Partition and Schedule Algorithms for Neural Network Accelerators

### Abstract-TBD

### Introduction-TBD

### 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:

### Partition and Scheduling Algorithms

#### Transplanted Scheduling Algorithms

Different with heterogeneous systems, function units of neural network accelerators couldn’t support all types of operations and the traditional algorithms do not suit such kind of scenarios any more.

We transplanted Heterogeneous-Earliest-Finish-Time (HEFT) and Critical-Path-on-a-Processor (CPOP) algorithms to adopt neural network applications. The NN-HEFT algorithm firstly sets the priority of each node with the upward rank value, , which is based on mean computation and mean communication costs. For a specific node with an operation type, we select function units which support such kind operation to get the upward rank.

In the original CPOP algorithm, nodes in the critical path use the same processor to eliminate data transmission cost. But in the neural network accelerators model, we extend that restriction that nodes with the same type of operations must be in the same function unit.

#### The Iterative Partition-Scheduling Algorithm

To leverage the parallelism of function unit, we try to introduce the partition procedure to improve the result of each scheduled result. The Iterative Partition-Scheduling algorithm is an algorithm framework that is suitable for all kind of scheduling algorithms. In this framework, we firstly run the scheduling algorithm to get a critical path for the application. Then, we try to partition a node in the path to minimum the scheduling time, which can get a local optimal partition node. By iteratively schedule and partition the graph, the origin topology will be split to much more easily to schedule. The graph of next iteration is the partition result of current iteration, which provides the probability to jump out of the locally optimal solution.

If the schedule time of current iteration is larger than the previous iteration, it is persuasive that there are chances to have a better scheduling result. And in order to jump out of the locally optimal solution, there exist a iteration threshold to make sure there are at least iteration threshold times of partition.

|  |
| --- |
| The Iterative Partition-Scheduling Algorithm  1: set optimal make span of G(V, E)  2: set iteration number  3: do  4: schedule G(V, E) by a specific algorithm to get makespan  5: find the critical path of scheduled graph  6: set the gain of splitting a node  7: set the target node to split is  8: for node in the critical path  9: try to split and estimate gain .  10: update gain of split node and critical node  10: end for  11: split G(V, E) to G’(V’, E’) by splitting critical node evenly.  13: update G(V, E) by G’(V’, E’) ,, ) and increase by 1  14: while or |

In every time of trying to partition a node, we can get the number of partition number by the available function unit.

|  |
| --- |
| Partition number decision algorithm for a node  1: set partition number  2: for  3: if  4: if  5:  6: end if  7: end if  8: end for |

#### The Partition-Schedule-Combine Algorithm

The Partition-Schedule-Combine algorithm is a three stages algorithm. In the first stage, we partition the node in the original graph into child nodes evenly, and the number of child nodes equals to the number of function units that support the operation of the father node. In the second stage, we schedule the updated graph by a specific scheduling algorithm. In the last stage, we try to combine child nodes in the same function units to decrease effort of the scheduling algorithm.

To leverage the parallelism of neural network application, we use a partition procedure to split each node into child nodes. Intuitively, these child nodes could be scheduled into different function units and be executed parallelly. Since the partition procedure could incur additional data transmission or computational cost, a node combination procedure is introduced to combine separated nodes to erase unnecessary cost.

|  |
| --- |
| The Partition-Schedule-Combine algorithm  for each node in G(V, E)  get the number if function units that support operation type of  split into parts evenly.  end for  update G(V, E) to G’(V’, E’).  Schedule G’(V’, E’)  for each function unit  try to combine nodes in function units.  end for |

### Experiments and Analysis

In this section, we present the schedule results of transplanted scheduling algorithms and the results of our two partition-associated scheduling algorithms. To show the effect of our designment, we chose several typical neural networks and some auto-generated networks as the input. Then we show the result of scheduled algorithms. Finally, we analyze results of each algorithm and explain the reasons.

#### Experimental Setup

We implement our experiment on a simulator, which is equipped with 6 CPUs, which has 8GB memory, working at 3 GHZ, and the CPU model is Intel(R) Core(TM) i5-8500.

We use two kinds of neural networks, typical neural networks for real world applications and randomly generated neural networks. For existed neural network models, we select Lenet, Alexnet, vgg16, vgg19, googlenet, inceptionv3, resnet18 and resnet50 as our benchmark. And for the auto generated neural networks, we can adjust the layer types, branches and some other parameters that we are interested in.

