Improving Direct Convolution through Tensor Slicing, Vectorized Packing and ISA Extensions

Victor Ferrari (author)[1], **Guido Araujo (advisor)**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
13083-852 – Campinas – SP – Brasil

v187890@dac.unicamp.br, guido@unicamp.br

Abstract. *Convolution is one of the most computationally intensive machine learning model operations, usually solved by the traditional Im2Col + BLAS method. This work describes SConv: a novel direct-convolution algorithm to improve upon Im2Col + BLAS by introducing compile-time and execution time components to tile, vectorize and optimize the computation. SConv's speed-up over an Im2Col + BLAS method based on current BLAS implementations for end-to-end machine-learning model inference is in the range of 11% – 27% for Intel x86 and 11% – 34% for IBM POWER10 architectures. The total convolution speedup for model inference is 13% – 28% on Intel x86 and 23% – 39% on IBM POWER10. SConv also outperforms oneDNN in 6 out of 7 models.*

1. Introduction

The world is becoming increasingly connected, and the volume of data to be processed in many applications is constantly expanding. In this context, machine learning algorithms, specifically deep learning and neural networks, are rapidly becoming important and powerful tools for building computational models that can represent non-analytically complex problems involving many variables in a multidimensional hyperspace.

Convolutional Neural Networks (CNN) are useful tools for addressing problems with huge data sets captured from real-world sensors. The steady increase in the adoption of CNNs is driven mostly by applications in the Computer Vision domain, where it addresses problems like Object Recognition [He et al. 2016, Zhou et al. 2014], Object Detection [Girshick et al. 2014], and Video Classification [Karpathy et al. 2014].

Since their origins, the size and complexity of state-of-the-art CNNs have grown significantly. It is well-known that convolution is the most expensive operation of a CNN [Chellapilla et al. 2006]. Furthermore, this operation is also important for other applications, such as image processing and other types of signal processing. Bigger CNN models contain larger convolutions that not only demand more computational power but also result in a significant increase in data movement between different levels of the memory hierarchy. Consequently, improving convolution performance requires attention to hardware throughput and memory access.

Convolution is usually solved with a method called Im2Col + BLAS [Chellapilla et al. 2006], consisting of a transformation to a GEMM (Generic Matrix Multiplication) problem followed by an efficient solution from a BLAS library. This method has two packing steps: the Im2Col transformation and the BLAS GEMM packing routine. This approach, combined with the GEMM-specific cache blocking solution, is not

ideal for convolution [Zhang et al. 2018]. In certain conditions, direct convolution can outperform the traditional Im2Col reduction.

Even though GPUs and other accelerators have been growing in popularity in the past few years, ML inference in CPUs is still important for many applications ranging from smaller models to executing in general consumer hardware. Newer ISA extensions are improving performance of vector and matrix operations without the need for dedicated accelerators, with less power consumption requirements and lower price.

This research project had the goal of exploring direct convolution, its properties, and generalizations to support ISA extensions such as the IBM POWER10 MMA engine [Moreira et al. 2021], to improve convolution (and consequently, machine learning inference) performance in different architectures with more complex SIMD and matrix units. Better performance can be achieved by improving on some of the previously described limitations of Im2Col + BLAS.

Goto and van de Geijn [Goto and Geijn 2008] introduced a macro-kernel/micro-kernel structure for GEMM, used by BLAS libraries, and this can be adapted to construct a novel convolution algorithm with specific cache blocking, reduced data movement, and good hardware usage.

This project's main contribution is the introduction of SConv: a direct-convolution algorithm that uses architectural information to improve convolution's cache utilization and ISA extensions to accelerate data packing and computation, suitable for SIMD architectures. The algorithm can leverage Instruction Set Architecture (ISA) acceleration extensions (*e.g.* IBM POWER10 MMA) to compete with optimization libraries such as BLAS [Xianyi et al. 2011]. SConv itself provides four major contributions:

- A compiler-based solution for convolution code generation that does not depend on linking with math optimization libraries such as BLAS;
- Convolution Slicing Analysis (CSA), a generic compiler analysis pass that can be used to determine a tiling strategy for convolution in different types of CPU architectures, given convolution and hardware information;
- Convolution Slicing Optimization (CSO), a code-generation pass that results in a direct convolution macro-kernel that improves performance when compared to Im2Col+BLAS for two CPU architectures (Intel x86 and IBM POWER10) using specialized micro-kernels;
- Vector-Based Packing, an input-tensor packing solution that leverages shift instructions in vector registers for better performance in unitary stride convolutions.

