Point Cloud Generation with Stereo and Monocular Depth Estimation on FPGA

Alejandro Perez-Vicente[1,2], David Hernandez[1], Tianyang Fang[1], Jafar Saniie[1]

[1]*Embedded Computing and Signal Processing Research Laboratory (http://ecasp.ece.iit.edu/)*
Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616
[2]*ETSI Industriales, Universidad Politécnica de Madrid, Madrid, Spain*

***Abstract*— This research proposes a system that leverages stereo vision and monocular depth estimation to form a depth map from which a 3D point cloud scene is extracted. The emergence of competitive neural networks for depth map estimation presents an opportunity for deployment on embedded systems. Both stereo and monocular depth algorithms were tested on multiple FPGA hardware platforms. To perform acceleration on configurable hardware, a Deep Learning Processing Unit (DPU) IP was used for processing AI workloads, together with a Pseudo-LiDAR reprojection for 3D reconstruction on the Arm® processor. The DPU performance configuration will depend on the logic resources the targeted board has. This paper highlights the performance, resource utilization, and bottlenecks that came across different tested combinations of both hardware and neural network architectures.**

Keywords—FPGA, DPU, 3D Reconstruction, Pseudo-LiDAR, Monocular Depth, Stereo Vision

I. Introduction

3D scene reconstruction is highly relevant for cyber-physical systems, whether it be exploring unknown terrain, autonomous driving, or robotics. The most common data for computer vision is obtained from single CMOS cameras as a 2D scene representation, having different resolutions and color channels. This type of representation lacks depth information which is very valuable for applications where the Euclidean space geometry is not expendable.

Three-dimensional representations help to determine surface locations and their interaction with the environment. This type of representation can be made by using ToF 3D sensing technologies [1] such as LiDAR or by a combination of sensors involving CMOS cameras and IR modules. Relevant sensor fusion techniques include structured light pattern generation and stereoscopic vision. These two last methods are widely used for embedded systems applications where physical design, memory resources, and computing power become limited. Advancements in hardware accelerators and image processing algorithms have made it possible for CMOS cameras to be suitable sensors for 3D vision on low-power devices. The most common way to run complex algorithms is by utilizing dedicated hardware such as GPUs which excel in performance for parallel computations. This type of hardware is still the dominant player in neural network training. However, the inference space is becoming populated with custom hardware accelerators specialized in performing graph computations [2]. Recent trends in FPGA hardware design have enabled faster inference acceleration compared to peer devices.

Therefore, we try to come up with an FPGA point cloud generator by running stereo and monocular depth neural networks on DPU IP cores for the Zynq® UltraScale+™ architecture. The number of logic resources in the FPGA will be crucial for determining the number of DPU cores that can be routed onto the programable logic. The objective will be to maximize the utilization of the programable resources by the DPU cores to increase the performance of the designated system.

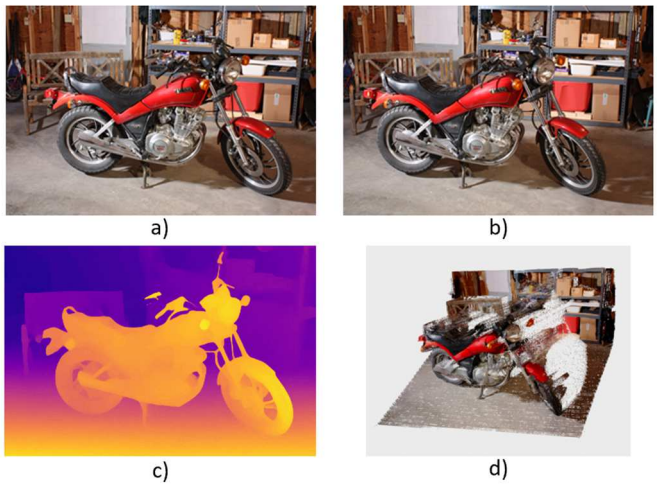

Fig. 1. The proposed system receives a left image(a) and a right image (b) to perform a disparity map (c) and output a point cloud (d) from a 3D depth map projection. For monocular vision, the depth map will be done by only using the left image channel from the stereo pair.

