Architecture of an FPGA Accelerator for LDA-Based Inference

Taisuke Ono, Hasitha Muthumala Waidyasooriya and Masanori Hariyama
Graduate School of Information Sciences, Tohoku University
6-3-09, Aramaki-Aza-Aoba, Aoba, Sendai,
Miyagi 980-8579, Japan
Email: {ono52@dc., hasitha@, hariyama@}tohoku.ac.jp

Tsukasa Ishigaki
Graduate School of Economics and Management,
Tohoku University
27-1, Kawauchi, Aoba, Sendai,
Miyagi 980-8576, Japan
Email: isgk@econ.tohoku.ac.jp

***Abstract*—Latent Dirichlet allocation (LDA) based topic inference is a data classification method, that is used efficiently for extremely large data sets. However, the processing time is very large due to the serial computational behavior of the Markov Chain Monte Carlo method used for the topic inference. We propose a pipelined hardware architecture and memory allocation scheme to accelerate LDA using parallel processing. The proposed architecture is implemented on a reconfigurable hardware called FPGA (field programmable gate array), using OpenCL design environment. According to the experimental results, we achieved maximum speed-up of 2.38 times, while maintaining the same quality compared to the conventional CPU-based implementation.**

***Index Terms*—Latent Dirichlet allocation, Gibbs sampling, data classification, OpenCL for FPGA, machine learning.**

I. Introduction

Recently, big data are obtained in everywhere with the intention of contributing to the society in terms of decision making, problem solving, quality improving, etc. However, it is not possible for human to analyze such an enormous amount of data to make any meaningful contribution. Therefore, automatic classification of the data in to topics desired by human is necessary. Latent Dirichlet allocation (LDA) [1] based topic inference is one of such automatic classification method, that is used efficiently for extremely large data sets. LDA is used in many different areas such as image classification [2], customer data classification [3], etc.

Markov Chain Monte Carlo method (MCMC) [4], [5] is used for sampling in LDA based topic inference. The MCMC approach uses previous sample values to randomly generate the next sample value. Therefore, the processing time increases in proportion to the number of words, and the total processing time is very large. The computation is sequential so that it is difficult to increase the processing speed even using modern day multicore CPUs.

In this paper, we propose a hardware accelerator for LDA based topic inference to increase the processing speed. The proposed accelerator is based on an MCMC method called Gibbs sampling [6]. We use a pipelined approach to exploit the time-step level parallelism and to decrease the processing time. Memory allocation method is proposed to eliminate the data dependency of the variables and to design a pipelined accelerator. We use a field programmable gate array (FPGA) to implement the proposed accelerator. FPGA is an integrated circuit that users can reconfigure the circuit design, after manufacturing. OpenCL (Open Computing Language) based FPGA accelerator design method [7] is used for the proposed implementation. According to the results, the processing speed of the proposed accelerator is 2.38 times and the quality of the results is the same, compared to those of the LDA-based topic inference on a CPU. Much larger speed-up could be achieved by using an advanced FPGA board with a larger memory bandwidth.

II. Latent Dirichlet Allocation

Latent Dirichlet allocation (LDA) [1] is a generative statistical model that allows sets of observations to be explained by unobserved groups. For example, if observations are words in documents, each document is a mixture of a small number of topics. Variables in this topic model are as follows.

θ_d	The topic distribution in document d.
ϕ_k	The word distribution in topic k.
w_i	The i^{th} word.
d_i	The document of i^{th} word.
z_i	The topic of i^{th} word.
K	The number of topics.
D	The number of documents.
L	The total number of words.
W	The number of vocabulary words.

The process of generation is as follows.

1) Choose $\theta_d \sim Dir(\alpha)$ for each document.
2) Choose $\phi_k \sim Dir(\beta)$ for each topic.
3) For each i^{th} word in $w = \{w_1, ..., w_L\}$
   (a) Choose $z_i \sim Cat(\theta_{d_i})$.
   (b) Choose $w_i \sim Cat(\phi_{z_i})$.

The Dirichlet distribution with parameter α is denoted by Dir(α), and the categorical distribution with parameter θ_d is denoted by Cat(θ_d).

