A-U3D: A Unified 2D/3D CNN Accelerator on the Versal Platform for Disparity Estimation

Tianyu Zhang, Dong Li, Hong Wang, Yunzhi Li, Xiang Ma, Wei Luo, Yu Wang, Yang Huang, Yi Li,
Yu Zhang, Xinlin Yang, Xijie Jia, Qiang Lin, Lu Tian, Fan Jiang, Dongliang Xie, Hong Luo, Yi Shan
AMD
{ tianyu.zhang, d.li, hong.wang, yunzhi.li, xiang.ma, wei.luo, yu.w, huang.yang, yi.li,
alex.zhang, xinlin.yang, xijie.jia, qiang.lin, lu.tian, f.jiang, dongliang.xie, hong.luo, yi.shan }@amd.com

***Abstract*—3-Dimensional (3D) convolutional neural networks (CNN) are widely used in the field of disparity estimation. However, 3D CNN is more computationally dense than 2D CNN due to the increase in the disparity dimension. To enable more practical applications in autonomous driving, robotics, and other scenarios on embedded devices, we propose a unified 2D/3D CNN accelerator (A-U3D) design. This design unifies 3D standard / transposed convolution into 2D standard convolution, respectively. Our processing unit can support 2D and 3D convolution in the same mode without additional structures. Based on PSMNet, a 3D-based CNN for disparity estimation, we build a heterogeneous multi-core system integrated with A-U3D in conjunction with CPU, DSP, and AI Engines on the Xilinx Versal ACAP platform. Running the pruned 8-bit model, our A-U3D system achieves 0.289s latency, which is 11.5× faster than the state-of-the-art solution on the same platform, and reaches an end-to-end (E2E) performance of 10.1 frames per second (FPS). Our proposed system explores the feasibility of deploying 3D CNNs with large workloads on FPGA.**

***Index Terms*—3D-CNN, PSMNet, FPGA, Versal, AI Engine, Hardware Acceleration, Heterogeneous architecture**

I. Introduction

Disparity estimation is a fundamental computer vision task which predicts the disparity for each pixel given a pair of rectified stereo images and has wide applications in autonomous driving, robotics, etc. Recent CNN-based methods apply 2D/3D convolution and disparity regression for disparity estimation. 2D CNN based methods [1], [2] often require more complicated cost volume construction (e.g., correlation). 3D CNN based methods [3]–[6] are dominant in this field and can simplify the cost volume construction. PSMNet [4] is one of the popular 3D CNN-based solutions with good fidelity results but consumes a large computation overhead. In this work, we aim to provide a reference solution for the general disparity estimation task on embedded devices.

How to efficiently implement convolution is the key of designing CNN accelerators [7]. Prior works have proposed various accelerator schemes on different platforms, including CPU, GPU [8], ASIC, and FPGA. In light of the stability and efficiency, we adopt FPGA and design the accelerator on the Xilinx Versal ACAP platform [9]. As a hybrid computing platform [10], ACAP integrates CPU (ARM-Advanced RISC Machines), FPGA, and AI Engine (AIE).

To run neural networks efficiently, many FPGA-based convolution accelerators have been proposed [11], [12]. These accelerators basically support common operations in CNN, such as 2D convolution (CONV2D) [13]. To further improve efficiency, [14] implements efficient low-bit computation on FPGA. With the development of networks, 3D convolution (CONV3D) has become more and more widely used [15]. However, the computational complexity and memory overhead of CONV3D are higher than CONV2D, which hinders its deployment on embedded devices. Researchers started to propose new accelerators for 3D CNN. In [16], the authors propose a hardware-aware pruning approach that can compress the model for acceleration of inference. Some 3D accelerators are designed as dedicated computational structures in the temporal direction for CONV3D tasks [17], [18]. But these designs also face some issues. To process CONV3D, additional overhead is added in the temporal direction, such as addition tree and multiplexer. Some accelerators proposed some efficient specific 3D CNN models for their own structures, which lack some generality. Moreover, more research focuses on lightweight 3D models, such as C3D, P3D, etc., which lack the deployment possibility of large workload models.

To alleviate these issues, we first prune and quantize the model since 3D convolution consumes large FLOPs (e.g., 2.16T FLOPs for PSMNet). Then, we design a unified 2D/3D CNN accelerator on the Xilinx Versal ACAP platform, which can support both 2D and 3D convolution. Furthermore, we integrate a heterogeneous multi-core solution for disparity estimation. Our main contributions are summarized as follows:

- We unify the CONV3D into the basic CONV2D operation so that the calculation unit can process the two calculations simultaneously in the same mode. We also optimize the computation of the transposed CONV3D to improve the efficiency of A-U3D.
- We design a heterogeneous architecture on the Versal Platform to enable CPU (ARM), DSP, and AI Engines computing in parallel under the scheduling by CPU.
- We propose a solution based on A-U3D for disparity estimation. This system can achieve an E2E performance of 10.1FPS for our pruned 8-bit PSMNet.

