ScaleFlow: High-Level Synthesis for Large Dataflow Applications

Hanchen Ye, Deming Chen
University of Illinois at Urbana-Champaign
hanchen8@illinois.edu, dchen@illinois.edu

Abstract

Dataflow architectures are growing in popularity due to their potential to mitigate the challenges posed by the memory wall inherent to the Von Neumann architecture. At the same time, High-level synthesis (HLS) has demonstrated its efficacy as a design methodology for generating efficient dataflow architectures within a short development cycle. However, existing HLS tools rely on developers to explore the vast dataflow design space or are structured in a manner that ultimately leads to suboptimal designs. This phenomenon is especially concerning as the number of HLS designs grows. To tackle these challenges, we introduce ScaleFlow*, a new scalable and hierarchical HLS framework that can systematically convert an algorithmic description into a dataflow hardware implementation. We propose a collection of efficient and versatile dataflow representations for modeling the hierarchical dataflow structure within ScaleFlow. Capitalizing on these representations, we develop an automated optimizer that decomposes the dataflow optimization problem into multiple levels based on the inherent dataflow hierarchy, achieving scalable task partitioning, dataflow scheduling, and parallelization. Based on FPGA evaluations using a set of neural networks, ScaleFlow achieves up to 8.54× higher throughput compared to the state-of-the-art (SOTA) HLS optimization tool. Furthermore, despite being fully automated to handle different models, remarkably, ScaleFlow can achieve 1.29× higher throughput over the SOTA RTL-based neural network accelerators on an FPGA.*

1. Introduction

With the decline of Moore's law, the industry and the research community are being forced to rethink how we can extract that extra bit of performance even when technology scaling is stopping. In this context, *customized* and *domain-specific* accelerators are becoming well accepted in combating the physical limitations of silicon, including those implemented on ASICs [8, 20, 11] and reconfigurable platforms, such as FPGAs [45, 51, 42].

Dataflow Architecture. An important computation architecture for customized hardware accelerators is *dataflow*, which enables the parallel temporal execution of multiple processors or coarse-grained tasks [27, 31, 47]. Unlike the Von Neumann-based architecture that constantly grapples with the memory wall, dataflow architecture can exploit the streaming or tile buffering between adjacent tasks to avoid frequent external memory access. As long as an application is dataflow feasible, a well-designed dataflow architecture can efficiently execute the application with reduced power and bandwidth utilization [37, 46, 38, 10].

High-level Synthesis. Historically, the cost of developing customized hardware accelerators has always remained astronomically high. In this context, high-level synthesis (HLS) is a promising solution that can *synthesize* high-level algorithmic description to an equivalent RTL implementation [5]. The higher level of abstraction allows the designer to experiment with various design choices easily, shortening the design space exploration (DSE) phase and avoiding suboptimal design points [33, 22]. Furthermore, given that HLS can enable rapid evaluation of different design points, many HLS-augmentation tools have further improved the quality of HLS-generated accelerators [34, 4].

Existing Dataflow Approaches. To implement dataflow architectures using commercial HLS tools, users must use high-level programming interfaces such as AMD Vitis HLS `dataflow` compiler directive [17], Intel HLS *system of tasks* [18], and LegUp `thread` APIs [19]. However, it is difficult for users to implement a dataflow-oriented HLS design with sequential languages, such as C/C++. Therefore, academic HLS tools have pushed for approaches that define the hierarchical dataflow structure through decoupled hardware customization primitives [2, 24, 39, 16] or by using specialized primitives [26, 23, 3, 12]. These approaches have effectively improved productivity and quality compared to industrial HLS tools. Note that there also exist recent frameworks [40, 9] that can automatically generate dataflow design without manual code rewriting. However, these automated frameworks cannot model dataflow architectures systematically, limiting them to the generation of simple designs with suboptimal quality.

Unexplored Opportunities. Although existing HLS tools can enable dataflow designs, they still heavily rely on the user to make the hard design decisions, including parallelization strategy, tiling strategy, memory hierarchy, data layout, etc. More importantly, the design spaces of different tasks in the dataflow are tightly coupled with one another due to two reasons: (1) an efficient dataflow architecture demands balanced latency across different tasks because the critical task determines the overall achievable performance; (2) the inter-task communications are often established through streaming channels or on-chip buffers instead of hierarchical shared memory. Meanwhile, large-scale dataflow often gravitates towards a *hierarchical* structure, as dataflow tasks are naturally represented by nested graphs, further complicating the design space.

