Simple, Correct Parallelization for Blocked Gibbs Sampling

Graham Neubig

November 16, 2014

Abstract

We present a method for distributing collapsed Gibbs sampling over multiple processors that is simple, statistically correct, and memory efficient. The method uses blocked sampling, dividing the training data into relatively large sized blocks, and distributing the sampling of each block over multiple processors. At the end of each parallel run, Metropolis-Hastings rejection sampling is performed to ensure that samples are being drawn from the correct distribution. Empirical results on part-of-speech tagging and word segmentation tasks show that the proposed blocked sampling method can sample from the true distribution while achieving convergence speed comparable with previous parallel sampling methods.[1]

1 Introduction

Bayesian techniques are increasingly being applied to large-scale learning problems, a result of advances in both modeling and inference techniques. As increases in speed of individual processors have not been able to match the increase in data sizes, many have turned to parallelism as a means of performing processing on large data sets.

However, for collapsed Gibbs sampling, one of the major methods for Bayesian inference, a general framework for statistically correct parallelization has proven elusive. As opposed to explicit Gibbs sampling, where parameters are treated as normal variables to be sampled, collapsed Gibbs sampling analytically integrates out the parameters, calculating probabilities directly from the configurations of the latent variables [1]. Collapsed sampling has a number of practical advantages such as eliminating the explicit parameter sampling step and allowing for the simple handling of non-parametric distributions with an effectively infinite number of parameters. However, every time a variable is resampled, the probability distribution over parameters changes, inducing complex dependencies between all of the hidden variables in the training data, causing problems for parallelization.

[1] The software is available as the pgibbs toolkit at `http://www.phontron.com/pgibbs`.

Previous research on parallelizing collapsed Gibbs samplers has run multiple samplers in parallel, combining their results at the end of every iteration [2, 3]. This amounts to ignoring the dependencies between variables sampled by different cores until the end of the sampling run, and is therefore not guranteed to sample from the proper distribution, although losses have been empirically shown to be small for latent variable models when an appropriate choice of a combination heuristic is made [2, 4]. Another thread of research analyzes the particular model under consideration and divides variables into dependency sets, dispatching variables from different dependency sets to different processors [5, 6, 7]. This method is guaranteed to sample from the proper distribution, but requires a detailed analysis of each model and cannot be used with models where these independence assumptions cannot be made.

In this paper, we propose a method for parallelizing collapsed Gibbs sampling that makes no model assumptions, and is both statistically correct and memory efficient. Inference is achieved through the use of block sampling [8], splitting variables into relatively large sized blocks, and distributing the sampling of variables in each block over multiple processors. In particular, we focus on tasks where the majority of the computation burden occurs during the sampling of new values, and thus this framework allows for the most cumbersome computation to be split between multiple processors. To ensure that dependencies between the sampled variables are properly considered, Metropolis-Hastings rejection sampling is performed at the end of each block. In addition, while when using multiple samplers it is necessary to make a separate copy of all variable configurations for each processor, in the proposed method a single copy can be used by all processors, greatly reducing memory use.

We evaluate the proposed method on two natural language processing tasks. The first is unsupervised part of speech (POS) induction using Bayesian hidden Markov models (HMMs), a parametric clustering task [9]. The second is unsupervised word segmentation (WS) using the hierarchical Pitman-Yor language model (HPYLM, [10]), a non-parametric structured prediction task [11]. In our experiments, we find that the proposed method allows for simple, correct, and memory efficient sampling on multi-core systems with comparable computation time traditional parallelization methods.

2 Gibbs Sampling

In the Bayesian learning of probabilistic models, we assume that we have a set of observed variables X, and want to learn hidden variables Z. Z can be decomposed into two sets: latent variables Y, which are dependent only on a small subset of X, and parameters θ, which indicate model probabilities that are dependent on a large number of variables in both X and Y. Standard methods used to learn models in this framework include variational Bayes (VB) and Gibbs sampling. In this work, we focus on Gibbs sampling [12], which has proven popular for a number of tasks due to its flexibility and ease of implementation.

