LOCAL DISTRIBUTED MOBILE COMPUTING SYSTEM FOR DEEP NEURAL NETWORKS

by

Jiachen Mao

B.S. Software Engineering, Shanghai Jiao Tong University, 2015

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2017

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Jiachen Mao

It was defended on

April 3, 2017

and approved by

LOCAL DISTRIBUTED MOBILE COMPUTING SYSTEM FOR DEEP NEURAL NETWORKS

Jiachen Mao, M.S.

University of Pittsburgh, 2017

Nowadays, Deep Neural Networks (DNN) are emerging as an excellent candidate in many applications (e.g., image classification, object detection and natural language processing). Though ubiquitously utilized in many fields, DNN models are generally hard to be deployed on resource-constrained devices (e.g., mobile devices). In the prior arts, the research topics mainly focus on client-server computing paradigm or DNN model compression, which, respectively, ask for either outside infrastructure support or special iterative training phases. In this work, I propose a local distributed mobile computing system for the testing phase of DNNs called MDNN, short for Mobile Deep Neural Network. MDNN partitions already trained DNN models onto several mobile devices with the same local wireless network to accelerate DNN computations by alleviating device-level computing cost and memory usage. Two model partition schemes are also designed to minimize non-parallel data delivery time, including both wakeup time and transmission time. Experimental results show that when the number of worker nodes increases from 2 to 4, MDNN can accelerate the DNN computation by 2.17-4.28. Besides the parallel execution, the performance speedup also partially comes from the reduction of the data delivery time, e.g., 30.02% w.r.t. conventional 2D-grids partition. Furthermore, a model compression using group lasso is utilized for simultaneously alleviating computing cost and transmission cost.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

Among many people who helped me with this work, I want to first thank my advisor, Dr. Yiran Chen, for his relentless support throughout the entire duration of my graduate research, which forms the foundation of this thesis. He invited me to his excellent research group in which I initiated my first research project and have been actively participated during my PhD program. It was him who come up with the core idea of my research topic and provided a lot of useful related literatures to me.

Furthermore, I would like to thank Dr. Hai Li for her instructive advice, helping me establishing my research experiences from ground up and follow the right direction. Her strong enthusiasm motivates me to concentrate on my high performance computing research. Without her help, I could have never done this work.

I also want to thank Prof. Zhi-Hong Mao and Prof. Samuel Dickerson for being on my program committee and giving me constructive advice on this dissertation. I highly appreciate their time spent on reviewing the dissertation.

1.0 INTRODUCTION

In recent years, Artificial Intelligence (AI) arouses many attention and has been viewed as the beginning of next era in computer science and engineering. Among all the machine learning approaches, Neural Networks occupy a key position because of their dominant advantages like their high accuracy, self-adaptive property, scale flexibility and data-driven property.

Neural networks refer to a computing system made up of several interconnected processing neurons, which can process the external input data to get their information. Typically, neural networks are organized in layers, which consist a number of interconnected neurons. For each neuron, an activation function will be included for non-linearization so that the outputs of neuron networks are not necessary to be strictly linearly related to the external inputs.

Different from conventional computing algorithm, neural networks are not deterministic. All the information is contained in the activation state of the networks so as to extract the deep features of the inputs and induce the output. When represented in computing platforms, a neural network contains two kinds of data: symbol file and parameter file. Symbol files express the detailed layer structures accompanied with their meta data such as the number of neurons, the type of activation functions, and the filter size. Parameter files embody the weights of the neural network which connect the current layer with the previous one where the values of the weights quantify the affect of the previous neurons on the current one.

With the ever-increasing complexity of the datasets in the existing problem to be solved, the scale of the neural networks come to be unprecedented large so as to reach the target functionality (e.g. ImageNet). Those kinds of large-scale neural networks are called Deep Neural Networks (DNN), which are both memory-intensive and computing-intensive when deployed on computing devices, especially mobile platforms. Hence, the popularity of DNN incurs the emergence of many NN-oriented hardware design. [17] implemented a neuromorphic computing systems (NCS) on memristor crossbar. [15] demonstrated the promise of using resistive random access memory (ReRAM) to perform neural computations in memory and gave dedicated hardware-level architecture for executing deep neural networks in both the forward and backward procedures. [12] accelerated Field Programmable Gate Array (FPGA) when running Recurrent neural network (RNN) based language model (RNNLM). The ever-increasing bandwidth of mobile networks inspired rapid growth of multimedia interactive applications on mobile devices, which involve intensive object recognition and classification tasks. With the emergence of massive mobile network bandwidth, novel interactive applications in mobile devices utilize more and more multimedia processing, boosting the performance requirement of object recognition and classification.

Deep Neural Networks (DNN) have been widely used in performing these tasks due to their high accuracy and self-adaptiveness property. However, execution of DNN incurs considerably resources. A representative example is VGG [14], which demonstrates state-of-the-art performance in ImageNet Large Scale Visual Recognition Challenge 2014 (ILSVRC14). VGG has 15M neurons, 144M parameters, and 3.4B connections. When deployed on a mobile device, VGG spends approximately 16 seconds to complete the identification procedure for one image, which is intolerable in practical.

The gap between large computing workloads of DNN and limited computing resources of mobile devices adversely impact user experience and inspired some research works to fill the gap. Thus, some research works have been done to fill the gap. To efficiently offload the huge computing cost to outside infrastructure, client-server computing paradigm is the most straight-forward solution: In [8], Hauswald proposed a data offloading scheme in a pipelined machine learning structure; In [11], Li established an efficient distributed parameter server framework for DNN training. In addition, many studies have been performed to reduce the computing workloads of DNN, such as model compression: To enable local execution of DNN models on mobile devices, attempts have also been made in model compression during the training phase of DNN: In [7] Han deeply compressed the DNN models using a three stage pipeline: pruning, trained quantization, and Huffman coding; In [2], Chen introduced a low-cost hash function to group weights into hash buckets for parameter sharing purpose.

We note that there is an important scenario that has not been fully explored yet in all the previous works, say, running DNN on a local distributed mobile computing system. Compared to client-server paradigm where a single mobile device is supported by external infrastructure, local distributed mobile computing systems offer several important advantages, including more local computing resources, higher privacy, less dependency on network bandwidth, etc. However, in previous research works, the opportunity of local distributed mobile computing systems are overlooked with its huge potential in accumulated resources to identify an image with DNN models. Moreover, such a system also offers the additional advantages of optimal privacy security, infrastructure-less networks, no information loss of DNN models, and no extra training phases.

