35:  end for
36: until convergence
```

threads, we divide the document-word matrix into $(O \times P) \times (O \times P)$ partitions, where O is the over-decomposition factor. Since there is no synchronization within each step, the threads can dynamically acquire a partition. For example, in Figure 2, threads 0, 1, 2, and 3 initially process partitions 5, 18, 31, and 44, respectively. Thread 1 and thread 2 only have 10 units of work, compared to 100 units for thread 0 and 35 for thread 3. Therefore thread 1 and 2 will complete partitions 18 and 31 and then process partition 57 and 70, respectively.

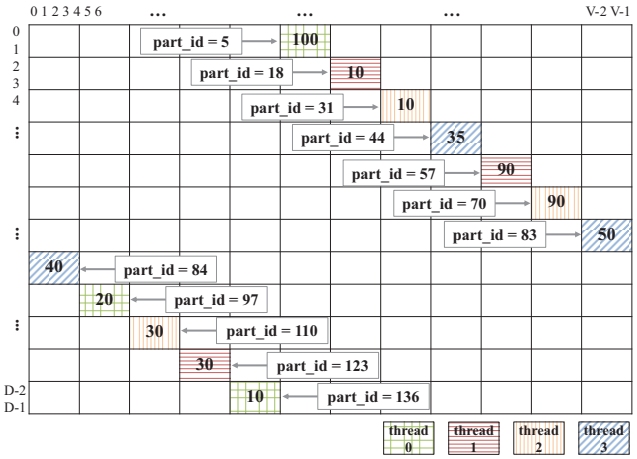

Figure 2. Distribution of threads in a step via over-decomposition, for 12 partitions and 4 threads. The number in each data block indicates the number of word tokens.

D. Alternate Atomics-free Scheme

The previously presented approach requires atomic operations to update the topic-count vector. In this subsection, we provide an alternate scheme that does not require any atomic operations. The core idea behind the new scheme is to use a thread-local *delta* vector to keep track of the local changes to the topic-count vector. The global state of topic-count is maintained by the global topic-count vector. When a thread requires a particular topic's count, it first reads the corresponding global topic-count vector value and adds the corresponding value in its local *delta* vector. When a thread increments/decrements the topic-count vector, the increment/decrement operation is simply performed on the local *delta* vector. Since the updates are made locally, atomic operations are not required.

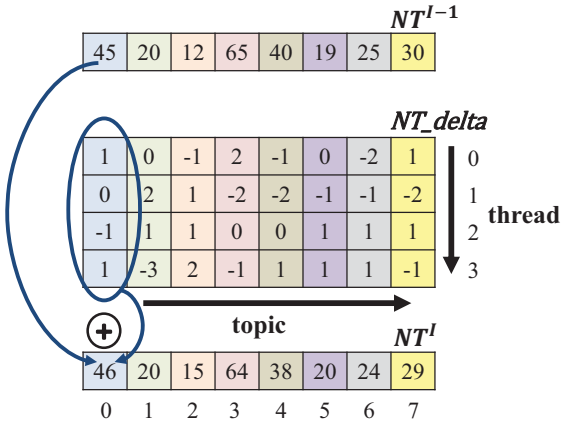

Figure 3. Reduction in alternate atomics-free scheme

The entire set of *delta* vectors can be view as matrix (NT_delta matrix). Figure 3 shows the NT_delta matrix. The number of rows is equal to the number of threads and the number of columns is equal to the number of topics.

At the beginning of each outer iteration, the NT_delta matrix is initialized to zero. In Algorithm 2, atomic operations are used to increment (Line 30)/decrement (Line 16) the topic-count vector. Instead, in the new approach, the increment is done as: NT_delta[tid][topic] += 1. The decrement operation is done similarly. In the new scheme, a thread reads the topic-count vector (equivalent of Line 19) as follows: NT_delta[tid][topic] + NT[topic]. Thus, there are no atomic operations for reads or writes.

At the end of each iteration, the sum of each column of NT_delta is added to the corresponding column of the NT vector. In order to perform this operation, the columns of the NT_delta matrix are partitioned into column panels and each column panel is processed by a single thread. Thus, the latter reduction step can also be performed in parallel and without atomics.

IV. Experimental Evaluation

We evaluated the new parallel LDA variants using two common datasets: the NIPS and NYTimes datsets from the UCI Machine Learning Repository [14]. Documents that do not contain any word token were discarded. Table I describes the characteristics of the datasets.

We used 90% of the documents as a training set, and the remaining 10% as the test set and used cross-validation evaluation [15]. As suggested in [16], the α and β were set to 50.0/*K* and 0.1, respectively. We used GibbsLDA++ [17], which is a standard C++ implementation of sequential LDA with CGS, to compare the performance and accuracy of the parallel OD-LDA implementation. We first present an evaluation of the base algorithm and then discuss the fully atomics-free alternate scheme. The LDA models we used in our experiments are:

- Sequential collapsed: sequential GibbsLDA++ with CGS
- Parallel OD-LDA: parallel LDA with CGS

Table I
STATISTICS OF DATASETS USED IN THE EXPERIMENTS. *D* IS THE NUMBER OF DOCUMENTS, *W* IS THE TOTAL NUMBER OF WORD TOKENS AND *V* IS THE SIZE OF THE ACTIVE VOCABULARY.

Dataset	*D*	*W*	*V*
NIPS	1,500	1,932,365	12,375
NYTimes	299,752	99,542,125	101,636

A. Experimental Environment

We evaluated performance on an 8-core Intel Xeon CPU. Table II shows the details of the benchmarking machine.

Table II
MACHINE DETAILS

Hardware	Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz (8 cores) 16 GB RAM (1866 MHz)
Software	Red Hat Enterprise Linux Server release 6.7 GCC version 4.9.2

B. Evaluation Metric

To evaluate the quality of solution, we measured the per-word log-likelihood.

$$p(\mathbf{x}^{test}) = \prod_{ij} log \sum_{k} \frac{N_{w|k} + \beta}{\sum_w N_{w|k} + V\beta} \frac{M_{j|k} + \alpha}{\sum_k M_{j|k} + K\alpha} \tag{2}$$

$$\text{log-likelihood} = \frac{1}{W^{test}} log(p(\mathbf{x}^{test})) \tag{3}$$

where W^{test} is the total number of word tokens in the test set. The higher the log-likelihood, the better the generalization of the model to unseen data.

Table III
COMPARISON OF THE ELAPSED TIME PER ITERATION ON NIPS AND NYTIMES DATASETS, *K*=128 AND *P*=300 ON NIPS AND NYTIMES DATASET.

	Number of threads	**NIPS** Elapsed time per iteration (s)	**NYTimes** Elapsed time per iteration (s)
Sequential collapsed	None	1.1929	62.6343
Parallel OD-LDA	1	1.2406	66.2232
Parallel OD-LDA	2	0.7132	36.5454
Parallel OD-LDA	4	0.4421	21.0296
Parallel OD-LDA	8	0.2857	12.6711

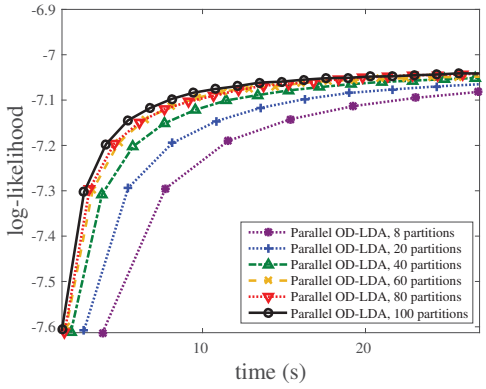

Figure 4. Convergence over time with different number of partitions on NIPS dataset, *K*=128, 4 threads.

Figure 4 shows the performance variation when the number of partitions is varied for the NIPS dataset. The parallel LDA with 100 partitions converges 2.8× faster than 8 partitions because of better load balance.

