Work-in-Progress: Fast Generation of Optimized Execution Plans for Parameterizable CNN Accelerators

Hyemi Min
hyemi@csap.snu.ac.kr
Seoul National University
Seoul, South Korea

Jungyoon Kwon
jungyoon@csap.snu.ac.kr
Seoul National University
Seoul, South Korea

Bernhard Egger
bernhard@csap.snu.ac.kr
Seoul National University
Seoul, South Korea

ABSTRACT

Generating an optimal execution plan for a given convolutional neural network (CNN) and a parameterizable hardware accelerator is a challenge. We present a framework that finds an execution plan that maximizes throughput for a given network and a specific configuration of our parameterizable accelerator. The framework first generates tiled dataflows for each layer, then maps the dataflows to the different independent hardware units using techniques borrowed from traditional list scheduling. Evaluated with a number of different networks and different hardware configurations, the presented framework clearly outperforms existing approaches in terms of speedup or schedule generation time.

KEYWORDS

Neural network accelerators, operation scheduling

ACM Reference Format:
Hyemi Min, Jungyoon Kwon, and Bernhard Egger. 2021. Work-in-Progress: Fast Generation of Optimized Execution Plans for Parameterizable CNN Accelerators. In *Proceedings of the 2021 International Conference on Compilers, Architectures and Synthesis for Embedded Systems Companion (CASES'21 Companion), October 8–15, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3451939.3477593

1 INTRODUCTION

Thanks to excellent performance in various application domains, CNNs have received an unprecedented amount of attention. In the quest for higher accuracy, networks continue to grow more complex. At the same time, execution of CNNs on mobile and edge devices is becoming more important [1, 4]. To accommodate for different markets, embedded accelerators are often parameterizable with respect to compute array and the on-chip memory.

To achieve high throughput, the utilization of computational and memory units must be maximized and memory transfers need to overlap with computation [2, 5]. Since the available on-chip memory cannot hold all required data at once, the computation of a layer split into smaller sub-tasks through a task known as *tiling*. The network layer, the data layout, and the order of computation determine the order of operations and which data chunks needs to be brought on/off-chip at what times. This leads to a large number of possible executions with which a layer can be computed. Finding the best configuration a given CNN and specific accelerator in this large optimization space is a tedious and error-prone task.

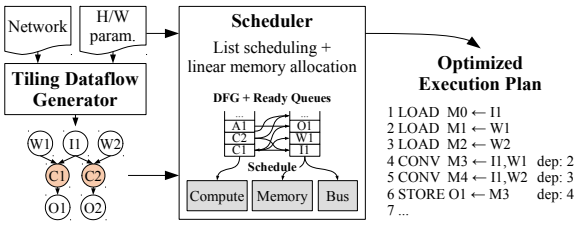

Figure 1: Generation of optimized execution plans.

In this paper, we present our ongoing work on a compiler framework that facilitates the task of finding an optimized execution plan for a given CNN and our parameterizable accelerator (Figure 1). The inputs to the framework are the structure of a network and hardware parameters of the accelerator. These parameters include the compute array (size and shape, supported data layouts), the on-chip memory (size and organization, single vs. dual ported), and the buses available for DMA transfers. For each layer, the framework generates a number of tiled data flow graphs (DFG) that represent the computation of a layer. The vast number of possible tilings and data flows is pruned by eliminating infeasible or inefficient tilings. Employing list scheduling and linear memory allocation techniques, the framework then generates a schedule and its execution time for each DFG. The shortest schedule is converted into a binary execution plan that can be executed directly on the accelerator.

2 EXECUTION PLAN GENERATION

2.1 Pruning

In a first step, the framework generates a number of tiled DFGs. The vast number of possible tilings is pruned by eliminating inefficient tilings for which the size or shape of the tiles leads to an inefficient execution on the given compute array, tilings whose memory requirements exceed the available memory size, or tilings that cause an imbalance in computation versus data movements. For example, with a compute array of 32x32 processing elements (PEs) where the input and output channels are mapped to the rows and columns, respectively, tilings that are not close to a multiple of 32 lead to low utilization of the compute array and are thus not considered. Another aspect are supported memory layouts. For example, for a memory layout `[N,C,H,W]` (data is stored consecutively in the width dimension), tilings that cause non-consecutive memory accesses are not considered. These heuristics eliminate over 99% of all possible DFGs.

CASES'21 Companion, October 8–15, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8378-3/21/10…$15.00
https://doi.org/10.1145/3451939.3477593

2.2 Scheduling

The framework's scheduler borrows from traditional list instruction scheduling techniques. Internally, the scheduler is composed of an operation manager and a memory manager with a number of queues and lists as shown in Figure 1.