The original thesis covers all these contributions in detail, as well as a deep exploration of the micro-kernel and a large experimental evaluation of SConv in both a realistic machine-learning setup and individual convolution unit tests.

This project led to a presentation at CASCON x EVOKE 2021, a poster at PACT '22 [Ferrari et al. 2023a], and a full-length article in ACM TACO [Ferrari et al. 2023b], presented at HiPEAC 2024. Additionally, it resulted in a presentation at the Open MLIR Meeting (March 9, 2023) and a RFC [Ferrari 2023].

2. Related Work

Chellapilla *et al.* proposed the computation of convolution using the Im2Col transformation to reduce the problem to GEMM and solve it using a BLAS li-

brary [Chellapilla et al. 2006]. The BLAS's GEMM routine already contains packing steps. Thus two separate steps require data manipulation: expansion (Im2Col) and packing. This approach has a large memory footprint because of the large matrix resulting from Im2Col. Other authors address the memory footprint issue while still using BLAS GEMM as a foundation to solve the convolution problem [Anderson et al. 2020, Cho and Brand 2017].

Zhang *et al.* [Zhang et al. 2018] discuss the limitations and inefficiencies of the Im2Col + BLAS convolution algorithm and present an argument for using direct convolution instead. They use a model architecture with fused multiply-add instructions to explore cache blocking and data reuse, given a friendly memory layout.

Goto and van de Geijn [Goto and Geijn 2008] introduce the idea of exploiting an optimized micro-kernel (called "inner-kernel") by tiling the problem in an external macro-kernel. These tiles are then packed to a friendly layout for the micro-kernel. The CSA algorithm resembles the algorithm proposed by Goto and de Gejin in its functionalities, but it applies the strategy to convolution instead of applying it to GEMM.

Li *et al.* [Li et al. 2021] proposes a tiling algorithm that initially assesses how different permutations (loop orders) of the nested tiling loops may affect data movement. This analysis results in eight different permutations that are evaluated through a cost model that uses a non-linear optimization problem (min-max) to produce tile sizes and tries to reduce the data movement between each cache level. Tollenaere *et al.* [Tollenaere et al. 2023] design a solution that selects up to two micro-kernels which together are divisible by the output height dimension, as a way to avoid partial tiling.

Juan *et al.* [Juan et al. 2020] modify the BLIS library's GEMM routine to apply Im2Col in its packing step. The result is a convolution and GEMM hybrid named ConvGEMM, which takes advantage of Goto and van de Gejin's GEMM tiling and structure with the Im2Col transformation applied on the fly, eliminating the double-packing problem. This process is similar to Packing on Demand, but the tiling is still GEMM-based, and therefore has different cache usage, tile reuse, and packing patterns.

Korostelev *et al.* [Korostelev et al. 2023] combine the ideas of modifying the GEMM routine and decomposing convolution into multiple GEMM operations [Anderson et al. 2020] to create a new method that avoids packing redundancy while keeping the overall GEMM structure (tiling, packing, micro-kernel).

Following the work from Zhang *et al.* [Zhang et al. 2018], Barrachina *et al.* [Barrachina et al. 2023] propose two new direct-convolution algorithms for the NHWC layout (batch N, height H, width W, and channels C) on ARM processors. Like SConv, they tile in the channel dimension and use a BLAS micro-kernel.

This project follows the work done by Sousa *et al.* [Sousa et al. 2021] in tiling 3D convolutions for NPUs, by generalizing its tiling analysis for CPUs in the CSA pass. In that work, scheduling strategies are used in a different context, and the modeling focuses on taking advantage of memory bursts while loading data to the scratchpad memories.