We can estimate θ and ϕ if we can calculate the conditional probability $P(z|w)$ of the topic assignment z_i of word $w = \{w_1, ..., w_L\}$. However, It is difficult to calculate this conditional probability directly. We use collapsed Gibbs sampling

(CGS) method [8] to generate this conditional probability. Eq.(1) is the conditional distribution of z_i. The $\neg i$ notation of a variable indicate that it does not contain the current topic assignment of w_i.

n_{wk}	The number of word w of topic k.
n_k	The number of words of topic k.
n_{dk}	The number of words of topic k in the document d.

$$P(z_i = k|\boldsymbol{z}^{-i}, \boldsymbol{w}) \propto \frac{n_{wk}^{-i} + \beta}{n_k^{-i} + W\beta} n_{dk}^{-i} + \alpha \tag{1}$$

Samples of topic assignment are given by the following process.

Step1: Assign a topic randomly to each word.
Step2: For all words, update the topic assignments based on Eq.(1).
Step3: Repeat Step2 for N times.

III. FPGA Implementation

A. OpenCL for FPGA

OpenCL is a framework to write programs to execute across heterogeneous parallel platforms [9]. It views a system as a number of computing devices (OpenCL devices) connected to a host. The host is usually a CPU while the devices can be any of OpenCL capable CPUs, GPUs, FPGAs, etc. The OpenCL for FPGA compiler is called “offline compiler”. Loops in a device code are implemented as pipelines on FPGA for parallel processing. Fig.1 show a loop that contains four kinds of operations, load, addition, subtraction and store. When this code is implemented on an FPGA, functional units are generated for computations, while registers are placed in between two functional units to hold result of the previous operation. Fig.2 shows the time-chart of pipeline processing. In the clock cycle t_0, A[0] and B[0] are loaded. In the clock cycle t_1, the addition operation for A[0] and B[0] is done and A[1] and B[1] are loaded simultaneously. In the clock cycle t_2, the subtraction operation is also done, after the clock cycle t_3, all four operations are done simultaneously for different data.

OpenCL based FPGA design uses a hierarchical memory structure that consists of global, constant, local and private memories. The global memory is the largest memory and it is place outside the FPGA. Therefore, we also call it the external memory. The latency of the global memory could be large as few hundred cycles. The memory access throughput can be changed due to the access patterns such as sequential, random, etc. The throughput of the global memory is usually large for read only or write only access, compared to read-write access. The local and private memories are implemented using block RAMs and registers inside the FPGA chip. As a result, the latency is very small and does not change with the memory access patterns. However, the memory size is extremely small. The constant memory is initialized on the global memory, but downloaded later to the local memory when the circuit is executed on the FPGA.

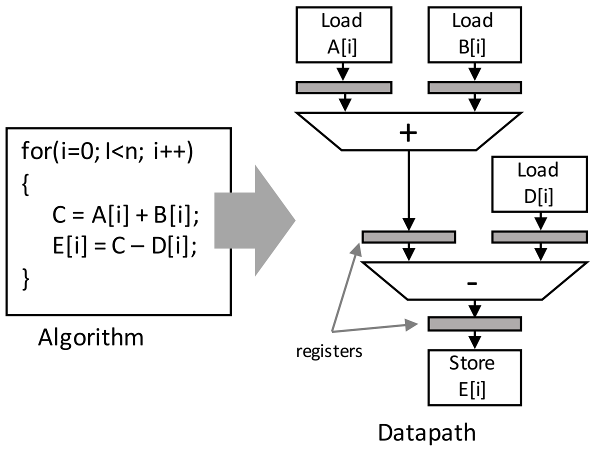


Fig. 1. Loop pipelining in OpenCL for FPGA.

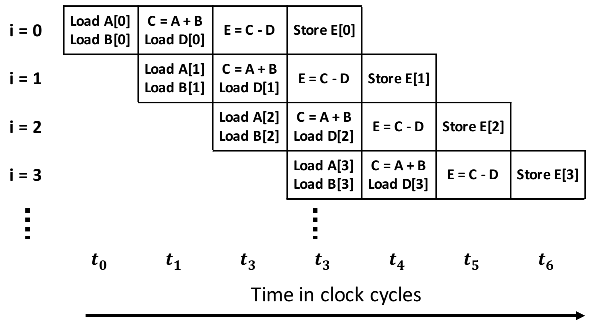


Fig. 2. Time chart of the computation in Fig.1.

```
for (int i = 0; i < L; i++)
{
  ushort w = word[i]
  ushort d = doc[i];

  old_topic = the topic of word w that belongs to
     the document d

  numWK[w][old_topic]--;
  numDK[d][old_topic]--;
  numK[old_topic]--;

  for (int k = 0; k < K; k++)
  {
    p[k] = (numDK[d][k]+alpha) *
      (numWK[w][k]+beta)/(numK[k] + sum_beta);

    if (k != 0)
      p[k] += p[k - 1];
  }

  //determine the new topic;
  ...

  numWK[w][new_topic]++;
  numDK[d][new_topic]++;
  numK[new_topic]++;
}
```

Listing 1. Computation of loop 2 of Fig.3.

