Parallel Calculation of Local Scores in Bayesian Network Structure Learning using FPGA

Ryota Miyagi[1,a)] Hideki Takase[2,b)]

Abstract: A Bayesian network (BN) is a directed acyclic graph that represents the relationships among variables in datasets. Because learning an optimal BN structure is generally NP-hard, scalability is typically limited depending on the amount of available memory. This study proposes a novel scalable method for learning an optimal BN structure using a field-programmable gate array (FPGA). To reduce the amount of required memory, the approach limits the size of the parent set to calculate local scores and does not store the results. Therefore, the proposed method has an advantage over previous dynamic programming algorithms in terms of memory efficiency because these existing algorithms store all exponentially sized local scores. Furthermore, we propose an accelerator for local scores calculation by iteratively processing elements in parallel. When it was evaluated with a 30-variable BN, the accelerator calculated local scores up to 230 times faster than the single-core implementation, and its performance improved drastically with increasing FPGA resources. Moreover, structure learning with the accelerator was performed up to 3.5 times faster than structure learning with the single-core implementation.

Keywords: FPGA, Bayesian networks, reconfigurable computing, high-level synthesis, codesign, big data

1. Introduction

Artificial intelligence technology, including machine learning, has become one of the most active academic fields in recent years. Currently, innumerous devices worldwide are connected to the Internet, and a large amount of data is collected from them. There is a need to accurately and efficiently analyze and utilize these big data. A Bayesian network (BN) is a probabilistic graphical model that encodes conditional independence relations among random variables using a directed acyclic graph (DAG). A BN visualizes the relationships among random variables and efficiently infers the posterior distribution of variables from the given evidence.

Learning the optimal structure of a BN can be formulated as a combinatorial optimization problem that maximizes a scoring function that evaluates how well the candidate BNs represent the relationships between events. However, because this problem is NP-hard [1], the number of BN variables is limited in terms of space complexity, and learning the structure of a BN is time consuming. Dynamic programming algorithms have been developed to learn the provably optimal structure of a BN [2], [3], [4]. In these methods, local scores defined by each substructure are calculated in advance and stored in memory to eliminate redundant calculations. However, owing to the significant number of local scores, previous algorithms required considerable memory space.

This study proposes a scalable parallel calculation method of local scores in BN structure learning using FPGA. Our method does not store a vast number of local scores to reduce the amount of required memory. We further calculate local scores in parallel using FPGA. This method promises to increase the scalability as the FPGA resources increase.

The remainder of this paper is organized as follows. In Sec. 2, as preliminaries, we describe BN and related works on learning the optimal structure of a BN as preliminaries. Sec. 3 introduces the structure learning and parallel calculation of local scores using the proposed method. Sec. 4 provides a detailed overview of the system and architecture of the FPGA accelerator. In Sec. 5, we describe the evaluation of the proposed method. Finally, Sec. 6 concludes the paper and discusses future works.

2. Preliminaries

2.1 Bayesian network

BN, which was proposed in 1985 by Judea [7], is a probabilistic graphical model that encodes conditional independence relations among random variables using DAG. It visualizes the relationships among random variables and efficiently infers the posterior distribution of variables from the given evidence. Therefore, BN has various applications in various fields, such as medical diagnosis, genetic phylogenetic analysis, speech recognition, finance, and gene sequence analysis.

There is significant interest in learning BNs from data. The log marginal likelihood with Dirichlet priors over the BN parameters is commonly used to quantitatively evaluate how well the DAG structure of the BN candidate encodes the conditional independence relations among events in the dataset. Let G be the DAG structure of the BN, and D be the observed data. Then, the log marginal likelihood score of the BN is written in the following form:

[1] Kyoto University, Sakyo, Kyoto 606–8501, Japan
[2] University of Tokyo, Bunkyo, Tokyo 113–8654, Japan
[a)] miyagi.ryota.86s@st.kyoto-u.ac.jp
[b)] takasehideki@hal.ipc.i.u-tokyo.ac.jp

$$\log P(G|D) = \sum_{i=0}^{n}\left[\sum_{j=1}^{q_i}\log\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ik})}+\sum_{k=1}^{r_i}\log\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}\right] \quad (1)$$

where

$$\alpha_{ij}=\sum_{k=1}^{r_i}\alpha_{ijk},\ N_{ij}=\sum_{k=1}^{r_i}N_{ijk} \quad (2)$$

where n is the number of variables in BN, q_i is the number of combinations of the ith variable's parent variable values, r_i is the number of ith variable values, and N_{ijk} is the number of data whose ith variable is k. The ith variable's parent variable is the jth combination and α_{ijk} is the hyperparameter of the Dirichlet prior, which represents the prior knowledge corresponding to N_{ijk}.

These scores can be re-written as follows.

$$\log P(G|D)=\sum_{i=0}^{n}LocalScore(v_i,Pa(v_i),D) \quad (3)$$

where $Pa(v_i)$ represents the parent variable set of v_i in G. In other words, the score of the whole graph is decomposed into local scores defined by each variable v_i and its parent variables $Pa(v_i)$. In general, local scores are calculated in advance, which simplifies the evaluation of DAG candidates. However, it is not easy to calculate and store all local scores because their number is $O(n2^n)$. Many parent variables are rarely chosen as a single variable. In some cases, the number of parent variables can be guaranteed to be less than a specific value by mathematical analysis of the scoring function [5]. For these reasons, it is expedient to limit the number of parent variables. However, even if the number of parent variables is limited, the calculation of local scores still takes a long time.

$$\begin{aligned}&LocalScore(x_2,\{x_1,x_4\}) \quad (4)\\ &=\log\Gamma(N_{00*0}+1)+\log\Gamma(N_{01*0}+1)-\log\Gamma(N_{0**0}+2)\\ &+\log\Gamma(N_{00*1}+1)+\log\Gamma(N_{01*1}+1)-\log\Gamma(N_{0**1}+2)\\ &+\log\Gamma(N_{00*2}+1)+\log\Gamma(N_{01*2}+1)-\log\Gamma(N_{0**2}+2)\\ &+\log\Gamma(N_{10*0}+1)+\log\Gamma(N_{11*0}+1)-\log\Gamma(N_{1**0}+2)\\ &+\log\Gamma(N_{10*1}+1)+\log\Gamma(N_{11*1}+1)-\log\Gamma(N_{1**1}+2)\\ &+\log\Gamma(N_{10*2}+1)+\log\Gamma(N_{11*2}+1)-\log\Gamma(N_{1**2}+2)\end{aligned}$$

Eq. (4) shows an example of a local score. Assume a uniform distribution ($\forall i,\forall j,\forall k\ \alpha_{ijk}=1$) as the non-informative prior. This log marginal likelihood score was used as the K2 score [6]. There are four variables: $x_1,x_2,x_3,andx_4$. x_1 and x_2 are assumed to be binary in $\{0,1\}$; x_3 and x_4 are assumed to be ternary in $\{0,1,2\}$. Because the parent variable set is $\{x_1andx_4\}$, there are six combinations of parent variable values: $(0,0),(0,1),(0,2),(1,0),(1,1),(1,2)$. Here, $N_{p_1p_2p_3p_4}$ represents the number of data in D for which $x_1=p_1$, $x_2=p_2$, $x_3=p_3$ and $x_4=p_4$. $p_i=*$ represents a wildcard (i.e., $N_{00*0}=N_{0000}+N_{0010}+N_{0020}$).

Thus, the calculation of local scores depends on the number of data points corresponding to each combination of parent variable values. Other log marginal likelihood scores, such as BDe(u), can be calculated similarly.

2.2 Related Work

The value of $LocalScore(v_i,Pa(v_i),D)$ can be calculated using the contingency table of D for the set of variables $\{v_i\}\sqcup Pa(v_i)$. Sillader et al. [2] start by calculating the contingency table for all the variables and then recursively calculate contingency tables for smaller variable subsets by marginalizing a variable out of them. Once a contingency table is obtained, local scores can be calculated regardless of the size of the large dataset. However, it is difficult to limit the number of parent variables because the smaller set contingency table is obtained from the larger set contingency table. As a result, the calculation takes a long time.