Traditional Gibbs sampling approximates the distribution $P(Z)$ by going through the variables and sampling each z_i one at a time according to the posterior distribution over z_i given all the other variables, both hidden and observed

$$z_i \sim P(z_i|X, Z\backslash z_i) \tag{1}$$

where $Z\backslash z_i$ indicates the set Z with z_i removed.

In the previously described method, both θ and Y are treated in the same fashion, sampled once every iteration. In contrast, in many cases it is possible to explicitly integrate out the model parameters θ, and sample directly from the marginal distribution for y_i

$$y_i \sim \int P(y_i|\theta, X, Y\backslash y_i)P(\theta|X, Y\backslash y_i)d\theta. \tag{2}$$

Samplers in which parameters are integrated out analytically are called collapsed samplers [1]. Collapsed sampling has been used in learning a wide variety of models, and is particularly useful for non-parametric Bayesian models which have a potentially infinite number of parameters, and are thus not straightforward to sample explicitly.

As it is not practical to consider every variable in Y when taking a new sample, most models store the sufficient statistics that must be used to calculate these probabilities. For simpler discrete models such as multinomial distributions with Dirichlet priors, the only sufficient statistics that must be stored are occurrence counts $c(z_t)$. However, in more expressive models such as hierarchical models [13], or models using Pitman-Yor processes [14], it is often necessary to keep track of other variables such as the table configurations of the Chinese Restaurant Process.

3 Multi-Sampler Parallelization

While Gibbs sampling is generally more flexible and easier to implement than other methods such as variational Bayes, it also has the disadvantage of being relatively slow. There have been a number of works that have examined the possibility of performing collapsed sampling in parallel on multiple processors [2, 3] to increase the speed of the training process.

When given cores $C = \{c_1, \ldots, c_J\}$ these approaches perform a single iteration of parallel sampling according to the following process.

1. Create J copies of the current configuration of Y or the sufficient statistics that summarize Y, and distribute them to the cores.

2. Divide Y into J partitions, and notify core c_j of the values $\boldsymbol{y}^{(j)}$ that it is responsible for.

3. In parallel, for each core c_j, perform a single run of traditional Gibbs sampling

4. When each core has finished its sampling, merge new variables for each partition $\boldsymbol{y}^{(j)}$ into a new copy of Y.

5. Re-calculate the sufficient statistics according to Y.

This method provides a relatively straight-forward approach to parallelizing sampling, but also has a number of hidden difficulties.

First, as each sampler independently modifies the sufficient statistics, a copy of the statistics must be made for each running thread. As statistics trained on large data will naturally also be large, making additional copies of the statistics results in a significant memory burden. In addition, merging samples and calculating new sufficient statistics for models is not always an entirely straightforward process. When the only necessary statistics are occurrence counts, these can be merged by simply taking the sum over all partitions. However, in the case of table configurations, the only principled method is to first calculate and combine the counts, then to re-sample the table distributions based on these counts. This requires an extra pass through the data that adds significant time to the training process.

Finally, the running of multiple Gibbs samplers in parallel is not mathematically correct. While sampling from the distribution $P(y_i|X, Y\backslash y_i)$, it is necessary to condition on all variables except for y_i. When running multiple Gibbs samplers in parallel, updates of the sufficient statistics made on other cores will not be reflected in the sufficient statistics until the merge in Step 5 of the above process. As a result, the majority of the samples are taken from an out-dated approximation of the sufficient statistics that would actually be calculated from $Y\backslash y_i$. This has been shown to have a negative effect on the learning of topic models, although negative effects can be ameliorated to some extent through combination heuristics [2].

4 Blocked Parallelization

In this section, we propose a parallelization technique based on blocked sampling that overcomes the issues mentioned in the previous section. We first give a brief summary of blocked sampling, then describe the proposed method and compare it with the traditional method for parallelization.

4.1 Blocked Sampling

In block sampling, we first separate Y into disjoint subsets (blocks) $\mathcal{B} = \boldsymbol{b}_1, \ldots, \boldsymbol{b}_L$ such that

$$Y = \cup_{l=1}^{L} \boldsymbol{b}_l. \tag{3}$$

While in traditional Gibbs sampling each sample considers a single variable y_i from the distribution $P(y_i|X, Y\backslash y_i)$, in blocked sampling, we sample the configuration of all parameters in the block at a single time

$$\boldsymbol{b}_l \sim P(\boldsymbol{b}_l|Y\backslash \boldsymbol{b}_l, X). \tag{4}$$

In previous work on a number of tasks [15, 11, 16], blocked sampling has proven a more efficient way to sample from distributions where blocks of variables are highly related. Taking the Bayesian learning of HMMs for POS induction (defined in detail in section 5.1) as an example, all POS tags from a single sentence can be grouped together in a single block, and a version of the forward-backward algorithm can be used to sample all variables in the block effectively [9, 17].

However, while block sampling algorithms based on dynamic programming can be used to effectively approximate the true distribution, there are often still some interactions between variables in the blocks that they cannot account for effectively. In the Bayesian HMM example, if a single sentence contains two identical words that are not adjacent to each-other, the "rich-gets-richer" effect intrinsic in Bayesian models will indicate that these two words are likely to be assigned to the same tag. However, if this information cannot be efficiently integrated into the dynamic programming algorithm, as it will be necessary to consider dependencies between non-neighboring tags.

A method for performing Gibbs sampling even in the case that all interactions between variables in the block cannot be accounted for has been proposed by [15]. We note that the sample candidate is first drawn from the approximate distribution that can be calculated efficiently

$$Q(\boldsymbol{b}_l|X, Y\backslash\boldsymbol{b}_l). \tag{5}$$

In the case of the blocked sampling we consider here, this approximate distribution is similar to the true distribution, but ignores the interactions between the variables in block $\boldsymbol{b}_l$. Next, the Metropolis-Hastings method [18] is used to calculate the probability that the new sample is accepted according to probability

$$\alpha = \min(1, \frac{P(\boldsymbol{b}_{l,new}|X, Y\backslash\boldsymbol{b}_l)Q(\boldsymbol{b}_{l,old}|X, Y\backslash\boldsymbol{b}_l)}{P(\boldsymbol{b}_{l,old}|X, Y\backslash\boldsymbol{b}_l)Q(\boldsymbol{b}_{l,new}|X, Y\backslash\boldsymbol{b}_l)}). \tag{6}$$

This step compensates for the difference between the proposal and true distributions, and thus we are ensured that we are correctly sampling from the distribution that we are interested in.

4.2 Block Sampling for Multiple Cores

In this work, we present a method for parallelizing sampling based on this block sampling algorithm. The key to the proposed method is that, as long as we perform the Metropolis-Hastings rejection step similar to that of Equation (8), we are free to choose any proposal distribution Q that we would like. In the particular case described in the previous section, this was a distribution that ignored the interactions between the variables in block $\boldsymbol{b}_l$.

However, it is also possible to think about a proposal distribution that makes an independence assumption between a larger subset $B \in \mathcal{B}$ where $B = \{b_1, \ldots, b_k\}$

$$Q(\boldsymbol{b}_l|X, Y\backslash B) \tag{7}$$

where $b_l \in B$. If $b_l \neq B$, the proposal distribution will by different than that of Equation (5), but if Y is sufficiently large, we can expect the model to be reasonably well trained without the influence of B. As a result, the samples generated by both distributions will be similar, and thus we can substitute Equation (7) for Equation (5) in the sampling process.

Finally, to allow for parallel sampling based on this new distribution, we note that all of the sub-blocks $\{b_1, \ldots, b_k\}$ can be simultaneously sampled according to Equation (7). This is done according to the following process.

1. Given the old sufficient statistics Y_{old}, remove all counts resulting from block B to acquire new sufficient statistics $Y \backslash B$.

2. Assign each sub-block in $b_l \in \{b_1, \ldots, b_k\}$ to a core c_j, and use this core to acquire a new sample $b_{l,new}$ according to $Q(\boldsymbol{b}_{l,new}|X, Y \backslash B)$. At the same time, calculate the proposal probability $Q(\boldsymbol{b}_{l,old}|X, Y \backslash B)$.