An adaptation of the GEMM routine to a convolution algorithm by lazy Im2Col packing was made for GPU hardware by Chetlur *et al.* [Chetlur et al. 2014]. Adapting SConv to GPUs would require a different tiling analysis focusing on the SIMT (Single

Instruction Multiple Threads) properties of the hardware and memory coalescing, instead of CPU cache hierarchies. This variation would require fundamental changes to the algorithm described in this work, but it may preserve the compilation flow described in the following section.

3. SConv Overview

SConv is a solution to convolution that mixes compile-time data transformation for filter packing and efficient code generation to better utilize memory and architecture resources. This compilation flow was created to fit in most machine learning frameworks based on compiling models for inference, such as ONNX-MLIR [Le et al. 2020]. The compilation flow is described in Figure 1.

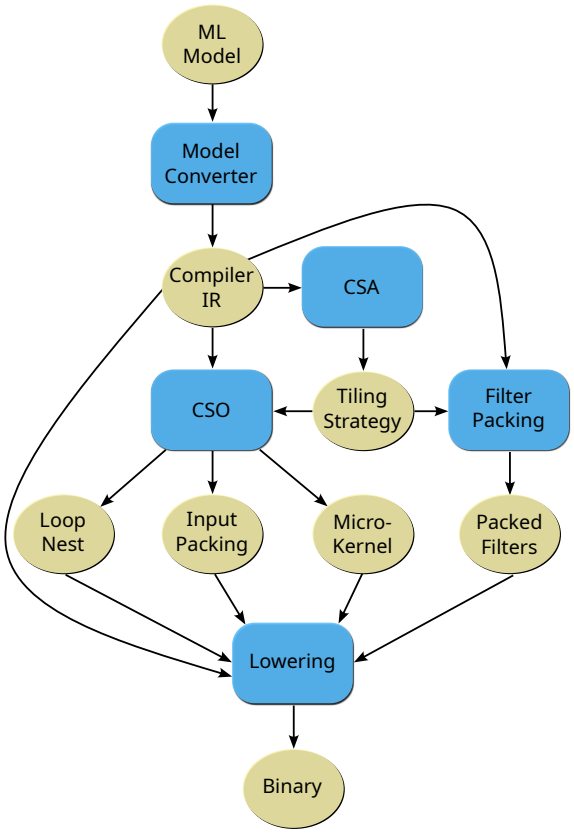

Figure 1. Compilation flow in a machine-learning compiler using SConv to optimize convolutions.

In order to attain peak hardware performance, a hand-optimized outer-product-based micro-kernel is used to compute a single tile of the convolution output. This micro-kernel is architecture-specific, leveraging ISA acceleration extensions such as IBM MMA [Moreira et al. 2021] and Intel AVX-512. The outer product is generally useful due to its versatility for computing higher-rank operations and high throughput, computing n^2 output elements from $2n$ input elements. For those reasons, it is widely used in high-performance linear algebra libraries, and it is being incorporated in CPU ISA extensions such as IBM POWER10 MMA. This process is extensively detailed in Chapter 5 of the original thesis.

