

Towards a Lingua Franca for In-Memory Compute Systems at the Edge

Tauseef Ahmed*, Tassilo Tanneberger*, Lars Schütze*, João Paulo C. de Lima*[†], Jeronimo Castrillon*[†]

*Technische Universität Dresden, [†]ScaDS.AI Dresden/Leipzig

Dresden, Germany

{tauseef.ahmed, tassilo.tanneberger, lars.schuetze, joao.lima, jeronimo.castrillon}@tu-dresden.de

Abstract—Cyber-physical systems (CPSs) are becoming increasingly complex, autonomously making decisions that replicate human behavior across a wide range of domains, which makes their design extremely challenging. Lingua Franca (LF) is a reactor-oriented coordination language that addresses this challenge through a deterministic, time-aware concurrency model. While LF supports multiple hardware targets, extending it to emerging accelerators such as near- or in-memory computing (IMC), which reduce data movement relative to conventional von Neumann architectures, offers a promising opportunity for data-intensive execution at the edge. This paper explores the compilation and deployment of deterministic reactive applications onto IMC based accelerators. We propose an MLIR based integration of LF with the CINM compiler for IMC based accelerators, and extend the LF runtime to support offloading computation to these accelerators. We evaluate our approach on a state-of-the-art Advanced Driver Assistance System (ADAS) system adapted for in-memory execution, showing that the deterministic guarantees provided by the LF runtime introduce an overhead of 20% relative to a non-LF, idealized-case implementation that offers no such guarantees. Our end-to-end lowering pipeline enables transparent, deterministic execution of reactive LF applications on IMC-enabled platforms, bridging deterministic coordination languages and emerging memory-centric architectures. These results provide us insights into future work on efficient scheduling techniques and hardware-software co-design.

Index Terms—Cyber-Physical Systems, Coordination Languages, In-Memory Computing, Domain-Specific Languages.

I. INTRODUCTION

AI has been evolving at a rapid pace, enabling a wide range of applications within the CPS domain. CPSs are reactive systems operating at the interface of software and the physical world, increasingly relying on multiple heterogeneous sensors and interoperation with other CPSs to accurately perceive the state of complex, shared environments, such as those found in autonomous driving, industrial automation, and robotics, all while meeting strict timing deadlines. These systems increasingly require processing a huge volume of data, recently reported at around 1 TB per hour [1]. This gives rise to two primary challenges. Firstly, because this data arrives continuously from multiple heterogeneous sources at different sampling intervals, even a small change in the order in which it is processed can have severe consequences for the behavioral correctness of such systems. Prior work [2] identifies such shortcomings even in production-ready Adaptive AUTOSAR systems, reporting dropped frames and input mismatch errors in a brake assistant application, which could cause the system to fail to detect a pedestrian, undermining the very purpose such systems are designed to serve. Secondly, processing such large volumes of data is shifting the performance bottleneck from compute-bound to data-bound, as conventional von Neumann architectures spend an increasing share of time and energy moving data between memory and compute units rather than performing computations on it. LF [3] was designed to address precisely the former challenge by incorporating time as a first-class programming construct. Recent work [4], [5] has demonstrated the effectiveness

of LF in modeling complex applications, including Robot Operating System (ROS)-based systems, while preserving deterministic semantics and simplifying the development of reactive software. Notably, re-implementing the brake assistant application under the reactor model eliminated the observed issues, yielding deterministic, correct execution and demonstrating the potential of LF for such safety-critical applications [2].

The latter challenge has driven interest into new hardware architectures that reduce data movement [6]. IMC is one such promising paradigm, addressing this bottleneck by moving computation to where the data resides, or nearby, for data-intensive tasks, thereby reducing both latency and energy consumption. Existing IMC toolchains primarily address computational kernels, while CPS applications additionally require system-level coordination among kernels, sensors, and control tasks. We present LFCIM, a compiler infrastructure for integrating IMC acceleration into LF applications. LFCIM connects both abstraction levels, bridging deterministic reactive programming with the performance and efficiency gains offered by IMC. Beyond the integration presented in this work, LFCIM provides a basis for exploring heterogeneous scheduling and hardware-software co-design for CPS applications. The main contributions of this paper are as follows:

- We present LFCIM, an MLIR-based compiler that lowers LF programs to the `rflow` dialect and integrated CINM as a backend for executing selected instructions on IMC accelerators.
- We extend the LF execution flow with support for invoking accelerator-generated kernels while preserving the reaction dependencies, coordination, and timing semantics of the LF programming model.
- We demonstrate the approach on an automotive CPS workload and quantify the overhead of the LF-based coordination relative to an optimistic fixed-schedule implementation using the same acceleration kernels.