II. Model Compression

A. Pruning

Pruning is used to reduce the number of parameters and FLOPs of the neural network. There are two common approaches to pruning: unstructured [19] and structured [20]. Unstructured pruning removes any unimportant weights and structured pruning removes filters of convolution layers. Unstructured pruning allows pruning of the network at the finest granularity. However, most hardware cannot accelerate sparse computations unless they are specifically designed to do so. We chose structured pruning because no hardware dedicated for pruning model is required. The importance of a filter in the convolution layer is measured by the sum of its absolute weights, i.e. L1-norm. Filters with the smallest L1-norm will be removed. The pruning process is done in an iterative approach. We remove some filters from the model, then retrain this model, then remove more filters from the model, and retrain it again. Pruning the model to the target size in one step will cause much loss of model accuracy that is too difficult to recover. Pruning, followed by retraining, is called one iteration. We perform 5 iterations to get the pruned model.

B. *Quantization*

Most current Neural Networks are over-parameterized, so we have the opportunity to further decrease the memory footprint, power and latency by reducing precision without impacting accuracy.

To move from floating-point to efficient fixed-point operations, a scale is used to map the floating-point values to integers. For 8-bit integers, we can represent 256 different values. Uniform quantizer is the mainstream scheme that can accelerate computing using high-throughput parallel. The quantization function is as follows:

$$Q(r) = Clip(Round(r/s)) - z \tag{1}$$

where Q is the quantization operator, r is the real values, s is a scale factor, and z is an integer zero-point which maps to the real zero. The *clip* is a function that clips outliers that fall outside the range of integer representation, $[-127, 128]$ for signed INT8. Round is a function to round floating numbers to integers. It is possible to undo the scaling step to recover the real values r from the quantized values, which is called De-quant operation:

$$\overline{r} = s(Q(r) + z) \tag{2}$$

Note that the recovered real values $\overline{r}$ will not be exactly equal to r due to the round operation. For efficient fix-point inference on FPGA, we introduce some constraints on the quantization scheme. We will let $z = 0$ in equation (1) to reduce the computational overhead of dealing with zero-point in matrix multiply or convolution operation. We choose the per-tensor quantization with a single scale factor. The scale factor is restricted to a power-of-two, $s = 2^{-n}$ (n can be positive or negative. When n is positive, it denotes the length of the fractional part of the fix-point number). This enables scaling with s corresponding to simple and efficient bit-shift.

Post-Training Quantization (PTQ) performs quantization with calibration and adjusts weight without supervised training [21]. Quantization-Aware Training (QAT) retrains the neural

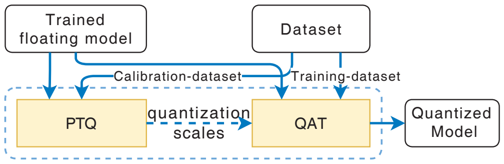

Fig. 1. PSMNet Quantization Flow. In PTQ, a pretrained floating model is calibrated with a small calibration dataset. In QAT, the floating model is converted to a quantized model by inserting linear quantizers. This quantized model uses training data to adjust parameters (e.g., weight, bias, threshold, etc.) and recover accuracy loss. The quantization scales calculated from PTQ can be adopted as the initial values for the learnable parameters in QAT.

network with quantized parameters so that the model can converge to a better loss [22]. To achieve a fully quantized model, we apply both PTQ and QAT in the PSMNet optimization process as shown in Fig.1. Multiple approaches such as weight equalization, bias correction, and training quantization thresholds (TQT) [23], [24] have been employed to push the performance of the quantized 8-bit model to the extreme.

III. Unified 2D/3D Accelerator

Based on PSMNet, we propose a unified 2D/3D neural network accelerator (A-U3D) on the Xilinx Versal ACAP platform. A-U3D is a general design in two aspects. First, A-U3D can be applied to other 3D CNN based models for disparity estimation. Second, A-U3D also supports well-known 2D CNN such as VGG, ResNet, and MobileNets. For PSMNet, apart from optimizing for 3D convolution, the proposed system makes full use of the advantages of the Versal heterogeneous platform to implement other special operations by collaboration between CPU, DSP, and AIE. Fig.2 illustrates the proposed data flow mapped to the heterogeneous engines for PSMNet. Fig.3 shows the top-level block diagram of the proposed system design.