3. Gather all these new samples into $B_{new} = \{b_{1,new}, \ldots, b_{k,new}\}$ for processing on a single processor.

4. Replace B_{old} into the sufficient statistics $Y \backslash B$, so we recover Y_{old}, which we will use as the starting point of our Metropolis-Hastings pass. We will use Y to represent the current configuration of the latent variables during the Metropolis-Hastings pass, so we initially set $Y \leftarrow Y_{old}$.

5. For each block b_l in B:

   (a) Given the current sufficient statistics Y, remove counts resulting from $b_{l,old}$.

   (b) Perform a Metropolis-Hastings rejection sampling step using the current sufficient statistics to calculate the true distribution, and the proposal probabilities calculated in parallel during step 2:

   $$\alpha = \min(1, \frac{P(\boldsymbol{b}_{l,new}|X, Y \backslash \boldsymbol{b}_{l,old}) Q(\boldsymbol{b}_{l,old}|X, Y \backslash B)}{P(\boldsymbol{b}_{l,old}|X, Y \backslash \boldsymbol{b}_{l,old}) Q(\boldsymbol{b}_{l,new}|X, Y \backslash B)}). \tag{8}$$

   (c) If the result is "accept," merge $\boldsymbol{b}_{l,new}$ into Y, and if it is "reject," merge $\boldsymbol{b}_{l,old}$ into Y.

By repeating this process for all blocks in the corpus for several iterations, we can perform Gibbs sampling over the entirety of the training data.

This blocked parallelization method overcomes the three difficulties of the multi-sampler method mentioned in the previous section. It is mathematically correct, as the approximation due to dividing variables into blocks is compensated for by the Metropolis-Hastings step run at the end of every block[2]. There

[2]It should be noted that [4] have proposed a method for correct Metropolis-Hastings sampling within the multi-sampler framework, but it was found to be significantly slower than inexact sampling. In addition, it requires synchronization of random seeds between processors, rewinding of the proposal process and other additional steps not necessary in traditional samplers.

is also no need to perform an additional sweep over the data at the end of every iteration, as the variables are updated correctly block-by-block. Finally, as model probabilities are not modified during each block sampling step, the sufficient statistics for every core can be held in shared memory, removing the need to make new copy of the sufficient statistics for every core if processors have access to a common memory.

It should be noted that the proposed method does require the choice of an appropriate block size. If the size of B is too large, it is possible that the proposal distribution will diverge too much from the true distribution, resulting in higher rejection rates. We examine the tradeoff between block size and rejection rates empirically in Section 6.

5 Bayesian Models

This section describes the two Bayesian learning tasks on which we tested our models: part-of-speech estimation and word segmentation. We chose these tasks specifically as they are representatives for two broader groups of tasks: clustering (which also includes topic models [19]), and structure prediction (which also includes unsupervised parsing [15] and word alignment [3]).

5.1 Part-of-Speech Estimation

The first task is unsupervised part of speech (POS) estimation. X is a corpus of word strings, and Y is a list of latent classes associated POS sequences. As a model, we use a Bayesian version of the first-order hidden Markov model (HMM) similar to that described in [17], where each word x_i is emitted by a hidden state y_i, with y_i corresponding roughly to parts of speech.

The probabilities are parameterized with multinomial transition and emission distributions ($\boldsymbol{\theta}_{T,y_i}$ and $\boldsymbol{\theta}_{E,y_i}$) respectively. We give these parameters a Pitman-Yor process prior [14]:

$$\boldsymbol{\theta}_{T,y_i} \sim PY(d_T, s_T, P_{base,T}) \tag{9}$$

$$\boldsymbol{\theta}_{E,y_i} \sim PY(d_E, s_E, P_{base,E}). \tag{10}$$

The hyperparameters d and s are given weak priors and sampled according to the auxiliary variable method described in [10]. $P_{base,T}$ is a uniform distribution over all states, the number of which we fixed to 30. $P_{base,E}$ is a uniform distribution over all words in the corpus. Sampling for each sentence is performed using the dynamic programming technique described by [9].