In this work, we propose MoDNN - a local distributed mobile computing system for DNN that can work over a Wireless Local Area Network (WLAN). MoDNN can significantly speedup the computation of DNN by introducing execution parallelism among multiple mobile devices. As the overheads of non-parallel transmissions between the mobile devices in the formed computing cluster are considerably high, our research particularly focuses on minimizing the overheads of non-parallel transmissions between the mobile devices in the formed computing cluster. Our contributions include:

1) We investigate the method of building a computing cluster in WLAN with multiple authorized Wi-Fi enabled mobile devices for DNN computations. The mobile device that carries the testing data (e.g., image) acts as the Group Owner (GO) and the other devices act as the worker nodes; 2) We propose two partition schemes to minimize the data delivery time between the mobile devices based on the unique properties of two types of DNN layers (convolutional layers and fully-connected layers) and various mobile computing abilities; 3) We employ a middleware on each mobile device in the computing cluster to schedule the whole execution process. To the best of our knowledge, this is the first work that utilizes heterogeneous mobile devices in WLAN as computing resources for DNN with several innovations in execution parallelism enhancement and data transmission. Experimental results show that when the number of worker nodes increases from 2 to 4, MoDNN can speedup the DNN computation by 2.17 to 4.28 times, thanks to the achieved high execution parallelism and the significantly reduced data delivery time.

2.0 PRELIMINARY

2.1 OPPORTUNISTIC MOBILE NETWORK

Benefiting from high transmission bandwidth and robust protocol, WLAN serves as an ideal environment to create opportunistic mobile networks for cluster computing. With the ever-developing transmission bandwidth and protocol robustness, WLAN creates an optimal environment for an opportunistic mobile network from which cluster computing is highly benefited. The topology of opportunistic mobile networks allows mobile devices to communicate over a WLAN.

Such a proximity-based communication characteristic also enables high data transmission bandwidth between the mobile devices. In this work, we adopt opportunistic mobile networks as our network foundation and implement MoDNN using WiFi-Direct [1], which allows for a transfer speed of up to 250 Mbps with an energy efficiency better than a cellular network.

2.2 DISTRIBUTED PROGRAMMING MODEL

MapReduce is a programming model for simplifying parallel data processing in distributed systems [4]. Its effectiveness has been proven in many machine learning applications through maximizing the usage of computing resources of the nodes in a computing cluster [13]. There are two primitives in MapReduce applications: Map and Reduce. The Map procedure partitions a task to pieces that can be executed in parallel while the Reduce procedure merges the intermediate data from the Map procedure. In this work, we use these two primitives in our single-GO multiple-clients network topology to describe the data transmissions between DNN layers.

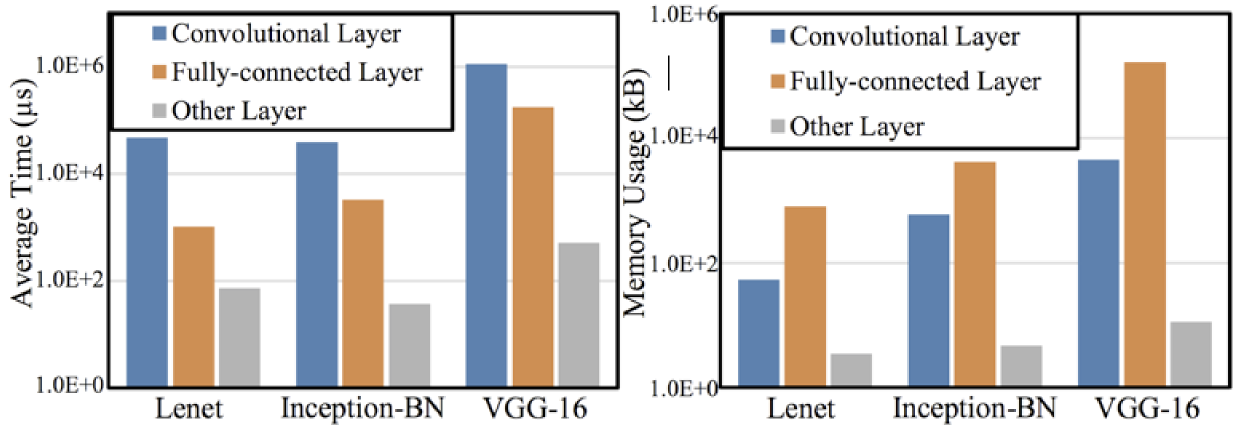


Figure 1: Average computing time and memory usage of the layers in DNNs.

2.3 PROPERTIES OF TWO TYPES OF LAYERS IN DNN

In a DNN, the most computing-intensive and memory-intensive layers are Convolutional Layers (CL) and Fully-connected Layers (FL). Fig. 1 depicts our measurement results of the computing time and the memory usage of different network layers from three popular DNN models running on smartphones: Lenet [10], Inception-BN [9], and VGG [14]. Although these three models have very different scales, two common properties are observed in all three measurements: (1) CLs contribute to the majority (e.g., 86.5% to 97.8%) of the total computing time; (2) FLs contribute to more than 87.1% of the total memory that are used to store the parameters of the DNN model. Hence, in this work, we will particularly investigate the partition schemes of these two types of layers in the DNN. Sparsifying FLs is a promising technique that can effectively reduce the associated computing cost [13]. For example, the connectivity of FLs in VGG-16 can be reduced by 95.6% without incurring any accuracy loss [7]. As we shall show in the following section, model sparsity in DNNs offer a great opportunity for MoDNN to reduce the data delivery time by optimizing the network weight partition of sparse FLs.

