Neurocomputing 000 (2017) 1–10

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Node-level parallelization for deep neural networks with conditional independent graph☆

Fugen Zhou[a], Fuxiang Wu[a,*], Zhengchen Zhang[b], Minghui Dong[b]

[a] Beihang University, XueYuan Road No. 37, HaiDian District, Beijing, China
[b] Institute for Infocomm Research (I^2R), Agency for Science, Technology and Research, Singapore

ARTICLE INFO

Article history:
Received 13 August 2016
Revised 26 February 2017
Accepted 3 June 2017
Available online xxx

Communicated by Dr. Guan Ziyu

Keywords:
Node-level parallelization
Deep neural networks
Conditional independent graph
OpenMP
Concurrent kernels

ABSTRACT

Deep neural networks require high performance computing and highly effective implementation to constrain the running time into a reasonable range. We proposed a novel node-level parallelization, conditional independent parallelization, of the forward and backward propagations to improve the level of concurrency. The propagations exploit a conditional independent graph (CIG) built in $O(N)$ times, which consists of conditional independent sets of nodes. Each set in the CIG is sequentially visited, while the nodes in the set are calculated concurrently. Besides, we analyze the properties of the CIG and prove the correctness of the propagations with the CIG, then study the theoretical speedup ratios of the parallelization. Moreover, this parallelism can be applied to arbitrary structures of neural networks without influencing convergence, which only needs a conditional independent graph. It can be further integrated into other frameworks with batch-level and data-level parallelism to improve the level of concurrency. Since modern GPU supports concurrent kernels, the parallelization can also be implemented on GPU directly. To verify the parallelization in experiments, we implement an autoencoder, a dependency parser and an image recognizer with the parallelization and test them on a 4-core CPU I7 4790K with 32 GB memory. The results demonstrate that it can achieve maximum speedups of 3.965 × for the autoencoder, of 3.106 × for the parsing and of 2.966 × for the recognizer.

1. Introduction

Recently, deep learning, which is a subfield of machine learning, has attracted significant attention in the community and achieved extraordinary results in speech recognition [1], natural language processing [2], computer vision [3], etc. By exploiting deep-layered hierarchical architectures, deep learning can efficiently abstract high-level pattern from data. There are two typical structures of deep learning: feedforward neural networks (FNNs) and recurrent neural networks (RNNs). FNNs include feedforward convolutional neural networks, which are acyclic and widely used in visual processing [4,5]. However, RNNs take their last output and the next element in an input sequence as inputs, which will lead to forming directed cycles. Therefore, RNNs can process arbitrary sequences, and are successfully applied in speech recognition [6], nature language processing (NLP) [2,7], etc. Normally, neural networks (NNs) require two propagating stages: the forward propagation stage, in which neurons are orderly updated to generate output, and backward propagation stage where the gradients of weight matrices of a neuron are computed in a revert order to train the neural networks. Besides, a gradient descent algorithm is exploited to train the neural networks, such as stochastic gradient descent (SGD) [8], adaptive gradient descent algorithm (ADAGRAD) [9], mini-batch stochastic gradient descent (mini-batch SGD) [10], etc.

Since the computational load of neural networks is very high, especially for training, many parallel devices are exploited to accelerate that computing. In fine grain parallelism, as the computing of neural networks mainly composed of the operators between matrices and vectors, the acceleration can be well tackled by highly optimized basic linear algebra subroutines, such as the Intel MKL packages or GPUs libraries like cuBLAS and cuDNN. In coarse grain parallelism, since the network topology for a problem is unknown, the batch-level parallelization strategy [11] is widely used in the community. The strategy is commonly adopted in mini-batch training, where the evaluations for each sample in a mini-batch are unordered and commutative so that they can be evaluated in parallel directly. Moreover, there is a native node-level

☆ This work was sponsored in part by National Natural Science Foundation of China (61233005).

* Corresponding author.
E-mail addresses: zhfugen@buaa.edu.cn (F. Zhou), fxwuedu@buaa.edu.cn, fxwuedu@buaa.edn.cn (F. Wu), zhangzc@i2r.a-star.edu.sg (Z. Zhang), mhdong@i2r.a-star.edu.sg (M. Dong).

http://dx.doi.org/10.1016/j.neucom.2017.06.002
0925-2312/

parallel strategy, which is finer than batch-level parallelization. TensorFlow [12] keeps track of dependencies of a node, which represents a computation, such as matrix multiply, and calculate that node if all dependencies have been executed. Thus, there may be several nodes that are ready to compute simultaneously . However, similar to the driver overhead problem in the general-purpose graphics processing unit (GPGPU) [13–15], the native node-level parallelization may suffer that problem. Since the parallelization directly tracks the dependencies of a node, the system needs to maintain state variants for each node, which should be thread-safe. Moreover, the system needs to check each node in a node set, which has not been executed, to schedule them. Thus the overhead is high, which may degrade the speedup ratio of the system.

In this paper, we propose a conditional independent extracting algorithm for node-level parallelism. In the algorithm, given a computational graph transformed from the neural networks, a conditional independent graph is built in $O(n)$ times to extract conditional independent sets of operator nodes in the computational graph. And then each element in the set can be concurrently executed in both forward and backward propagation stages. To verify efficiency and effectiveness of the parallelization, we utilize two NLP applications and an image application, namely, an autoencoder for paragraphs, a dependency parser and an image recognizer, and the results show that the parallelization can help to speed-up processes in training and testing efficiently.