As a result, the vast design space can prohibit programmers from reasoning about various design choices and finding the

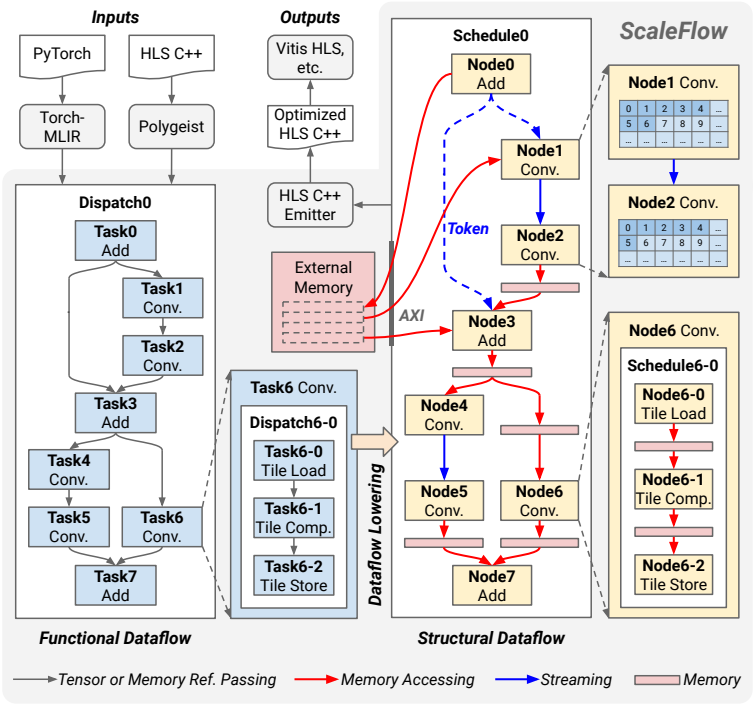

Figure 1: ScaleFlow framework overview.

Table 1: ScaleFlow-IR key operations. *Region* is a sequential list of operations to be executed.

Operation	Description
	Functional Dataflow
`task`	Own a transparent region, can contain nested dispatch operation with sub-tasks.
`dispatch`	Launch multiple tasks in its region.
	Structural Dataflow
`node`	Own an isolated region, can contain nested schedule operation with sub-nodes. Carry explicit I/O memory effect information.
`schedule`	An isolated region with multiple nodes. Carry explicit scheduling information.
`buffer`	A buffer with variadic stages and ports and automatic ping-pong buffering semantics. Carry explicit partition and layout information.
`stream`	A stream channel with variadic entries.
	Module Interface
`port`	A memory or stream port with explicit type.
`bundle`	A named bundle of ports.
`pack`	Pack an external memory block into a port.

optimized design point. This can eventually lead to non-ideal performance and efficiency, thereby thwarting the promise of existing dataflow approaches. We observed that many HLS-augmentation tools have proposed DSE engines using different algorithms, including polyhedral techniques [52, 53, 1, 49], graph analysis [50, 48, 15, 41], and machine learning [36, 43, 9, 21]. These tools can effectively explore the local design space of a single task or kernel. However, they cannot handle the dataflow-oriented exploration of multiple tasks due to the inter-task coupling and the complicated dataflow hierarchy.

ScaleFlow Approach. With the discussion above, we concluded that the challenges presented in the design and optimization of dataflow architecture *cannot* be fully addressed by existing HLS approaches, which rely on programmers to explore the vast design space manually. We argue that compilers will and should play an important role in the design process - the hierarchical characteristics of dataflow architecture should be systematically represented and modeled, on which an optimization pipeline should be built to handle the inter- and intra-task optimizations comprehensively.

Under this mantra, we propose *ScaleFlow*, an HLS framework with scalable dataflow intermediate representations (IR) and optimizations, enabling the automated transformation of algorithmic hardware descriptions to efficient dataflow architectures. The main contributions of ScaleFlow are as follows:

- We propose a new dataflow IR called ScaleFlow-IR, which models dataflow at two different levels of abstraction, *Functional* and *Structural*, to capture the dataflow characteristics and multi-level hierarchy, enabling effective optimizations.
- We propose a new dataflow optimizer called ScaleFlow-OPT, featuring a pattern-driven task partition algorithm and an intensity- and connection-aware dataflow parallelization algorithm geared toward maximum efficiency.
- We enable an end-to-end and extensible compilation stack supporting PyTorch and C++ inputs, empowering the user to rapidly experiment with various design parameters and prototype new dataflow architectures.
- We perform comprehensive FPGA evaluations of ScaleFlow. On a set of neural networks, ScaleFlow achieves 8.54× and 1.29× higher throughputs over the SOTA HLS optimization framework and RTL-based neural network accelerator.

