Towards Optimal Winograd Convolution on Manycores

Zhen Jia[*†] **Aleksandar Zlateski**[*‡] **Fredo Durand**[‡] **Kai Li**[†]
[†]**Princeton University** [‡]**Massachusetts Institute of Technology**
[*]**Equally contributed**

Motivation and Previous Work

As convolutional layers are computationally intensive and dominate the total execution time of modern, deep Convolutional neural networks (ConvNets) [15, 22, 24, 25], many efforts have been made to improve the performance of the convolutional primitives for CPUs [1, 6, 26, 31, 33], GPUs [4, 7, 19, 27] or both [32]. An important class of improvements is to reduce the computations required for a convolution. Several efforts employed FFT-based convolutions to reduce the required computations for GPUs [19, 27] and CPUs [31, 32].

Recently, Lavin et al. [16] proposed an algorithm based on the Winograd algorithm for minimal filtering, originally developed for fast computation of finite impulse response (FIR) filters [30]. The key idea of the Winograd–based convolution is similar to the one based on FFTs. The inputs and the kernels are first transformed, and then an element–wise multiplication, which is an equivalent problem to a matrix multiplication, is performed. An inverse transformation of the result yields the result of the convolution. Unlike the FFT–based convolution, where the point-wise multiplications are done in the complex domain, the Winograd–based convolution operates on real numbers, thus requiring fewer operations.

After Lavin et al. [16] demonstrated that Winograd–based convolution can be more efficient than FFT in reducing the number of multiplications, especially for small 2D kernels (e.g. 3×3), Nervana [4] and Nvidia's cuDNN [7] had implemented Winograd–based convolution for GPUs. CPU implementations were also provided by FALCON [1], LIBXSMM[5], Intel MKL-DNN [2] and Budden et al. [6]. In addition, currently available implementations support only 2D convolutions and single kernel size (3×3), which restricts the range of the application of Winograd–based convolution. 3D ConvNets are becoming important, as they have been successfully applied to many fields [9, 14, 20, 21].

Our work on the design, implementation and evaluation of a fast Winograd-based implementation is motivated by the fact that current Winograd–based implementations for multicore or manycore CPUs perform well below the hardware capability. In many cases, they under–performed more computationally expensive, but more optimized implementations, such as direct convolutions.

Our Contributions and Novelty

Several challenges make it difficult to fully utilize the hardware resources on modern CPUs. **(1)** Winograd–based convolution might access the memory subsystem inefficiently, which may cost more than the time saved by performing fewer computations. **(2)** Winograd-based convolution requires efficient matrix multiplications on tall and skinny matrices. Optimized libraries for matrix multiplication, such as Intel MKL and LIBXSMM [10, 11] do not achieve satisfactory performance on such matrices [17, 29]. **(3)** The increasing of thread and data level parallelisms requires new scheduling algorithms that minimize load balancing and synchronization overhead for manycore CPUs.

This paper offers two main contributions. First, we present a new implementation for Winograd–based convolution for manycore CPUs. It is the first publicly available implementation that supports N–dimensional ConvNets with arbitrary kernel and tile (transformation) sizes. Second, to address the aforementioned challenges, we propose several novel optimizations for manycore CPUs, including:

- a custom data layout that allows efficient memory access by using only vector loads and stores, and using streaming stores when the data will not be required in the near future,
- a method to organize memory access patterns to minimize TLB misses, as well as amortize the memory access overhead by interleaving computation with memory operations,
- a novel JIT code generator to generate optimal matrix multiplication routines for matrices of relevant sizes to maximize cache locality, and
- a static scheduling method for even parallel execution on the available cores to minimize synchronization overheads.

We have implemented these ideas and evaluated our implementation with several representative ConvNets on an Intel manycore CPU (Knights Landing) [12]. Our implementation is publicly available [3].

Evaluation

We benchmarked our implementation and other CPU implementations on an Intel Xeon Phi 7210 (Knights Landing, KNL) node, which has 64 cores, capable of approximately 4.5 TFLOPS of single precision floating points, and 400 GBytes/s bandwidth memory. For GPU methods, we evaluated on an Nvidia Titan X Pascal, capable of approximately 11 TFLOPS for FP32.

