Leveraging the MLIR infrastructure for the computing continuum

Jiahong Bi[1,†], Guilherme Korol[1,†] and Jeronimo Castrillon[1,*]

[1] *Technische Universität Dresden (TUD), Helmholtzstraße 18, 01069, Dresden, Germany*

Abstract

With an ever-increasing number of connected devices (e.g., IoT), cloud computing faces efficiency challenges due to complex infrastructure, high communication costs, and privacy. Fog and edge computing enable computing closer to data sources, offering alternatives to the limitations of relying exclusively on the cloud. When combined with high-performance cloud platforms, fog, and edge devices form a computing continuum. However, the continuum challenges designers who need to compile and deploy on distributed and heterogeneous devices and optimize for a diverse set of non-functional requirements. To ease the usage and ensure the full potential of the continuum, a Design and Programming Environment (DPE) that is interoperable, reusable, portable, and cross-layer is needed. In this context, the Multi-Level Intermediate Representation (MLIR) becomes vital since it provides an extensible and reusable compiler infrastructure. The project development of a continuum-oriented DPE leveraging the MLIR infrastructure is discussed in this paper as a work in progress.

Keywords

Computing continuum, Domain Specific Language, Compiler Optimizations

1. Introduction

Cloud computing has emerged as a critical technology in the industry over the past years due to its flexibility in managing information and resources across the Internet. It has also relieved users from the burden of configuring their working environments, allowing them to reduce infrastructural costs. However, in recent years, the rise of Artificial Intelligence (AI) related technologies and the Internet-of-Things (IoT) has made relying solely on cloud-based computing increasingly challenging. This is due to the significant energy consumption and communication costs associated with real-time interactions between the cloud and devices. New computing paradigms, such as fog computing and edge computing, have been introduced as extensions. These approaches aim to address the limitations of cloud-based computing by distributing computational tasks closer to the data source. Cloud, edge, and fog form a *computing continuum* [1], posing new challenges, such as partitioning an application between nodes, compiling applications to these distributed and heterogeneous devices, and seamlessly and efficiently migrating workloads across the continuum.

The `MYRTUS` [2] project aims to address these challenges. More specifically, `MYRTUS` aims to provide the technology to enable cyber-physical systems to evolve towards a living dimension, contributing to integrating edge, fog, and cloud computing platforms into a seamless execution environment and providing languages and tools to orchestrate collaborative, distributed, and decentralized components. One key component of the `MYRTUS` project is a so-called DPE, which deals with multiple aspects of high-level application modeling, model-based design and synthesis, and high-level compilation for adaptable execution on heterogeneous resources. This paper describes initial research and plans for the high-level compilation framework of the DPE known

CPSW'24: CPS Workshop, September 16–20, 2024, Alghero, Italy

[*] Corresponding author.

[†] These authors contributed equally.

jiahong.bi@tu-dresden.de (J. Bi); jeronimo.castrillon@tu-dresden.de (J. Castrillon); guilherme.dos_santos_korol@tu-dresden.de (G. Korol)

0009-0002-5827-1148 (J. Bi); 0000-0002-5007-445X (J. Castrillon)

as Node-Level Optimization and Deployment (NLOP). Notably, the proposed NLOP addresses the compilation in the following aspects:

- Interoperate with model-based frameworks for automatic code generation and deployment to ensure interoperability;
- Integrate with productivity-oriented programming frameworks and Python-based Domain Specific Languages (DSLs) to enhance developer efficiency and ease of use;
- Support different architectures with a focus on accelerators for efficient processing, such as Coarse-Grained Reconfigurable Architectures (CGRAs) and Field-Programmable Gate Arrays (FPGAs);
- Provide automatic insertion of adaptivity knobs for runtime adaptation.

The `MLIR` project will be leveraged to develop the features above in a unified compilation flow. `MLIR` offers us a framework in which we can extend our needs for the continuum, reusing state-of-the-art compilation tools like Low-Level Virtual Machine (LLVM)'s backends and optimizers, supporting hardware heterogeneity, and integrating external tools that will facilitate the construction of the DPE's NLOP.

