Design Space Description Language for Automated and Comprehensive Exploration of Next-Gen Hardware Accelerators

Shail Dave
Arizona State University
USA

Aviral Shrivastava
Arizona State University
USA

ABSTRACT

Exploration of accelerators typically involves an architectural template specified in architecture description language (ADL). It can limit the design space that can be explored, reusability and automation of system stack, explainability, and exploration efficiency. We envision *Design Space Description Language (DSDL)* for *comprehensive, reusable, explainable, and agile DSE*. We describe how its flow graph abstraction enables comprehensive DSE of modular designs, with architectural components organized in various hierarchies and groups. We discuss automation of characterizing, simulating, and programming new architectures. Lastly, we describe how DSDL flow graphs facilitate bottleneck analysis, yielding explainability of costs and selected designs and super-fast exploration.

1 NEED FOR DESIGN SPACE DESCRIPTION

Design space exploration (DSE) of accelerators, especially for machine learning [5, 10], require efficient HW/SW codesigns that meet strict execution constraints [20, 21, 23]. The need for a single accelerator for multiple workloads necessitates bottom-up exploration.

ADL-based design approach: Recent frameworks explore designs of a certain architecture (e.g., systolic arrays, PEs sharing unified buffer that is filled by DMAs) [12, 19, 33, 35]. They describe architectural template in the ADL [4, 15, 17]. So, design process focuses on specific architectural organization (i.e., specific types of computational and memory units interconnected in certain ways and hierarchy), and hardware design space is limited to values of architecture's hyperparameters [16, 37]. Execution costs are provided by either expert-maneuvered analytical models for the architecture [7, 34] or synthesizing each design (which is time-consuming). Space of algorithm-to-accelerator mappings is also formulated based on the template [7, 19]. Thus, DSE frameworks lack following capabilities:

• **Exploring efficient solutions from broad design space:** Since design space gets restricted to the template architecture (e.g., consider one-level, shared buffer as a memory), a vast space of architectures is left unexplored (multi-level buffers, unified buffers, DMA ports instead of buffers), even if some can be more effective.

• **Reusability of design flow for novel, wide range of architectures:** Since design tools are developed for a single template, they can be incompatible with architectures from a broad space, which impacts their reusability. Because, when design space is broadened, such as by integrating new functionality or novel implementation of existing architectural components, previously maneuvered tools—execution cost models, simulators, and algorithm-to-architecture compilers are incompatible with enhanced architectures.

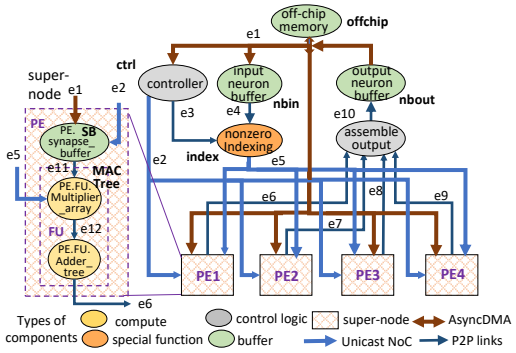


Figure 1: Flow graph of Cambricon-X [36] like accelerator.

• **Explainability of explored designs:** Execution cost models and simulators [7, 8, 14, 19, 32] typically do not provide designers insights about obtained costs. Also, recent approaches for accelerator designs (e.g. [2–4, 11, 16, 17, 22, 25, 26]) use non-feedback or black-box optimization. It makes challenging to reason about the efficacy of the acquisition mechanism of DSE for lowering costs.

• **Quick DSE for dynamic invocations:** In addition to obtaining an efficient solution that meets tight constraints on execution costs, DSE needs to be quick when it is dynamic, e.g., deploying a new DNN on a reconfigurable platform (cloud and end-user). However, due to underlying non-feedback or black-box optimization, DSE requires thousands of trials (or several days) for vast design space.

2 ENVISIONED APPROACH: DSDL

2.1 Specifying Architectural Design Space

Comprehensive DSE needs an abstraction that can allow describing various architectures. So, we envision design space of accelerators as flow graphs, which are specified and explored through Design Space Description Language (DSDL). In the flow graph of an architecture, each node represents an object from primary components for computation, buffer, and control logic [5, 18, 24, 28–30], user-defined special functionality, or even a sub-graph consisting of such components (Fig. 1 illustrates an example). Edges represent interconnects of various bandwidths for a fixed or configurable communication from X source nodes to Y destinations; communication can be concurrent, sequential, or asynchronous. The node corresponding to the beginning and termination of execution (memory, storage, or I/O) is denoted as the root. Such abstraction allows building compute/buffer/communication hierarchy of arbitrary levels, integrating special functionality like for sparsity [5], formulating workgroups for synchronization and load balancing, and specifying

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
LATTE '22, March 1, 2022, Virtual, Earth
© 2022 Copyright held by the owner/author(s).

super-nodes (top-level modules) for better interpretability. Plus, as algorithms are compiled as data flow graphs [1, 4, 6, 13, 30, 31], they can be conveniently mapped on flow graphs of accelerators.