BFBnB [4] uses a data structure called AD-Tree [8] to calculate local scores, exploiting the fact that the number of parent variables is guaranteed to be at most $\log(\frac{2N}{logN})$ by mathematical analysis of MDL scores. AD-Tree is a method for collecting count statistics from a dataset using an unbalanced tree structure. It can be constructed memory-efficiently and quickly for sparse data. However, this method requires memory space to store the local scores calculated in advance. Therefore, the local scores are written to the disk and read as needed.

An optimal BN structure learning method using integer programming has also been proposed [9], [10]. In this method, the number of candidate BNs is drastically reduced by the directed graph constraint, the constraint on the number of parent variables, and the cutting planes with acyclicity constraints. As a result, the method successfully finds the optimal structure of a BN with a maximum of 60 variables and a parent variable limit of three or less. However, if good cutting planes cannot be found, the number of candidate BNs cannot be reduced significantly; hence, the calculation may be time consuming.

We have already proposed an FPGA-based BN structure learning accelerator [11]. In this method, multiple processing modules for calculating sub problems are placed on the FPGA and used in parallel. However, it consumes a large amount of FPGA resources because a large number of sub problem solutions and the dataset are stored on the FPGA. As a result, the upper limit for the number of BN variables and dataset size are 10 and 1000 respectively.

3. Algorithm of Structure Learning

This section describes the algorithms for structure learning and the parallel calculation of the local scores used in the proposed method.

3.1 Dynamic Programming

The optimal BN structure was constructed using the local scores. All provably optimal DAG structures of variables can be determined by recursively adding a leaf variable to the optimal DAG structure of a subset of variables while selecting the optimal parent variables. We define the best graph score $BGS(V)$ as the score of the optimal DAG structure of the variable set V. The best parent score $BPS(x_i,V)$ is the maximum local score when choosing the parent variables of variable x_i from the variable set V. Subsequently, optimal structure learning by dynamic programming can be divided into three steps as follows:

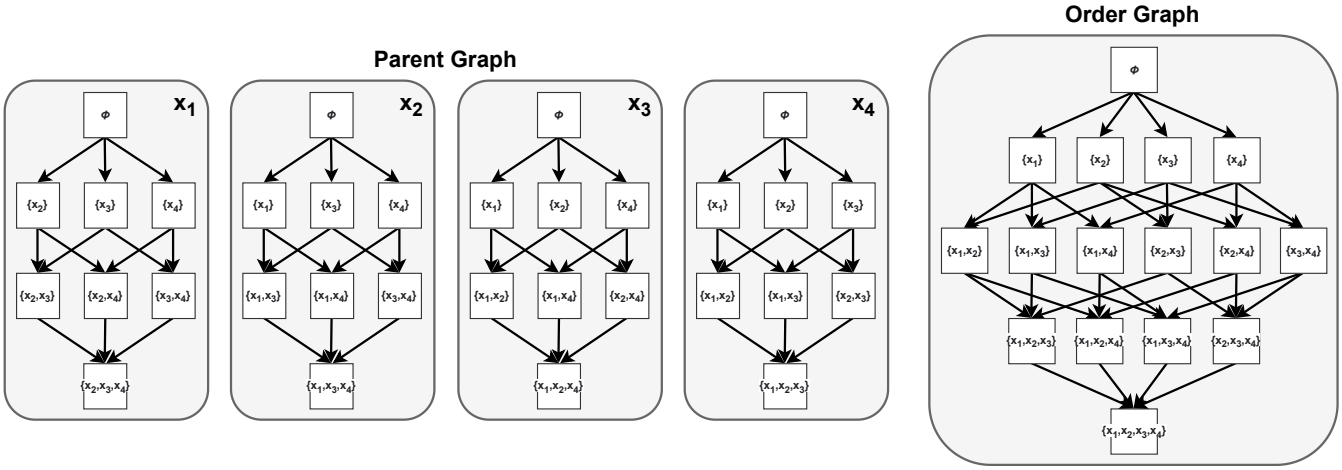

Fig. 1 Parent graph and order graph (n=4)

(1) Calculate $n2^{n-1}$ $LocalScore(x_i, V)$.