To evaluate 2D ConvNets, we chose the VGG-A version of OxfordNet [24] and FusionNet [23]. For 3D ConvNets, we chose C3D [21] and 3D U–Net [8]. We benchmarked the most computationally expensive convolutional layers of each network. Fig. 1 shows the execution times. For our implementation, we show the speeds of various tile sizes, i.e., $F(m, r)$, whose definition can be found in [16]. The columns annotated with "FX" assume no kernel transformation, which are suitable for inference–only computation.

2D networks FALCON's Winograd implementation only supports $F(2\text{x}2, 3\text{x}3)$. Our implementation achieved speedups between 1.08x and 5.39x. With larger $F(m, r)$, our implementation achieved up to 8.33x speedup. MKL-DNN and LIBXSMM only support $F(4\text{x}4, 3\text{x}3)$. Our implementation achieved speedups ranging from 1.59x to 2.84x over MKL-DNN and from 1.32x

The full version of this paper [13] has been accepted at PPoPP 2018.

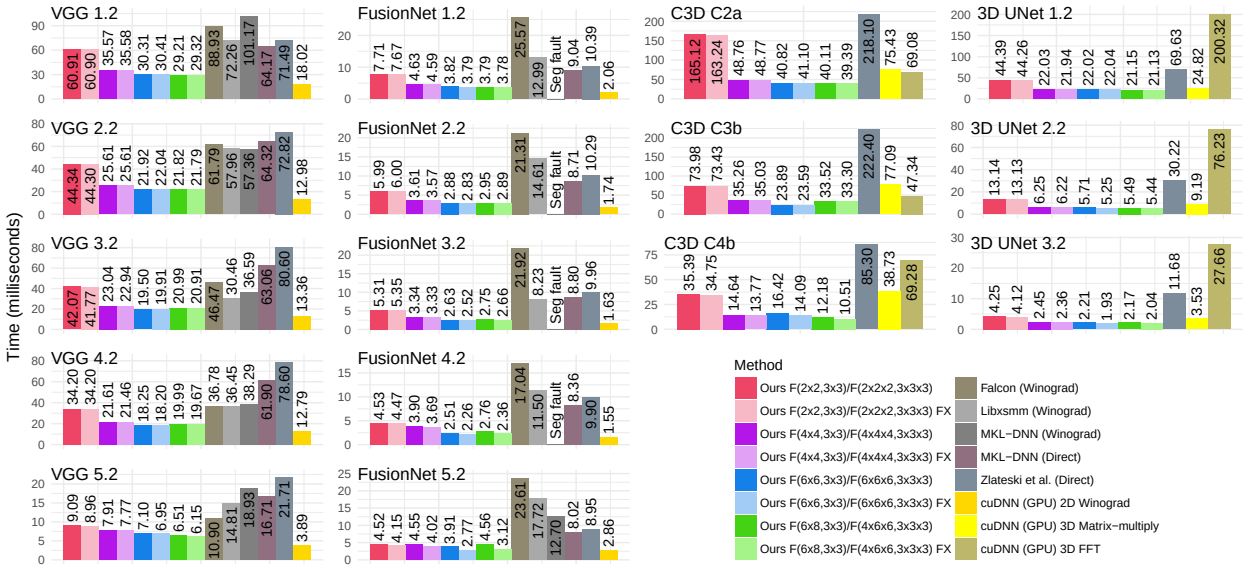

Figure 1: Convolution layers' runtime with different implementations. MKL-DNN's Winograd–based convolution produces segmentation faults for 4 of 5 FusionNet layers. The columns annotated with "FX" do not perform kernel transformations.

to 4.05x over LIBXSMM. With larger $F(m, r)$, our implementation achieved up to 3.34x speedup over MKL-DNN and 5.07x over LIBXSMM. CuDNN's 2D Winograd implementation for the GPU outperformed ours by an average of 1.5x, while running on a GPU that is capable of roughly 2.5x FLOPS than the KNL processor, indicating that we better utilize the hardware.