2. Background

This section presents a brief background on the concepts and tools that will serve to develop the NLOP. This includes, a brief introduction to `MLIR`, initial `MLIR`-based infrastructure developed in a previous EU project, and fundamentals of the adaptable models of computation.

2.1. MLIR

`MLIR` is a promising framework for constructing reusable and extensible compiler infrastructure. It aims to tackle software fragmentation, enhance compilation for diverse hardware systems, considerably lower the expenses associated with developing domain-specific compilers, and facilitate the integration of existing compilers. It extends the monolithic `LLVM IR` into multi-level abstractions, each of which serves its own purpose and has specific functionalities, such as **`arith`** for arithmetic operations and **`linalg`** for linear algebra. The infrastructure of `MLIR` makes it possible to seamlessly transition from abstract, high-level representations to concrete executable code. This makes `MLIR` an enabler for the efficient implementation of DSLs and other programming constructs [3].

In `MLIR`, we can roughly understand that everything is an operation. These operations exist in different *dialects*, each serving its own abstraction level. `MLIR` allows users to define custom dialects, along with custom types, interfaces, etc. One of the most important infrastructure within `MLIR` is `Pass`, with which we can lower or convert one dialect to another by giving several rewrite patterns.

2.2. System Development Kit of `EVEREST`

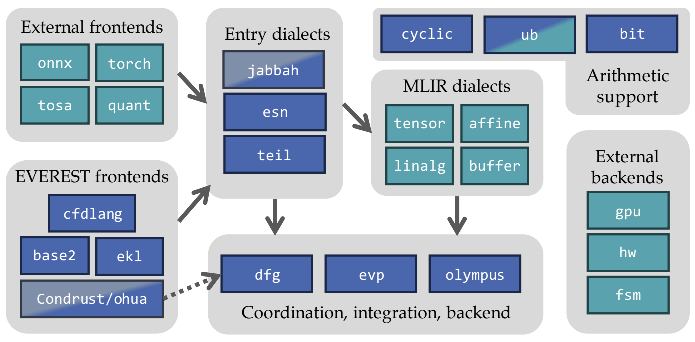

Figure 1: `EVEREST` `MLIR` Stack from [4]

A compelling example of how the `MLIR` multi-level abstractions can be leveraged is demonstrated in the Software Development Kit (SDK) of the `EVEREST` project. `EVEREST` is a H2020 EU project that aims to simplify the development of complex big data applications for FPGA-based data centers [5]. The `EVEREST` SDK is a framework designed to optimize

```
dfg.operator @add
    inputs(%in0: i32, %in1: i32)
    outputs(%sum: i32)
{
    %0 = arith.addi %in0, %in1 : i32
    dfg.output %0 : i32
}
```

(a) OperatorOp

```
dfg.process @add inputs(%in0: i32, %in1: i32)
                 outputs(%sum: i32) {
  dfg.loop inputs(%in0: i32, %in1: i32) {
    %0 = dfg.pull %in0 : i32
    %1 = dfg.pull %in1 : i32
    %2 = arith.addi %0, %1 : i32
    dfg.push(%2) %sum : i32
} }
```

(b) ProcessOp

Figure 2: An example using `dfg-mlir` dialect

selected kernels in the application workflow [4]. Built upon `MLIR`, the SDK supports different input languages into a unified system and hardware generation, connecting to different downstream High-Level Synthesis (HLS) tools. Several dialects, optimizations, and abstraction lowerings are implemented for this workflow.

The main dialects and their relations are shown in Figure 1. Machine Learning (ML) applications from `tvm` can be translated into the `jabbah` dialect. The SDK also includes dialects for kernel language frontend (`ekl`), the coordination dialect `dfg-mlir`, and a DSL `cfdlang`. `ekl` and `cfdlang` can be converted to an `MLIR` implementation of the intermediate tensor language `teil` [6, 7] and a dialect for Einstein notation `esn`. These abstractions are employed to execute a series of transformations. The `EVEREST` `MLIR` stack demonstrates the Multi-Level abstraction methodology to deploy applications *within a cluster* with FPGAs [8]. In `MYRTUS`, we will build on top of these abstractions, extend them, and enable deployment on the computing continuum.