II. System Design

This chapter will have an overview of the targeted hardware specifications and the selected algorithms for both depth map and point cloud generation. The proposed system will be evaluated on different platforms of the Zynq® UltraScale+™ MPSoC family, differing mainly on the amount of logic units inside the chip. The MPSoC is based on a hybrid CPU-FGA architecture where the configurable logic and a multi-core processor are routed into a single chip. The programmable logic (PL) consists of a combination of logic cells (LUTs and FFs), BRAM, URAM and DSP slices. In addition to the PL, the MPSoC includes Processing System (PS) with a 1.5 GHz Quad-core Arm® Cortex®-A53 processor, a 600 MHz Dual-core

Cortex-R5 RT processor, Mali™-400 MP2 GPU, a memory controller with DMA channels, and high-performance AXI4 slave ports for communication between the PL and PS.

A. Xilinx Kria KV260

Xilinx Kria KV260 is an entry-level FPGA development board optimized for embedded vision purposes. The chip itself is packed as K26 System on Module (SOM) which can be easily connected to a customized PCB for the targeted application. The Kria SOM has a Zynq® UltraScale+™ MPSoC with 256K logic cells, 144 BRAM, 64 URAM, and 1.2K DSP slices, together with 4GB of DDR4 memory and 512Mb of QSPI for primary boot memory.

B. Xilinx ZCU104

The Xilinx ZCU104 board is categorized as an AWS-qualified device for IoT applications for its rapid prototyping and high compute performance for a low power consumption device. The Zynq® UltraScale+™ MPSoC comes with a total of 504K system logic cells, 312 BRAM, 96 URAM, and 1.7K DSP slices. In addition, 2GB of DDR4 memory for the PS and 512Mb of QSPI flash.

C. Xilinx ZCU102

The Xilinx ZCU102 device can be used for multiple application purposes as it comes with many extension ports, FMC, and SFP interfaces. ZCU102 is highly suitable for AI inference acceleration since it can fit many DPU cores inside the PL logic. The MPSoC is equipped with 600K system logic cells, 912 BRAM and 2.5K DSP slices, 512Mb and 4Gb of DDR4 memory for the PL and PS respectively, plus 64Mb of QSPI flash memory.

D. Deep Learning Unit

The Xilinx DPU is a configurable IP block that has the purpose of running on the PL as a compute engine for the parallel execution of convolutional neural networks. The DPU IP core includes memory-mapped AXI interfaces to support status register configurations and accessing data. These interfaces facilitate its integration in the block design. The architecture of the DPU cores can be modified depending on the balance between performance, use of PL, and power draw specifications.

E. Fast and Accurate Disparity Network (FADNet)

FADNet is a neural network [3] used for disparity estimation and scene flow estimation. The FADNet architecture is based on DispNetC and DispNetS [4] which use an encoder-decoder structure but there are some differences made in FADNet. The Dual-Conv modules shrink the feature map size by half, there is also a correlation layer to reduce the end-points error. Also, dual residual blocks are added to the architecture to increase feature extraction of the network.

The dual residual block allows us to have very deep models without increasing training difficulty along with allowing us to increase feature extraction and down-sampling layers. This type of network leverages the disparity refinement method by applying multi-scale residual learning [5], which consists of making the second network learn the disparity residuals and make an accumulation of the predicted disparity by the first

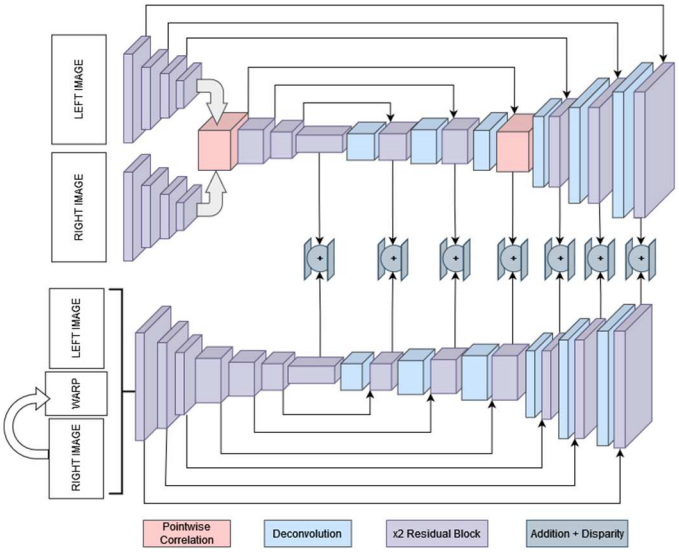

Fig. 2. FADNet architecture

network. This architecture will be compared in terms of inference and result performance with a simplified version of DispNetC and a monocular depth estimation network.