We build a CONV engine on the AIE array to implement CONV2D. The CONV engine supports multi-batch implementation. We use a shared weight design among batches, which enables lower DDR bandwidth requirements. We propose a method to split the CONV3D task into a CONV2D operator implementation. And we eliminate a large number of invalid computations by construction fusion of transposed convolution (TCONV). The other non-CONV operations in CNN, such as element-wise addition, pooling, etc., are aggregated into a general engine on the PL side called ALU to simplify both the hardware and software design and reduce the PL resource utilization. The ALU engine uses the DSP as the computational unit and its parallelism (number of DSPs) can be adjusted

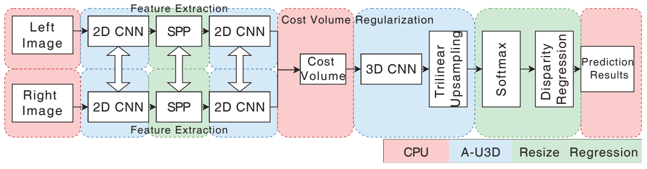

Fig. 2. PSMNet network and mapping of data flow and heterogeneous system.

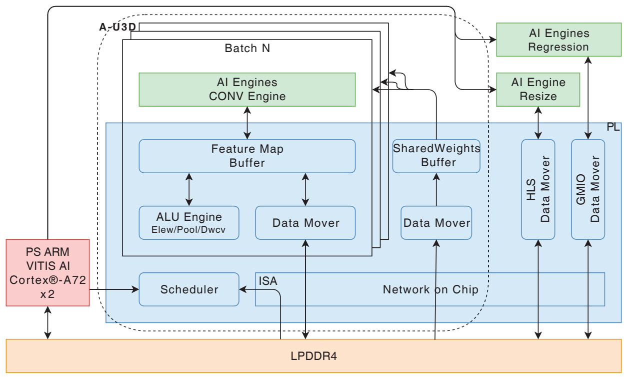

Fig. 3. The top level block diagram of the proposed system design.

according to the workload. The AIE array and ALU engine can calculate in parallel, improving the calculation efficiency.

A. Implementation of 2D Standard Convolution

The input feature map in CONV2D has 3 dimensions, which are height (IH), width (IW), channel (IC). Similarly, the dimensions of the output feature map are OW, OH, and OC. The weight kernel in CONV2D has 4 dimensions, which are kernel height (KH), kernel width (KW), input channel (IC), and output channel (OC). Each kernel needs to slide along the IW and IH dimensions, as shown in Fig.4(a).

The CONV engine in A-U3D consists of 16 2-core-cascaded units. Each unit does 2×128 MAC operations per cycle. Corresponding to convolution, the whole engine can implement a input feature map matrix ($8 \times 1 \times 16$) multiplied by a weight matrix (16×32) to get a output feature map matrix ($8 \times 1 \times 32$) per cycle. For the $8 \times 1 \times 16$ input feature map matrix, 8 is the parallelism of feature map height (OHP), 1 is the parallelism of feature map width (OWP), 16 is the parallelism of the input channel (ICP). For the 16×32 weight matrix, 16 is ICP, 32 is the parallelism of output channel (OCP). According to the MAC task of CONV2D, after $ceil(KH \times KW \times IC/ICP)$ accumulation cycles, the output feature map of $OHP \times OWP \times OCP = 8 \times 1 \times 32$

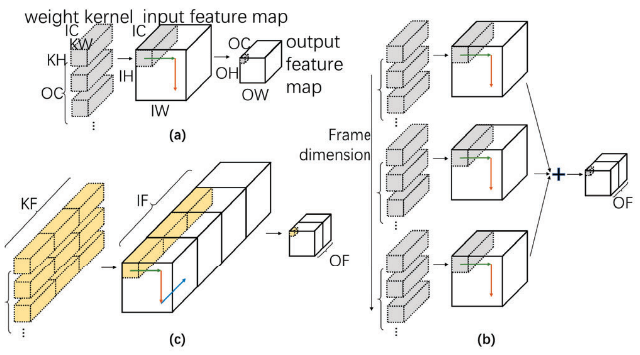

Fig. 4. Differences and unification between 2D and 3D convolutions. (a) 2D convolution. (b) 3D convolution. (c) The unification of CONV3D into CONV2D operation.

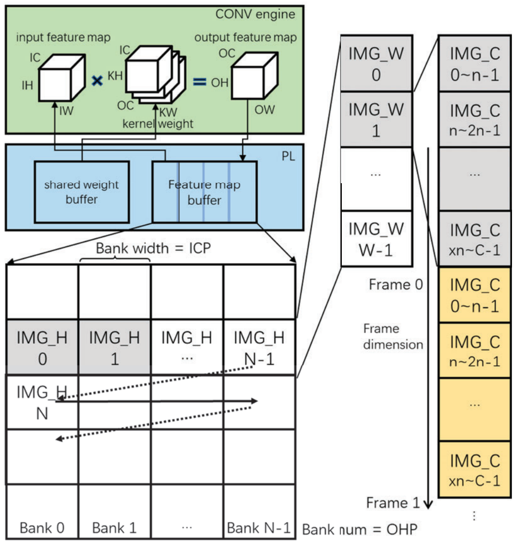

Fig. 5. The pattern of the feature map in the data buffer.

can be obtained. Thus, the complete convolution output can be obtained through multiple iterations ($ceil(IH/OHP) \times ceil(IW/OWP) \times ceil(OC/OCP)$).