2.2.1. The `dfg-mlir` Dialect

The `dfg-mlir` of the `EVEREST` SDK will be extended to cater for requirements in `MYRTUS`. In the `dfg-mlir` dialect, a user can define an Homogeneous Synchronous Data-Flow (HSDF) node using a custom operation `dfg.operator` (see Figure 2a). Users can define input and output ports and perform any operations with them. The definition of ports is followed by an `MLIR region` with only one block. Users can use any `MLIR` operation inside this region such as `arith.addi` from the `arith` dialect. The returned result is an `MLIR Value` that can be used in other operations as `operands`. An `Output` operation indicates which values should be output. The input/output ports are connected to channels, which are implicitly pulled/pushed from/to at the beginning and end of the region.

For broader modelling of a Data-Flow Graph (DFG), `dfg-mlir` also provides a `Process` operation. This operation has a similar syntax to an `Operator` but is capable of describing a Kahn Process Network (KPN) node, which means that users can pull/push from/to the channels multiple times. There is a `Pass` inside `dfg-mlir`, which can convert every `Operator` to the equivalent `Process` operation, as shown in Figure 2b.

`dfg-mlir` supports different hardware platforms, enabling parallel execution of DFGs. The CPU backend, for instnace, lowers to OpenMP. For hardware generation, `dfg-mlir` can be lowered to `Olympus` [9] which, with the help of the `Bambu` [10] HLS tool, can deploy the graph onto CPU-FPGA heterogeneous system. An extended FPGA backend was introduced in [11], which allows for a more generic execution on FPGAs using the `CIRCT` project as backend.

2.3. Adaptable Models of Computation

`dfg-mlir` demonstrates the usage of the data flow Model of Computation (MoC), which depicts systems as graphs of computational entities and communication channels. MoCs introduce an alternative to traditional programming methods for fully leveraging highly heterogeneous platforms such as the ones in the `MYRTUS` continuum. However, mapping DFGs is a widely studied yet not solved challenge [12]. Traditionally, mappings can be determined at design time using Design and Space Exploration (DSE) or at runtime based on the current workload of the hardware, managed by a Runtime Manager (RM). Currently, `dfg-mlir` relies on the system's RM. To leverage both mapping methods, the Hybrid Application Mapping (HAM) approach is introduced to find near-optimal mappings at design time and adapt to workload changes at runtime [13]. Taking this further, Khasanov et al. recently enhanced HAM by leveraging a genetic algorithm to find spatial-temporal mappings for the MoC [14, 15]. This approach considers expected workload changes and generates more efficient mappings.

For DSE, tools like `Mocasin` can be utilized. `Mocasin` [16] is an open-source research environment to explore mapping algorithms and novel data structures representing the mapping space. `Mocasin` features an abstract modular architecture encompassing commonly used DFG MoCs and the related tool flows, enabling the composition of these flows. There is an integrated high-level simulator that can generate a tracing file, with which users can check the execution of each node in the DFG. `Mocasin` can run DSE to find the Pareto points in the design space based on the DFG and platform. Objectives can be selected from execution time, energy consumption, and resource utilization. Within `Mocasin`, users can freely design their platforms using the provided infrastructure, such as the definition of clusters, Processing Elements (PEs), and Network on Chip (NoC). `Mocasin` also allows users to define their own MoC input for instance, a custom format in YAML.

Dataflow lacks reactive behavior to inputs from the environment which is key in the context of Cyber-Physical System (CPS). Recently, `LinguaFranca` [17] emerged as a coordination language for CPSs, extending dataflow with time semantics using the discrete event model with explicit semantics of time [18]. `LinguaFranca` adopts the reactor model [19] and supports various runtimes capable of concurrent and distributed execution. The reactor model also supports topological changes to the underlying dataflow graph for adaptable execution. In `MYRTUS`, we will borrow ideas from `LinguaFranca` to enable reactive and adaptive execution in the computing continuum.

