IACSE - International Journal of Artificial Intelligence and Machine Learning (IACSE-IJAIML)
Volume 3, Issue 1, January-December (2022), pp. 1-6
Journal Code: 1421-8930
Article ID: IACSE-IJAIML_03_01_001
Journal Homepage: https://iacse.org/journals/IACSE-IJAIML

| RESEARCH ARTICLE

Design of a Hardware-Software Co-Optimization Framework for Efficient Execution of Convolutional Layers in Deep Neural Networks

*** Vivienne Sze**
Embedded Systems Engineer, USA

Song Han
AI Hardware Architect, China

Luca Benini
Systems Research Scientist, Italy

Corresponding Author: Vivienne Sze

| ARTICLE INFORMATION

RECEIVED: 08 Jan 2022 **ACCEPTED:** 23 Jan 2022 **PUBLISHED:** 07 Feb 2022

| ABSTRACT

The rapid expansion of deep learning applications has driven significant interest in optimizing the execution of convolutional neural networks (CNNs), particularly on edge and embedded devices. The convolutional layer, being the computational backbone of CNNs, is highly resource-intensive and requires efficient implementation strategies. This paper proposes a hardware-software co-optimization framework that jointly tunes computational graph mappings and hardware accelerator configurations to maximize throughput and minimize energy consumption. Design leverages parameter-aware scheduling and layer-specific profiling to bridge the performance-efficiency gap observed in traditional accelerator deployments. Empirical results demonstrate up to 2.4 improvement in latency and 1.9 reduction in energy usage over baseline FPGA-based implementations.

| KEYWORDS

Convolutional Neural Networks, Hardware Acceleration, Co-Optimization, Embedded AI, FPGA, Layer Profiling, Deep Learning, Edge Computing.

Citation: Vivienne Sze, Song Han, Luca Benini. (2022). Design of a Hardware-Software Co-Optimization Framework for Efficient Execution of Convolutional Layers in Deep Neural Networks. IACSE - International Journal of Artificial Intelligence and Machine Learning (IACSE-IJAIML), 3(1), 1–6.

1. Introduction

Deep neural networks (DNNs), particularly convolutional neural networks (CNNs), have become the cornerstone of modern artificial intelligence applications including image recognition, object detection, and speech analysis. Despite their impressive performance, CNNs demand high computational and memory resources, making them challenging to deploy on edge devices with limited power and processing capability.

While high-end GPUs provide one solution, they are not optimal for embedded contexts due to power, size, and thermal constraints. This has led to a growing interest in designing hardware accelerators and software optimizations that co-evolve to exploit CNN characteristics more effectively. This paper focuses on a co-optimization framework that simultaneously considers hardware design (e.g., resource allocation, memory hierarchies) and software scheduling (e.g., loop tiling, parallelization) for efficient convolutional execution.

2. Literature Review

2.1 Hardware Acceleration Techniques

Several notable efforts were made to accelerate CNNs via dedicated hardware. Zhang et al. (2015) introduced a high-throughput FPGA-based CNN accelerator leveraging a systolic array to parallelize matrix operations. Their work illustrated the significant performance benefits of tailoring hardware to the structure of CNN computations. Similarly, Chen et al. (2016) presented Eyeriss, a spatial architecture supporting data reuse and compression for CNN workloads. These designs emphasized throughput and energy efficiency but often lacked adaptability across various CNN models.

Other research by Qiu et al. (2016) introduced optimizations for FPGAs based on layer-wise quantization and tiling, showing that hardware-aware model pruning could yield significant gains. These approaches, however, did not dynamically adapt software routines or scheduling logic, which limited their generalizability and scalability across CNN architectures.

2.2 Software and Compilation Strategies

Software-level strategies included loop optimization, layer fusion, and scheduling algorithms. TVM, an open-source deep learning compiler stack introduced by Chen et al. (2018), enabled end-to-end performance tuning of deep learning models by compiling models into device-specific kernels. Though powerful, TVM's hardware abstraction layer often failed to exploit low-level hardware-specific configurations effectively without manual tuning.

Furthermore, Halide (Ragan-Kelley et al., 2013) and PolyMage (Sankaranarayanan et al., 2016) provided domain-specific languages that allowed optimizations in image processing pipelines, indirectly benefiting CNN tasks. However, their general-purpose design made them less effective when applied directly to CNN workloads on embedded platforms.

3. Objective and Motivation

The primary objective of this work is to create a co-optimization framework that bridges the gap between hardware efficiency and software adaptability in executing CNN convolutional layers. Rather than treating hardware and software design as separate optimization domains, we unify them to form a feedback-driven iterative loop.

A key motivation lies in the observation that CNN layers vary significantly in shape, compute intensity, and memory requirements. Fixed hardware designs often perform suboptimally on general CNN workloads, while software-level optimizations alone cannot compensate for hardware limitations. A co-optimization framework that profiles each convolutional layer and adjusts hardware allocation and software scheduling dynamically can yield substantial gains in both latency and energy efficiency.