#### Experimental Results

As for the experiment, we use the following three metrics to show efficiency of our algorithms.

Makespan: execution time of the scheduled neural network.

Speedup: For a given neural network, the ratio of makespan of the scheduled neural network to the minimum sequential execution time. In our model, the sequential execution time is computed by assigning one kind of operations to a function unit that minimizes the cumulative time of computation and data transmission costs.

Schedule Length Ratio(SLR): this metric is mainly used to describe character of different neural networks. SLR is computed is normalized to a lower bound by reckon the ratio of the makespan to the computation costs on the critical path.

#### ~~Analysis of Evaluations~~

### Conclusions and Future Work

With the rapid development and applications of neural networks, from the design of neural network algorithms to accelerators, many efforts have been made to improve the execution efficiency. In our work, we migrate heterogeneous processors scheduling algorithms HEFT and CPOP to suit the model of neural network. To utilize parallelism of neural network accelerators, we combine methods of partitioning and scheduling. Through the experimental result of each algorithm on real world and randomly generated neural network applications, the Iterative Partition-Scheduling algorithm performs well in linear neural networks and the Partition-Schedule-Combine algorithm performs well in multi-branch networks. Besides, the partition scheduling frameworks can be associated with different scheduling algorithms which shows the advantage of efficient and scalability.

In our future work, we will take hierarchical memory organizations into consideration and try to guide the design of neural network accelerators by the scheduling results.

**Acknowledges** This work is partially supported by the National Key Research and Development Program of China (under Grant 2017YFA0700900, 2017YFA0700902, 2017YFA0700901，2017YFB1003101), the NSF of China (under Grants 61432016, 61532016, 61672491, 61602441, 61602446, 61732002, 61702478, 61732007 and 61732020), Beijing Natural Science Foundation (JQ18013), the 973 Program of China (under Grant 2015CB358800), National Science and Technology Major Project (2018ZX01031102), the Transformation and Transfer of Scientific and Technological Achievements of Chinese Academy of Sciences (KFJ-HGZX-013) and Strategic Priority Research Program of Chinese Academy of Science (XDB32050200，XDC01020000).

### References

Topcuoglu H, Hariri S, Wu M. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE transactions on parallel and distributed systems, 2002, 13(3): 260-274.

Augonnet C, Thibault S, Namyst R, et al. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures[J]. Concurrency and Computation: Practice and Experience, 2011, 23(2): 187-198.

Baruah S K. Task Partitioning Upon Heterogeneous Multiprocessor Platforms[C]//IEEE real-time and embedded technology and applications symposium. 2004: 536-543.

Ijaz S, Munir E U, Anwar W, et al. Efficient scheduling strategy for task graphs in heterogeneous computing environment[J]. Int. Arab J. Inf. Technol., 2013, 10(5): 486-492.

Hakem M, Butelle F. Dynamic critical path scheduling parallel programs onto multiprocessors[C]//19th IEEE International Parallel and Distributed Processing Symposium. IEEE, 2005: 7 pp.

Yang C H, Lee P Z, Chung Y C. Improving static task scheduling in heterogeneous and homogeneous computing systems[C]//Proceedings of the 2007 International Conference on Parallel Processing. IEEE Computer Society, 2007: 45.

Kwok Y , Ahmad I . Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors[J]. Acm Computing Surveys, 1999, 31(4):406-471.

Eswari R , Nickolas S . Path-Based Heuristic Task Scheduling Algorithm for Heterogeneous Distributed Computing Systems[C]// International Conference on Advances in Recent Technologies in Communication & Computing. IEEE Computer Society, 2010.

Daoud M I , Kharma N . A high performance algorithm for static task scheduling in heterogeneous distributed computing systems[J]. Journal of Parallel and Distributed Computing, 2008, 68(4):399-409.

Wang C , Gu J , Wang Y , et al. A Hybrid Heuristic-Genetic Algorithm for Task Scheduling in Heterogeneous Multi-core System[C]// International Conference on Algorithms and Architectures for Parallel Processing. Springer, Berlin, Heidelberg, 2012.

Jing-Mei L I , Xue W , Qi-Long H . A Heuristic Integrated Task Scheduling Algorithm for Heterogeneous Multi-core Processor[J]. Computer Engineering, 2014.

Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

Chen Y H, Krishna T, Emer J S, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks[J]. IEEE Journal of Solid-State Circuits, 2017, 52(1): 127-138.