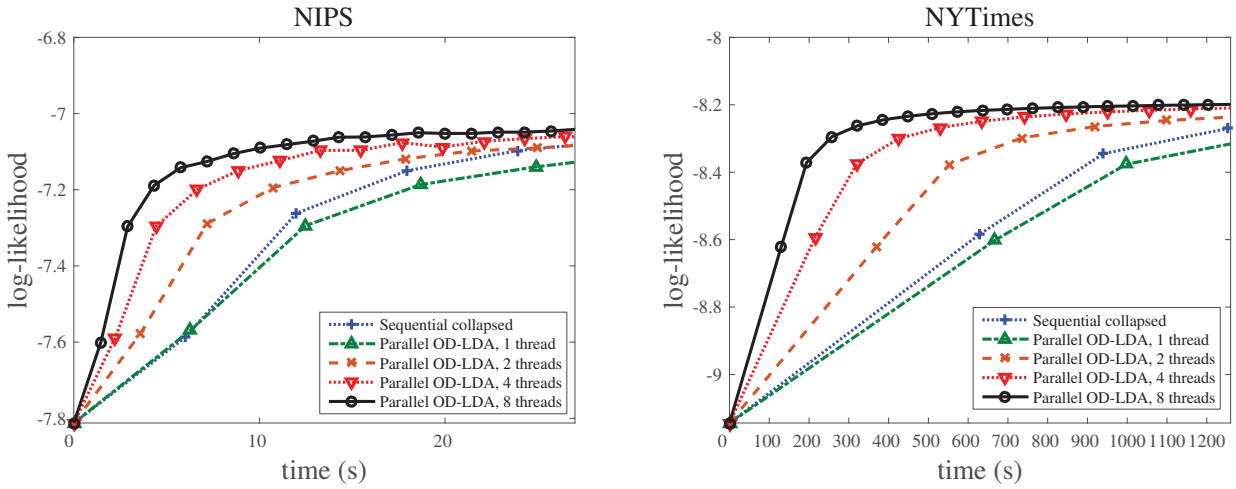

Figure 5. Convergence over time with 1, 2, 4 and 8 threads, on NIPS and NYTimes datasets, K=128. The number of partitions is set to 300 for both NIPS and NYTimes datasets. X-axis: elapsed time in seconds; Y-axis: per-word log-likelihood.

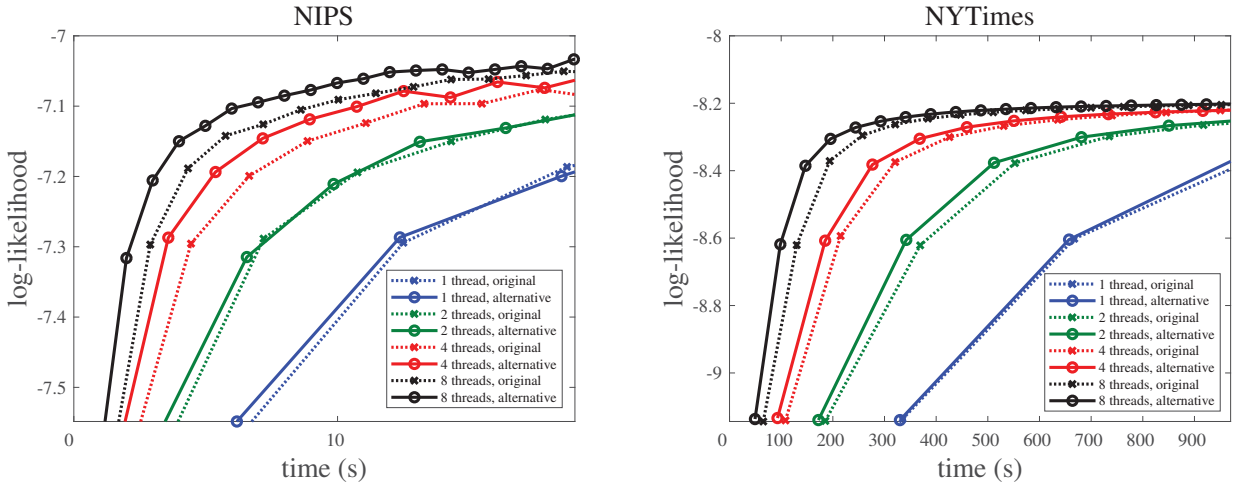

Figure 6. Comparison between alternate atomics-free scheme and the original scheme

C. Speedup

Table III compares our approach with the sequential approach. We measured the elapsed time per iteration to evaluate speedup of the parallel algorithm. The execution time of the baseline sequential collapsed model is similar to the parallel OD-LDA model with 1 thread. As expected, the execution time decreases as the number of threads increases. With 8 threads, the parallel LDA scheme achieved 4.1× speedup on the NIPS dataset and 4.9× speedup on the NYTimes dataset.