F. Monocular Depth Estimation

The CNN model used for monocular depth estimation was based on [6] which uses a similar encoder-decoder architecture to U-Net [7]. A ResNet-18 [8] was used as an encoder er network, achieving faster inference performance compared to a ResNet-50 or disparity networks such as DispNet. The encoder feature extraction is followed by a decoder module based on successive deconvolution layers. By having a monocular depth approach, the neural network compilation process can be simplified. Also, scheduling the execution kernel onto the DPU is more effective compared to deploying a complex model, even though the depth map generation is relatively diminished.

G. Depth to Point Cloud Transformation

It is important to understand the camera properties from which the depth map was generated to create a 3D point cloud. One of the most important parameters is the focal length of the camera. This parameter is the distance between the pinhole and the image plane of the camera. Stereo disparity algorithms are based on taking a pair of left-right images as input, both captured from a pair of cameras with a horizontal baseline, and output a disparity map, usually with the same (u, v) dimensions if no encoding transformation is applied after. Either the left or right image is treated as a reference, and the horizontal disparity is applied to each pixel of the corresponding stereo-pair image. By leveraging the information provided by the horizontal focal length of the reference camera f_u, the depth map can be obtained via the following equation:

$$Depth(u, v) = \frac{baseline \times f_u}{disparity}$$

The depth map can be included as an additional channel to the RGB channels of the reference image. Other approaches instead try to derive the 3D coordinates (x, y, x) given the pixel

(u, v) location corresponding to the camera center (c_u, c_v). This is referred to as Pseudo-LiDAR [9] point cloud generation.

$$x = \frac{(u - c_u) \times z}{f_u}$$

$$y = \frac{(v - c_v) \times z}{f_v}$$

$$z = Depth(u, v)$$

The resulting transformation is a pixel projection into the 3D coordinates, outputting a point cloud reconstruction of the reference image. The point cloud density will be equivalent to the number of pixels in the disparity map.

III. IMPLEMENTATION ON FPGA

A. DPU Block Design

The Zynq® UltraScale+™ MPSoC family has a supported DPU IP block denoted by DPUCZX8G. Most of the PL inside the MPSoC will be dedicated to the customized DPU IP. Different configurations can be selected based on the desired performance logic usage limitations. To achieve higher performance, the amount of DPU cores can be increased up to 4 cores. To improve performance, these cores can be customized for achieving higher parallelism. The parallelism can be modified across three dimensionalities: pixel parallelism, input parallelism, and output parallelism. For the proposed implementation, the logic resources were prioritized for DPU core parallelism against the amount of DPU cores inside the PL.

The selected DPU core architecture will be B4029 as it provides the highest operations per clock. Operations like Depthwise Convolutions, Elementwise multiply, Max Pooling, ReLU, and Softmax are computed inside the PL to increase the hardware acceleration of the DPU cores. The last configuration to be performed on the DPU is reducing the cascade length of the DSP48 slices. By setting a lower cascade length the DPU timing performance is increased at a cost of requiring extra DSP48 for achieving higher pipelining. The DPU IP clocking is composed of three different clock domains, one for the register, another for the data controller, and lastly a computing clock. For the register configuration module, a 100 MHz clock is set by recommendations of the IP guide, connected through the AXI slave interface. The data controller will schedule the transfer of data with a 325 MHz clock. Lastly, a Dual Data Rate (DDR) clocking technique is applied to enhance the DSP48 performance. To achieve DDR on the DSP slices, the corresponding clock domain will be set to double the clock frequency of the data controller.