(2) Calculate $n2^{n-1}$ $BPS(x_i, V)$ recursively.

$$BPS(x_i, V) = \max \begin{cases} LocalScore(x_i, V) \\ \max_{x_j \in V} BPS(x_i, V \backslash \{x_j\}) \end{cases} \quad (5)$$

(3) Calculate 2^n BGS(V) recursively.

$$BGS(V) = \max_{x_i \in V} (BGS(V \backslash \{x_i\}) + BPS(x_i, V \backslash \{x_i\})) \quad (6)$$

This formulation enables us to achieve optimal structural learning without redundant calculations. Notably, $localscore(x_i, V)$ is used only once in the calculation of $BPS(x_i, V)$, as shown in Eq. (5).

The dependences of the best parent score and best graph score are called the parent graph and the order graph respectively. Fig. 1 shows the parent and order graphs for $n = 4$. Each node has a set of variables V that correspond to $BPS(x_i, V)$ and $BPG(V)$ respectively. There are n parent graphs, each of which is an (n-1)-dimensional hypercube with 2^{n-1} nodes. The order graph is an n-dimensional hypercube with 2^{n-1} nodes.

Both graphs can be divided into layers according to the variable set V size, and each node depends only on the previous layer. Therefore, by expanding the parent and order graphs layer by layer, except for the calculation of local scores, the maximum amount of memory used during the calculation can be reduced from $O(n2^n)$ to $O(n_n C_{\frac{n}{2}})$. BFBnB uses this technique to reduce the amount of memory used at a time [4]. Notably, to expand the mth layer of the parent graphs, we need $\{localscore(x, V) \mid |V| = m\}$.

3.2 Calculation of Local Scores

Algorithm 1 shows the pseudocode used to calculate the local score of the log marginal likelihood score shown in Eq. (1). The count-up, calc-term, and add-term phases are iterated for all combinations of the parent variable values. In the count-up phase, the number of data corresponding to a particular combination of parent variable values is counted. In the calc-term phase, one term of the local score is calculated using the values counted in the count-up phase. In the add-term phase, the terms calculated in the calc-term phase are added to the partial sum of the local scores.

Algorithm 1 $CalcLocalScore(v_i, V, D[N])$

```
ls_i ← 0
for j, ···, NumofCombinations(V) do
    for k = 1, ···, r_i do
        N_ij[k] ← 0
    end for

    /* count-up phase */
    for all d ∈ D do
        for k = 1, ···, r_i do
            if d matches v_i = k ∧ V = j then
                N_ij[k] += 1
            end if
        end for
    end for

    /* calc-term phase */
    N_ij ← 0
    α_ij ← 0
    ls_ij ← 0
    for k = 1, ···, r_i do
        N_ij += N_ij[k]
        α_ij += α_ijk
        ls_ij += log Γ(α_ijk + N_ij[k]) − log Γ(α_ijk)
    end for
    ls_ij *= (log Γ(α_ij) − log Γ(α_ij + N_ij))

    /* add-term phase */
    ls_i += ls_ij
end for
return ls_i
```

As the pseudocode shows, the calculation of different local scores is not dependent on each other. Thus, we can calculate the local scores in parallel. However, each parallel calculation requires a dataset D, which may be very large. Therefore, it is difficult to store a dataset for each parallel calculation.