5.2 Word Segmentation

In unsupervised word segmentation (WS), X is a corpus of unsegmented character strings, and Y is a corpus of segmented word strings. We follow [11] in

defining a hierarchical Pitman-Yor language model (HPYLM) over the word strings in Y.

$$\boldsymbol{\theta}_{LM} \sim HPYLM(\boldsymbol{d}, \boldsymbol{s}, P_{base}). \tag{11}$$

All our experiments use a bigram model and omit the nested unknown word model for P_{base}, instead replacing it with a simple model that chooses word length l from an exponential distribution with an average length of 2, then chooses characters from a uniform distribution over all characters. Sampling is performed using the forward-filtering/backward-sampling algorithm described by [11], and hyperparameters are sampled according to the auxiliary variable method of [10].

6 Experiments

In this section, we present experiments comparing the proposed method with the multi-sampler method. We performed all of our experiments on data from the Chinese Treebank version 5.0 [3].

Five independent 1000 iteration runs of the sampler were performed for all settings, and results over these five runs were averaged. The order of the sentences to be sampled was shuffled after every iteration. The likelihood results are all calculated with cross validation, leaving one sentence out of the model and calculating the probability of that sentence according to the probability of the true distribution given the variable configurations in all other sentences.

6.1 Effect of Parallelization on Convergence Speed

Our first experiments examine the block-based and multi-sampler methods, and note the interaction between block size or number of threads and the convergence properties of the sampler. First, Figure 1 demonstrates the likelihood achieved after 2000 iterations of sampling by the two methods. It can be seen that in general, the likelihoods after 2000 iterations are comparable with both methods over both tasks, indicating that both methods will achieve similar likelihoods when given enough time. There is, however, a slight but consistent downward trend as the number of threads are increased with the multi-sampler method.

Thus, the next natural question to ask is how much time it takes for samplers using both methods to achieve these results. We first examine this by noting the number of iterations it takes for samplers of each type to converge. As there is no simple way to measure whether a Gibbs sampler has mixed or not, we instead use the likelihood at the 2000th iteration as a proxy for the likelihood of the converged chain. As a measure of convergence speed, we count the number of iterations that are required to achieve a likelihood within 1% of the likelihood on the 2000th iteration.

The results in Figure 2 show the results of the experiments. It can be seen that for the HMM model, there is no clear trend in the multi-sampler or blocked

[3] Available from the Linguistic Data Consortium, LDC Catalog Number LDC2005T01.

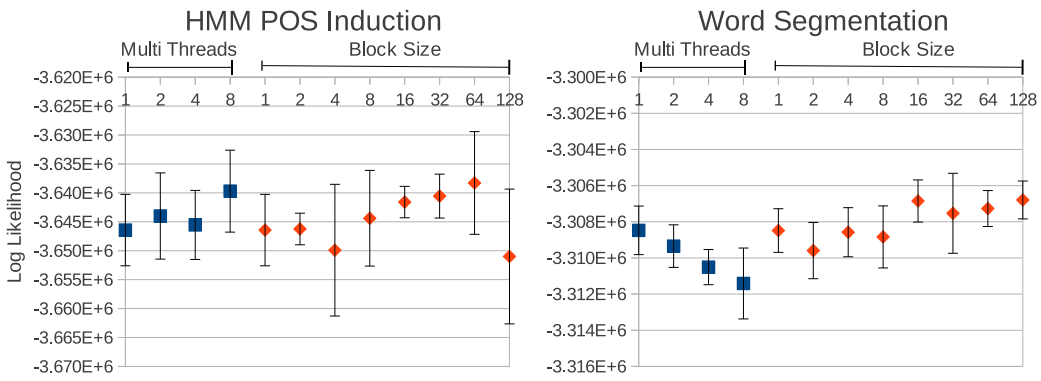

Figure 1: Likelihood for HMM and WS after 2000 iterations for multi-sampler and blocked parallelization. Error bars indicate the standard deviation over five independent runs.