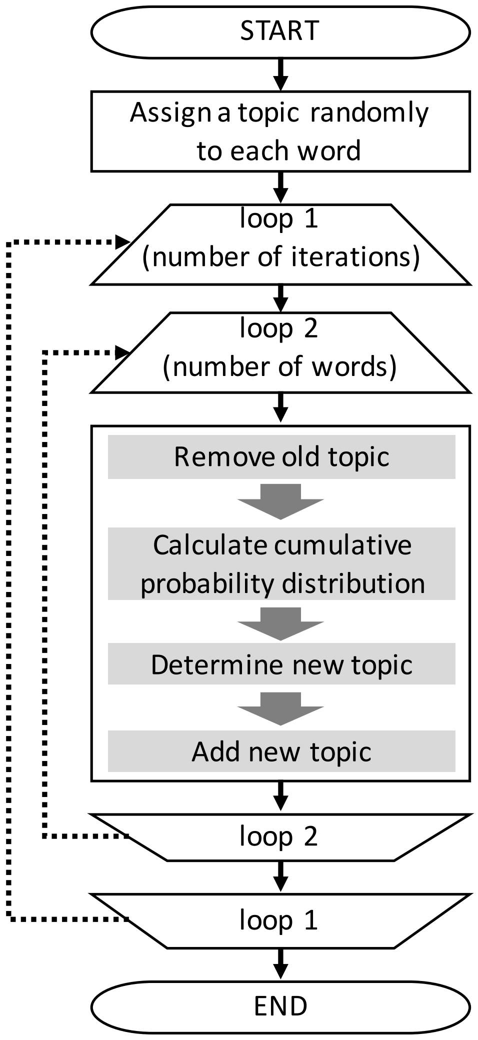

START
Assign a topic randomly to each word
loop 1 (number of iterations)
loop 2 (number of words)
Remove old topic
Calculate cumulative probability distribution
Determine new topic
Add new topic
loop 2
loop 1
END

Fig. 3. Tasks of the collapsed Gibbs sampling.

B. FPGA Implementation of collapsed Gibbs sampling (CGS)

Fig.3 shows the tasks of the conventional CGS algorithm. The random topic assignment is done only once and the topics are allocated after computing the probabilistic distribution. The topic allocation is done for all words and for many iterations. Computation of the two loops in Fig.3 requires the most processing time. Therefore, we allocated these two loops to the FPGA and the initial random topic assignment to the CPU.

Listing 1 shows the computation of loop 2 in Fig.3. Loop 2 runs for L iterations to infer topics for all words. All words are stored in `word` array, while the document number of each word is stored in `doc` array. The arrays `numWK`, `numDK` and `numK` are corresponds to n_{wk}, n_{dk} and n_k in Eq.(1) explained in section II. The number of vocabulary word w of each documents that are assign to topic k is stored in `numWK`, while the number of documents that the vocabulary word w is assigned to topic k is stored in `numDK`. Since there are many documents and each document contains many words, the sizes of `word`, `doc`, `numWK` and `numDK` arrays are larger than the size of the local memory in FPGA. Therefore, they should be stored in the global memory of the FPGA board. The number of words of topics k is stored in `numK`. When the number of topics are small, the size of `numK` array is also small and we can store it in the local (or private) memory of the FPGA.

Fig.4 shows the FPGA architecture based on the conventional CGS algorithm. In loop 2, a word from the `word` array, and its appropriate elements from `numWK` and `numDK` arrays are accessed form the global memory. The array `numK` is transferred from the global memory to the private memory at the beginning of the computation. Therefore, it is accessed from the private memory. After computing a new topic in FPGA, the `numWK` and `numDK` arrays are updated and write back to the global memory. The next word is accessed only after the updated values are stored in the global memory. If the next word is accessed before the updated values are stored and the next word is also the same as the previous word with the same topic, we may load the previous values of `numWK` array. Similarly, if the next word belongs to the same document and have the same topic, we may load the previous values of `numDK` array. This leads to incorrect results. As explained in section III-A, the latency of the global memory access is very large and changes according to the memory access pattern. Therefore, we cannot determine the latency of the global memory access. As a result, offline compiler cannot implement pipelines for loop 2, and the topic assignment is done for each word after the topic assignment of its previous word is completed. Such serial processing does not increase the processing speed.

To increase the processing speed, we have to implement pipelines for loop 2. For this purpose we propose a memory allocation scheme to remove the data dependency of the memory access. We allocate all words in to small sets, where two words w_x and w_y in the same set must be different ($w_x \neq w_y$) and also belongs to different documents ($d_x \neq d_y$). Fig.5 shows the memory allocation of the `word` and `doc` arrays. All the words in the same set are different, and belongs

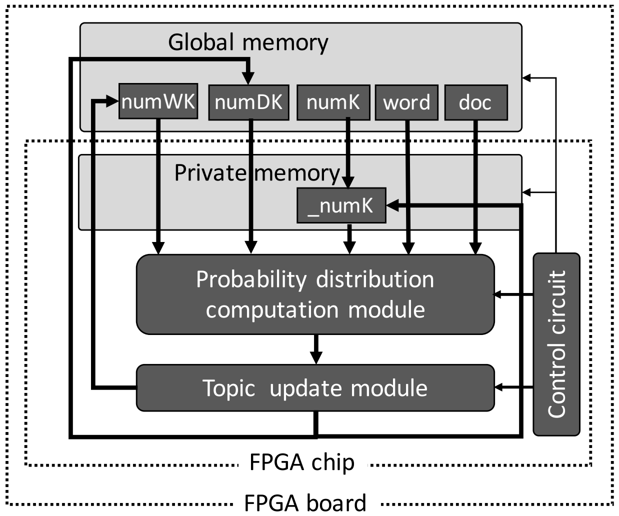
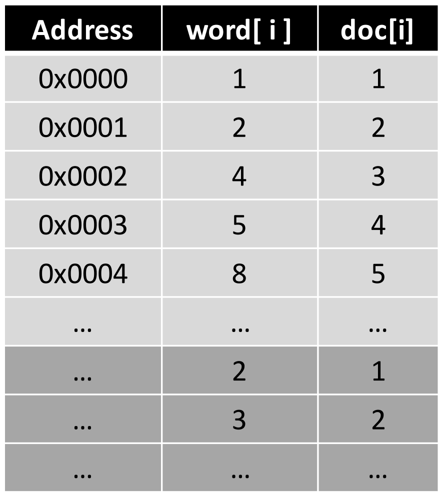
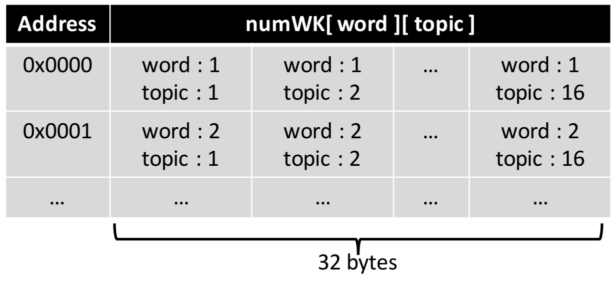


Fig. 4. FPGA architecture based on conventional CGS algorithm.



Address	word[i]	doc[i]
0x0000	1	1
0x0001	2	2
0x0002	4	3
0x0003	5	4
0x0004	8	5
…	…	…
…	2	1
…	3	2
…	…	…

Fig. 5. Memory allocation of `word` array and `doc` array.



Address	numWK[word][topic]			
0x0000	word : 1 topic : 1	word : 1 topic : 2	…	word : 1 topic : 16
0x0001	word : 2 topic : 1	word : 2 topic : 2	…	word : 2 topic : 16
…	…	…	…	…

Fig. 6. Memory allocation of `numWK` array.

to different documents. The allocation of words to sets is done on CPU, and the processing time required for this task is negligible compared to that of the whole computation.

Fig.6 shows the memory allocation of the array `numWK`. The allocation of the array `numDK` is also done similarly. The data belongs to all topics for a vocabulary word are allocated to consecutive memory locations, where each entry occupy 2 bytes. For 16 topics, each array entry of a word is 32 bytes long.