Subsequently, the dataset is stored in one place and streamed to each parallel calculation module at the appropriate time. As a re-

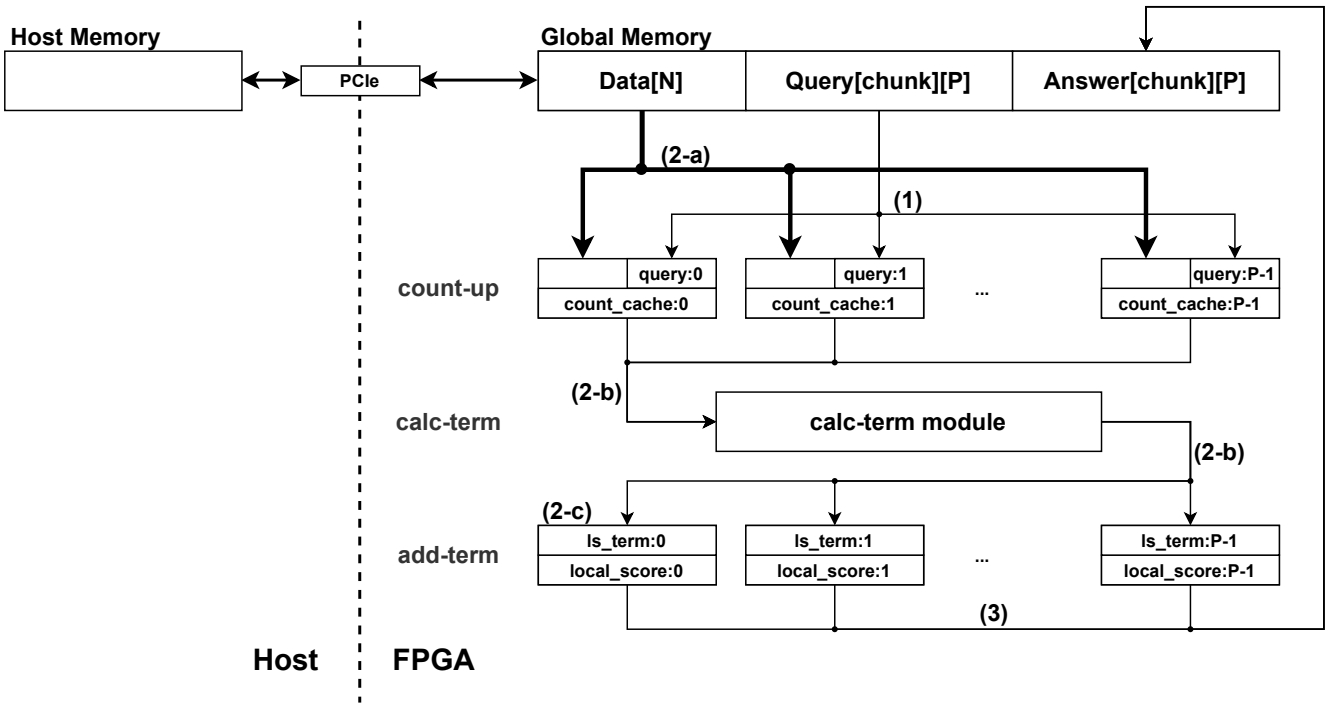

Fig. 2 Overview of the system and the architecture of the parallel calculation module

sult, a high degree of parallelism can be extracted with very few memory resources. FPGAs are ideal devices for such dataflow calculations.

3.3 Strategy of the Proposed Algorithm

First, we expand the parent graphs and order graph layer-by-layer as in BFBnB, which reduces the maximum size of memory required in the computation from $O(n2^n)$ to $O(n_nC_{\frac{n}{2}})$.

Second, we calculate the local scores as needed and do not store their results. On the contrary, previous dynamic programming algorithms calculate large local scores in advance and store them until they are used, even if they are used only once. Thus, the proposed method has an advantage over previous dynamic programming algorithms in terms of memory efficiency.

Finally, we use an FPGA to calculate the local scores in parallel. Notably, we do not store a dataset for each parallel calculation module. Instead, as we mentioned earlier, we store the dataset in one place and stream it to each parallel calculation module at the appropriate time. As a result, a high degree of parallelism can be extracted with very few memory resources.

4. Architecture

4.1 Overview

Fig. 2 shows the overview and architecture of the proposed accelerator. The accelerator is connected to the host PC via the PCI Express interface, on the FPGA board. It is implemented on the FPGA as an OpenCL kernel.