3.0 SYSTEM FRAMEWORK OF MODNN

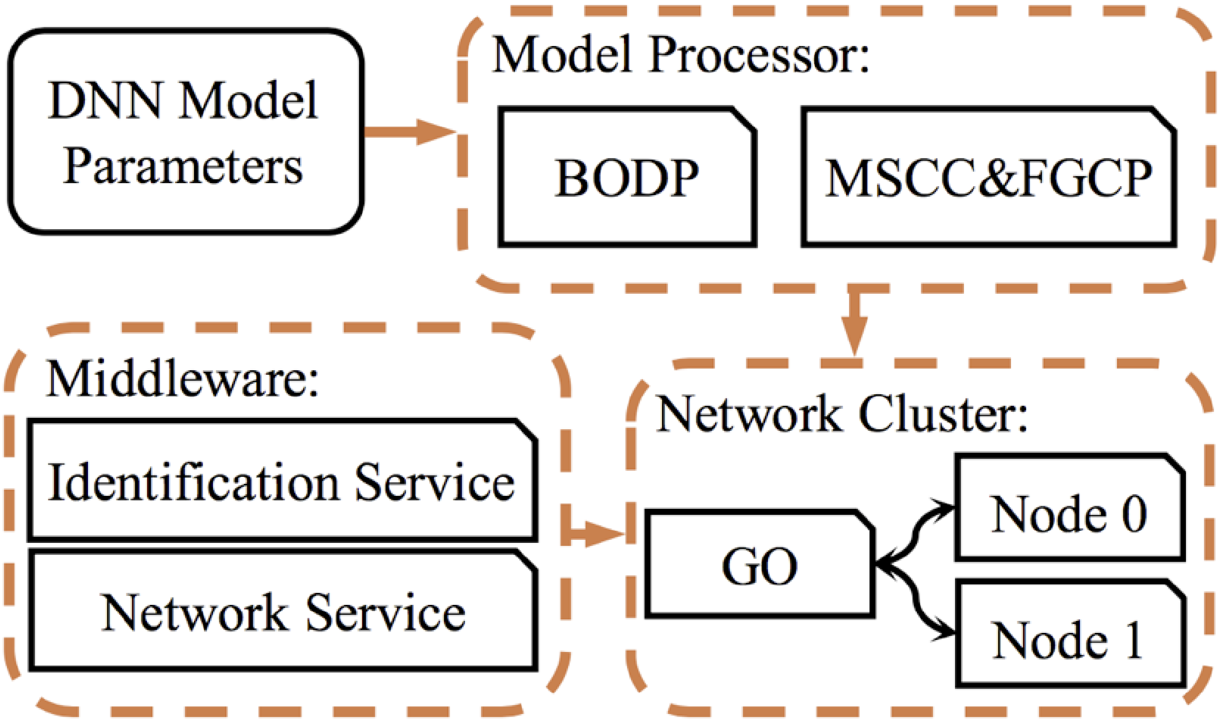


Figure 2: System overview of MoDNN.

3.1 OVERVIEW OF MODNN

Fig. 2 presents an overview of MoDNN which includes three main components: (1) A local distributed network cluster formed by GO and multiple worker nodes; (2) A model processor which partitions the DNN model onto the worker nodes; and (3) A middleware that performs data delivery and identification services of the DNN.

We note that the computing cost of CLs is primarily dependent on its input size. Hence, we introduce a Biased One-Dimensional Partition (BODP) scheme to partition the CLs. On the contrary, the memory usage of FLs is mainly decided by the number of weights in the layer. As a result, a weight partition scheme that consists of Modified Spectral Co-Clustering (MSCC) and Fine-Grain Cross Partition (FGCP) is introduced specifically for sparse FLs.

It is worth noting that here the DNN model partition only need to be performed once in the application once the DNN is trained. Thus, the partition cost can be amortized over the execution of the system as long as the trained DNN keeps the same.

3.2 NETWORK ESTABLISHMENT AND SETUP

In our proposed system topology, the concept of MapReduce is adopted. In our proposed MapReduce-based topology, each worker node is mapped with a part of the layer inputs and the outputs are reduced back to the GO, which generates the inputs of the new layer in the following map procedure.

In order to form a computing cluster for DNN execution, the GO enables its WiFi module to act as an AP that is prepared for responding to potential worker nodes. In the mean time, the available worker nodes with extra computing resources are searching for the GO in an opportunistic mobile network domain. Opportunistic mobile network means a form of mobile ad hoc networks that exploit the human social characteristics, such as similarities, daily routines, mobility patterns, and interests to perform the message routing and data sharing.

3.3 DEFINITION OF TERMINOLOGIES AND VARIABLES

We define the terminologies and variables that are referred to in following sections as follows:

- **Total Worker Nodes** (k)**:** Total number of the available worker nodes within the computing cluster;
- **Workload** ($W_{[i]}$)**:** The workload assigned to node i;
- **Estimated Time** ($ET_{[i]}$)**:** Estimated time for node i to execute workload $W_{[i]}$ plus data delivery time;
- **Computing Ability** ($CA_{[i]}$)**:** The normalized performance of node i, e.g., FLOPS;
- **SpMV Time** ($SPT_{[i]}(n)$)**:** Time for node i to do Sparse Matrix-Vector multiplication (SpMV) in which the matrix is represented by a linked list of size n;
- **GEMV Time** ($GET_{[i]}(r, c)$)**:** Time for node i to perform General Matrix-Vector multiplication (GEMV) in which the matrix is represented by $r \times c$ array;
- **Sparsity Threshold** ($Thld_{[i]}(r, c)$)**:** Sparsity threshold of node i that achieves equivalent computing time of the $r \times c$ matrix using SpMV and GEMV;
- **Data Delivery Time:** Data delivery time denotes the total time consumption for the data being transmitted between nodes. Data delivery time includes two parts: wakeup time and transmission time. Wakeup time represents the amount of time for the head of the data traveling from the sender to the receiver and transmission time denotes the amount of time for the receiver receiving from the first bit to the last bit of the data.

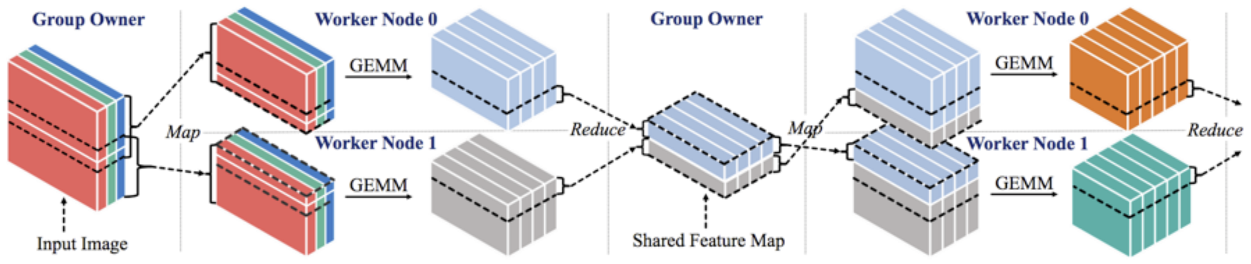


Figure 3: Processing flow of MoDNN with optimized Mobile MapReduce on two worker nodes.

3.4 CONNECTION AND REGISTRATION

In our design, after getting the permission of the user, a mobile device will be permanently trusted by the GO and automatically connected to the group when it is within the reachable WiFi range. Once connected, in addition to the device IP address, performance-related meta data are also sent to the GO for later utilization in partition schemes. The meta data includes the previously defined variables and functions like computing ability, sparsity threshold and matrix multiplication etc. The motivations to define and generate sparse matrix multiplication, general matrix multiplication and threshold function will be discussed in detail later.

3.5 DATAFLOW IN MODNN

Fig. 3 demonstrates the whole dataflow of DNN execution. Thanks to the exists excellent data affinity for the shared feature map between worker nodes, a comparatively small size of the feature map is needed for transmission due to the small kernel size of convolutional layers, which is expressed as the cubes in dashed contours in Fig. 3.