2. ScaleFlow Framework

Figure 1 shows the overall architecture of the ScaleFlow framework. ScaleFlow is built on top of MLIR [25, 6] and can take deep learning models written in PyTorch [29] or generic HLS C++ code as design entries and produce optimized HLS C++ code. For the PyTorch and C++ inputs, we use Torch-MLIR [7] and Polygeist [28] as front-ends to parse source codes into ScaleFlow, respectively. After the optimizations are completed in ScaleFlow, we use an HLS C++ emitter [40] to generate synthesizable HLS C++ code, which can then be mapped to RTL designs with downstream HLS tools [17, 18, 19]. Inside ScaleFlow, we propose two new techniques to handle the *representation* and *optimization* of dataflow compilation, which are the key enablers to tackle the challenges discussed in Section 1:

- *Hierarchical Dataflow IR (ScaleFlow-IR).* As shown in Figure 1, ScaleFlow consists of two levels of dataflow abstrac-

Table 2: Evaluation results of ScaleFlow for C++ kernels. The *ScaleHLS* designs are automatically generated by [40]. The *Vitis* designs are solely optimized by Vitis HLS.

Kernel	Compile Time (s)	LUT Number	FF Number	DSP Number	Throughput (Samples/s) Improvements		
					ScaleFlow	ScaleFlow v.s. ScaleHLS	ScaleFlow v.s. Vitis
2mm	0.65	38.8k	27.4k	269	239.22	122.39 (1.95×)	1.23 (194.88×)
3mm	0.79	38.7k	27.8k	243	175.43	92.33 (1.90×)	1.04 (167.99×)
atax	2.06	44.6k	34.6k	260	1,021.39	932.26 (1.10×)	103.18 (9.90×)
bicg	0.72	16.0k	15.1k	61	2,869.69	2,869.61 (1.00×)	104.19 (27.54×)
correlation	0.91	14.5k	12.3k	66	67.33	59.77 (1.13×)	1.32 (50.97×)
gesummv	0.60	34.2k	22.8k	232	31,685.68	31,685.68 (1.00×)	266.65 (118.83×)
jacobi-2d	1.98	91.4k	56.6k	352	257.27	128.63 (2.00×)	2.71 (94.95×)
mvt	0.42	23.8k	16.5k	162	9,979.04	4,989.02 (2.00×)	62.13 (160.62×)
seidel-2d	3.59	5.5k	2.5k	4	0.14	0.14 (1.00×)	0.11 (1.28×)
symm	1.05	14.9k	9.5k	74	2.62	2.62 (1.00×)	2.02 (1.29×)
syr2k	0.69	14.3k	12.8k	78	27.68	27.67 (1.00×)	1.44 (19.23×)
Geo. Mean	**0.99**					**1.29**×	**31.08**×

tion carved for different purposes. Table 1 summarizes the key operations of ScaleFlow-IR. The *Functional* dataflow is designed to capture the high-level characteristics and hierarchy of HLS designs, driving the algorithmic optimizations and task partitioning. In contrast, the *Structural* dataflow is a low-level abstraction that captures the micro-architectural details and is optimized to handle the scheduling, parallelization, and code generation.

- *Hierarchical Dataflow Optimizer (ScaleFlow-OPT).* ScaleFlow decouples the optimization problems of *Functional* and *Structural* dataflow to handle HLS designs at scale. At the *Functional* level, ScaleFlow-OPT is focused on effective task partitioning toward workload balancing and low communication cost. At the *Structural* level, ScaleFlow-OPT can optimize the dataflow scheduling through multi-producer elimination and data path balancing. Meanwhile, an intensity- and connection-aware algorithm is introduced to improve the dataflow parallelism while minimizing the resource utilization.

3. Evaluation

To evaluate ScaleFlow, we use FPGAs as target platforms and perform two sets of experiments using C++ and PyTorch inputs and an ablation study on a ResNet-18 model. As depicted in Figure 1, AMD Vitis HLS 2022.1 [17] is used for generating RTL code. All reported performances and resource utilization are collected from the synthesis results of Vitis HLS.