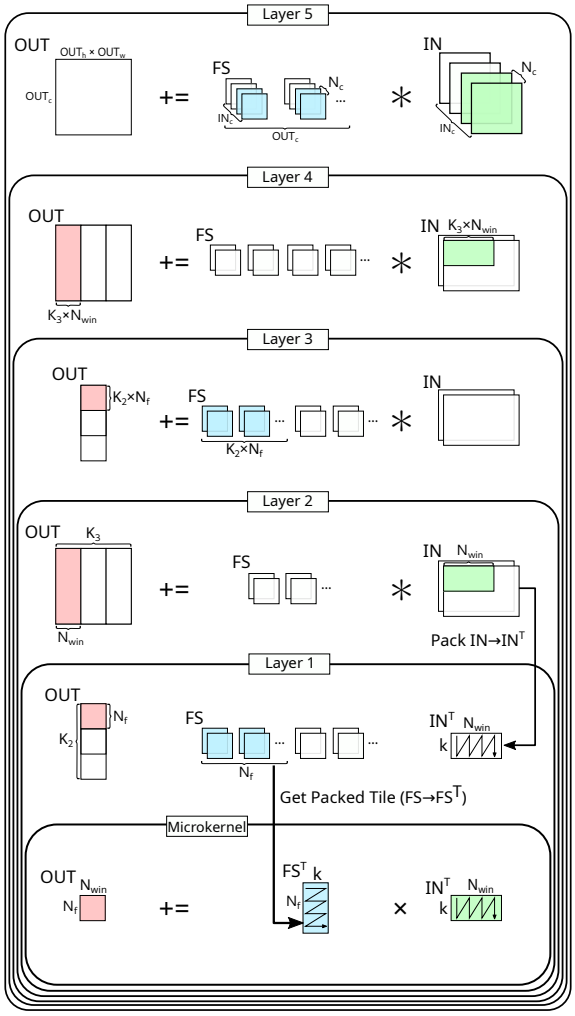

Figure 2. Convolution loop nest macro-kernel generated by CSO using CSA's tiling parameters. Illustrates the process for Input Stationary scheduling.

Convolution Slicing Optimization (CSO), as shown in Figure 2, is a code-generation pass that generates the macro-kernel responsible for ensuring the best usage of the micro-kernel by tiling, packing and scheduling the convolution data according to the tiling strategy.

The tiling strategy, comprised of convolution tile size definition, tile scheduling and distribution over the cache hierarchy [Sousa et al. 2021] are defined by Convolution Slicing Analysis (CSA) so to leverage faster access to cache memory and tile reuse. The tile sizes are selected based on micro-kernel information and cache size, so that one tile of the input tensor, filter and output fit in the L1 cache at the same time. These tiles are then split into sets that indicate a tile distribution (Figure 3) over the higher cache levels based on these cache sizes, and scheduled for computation in an order that maximizes tile reuse. Scheduling can be Input Stationary or Weight Stationary, and the decision is made based on a cost model that analyzes data movement cost for each option.

By creating a convolution-specific tiling strategy, SConv can pack the input tensor data right before the tile is used in the micro-kernel, in a single step. We call this approach Packing On Demand. The two main problems of Im2Col + BLAS: multiple packing steps

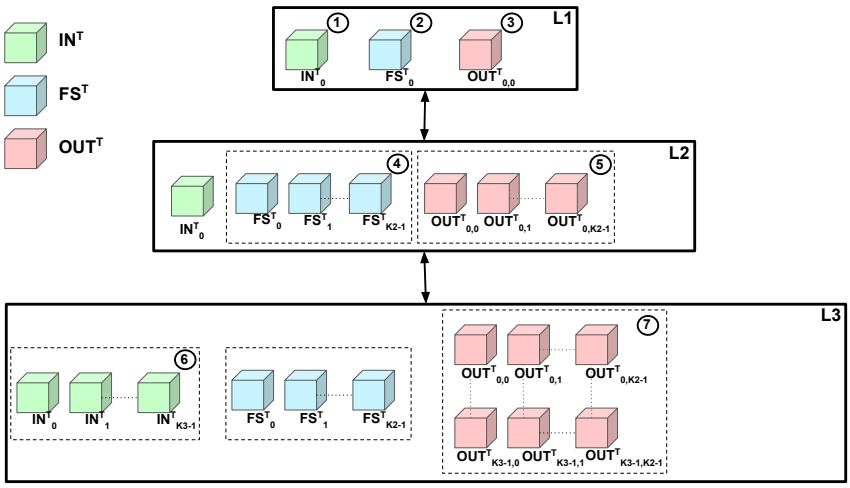

Figure 3. CSA tile distribution over a three-level cache hierarchy with Input Stationary scheduling.

and suboptimal tiling, are therefore improved upon with SConv.

Packing is a way to connect the tiling process to the micro-kernel. The low level routine reads data from memory sequentially, and this access pattern does not match the starting convolution data layout. Therefore, in order to correctly use the micro-kernel each tile needs to be reordered in memory, and in the process loaded to cache for faster access. This step is called packing, and its layout is defined by the micro-kernel, which is architecture-specific.