Figure 5 compares the log-likelihood versus time for various approaches. The parallel approach converges much faster than the sequential version while maintaining similar log-likelihood.

Finally, Figure 6 shows performance comparison of the alternate scheme and original scheme. It can be seen that the alternative scheme takes less time for convergence.

V. Conclusion

In this paper we have presented a parallel CGS for LDA. We evaluate the approach on two commonly used datasets: the NIPS and NYTimes datasets. The new parallel implementation is able to effectively parallelize the CGS approach while maintaining the same log-likelihood. We use over-decomposition techniques for load balancing. The experimental section demonstrates that we achieve 4.1× speedup for NIPS and 4.9× speedup for NYTimes.

References

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[2] D. Newman, A. Asuncion, P. Smyth, and M. Welling, "Distributed algorithms for topic models," *Journal of Machine Learning Research*, vol. 10, no. Aug, pp. 1801–1828, 2009.

[3] Y. Wang, H. Bai, M. Stanton, W.-Y. Chen, and E. Y. Chang, "Plda: Parallel latent dirichlet allocation for large-scale applications." *AAIM*, vol. 9, pp. 301–314, 2009.

[4] F. Yan, N. Xu, and Y. Qi, "Parallel inference for latent dirichlet allocation on graphics processing units," in *Advances in Neural Information Processing Systems*, 2009, pp. 2134–2142.

[5] M. Lu, G. Bai, Q. Luo, J. Tang, and J. Zhao, "Accelerating topic model training on a single machine," in *Asia-Pacific Web Conference*. Springer, 2013, pp. 184–195.

[6] H. Zhao, B. Jiang, J. F. Canny, and B. Jaros, "Same but different: Fast and high quality gibbs parameter estimation," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1495–1502.

[7] Z. Liu, Y. Zhang, E. Y. Chang, and M. Sun, "Plda+: Parallel latent dirichlet allocation with data placement and pipeline processing," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 26, 2011.

[8] B. Zhang, B. Peng, and J. Qiu, "High performance lda through collective model communication optimization," *Procedia Computer Science*, vol. 80, pp. 86–97, 2016.

[9] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling, "Fast collapsed gibbs sampling for latent dirichlet allocation," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 569–577.

[10] P. Smyth, M. Welling, and A. U. Asuncion, "Asynchronous distributed learning of topic models," in *Advances in Neural Information Processing Systems*, 2009, pp. 81–88.

[11] J.-B. Tristan, D. Huang, J. Tassarotti, A. C. Pocock, S. Green, and G. L. Steele, "Augur: Data-parallel probabilistic modeling," in *Advances in Neural Information Processing Systems*, 2014, pp. 2600–2608.

[12] J.-B. Tristan, J. Tassarotti, and G. Steele, "Efficient training of lda on a gpu by mean-for-mode estimation," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 59–68.

[13] P. Xue, T. Li, K. Zhao, Q. Dong, and W. Ma, "Glda: Parallel gibbs sampling for latent dirichlet allocation on gpu," in *Conference*. Springer, 2016, pp. 97–107.

[14] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[15] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2. Stanford, CA, 1995, pp. 1137–1145.

[16] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[17] X.-H. Phan and C.-T. Nguyen, "Gibbslda++: A c/c++ implementation of latent dirichlet allocation (lda)," 2007. [Online]. Available: http://gibbslda.sourceforge.net/