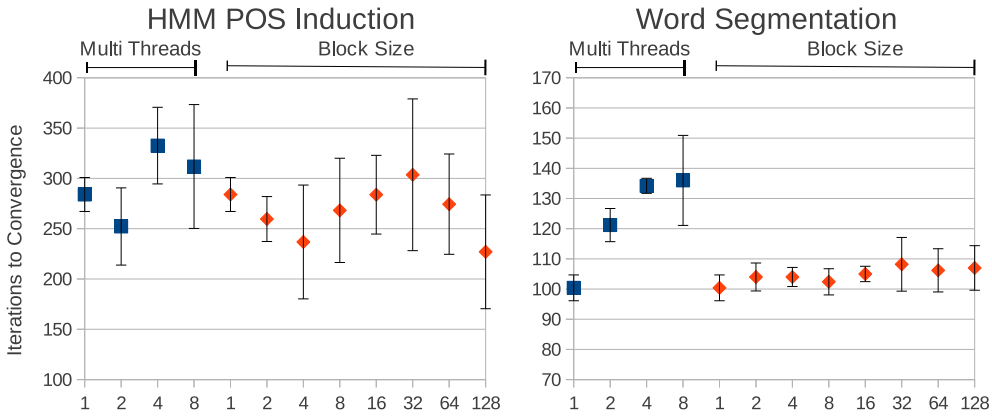

Figure 2: The number of iterations required to converge to a solution that has a likelihood within 1% of that of the solution after 2000 iterations.

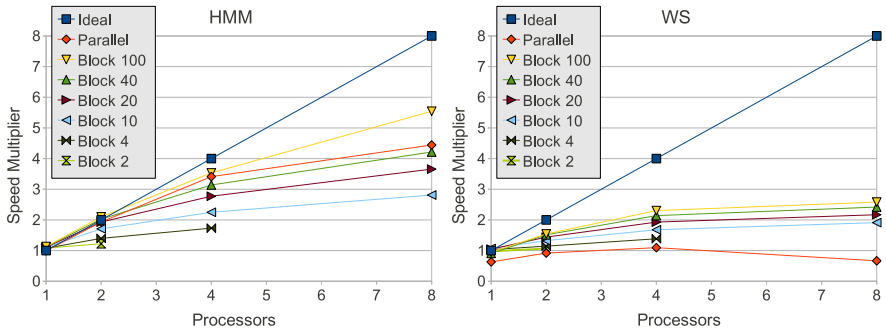

Figure 3: The speed increase provided by different methods for distributed sampling for the HMM and WS models.

sampling methods changing convergence times. However, for the WS model, it can be seen that the multi-sampler method clearly and significantly increases the amount of time required to converge to a good solution, with the 8-threaded model taking 136 iterations compared to 100 iterations for a traditional Gibbs sampler. On the other hand, the proposed blocked sampling method is generally comparable to standard Gibbs sampling, even with larger block sizes.

6.2 Time Efficiency of Distributed Sampling

We also ran experiments to test the convergence speed of the two sampling methods in a shared-memory distributed environment with 1, 2, 4, or 8 parallel 2.93 GHz Xeon processors. Figure 3 shows the increases in speed averaged over the last 100 iterations of training for block-based and parallel sampling using the HMM and WS models.

It can be seen that as block size gets larger, efficiency improves, particularly for larger numbers of processors. This is because larger block sizes require fewer overall jobs be dispatched, reducing overhead. Overall, the speed improvement for the HMM is significantly larger than for WS as the distributed sampling phase takes a larger portion of the time compared to the non-distributed Metropolis-Hastings and model update steps.

The multi-sampler method is relatively efficient for the HMM, achieving speeds comparable to block sampling with a block size of 40 to 100 sentences. However, for WS parallel sampling provides only modest gains or even decreases in iteration speed, as the extra time required to copy and merge the larger 2-gram HPYLM model outweighs the gains achieved by parallelizing sampling.

Finally, we show results for the total amount of time required for the model to converge to a likelihood within 1% of the likelihood value at the end of 2000 iterations of training. In Figure 4 it can be seen that the block sampling method converges slightly slower than parallel sampling for the HMM, and at a similar speed for the WS method. For blocked sampling, choosing the correct

