CNN and RVV Co-design for Efficient Model Serving

CNN and RVV Co-design for Efficient Model Serving

Sonia Rani Gupta
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
soniar@chalmers.se

Nikela Papadopoulou
University of Glasgow
Glasgow, United Kingdom
nikela.papadopoulou@glasgow.ac.uk

Jing Chen
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
chjing@chalmers.se

Miquel Pericàs
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
miquelp@chalmers.se

Abstract

Convolutional algorithm performance depends on layer dimensions, with SIMD demands and cache sharing influencing runtime selection. To identify the best settings, we perform a co-design exploration of convolutional layer parameters and three algorithms: Direct, im2col+GEMM, and Winograd, jointly with hardware parameters for RISC-V vector architectures. Our results show that incorporating hardware parameters with layer dimensions boosts execution time and efficiency, emphasizing the need for co-design.

ACM Reference Format:
Sonia Rani Gupta, Nikela Papadopoulou, Jing Chen, and Miquel Pericàs. 2025. CNN and RVV Co-design for Efficient Model Serving. In *The 19th ACM International Conference on Distributed and Event-based Systems (DEBS '25), June 10–13, 2025, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3701717.3733226

1 Introduction

Convolutional Neural Networks (CNNs) are widely used for vision tasks and increasingly deployed on CPUs [5], leveraging their availability, high core counts, and SIMD acceleration. CNN layers differ in dimensions (input/output size, kernel, channels, stride), and different convolution algorithms vary in compute and memory needs; im2col+GEMM, for example, enhances performance but increases cache usage. Moreover, these algorithms need optimizations to utilize SIMD units efficiently.

Model serving frameworks replicate models and distribute incoming requests for load balancing, but concurrent execution competes for cache resources, making convolution algorithms sensitive to co-running tasks. Despite extensive research on CNNs, the interaction between convolution algorithms and hardware parameters remains underexplored, limiting utilization and hindering efficient CPU design for CNN serving.

This work compares Direct, Winograd, and two im2col+GEMM variants across all convolutional layers in YOLOv3 and VGG16 using the RISC-V Vector Extension (RVV) v1.0 [2]. Results show no single best algorithm across layers. Co-design analysis reveals algorithm performance depends on layer dimensions and hardware. To improve efficiency, we train a Random Forest classifier using 5-fold validation, achieving a prediction accuracy of 92.8%. Finally, performance-to-area trade-offs show that per-layer algorithm selection improves efficiency, even with multiple model instances.

DEBS '25, Gothenburg, Sweden

ACM ISBN 979-8-4007-1332-3/25/06
https://doi.org/10.1145/3701717.3733226

2 Methodology

We target the RISC-V Vector Extension [2] (RVV) within the RISC-V Architecture using a fork of the gem5 [1] with *RiscvMinorCPU* at 2GHz with an integrated vector unit implementing RVV v1.0. Vector lengths range from 512 to 4096 bits, and L2 cache from 1MB to 64MB. We evaluate YOLOv3 [6] and VGG-16 [8], where the convolutional layers consume 96% and 64% of total time, respectively, when profiled on A64FX using Linux `perf`.

We evaluate three algorithms for convolutional layers: Winograd, im2col+GEMM, and Direct. For im2col+GEMM, we use the RVV-optimized 3-loop and 6-loop variants from [4], denoted as *im2col+GEMM - 3 loops* and *im2col+GEMM - 6 loops*. Winograd is vectorized and optimized as described in [3]. For Direct, we adopt the long-vector algorithms by Santana et al. [7].

3 Evaluation

This section covers per-layer algorithm selection for VGG-16 and YOLOv3 on RVV core, co-design results, predictor for algorithm selection and a performance area tradeoffs on a 7nm RVV chip.

Performance Comparison of Convolution Implementations. We evaluate each convolutional layer of both models with a 512-bit vector length and 1MB L2 cache. Figure 1 shows per-layer performance for the VGG-16 model. Direct performs best when input/output dimensions are high but input/output channels are low, while im2col+GEMM with 6 loops excels for skinny matrices (low input/output dimensions, high input/output channels). Winograd performs comparably or better for most layers, except those with high input channels. YOLOv3 shows similar trends.

Increasing vector lengths from 512-bit to 4096-bit shows that Direct shows the maximum scalability of 2.4×–5.8× with longer vector lengths and outperforms the other algorithms. Im2col+GEMM offers better performance for vector lengths higher than 1024-bit for the skinny matrices. Increasing the L2 cache from 1MB to 64MB

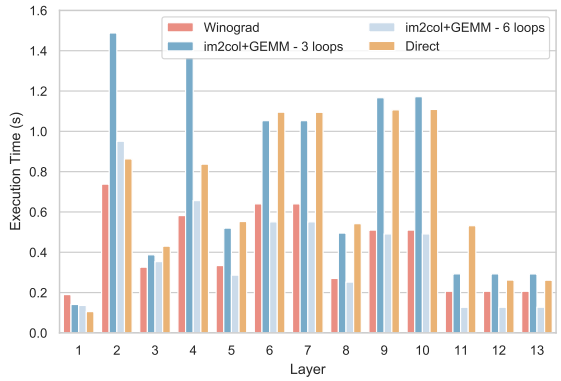

Figure 1: Comparative analysis for different algorithms for VGG-16 convolutional layers on RVV with gem5, with a vector length of 512 bits and 1MB of L2 cache.