II. BACKGROUND: IMC FOR CPS

This section provides the background necessary to understand our approach to compiling reactive programs for IMC. It introduces the Multi-Level Intermediate Representation (MLIR) as the compiler infrastructure underlying our work, surveys near- and in-memory processing compilers for data-intensive workloads, and presents LF as the reactive programming model targeted by our compiler.

A. MLIR as a Progressive Lowering Infrastructure

MLIR is a compiler infrastructure for defining and transforming intermediate representations (IRs). It enables representations at different abstraction levels to coexist and interact through *dialects*, making it particularly suitable for implementing domain-specific languages (DSLs). Dialects define a collection of operations, attributes, and types. MLIR provides a set of core dialects that offer reusable

abstractions for common concepts. In particular, the `llvm` dialect represents LLVM IR constructs within MLIR. Programs represented using higher-level dialects can be progressively transformed, i.e., *lowered*, through intermediate dialects to the `llvm` dialect which can subsequently be translated to LLVM IR and processed by LLVM. As a compiler framework, MLIR provides infrastructure for passes, analyses, diagnostics, and IR transformations. An MLIR compiler typically organizes analyses and transformations into a pipeline of passes that progressively optimize or lower the IR. Finally, the compiler either produces a transformed and potentially optimized IR or emits an artifact such as generated C code.

B. Emerging Processing-In-Memory Architectures

Shrinking transistor sizes historically increased compute and memory per unit area, fueling decades of application-space growth. With the end of Dennard scaling and the slowing of Moore’s law [7], this no longer suffices. Compute throughput and memory bandwidth have grown exponentially, while memory latency and energy efficiency have not kept pace [8], making data movement a dominant performance and energy bottleneck.

IMC [6] has emerged as a promising paradigm to address this bottleneck by performing computation close to or within memory, reducing data transfer overhead and improving energy efficiency. Existing in-memory accelerators are predominantly designed for data-parallel operations, such as bulk bitwise computation and matrix-vector multiplication (MVM) vector-matrix multiplication, which are common in neural network inference and data analytics workloads. Analog IMC (AIMC) extends this paradigm further by exploiting device-level physical quantities to perform multiply-accumulate operations directly within the memory array. Leveraging these data-parallel capabilities for computation embedded within CPS applications presents an opportunity to improve both performance and energy efficiency. Several edge architectures have leveraged AIMC to accelerate such workloads [9]–[12]. Among these, ALPINE [12] is one such promising candidate, as its `gem5`-based simulation framework renders it well suited for software-hardware co-design.

Programming in-memory architectures, however, remains challenging due to their specialized, low-level execution models and hardware constraints. To reduce this programmability burden, a compiler must transparently lower applications onto heterogeneous architectures, absorbing hardware complexity and enabling developers to focus on the inherent complexity of modern applications rather than on device-specific programming details [13]–[15]. Cinnamon (CINM) [16] is a compilation infrastructure that addresses this challenge for a wide range of existing in-memory backends, introducing a generic `cinn` abstraction that supports both compute-in-memory (CIM) and compute-near-memory (CNM) paradigms, hierarchically lowered through dedicated `cim` and `cnm` dialects into device-specific backends. CINM has been shown to generate code that matches or outperforms hand-optimized baselines by up to an order of magnitude while reducing the lines of code required by up to $15\times$ on average compared to low-level device programming models. With these abstractions it reduces the programming complexity of current in-memory accelerators and provides an extensible baseline for supporting future ones.

C. The Lingua Franca Coordination Language

LF [3] is a polyglot coordination language realizing the reactor model semantics [17] in which components communicate via timestamped, immutable messages. Application logic is written in supported high-level programming languages. A reactor is composed

of reactions, its atomic units of execution, each triggered by an event such as a timer, input, or action, running to completion at a single logical timestep [18], producing outputs or scheduling future events. Time is treated as a first-class programming construct in LF, allowing programmers to directly control when reactions execute instead of leaving timing behavior to be inferred from the underlying platform. LF imposes a deterministic, time-aware concurrency model that makes the inherent parallelism and data flow of an application explicit, allowing the LF runtime to construct efficient schedules for faster execution, while guaranteeing a consistent view of shared state across all systems running LF. Recent work [4], [5] has demonstrated the effectiveness of LF in modeling complex applications, including ROS-based systems, while preserving deterministic semantics and simplifying the development of reactive software.