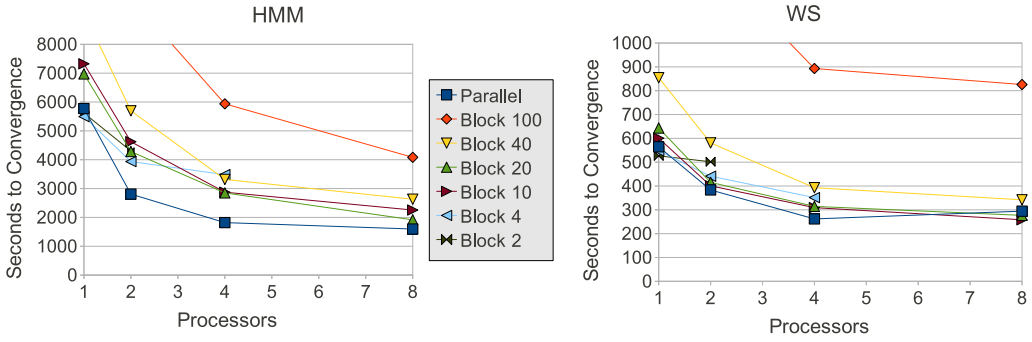

Figure 4: The time required for convergence when using various sampling techniques and processor counts.

block size showed to have some effect on the convergence speed. Block sizes approximately twice as large as the number of threads achieved the best results in this particular situation.

On the other hand, block sampling used significantly less memory than parallel sampling for both tasks. When 8 threads were used, parallel sampling used 8 copies of the statistics for the child processes, and 1 copy of the statistics for the parent process, resulting in a total of 9.22 times more memory used for WS (1140M vs. 123M), and 9.06 times more memory used for the HMM (282M vs. 31M). Considering the comparable speed, superior memory usage, and statistical correctness of the blocked method, it appears to be an attractive alternative to the traditional multi-sampler approach.

6.3 Effect of the Metropolis-Hastings Step

The results presented in the previous section indicate that the high rejection rates cause slower learning in the blocked method. Thus, it is natural to ask what effect omitting the rejection step will have on the sampling process. Figure 5 shows likelihood results for when the Metropolis-Hastings step is omitted from the training.

Comparing these results to when the Metropolis-Hastings step is performed (Figure **??**), it can be seen that omitting the Metropolis-Hastings step generally *improves* convergence speed and final likelihood results when only a single sentence is used as a block. The reason for this improvement lies in the fact that rejection of an entire sentence affects an unnecessarily large area. Many rejections are caused by sentences that contain two identical words that should be given consistent interpretations, while this fact is not considered by the proposal distribution. However, most of the other non-identical words in the rejected sentence have been sampled properly, and thus throwing them out causes an overall loss in the sampling efficiency.

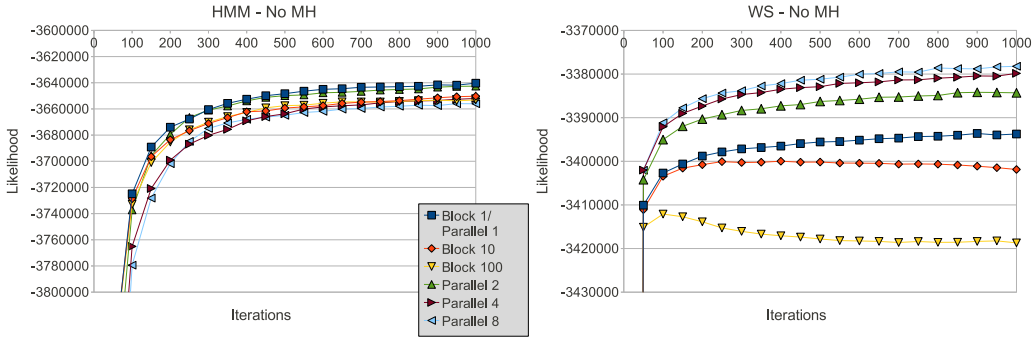

Figure 5: Likelihood curves for the HMM and WS when the Metropolis-Hastings step is not performed.

For the HMM model, this resulted in the blocked sampling method with no MH step achieving the fastest convergence, with block size 100 and 8 threads converging in 752 seconds, compared to 1042 seconds for the multi-sampler method with 8 threads and no MH step. However, for the WS model, it can be seen that the likelihood initially increases, but then drops significantly before leveling out at a relatively low value. This is due to the fact that when all instances of a particular word occur only in a single block, the count of the word will be reduced to zero, effectively making it an unknown word. As the model has a strong bias against initially adding unknown words, there is a good chance that lost words will not be re-added to the vocabulary for many samples, even if their inclusion in the vocabulary is motivated by the model probabilities. When using MH, on the other hand, samples that remove all instances of a word that occurs multiple times will tend to be rejected, preventing this sort of over-aggressive removal.