3.1. C++ Kernels Evaluation

Experiment Settings. We evaluate ScaleFlow with a set of C++ benchmarks from PolyBench [30]. The benchmarks cover multiple categories, including blas routines (gesummv, symm, and syr2k), linear algebra kernels (2mm, 3mm, atax, bicg, and mvt), data mining (correlation), and stencils (jacobi-2d and seidel-2d). The targeted platform is AMD-Xilinx ZU3EG FPGA. Table 2 shows the evaluation results. Compared with Vitis HLS, although Vitis HLS can automatically apply optimizations such as loop `pipeline` to a certain degree, it cannot conduct complex dataflow optimizations. As a result, ScaleFlow achieves 31.08× higher throughput.

Comparison with Previous Works. Compared with the State-of-the-Art (SOTA) HLS optimization framework ScaleHLS [40], ScaleFlow achieved 1.29× higher throughput, respectively. We observed that for single-loop kernels (bicg, gesummv, seidel-2d, symm, and syr2k), the performance of ScaleFlow was on par with ScaleHLS due to single-loop kernels not presenting any dataflow optimization opportunities. In contrast, for the other multi-loop kernels, ScaleFlow outperforms ScaleHLS due to dataflow optimizations. When only considering multi-loop kernels, ScaleFlow achieves 1.57× higher throughput than ScaleHLS. We concluded that the dataflow scheduling and parallelization problems are pervasive based on the evaluation results. Thus, ScaleFlow-OPT can better optimize these kernels, ultimately leading to an increased performance.

3.2. PyTorch Models Evaluation

Experiment Settings. We evaluate ScaleFlow with a set of PyTorch deep neural networks (DNN) to understand its performance on large-scale dataflow applications. The benchmarks cover multiple categories of DNNs, including image classification (ResNet-18 [13], MobileNet [14], ZFNet [44], VGG-16 [35]), object detection (YOLO [32]), and fully-connected networks (MLP). The optimization for these models exhibit significant variations under dataflow setting, owing to the distinct layer types and interconnections between layers. The target platform is one super logic region (SLR) of an AMD-Xilinx VU9P FPGA. Table 3 shows the evaluation results.

Comparison with Previous Works. Again, we compare ScaleFlow with ScaleHLS [40], where we observe an 8.54× higher throughput. The throughput gains are much more significant than the C++ kernels due to large DNN models exposing

Table 3: Evaluation results of ScaleFlow for PyTorch neural networks. The *DNNBuilder* results are directly from their paper [46]. To make fair comparison, we constrained the FPGA resources to the same as DNNBuilder. The *ScaleHLS* designs are automatically generated by [40].

Model	Compile Time (s)	DSP Number	Throughput (Samples/s) Improvements			DSP Efficiency Improvements		
			ScaleFlow	ScaleFlow v.s. DNNBuilder	ScaleFlow v.s. ScaleHLS	ScaleFlow	ScaleFlow v.s. DNNBuilder	ScaleFlow v.s. ScaleHLS
ResNet-18	83.1	667	45.4	-	3.3 (13.88×)	73.8%	-	5.2% (14.24×)
MobileNet	110.8	518	137.4	-	15.4 (8.90×)	75.5%	-	9.6% (7.88×)
ZFNet	116.2	639	90.4	112.2 (0.81×)	-	82.8%	79.7% (1.04×)	-
VGG-16	199.9	1118	48.3	27.7 (1.74×)	6.9 (6.99×)	102.1%	96.2% (1.06×)	18.6% (5.49×)
YOLO	188.2	904	33.7	22.1 (1.52×)	-	94.3%	86.0% (1.10×)	-
MLP	40.9	164	938.9	-	152.6 (6.15×)	90.0%	-	17.6% (5.10×)
Geo. Mean	**108.7**			**1.29×**	**8.54×**		**1.07×**	**7.49×**

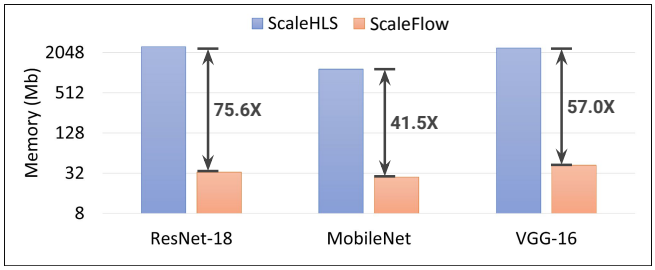

Figure 2: Memory utilization comparison with ScaleHLS [40].