To efficiently obtain the data for computation, we buffer the feature map and weight in the on-chip RAM on the PL side. The kernel data is a priori data, so its data pattern can be shuffled in advance. The feature map data is obtained by real-time calculation, so it is necessary to locate enough data as easily as possible when storing it in the RAM. We divide the on-chip RAM resources into banks. The data pattern in the bank is buffered by channel, followed by width, and finally by height, as shown in the gray part in Fig.5.

In this bank design, the number of banks is equal to OHP, and the width of each bank is ICP. When calculating CONV2D, the feature map data can be read from the bank by $OHP \times ICP$ per cycle. The CONV engine only needs to sequentially read the data in the bank when performing the MAC tasks in the IC and IW dimensions. Take a 2×2 weight kernel as an example. To calculate the first data of the output feature map ($0_{OH}, 0_{OW}, 0_{OC}$), we need to MAC the input feature map data ($0 \sim 1_{IH}, 0 \sim 1_{IW}, all_{IC}$) in gray part of Fig.5 with the first the kernel. These feature map data can be located by bank index and bank address. For the next data in the width direction of the output feature map ($0_{OH}, 1_{OW}, 0_{OC}$), we only need to add the step to the bank address of the original feature map (gray). The direction of height is the same way. In this way, the data required by each iteration of the CONV2D can be obtained.

B. Unity of 2D and 3D Convolution

CONV3D has one more frame dimension (IF) than CONV2D. The kernel also increases the frame dimension (KF), and the sliding direction is increased to three dimensions, which are IW, IH, and, IF. For the MAC task, the results of the 2D convolution need to be additionally accumulated in the frame dimension, as shown in Fig.4(b).

If addition logic is added for the extra dimension, the engine will reduce the computational efficiency when computing CONV2D. To support both CONV3D and CONV2D, we unify the CONV3D into the CONV2D by putting the data of the IF dimension behind the IC dimension, as shown in Fig.4(c). For the CONV engine, only the original three dimensions need to be processed, and the newly added IF dimension is merged into the IC dimension. The MAC task in frame dimension is accomplished by increasing the cycles from $ceil(KH \times KW \times IC/ICP)$ to $ceil(KH \times KW \times KF \times IC/ICP)$. With a granularity of 1 in the frame dimension, the number of iterations required for calculation increases $ceil(OF/1)$ times. For the feature map pattern, The data in the frame dimension is arranged sequentially after the channel dimension, as shown in the yellow part of Fig.5. By splitting CONV3D, the engine can maintain the same efficiency in the calculation that is not aware of the frame dimension. This reduces the complexity of the design while increasing generality.

C. *Optimization of Transposed Convolution*

Usually when transposed convolution is calculated, we convert it to direct convolution for processing. Because the accelerator has paid a lot of resources to build the convolutional computing array. When using direct convolution to simulate transposed convolution, many columns and rows of zeros need to be added to the input, resulting in wasted data bandwidth and computationally inefficient [25]. Taking Fig.6(a) as an example, when the equivalent convolution with 1 zero inserted between inputs, it contains a large number of zeros so that most computations can be considered redundant [26].

Fig.6(a) illustrates a transposed convolution example that convolves an enlarged 4×4 input feature map and 3×3 weight kernel to generate output. To avoid unnecessary zero computations, the weight kernel can be decomposed into four cases, which are center (1×1), two horizontal endpoints (1×2), two vertical points (2×1), and four corners endpoints (2×2) as shown in the middle of Fig.6(b). Then the decomposed weight kernels just multiply with normal input directly without zero insertion [27]. The input data corresponding to the four decomposed weight kernels is different, thus four types of convolution should be generated for hardware computation.

In this section, an optimization method for the process of transposed convolution as mentioned above is proposed by construction fusion. To implement the reduction of generated constructions, the 4 types of weight kernels are filled with zeros on the right and bottom respectively, and all of them are expanded to four endpoints (2×2), as shown on the right side of Fig.6(b). After that weight dilation, the input feature map of the four decomposed weight kernels is dense and identical, which is described in Fig.6(c). Due to the same input data and decomposed weight kernel's size, the four types of convolution can be fused into one set of convolution. This optimization allows the four weight kernels to reuse the same input feature map, thereby reducing the feature map bandwidth requirement. The impact of the increased weight data volume will be compensated by the shared weight design. The added redundant calculation is a balance of the mismatch between bandwidth and computing power.