As packing can be a bottleneck, a unique and faster routine is defined to rearrange the data of the input tensor to the format required for its computation. This process can also use hardware-specific shift instructions for better performance when the convolution has unitary stride, as described in Section 7.2.2 of the original thesis. When packing the filter set, a different optimization can be leveraged when used within a machine-learning compiler setting, targeting inference. In such cases, the filter set is fixed, constant and part of the model information, so this packing step can be performed as a data transformation optimization pass at compile time, as shown in Figure 1.

The tiling process (CSA + CSO) uses architecture information as parameters, but is otherwise generic. The micro-kernel and packing steps are highly hardware-specific, and should be developed separately for each architecture target.

4. Experimental Results

A SConv prototype was integrated as a library in the ONNX-MLIR [Le et al. 2020] framework for testing. The results were compared with the same framework using the standard Im2Col + BLAS method, composed by Caffe's [Jia et al. 2014] implementation of Im2Col and OpenBLAS's [Xianyi et al. 2011] GEMM routine, henceforth called Base. Seven well-known machine learning models with different sizes and convolution shapes were used for evaluation on two CPU architectures: Intel Cascade Lake x86 and IBM POWER10, both with the same cache hierarchy and cache sizes (Thesis Section 8.1).

As shown in Table 1, SConv out-performs the baseline in every model and on

Model	Convolution Speedup		Model Speedup		Convolution Time Share	
	x86	P10	x86	P10	x86	P10
GoogleNet	1.20	1.26	1.11	1.11	0.48	0.45
InceptionV2	1.23	1.28	1.18	1.19	0.77	0.64
ResNet-18	1.28	1.39	1.27	1.34	0.94	0.85
ResNet-50	1.16	1.28	1.16	1.22	0.90	0.75
ResNet-152	1.18	1.24	1.17	1.19	0.93	0.78
SqueezeNet	1.13	1.23	1.11	1.16	0.75	0.60
VGG-16	1.28	1.28	1.17	1.13	0.70	0.51
Geo Mean	1.21	1.28	1.17	1.19	0.76	0.64

Table 1. SConv performance speedup over Base in machine-learning models.

both architectures. The convolution speedup range from 23% to 39% on POWER10 and from 13% to 28% on x86. These results include every convolution from each model. Since machine learning models contain many different layer types, the SConv end-to-end model relative performance against Base is lower than the convolution performance. Speedup results range from 11% to 34% on POWER10 and from 11% to 27% on x86.

As expected, the most improved step of the convolution algorithm was packing, with an average performance improvement of 2.9x on x86 and 4.7x on POWER10. Faster hardware-supported micro-kernel makes the packing time more significant in the overall execution, so the more powerful IBM POWER10 architecture with MMA support allows for better speedup results (Thesis Section 8.3.5).

SConv achieves speedup against Base in most of the 393 total convolutions from the seven models, more specifically 89% of the convolutions on POWER10 and 90% on x86. Our approach speeds-up individual convolutions up to 2.17x on POWER10, and 2.14x on x86 (Thesis Section 8.3.3).

POWER10 has a theoretical peak throughput of 249.6 GFLOPS/s, and the x86 architecture can reach up to 67.2 GFLOPS/s. SConv reaches 78% of this theoretical peak (194.8) in POWER10 and 90% (60.2) in x86. The actual values vary greatly depending on the convolution sizes. Since POWER10 has a faster micro-kernel, packing time is more significant and responsible for a larger share of the execution time.

SConv also reduces cache misses in all levels. In SConv, around 90% of all loads resolve in the L1 cache, compared with 83% in Base — a 1.9$\times$ improvement. The difference between the methods increases at lower cache levels: on average, SConv has 5.8x fewer L2 misses and 9.9x fewer L3 misses, though both represent a small percentage of the total number of loads. SConv also performs fewer total loads in 63% of the tests.