more opportunities for ScaleFlow to optimize the dataflow architecture. For ZFNet and YOLO, ScaleHLS cannot produce results due to the DNNs having irregular convolution sizes and high-resolution inputs, respectively, demonstrating the superior flexibility and scalability of ScaleFlow. For the four benchmarks supported by ScaleHLS, we use DSP efficiency to compare the two frameworks, calculated using:

$$Efficiency_{DSP} = \frac{Throughput \times OPs}{Number_{DSP} \times Frequency}, \quad (1)$$

where *OPs* denotes the total number of multiply and accumulation (MAC) operations per sample of the DNN, *Throughput* is samples per second, and *Frequency* denotes the clock frequency constant at 200MHz for both ScaleHLS and ScaleFlow. DSP efficiency is a common metric for comparing the efficiency of DNN accelerators across different platforms or frameworks. A 100% of DSP efficiency indicates all instantiated DSPs in the accelerator continuously operating without stalling. ScaleFlow achieves 7.49× higher DSP efficiency than ScaleHLS on average and 14.24× for ResNet-18. We attribute the much higher efficiency for ResNet-18 to ScaleFlow's ability to optimize shortcut data paths.

Apart from the throughput and efficiency improvements, we also observe substantial on-chip memory reduction by ScaleFlow compared to ScaleHLS. As Figure 2 shows, ScaleFlow can reduce memory utilization by 41.5× to 75.6× due to several factors: (1) ScaleFlow can leverage loop tiling and local buffer creation to only cache small tiles of intermediate results while enabling the dataflow execution. In comparison, ScaleHLS must keep all intermediate results on-chip due to the lack of external memory access support. (2) The more advanced dataflow parallelization can drastically reduce the buffer sizes. In summary, ScaleFlow can utilize computation and memory resources more efficiently and achieve substantial throughput improvements on DNN models.

Comparison with Dedicated DNN Accelerator. In addition to the comparison with SOTA HLS optimization frameworks, we further compare ScaleFlow with a dedicated DNN acceleration framework, DNNBuilder [46]. DNNBuilder has RTL-based and hand-tuned IPs for accelerating multiple types of layers in modern DNNs and can enable the dataflow execution of all the instantiated IPs to achieve SOTA throughput and efficiency on FPGAs. As shown in Table 3, ScaleFlow achieves 1.29× and 1.07× higher throughput and DSP efficiency compared to DNNBuilder, which already has an extremely high DSP efficiency. Note that ResNet-18 and MobileNet are not supported by DNNBuilder due to its lack of support for shortcut paths and depthwise convolutions. Through this comparison, we demonstrate the productivity and performance of ScaleFlow outperforming a dedicated DNN acceleration framework using human-generated customized RTL IPs. Additionally, we demonstrate the flexibility of ScaleFlow, which can adapt to a wide range of computational patterns.

4. Conclusion

In this paper, we propose ScaleFlow, which is an HLS framework that can systematically transforms the algorithmic description of hardware into efficient dataflow implementations. We propose a two-level dataflow representation, ScaleFlow-IR, and a hierarchical dataflow optimizer, ScaleFlow-OPT, significantly improving the productivity, performance, and scalability of HLS-based dataflow accelerators. To demonstrate the performance of ScaleFlow, we evaluate a set of DNNs and C++ kernels, where ScaleFlow outperforms the existing SOTA hand-tuned RTL-based DNN accelerator and compilation-based HLS frameworks. We plan to open-source the ScaleFlow framework and hope that it will serve as a new open infrastructure for future dataflow architectural research, allowing researchers to explore the vast design space effectively and efficiently.

5. Acknowledgements

This work is supported in part by NSF 2117997 grant through the A3D3 center and Semiconductor Research Corporation (SRC) 2023-CT-3175 grant.

References

[1] Nicolas Bohm Agostini, Serena Curzel, Vinay Amatya, Cheng Tan, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, David Kaeli, and Antonino Tumeo. An mlir-based compiler flow for system-level design and hardware acceleration. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[2] Tal Ben-Nun, Johannes de Fine Licht, Alexandros N Ziogas, Timo Schneider, and Torsten Hoefler. Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2019.