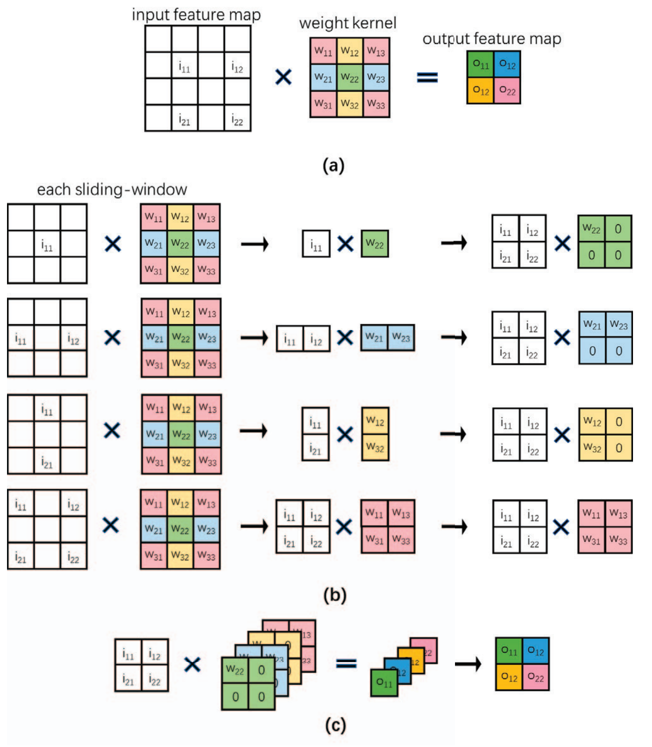

Fig. 6. Optimization of Transposed Convolution. (a) A transposed convolution example with 2×2 input and 3×3 kernel to generate enlarged 2×2 output by inserting zeros between the adjacent input. Four kinds of nonzero computations exist as denoted with different color squares in the Weight and Output. (b) The optimization method is to decompose the weight of 3×3 and enlarge it to a uniform size (2×2). (c) Restoration of output results.

IV. Heterogeneous Design

In this PSMNet case, the operations of bi-linear interpolation up-sample, concatenation, softmax, and cost volume would increase the complexity of the data flow if integrated into A-U3D. Adaptability and scalability are the most powerful features of Xilinx's latest Versal devices. So we analyzed the tasks that can map to ARM and AIEs and proposed two heterogeneous modules that co-working with A-U3D to achieve maximum computational efficiency as shown in Fig.7.

A. *Resize Module*

The *resize* module handles two operations in SPP of PSMNet: bi-linear interpolation up-sample and concatenation. The input data of this module is six tensors of different sharp, four of these input tensors are unsampled to $[144 \times 240 \times 32]$ then they are concatenated with the other two input tensors in the channel direction, finally we get the output tensor, the shape of output tensor is $[144 \times 240 \times 320]$.

We use HLS to develop a data mover on the PL side as its cache to access the random data in the external memory. The data mover does the data rearrangement work and generates the factors used by bi-linear interpolation up-sample. Producer data mover unit can do random access in external memory

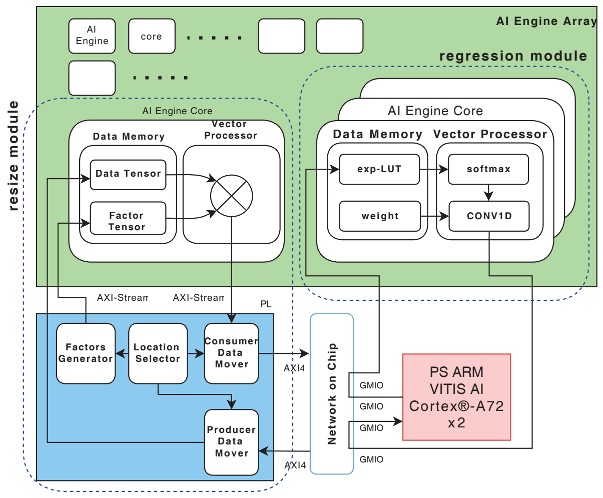

Fig. 7. The architecture of resize module and regression module.

and transform to the AIE. The result will be conveyed to the consumer data mover unit which writes the result to the external memory. The concatenation works have also been done by the data mover unit in PL to optimize the critical performance bottleneck and balance the workload among engines. The location selector unit is configured to arrange data partitioning and generates control signals to other units.

B. Regression Module

The regression module handles two operations in regression part of the PSMNet: softmax of 192 input and 1D convolution. The AIE core can do sequential access from external memory using the interface in PS, called GMIO.