3. Work-in-Progress

This section gives an overview of the ongoing works and plans for the `MYRTUS` DPE's NLOP . First, a general overview of the NLOP , including its main components, inputs, and outputs, is given. Next, we detail two work fronts currently taking place for extending the `dfg-mlir` dialect for the NLOP.

3.1. General Overview

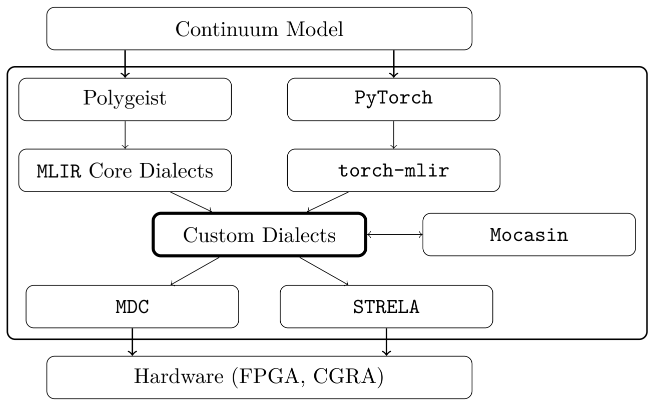

Figure 3: Node-Level Compilation and Deployment Overview

Figure 3 presents an overview of the NLOP covered in this project. The primary input of the NLOP is the continuum model, which encompasses the application code and additional deployment specifications (e.g., targeted kernels and platforms). From the continuum model, the code will be generated and fed into the compiler infrastructure. Finally, the compiler produces low-level code that can be deployed onto various hardware platforms. As illustrated in the figure, the code can originate as general C/C++ code or a deep learning model described in `PyTorch`. Utilizing `Polygeist` [20], C/C++ code can be translated into an `MLIR` program.

While the PyTorch models can be translated into torch-mlir dialect, all inputs (C/C++ or PyTorch) are converted into the MLIR domain. Additionally, NLOP will support techniques such as in [21] in MLIR to further optimize the computation of PyTorch models. These techniques are applied at the PyTorch level to save execution time and energy.

Subsequently, passes will be implemented to translate MLIR programs from the previous step into custom dialects (see Figure 3), including dfg-mlir. This process includes automatic DFG recognition and generation, converting them to the custom dialects while maintaining the same semantics. Once the program in custom dialects is generated, several analyses and optimizations will be performed to obtain the quasi-optimal DFG based on a cost function or similar technology. To that end, Mocasin will be used to run simulations and DSE to identify the best mapping and partitioning for deployment on heterogeneous nodes. Finally, the NLOP generates low-level Intermediate Representation (IR) or code for the supported FPGA platforms with MDC [22] and CGRAs with STRELA [23].

3.2. dfg-mlir with Mocasin

We assume that all the nodes in the generated dfg-mlir program will be an Opeartor, which means from/to each port, we only pull/push one data in each iteration. However, there is a limitation with the Operator operation shown in Figure 2a: it lacks the ability to take values from the previous iteration. With the syntax of Single Static Assignment (SSA), it is illegal to directly use the result value as operand. This limits our ability to translate a wider range of applications into an Opeartor in dfg-mlir (e.g., a Multiply Accumulate (MAC) operation).

```
dfg.operator @mac
        inputs(%in0: i32, %in1: i32)
        outputs(%out: i32)
        iter_args(%sum: i32)
    initialize {
        %0 = arith.constant 0 : i32
        dfg.yield %0 : i32
  } {
    %0 = arith.muli %in0, %in1 : i32
    %1 = arith.addi %0, %sum : i32
    dfg.output %1 : i32
    dfg.yield %1 : i32
}
```

Figure 4: Iteration arguments support

To address this issue, we introduced the iteration arguments syntax to Operator. As shown in Figure 4, an iter_args list can be added after defining the input and output ports. If this list is present, an initialize region must be appended to provide the initial values for each iteration argument. In the body region, these iteration arguments can be used like any other values in any operation. To pass the result of current iteration to the next, a Yield is used to update them at the end of Opeartor.

