### System Model and Problem Definition

#### The Configurable Neural Network Accelerators Model

Neural network accelerators use exquisitely designed function units to leverage neural network features. According to operational characteristics, neural network operations can be divided into several types. For instance, Convolution and Full Connection Operations can be accomplished by matrix-matrix multiplication and reduce sum operation, Activation Operations can be accomplished by lookup table, and Binary/Unary Operations like Scale/batch normalization can be accomplished by vector-vector operations. By implementing corresponding types of operations, Neural network can accelerate neural network computation.

Heterogeneous systems often use NUMA or UMA to pass messages. Different to heterogeneous systems, function units can communicate with other function units directly without the interpretation of DRAM. With regard to neural network applications, dataflow for each neural network layer can be consumed directly by the other layers, which don’t need to be buffered.

With different application scenarios, there are different demands for Neural Network Accelerators. For edge computing fields, power consumption and time delay are main focuses. While throughput and parallelism are the main ones for cloud servers. In our models, the number of function units, computational speed for each function unit, the connectivity for each function unit pair and the corresponding bandwidth are all configurable parameters.

**A Figure to show a NNA is needed here!**

Basically, a Neural Network Accelerators Model consists of series of function units and inter-connected data transfer paths. Each function unit could accomplish several types of neural network operations. For example, since convolution and full connection are matrix-matrix multiplication and reduce sum operation, they could be accomplished by a class of function units, which depends on the design of function unit. The interconnected data transfer paths between every two function units could exist or not. The separation of NN operations, computational speed for each operation on each computation unit and the IO bandwidth between computation units are all configurable parameters.

Assume neural network operations are divided into m sets, , , …, . Suppose a configured accelerator consists of function units, ,, …, . For function unit , we use the function to represent the computation speed for function unit .

.

Also, a bandwidth function is used to represent the bandwidth between and .

#### Formulation of the Scheduling Problem

The neural network application could be represented by a directed acyclic graph(DAG), . Each node represents an operation and an edge represents the data dependence between two operations.

Suppose a DAG application with layers, we have the following formulation:

Assume the operation type for node is, computation size for node is, and the selected function unit for . Then the computational time for node which could be reckoned out as follows:

Assume the data transmission size for node to node is , and the corresponding transmission time can be defined by

we introduce the conception of priority, nodes in each function unit. The priority of node is . Nodes on the same function unit should be executed orderly by their priorities.

We assume the start time of node is, and the finish time is . Then we have the follows equations:

Then the schedule problem is to find a function-unit assignment function and priority setting function to minimize the execution time.

#### The Partition Associated Scheduling Problem

Most neural network operations can be accomplished by matrix-matrix, matrix-vector, vector-vector operations. Based on the high parallelizable character, neural network can be partitioned to leverage operation parallelism.

Focusing on each operation’s implementation, we formulated partition methods respectively. For example, based on the partition direction, a 2-D convolution operation can be partitioned by 5 types. From the batch direction, no additional works involved, but if the batch number of application equals 1, this method wouldn’t work. From the input channel direction, an additional add operation is needed to add partial results from each partitioned child nodes. From the output channel direction, each sub-node need to get full input, which will increase data transfer size. From the height or width direction, there could be additional data transfer consumption for overlapped inputs on each child nodes. For a batch normalization operation, partition wouldn’t lead into additional workload. Similarly, we construct partition rules for each neural network operation.

Figures needed here!

For a partition procedure, a node would be replaced by new nodes and edges associated with would be replaced by new edges, and edges and the data transfer size on these edges is tightly interrelated with operation and parameters on it.

Assume we get a partition sequence

The partition associated schedule problem could be reformulated as: