# A Parallel Algorithm for LDA using Stochastic Collapsed Variational Bayesian Inference

Ashish Baghudana
Department of Computer Science
College of Engineering
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061
Email: ashishb@vt.edu

Vartan Kesiz Abnousi
Department of Agricultural and Applied Economics
College of Agriculture and Life Sciences
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061
Email: vkesizab@vt.edu

*Abstract*—**Parallel computation has become progressively more important for data mining and machine learning. A very popular text mining technique called latent Dirichlet allocation (LDA) is now part of every data scientist's toolkit. LDA helps understand semantic themes within a large corpus of text. However, the algorithm in its original form is serial and does not easily scale to millions of documents. Researchers have developed newer sampling techniques, specifically stochastic collapsed variational Bayesian (SCVB0) inference, that can learn topics in an online fashion. In this project, we implement the SCVB0 algorithm, parallelize it using OpenMP. We plot the algorithm's performance with respect to its operational intensity on the Roofline model and characterize the algorithm on two different datasets – Enron Emails with 39,861 documents and NIPS Full Papers with 1,500 documents.**

## I. Introduction

Topic models such as latent Dirichlet allocation (LDA) [1] have become mainstream in the modern machine learning and text mining toolkit. In their seminal work, Blei *et al.* introduced a generative probabilistic model that treats text corpora as a collection of topics, and hierarchically models a topic as distribution over words. With ever-increasing size of text data in the Internet-era, there is a growing need for text mining algorithms that scale, and can learn representations of such texts efficiently and accurately. An overview of topic modelling and the latent Dirichlet allocation algorithm is presented in Section II-A.

Traditional LDA techniques use Gibbs sampling (a Markov chain Monte Carlo method) [2], variational inference (using Expectation Maximization) [1], [3], or a combination thereof, to learn the "latent" model parameters. However, these methods do not scale easily to millions of documents – each iteration of the inference becomes time-consuming and as a result, the algorithm is not run till convergence.

Recent advances by Hoffman *et al.* introduced the stochastic variational inference algorithm for LDA topic models [3]. This method does not need to see all of the documents before updating the topics, and can therefore learn a good representation of the topics even before the first iteration is complete. Since the algorithm sees documents in an online fashion, it can be applied to corpora of large sizes without memory constraints. However, the online algorithm does not make use of the collapsed form of the sampling equation that allows for more accurate topic models. Foulds et al. introduced the stochastic collapsed variational Bayesian inference (SCVB0) algorithm to remedy the poorer accuracy of the Hoffman method, while retaining the same online nature.

In this project, we implement the SCVB0 algorithm. The algorithm poses opportunities for parallelization and optimization. The iterative approach of this technique lends itself to parallelization through OpenMP. Furthermore, the steps in the iteration are composed of basic arithmetic operations – addition, multiplication, and division. We also hope to characterize the algorithm on a standard dataset created by David Newman at the University of California, Irvine[1]. This will involve calculating running times of single-threaded, multi-threaded, and GPU-accelerated versions of the algorithm.

## II. Background and Related Work

In this section, we introduce the necessary background literature and related work in latent Dirichlet allocation

[1] https://archive.ics.uci.edu/ml/datasets/bag+of+words

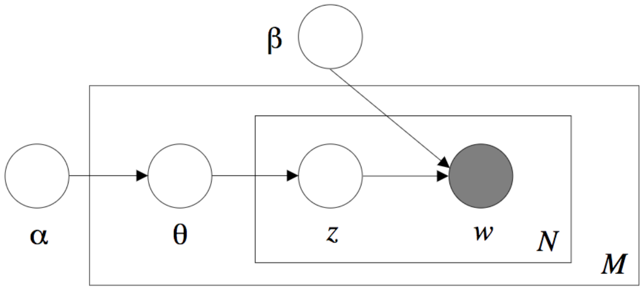

Fig. 1. Graphical model representation of LDA. The outer plate represents $M$ documents, each having $N$ words. Each document $m$ has a topic distribution $\theta$ and each word $n_m$ is associated with a topic $z$. $\alpha$ and $\beta$ are hyperparameters for the model.

and the different parallelization techniques in this domain. We split this into two parts – an introduction to LDA and a description of stochastic collapsed variational Bayesian inference algorithm for LDA.