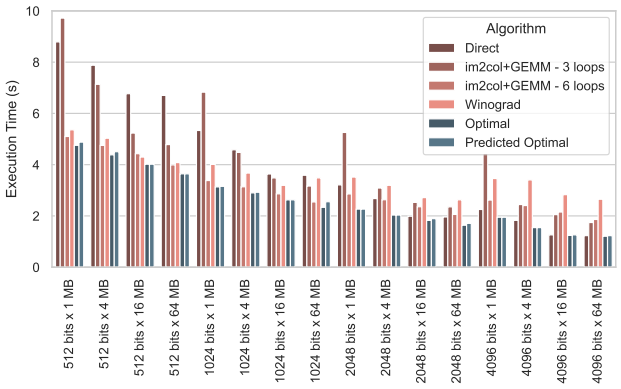

Figure 2: Execution time of VGG-16, for different vector lengths and L2 cache sizes, when a single algorithm is used for all layers (*Direct*, *im2col+GEMM-varients*, *Winograd*), compared against using the *Optimal* algorithm per layer, and using our algorithm selection model to predict the optimal algorithm per layer (*Predicted Optimal*).

reveals the limited scalability of Winograd, as its fixed tile size doesn't take advantage of larger caches. Both im2col+GEMM variants show limited scalability beyond 16MB L2 cache for extremely skinny matrices, while Direct benefits most from larger caches.

Algorithm Selection. Our results show no single algorithm consistently minimizes execution time across all layers. Hence, to predict the optimal algorithm per layer, we use random forests from the Python `Scikit-learn` 1.3.2 package. We use as input parameters the vector length, the L2 cache, input/output channels, height, width, stride, and padding, height, and width, and the kernel height and width, totaling 12 parameters. The model outputs the algorithm predicted to perform the best. We partition the data into 80% for training and 20% of testing, and use 5-fold cross-validation and shuffling. Our evaluation shows that the algorithm selection model achieves an average prediction accuracy of 92.8% across the 5 cross-validation sets. To assess its impact, we apply the model to select algorithms for VGG-16 and YOLOv3. Selecting the optimal algorithm improves the execution time by up to 1.85× compared to always using Direct and up to 1.73× over using the 6-loop implementation of im2col+GEMM in the case of VGG16 as shown in Figure 2. Similarly, for YOLOv3, selecting the optimal algorithm improves the execution time by 1.33× and 2.11× over always using Direct and 6-loop implementation of im2col+GEMM, respectively.

Performance-Area Tradeoffs. To assess the performance-area tradeoff within a fixed area, we first analyze a single model running on an RVV core with an integrated VPU in 7nm FinFET technology (Section 2). Selecting the optimal algorithm per layer improves performance by 1.18×–1.6× for YOLOv3 and 1.26×–2.32× for VGG-16 compared to using a single algorithm across layers.

Next, we evaluate a multi-core RVV chip with configurations ranging from 1 to 64 cores, vector lengths of 512 to 4096 bits, and shared L2 caches of 1MB to 256MB, simulating a realistic server in a model-serving context with 200 hardware configurations. Our results show that, for a 64-core configuration with 256MB of cache and 4096-bit vector units, co-locating model instances with the optimal algorithm per layer improves throughput by 1.16× over using the best-performing algorithm (Direct) for all layers.

4 Conclusion

Our study shows that the optimal algorithm per layer depends on both layer parameters and hardware factors like vector length and L2 cache size. We build a Random Forest classifier, achieving 92.8% accuracy, to select the best algorithm per layer, improving execution time and enabling more efficient performance–area tradeoffs. When coupled with model co-location, it boosts throughput per area, emphasizing the importance of co-design for model serving.

Acknowledgments

This work was funded by the European High-Performance Computing Joint Undertaking (JU) under grant agreements No. 956702 (eProcessor), No. 101036168 (EPI SGA2, via FPA No. 800928), and No. 101034126 (European PILOT), with support from the EU's Horizon 2020 programme and member states including Spain, Sweden, Greece, Italy, France, and Germany. Additional funding was provided by the Swedish Foundation for Strategic Research (PRIDE, ref. CHI19-0048) and the Swedish Research Council (P4PIM, project ID 2020-04892). Computations were enabled by NAISS resources, partially funded by the Swedish Research Council (grant No. 2022-06725).

References

[1] [n.d.]. plct-gem5. https://github.com/plctlab/plct-gem5
[2] [n.d.]. RISC-V Vector. https://github.com/riscv/riscv-v-spec/releases
[3] Sonia Rani Gupta, Nikela Papadopoulou, and Miquel Pericàs. 2023. Challenges and Opportunities in the Co-Design of Convolutions and RISC-V Vector Processors. In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W '23)*. Association for Computing Machinery, New York, NY, USA, 1550–1556.
[4] Sonia Rani Gupta, Nikela Papadopoulou, and Miquel Pericàs. 2023. Accelerating CNN inference on long vector architectures via co-design. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 145–155.
[5] Yizhi Liu, Yao Wang, Ruofei Yu, Mu Li, Vin Sharma, and Yida Wang. 2019. Optimizing {CNN} Model Inference on {CPUs}. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 1025–1040.
[6] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *arXiv* (2018).
[7] Alexandre de Limas Santana, Adrià Armejach, and Marc Casas. 2023. Efficient Direct Convolution Using Long SIMD Instructions. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP '23)*. Association for Computing Machinery, New York, NY, USA, 342–353.
[8] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:cs.CV/1409.1556