7 Conclusion and Discussion

We presented a method for distributing Gibbs sampling over multiple processors using block sampling. The two major advantages of the proposed method over existing methods are that it is able to sample from the correct probability distribution in a competitive amount of time, and that it uses significantly less memory than the multi-sampler method. In addition, we showed that different parallelization methods show very different convergence properties depending on the model used. Using multiple samplers generally improves results for unsupervised word segmentation, and skipping the Metropolis-Hastings step improves results for learning HMMs.

While we performed experiments in a shared memory environment, it is not difficult to expand to parallel systems that do not share a common memory. Like

other parallel sampling methods, which communicate the changes to models at the end of a full sampling iteration, it is simply necessary to communicate the model changes and sampling results at the end of each block.

Future work in this area includes the development of improved proposal distributions to help reduce the rejection rate when larger blocks are used. While it is not possible to consider interactions between variables processed on different cores, it is likely possible to consider the interactions between variables processed on the same core to develop a proposal distribution that comes closer to the true distribution.

References

[1] Jun S. Liu. The collapsed Gibbs sampler in Bayesian computations with applications to a gene regulation problem. *Journal of the American Statistical Association*, 89(427), 1994.

[2] David Newman, Arthur Asuncion, Padhraic Smyth, and Max Welling. Distributed algorithms for topic models. *Journal of Machine Learning Research*, 10, 2009.

[3] Phil Blunsom, Trevor Cohn, Chris Dyer, and Miles Osborne. A Gibbs sampler for phrasal synchronous grammar induction. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 782–790, 2009.

[4] Finale Doshi-Velez, David Knowles, Shakir Mohamed, and Zoubin Ghahramani. Large scale nonparametric Bayesian inference: Data parallelisation in the Indian buffet process. *Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS)*, 22, 2010.

[5] Feng Yan, Ningyi Xu, and Yuan Qi. Parallel inference for latent Dirichlet allocation on graphics processing units. In *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS)*, 2009.

[6] Songfang Huang and Steve Renals. A parallel training algorithm for hierarchical Pitman-Yor process language models. In *Proceedings of the 10th Annual Conference of the International Speech Communication Association (InterSpeech)*, 2009.

[7] Joseph Gonzalez, Yucheng Low, Arthur Gretton, and Carlos Guestrin. Parallel gibbs sampling: From colored fields to thin junction trees. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 324–332, 2011.

[8] Claus S. Jensen, Uffe Kjærulff, and Augustine Kong. Blocking Gibbs sampling in very large probabilistic expert systems. *International Journal of Human Computer Studies*, 42(6), 1995.

[9] Steven L. Scott. Bayesian methods for hidden Markov models: Recursive computing in the 21st century. *Journal of the American Statistical Association*, 97(457), 2002.

[10] Yee Whye Teh. A Bayesian interpretation of interpolated Kneser-Ney. Technical report, School of Computing, National Univ. of Singapore, 2006.

[11] Daichi Mochihashi, Takeshi Yamada, and Naonori Ueda. Bayesian unsupervised word segmentation with nested Pitman-Yor modeling. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2009.

[12] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6), 1984.

[13] Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476), 2006.

[14] Jim Pitman and Marc Yor. The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, 25(2), 1997.

[15] Mark Johnson, Thomas Griffiths, and Sharon Goldwater. Bayesian inference for PCFGs via Markov chain Monte Carlo. In *Proceedings of the Human Language Technologies 2007: The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2007.

[16] Phil Blunsom and Trevor Cohn. Inducing synchronous grammars with slice sampling. In *Proceedings of the Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 238–241, 2010.

[17] Jianfeng Gao and Mark Johnson. A comparison of Bayesian estimators for unsupervised hidden Markov model POS taggers. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

[18] W. Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1), 1970.

[19] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3, 2003.