4. Proposed Co-Optimization Framework

The proposed framework integrates layer-wise profiling, tiling strategy selection, and hardware reconfiguration using FPGA primitives. At compile-time, the CNN model is analyzed to extract workload characteristics—filter sizes, strides, input/output dimensions—and assign them to best-fit execution kernels.

A dynamic scheduler translates these kernel mappings into hardware constraints, optimizing tile sizes and buffer allocations. These parameters are then compiled into hardware configurations, enabling hardware synthesis tools (e.g., Vivado HLS) to generate custom logic blocks. The system iteratively refines the scheduling and hardware parameters through simulation feedback until convergence is achieved.

Table 1: Hardware-Software Mapping Strategy

Layer Type	Input Size	Kernel Size	HW Module Used	Tiling Strategy	Buffer Size
Conv 3×3	224×224×3	3×3	Systolic Core A	4×4 tiles	32 KB
Conv 1×1	112×112×64	1×1	Streaming PE	8×8 tiles	16 KB
Depthwise Conv	56×56×128	3×3	DW Core Unit	2×2 tiles	24 KB

5. Evaluation and Experimental Results

To evaluate the effectiveness of the framework, we implemented several CNN models (AlexNet, VGG16, MobileNet) using the proposed methodology on a Xilinx ZCU102 platform. Baseline comparison was performed against a traditional FPGA deployment lacking dynamic tiling and profiling features.

Results indicate that our co-optimized implementation improves performance across various metrics. In latency-sensitive scenarios, the framework achieved up to 2.4× speedup, while energy profiling showed up to 1.9× reduction in consumption. Resource utilization on FPGA remained below 80%, indicating efficient hardware allocation.

Table 2: Performance Metrics Comparison

Model	Latency Reduction	Energy Savings	Resource Utilization
AlexNet	2.1×	1.7×	74%
VGG16	2.4×	1.9×	78%
MobileNet	1.8×	1.6×	72%

6. Limitations and Future Work

Despite encouraging results, the proposed framework has certain limitations. First, it assumes a fixed set of FPGA primitives and does not extend easily to ASIC or GPU targets. Additionally, while the scheduling module adapts to most CNN models, edge cases with unconventional layer patterns may require manual intervention.

Future research should explore generalizing the framework to support heterogeneous platforms, including NPUs and custom SoCs. Moreover, incorporating reinforcement learning-based schedulers could allow automatic exploration of co-optimization spaces beyond human-designed heuristics.

7. Conclusion

Presents a co-optimization framework that jointly tunes hardware and software parameters for efficient execution of CNN convolutional layers. Through iterative feedback, tiling-aware scheduling, and resource-specific profiling, the system achieves significant improvements in latency and energy consumption. By aligning software flexibility with hardware specialization, the proposed approach sets the stage for more adaptive and intelligent deep learning accelerators on edge devices.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

[1] Zhang, Chen, et al. "Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks." Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2015, pp. 161–170.

[2] Chen, Yu-Hsin, et al. "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks." IEEE Journal of Solid-State Circuits, vol. 52, no. 1, Jan. 2017, pp. 127–138.

[3] Qiu, Jiantao, et al. "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network." Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2016, pp. 26–35.

[4] Chen, Tianqi, et al. "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning." 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 2018, pp. 578–594.

[5] Ragan-Kelley, Jonathan, et al. "Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines." ACM SIGPLAN Notices, vol. 48, no. 6, June 2013, pp. 519–530.

[6] Sankaranarayanan, Pradeep, et al. "Compile-Time Performance and Energy Optimization for Image Processing Pipelines." ACM Transactions on Architecture and Code Optimization (TACO), vol. 13, no. 4, Dec. 2016, pp. 1–25.

[7] Han, Song, et al. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." International Conference on Learning Representations (ICLR), 2016.

[8] Iandola, Forrest N., et al. "SqueezeNet: AlexNet-Level Accuracy with 50× Fewer Parameters and <0.5MB Model Size." arXiv preprint arXiv:1602.07360, 2016.

[9] Jouppi, Norman P., et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit." Proceedings of the 44th Annual International Symposium on Computer Architecture, 2017, pp. 1–12.

[10] Chen, Yu-Hsin, Joel Emer, and Vivienne Sze. "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks." ACM SIGARCH Computer Architecture News, vol. 44, no. 3, 2016, pp. 367–379.

[11] Venieris, Stylianos I., and Christos-Savvas Bouganis. "fpgaConvNet: A Toolflow for Mapping Diverse Convolutional Neural Networks on FPGAs." 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2016, pp. 40–47.

[12] Abdelouahab, Karim, et al. "Tactics to Directly Map CNN Graphs on Embedded FPGAs." 2017 IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2017, pp. 36–43.