The FPGA accelerator consists of a global memory and an FPGA. We allocate a data, query, and answer regions in the global memory. The data region stores the dataset D; the query region stores an array of local score argument pairs (x, V), and the answer region stores an array of calculated $localscore(x, V, D)$. The FPGA contains P count-up, calc-term, and P add-term modules. The count-up and add-term modules are placed side by side, and the calc-term module is pipelined. The accelerator calculates the P local scores in parallel.

The accelerator streams the dataset from the global memory and does not store it in the FPGA. Therefore, this architecture can be applied to a large dataset of several GB sizes.

4.2 Calculation Flow

First, at the beginning of structure learning, the dataset is transferred from the host PC to the data region of the global memory. When expanding each layer of the parent and order graphs, the host PC lists the necessary local scores and transfers the argument array of those local scores to the query region of the global memory.

The accelerator works as follows:

(1) **Initialize phase** Burst transfer arguments of each local score from the query region to each count-up module. Each count-up module identifies the target data pattern using this argument.

(2) **Running phase** Iterate the following three phases for all combinations of parent variable values.

- (2-a) **Count-up phase** Stream dataset from the data region to each count-up module, which counts the number of data corresponding to the target combination of parent variable values in parallel.
- (2-b) **Calc-term phase** Calculate a term of each local score in the calc-term pipeline using the number counted in each count-up module.
- (2-c) **Add-term phase** Add each term calculated in the calc-term pipeline to each partial sum.

(3) **Terminate phase** Burst transfer all local scores to the answer region.

Thus, the accelerator calculates P local scores at a time.

If there are more than P queries, the accelerator iterates until all of them are calculated. Subsequently, the accelerator transfers all local scores from the answer region of the global memory to the host PC.

Finally, the host PC expands the parent and order graphs using the local scores and calculates the next layer. If the number of parent variables is limited to m or less, the host PC expands the parent and order graphs to $m + 1$ or more layers without local scores.

4.3 Calculate Log-Gamma Function

In the calc-term phase, we need the value of the logarithm of the gamma function of a given value. The lanczos approximation is a method for numerically calculating the gamma function [12]. It approximates the gamma function with high accuracy. In addition, it can be calculated using only the constants and elementary functions. Therefore, the lanczos approximation is ideal for calculating the gamma function on an FPGA. The implementation uses the single-precision floating-point log function of the Xilinx Vitis high-level synthesis (HLS) math library.

The floating-point log-gamma function consumes many DSP resources in the FPGA. Therefore, it is impossible to place many log-gamma modules, such as the count-up and add-term modules, in the system. Thus, we pipelined the log-gamma module to save the DSP resources.

4.4 Approximate Calculation Time

When calculating local scores with many parent variables, the running phase takes up most of the total calculation time. Let the size of all datasets be N, and let the number of count-up modules be P. Because the size of the dataset is N, the count-up phase requires N clocks. Because the calc-term module is pipelined, the calc-term phase takes $P + c_1$ clocks, where c_1 is constant. The add-term phase takes a constant c_2 clock. The accelerator calculates P local scores at a time. Thus, the number of clocks required to calculate a local score is proportional to $\frac{N+P+c_1+c_2}{P} \sim \frac{N}{P} + 1$. As a result, parallelism does not contribute significantly to acceleration when N and P are close in value. Conversely, if N is sufficiently larger than P, the acceleration is expected to be close to P.

Table 1 Result of logic synthesis (P=1024)

Resource	LUT	LUTRAM	FF	BRAM	DSP
kernel	324129	16861	441186	213	490
Available	870016	402016	1740032	1344	5940
Utilization(%)	37.26	4.19	25.36	15.85	8.25

Table 2 Result of logic synthesis (P=2048)

Resource	LUT	LUTRAM	FF	BRAM	DSP
kernel	476544	12417	656236	200	490
Available	870016	402016	1740032	1344	5940
Utilization(%)	54.77	3.02	37.71	14.88	8.25

5. Evaluation

We prepared two accelerators with parallelism P of 1024 and 2048 and compared three implementations: a single-core software execution (SW), single-core software execution using an FPGA accelerator with parallelism 1024 (HW (P = 1024)), and single-core software execution using an accelerator with parallelism 2048 (HW (P = 2048)).