2. Related work

There are some deep learning frameworks accelerated by exploiting GPU, multi-threading or distributed computing, such as TensorFlow [12,16], Caffe [17], Theano [18], Torch7 [19], etc. We compare the parallel ways of them as following:

- *TensorFlow* represents a deep learning algorithm as a dataflow graph and supposes multiple concurrent executions of subgraphs of the dataflow graph, which are manually defined by steps, invocations of the API. Moreover, it exploits the native node-level parallelism, fine grain parallelism and batch-level parallelism to speed up the calculation.
- *Caffe* is layer-wise design, where the layer is similar to the node in TensorFlow. Caffe mainly calculates the operations in layers with GPU, which is fine grain parallel.
- *Theano* leverages multi-core CPU architectures only in BLAS and Conv2D, and automatically generates CUDA code for the mathematical operations of nodes. It adopts fine grain parallelization.
- *Torch7* utilizes two parallel libraries: OpenMP and CUDA. However, as described in the work of Collobert et al. [19], Torch7 mainly uses those libraries in the tensor library and neural networks, which indicates that Torch7 adopts fine grain parallelism.

Therefore, some frameworks do not suppose the batch-level parallelism, but the work of Tallada et al. [11] demonstrates that the batch-level parallelism is independent of the supporting libraries so that it can be integrated into the frameworks directly. Besides, the work of Chambers et al. [20] and TensorFlow simultaneously execute independent operations, which complement the batch-level parallelism and fine grain parallelism. However, there are some intrinsic conditional independent properties in the computational graph, which can be further employed to accelerate the frameworks in node-level. Thus, our work complements their works in node-level and automatically extracts conditional independent nodes of the computational graph in $O(n)$ times to improve the node-level parallelization.

3. Conditional independent parallelization

The deep learning model normally consists of many layers of non-linear processing units (neural networks), and it can be expressed as a dataflow graph or computational graph. Given the input data, the model is evaluated via forward propagation, which traverses nodes of the graph from inputs to outputs. For model training, after executing the forward propagation, the backward propagation, which traverses the nodes from outputs to inputs, is invoked to compute gradients of the parameters. And then the parameters are updated with a type of gradient descent algorithms.

3.1. Computational graph

A computational graph $G = \{V, E\}$ consists of a set of nodes (vertexes) $v_i \in V$ and a set of edges $e_{ij} \in E$. G is a directed graph which represents the data flowing and manipulating of a deep learning model. Each v_i represents a computation of the ith node, such as add, multiply, tanh, etc., and e_{ij} represents data flows from the parent node v_i (output) to the child node v_j (input). Fig. 1 illustrates an example of the computational graph G of the 3-gram neural network language model (NNLM) [21], where e_i is an embedding vector of the ith word, and N is the size of vocabulary. Besides $M_{\cdot,\cdot}$ and $V_{\cdot,\cdot}$ are 2-D matrix as described in Arisoy et al. [21].

Since the time of executing a different node may be different, we denote that of a node v_i as a function $t(v_i)$. Thus, running the total computational graph G consumes $\sum_{v_i \in V} t(v_i)$ without accelerating (Since we only consider the performance on a single-machine, so ignore the time of data communication.). Besides, we assume the input data have been transferred to memory, thus $t(v_k) = 0$ if v_k is an input node (such as the nodes $v_1 - v_6, v_{11}$ and v_{14} in Fig. 1).

3.2. Conditional independent graph

The computational graph is analogous to a Bayesian network [22], where a node is independent with the other nodes given its Markov blanket. Inspired by the d-separate definition in Bayesian network, we can extract sets of conditional independent nodes from an arbitrary computational graph. For example, v_{12} and v_{13} are conditionally independent in Fig. 1, and they can be executed concurrently given the set of nodes v_1–v_6, v_{11}, and v_{14}. Therefore, the following Definition 1 describes conditional independent graph which stores the sets of conditional independent nodes for a directed acyclic graph (DAG).

Definition 1. Given a computational graph G, its conditional independent graph $\mathcal{G}$ is a set of the conditional independent sets (CIS), namely, $\{S_1, S_2, \ldots, S_{d(G)}\}$, where $d(G)$ is the size of $\mathcal{G}$, $|\mathcal{G}|$, and owns following properties,

(a) *Independent:* a CIS S_i is topology-independent from other CIS S_j *iif* $j > i$.
(b) *Parallel:* the nodes of a CIS S_i are topology-independent with each other and ready to be executed when the nodes in $\cup_{k<i} S_k$ have been executed or given.

where the notation topology-independent v_i from v_j refers to no incoming edge from v_j to v_i. For simplifying, we will express topology-independent and topology-dependent as independent and dependent respectively in the following part. The graph will group the nodes into different sets to support concurrently running. If the computational graph G is built from bottom to top, the node $v_i \in V$ is independent of the nodes $\{v_j\}_{j>i}$, which is similar to the partially causal sequence introduced by Schmidhuber [23], and we denote the property of V as well-form. Thus, we assume that V is well-form (If not, we sort them via traversing the

Please cite this article as: F. Zhou et al., Node-level parallelization for deep neural networks with conditional independent graph, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.06.002