To integrate with Mocasin we implemented a YAML reader as well as a new CGRA platform. Within LLVM, we developed a transformation pass that outputs the internal dataflow of an Operator to a YAML file. This file contains the information on each node, their ports and the channels connecting them. If there is iteration argument, an initial token will be added in the channel, representing a backedge in the loop graphically.

As mentioned in Section 2, we will expand the semantics of the underlying MoC to account for runtime adaptivity and reactive behavior.

3.3. dfg-mlir atop CIRCT

Currently, CIRCT relies on Polygeist to read in C/C++ programs into MLIR. Each function is then turned into CIRCT's entry dialect, called handshake. Passes are available to lower handshake into low-level dialects within CIRCT, ultimately generating System Verilog code. However, handshake can only describe HSDFs, as it assumes that users will only pull/push one data from/to the ports by default. More expressive computational graphs coming from high-level DSLs, e.g., using Synchronous Data-Flow (SDF) or KPN semantics, cannot use CIRCT as backend at the moment.

As discussed in Section 2, dfg-mlir supports more expressive MoCs (ProcessOp for KPNs). Within dfg-mlir, we have also implemented passes that can directly generate low-level dialects in CIRCT before generating the System Verilog code, such as fsm for Finite State Machine (FSM) generation and sv for SystemVerilog syntax. dfg-mlir also uses *elastic circuit* for each port, the same as handshake, meaning each port will be converted into three signals: valid, data, and ready. For the multiple pulls/pushes behavior in a KPN, we will generate a FSM to correctly handle the elastic circuit signals.

Another consideration is that handshake only inserts a buffer operation, which has a capacity of two elements between two ports for synchronizing different modules. In contrast, we implement a more flexible channel inspired by Chisel [24]. Currently, the channel's capacity is manually controlled. By utilizing the work of Josipović et al. [25], we aim to improve the sizing of channels automatically. With all these features, our approach allows users to have a more powerful way to describe a wider range of DFGs. In the project, this will be extended to support reactive and adaptive execution.

4. Conclusion and Future Work

In this paper, we introduced current efforts to implement the NLOP phase of the MYRTUS DPE. Naturally, some challenges remain to be tackled throughout the NLOP development , including:

- **System integration**: This involves connecting application components, utilizing various abstractions, navigating among transformations and trade-offs in heterogeneous and distributed computing environments, with custom MLIR dialects code generation being the final step.
- **CGRA mapping**: The integration of dfg-mlir and Mocasin for CGRA mapping exploration will not be limited to supporting specific architectures such as STRELA in Figure 3. The approach will support different CGRAs by accepting architecture properties.
- **CIRCT extension**: CIRCT offers an alternative approach to HLS but has some limitations that must be addressed. For instance, the handshake dialect can adopt the dfg-mlir semantics. Another critical extension is the support for pipelining, which is typically available in HLS tools as *pragmas.* To avoid vendor-locking, how to automatically apply different optimization pragmas will also be explored.
- **Adaptive MoC execution**: Applying HAM at the MLIR level and taking LinguaFranca's time semantics could also be explored.

By addressing these points, the DPE's NLOP can fully leverage the resources of the continuum.

Acknowledgments

This work was funded by the EU Horizon Europe Programme under grant agreement No 101135183 (MYRTUS). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, R. Prodan, Cloud, fog, or edge: Where to compute?, IEEE Internet Computing 25 (2021) 30–36.