### A. Latent Dirichlet Allocation

Researchers in information retrieval for large text corpora initially proposed the *tf-idf* (term frequency-inverse document frequency) method of weighting terms and assigning importance to words based on their frequencies [4]. The *tf-idf* algorithm counts the number of times a term occurs in the corpus and normalizes it against the number of times the term occurs in a document. The *tf-idf* reduction establishes a set of terms important for each document, and subsequently for the corpus. However, it failed to capture the semantic structure of documents. This was partially alleviated by Deerwester et al. in the latent semantic indexing (LSI) technique [5]. LSI uses a singular value decomposition of the tf-idf matrix to identify a linear subspace in the space of tf-idf features that captures most of the variance in the collection. Following this, Hoffman et al. proposed the probabilistic latent semantic indexing (pLSI) algorithm that modeled a document as a mixture of topics based on the likelihood principle. However, there is no generative process for LSI-based algorithms, and it cannot be used to assign topic probabilities to documents outside the training set.

Latent Dirichlet Allocation (LDA) is an unsupervised machine learning algorithm, which assumes a hierarchical Bayesian dependency between documents, topics, and words in a corpus. Since the algorithm is generative in nature, it can be used to predict topic probabilities for unseen documents as well.

LDA assumes the following generative process:

- For each topic $k \in K$, draw topic distribution $\phi_k \sim Dirichlet(\beta)$
- For each document $m$,
  - Draw topic distribution $\theta \sim Dirichlet(\alpha)$
  - For each word $n$ in document $m$,
    - Draw topic $z_{m,n} \sim Multinomial(\theta_m)$
    - Draw word $w_{m,n} \sim Multinomial(\phi_{z_{m,n}})$

Stated alternatively, LDA reduces to the calculation the posterior distribution of the hidden variables in the document.

$$P(\theta, z|w, \alpha, \beta) = \frac{P(\theta, z, w|\alpha, \beta)}{P(w|\alpha, \beta)} \tag{1}$$

However, the posterior distribution is intractable [1] for exact inference, but can be approximated using variational inference and Markov chain Monte Carlo (MCMC). The original implementation used variational inference code for which is available on Github[2]. Subsequently, Griffith et al. use MCMC to show that the topics obtained were consistent with that in variational inference. This technique is also referred to as collapsed Gibbs sampling. Several versions of LDA implementations are available online in different languages. We refer to the GibbsLDA++ which is publicly available [3].

However, both approximation techniques are run serially and require that all documents to be available at the time of running the algorithm. The serial nature of these inference techniques makes it difficult to scale for large datasets.

### B. Stochastic Collapsed Variational Bayesian (SCVB) Inference for Latent Dirichlet Allocation

To alleviate the serial restrictions of LDA, researchers have come up with different algorithms and architectures that speed up LDA. Smola et al. developed PLDA, a MapReduce-based algorithm [6] that uses a novel sampling architecture and a distributed key-value store for synchronizing the sampler state between computers. This algorithm does not make LDA faster, but increases its throughput through parallelization. While the algorithm is useful in its own right, the result is orthogonal to faster convergence.

To increase the speedup of the LDA algorithm, Hoffman et al. develop an online learning technique [3] that uses stochastic optimization. The algorithm is briefly described in Algorithm 1. Because the algorithm does not need to see all of the documents before updating

[2] https://github.com/blei-lab/lda-c

[3] https://github.com/mrquincle/gibbs-lda

**Algorithm 1** Stochastic Variational Inference (Hoffman et al.) courtesy (Foulds et al.)

```
Input: Data points $x_1, \dots, x_D$ (word count histogram
  for documents), step sizes $\rho_t, t = 1 : m$ where $m$ is
  the number of iterations.
  Initialization: Randomly initialize “global” (e.g. topic)
  parameters G
  for $t = 1 : m$ do
    Select a random data point $x_j$, $j \in \{1 \dots D\}$
    Compute “local” variational parameters $\mathbf{L_j}$
    $\hat{G} = D \times L_j$
    $\mathbf{G} := (1 - \rho_t) \times \mathbf{G} + \rho_t \times \hat{G}$
  end for
```

the topics, this method can often learn good topics before a single iteration of the traditional batch inference algorithms would be completed. However, this online LDA algorithm does not take advantage of the collapsed representation of the model and uses variational inference. As described before, topics obtained from collapsed representation tend to be more accurate than ones from the uncollapsed representation.