[3] Yuze Chi, Licheng Guo, Jason Lau, Young-kyu Choi, Jie Wang, and Jason Cong. Extending high-level synthesis for task-parallel programs. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 204–213. IEEE, 2021.

[4] Jason Cong, Jason Lau, Gai Liu, Stephen Neuendorffer, Peichen Pan, Kees Vissers, and Zhiru Zhang. Fpga hls today: successes, challenges, and opportunities. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 15(4):1–42, 2022.

[5] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, and Zhiru Zhang. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, 2011.

[6] MLIR Contributors. The mlir project. https://github.com/llvm/llvm-project/tree/main/mlir. Accessed: 2022-04-13.

[7] Torch-MLIR Contributors. The torch-mlir project. https://github.com/llvm/torch-mlir. Accessed: 2022-04-13.

[8] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 92–104, 2015.

[9] Adel Ejjeh, Leon Medvinsky, Aaron Councilman, Hemang Nehra, Suraj Sharma, Vikram Adve, Luigi Nardi, Eriko Nurvitadhi, and Rob A Rutenbar. Hpvm2fpga: Enabling true hardware-agnostic fpga programming. In *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–10. IEEE, 2022.

[10] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *arXiv preprint arXiv:2103.05579*, 2021.

[11] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, et al. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 769–774. IEEE, 2021.

[12] Licheng Guo, Yuze Chi, Jason Lau, Linghao Song, Xingyu Tian, Moazin Khatti, Weikang Qiao, Jie Wang, Ecenur Ustun, Zhenman Fang, et al. Tapa: A scalable task-parallel dataflow programming framework for modern fpgas with co-optimization of hls and physical design. *arXiv preprint arXiv:2209.02663*, 2022.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[15] Sitao Huang, Kun Wu, Hyunmin Jeong, Chengyue Wang, Deming Chen, and Wen-mei Hwu. Pylog: An algorithm-centric python-based fpga programming and synthesis flow. *IEEE Transactions on Computers*, 70(12):2015–2028, 2021.

[16] Yuka Ikarashi, Gilbert Louis Bernstein, Alex Reinking, Hasan Genc, and Jonathan Ragan-Kelley. Exocompilation for productive programming of hardware accelerators. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 703–718, 2022.

[17] Advanced Micro Devices Inc. *Vitis High-Level Synthesis User Guide UG1399 (v2022.2)*. 2022.

[18] Intel Inc. *Intel High Level Synthesis Compiler Pro Edition Reference Manual (22.4)*. 2022.

[19] Microchip Technology Inc. *LegUp 2021.1 Documentation*. 2021.

[20] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[21] HyeGang Jun, Hanchen Ye, Hyunmin Jeong, and Deming Chen. Autoscaledse: A scalable design space exploration engine for high-level synthesis. *ACM Trans. Reconfigurable Technol. Syst.*, 2023.

[22] Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer. *Parallel programming for FPGAs*. 2018.

[23] David Koeplinger, Matthew Feldman, Raghu Prabhakar, Yaqi Zhang, Stefan Hadjis, Ruben Fiszel, Tian Zhao, Luigi Nardi, Ardavan Pedram, Christos Kozyrakis, et al. Spatial: A language and compiler for application accelerators. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 296–311, 2018.

[24] Yi-Hsiang Lai, Yuze Chi, Yuwei Hu, Jie Wang, Cody Hao Yu, Yuan Zhou, Jason Cong, and Zhiru Zhang. Heterocl: A multi-paradigm programming infrastructure for software-defined reconfigurable computing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 242–251, 2019.

[25] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14. IEEE, 2021.

[26] Steven Margerm, Amirali Sharifian, Apala Guha, Arrvindh Shriraman, and Gilles Pokam. Tapas: Generating parallel accelerators from parallel programs. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 245–257. IEEE, 2018.

[27] Thierry Moreau, Tianqi Chen, Ziheng Jiang, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Vta: an open hardware-software stack for deep learning. *arXiv preprint arXiv:1807.04188*, 2018.

[28] William S Moses, Lorenzo Chelini, Ruizhe Zhao, and Oleksandr Zinenko. Polygeist: Raising c to polyhedral mlir. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 45–59. IEEE, 2021.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[30] Louis-Noël Pouchet et al. Polybench: The polyhedral benchmark suite. *URL: http://www. cs. ucla. edu/pouchet/software/polybench*, 437:1–1, 2012.

