A LOW-LATENCY SPARSE-WINOGRAD ACCELERATOR FOR CONVOLUTIONAL NEURAL NETWORKS

Haonan Wang, Wenjian Liu, Tianyi Xu, Jun Lin and Zhongfeng Wang
Department of Electronic Science and Engineering, Nanjing University, P.R. China

Motivation

Leverage Sparsity and Winograd Algorithm [1]

- Make Winograd orthogonal to pruning
- Fully exploit sparsity in the dataflow

Drawbacks of current accelerators

- Only support Winograd or pruning
- Fail to use the sparsity in activations

Main Contributions

- A hardware architecture is designed to efficiently support sparse models and leverage Winograd Algorithm
- Sparsity in both activations and weights are fully utilized via fast mask indexing scheme
- The FMI module is proposed to eliminate all redundant multiplication operations and the corresponding cycles

Comparisons

Work	[2]	[3]	Ours
Precision	16bits fixed	16bits fixed	16bits fixed
Board	ZC706	ZC706	ZC706
Freq. (MHz)	166	166	**250**
BRAM (Kb)	540x18	732x18	**528x18**
DSP	532	768	**380**
LUT	**90k**	155k	93k
Flip-Flop	**92k**	153k	96k

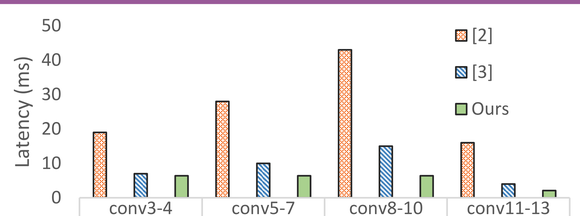

Table 1: Implementation Results and Comparisons

References

1. Liu, Xingyu, et al. "Efficient sparse-winograd convolutional neural networks." arXiv preprint arXiv:1802.06367 (2018).
2. Lu, Liqiang, et al. "Evaluating fast algorithms for convolutional neural networks on FPGAs." 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2017.
3. Lu, Liqiang, and Yun Liang. "SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs." 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). IEEE, 2018.

Architecture

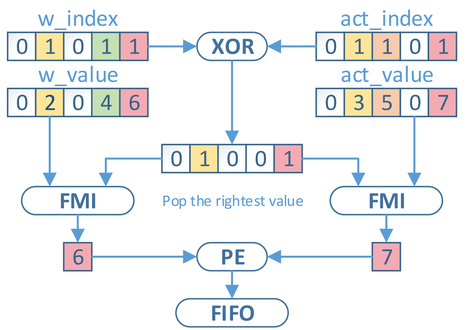

Figure 1: Scheme of Fast Mask Indexing

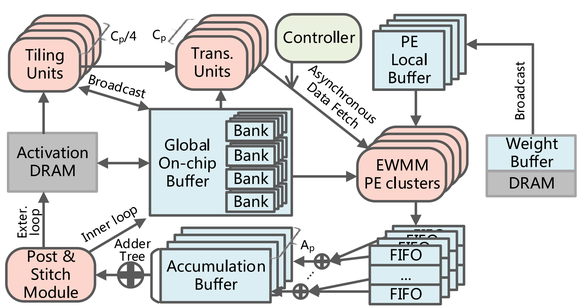

Figure 2: Overview of architecture

Computation flow

- **Step 1:** Activation tensors are split into tiles, then Winograd transformation and ReLU operation are performed. Both sparse activations and pre-trained Winograd weights are labeled with mask index.
- **Step 2:** Each cycle, one PE processes multiplications with non-zero values poped from the FMI module according to the XOR-ed index.
- **Step 3:** The dataflow exploits weight-stationary scheme, so results of multiplications are accumulated to the partial sums via FIFOs. When an output tile is ready, post-processing is instantly performed.

Architecture

- Global Buffer unit is a SRAM which stores intermediate data, forming an inter-module pipeline.
- PE Local Buffer saves weights for reusing.
- EWMM modules continuously process non-zero multiplications and dump the results to FIFOs.
- Partial sums are read from the Accumulation Buffer to be added with outputs of FIFOs and then are restored. The adder tree is used to sum all channels.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grants No. 61774082 and 61604068; the Fundamental Research Funds for the Central Universities under Grant No. 021014380065