Designers can specify the architectural space in DSDL (Fig. 2a) with: (1) *set of components* for nodes and edges; (2) *parameters*, which are counts of each component and lists of values of hyperparameters of the components; (3) *legality constraints* that specify, when formulating architecture, which kind of component must (not) be connected with which component and how. Designers can also specify constraints between hyperparameters of components (e.g., sizes of the buffers should double for a buffer-hierarchy towards the root); (4) *optimization constraints* for meaningful and optimized exploration, e.g., homogeneous architectures through alike child super-nodes, hierarchy (specific counts and interconnection of units); (5) *IRs or dataflow graphs* of target workloads. Moreover, like conventional DSE with ADL [16, 17, 27], DSDL users should specify (6) *constraints on execution costs*; (7) DSE *objectives*.

Flow graphs can be described with a library and APIs for modular construction of architectures (Fig. 2c). For reusable, automated system stack, architectural components are supplied with their definitions for various tools through a library (Fig. 2b). For each component class, different subroutines define various design flow steps (e.g., analytical cost, simulation functionality, program representation). Then, DSDL can construct these tools for each flow graph, as needed. Developers can extend the library with new components for special-purpose or high-level architectures to enable DSE of new architectures. Following principles drive our approach:

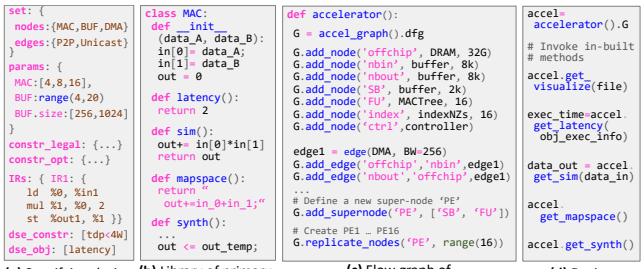
- **Modularity**: An accelerator or a multi-accelerator architecture can be described with pre-defined or user-specified components.
- **Design Flexibility**: Users can specify a comprehensive design space with various hierarchies and grouping of components.
- **Extensibility**: Extending support for new components or new implementations of existing components should be possible.
- **Explainability**: Users can reason about contributions of components to overall execution costs and selection of optimal solutions.
- **Easy-to-Use**: Users can specify, characterize, simulate, program, and explore designs with a few mouse clicks or lines of code (LOC).
- **Compatibility**: Generated outputs (e.g., program representations) should be compatible with relevant tools (LLVM/MLIR).

2.2 Comprehensive DSE with DSDL

DSDL framework enables comprehensive DSE of accelerators by exploring various flow graphs for target functionalities. It invokes DSE with inputs from the designers, and it outputs an HW/SW codesign with minimized objective cost (or a Pareto front) while satisfying all constraints. Besides enforcing specified legality and optimization constraints, it uses *in-built legality constraints*, such as ensuring that a candidate architecture is suitable for target functionality. DSE is dual-mode: *vertical DSE* yields new architectures, and *horizontal DSE* optimizes hyperparameters of each. Explainable DSE with bottleneck analysis can enable smooth joint exploration.

2.3 Full-stack Automation for a Flow Graph

In-built methods for flow graphs (Fig. 2d) can automate visualization, performance/area/power characterization, program representation and mapping space generation, functionality simulation, and



(a) Specifying design space in DSDL **(b)** Library of primary components **(c)** Flow graph of accelerator of Fig. 1 **(d)** Design Automation

Figure 2: (a)-(b) Specifying design space in DSDL. (c)-(d) Flow graph specification of an architecture and its automation.

synthesis of an accelerator design. Some methods may require additional information, such as invocation of components (for analytical characterization), actual data (simulation), and an algorithm to be executed (compilation). DSDL automates a tool's construction by parsing the flow graph and invoking subroutines module-wise.

Automating Execution Cost Modeling. For a flow graph, execution costs can be generated automatically from costs consumed by underlying nodes during execution. A parent's cost is an aggregation of the costs of children. Aggregation depends on cost functions of edges (e.g., maximum/addition of latency, for concurrent/sequential execution over parallel paths; addition for area or power). After propagating costs through parents, the root provides the total cost.