[2] F. Palumbo, M. K. Zedda, T. Fanni, A. Bagnato, L. Castello, J. Castrillon, R. D. Ponte, Y. Deng, B. Driessen, M. Fadda, et al., Myrtus: Multi-layer 360 dynamic orchestration and interoperable design environment for compute-continuum systems, in: Proceedings of the 21st ACM International Conference on Computing Frontiers Workshops and Special Sessions, 2024, pp. 101–106.

[3] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, O. Zinenko, Mlir: Scaling compiler infrastructure for domain specific computation, in: 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), IEEE, 2021, pp. 2–14.

[4] C. Pilato, S. Banik, J. Beránek, F. Brocheton, J. Castrillon, R. Cevasco, R. Cmar, S. Curzel, F. Ferrandi, K. F. Friebel, et al., A system development kit for big data applications on fpga-based clusters: The everest approach, in: 2024 Design, Automation and Test in Europe Conference and Exhibition (DATE), IEEE, 2024, pp. 1–6.

[5] C. Pilato, S. Bohm, F. Brocheton, J. Castrillon, R. Cevasco, V. Cima, R. Cmar, D. Diamantopoulos, F. Ferrandi, J. Martinovic, G. Palermo, M. Paolino, A. Parodi, L. Pittaluga, D. Raho, F. Regazzoni, K. Slaninova, C. Hagleitner, EVEREST: A design environment for extreme-scale big data analytics on heterogeneous platforms, in: Proceedings of the 2021 Design, Automation and Test in Europe Conference (DATE), DATE'21, 2021, pp. 1320–1325. URL: https://ieeexplore.ieee.org/document/9473940. doi:`10.23919/DATE51398.2021.9473940`.

[6] N. A. Rink, J. Castrillon, TeIL: a type-safe imperative Tensor Intermediate Language, in: Proceedings of the 6th ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY), ARRAY 2019, ACM, New York, NY, USA, 2019, pp. 57–68. URL: http://doi.acm.org/10.1145/3315454.3329959. doi:`10.1145/3315454.3329959`.

[7] A. Susungi, N. A. Rink, A. Cohen, J. Castrillon, C. Tadonki, Meta-programming for cross-domain tensor optimizations, in: Proceedings of 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE'18), GPCE 2018, ACM, New York, NY, USA, 2018, pp. 79–92. URL: http://doi.acm.org/10.1145/3278122.3278131. doi:`10.1145/3278122.3278131`.

[8] S. Soldavini, K. F. A. Friebel, M. Tibaldi, G. Hempel, J. Castrillon, C. Pilato, Automatic creation of high-bandwidth memory architectures from domain-specific languages: The case of computational fluid dynamics, ACM Transactions on Reconfigurable Technology and Systems (TRETS) 16 (2023). URL: https://doi.org/10.1145/3563553. doi:`10.1145/3563553`.

[9] S. Soldavini, C. Pilato, Platform-aware fpga system architecture generation based on mlir, arXiv preprint arXiv:2309.12917 (2023).

[10] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Lattuada, M. Minutoli, C. Pilato, A. Tumeo, Bambu: an open-source research framework for the high-level synthesis of complex applications, in: 2021 58th ACM/IEEE Design Automation Conference (DAC), IEEE, 2021, pp. 1327–1330.

[11] J. Bi, A lowering for high-level data flows to reconfigurable hardware (2024).

[12] J. Castrillon, K. Desnos, A. Goens, C. Menard, Dataflow Models of Computation for Programming Heterogeneous Multicores, Springer Nature Singapore, Singapore, 2023, pp. 1–40. URL: https://doi.org/10.1007/978-981-15-6401-7_45-2. doi:`10.1007/978-981-15-6401-7_45-2`.

[13] R. Khasanov, J. Castrillon, Energy-efficient runtime resource management for adaptable multi-application mapping, in: Proceedings of the 2020 Design, Automation and Test in Europe Conference (DATE), DATE '20, IEEE, 2020, pp. 909–914. URL: https://ieeexplore.ieee.org/document/9116381. doi:`10.23919/DATE48585.2020.9116381`.