2.2.1 Operation Scheduling. The operation scheduler maintains different work and ready queues for independent hardware units such as the compute array and the buses. A DFG only contains the computational operations and is first augmented by inserting load/store operations for each accessed memory tile and converted into the data dependency graph (DDG). Operations (compute and memory) are inserted into the ready queues as their data dependencies are satisfied. Ready operations are scheduled onto their respective hardware units and assigned a start/end time. Targeting our in-house neural network accelerator, the latency of a operation can be computed accurately in dependence of the type of the operation and the size of the involved data. After scheduling an operation, successors that have all data dependencies satisfied are moved from the work into the ready queue.

2.2.2 Memory Management. The memory manager schedules memory operations and manages the on-chip memory space. Priority is given to memory operations that are needed earlier by computational operations. A lack of on-chip memory may delay memory operations and thereby also push back dependent computational operations.

- ***Garbage management & Memory deallocation*** On-chip memory is managed with garbage management. To maximize data reuse, the memory manager marks memory as a garbage instead of deallocating it immediately in response to a store operation. Before scheduling a load operation, the memory manager checks whether the requested chunk of memory is already on-chip and discards the load operation if possible.
- ***Memory allocation & Synchronization*** The memory manager allocates memory for load operations and computational operations with an output. If sufficient memory is available for the operand, the on-chip memory address is recorded in the corresponding binary operation. If no memory is available, memory blocks marked as garbage are iteratively released until sufficient memory is available. If this fails as well, the memory operation's start time is delayed until some compute operations have completed and memory can be allocated.

3 EVALUATION

We evaluate the presented framework with a number of representative CNNs and compare it to existing work that aims at optimizing execution plans for embedded accelerators [3, 4, 6]. Table 1 compares the strategies for planning of the existing related works. Due to the limited space, we show results for only two distinct hardware configurations of our embedded accelerator. The first configuration comprises a 64x64 PE compute array and 512 KiB of on-chip dual ported memory. The second configuration features 32x32 PEs and 256 KiB of on-chip dual ported memory.

Figure 2 shows the results of the presented framework. Performance is shown as the speedup normalized to the end-to-end latency of the entire network on our accelerator relative to Li et al. [3]. Our framework clearly outperforms the other three approaches by a significant margin. An analysis of the generated execution schedules reveals that this is due to the ability of the presented scheduling technique to identify and select the best tiling strategy and data dimension for each individual layer, whereas related work use more restrictive tiling strategies that incur severe performance penalties for certain layers.

Table 1: Execution plan generation strategies of related work.

	Planning Strategy	Details
Smart Shuttle [3]	empirical rules	fully-tiled W dimension
Ma et al. [4]	minimize output reload	fully-tiled W dimension best tilings for remaining dimensions
Zhang et al. [6]	output stationary	fully-tiled W dimension best tilings for remaining dimensions

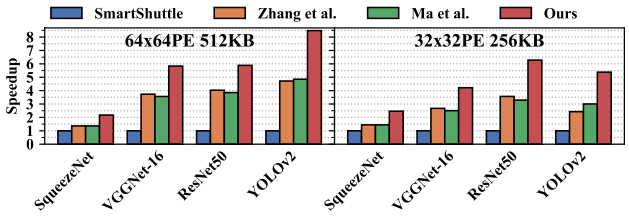

Figure 2: Performance of related work normalized to Smartshuttle which has the biggest latency for different HW configurations.

4 CONCLUSION AND OUTLOOK

The presented framework is able to generate efficient execution plans for various convolutional networks and accelerator configurations. The framework evaluates different dataflows and tilings using instruction scheduling techniques to identify the best execution mode for each layer of a network. In ongoing work, we are evaluating the framework with more networks and are adding support for multi-core accelerators with shared on-chip memory.

REFERENCES

[1] Barend Harris, Inpyo Bae, and Bernhard Egger. 2018. Architectures and algorithms for on-device user customization of CNNs. *Integration* (2018). https://doi.org/10.1016/j.vlsi.2018.11.001

[2] D. Kang, J. Kang, H. Kwon, H. Park, and S. Ha. 2019. A Novel Convolutional Neural Network Accelerator That Enables Fully-Pipelined Execution of Layers. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. 698–701. https://doi.org/10.1109/ICCD46524.2019.00102

[3] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li. 2018. SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 343–348. https://doi.org/10.23919/DATE.2018.8342033

[4] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '17)*. ACM, New York, NY, USA, 45–54. https://doi.org/10.1145/3020078.3021736

[5] NVIDIA. 2018. *NVDLA Hardware Manual.* http://nvdla.org/hw/v1/hwarch.html

[6] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '15)*. ACM, New York, NY, USA, 161–170. https://doi.org/10.1145/2684746.2689060