Hence, we introduce MoDNN under the concept of MapReduce as the distributed architecture with several re-designed details for the low-level processing flow optimized to our DNN execution scenario. Fig. 3 illustrates our optimized data processing flow for two worker nodes under the guidance of the characterization in distributed DNN execution. GO first partitions the input image and maps them to each the worker node. Then, after the computation of the convolution operation, each worker node reduces the shared part of the output feature map back to GO by key-value pairs. Finally, the GO gathers the output from the worker nodes and maps them back to the worker nodes so that the they can combine the received data together with their local output feature map in order to create the input feature map for the computation of next layer. Concretely, the main optimizations include:

3.5.1 In-memory Weights

Concerning the response time and relatively small data size, in MoDNN, the intermediate data are not stored on local disk. Instead, the overlapping part of the output feature map, namely the intermediate data, will be immediately reduced from the worker nodes to GO for next map procedure as shown in the reduce procedure in Fig. 3. Each weight of the DNN model is indexed by a specific number so that it can be accessed fast by hashing. Such scheme actually tradeoffs the execution time with the system robustness. However, because of the comparative small execution granularity, the whole system can fast recover from the potential system failure.

3.5.2 Flexibility

The worker nodes in distributed mobile system is dynamically changing, calling for a partition scheme that can adaptively fit all the situations with different worker nodes. Due to the mobility of the system, MoDNN can dynamically detect the changes in the list of existing worker nodes and adjust the scheduling and partitioning method accordingly. Therefore, as will be shown in the following sections, we partition each layers with our proposed scheme dynamically according to the total available nodes in the computing cluster.

3.5.3 Sweet Spot for the Worker Node

The total execution time does not decrease linearly with the increase of worker nodes because of the communication overhead. When the number of worker nodes increases, more separate communication channels will be established, leading to narrower communication bandwidth available for each worker node in the computing cluster.

Hence, the optimal number of worker nodes will be chosen to avoid the performance degradation caused by network congestion. More specifically, by estimate the computing time based on the available resources, our system can predict an optimal scenario for distributed DNN execution.

4.0 INPUT PARTITION FOR CLS

4.1 CONVENTIONAL PARTITION SCHEME

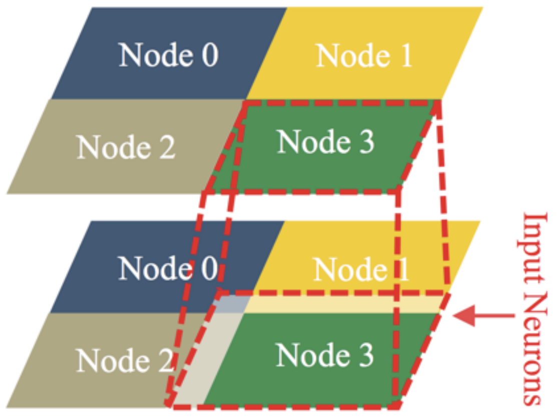


Figure 4: Neurons in 2D-grids of 4 nodes.

Conventional partition schemes of CLs on other platforms usually maintain a structural symmetry for the layer inputs. For example, in [3], Coates arranged a GPU cluster into 2D-grids and partitioned the input neurons along the two-dimensional space, as shown in Fig. 4.

However, such a two-dimensional partition may not be suitable for the proposed local distributed mobile computing system. Unlike in the GPU cluster, the wakeup time, rather than the transmission time, dominates the data delivery time in MoDNN. The time interval between states, determined in both previous research works and in our own experiments, is significantly greater than transmission time itself. It is because of the Opportunistic Power Save Protocol that support the sleep mode of the clients: If a mobile device has not been used for a certain time period, it will turn off its radio modules automatically [1]. Turning on the radio modules and establish the transmission channel takes a time period significantly longer than the data transmission itself. This comes to be the most critical implementation obstacle and bottleneck, which calls for a partition scheme including minimal number of transmission channels to be established.

4.2 PROPOSED 1-D PARTITION

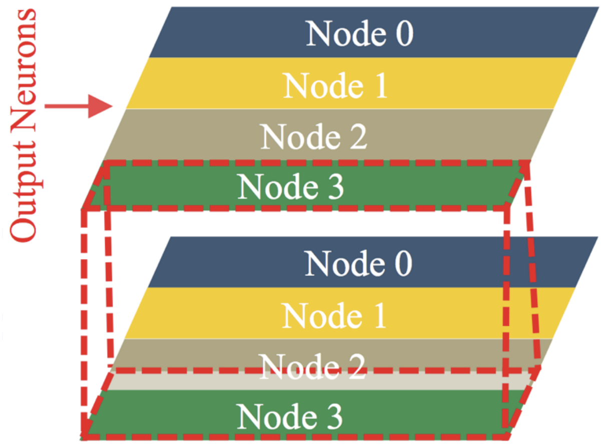


Figure 5: BODP for 4 worker nodes.

In MoDNN, BODP is proposed to partition the input neurons along the longer edge of the input matrix according to the computing abilities of individual node, as illustrated in Fig. 5. Using node3 as an example: the input of node3 overlaps all the other three nodes in the 2D partition in Fig. 4, while the input of node3 in Fig. 5 only overlaps with that of node2. Note that only the overlapped parts of the layer inputs need to be transferred during the computation. The size of the overlapping part can be formulated as:

$$W_{trans} = \text{Min}(H, W) \cdot C \cdot (F - 1) \cdot Z. \tag{4.1}$$

Eq. 1 expresses the amount of bits to be shared between two worker nodes before the execution of each convolutional layer. For a convolutional layer, we define F and C as the kernel size and the channel size of the filters sliding across the feature map of size H W and there are K kernels in total. Because the filter size F is relatively small in almost all the mainstream DNN models. The transmission amount is thus not so much. To achieve this, BODP utilizes the tradeoff between the transmission time and the propagation delay.

Since the wakeup time in MoDNN is greatly impacted by the number of the established transmission channels, reducing the number of the neighbor nodes from 4 (in conventional 2D partition) to 2 (in BODP) will effectively minimize the associated high propagation delay. More analysis on the effectiveness of BODP can be found in the following section.

5.0 WEIGHT PARTITION FOR SPARSE FLS

5.1 TWO APPROACHES FOR MATRIX MULTIPLICATION

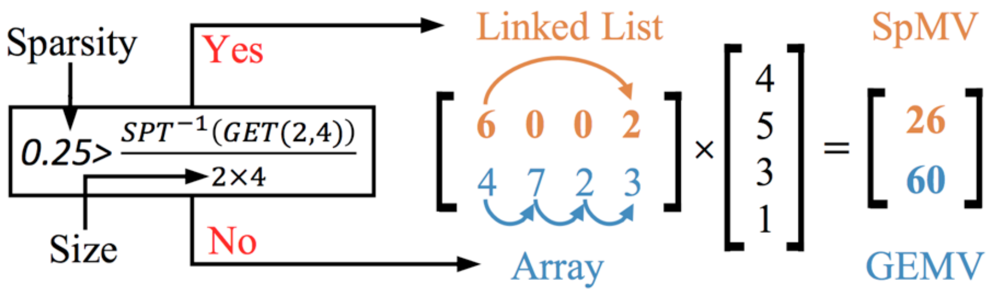

Sparsity
Yes
$0.25 > \frac{SPT^{-1}(GET(2,4))}{2\times4}$
Size
No
Linked List
Array
SpMV
GEMV
$\begin{bmatrix} 6 & 0 & 0 & 2 \\ 4 & 7 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 4 \\ 5 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 26 \\ 60 \end{bmatrix}$

Figure 6: Hybrid matrix representation for SpMV and GEMV.