Foulds et al. introduce a stochastic collapsed variational Bayesian (SCVB0) inference algorithm for LDA [7] which can run fast, as well as, produce more accurate topics than [3]. The original implementation is in Julia[4], however there are ports of the implementation in C++. We use the one available on Github as part of the MeTA (ModErn Text Analysis) toolkit[5].

Without delving deep into the sampling techniques in SCVB0, we briefly describe the notations (Table I) and algorithm (Algorithm 2) used. The equations for the updates are mentioned in the appendix and can also be found in [7].

The SCVB0 algorithm provides scope for parallelization at the minibatch level. Each minibatch can be executed independent of the other and can be run across multiple CPUs (and possibly in a distributed fashion as well). In this project, we attempt to measure the relative speeds of single-threaded SCVB0, multi-threaded SCVB0, GPU-accelerated SCVB0 and vanilla-LDA (single-threaded Gibbs Sampling). We plan to use OpenMP and OpenACC extensively for parallelizing our loops.

TABLE I
LIST OF NOTATIONS USED IN THE SCVB0 ALGORITHM. TABLE BORROWED FROM [7].

| | |
|---|---|
| $K$ | Number of topics |
| $D$ | Number of documents |
| $C$ | Number of words in the corpus |
| $C_j$ | Number of words in the document $j$ |
| $z_{i,j}$ | Topic for $(i, j)$, the $i$th word of the $j$th document |
| $w_{i,j}$ | Dictionary index for word $(i, j)$ |
| $\theta_j$ | Distribution over topics for document $j$ |
| $\phi_k$ | Distribution over words for topic $k$ |
| $\alpha$ | Dirichlet prior for $\theta$ |
| $\eta$ | Dirichlet prior for $\phi$ |
| $\gamma_{ij}$ | Variational distribution for word $(i, j)$ |
| $N^\Theta$ | Expected topic counts per document |
| $N^\Phi$ | Expected topic counts per word |
| $N^Z$ | Expected topic counts overall |
| $Y^{(ij)}$ | Estimate of $N^\Phi$ based only on word $(i, j)$ |
| $M$ | Minibatch, a set of documents |
| $\hat{N}^\Phi$ | Estimate of $N^\Phi$ from current minibatch |
| $\hat{N}^Z$ | Estimate of $N^Z$ from current minibatch |
| $\rho_p^\Theta$ | Step size for $N^\Theta$ at timestep $t$ |
| $\rho_p^\Phi$ | Step size for $N^\Phi$ and $N^Z$ at timestep $t$ |
| $w_{aj}$ | Dictionary index for $a$th distinct word of $j$ |
| $m_{aj}$ | Count of $a$th distinct word of $j$ |
| $\gamma_{aj}$ | Variational dist. for $a$th distinct word of $j$ |

## III. Dataset and Methods

In this section we describe the dataset used, the preprocessing steps involved, and finally the method we intend to use.

### A. Dataset

We use a dataset collection curated by David Newman[6] at University of California, Irvine (UCI). The dataset is available publicly on the Machine Learning Repository hosted by UCI[7]. The dataset consists of 5 text collections (Table II). The collections vary in the number of documents and the number of words. The preprocessing pipeline consists of various steps (this too can be parallelized for each document).

1) Each document is lowercased
2) Each document is tokenized into individual words
    a) Discard words that are stopwords (for example: a, the, with, etc.)

[4] https://github.com/jrfoulds/Stochastic-CVB0

[5] https://github.com/meta-toolkit/meta

[6] http://www.ics.uci.edu/newman/

[7] https://archive.ics.uci.edu/ml/datasets/bag+of+words

**Algorithm 2** Stochastic Collapsed Variational Bayesian Inference (SCVB0)

```
Input: Documents $D$, number of topics $K$, number of words $C$
  Initialization: Random $N^{\Theta}$, $N^{\Phi}$; $N^{Z} := \sum_{w} N^{\Phi}_{w}$
  for each minibatch $M$ do
    $\hat{N}^{\Phi} := 0$;
    $\hat{N}^{Z} := 0$;
    for each document $j$ in $M$ do
      if $pass < burn\_in$ then
        for each token $i$ in $j$ do
          Update $\gamma_{ij}$ (Equation 2)
          Update $N^{\Phi}_{j}$ (Equation 3)
        end for
      else
        for each token $i$ in $j$ do
          Update $\gamma_{ij}$ (Equation 2)
          Update $N^{\Phi}_{j}$ (Equation 3)
          $\hat{N}^{\Phi}_{w_{ij}} := \hat{N}^{\Phi}_{w_{ij}} + C_{\gamma_{ij}}$
          $\hat{N}^{Z}_{w_{ij}} := \hat{N}^{Z}_{w_{ij}} + C_{\gamma_{ij}}$
        end for
      end if
    end for
    Update $N^{\Phi}$ (Equation 4)
    Update $N^{Z}$ (Equation 5)
  end for
```