For the Winograd algorithm proposed by Budden et al. [6] for CPUs, there is no publicly available code. Their measured throughput of the sample network (3 layers with 32 channels each, and unusual kernel size of 4x4) on an 18-core Intel E7-8890 CPU (Haswell) was 10.9 MVox/s. For the same network, our approach achieved roughly 100 MVox/s (9x speedup) on the KNL. Since the peak FLOPS of the E7-8890 CPU is roughly 1/3 of the KNL processor, we can estimate that our algorithm achieves 3x better utilization of the hardware.

3D networks Since all Winograd (CPU and GPU) implementations support only 2D, we could only perform limited benchmarks for 3D ConvNets. We achieved better performance than the approach proposed by Zlateski et al. [33] by 6x, cuDNN's matrix–multiply based convolution by 2x, and cuDDN's FFT based convolution by 8x.

Effects of tile size Theoretically, for Winograd, i.e. $F(m, r)$, the larger the m, the more operations can be saved at the matrix multiplication stage. However, in reality, large values of m can lead to a computation overhead due to the following reasons: (1) The dimension length of output images has to be divisible by m, otherwise, the image is zero padded, increasing the number of operations at both transformation and matrix multiplication stages. This is the main reason why, for some layers, larger ms did not achieve better performance. (2) The number of operations for the image and kernel transformations increases quadratically with m [18]. Since Winograd is numerically unstable [6, 16, 28], only small tile sizes can be used. Our measurements [13] show that $F(6^2, 3^2)$ for 2D and $F(4\text{x}6^2, 3^3)$ for 3D have errors small enough. Tile sizes larger than those yield numerical errors two or more orders of magnitude larger, when compared to direct convolution [13, 16].

Inference vs training All speedups reported above are for training – using the implementation that transforms kernels as well. The "FX" variation, that assumes memoized values of the kernel transforms, can further improve the performances in certain cases. For most of the layers, the kernel transformations only require a small percentage of the total execution time. However, for layers with a large number of input/output channels, the kernel transformations can take a large percentage of time, especially when the batch size is one. This is notable for FusionNet (layers 4.2 and 5.2).

Summary Overall, we observed an average of 66% utilization of the theoretical peak FLOPS and roughly 80% of the memory bandwidth. When compared to other existing Winograd–based implementations, on the same hardware, an average speedup of 3x, and in some cases more than 8x, was observed.

Conclusion

While the Winograd–based convolution algorithm can greatly reduce the computational complexity of convolutional layers, it is challenging to implement it to fully exploit the hardware performance of manycore CPUs.

Our implementation can achieve substantially better performance than the existing Winograd implementations for the CPU, competitive with GPU implementations for 2D and faster for 3D. This was achieved by the proposed interdependent optimizations, each motivated by a specific hardware feature or limitation, but designed to work together to improve overall performances.

References

[1] 2016. FALCON Library: Fast Image Convolution in Neural Networks on Intel Architecture. "https://colfaxresearch.com/falcon-library/". (2016).

[2] 2016. Intel(R) Math Kernel Library for Deep Neural Networks. "https://github.com/01org/mkl-dnn". (2016).

[3] 2018. N-Dimensional Winograd–based convolution framework. https://bitbucket.org/poozh/ond-winograd. (2018).

[4] Accessed: 2018-01-01. Intel® Nervana reference deep learning framework. https://github.com/NervanaSystems/neon. (Accessed: 2018-01-01).

[5] Accessed: 2018-01-01. LIBXSMM. https://github.com/hfp/libxsmm. (Accessed: 2018-01-01).

[6] David Budden, Alexander Matveev, Shibani Santurkar, Shraman Ray Chaudhuri, and Nir Shavit. 2016. Deep Tensor Convolution on Multicores. *arXiv preprint arXiv:1611.06565* (2016).

[7] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[8] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 2016. 3D U-Net: learning dense volumetric segmentation from sparse annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 424–432.