The partition scheme of FLs in MoDNN targets the state-of-the-art sparse FLs. Because of the comparatively short execution time of FLs, mobile devices will keep the wireless radio in an active state and the transmission time dominates the data delivery time. The object of the proposed partition scheme is to reduce the size of the data to be transmitted for the reduction of the transmission time.

There are two approaches to compute matrix-vector multiplication, which is the main operation in DNN: General Matrix-Vector multiplication (GEMV) and Sparse Matrix-Vector multiplication (SpMV). GEMV is usually used to compute a dense matrix which often uses arrays to represent the data while SpMV is effective in computing a sparse matrix that can be efficiently stored in a linked-list, as illustrated in Fig 6. Modern machine learning platforms use array to represent matrix, which takes the advantage of low-level optimization such as Single Instruction Multiple Data (SIMD), cache optimization and multi-threads support. However, for sparse layers, high proportion of zeros are left in array structure, costing tremendous meaningless calculations and extra storage spaces. An alternative way is to adopt linked-list structure for sparse matrix multiplication, which converts each row of the original matrix to index-value pairs of non-zeros indexed by column number so that no redundant space or computing resources are cost.

The selection of the appropriate data representation can be decided by comparing the target matrix sparsity with threshold function. Here sparse matrix multiplication and general matrix multiplication are the time spent on the computation of the matrix-vector multiplication using SpMV and GEMV, respectively. They can be obtained from real measurements on the mobile devices via a linear regression method. When the sparsity of the matrix is larger than the threshold, SpMV will be used for the computation; otherwise, GEMV will be applied.

5.2 MODIFIED SPECTRAL CO-CLUSTERING (MSCC)

5.2.1 Proposed partition scheme

We note that GEMV is more computationally efficient per matrix element than SpMV due to its higher computing parallelism. Hence, it will be beneficial to partition the weight matrices onto the worker nodes in a dense structure.

In the weight partition scheme of FLs in MoDNN, a clustering algorithm is leveraged to group the nonzero weights into several clusters and minimize the number of the nonzero weights outside the clusters. If we consider the weight matrix as an undirected graph, generating k clusters with minimal connections between them is a NP hard problem [6]. In MoDNN, we use spectral clustering technique to find the solution heuristically.

Spectral clustering technique is widely used in graph partition problems, aiming at minimizing between-cluster similarities [16]. In MoDNN, the sparse FLs are treated as undirected graphs where the graph vertices represent the input and output neurons and the edges represent the network weights. Hence, we redefine the similarity in spectral clustering technique as the number of between-clusters connections. Hence, the input neurons corresponding to these clusters are first transmitted for parallel execution.

However, traditional spectral clustering technique works only on a matrix with the same row and column size, which greatly limits its applicability and scalability in DNN computations. Therefore, spectral co-clustering algorithm is introduced to address this drawback by normalizing the original connection matrix A to Anorm and performing Singular Value Decomposition (SVD) on Anorm [5]. Here the elements of weight matrix A are binary where '1' represents a connection between two neurons and '0' otherwise.

The Spectral clustering algorithm is commonly used in graph partitioning problems, targeting at minimizing between-cluster similarities [5]. This algorithm converts r rows and c columns of the original matrix A to a matrix Z of r+c rows using the results generated in obtaining Anorm. Each column of Z is an eigenvector of A so that we can cluster matrix together based on the rows of Z. Then, we apply appropriate data structure to each cluster based on their sparsity for computing time reduction. We name this clustering procedure as modified spectral co-clustering (MSCC).

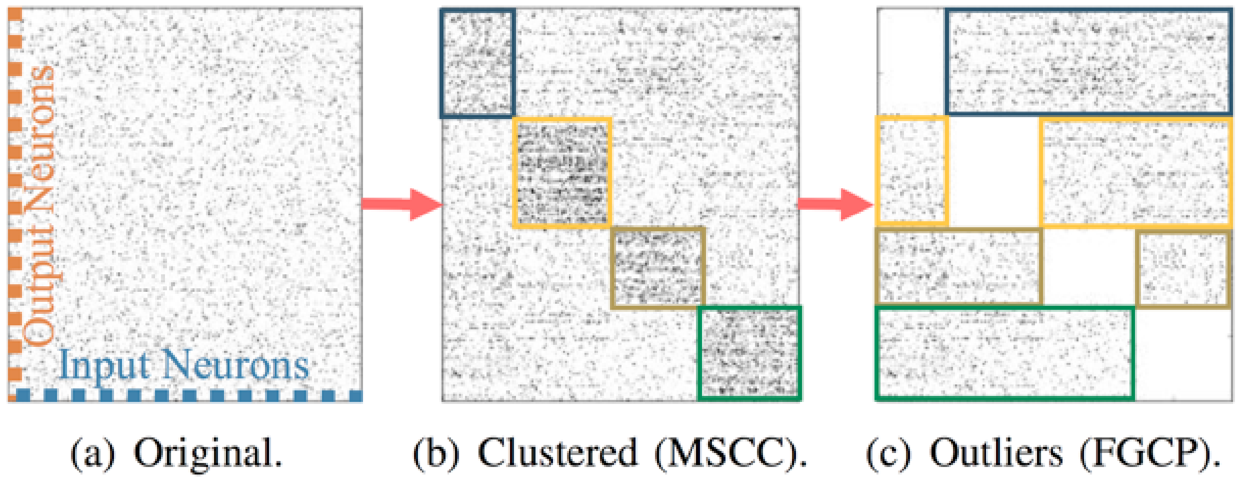


Figure 7: Two-stage processing: MSCC&FGCP.

5.3 FINE-GRAIN CROSS PARTITION (FGCP)

Spectral co-clustering focuses on reducing only the external connectivity between the clusters without considering the internal cluster density. To solve this problem, we propose FGCP to partition the remaining outliers in the weight matrix after MSCC to balance the workloads between the GO and the worker nodes. The basic idea here is to identify the sets of the weights with minimal number of nonzero elements and keep them computed on the GO rather than sending to the worker nodes to avoid the high cost introduced by the long data delivery time.