The input data type is an 8-bit integer, so, here we can use lookup table (LUT) to process the exponential function in softmax operation. we only need to generate a LUT with a length of 256 that can complete the softmax operation efficiently. The value of the LUT is generated by PS and transmitted to the AIE core's data memory through the GMIO. Finally, the result of softmax is convolved with a fixed 192-length sequence to get the final result and written to external memory through GMIO.

C. Cost Volume Computation

Cost volume operation is performed by the ARM. To adapt to fixed-point computing on FPGA, the fixed-point position of the input and output needs to be considered when fusing the left and right images. In the original paper, the dimensions of the inputs are $NCHW$, where W changes in each round of depth estimation. The dimension used in the calculation on FPGA is $NHWC$, and the size of C is constant each time. Therefore, we use Single Instruction Multiple Data (SIMD) technology to batch the cost volume based on constant C.

V. Evaluation

The proposed system is implemented on the Xilinx VCK190 evaluation board [28], which features Xilinx Versal ACAP

TABLE I
The Resource Utilization of each module in the system.

	3D/2D Accelerator	resize	regression
AIE Cores	96	1	8
LUTs	248837	5610	0
FF	302979	11965	0
DSP	413	13	0
BRAM	294	6	0
URAM	391	0	0
PL NMU	18	2	0

XCVC1902 device. The system is built with the Xilinx Vitis 2021.1 tool-chain environment. The system includes 3 modules: A-U3D, *resize*, and *regression*. The A-U3D supports batch instantiation. The configuration is set as 3 batch (totally 96 AIE cores are adopted). Considering that more batches will affect the latency performance in the limited DDR bandwidth, we did not use up the 400 AIE cores of the board.

A. Accuracy

We evaluate the accuracy of PSMNet on the standard Scene Flow benchmark. Scene Flow is a large-scale stereo dataset that contains 35454 training and 4370 test images. Each image is resized to 576×960 to the network for inference. We use the standard end-point error (EPE) as the accuracy metric that computes the average L1 pixel error (lower is better).

Fig.8 shows the accuracy of our compressed PSMNet. Our reproduced PSMNet achieves 0.902 EPE, which is better than [4]. By pruning, we can obtain comparable accuracy with the unpruned model. Fig.9 shows the number of output channels of convolutional layers before and after pruning, which means the pruned model has 68% less FLOPs than the unpruned model. We obtain a similar EPE (0.961 vs. 0.902) with the unpruned model. By QAT, we can improve the result of PTQ and obtain the 8-bit model with slightly increased EPE compared to the floating model (1.022 vs. 0.961).

B. Performance

Fig.10 is the profiling of the system's pipeline calculation for 2 frames when running PSMNet. These four modules can be pipelined in parallel to achieve the highest efficiency. When the pipeline computes multiple frames, the overhead of all

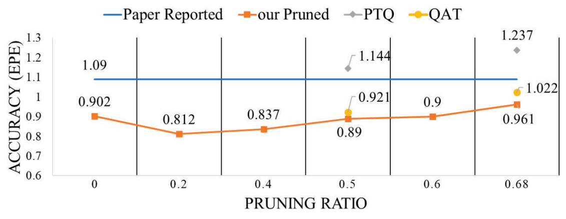

Fig. 8. Pruning and Quantization Results.

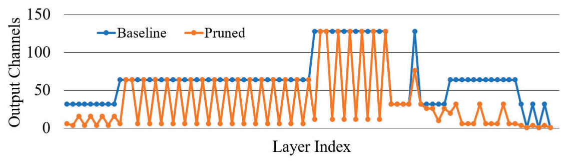

Fig. 9. The number of output channels of convolutional layers.

TABLE II
PERFORMANCE COMPARISON ON DIFFERENT PLATFORMS.

	CPU	**GPU [4]**	**GPU**				**FPGA [29]**	**Our Work**
Env	Intel(R) Xeon(R) Gold 6136	Nvidia GTX Titan Xp	Nvidia Jetson AGX Xavier				Xilinx ACAP XCVC1902	Xilinx ACAP XCVC1902
Frequency	3.00GHz	1.58GHz	2.26GHz				PL@333MHz AIE@1.33GHz	PL@333MHz AIE@1.33GHz
Power	150W(TDP)	250W(TDP)	30W(TDP)				125W(TDP)	37.6W(Measured)
Model	PSMNet	PSMNet	PSMNet	PSMNet	PSMNet_8bit	PSMNet_8bit	PSMNet_8bit	PSMNet_8bit
Accuracy	1.09 EPE	1.09 EPE	0.902 EPE	0.961 EPE	1.256 EPE	1.022 EPE	1.022 EPE	1.022 EPE
Workload	2.16TOP	2.16TOP	2.16TOP	696GOP	2.16TOP	696GOP	696GOP	696GOP
Latency	32.157s	0.41s	1.602s	1.140s	0.656s	0.511s	3.333s	0.289s

TABLE III
COMPARISON WITH PREVIOUS IMPLEMENTATIONS ON FPGA.