For the evaluation, we employed Xilinx Alveo U50 [13], which is an FPGA board used for data center accelerators. We used Vitis 2020.2 to design the entire system, including the host application and accelerator. The host PC employs Intel Xeon W-2265 (3.5GHz 12-core/24-thread processor), 64 gigabytes of memory and runs on Ubuntu version 18.04. We designed the accelerator in C++ and applied high-level synthesis to generate RTL circuit descriptions. We used the K2 score, which is a log marginal likelihood score.

5.1 Synthesis Results

High-level synthesis and logic synthesis were performed using the Vitis 2020.2 toolchain. Table 1 and 2 show the resource utilization of HW (P = 1024) and HW (P = 2048), respectively. The operating frequencies of both were 300 MHz.

The accelerator does not use many BRAMs because it does not store the dataset on the FPGA. In addition, it does not use much DSP because the log-gamma module is implemented as a pipeline. However, it uses many LUTs to transfer data to each module. Thus, the amount of LUT resources determines the upper limit of parallelism.

Table 3 Comparison of local scores calculation time on SW/HW [s]

N	m	SW	HW(P=1024)	HW(P=2048)
1000	1	0.004	0.020	0.022
1000	2	0.085	0.021	0.023
1000	3	1.290	0.037	0.036
1000	4	14.483	0.205	0.192
1000	5	130.059	1.824	1.667
1000	6	949.879	14.270	12.987
1000	7	5930.838	93.125	84.612
1000	8	N/A	510.549	463.556
1000	9	N/A	2378.568	2158.771
10000	1	0.079	0.020	0.024
10000	2	1.390	0.023	0.026
10000	3	16.216	0.080	0.064
10000	4	154.459	0.778	0.528
10000	5	1268.540	7.552	5.010
10000	6	9175.792	60.080	39.712
10000	7	N/A	394.153	260.212
10000	8	N/A	2166.152	1429.310
10000	9	N/A	10104.604	6665.651

Table 4 Comparison of structure learning time on SW/HW [s]

N	m	SW	HW(P=1024)	HW(P=2048)
1000	5	4250	4128	4102
1000	6	5180	4146	4113
1000	7	11678	4229	4237
1000	8	N/A	4769	4660
1000	9	N/A	7134	6812
10000	5	5580	4156	4131
10000	6	14816	4174	4132
10000	7	N/A	4603	4450
10000	8	N/A	6794	5876
10000	9	N/A	16868	12532

5.2 Performance of the Accelerator

Using a BN with 30 variables, we measured and compared the time required to calculate all local scores for a specific number of parent variables with datasets of size 1000 or 10000. We assume that the value of each random variable is binary. Table 3 shows the execution time results for each configuration. It should be noted that N is the size of the dataset, and m is the number of parent variables. Each execution was terminated after 5 h (18,000 s). Therefore, N/A indicates that the calculation takes at least 18,000 s.

When $m = 1$ and $N = 1000$, SW is faster than HW because of the latency of communication. In other cases, however, HW is faster. The calculation time for SW increases as the number of parent variables increases. As a result, SW can be calculated only when $N = 1000$ and $m < 7$, or $N = 10000$ and $m < 6$. HW (P = 1024) and HW (P = 2048) calculated all local scores over time in all cases. Furthermore, HW (P = 2048) is approximately 70 times faster than SW when $N = 1000$ and $m = 7$, and 230 times when $N = 10000$ and $m = 6$.

The comparison of HW (P = 1024) and HW (P = 2048) proves that the more the FPGA resources, the faster is the calculation of local scores. When $N = 1000$, there is not much difference in the calculation time because the values of N and P are close. In contrast, when $N = 10000$, there is a significant difference in the computational time.

5.3 Performance of the Structure Learning

Using the same BN discussed in Sec. 5.2, we measured and compared the time for structure learning when varying the upper limit of the number of parent variables with datasets of size 1000 or 10000, respectively. Table 4 shows the execution time results for each configuration. Each execution was terminated after 5 h (18,000 s). The calculation for structure learning, except for the calculation of local scores, is common and takes approximately 4100 s.