TABLE II
TEXT COLLECTIONS IN THE DATASET

| Collection | Documents | Words |
|---|---|---|
| Enron Emails | 39,861 | 28,102 |
| NIPS Full Papers | 1,500 | 12,419 |

b) Discard any token that is just a punctuation mark

c) Discard any token that is not ASCII

d) Lemmatizing words (for example: running, run, ran should all be transformed to run)

Subsequently, we build a vocabulary where we map each token an integer ID and represent each document as a bag-of-words.

### *B. Methods*

Running times are affected by the number of documents, the number of unique words, length of the documents, and the number of topics. We plan to run the algorithm on two datasets – one small and one large. We choose the NIPS full paper collection and the Enrol Email collection respectively for the two cases. The number of documents and words respectively, are presented in Table II.

In Algorithm 2, we note that SCVB0 divides the dataset into multiple minibatches. Each mininbatch is independent of the other. At the end of the minibatch, the total counts are updated and the algorithm is run through another iteration. Since the algorithm iteratively tries to converge to the global minima and computes the step by updating the counts after every minibatch, it can be parallelized easily at this level.

We develop two implementations of the SCVB0 algorithm.

1) Single-threaded version
2) Multi-threaded version using OpenMP

We use the MeTA implementation as our baseline and reference the original Julia implementation as a guideline. Additionally, we will plot the algorithm on the Roofline model using the rlogin and ironclaw clusters.

*1) **Roofline Model:*** Roofline is a visually intuitive performance model used to bound the performance of various numerical methods and operations running on multicore, manycore, or accelerator processor architectures. Instead of using percent-of-peak estimates, the model can be used to assess the quality of attained performance by combining locality, bandwidth, and different parallelization paradigms into a single performance figure. One can examine the resultant Roofline figure in order to determine both the implementation and inherent performance limitations. In this paper, we implemented Berkeley’s Lab open source ”Empirical Roofline Tool” and ”Roofline Visualizer” for rlogin and ironclaw1. The code was developed by Berkeley Lab and it is open sourced [8] [8]. As we discuss analytically in the Section IV, the y-axis gives us the performance bound (GFlop/s) of the machine while the x-axis reflect the Arithmetic Intensity (Flops/Bytes) of the operation, in this case, the SCVB0 algorithm. An intuitive visualization of different operations on their arithmetic intensity is illustrated in Figure 2.

*2) **Operational Intensity:*** The Operational Intensity (henceforth OI), sometimes referred to as Arithmetic Intensity, is the ratio of total floating-point operations to total data movement (bytes). Some authors differentiate OI and Arithmetic Intensity because, among other reasons, arithmetic intensity measures the traffic between the processor and the cache, whereas efficiency-level

[8] https://bitbucket.org/berkeleylab/cs-roofline-toolkit/src/master/

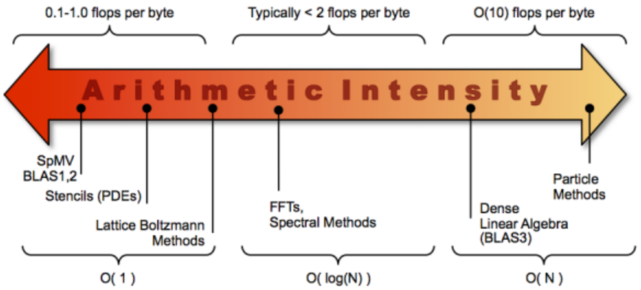

Fig. 2. Arithmetic Intensity of different operations. Source: USDE Berkeley Lab Computational Research

programmers want to measure traffic between the caches and DRAM [9].This is visualized in Figure 2. There have been various attempts to quantify OI. OI is both architecture- and kernel dependent and thus must be calculated for every kernel architecture combination. One way to calculate OI is to use performance counters to measure the actual number of operations and to measure the actual amount of memory traffic when running the kernel. Tools like Intel VTune [10] follow this methodology. In practice, depending on the kernel, it may be easy to calculate both the number of interesting operations and the minimum memory traffic by hand, which is our preferred method. Thus, one can bound the OI. Subsequently, the formula [9] that we use to compute the OI is as follows:

$$OI : I(n) = \frac{W(n)}{Q(n)} \tag{2}$$

Where $W(n)$ is the number of flops per input size, $Q(n)$ is the number of bytes transferred or memory traffic for input size n. Based on our estimates, the OI is approximately 0.135 (Flops/Byte). However, in the future we conduct a more scrupulous analysis to validate this number by implementing some of the toolkits and methods that we underlined.