The DPU is set to use RAM for buffering weights, biases, and other intermediate features into the on-chip memory. This on-chip memory can be either BRAM, URAM, or a combination of both. For the proposed implementation, the DPU cores will be configured as high RAM usage since most of the FPGA logic design is reserved to improve the DPU performance. The higher RAM usage will make DPU cores more flexible when handling intermediate features. Both KV260 and ZCU104 will use a combination of URAM and BRAM, more precisely, 40 URAM blocks for each DPU core. In the case of ZCU102 which doesn't have URAM block, the on-chip buffer will only be built with BRAM.

Considering the PL limitations of each targeted board, the number of cores was 1, 2, and 3 for KV260, ZCU104, and ZCU102 respectively. All DPU cores share the same configuration except for the ZCU102 cores which don't use URAM blocks.

Table I. Resource utilization for the targeted boards

Resource \ Usage	KV260	ZCU104	ZCU102
LUT	58,450	107,901	157,050
LUTRAM	6,145	11,729	17,331
FF	106,316	204,298	301,537
BRAM	111	218	775
URAM	40	80	N/A
DSP	704	1,394	2,084
Num. DPU cores	1	2	3

B. Embedded Hardware Platform

To run deep learning models on the DPU cores, it is necessary to create a Vitis hardware platform. When targeting a heterogeneous system like the MPSoC, the inference application is split between the embedded host processor (PS) and the hardware accelerator (PL). For the Arm® processor, an executable is written in C++ with the usage of APIs like OpenCL, Vitis-AI, and Xilinx Runtime (XRT). On the other hand, the acceleration kernels will be compiled as an executable device binary (*xclbin*) that will run on the PL. For targeting an FPGA for DPU inference, the hardware design with the DPU IP is compiled as an executable RTL kernel. The host program API calls are responsible for controlling the processing transactions between the PS and the PL, interfacing through the XRT driver and AXI interfaces.

For Zynq® UltraScale+™ MPSoC edge devices, the XRT driver (*ZOCL*) must be included in the device tree of the Linux kernel to manage the resource allocation for the accelerated kernels. The embedded OS used for running the inference applications is a lightweight Ubuntu 20.04 distribution optimized for aarch64 architecture. The Linux kernel included the necessary firmware and libraries for deploying the neural network onto the FPGA device.

C. Neural Network Graph Deployment

Once the hardware platform is running on the embedded OS, the next step is to compile, optimize and transform the neural network into a DFG representation that can be interpreted by the ZOCL driver. The neural networks models for depth map estimation were trained with the KITTI [10] dataset for stereo vision architectures.