For the sparse outlier matrix shown in Fig. 7(c), for example, since the number of its columns is smaller than its rows, FGCP initially assigns the elements on the same rows where the cluster (obtained in MSCC) resides to node i. Then FGCP iteratively finds the worker node i with the maximum ET[i] and offloads the initially assigned weights on the same column in the outlier matrix with the minimal number of non-zero elements from the worker node i to the GO. In addition, FGCP needs to consider the discrepancy of execution time between the GO and the worker nodes during the offloading process, especially the data delivery time on the network between the worker node x and the GO, which can be conceptually formulated by:

$$Initial\_ET(x) = \frac{(\sum_{i=0}^{x} C_{column[i]} + \sum_{i=x}^{k} C_{row[i]})}{TPT}, \tag{5.1}$$

where TPT is the mobile network throughput; the remaining parts describe the total non-overlapping data size of the input and output neurons to be transmitted during the execution. Obviously, the more sparse the outlier matrix is, the more elements can be possibly offloaded to the GO to balance the workload.

6.0 MODEL PROCESSOR

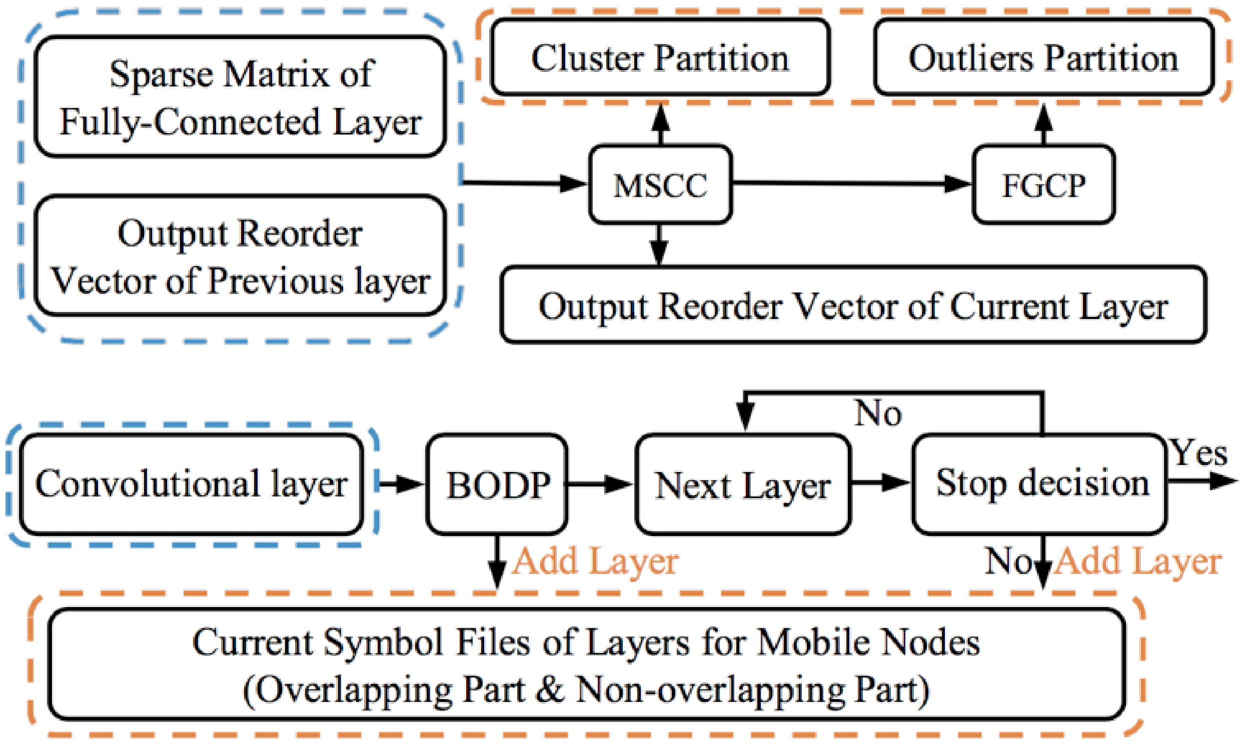

Sparse Matrix of Fully-Connected Layer
Output Reorder Vector of Previous layer
Cluster Partition
Outliers Partition
MSCC
FGCP
Output Reorder Vector of Current Layer
Convolutional layer
BODP
Next Layer
Stop decision
No
Yes
Add Layer
No
Add Layer
Current Symbol Files of Layers for Mobile Nodes (Overlapping Part & Non-overlapping Part)

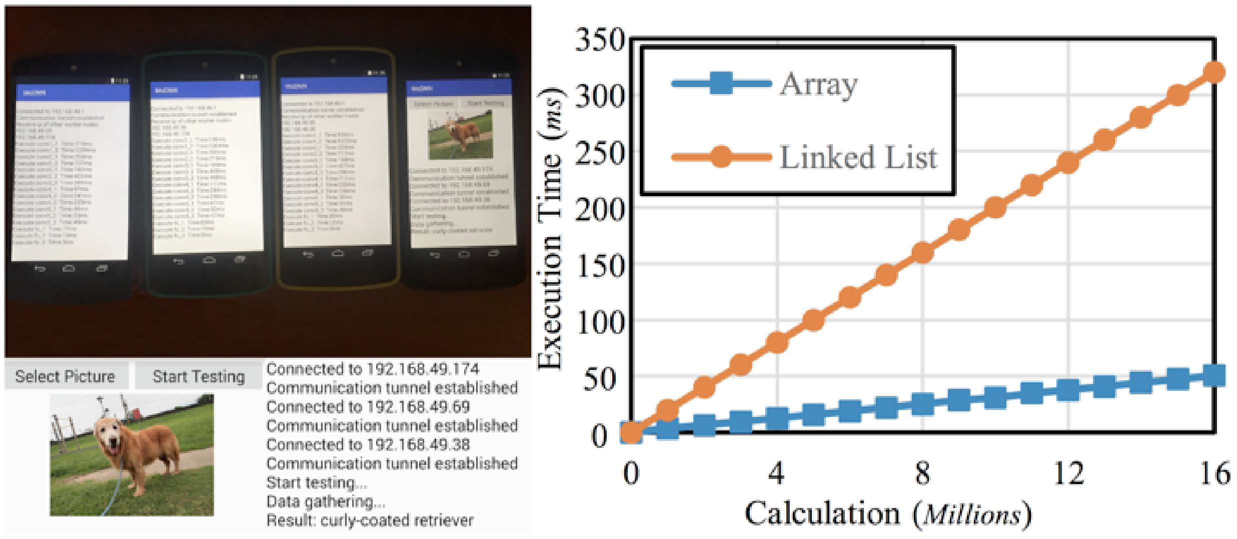
Figure 8: Two processing flows in model processor of MoDNN.

Fig. 8 depicts the processing flows of CLs and FLs in MoDNN and how the two parts are integrated. Given a trained DNN, the model processor scans each layer and identify their type. If a CL is detected, the layer?s input will be partitioned by BODP into small pieces, which are then combined with the subsequent non-overlapping layer structures e.g., ReLu layers, pooling layers, normalization layers, etc. for computation. If a sparse fully-connected layer is detected, MSCC and FGCP will be applied in sequence to assign the workloads to the worker nodes in clusters and the workloads for outliers, respectively, in order to achieve the minimum total execution time.

In the developing framework adopted in this work (MXNet), which can be spelled as mix net or max net, the files that contain the model structure information is suffixed with .json . JavaScript Object Notation (JSON) is a way to store information in an organized, easy-to-access manner. In a nutshell, it gives us a human-readable collection of data that we can access in a really logical manner. On the other hand, the files consist the weight information is formatted as .params , which is a self-defined file structure by MXNet. We can read the contents of both file types through Python interface.

7.0 EXPERIMENTS

7.1 ENVIRONMENT SETUP AND TESTBENCH SELECTION



```
Connected to 192.168.49.174
Communication tunnel established
Connected to 192.168.49.69
Communication tunnel established
Connected to 192.168.49.38
Communication tunnel established
Start testing...
Data gathering...
Result: curly-coated retriever
```

Figure 9: Two processing flows in model processor of MoDNN.

The implementation of MoDNN is based on MXNet, which is a deep learning framework developed by the Distributed Machine Learning Community (DMLC) team for desktop platform using C++. We modified and recompiled the MXNet libraries so that it can support Android systems with ARM architecture through JAVA Native Interface (JNI) [19].

We adopt a pre-trained DNN model from ImageNet database: VGG-16 [4], as the testbench in our experiments. VGG is a popular and clear-in-structure Convolutional Neural Network (CNN) model that includes all mainstream layer types so that the significance of each component of

MoDNN can be distinctly evaluated. In our experiments, VGG are executed locally or distributed to different numbers of worker nodes by MoDNN; the adopted mobile devices are LG Nexus 5 running Android 4.4.2 with a 2.28 GHz processor and 2GB RAM.

The experiment setup is depicted in Fig. 9. Fig. 9 also presents the characterized results of SPT[i](n) and GET[i](r,c), which are two important parameters used in the partition schemes of MoDNN. Linked list and array structures are used in characterizing SPT[i](n) and GET[i](r,c), respectively. Here x-axis denotes the amount of computations, i.e., n non-zeros for SPT[i](n) and matrix for GET[i](r,c), respectively.

The results show that the calculation time of the worker nodes is proportional to the calculation number. For the same workload, SpMV is much slower than GEMV. SpMV only taken as priority when the matrix sparsity is below approximately 15.8%. Hence, we set Thld[i](r,c) to 15.8% in our scheme. The measured average WLAN wakeup time and transmission throughput are 54.7ms and 43.8Mbps, respectively.

7.2 DATA DELIVERY TIME EVALUATION OF BODP

The bars in Fig. 10 show the computing times of 13 CLs in VGG-16 during testing phase, excluding the data delivery time. The results of running locally and on 2, 3, and 4 worker nodes in MoDNN are depicted. For comparison purpose, the results of using conventional 2D-grids partition scheme for 4 worker nodes is also included in the figure. When the number of the worker nodes increases, the execution time of each CL keeps reducing, proving the effectiveness of MoDNN in parallel computing.

The results of BODP with 4 worker nodes and 2D-grids partition are very close, implying little impact of the input shapes of the CLs on the computing time. As also illustrated by the dot lines in Fig. 10, compared to 2D-grids partition, BODP also slightly increases the average data transmission size of each CL from 41048 bytes to 59856 bytes and hence, increases the average transmission time from 7.15ms to 10.43ms. Such a close time consumption in execution of CLs diverts our focus to the data delivery time. Nonetheless, when taking into account that the total wakeup time contribute to approximately 30% of the total data delivery time in each data shar-

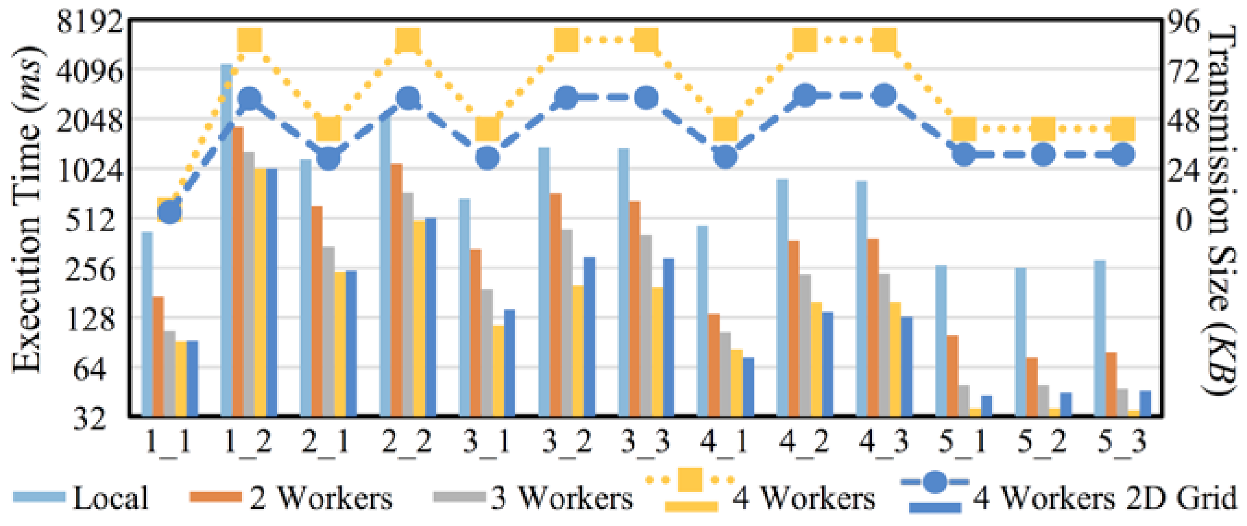


Figure 10: Computing time and transmission time of CLs in VGG-16.

ing procedure. If we take into account the contribution from the data transmission time, BODP still achieves shorter total data delivery time than 2D-grids partition for 4 worker nodes as less transmission channels need be established.

7.3 TRANSMISSION SIZE EVALUATION OF MSCC & FGCP

In order to evaluate the effectiveness of MSCC and FGCP on large-scale, sparse FLs, the FLs in VGG-16 are sparsified by L1-norm group lasso with a predefined discarding threshold to control the sparsity.

As our proposed partitioning scheme is specifically utilized for sparse FLs, we restrained the sparsity of the layers within 70% to 96%. Fig. 11 shows the transmission size decrease ratio of all the three FLs in *VGG*-16 of different sizes compared with the baseline implementation. We define the evaluation baseline as the one-dimensional partition that divides the weight matrix along its longer side without any overlaps between the partitioned parts.