Automating Mapping Space Generation, Code Generation, and Simulation. Each node can be associated with program representation, and edges can be associated with invoking subroutines for communication (e.g., DMA). For instance, a buffer represents memory accesses, and a computational unit represents an arithmetic/logical operation(s) on a data stream (Fig. 2b). Combining such representations meaningfully can realize a collective code representation, transforming to which becomes necessary for executing any algorithm on the accelerator. For instance, consider accelerating matrix multiplication (triply nested loop) on an architecture with PEs sharing a unified buffer where PEs have a local buffer, and DMA fills data in the shared buffer from DRAM. The DSDL-generated mapping space contains all possible transformations (e.g., 12 tiled loops with $(3!)^4$ orderings [7, 19]). Simulation of an accelerator occurs as a dataflow through the flow graph. Control triggers for a looped execution and live-in/out data are communicated through a synthetic controller, which also simulates non-accelerator functionality.

2.4 Explaining Costs by Bottleneck Analysis

Execution cost models get available by processing a flow graph. Such processing can provide information about the costs consumed by different datapaths of a design's graph (e.g., computation time vs. DMA time [5, 9]) and relevant execution characteristics (e.g., total loop iterations invoked, size of transferred data, data reuse). Thus, it facilitates a systematic and accurate bottleneck analysis of the design, improving reasoning about the design's efficacy.

2.5 Dynamic DSE Empowered by Explainability

Explainability informs bottlenecks behind the high costs of a design and how to mitigate them. DSE using bottleneck analysis iteratively explores among effectual candidates and avoids arbitrary trials. Hence, it can be super-fast, especially for optimizing an architecture.

ACKNOWLEDGMENTS

This work was supported in part by NSF under Grant CCF 1723476 — NSF/Intel Joint Research Center for Computer Assisted Programming for Heterogeneous Architectures (CAPA).

REFERENCES

[1] Tal Ben-Nun, Johannes de Fine Licht, Alexandros N Ziogas, Timo Schneider, and Torsten Hoefler. 2019. 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*. 1–14.

[2] Oliver Bringmann, Wolfgang Ecker, Ingo Feldner, Adrian Frischknecht, Christoph Gerum, Timo Hämäläinen, Muhammad Abdullah Hanif, Michael J Klaiber, Daniel Mueller-Gritschneder, Paul Palomero Bernardo, et al. 2021. Automated HW/SW Co-design for Edge AI: State, Challenges and Steps Ahead: Special Session Paper. In *2021 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 11–20.

[3] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. Learning to Optimize Tensor Programs. *Advances in Neural Information Processing Systems* 31 (2018).

[4] S Alexander Chin, Noriaki Sakamoto, Allan Rui, Jim Zhao, Jin Hee Kim, Yuko Hara-Azumi, and Jason Anderson. 2017. CGRA-ME: A unified framework for CGRA modelling and exploration. In *2017 IEEE 28th international conference on application-specific systems, architectures and processors (ASAP)*. IEEE, 184–189.

[5] Shail Dave, Riyadh Baghdadi, Tony Nowatzki, Sasikanth Avancha, Aviral Shrivastava, and Baoxin Li. 2021. Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights. *Proc. IEEE* 109, 10 (2021), 1706–1752.

[6] Shail Dave, Mahesh Balasubramanian, and Aviral Shrivastava. 2018. RAMP: Resource-aware mapping for CGRAs. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[7] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. 2019. Dmazerunner: Executing perfectly nested loops on dataflow accelerators. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–27.

[8] Shail Dave and Aviral Shrivastava. 2018. Ccf: A cgra compilation framework.

[9] Shail Dave, Aviral Shrivastava, Youngbin Kim, Sasikanth Avancha, and Kyoungwoo Lee. 2020. dmazerunner: Optimizing convolutions on dataflow accelerators. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1544–1548.

[10] Jeff Dean, David Patterson, and Cliff Young. 2018. A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro* 38 (2018).

[11] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 622–636.

[12] Yi-Hsiang Lai, Hongbo Rong, Size Zheng, Weihao Zhang, Xiuping Cui, Yunshan Jia, Jie Wang, Brendan Sullivan, Zhiru Zhang, Yun Liang, et al. 2020. Susy: A programming model for productive construction of high-performance systolic arrays on fpgas. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.

[13] Zhaoying Li, Dhananjaya Wijerathne, Xianzhang Chen, Anuj Pathania, and Tulika Mitra. 2021. ChordMap: Automated Mapping of Streaming Applications onto CGRA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021).

[14] Jason Lowe-Power et al. 2020. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).