	2D [30]	**3D [30]**	**[18]**	**[31]**		**[11]**	**Our Work**		
Env	Xilinx VU VUS440	Xilinx VU VUS440	Intel Arria 10 SX660	Xilinx ZU+ ZCU102		Xilinx ZU+ ZCU102	Xilinx ACAP VCK190		
Frequency	200MHz	200MHz	150MHz	200MHz		333MHz	PL@333MHz AIE@1.33GHz		
Power	26W	26W	36W	10.2W		15.1W	39.5W	29.5W	37.6W
Model	VGG16	C3D	E3DNet	VGG16	C3D	FADNet	VGG16	FADNet	PSMNet
Type	2D_16bit	3D_16bit	3D_32bit	2D_8bit	3D_8bit	2D_8bit	2D_8bit	2D_8bit	3D_8bit
Workload	31GOP	38.2GOP	6.1GOP	31GOP	38.2GOP	154GOP	31GOP	154GOP	696GOP
Latency	37.5ms	49.1ms	35.3ms	27ms	28.2ms	601ms	3.2ms	123ms	289ms
FPS	26.7	20.5	29.3	37	35.4	2.4	353.3	13.9	10.1
Throughput	821GOP/s	784.7GOP/s	178.7GOP/s	1150GOP/s	1353GOP/s	369.6GOP/s	10952GOP/s	2140GOP/s	7029.6GOP/s
GOP/W	31.6	30.2	4.96	112.7	132.6	24.5	277.3	72.5	187

but the slowest modules is hidden by the pipeline. Through the optimization in Section IV-C, the cost volume time is optimized from 100.7ms to 41.8ms.

As we know, there are no many implementations of large workload based on FPGA for its complicated network. We compared with different platforms in Table II. Compared with [4], our work achieves 67× equivalent GOP/W with pruned 8-bit model. We demonstrate the effect of pruning and quantization using AGX. And our work shows 1.8× better latency than GPU. Compared to state-of-the-art accelerators of the same architecture, 0.7FPS, we achieve 10.1FPS and 14.4x E2E FPS improvements.

We deploy different models on our work to further demonstrate our performance and generality in Table III. We show an E2E throughput of 10952GOP/s in the lightweight workload model, which is 8.1× higher than [31]. We provide an evaluation of the model FADNet for 2D disparity estimation with a model accuracy of 1.158 EPE. We have 4.9× lower latency than [11]. While the accuracy of PSMNet using 3D CONV reaches 1.022 EPE, the workload is significantly improved to 696GOP. For such complex models with large workloads, our work achieves an E2E performance of 10.1FPS. Our throughput is 5.2× better than 8-bit accelerators, and more than 10× better than high-bit-width accelerators, which still has significant advantages after considering the device process and quantization accuracy. After considering the power consumption factor, we achieved the optimal performance of 277.3GOP/W in the table, and the performance of 187GOP/W under the deployment of a large model. Therefore, it can be concluded that our proposed solution is more practical than other accelerators.

VI. CONCLUSION

This paper proposes a unified 2D/3D CNN accelerator to solve the CONV3D task in disparity estimation. Our proposed optimization is general and can support more 3D convolution related models and tasks. Aiming at PSMNet, a multi-core heterogeneous system is designed on the Xilinx Versal ACAP platform. The system achieves 0.289s latency and E2E 10.1FPS running the pruned 8-bit model. This solution provides a reference design for deploying larger workloads and more complex models in real-world applications.

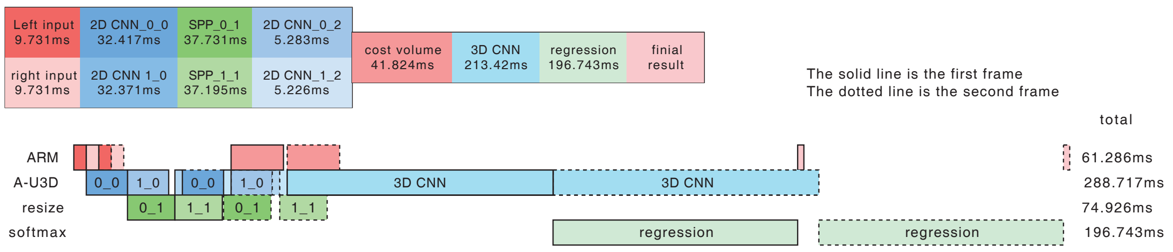

Fig. 10. The profiling and latency of each module when running PSMNet. 2 frames are depicted to show the pipeline between modules.

References

[1] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, “A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4040–4048.