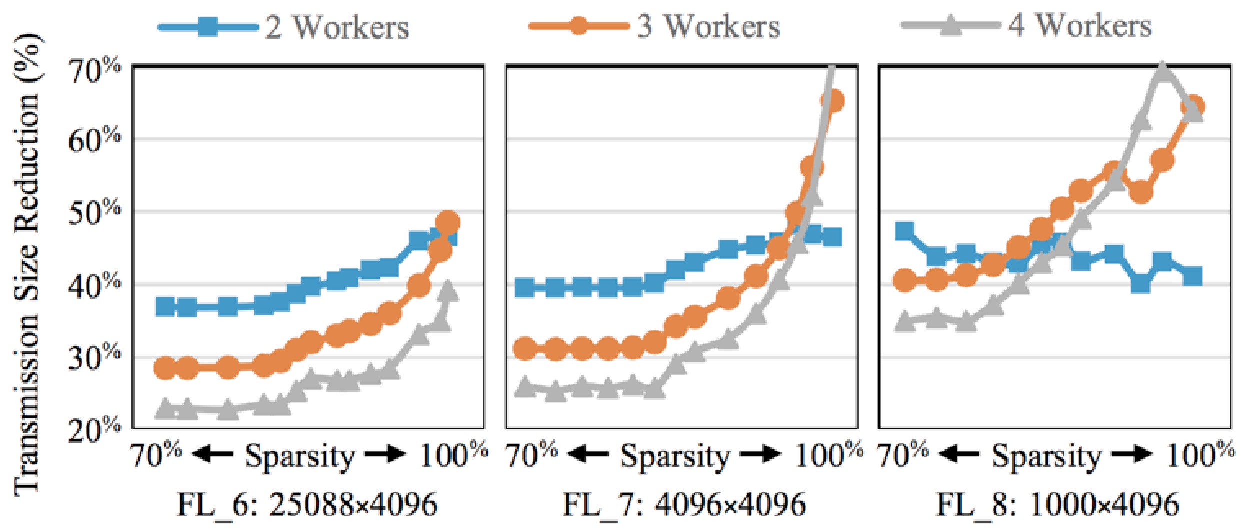


Figure 11: Two processing flows in model processor of MoDNN.

As our proposed partitioning scheme is specifically utilized for sparse FLs, we restrained the sparsity of the layers within 70% to 96%. Fig. 11 shows the transmission size decrease ratio of all the three FLs in *VGG*-16 of different sizes compared with the baseline implementation. The results are normalized by the one of the baseline, as shown in Fig. 11. According to the results, MSCC and FGCP effectively reduce the transmission size by at least 22.6% compared with the baseline implementation. In most cases, the transmission size reduction ratio keeps increasing with the layer sparsity, e.g., reaches as high as 49.3%, 69.2%, and 69% for FC6, FC7, and FC8, respectively at different numbers of worker nodes. One exception occurs in the FC8 with 2 worker nodes, which shows a decrease in the transmission size reduction ratio when the layer sparsity increases. It is because of the residual unbalance in the clustering due to the limited solution space of the small-scale layers (e.g. 1000 times 4096 for FC8). Nonetheless, following the increase of the number of the worker nodes, the effectiveness of MSCC and FGCP also increases, demonstrating a good design scalability. Moreover, with the layers being more sparse, the clustering algorithm comes more effective, which leads to the acceleration of communication decrease with more worker nodes converging to a high decrease ratio.

7.4 OVERALL EVALUATION OF MODNN

Table 1 summarizes the overall execution time to compute the whole V GG-16 model in DoDNN over different numbers of mobile devices. Following the increase of the number of the worker nodes, the overall execution time reduces significantly, demonstrating excellent computing parallelism: the purely computation time improves by 2.17-4.28 with 2 to 4 worker nodes. Table I also summarizes the data delivery time and the data transmission size of different scenarios, which indicates the extra cost introduced by the distributed computing mechanism of MoDNN. MoDNN also outperforms the conventional 2D-grids partition scheme by substantially reducing the data delivery time though the data transmission size is slightly increased.

Table 1: Overall evaluation of MoDNN with 2-4 worker nodes.

	Execution Time (ms)	Data Delivery Time (ms)	Transmission Size (KB)
Local	15809	0	0
2 Workers	8509	1819	1196
3 Workers	6884	2563	2257
4 Workers	5208	2567	3336
4 Workers 2D-grids	6324	3073	2256

8.0 CONCLUSION

In this work, we propose MoDNN–a local distributed mobile computing system to enable parallel computation of DNN on mobile platforms. As convolutional layers and fully- connected layers are identified as the major DNN components that contribute to the total execution time, several advanced partition schemes, i.e., BODP, MSCC, and FGCP are pro- posed to well balance the workloads of each worker nodes and minimize the data delivery time. Experiments show that MoDNN can achieve better than linear performance speedup on DNN computations, demonstrating great potential of mobile platforms in DNN applications.

BIBLIOGRAPHY

[1] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with Wi-Fi Direct: overview and experimentation," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 96–104, 2013.

[2] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," *Computing Research Repository*, 2015.

[3] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with cots hpc systems," *International Conference on Machine Learning*, vol. 20, no. 3, pp. 96–104, 2013.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, pp. 107–113, 2008.

[5] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[6] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete graph problems," *Theoretical Computer Science*, vol. 1, no. 3, pp. 237–267, 1976.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *Computing Research Repository*, vol. 2, 2015.

[8] J. Hauswald, T. Manville, Q. Zheng, R. Dreslinski, C. Chakrabarti, and T. Mudge, "A hybrid approach to offloading mobile image classification," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2014, pp. 8375–8379.

[9] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Computing Research Repository*, 2015.

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in

11th USENIX Symposium on Operating Systems Design and Implementation, 2014, pp. 583–598.

[12] S. Li, C. Wu, H. Li, B. Li, Y. Wang, and Q. Qiu, “Fpga acceleration of recurrent neural network based language model,” in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*. IEEE, 2015, pp. 111–118.

[13] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, “Planet: massively parallel learning of tree ensembles with mapreduce,” *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1426–1437, 2009.

[14] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *Computing Research Repository*, 2014.

[15] L. Song, X. Qian, H. Li, and Y. Chen, “Pipelayer: A pipelined reram-based accelerator for deep learning.” HPCA, 2017.

[16] U. von Luxburg, “A tutorial on spectral clustering,” *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[17] W. Wen, C. R. Wu, X. Hu, B. Liu, T. Y. Ho, X. Li, and Y. Chen, “An eda framework for large scale hybrid neuromorphic computing systems,” *ACM/IEEE Design Automation Conference*, pp. 1–6, 2015.