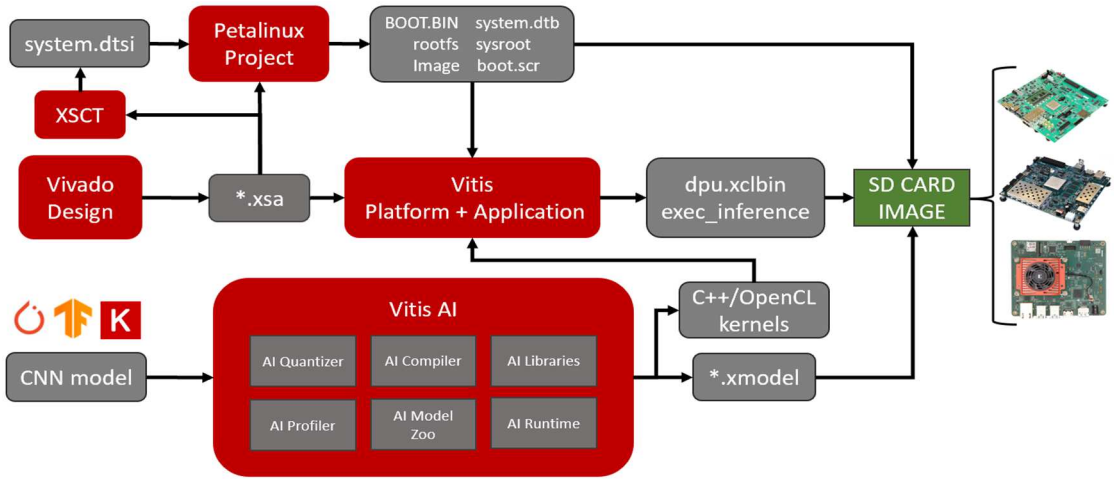

Fig. 3. DPU Hardware platform and model deployment diagram

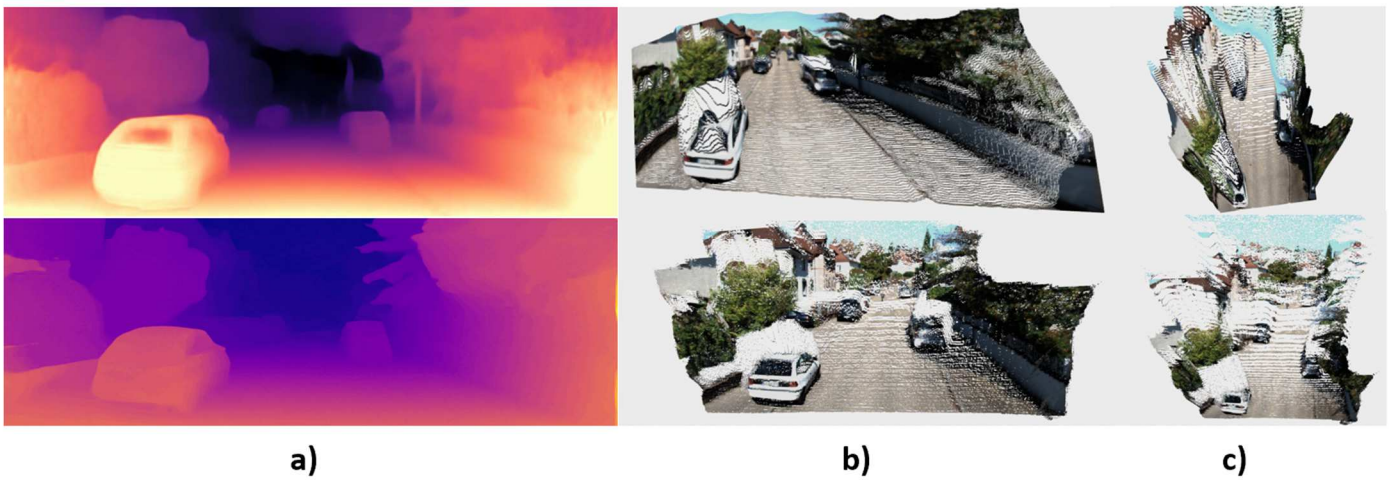

Fig. 4. This figure shows the 3D point cloud reconstruction (b) and the scene BEV(c) by following the two different approaches: monocular depth estimates (top a) and disparity estimation from a stereo-pair of images (bottom a).

The models were trained using data precision of FP32, which requires to be quantized since the DPU processing elements perform computations with INT8 precision. With a fixed-point precision, the model can perform computations requiring less memory and thus achieve a higher inference power efficiency. The models were quantized with the Vitis AI Post Training Quantization (PTQ) method, using cross-layer equalization [11] and AdaQuant layer calibration [12]. For the Vitis AI PTQ, a small sample of images was extracted from KITTI test set to perform calibration iterations. The resulting quantized model is followed by a compilation process. The Vitis AI compiler [13] will perform a graph intermediate representation (*XIR*) describing the neural network topology and its corresponding hyperparameters. The Vitis AI compiler will divide the DFG into multiple XIR subgraphs when some intermediate operation cannot be performed.

IV. Experiment Results

In the case of FADNet, the first approach was to directly use a trained and quantized model from the Vitis AI model Zoo repository. This model was subdivided into three graph object files (*xmodel*), two for the kernel computations on each stereo-pair image and one last for the subsequent point-wise correlation and disparity estimation. Since the cost volume computations cannot be performed on DPU, the subsequent subgraph will become a bottleneck to the whole inference pipeline.