## IV. Results

The analytic results are reported in Table III. Moreover, we report the absolute running times in seconds of LDA-C and LDA-SCVB0 for each dataset and machine. Furthermore, we present a number of plots that provide more insight on the results. In Figure 3, we plot the speedup of SCVB0 with the 1 threaded LDA-C algorithm as our baseline. We run all the implementations on two datasets  NIPS and Enron, and two machines  rlogin and ironclaw1. For the NIPS Full Papers data, we observe that there is no significant performance gains after 4 threads in both machines, even though ironclaw1 has a slightly better performance. In contrast, for the Enron Emails data we observe significant speedup gains up until 20 threads. Interestingly enough, this time rlogin has a better performance than ironclaw1. It should be stressed that the Enrol Emails dataset contains approximately 26.5 times more documents and 2.25 times more words than the NIPS Full Papers data. The implication is clear, the smaller dataset plateaus its speedup gains earlier than the significantly larger Enrol Emails data. In Figure 4, we plot speedup of SCVB0 with the 1 threaded SCVB0 as our baseline. As before, we run all implementations on two datasets  NIPS and Enron, and two machines  rlogin and ironclaw1. Unsurprisingly, we get the same results. The speedup for the smaller NIPS data stops plateaus after 4 threads, whereas for the Enrol Emails see significant speedup gains until 20 threads. In addition, the speedup is higher in rlogin than ironclaw1. This difference is attributed on the characteristics of each machine. Specifically, rlogin is using has an Intel(R) Xeon(R) CPU E5-2470 v3 @ 2.40GHz CPU model with a CPU clock frequency of 1548.937Mhz. On other hand, ironclaw1 has an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz CPU model with CPU clock frequency of 1200.093Mhz. We also plot the Roofline Models for the two machines in Figures 5 and 6. The results are identical except for the DRAM bandwidth, which due to the different specifications of the machines as indicated earlier. The black vertical line in the Roofline Models indicates the OI of the LDA-SCVB0 algorithm. As the black vertical line indicates, the LDA-SCVB0 algorithm is memory bound. Specifically, L1 and L2 lines are the cache memory bounds. On the other hand, the horizontal red line, labeled as GFLOPS in the plot, is the GFLOPS CPU-Bound. Finally, in Figure 7 we plot the number of time (in seconds) vs the number of topics for the LDA-SCVB0 model. by keeping the number of threads fixed (4 threads) after 100 iterations. The results indicate that there a strong scaling as we increase the number of topics. In addition, the time for the large Enrol Emails dataset increases almost linearly.

TABLE III
ABSOLUTE RUNNING TIMES OF LDA-C AND LDA-SCVB0 IN SECONDS.

| Dataset | Machine | LDA-C (sec) | LDA-SCVB0 (sec) | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 thread | 1 thread | 2 threads | 4 threads | 8 threads | 16 threads | 20 threads |
| NIPS | rlogin | 839.799 | 26.119 | 25.589 | 16.723 | 16.275 | 16.175 | 16.186 |
| NIPS | ironclaw | 821.238 | 23.029 | 23.568 | 16.491 | 16.361 | 16.956 | 16.577 |
| Enron | rlogin | 4222.106 | 166.832 | 136.425 | 85.988 | 64.468 | 54.625 | 53.277 |
| Enron | ironclaw | 4162.285 | 156.27 | 130.571 | 94.244 | 71.198 | 65.3 | 64.678 |