Listing 2 shows an extract of the OpenCL kernel code of the proposed accelerator architecture. Due to the serial nature of the CGS algorithm, we use single work-item kernel to implement the FPGA accelerator. Loop 2 is divided into two loops, one for the number of sets and the other is for the number of words in a set. The number of topics $K = 16$. For each word, we copy the data of all topics from the global memory to the private memory. The computation is done using the data of the private memory. The cumulative probability distribution is computed in parallel using loop unrolling. After the new topic is determined, the private memory and then the global memory are updated. However, the updated data are not required for the computation of any words in the same set. We use the directive `#pragma ivdep` to instruct the offline compiler that there is no data dependency between different words in the same set. As a result, offline compiler implement piplines for the innermost loop.

```
__kernel void cgs(
          global ushort* restrict word,
          global ushort* restrict doc,
          global ushort16* restrict numWK,
          global uint* restrict numDK,
          global ushort16* restrict numK,
          global uint* restrict pos_set_div,
          ... )
{
  ...
  for(int i = 0; i < iteration; i++) {
    for(int y = 0; y < num_sets; y++) {
      uint start = pos_set_div[y];
      uint end = pos_set_div[y+1];

      #pragma ivdep
      for(int x = start; x < end; x++) {
        ushort w = word[x]
        ushort d = doc[x];

        old_topic = the topic of w in set y,
                 which belongs to the document d

        ushort16_u _numWK, _numDK;
        _numWK.allTopics = numWK[w];
        _numDK.allTopics = numDK[d];

        #pragma unroll 16
        for(int k = 0; k < K; k++) {
          if(k == old_topic)
            p[k] = (_numDK.topic[k]-1 + alpha) *
            (_numWK.topic[k]-1 + beta) /
            (_numK[k]-1 + sum_beta);
          else
            p[k] = (_numDK.topic[k] + alpha) *
            (_numWK.topic[k] + beta) /
            (_numK[k] + sum_beta);
        }

        #pragma unroll
        for(int k = 1; k < K; k++)
          p[k] += p[k - 1];

        //determine the new topic;
        ...

        _numWK.topic[old_topic]--;
        _numDK.topic[old_topic]--;

        _numWK.topic[new_topic]++;
        _numDK.topic[new_topic]++;

        numWK[w] = _numWK.allTopics;
        numDK[d] = _numDK.allTopics;

        ...
      }
    }

    for(int k = 0; k < K; k++)
      numK[k] = 0;

    for(int i = 0; i < L; i++) {
      toic = the topic of word[i]
      numK[topic]++;
    }
  }
}
```

Listing 2. Extract of the OpenCL kernel that implements the proposed accelerator architecture.

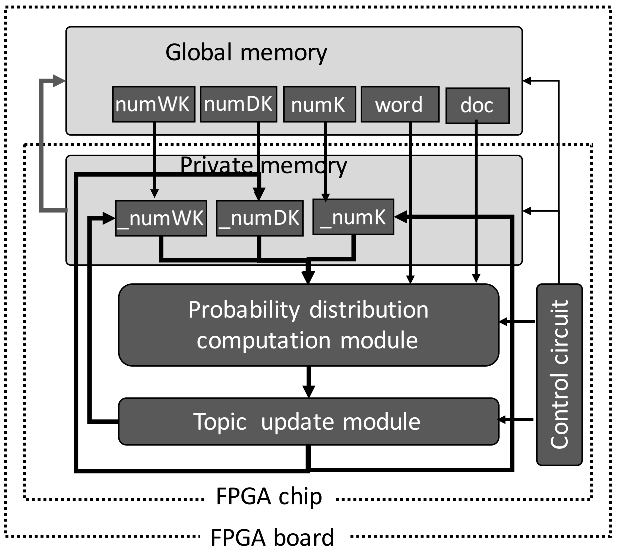


Fig. 7. FPGA architecture based on the proposed memory allocation method.

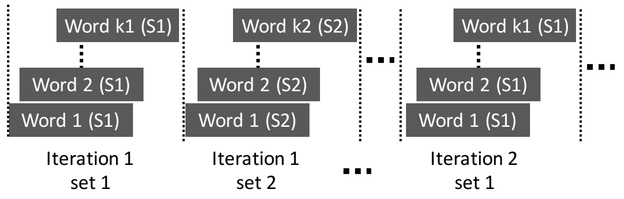


Fig. 8. Time chart of the computation.