The performance of other state-of-the-art convolution routines from the Intel OneAPI library [Intel 2022] against Base was tested in the x86 architecture, for comparison with SConv: (a) the Caffe Im2Col transformation followed by oneMKL GEMM, and (b) the forward convolution routine from oneDNN. As shown in Figure 4, across all models, SConv surpasses oneMKL, achieving a speedup difference ranging from 14% to 29%. Similarly, when comparing with oneDNN, SConv exhibits a speed advantage between 8% and 23% in all models except for VGG16, for which it performs 5% slower when comparing to Base (Thesis Section 8.3.8).

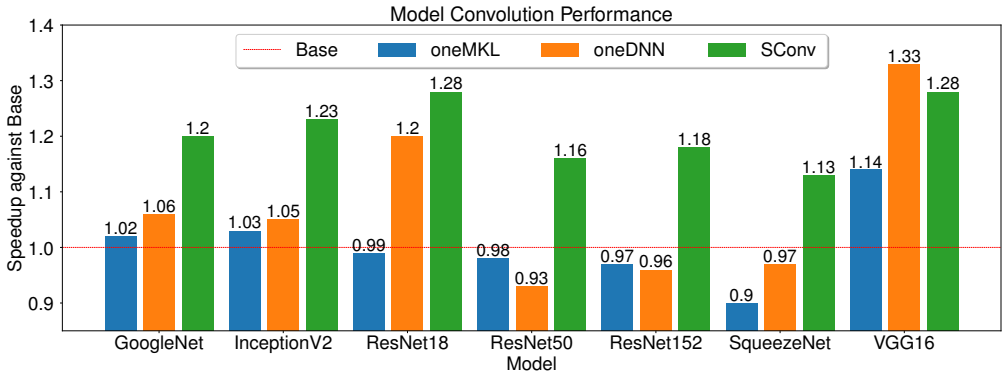

Figure 4. Performance speedup of SConv and oneAPI routines over Base in machine learning models.

Therefore, the research and development of SConv allowed for an improvement on the most important problems of the Im2Col + BLAS method while maintaining portability and exploiting novel ISA extensions and SIMD units. This resulted in a significant performance increase, outperforming both the original baseline and other state-of-the-art implementations of convolution solutions. It can be perceived as the generalization of the optimized direct convolution algorithm from [Zhang et al. 2018] for better micro-kernels and different CPU architectures, without the requirement for a different input memory layout and the modification of other operators. SConv can be integrated into any machine-learning compiler or framework seamlessly, with only two target-specific routines that need to be implemented for each architecture to achieve the best performance.

5. Future Work

SConv was developed as a single-threaded algorithm for CPUs. However, to be competitive in practical situations, the algorithm should be altered to leverage parallelism, which would change how CSA handles the cache structure. In this case, shared caches between cores would need to be modeled.

Currently, the CSA tiling analysis pass creates tiles based solely on micro-kernel information (number of windows and filters) and the number of input channels. More robust space exploration can be done by relaxing the spatial constraints of the tiles to better utilize the L1 cache in convolutions with few channels. At the current state, the spatial dimensions of the tiles are constrained by the micro-kernel shape. If this can be surpassed, a larger tiling space could be available for exploration, and better heuristics could be developed to maximize cache usage.

Generating code at the IR level can be both a significant advantage and a hindrance. The result of the compilation process has a fully sequential and unrolled code format, which can underutilize the L1 instruction cache and require many page reads. Solving this problem can improve performance in the fully code-generated version of SConv. Furthermore, micro-kernel portability is a research path that should be explored for an improved version of SConv. If a common abstraction for architectures with different features is provided [Vasilache et al. 2022], a fully generic version of the algorithm can be implemented without the need for direct support for every architecture.

References

Anderson, A., Vasudevan, A., Keane, C., and Gregg, D. (2020). High-performance low-memory lowering: Gemm-based algorithms for dnn convolution. In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 99–106.

Barrachina, S., Castelló, A., Dolz, M. F., Low, T. M., Martínez, H., Quintana-Ortí, E. S., Sridhar, U., and Tomás, A. E. (2023). Reformulating the direct convolution for high-performance deep learning inference on arm processors. *Journal of Systems Architecture*, 135:102806.