Table II shows the model inference time on each target FPGA board. To reduce the inference time and achieve real-time processing, the network was reduced to use only the DispNetC module, however, after training and quantizing the model, the resulting disparity map from the DPU inference was very distorted compared to the full FADNet architecture. For future work, it is proposed to implement a kernel module for disparity denoising which takes as input the DispNet disparity and outputs a corrected disparity map. The monocular depth estimation was a simpler architecture by just compiling the autoencoder architecture into one *xmodel*. The results were not very distorted when comparing the quantized model inference to the original floating-point data precision. In terms of inference, the model

performed better by a large margin compared to the FADNet disparity model.

Another aspect to highlight was the inference task parallelism that was achieved when running the model on hardware platforms with multiple DPU cores. In the case of FADNet, the multi-DPU core parallelism was restricted due to the concatenation phase where part of the computation kernel requires to be performed directly on the Arm® processor. After the inference, the projection of the depth map to a 3D point cloud was run on the Arm® processor. Since the three MPSoC hold the same PS, the transformation performance was the same for all targeted boards. On average, the 3D projection took 3ms although some cases reached up to 10ms.

Table II. Model inference time (ms) on each target board

Model \ Inference	KV260	ZCU104	ZCU102
DispNetC	33.26	24.22	20.44
FADNet	526.33	512.28	510.67
MonoDepth	14.34	10.29	9.34

V. CONCLUSION

In this research, we designed a point cloud reconstruction system using stereo and monocular vision by creating a disparity map and then performing a 3D point cloud depth projection. Our plan for the future is to create a real-time stereo-video system for depth map and point cloud generation using two embedded cameras. This future work will be more involved with using HLS for hardware acceleration, especially for deploying Pseudo-LiDAR transformation as a compute kernel.

REFERENCES

[1] Ahmed, E., Saint, A., Shabayek, A.E., Cherenkova, K., Das, R., Gusev, G., Aouada, D., & Ottersten, B.E. (2018). A survey on Deep Learning Advances on Different 3D Data Representations. arXiv: Computer Vision and Pattern Recognition.

[2] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the deployment of deep neural networks on commercial edge devices," in 2019 IEEE International Symposium on Workload Characterization (IISWC), 2019, pp. 35–48

[3] Wang, Q., Shi, S., Zheng, S., Zhao, K., & Chu, X. (2020). FADNet: A Fast and Accurate Network for Disparity Estimation. 2020 IEEE International Conference on Robotics and Automation (ICRA), 101-107.

[4] Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., & Brox, T. (2016). A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4040-4048.

[5] Li, Juncheng, Faming Fang, Kangfu Mei and Guixu Zhang. "Multi-scale Residual Network for Image Super-Resolution." ECCV (2018).

[6] Godard, Clément, Oisin Mac Aodha and Gabriel J. Brostow. "Digging Into Self-Supervised Monocular Depth Estimation." 2019 IEEE/CVF International Conference on Computer Vision (ICCV) (2019): 3827-3837.

[7] Ronneberger, Olaf, Philipp Fischer and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." MICCAI (2015).

[8] He, Kaiming, X. Zhang, Shaoqing Ren and Jian Sun. "Deep Residual Learning for Image Recognition." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 770-778.

[9] Wang, Yan, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark E. Campbell and Kilian Q. Weinberger. "Pseudo-LiDAR From Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving." 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019): 8437-8445.

[10] Menze, Moritz and Geiger, Andreas Object Scene Flow for Autonomous Vehicles, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2015.

[11] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, Max Welling: "Data-Free Quantization Through Weight Equalization and Bias Correction", 2019, The IEEE International Conference on Computer Vision (ICCV), 2019.

[12] Jhunjhunwala, Divyansh, Advait Gadhikar, Gauri Joshi and Yonina C. Eldar. "Adaptive Quantization of Model Updates for Communication-Efficient Federated Learning." ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2021): 3110-3114.

[13] Mingzhen Li, Yi Liu, Xiaoyan Liu, Qingxiao Sun, Xin You, Hailong Yang, Zhongzhi Luan, Lin Gan, Guangwen Yang, Depei Qian: "The Deep Learning Compiler: A Comprehensive Survey", 2020, IEEE Transactions on Parallel & Distributed Systems, vol. 32, no. 03, pp. 708-727, 2021.