In all cases, the HW performed structure learning faster than SW. HW (P = 2048) is approximately 2.7 times faster when $N = 1000$ and $m = 7$, and approximately 3.5 times faster when $N = 10000$ and $m = 6$, respectively. Furthermore, HW (P = 2048) performs fast structure learning even when $N = 10000$ and $m = 9$.

6. Conclusion

In this paper, we proposed an FPGA-based parallel calculation of local scores for BN structure learning. By expanding the parent and order graphs layer by layer, we reduced the required memory space from $O(n2^n)$ to $O(n_nC_{\frac{n}{2}})$. The proposed method calculates local scores as needed and does not store them, thus performing structure learning with less memory. In addition, we placed a large number of local score computation modules on FPGAs and used them repeatedly in parallel. The dataset was streamed to each calculation module at the appropriate time and was not stored in the FPGA. This allowed us to extract a high degree of parallelism with a small amount of memory.

The accelerator evaluation shows that the accelerator calculates the local score up to 230 times faster than the single-core implementation and proves that the more the FPGA resources there are, the faster is the calculation of local scores. Furthermore, structure learning using the accelerator was 3.5 times faster than using a single-core implementation. The proposed method promises to exploit higher parallelism using multiple FPGA cards.

As shown in the evaluation results of structure learning, it takes a long time to expand the parent and order graphs. We plan to focus on this problem to achieve further acceleration of structure learning.

Acknowledgments The part of this work was supported by JST PRESTO Grant Number JPMJPR18M8. The authors would like to thank Prof. Kentaro Sano at RIKEN R-CCS for helpful discussions.

References

[1] David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of Bayesian networks is NP-hard. Journal of Machine Learning Research, 20:1287–1330, October 2004.

[2] Tomi Silander and Petri Myllymaki. A simple approach for finding the globally optimal bayesian network structure. Proceedings of the 22nd Conference in Uncertainty in Artificial Intelligence, Cambridge, MA, USA, July 13-16, 2006.

[3] Mikko Koivisto, Kismat Sood, and Maxwell Chickering. Exact Bayesian structure discovery in Bayesian networks. Journal of Machine Learning Research, 5:549–573, 2004.

[4] Brandon Malone, Changhe Yuan, Eric A. Hansen, and Susan Bridges. Improving the Scalability of Optimal Bayesian Network Learning withExternal-Memory Frontier Breadth-First Branch and Bound Search. Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial IntelligenceJuly 2011.

[5] Jin Tian. A branch-and-bound algorithm for MDL learning Bayesian networks. Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, 580–588. Morgan Kaufmann Publishers Inc.

[6] Gregory F. Cooper, and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. Machine Learning, 9, 309-347, (1992).

[7] Judea Pearl. Probabilistic Reasoning in Intelligent Systems: Networks ofPlausible Inference. San Francisco, CA, USA: Morgan KaufmannPublishers Inc., 1988.

[8] Andrew W. Moore and Mary Soon Lee. Cached sufficient statistics for efficient machine learning with large datasets. J. Artif. Int. Res. 8:67–91. 1998.

[9] James Cussens. Bayesian network learning with cutting planes. UAI'11: Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial IntelligenceJuly, 153–160, 2011.

[10] Tommi Jaakkola, David Sontag, Amir Globerson, Marina Meila. Learning Bayesian Network Structure using LP Relaxations. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:358-365, 2010.

[11] Yasuhiro Nitta, Hideki Takase. An FPGA Accelerator for Bayesian Network Structure Learning with Iterative Use of Processing Elements. 2020 International Conference on Field-Programmable Technology (ICFPT).

[12] Lanczos, Cornelius. A Precision Approximation of the Gamma Function. Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, vol. 1, no. 1, pp. 86-96, January, 1964.

[13] Xilinx Inc., "Alveo U50 Data Center Accelerator Card" `https://xilinx.com/products/boards-and-kits/alveo/u50.html`