[14] R. Khasanov, M. Dietrich, J. Castrillon, Flexible spatio-temporal energy-efficient runtime management, in: 29th Asia and South Pacific Design Automation Conference (ASP-DAC'24), IEEE, 2024, pp. 777–784. URL: https://ieeexplore.ieee.org/document/10473885. doi:`10.1109/ASP-DAC58780.2024.10473885`.

[15] T. Smejkal, R. Khasanov, J. Castrillon, H. Härtig, E-Mapper: Energy-efficient resource allocation for traditional operating systems on heterogeneous processors, 2024. URL: https://arxiv.org/abs/2406.18980. `arXiv:2406.18980`.

[16] C. Menard, A. Goens, G. Hempel, R. Khasanov, J. Robledo, F. Teweleitt, J. Castrillon,

Mocasin—rapid prototyping of rapid prototyping tools: A framework for exploring new approaches in mapping software to heterogeneous multi-cores, in: Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings, 2021, pp. 66–73.

[17] C. Menard, M. Lohstroh, S. Bateni, M. Chorlian, A. Deng, P. Donovan, C. Fournier, S. Lin, F. Suchert, T. Tanneberger, H. Kim, J. Castrillon, E. A. Lee, High-performance deterministic concurrency using lingua franca, ACM Transactions on Architecture and Code Optimization (TACO) 20 (2023) 1–29. URL: https://doi.org/10.1145/3617687. doi:10.1145/3617687.

[18] M. Lohstroh, C. Menard, A. Schulz-Rosengarten, M. Weber, J. Castrillon, E. A. Lee, A language for deterministic coordination across multiple timelines, in: 2020 Forum for Specification and Design Languages (FDL), 2020, pp. 1–8. URL: https://ieeexplore.ieee.org/document/9232939. doi:10.1109/FDL50818.2020.9232939.

[19] M. Lohstroh, Í. Í. Romero, A. Goens, P. Derler, J. Castrillon, E. A. Lee, A. Sangiovanni-Vincentelli, Reactors: A deterministic model for composable reactive systems, in: R. Chamberlain, M. Edin Grimheden, W. Taha (Eds.), Cyber Physical Systems. Model-Based Design – Proceedings of the 9th Workshop on Design, Modeling and Evaluation of Cyber Physical Systems (CyPhy 2019) and the Workshop on Embedded and Cyber-Physical Systems Education (WESE 2019), Springer International Publishing, Cham, 2020, pp. 59–85. URL: https://link.springer.com/chapter/10.1007/978-3-030-41131-2_4. doi:10.1007/978-3-030-41131-2_4.

[20] W. S. Moses, L. Chelini, R. Zhao, O. Zinenko, Polygeist: Raising c to polyhedral mlir, in: 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT), IEEE, 2021, pp. 45–59.

[21] G. Korol, M. G. Jordan, M. B. Rutzig, J. Castrillon, A. C. S. Beck, Pruning and early-exit co-optimization for cnn acceleration on fpgas, in: 2023 Design, Automation and Test in Europe Conference and Exhibition (DATE), 2023, pp. 1–6. doi:10.23919/DATE56975.2023.10137244.

[22] F. Manca, F. Ratto, F. Palumbo, Onnx-to-hardware design flow for adaptive neural-network inference on fpgas, arXiv preprint arXiv:2406.09078 (2024).

[23] D. Vazquez, J. Miranda, A. Rodriguez, A. Otero, P. D. Schiavone, D. Atienza, Strela: Streaming elastic cgra accelerator for embedded systems, arXiv preprint arXiv:2404.12503 (2024).

[24] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, K. Asanović, Chisel: constructing hardware in a scala embedded language, in: Proceedings of the 49th Annual Design Automation Conference, 2012, pp. 1216–1225.

[25] L. Josipović, S. Sheikhha, A. Guerrieri, P. Ienne, J. Cortadella, Buffer placement and sizing for high-performance dataflow circuits, ACM Transactions on Reconfigurable Technology and Systems (TRETS) 15 (2021) 1–32.