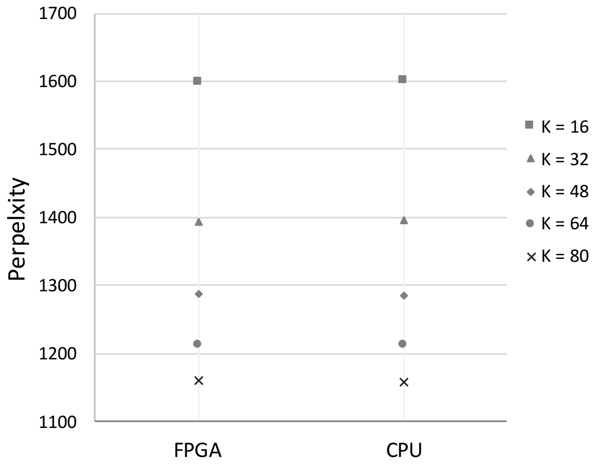
Fig.7 shows the expected accelerator architecture that would be generated by compiling the OpenCL code. All the arrays are initially stored in the global memory. At the beginning of the computation, array `numK` is transferred to the private memory. For each word, the required elements in `numWK` and `numDK` arrays are transferred to the private memory. The data in the private memory are used for the computation. The updated data are written back to the global memory while an another word of the same set is processed. The time chart of the computation is shown in Fig.8. For the same set, the topic inference is done in a pipelined manner where a topic is inferred in every clock cycle. All sets are processed one-by-one in the same pipeline. This process is repeated for all iterations.

IV. Evaluation

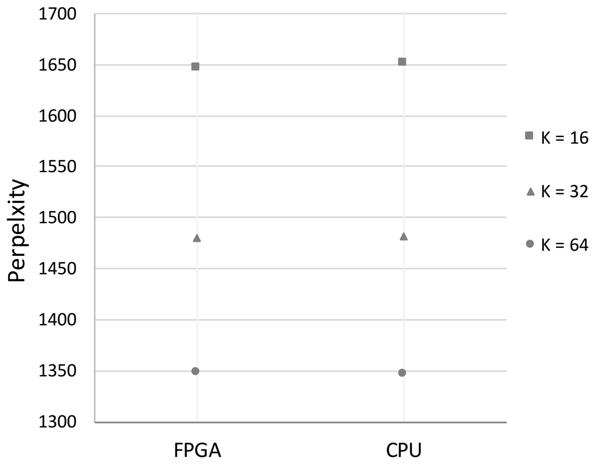
We evaluate the proposed FPGA accelerator for CGS in terms of the inference quality and the processing time. We use two data sets "NIPS" and "KOS" that are available online [10]. Details of the data sets are given in Table I. Training data set has 90% of the total words and the test data set has the remaining 10%. We use the Terasic DE5-Net board [11] that has Intel Stratix V FPGA. Kernel codes are compiled using Intel FPGA SDK for OpenCL Version 16.1. For the comparison with the conventional CPU-based implementation, we use a workstation that has Intel Core i7-4930K CPU and 16GB memory. The conventional CGS algorithm for topic inference is written in C-language, and compiled using Microsoft visual studio C-compiler with relevant optimization options.

TABLE I
Datasets for the experiments.

Dataset	D	L	W
NIPS	1,500	1,932,365	12,419
KOS	3,430	467,714	6,906



(a) Perplexity of the NIPS data set.

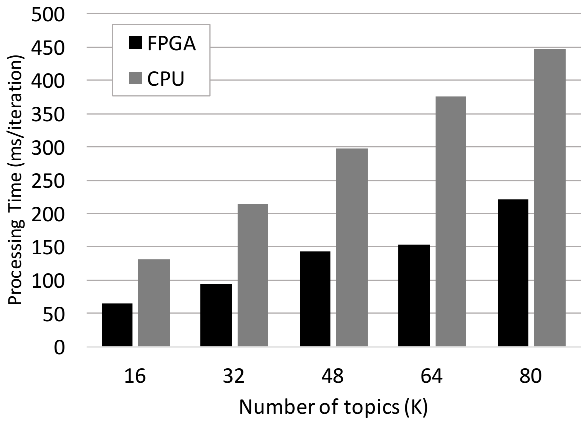


(b) Perplexity of the KOS data set.