[2] Q. Wang, S. Shi, S. Zheng, K. Zhao, and X. Chu, “Fadnet: A fast and accurate network for disparity estimation,” in *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2020, pp. 101–107.

[3] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, “End-to-end learning of geometry and context for deep stereo regression,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 66–75.

[4] J.-R. Chang and Y.-S. Chen, “Pyramid stereo matching network,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5410–5418.

[5] F. Zhang, V. Prisacariu, R. Yang, and P. H. Torr, “Ga-net: Guided aggregation net for end-to-end stereo matching,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 185–194.

[6] X. Guo, K. Yang, W. Yang, X. Wang, and H. Li, “Group-wise correlation stereo network,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3273–3282.

[7] F. Sun, C. Wang, L. Gong, Y. Zhang, C. Xu, Y. Lu, X. Li, and X. Zhou, “Unicnn: A pipelined accelerator towards uniformed computing for cnns,” *International Journal of Parallel Programming*, vol. 46, no. 4, pp. 776–787, 2018.

[8] W. Jiang, Y. Chen, H. Jin, R. Zheng, and Y. Chi, “A novel gpu-based efficient approach for convolutional neural networks with small filters,” *Journal of Signal Processing Systems*, vol. 86, no. 2, pp. 313–325, 2017.

[9] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, “Xilinx adaptive compute acceleration platform: Versaltm architecture,” in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 84–93.

[10] Xilinx, *Versal ACAP AI Engine Programming Environment User Guide (UG1076)*, 2021. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1076-ai-engine-environment/Overview

[11] ——, *DPUCZDX8G for Zynq UltraScale+ MPSoCs. Product Guide.*, 2021. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documentation/ip_documentation/dpu/v3_3/pg338-dpu.pdf

[12] L. Bai, Y. Zhao, and X. Huang, “A cnn accelerator on fpga using depthwise separable convolution,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 10, pp. 1415–1419, 2018.

[13] R. Ding, G. Su, G. Bai, W. Xu, N. Su, and X. Wu, “A fpga-based accelerator of convolutional neural network for face feature extraction,” in *2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*. IEEE, 2019, pp. 1–3.

[14] T. Zhang, T. Han, L. Tian, Y. Li, X. Jia, G. Liu, P. An, Y. Tan, L. Sui, S. Fang *et al.*, “Lpac: A low-precision accelerator for cnn on fpgas,” in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 316–316.

[15] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3d convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[16] M. Sun, P. Zhao, M. Gungor, M. Pedram, M. Leeser, and X. Lin, “3d cnn acceleration on fpga using hardware-aware pruning,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[17] H. Wang, M. Shao, Y. Liu, and W. Zhao, “Enhanced efficiency 3d convolution based on optimal fpga accelerator,” *IEEE Access*, vol. 5, pp. 6909–6916, 2017.

[18] H. Fan, C. Luo, C. Zeng, M. Ferianc, Z. Que, S. Liu, X. Niu, and W. Luk, “F-e3d: Fpga-based acceleration of an efficient 3d convolutional neural network for human action recognition,” in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 1–8.

[19] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” *Advances in neural information processing systems*, vol. 28, 2015.

[20] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” *arXiv preprint arXiv:1608.08710*, 2016.

[21] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, “Data-free quantization through weight equalization and bias correction,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.

[22] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” *arXiv preprint arXiv:1806.08342*, 2018.

[23] S. Jain, A. Gural, M. Wu, and C. Dick, “Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks,” *Proceedings of Machine Learning and Systems*, vol. 2, pp. 112–128, 2020.

[24] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, “Pact: Parameterized clipping activation for quantized neural networks,” *arXiv preprint arXiv:1805.06085*, 2018.

[25] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *arXiv preprint arXiv:1603.07285*, 2016.

[26] D. Im, D. Han, S. Choi, S. Kang, and H.-J. Yoo, “Dt-cnn: Dilated and transposed convolution neural network accelerator for real-time image segmentation on mobile devices,” in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[27] K.-W. Chang and T.-S. Chang, “Efficient accelerator for dilated and transposed convolution with decomposition,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[28] Xilinx, *VCK190 Evaluation Board User Guide (UG1366)*, 2021. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documentation/boards_and_kits/vck190/ug1366-vck190-eval-bd.pdf

[29] ——, *DPUCVDX8G for Versal ACAPs Product Guide.*, 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/pg389-dpucvdx8g/Introduction

[30] J. Shen, Y. Huang, Z. Wang, Y. Qiao, M. Wen, and C. Zhang, “Towards a uniform template-based architecture for accelerating 2d and 3d cnns on fpga,” in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 97–106.

[31] H. Deng, J. Wang, H. Ye, S. Xiao, X. Meng, and Z. Yu, “3d-vnpu: a flexible accelerator for 2d/3d cnns on fpga,” in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 181–185.