[31] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. Plasticine: A reconfigurable architecture for parallel paterns. *ACM SIGARCH Computer Architecture News*, 45(2):389–402, 2017.

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[33] Kyle Rupnow, Yun Liang, Yinan Li, and Deming Chen. A study of high-level synthesis: Promises and challenges. In *2011 9th IEEE International Conference on ASIC*, pages 1102–1105. IEEE, 2011.

[34] Benjamin Carrion Schafer and Zi Wang. High-level synthesis design space exploration: Past, present, and future. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2628–2639, 2019.

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

[36] Atefeh Sohrabizadeh, Cody Hao Yu, Min Gao, and Jason Cong. Autodse: Enabling software programmers to design efficient fpga accelerators. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(4):1–27, 2022.

[37] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 65–74, 2017.

[38] Xuechao Wei, Yun Liang, Xiuhong Li, Cody Hao Yu, Peng Zhang, and Jason Cong. Tgpa: Tile-grained pipeline architecture for low latency cnn inference. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. ACM, 2018.

[39] Shaojie Xiang, Yi-Hsiang Lai, Yuan Zhou, Hongzheng Chen, Niansong Zhang, Debjit Pal, and Zhiru Zhang. Heteroflow: An accelerator programming model with decoupled data placement for software-defined fpgas. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 78–88, 2022.

[40] Hanchen Ye, Cong Hao, Jianyi Cheng, Hyunmin Jeong, Jack Huang, Stephen Neuendorffer, and Deming Chen. Scalehls: A new scalable high-level synthesis framework on multi-level intermediate representation. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 741–755. IEEE, 2022.

[41] Hanchen Ye, HyeGang Jun, Hyunmin Jeong, Stephen Neuendorffer, and Deming Chen. Scalehls: a scalable high-level synthesis framework with multi-level transformations and optimizations. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1355–1358, 2022.

[42] Hanchen Ye, Xiaofan Zhang, Zhize Huang, Gengsheng Chen, and Deming Chen. Hybriddnn: A framework for high-performance hybrid dnn accelerator design and implementation. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[43] Mang Yu, Sitao Huang, and Deming Chen. Chimera: A hybrid machine learning-driven multi-objective design space exploration tool for fpga high-level synthesis. In *Intelligent Data Engineering and Automated Learning–IDEAL 2021: 22nd International Conference, IDEAL 2021, Manchester, UK, November 25–27, 2021, Proceedings 22*, pages 524–536. Springer, 2021.

[44] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[45] Bingyi Zhang, Rajgopal Kannan, and Viktor Prasanna. Boostgcn: A framework for optimizing gcn inference on fpga. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 29–39. IEEE, 2021.

[46] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. ACM, 2018.

[47] Xiaofan Zhang, Hanchen Ye, Junsong Wang, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. Dnnexplorer: a framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[48] Jieru Zhao, Liang Feng, Sharad Sinha, Wei Zhang, Yun Liang, and Bingsheng He. Comba: A comprehensive model-based analysis framework for high level synthesis of real applications. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 430–437. IEEE, 2017.

[49] Ruizhe Zhao, Jianyi Cheng, Wayne Luk, and George A Constantinides. Polsca: Polyhedral high-level synthesis with compiler transformations. In *32nd International Conference on Field Programmable Logic and Applications (FPL'22)*, 2022.

[50] Guanwen Zhong, Alok Prakash, Yun Liang, Tulika Mitra, and Smail Niar. Lin-analyzer: A high-level performance analysis tool for fpga-based accelerators. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.

[51] Peipei Zhou, Jiayi Sheng, Cody Hao Yu, Peng Wei, Jie Wang, Di Wu, and Jason Cong. Mocha: Multinode cost optimization in heterogeneous clouds with accelerators. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 273–279, 2021.

[52] Wei Zuo, Warren Kemmerer, Jong Bin Lim, Louis-Noël Pouchet, Andrey Ayupov, Taemin Kim, Kyungtae Han, and Deming Chen. A polyhedral-based systemc modeling and generation framework for effective low-power design space exploration. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 357–364. IEEE, 2015.

[53] Wei Zuo, Louis-Noel Pouchet, Andrey Ayupov, Taemin Kim, Chung-Wei Lin, Shinichi Shiraishi, and Deming Chen. Accurate high-level modeling and automated hardware/software co-design for effective soc design space exploration. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.