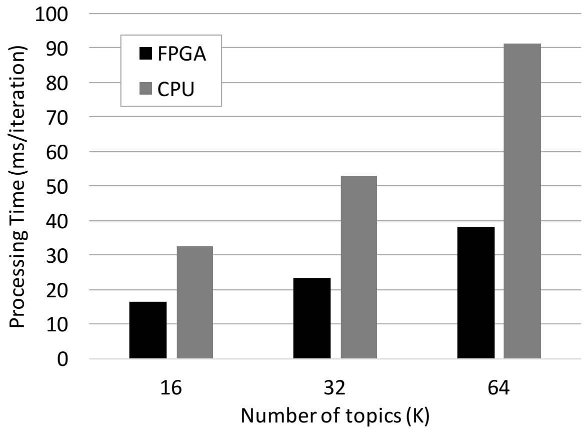
Fig. 9. Comparison of the perplexity against CPU implementation.

We measure the inference quality using perplexity. Perplexity [12] is defined by Eqs.(2), (3) and (4) for the test data set x. The value of n_d is the number of words in document d. Values of n_{wk}, n_k, n_{dk} and n_d are obtained by sampling training data. The number of sampling attempts is given by S.

$$Perplexity(\boldsymbol{x}) = exp(\frac{1}{N_x}\sum_i p(x_i)) \quad (2)$$



(a) Processing time of the NIPS data set.



(b) Processing time of the KOS data set.

Fig. 10. Processing time comparison against CPU implementation.

$$p(x_i) = \frac{1}{S}\sum_s \sum_k \theta^s_{kd_i}\phi^s_{x_i k} \tag{3}$$

$$\theta^s_{kd} = \frac{n^s_{dk}+\alpha}{n^s_d + K\alpha}, \quad \phi^s_{wk} = \frac{n^s_{wk}+\beta}{n^s_k + W\beta} \tag{4}$$

We set the parameters $\alpha = 50/K$, $\beta = 0.1$, $N = 500$, and $S = 10$. The burn-in period is 200 iterations. Fig.9 shows the comparison of perplexity of the proposed FPGA accelerator and the conventional CGS implementation on the CPU. Figs.9(a) and 9(b) shows the perplexity when using NIPS and KOS data sets respectively. There is no difference between the proposed and the conventional implementation in terms of perplexity. It shows that the inference quality of our FPGA accelerator is almost the same as that of the conventional CPU-based implementation.

We measure the processing time of the topic inference using training data. Fig.10 shows the processing time of the proposed FPGA accelerator and the conventional CGS implementation on the CPU. The processing time using NIPS and KOS data sets are given by Figs.10(a) and 10(b) respectively. The processing time of the proposed FPGA accelerator is approximately 50% of that of the conventional method for different K values. Note that, K is the number of topics. The maximum speed-up is 2.38 times.

V. Conclusion

We proposed a pipelined FPGA accelerator for LDA-based inference. We divide the words in to small sets, and allocate words in such a way that the same memory location is not accessed for the computation of two words in the same set. As a result, we were able to fully pipeline the computation of a set to achieve time-step level parallelism. Moreover, the quality of the results is the same, since word-by-word topic inference behavior of the CGS algorithm is preserved in the proposed method. However, the required data access throughput exceeds the bandwidth of the DE5 FPGA board, so that the memory access stall rate is reasonably large. Using an FPGA board with a larger memory bandwidth, or compressing the global memory data may solve this problem and increase the processing speed further.

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] W. Chong, D. Blei, and F.-F. Li, "Simultaneous Image Classification and Annotation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1903–1910, 2009.

[3] S. Mizoguchi, T. Ishii, Y. Nemoto, M. Kaneda, A. Bando, T. Nakamura, and Y. Shimomura, "A Method for Supporting Customer Model Construction: Using a Topic Model for Public Service Design," in *Serviceology for Smart Service System*, pp. 19–25, 2017.

[4] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[5] W. R. Gilks, S. Richardson, and D. Spiegelhalter, *Markov Chain Monte Carlo in Practice*. CRC press, 1995.

[6] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[7] "Intel FPGA SDK for OpenCL." https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html, 2016.

[8] 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.

[9] "The open standard for parallel programming of heterogeneous systems." https://www.khronos.org/opencl/, 2015.

[10] "UCI Irvine Machine Learning Repository." http://archive.ics.uci.edu/ml/datasets/Bag+of+Words.

[11] "DE5-Net FPGA Development Kit." http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=526&PartNo=4.

[12] D. Newman, A. U. Asuncion, P. Smyth, and M. Welling, "Distributed Inference for Latent Dirichlet Allocation," in *NIPS*, vol. 20, pp. 1081–1088, 2007.