Chellapilla, K., Puri, S., and Simard, P. (2006). High Performance Convolutional Neural Networks for Document Processing. In Lorette, G., editor, *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France). Université de Rennes 1, Suvisoft.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J. M., Tran, J., Catanzaro, B., and Shelhamer, E. (2014). cudnn: Efficient primitives for deep learning. *ArXiv*, abs/1410.0759.

Cho, M. and Brand, D. (2017). Mec: Memory-efficient convolution for deep neural network. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 815–824. JMLR.org.

Ferrari, V. (2023). Optimized Convolution for MLIR. `https://discourse.llvm.org/t/rfc-optimized-convolution-for-mlir/69454/`.

Ferrari, V., Sousa, R., Pereira, M., de Carvalho, J. a. P. L., Amaral, J. N., and Araujo, G. (2023a). Improving convolution via cache hierarchy tiling and reduced packing. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, PACT '22, page 538–539, New York, NY, USA. Association for Computing Machinery.

Ferrari, V., Sousa, R., Pereira, M., L. De Carvalho, J. a. P., Amaral, J. N., Moreira, J., and Araujo, G. (2023b). Advancing direct convolution using convolution slicing optimization and isa extensions. *ACM Trans. Archit. Code Optim.*, 20(4).

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587.

Goto, K. and Geijn, R. A. v. d. (2008). Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3).

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Intel (2022). *oneAPI Specification*. Intel Corporation.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

Juan, P. S., Castello, A., Dolz, M. F., Alonso-Jorda, P., and Quintana-Orti, E. S. (2020). High performance and portable convolution operators for multicore processors. In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 91–98, Los Alamitos, CA, USA. IEEE Computer Society.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732.

Korostelev, I., L. De Carvalho, J. a. P., Moreira, J., and Amaral, J. N. (2023). Yaconv: Convolution with low cache footprint. *ACM Trans. Archit. Code Optim.*, 20(1).

Le, T. D., Bercea, G.-T., Chen, T., Eichenberger, A. E., Imai, H., Jin, T., Kawachiya, K., Negishi, Y., and O'Brien, K. (2020). Compiling onnx neural network models using mlir. *ArXiv*, abs/2008.08272.

Li, R., Xu, Y., Sukumaran-Rajam, A., Rountev, A., and Sadayappan, P. (2021). Analytical characterization and design space exploration for optimization of cnns. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 928–942.

Moreira, J. E., Barton, K., Battle, S., Bergner, P., Bertran, R., Bhat, P., Caldeira, P., Edelsohn, D., Fossum, G., Frey, B., Ivanovic, N., Kerchner, C., Lim, V., Kapoor, S., Filho, T. M., Mueller, S. M., Olsson, B., Sadasivam, S., Saleil, B., Schmidt, B., Srinivasaraghavan, R., Srivatsan, S., Thompto, B. W., Wagner, A., and Wu, N. (2021). A matrix math facility for power ISA(TM) processors. *CoRR*, abs/2104.03142.

Sousa, R., Byungmin, J., Kwak, J., Frank, M., and Araujo, G. (2021). Efficient tensor slicing for multicore npus using memory burst modeling. In *2021 33th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE.

Tollenaere, N., Iooss, G., Pouget, S., Brunie, H., Guillon, C., Cohen, A., Sadayappan, P., and Rastello, F. (2023). Autotuning convolutions is easier than you think. *ACM Transactions on Architecture and Code Optimization*, 20(2):1–24.

Vasilache, N., Zinenko, O., Bik, A. J. C., Ravishankar, M., Raoux, T., Belyaev, A., Springer, M., Gysi, T., Caballero, D., Herhut, S., Laurenzo, S., and Cohen, A. (2022). Composable and modular code generation in mlir: A structured and retargetable approach to tensor compiler construction.

Xianyi, Z., Kroeker, M., Saar, W., Qian, W., Chothia, Z., Shaohu, C., and Wen, L. (2011). Openblas: An optimized blas library.

Zhang, J., Franchetti, F., and Low, T. M. (2018). High performance zero-memory overhead direct convolutions. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5776–5785. PMLR.

Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., and Oliva, A. (2014). Learning deep features for scene recognition using places database. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.