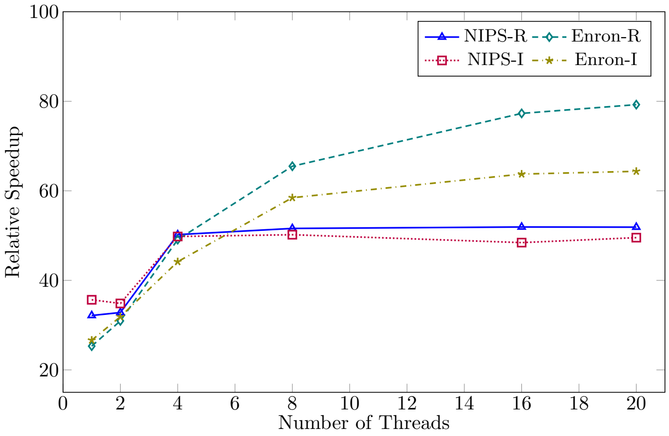

Fig. 3. Speedup relative to LDA-C. We assume that the baseline operates at 1x speed. We run all implementations on two datasets – NIPS and Enron, and two machines – rlogin and ironclaw1.

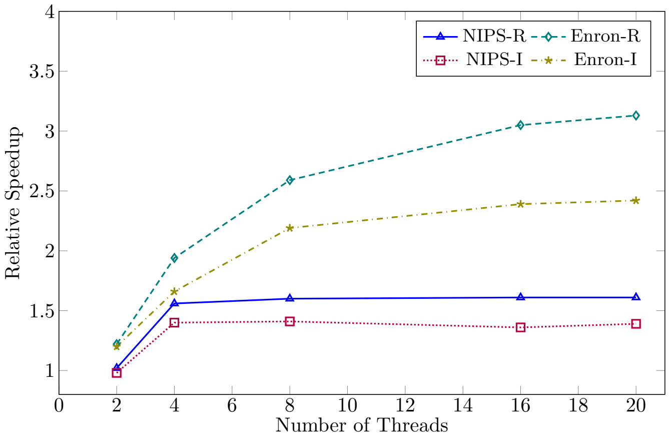

Fig. 4. Speedup relative to LDA-SCVB0 (1 thread). We run all implementations on two datasets – NIPS and Enron, and two machines – rlogin and ironclaw1.

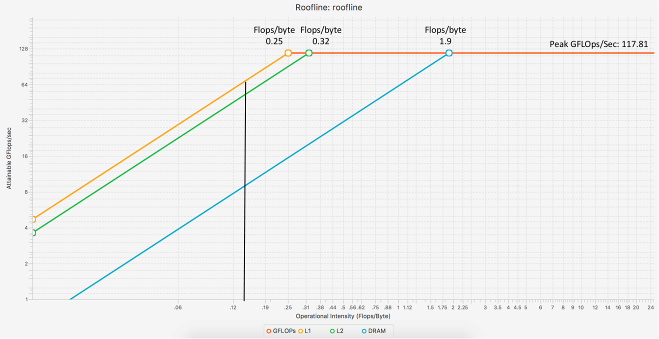

Fig. 5. Roofline model for “rlogin”

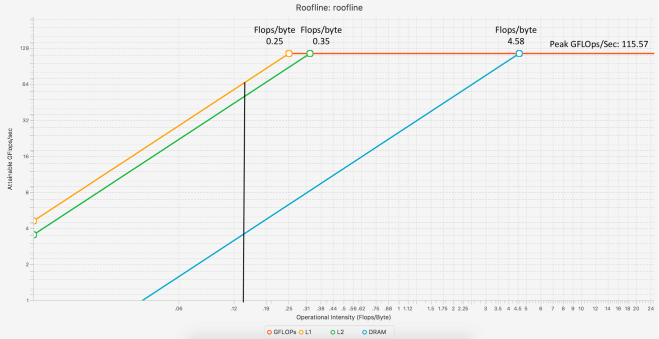

Fig. 6. Roofline model for “ironclaw1”

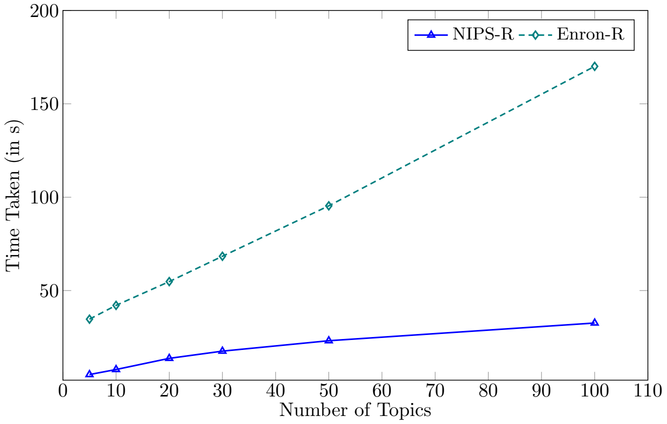

Fig. 7. Strong scaling of LDA-SCVB0 (4 threads) as we increase the number of topics. The results were obtained for 100 iterations on rlogin.