[9] Liuhao Ge, Hui Liang, Junsong Yuan, and Daniel Thalmann. 2017. 3d convolutional neural networks for efficient and robust hand pose estimation from single depth images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[10] Alexander Heinecke, Greg Henry, Maxwell Hutchinson, and Hans Pabst. 2016. LIBXSMM: accelerating small matrix multiplications by runtime code generation. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 981–991.

[11] Alexander Heinecke, Hans Pabst, and Greg Henry. 2015. Libxsmm: A high performance library for small matrix multiplications. *Poster and Extended Abstract Presented at SC* (2015).

[12] James Jeffers, James Reinders, and Avinash Sodani. 2016. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann.

[13] Zhen Jia, Aleksandar Zlateski, Kai Li, and Fredo Durand. 2018. Optimizing N-Dimensional, Winograd-Based Convolution for Manycore CPUs. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*.

[14] J Jiménez, S Doerr, G Martínez-Rosell, AS Rose, and G De Fabritiis. 2017. DeepSite: Protein binding site predictor using 3D-convolutional neural networks. *Bioinformatics* (2017).

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[16] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4013–4021.

[17] Jiajia Li, Casey Battaglino, Ioakeim Perros, Jimeng Sun, and Richard Vuduc. 2015. An input-adaptive and in-place approach to dense tensor-times-matrix multiply. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 76.

[18] Vijay K. Madisetti. 2009. *The Digital Signal Processing Handbook, Second Edition*. CRC Press.

[19] Michael Mathieu, Mikael Henaff, and Yann LeCun. 2013. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851* (2013).

[20] Daniel Maturana and Sebastian Scherer. 2015. 3d convolutional neural networks for landing zone detection from lidar. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 3471–3478.

[21] Daniel Maturana and Sebastian Scherer. 2015. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 922–928.

[22] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*. 2924–2932.

[23] Tran Minh Quan, David GC Hilderbrand, and Won-Ki Jeong. 2016. FusionNet: A deep fully residual convolutional neural network for image segmentation in connectomics. *arXiv preprint arXiv:1612.05360* (2016).

[24] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[25] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[26] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. 2011. Improving the speed of neural networks on CPUs. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, Vol. 1. 4.

[27] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. 2014. Fast convolutional nets with fbfft: A GPU performance evaluation. *arXiv preprint arXiv:1412.7580* (2014).

[28] Kevin Vincent, Kevin Stephano, Michael Frumkin, Boris Ginsburg, and Julien Demouth. 2017. On Improving the Numerical Stability of Winograd Convolutions. (2017).

[29] Yida Wang, Michael J Anderson, Jonathan D Cohen, Alexander Heinecke, Kai Li, Nadathur Satish, Narayanan Sundaram, Nicholas B Turk-Browne, and Theodore L Willke. 2015. Full correlation matrix analysis of fMRI data on Intel® Xeon Phi coprocessors. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 23.

[30] Shmuel Winograd. 1980. *Arithmetic complexity of computations*. Vol. 33. Siam.

[31] Aleksandar Zlateski, Kisuk Lee, and H Sebastian Seung. 2016. ZNN–A Fast and Scalable Algorithm for Training 3D Convolutional Networks on Multi-core and Many-Core Shared Memory Machines. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 801–811.

[32] Aleksandar Zlateski, Kisuk Lee, and H Sebastian Seung. 2016. ZNNi: maximizing the inference throughput of 3D convolutional networks on CPUs and GPUs. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 854–865.

[33] Aleksandar Zlateski and H Sebastian Seung. 2017. Compile-time optimized and statically scheduled ND convnet primitives for multi-core and many-core (Xeon Phi) CPUs. In *Proceedings of the International Conference on Supercomputing*. ACM, 8.

Acknowledgments

We thank Sebastian Seung for helpful discussions. We are grateful to Intel Corporation for supporting the Intel Parallel Computing Center at Princeton University, and to Toyota Research Institute for supporting the Toyota - CSAIL Joint Research Center at MIT. Zhen Jia was partially supported by IARPA (D16PC00005).