[15] Prabhat Mishra, Aviral Shrivastava, and Nikil Dutt. 2004. Architecture description language (ADL)-driven software toolkit generation for architectural exploration of programmable SOCs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 11, 3 (2004), 626–658.

[16] Luigi Nardi, David Koeplinger, and Kunle Olukotun. 2019. Practical design space exploration. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE.

[17] Julio Oliveira Filho, Stephan Masekowsky, Thomas Schweizer, and Wolfgang Rosenstiel. 2009. CGADL: An architecture description language for coarse-grained reconfigurable arrays. *IEEE transactions on very large scale integration (VLSI) systems* 17, 9 (2009), 1247–1259.

[18] Subhankar Pal, Siying Feng, Dong-hyeon Park, Sung Kim, Aporva Amarnath, Chi-Sheng Yang, Xin He, Jonathan Beaumont, Kyle May, Yan Xiong, et al. 2020. Transmuter: Bridging the efficiency gap using memory and dataflow reconfiguration. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. 175–190.

[19] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. 2019. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 304–315.

[20] Antonio Pullini, Davide Rossi, Igor Loi, Giuseppe Tagliavini, and Luca Benini. 2019. Mr. Wolf: An energy-precision scalable parallel ultra low power SoC for IoT edge processing. *IEEE Journal of Solid-State Circuits* 54, 7 (2019), 1970–1981.

[21] Alexander Ratner, Dan Alistarh, Gustavo Alonso, David G Andersen, Peter Bailis, Sarah Bird, Nicholas Carlini, Bryan Catanzaro, Jennifer Chayes, Eric Chung, et al. 2019. Mlsys: The new frontier of machine learning systems. *arXiv preprint arXiv:1904.03257* (2019).

[22] Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2017. A case for efficient accelerator design space exploration via bayesian optimization. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE.

[23] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[24] Stephanie Soldavini and Christian Pilato. 2021. Compiler Infrastructure for Specializing Domain-Specific Memory Templates. In *Proceedings of the Workshop on Languages, Tools, and Techniques for Accelerator Design (LATTE'21)*.

[25] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 16–25.

[26] Fengbin Tu, Weiwei Wu, Yang Wang, Hongjiang Chen, Feng Xiong, Man Shi, Ning Li, Jinyi Deng, Tianbao Chen, Leibo Liu, et al. 2020. Evolver: A deep learning processor with on-device quantization–voltage–frequency tuning. *IEEE Journal of Solid-State Circuits* 56, 2 (2020), 658–673.

[27] Stylianos I Venieris and Christos-Savvas Bouganis. 2016. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 40–47.

[28] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, et al. 2017. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 13–26.

[29] Jian Weng, Vidushi Dadu, Sihao Liu, and Tony Nowatzki. 2021. Generality is the Key Dimension in Accelerator Design. In *Proceedings of the Workshop on Languages, Tools, and Techniques for Accelerator Design (LATTE'21)*.

[30] Jian Weng, Sihao Liu, Vidushi Dadu, Zhengrong Wang, Preyas Shah, and Tony Nowatzki. 2020. Dsagen: Synthesizing programmable spatial accelerators. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 268–281.

[31] Lisa Wu, Andrea Lottarini, Timothy K Paine, Martha A Kim, and Kenneth A Ross. 2014. Q100: the architecture and design of a database processing unit. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. 255–268.

[32] Yannan Nellie Wu, Joel S Emer, and Vivienne Sze. 2019. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE.

[33] Pengfei Xu, Xiaofan Zhang, Cong Hao, Yang Zhao, Yongan Zhang, Yue Wang, Chaojian Li, Zetong Guan, Deming Chen, and Yingyan Lin. 2020. AutoDNNchip: An automated dnn chip predictor and builder for both FPGAs and ASICs. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.

[34] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. 2020. Interstellar: Using halide's scheduling language to analyze dnn accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 369–383.

[35] Dan Zhang, Safeen Huda, Ebrahim Songhori, Quoc Le, Anna Goldie, and Azalia Mirhoseini. 2021. A full-stack accelerator search technique for vision applications. *arXiv preprint arXiv:2105.12842* (2021).

[36] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-x: An accelerator for sparse neural networks. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 20.

[37] Yanqi Zhou, Xuanyi Dong, Berkin Akin, Mingxing Tan, Daiyi Peng, Tianjian Meng, Amir Yazdanbakhsh, Da Huang, Ravi Narayanaswami, and James Laudon. 2021. Rethinking co-design of neural architectures and hardware accelerators. *arXiv preprint arXiv:2102.08619* (2021).