## V. Project Timeline

We implemented the project in multiple phases. Table IV gives details about the persons responsible for each stage of the project along with the expected completion dates.

TABLE IV
Project plan and list of deliverables

| **Deliverable** | **Person** | **Date** |
|---|---|---|
| Single-threaded implementation | Ashish | 04/12/18 |
| OpenMP implementation | Ashish & Vartan | 04/16/18 |
| Roofline plotting | Vartan | 04/25/18 |
| Final report | Ashish & Vartan | 04/30/18 |

## VI. Conclusion

Through the project, we have achieved the following:

1) Built a robust parallelized version of latent Dirichlet allocation based on stochastic collapsed variational Bayesian inference (SCVB0).
    a) Compare the speedup against the single-threaded Gibbs sampling version of LDA
    b) Compared the speedup obtained with multiple threads against the single-threaded version of this algorithm
    c) Compared the scaling for different data as we change the computational intensity of the algorithm,in this case the number of topics, vs the time.
2) Characterized the algorithm on the Roofline model to determine it is more memory-bound than CPU-bound.

## VII. Future Work

The algorithm *can* also be implemented on OpenACC. But, from our observation, this will require significant refactoring to run on the GPU. There exists only one implementation of LDA on the GPU based on GibbsLDA++. We feel this is a potential area of interest as a future project.

The same algorithm can be implemented in Python and NumPy, and can be potentially open-sourced so that people can readily use it. If useful, it can be even be incorporated into other open-source libraries such as Gensim[9].

## Acknowledgment

The authors would like to thank Dr. Wu-chen Feng for his support in the course Advanced Parallel Computation. They would also like to acknowledge Dr. Chandan K. Reddy for his ideas on parallel computation projects in data mining. Finally, a quick shout-out to the course TAs, Brandon Cobbs and James Taylor for their assistance with infrastructure – the rlogin and ironclaw clusters.

## 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] 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.

[3] M. Hoffman, F. R. Bach, and D. M. Blei, "Online learning for latent dirichlet allocation," in *advances in neural information processing systems*, 2010, pp. 856–864.

[4] R. Baeza-Yates, B. Ribeiro-Neto *et al.*, *Modern information retrieval*. ACM press New York, 1999, vol. 463.

[5] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, p. 391, 1990.

[6] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 703–710, 2010.

[7] J. Foulds, L. Boyles, C. DuBois, P. Smyth, and M. Welling, "Stochastic collapsed variational bayesian inference for latent dirichlet allocation," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 446–454.

[8] "Berkeley lab roofline toolkit," 2018.

[9] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498785

[10] Intel, "Intelr vtunetm amplifier xe," 2018.

## VIII. APPENDIX

The following equations are analytically discussed in [7]. Furthermore, they are utilized in Algorithm 2. Equation 2 refers to the update of the variational distribution for the ith distinct word of document j in topic k. In Equation 3, we update $N_j^\Theta$ with an online average of the current value and its expected value. Equation 4 and 5 are the updates of the expected $N^\Phi$ and $N^Z$ respectively, after observing a minibatch M, which is the average of the per-token estimates, where $N^\Phi = \frac{C}{|M|}\sum_{ij} \in MY^{(ij)}$ and $N^Z = \frac{C}{|M|}\sum_{ij} \in M\gamma^{(ij)}$

$$\gamma_{ijk} :\propto \frac{N^\Phi_{w_{ij}k} + \eta_{w_{ij}}}{N^Z_k + \sum_w \eta_w}\left(N^\Theta_{jk} + \alpha\right) \tag{3}$$

[9] https://github.com/RaRe-Technologies/gensim

$$N_j^{\Theta} := \left(1 - \rho_t^{\Theta}\right) N_j^{\Theta} + \rho_t^{\Theta} C_j \gamma_{ij} \tag{4}$$

$$N^{\Phi} := \left(1 - \rho_t^{\Phi}\right) N_j^{\Phi} + \rho_t^{\Phi} \hat{N}^{\Phi} \tag{5}$$

$$N^{Z} := \left(1 - \rho_t^{\Phi}\right) N_j^{Z} + \rho_t^{\Phi} \hat{N}^{Z} \tag{6}$$