LF currently targets a diverse set of hardware platforms, including embedded architectures such as Patmos [19] and Arduino. However, as Moore’s law slows, the long-standing disparity between advances in processing and memory technologies has become increasingly significant, making data movement a primary performance bottleneck. Targeting this increasingly diverse hardware landscape is challenging with the existing LF compiler, since supporting each new custom architecture requires re-implementation of the backend. MLIR was designed to address precisely this problem, using its IR to progressively lower an application to a common abstraction, after which hardware-specific lowering passes can be integrated to target a given backend, avoiding the need to re-implement the entire compiler for each new target. This creates the need to lower LF programs to MLIR, enabling reactive programs to target emerging heterogeneous architectures without requiring a full backend re-implementation for each new platform.

III. RELATED WORK

Numerous compilers and toolchains have been developed to lower applications onto heterogeneous architectures incorporating in-memory computing [20], [21]. These approaches, however, are predominantly designed around a single neural network with a static dataflow schedule, treating the data-bound component as the primary scheduling concern for the application. Complex CPS applications in the automotive and robotics domain commonly involve multiple neural network inferences running concurrently, each with its own timing requirements. Building on this insight, our work considers a complete system comprising multiple neural network components alongside compute-bound components, modeled under the reactor model of computation. Thus, timing constraints can be explicitly captured enabling hardware-software co-design and verification of application deadline requirements, going beyond reducing the latency of the AI components alone. Some works have addressed deadline-aware scheduling for heterogeneous architectures [22], [23]; these approaches, however, primarily consider CPU, GPU, and NPU targets, and do not build on the reactor model of computation with time as an explicit programming construct.

Earlier work on real-time reactor-based applications introduced a precision-timed virtual machine for LF programs [24]. The extension of LF to coordinate reactions across host and FPGA [25] has been addressed by integrating accelerator-specific instructions into the LF program. Targeting multiple platforms requires dedicated implementations for each combination of reaction and target. Our work proposes an annotation based extension to the LF program. Offloading and communication is handled transparently by LFCIM. To the best of our knowledge, this is the first work to consider the


```

1  module {
2      rflow.reactor @Camera ... {
3          %1 = rflow.timer(%c0, %c1) {} : !rflow.timer
4          rflow.reaction @reaction0 { ... } ①
5      }
6      rflow.reactor @CameraEnc ... {
7          rflow.reaction @reaction1 ...
8          attributes {backend = "ALPINE"} { ②
9              %2 = linalg.matmul ...
10             ...
11         }
12     }
13 }

```

Fig. 2. A Sample LFCIM lowering in `rflow` dialect. ① is a reaction executed on host and ② is offloaded to CIM accelerator.

Table I gives an overview of operations and their semantics provided by the `rflow` dialect. The dialect and its operations primarily capture the dependencies among reactions, the connections between reactors, and persistent state, information that is essential for constructing the acyclic dependency graph that models the inter-reaction dependencies required by the code generator. For example, `rflow.reactor` and its inter-reactor connections are mapped onto the `Reactor` struct and the corresponding functions provided by the Lingua Franca C runtime.

Fig. 2 illustrates a sample lowering of part of an ADAS system to the `rflow` dialect, highlighting two reactions, one of which is marked for offloading to ALPINE. This reaction is subsequently passed through the CINM lowering passes to produce an object file, while the LF code generator processes all reactions, both those offloaded to ALPINE and those executed on the host processor, emitting the C code based on the dependency information captured by the dependency graph. The resulting translation units are then linked together into a single executable.

V. ADAS APPLICATION MODELLED IN LF

To showcase the benefit of integrating LF with emerging in-memory architecture based systems, we model a Level 3 ADAS based on the state of the art Bird's-Eye-View (BEV) fusion work [26] in LF. Such systems integrate multiple heterogeneous sensors to build a robust perception of their surrounding environment.

Fig. 3 shows a simplified version of the system modeled in LF. The three `Camera` actors each sample a simulated camera sensor and are triggered every 17 milliseconds. The `Camera Encoders` convert the image view into a BEV. Similarly, the `LiDAR` actors are sampled every 50 milliseconds, and the corresponding encoders generate the associated BEV representation. The `BEVPreProcessor` combines the different BEVs and sends them to the `EgoMotionEstimator`, which corrects the frames using the current position estimates from the `PositionEstimators`. This is then passed to the `BEVProcessor`, which processes and generates the encoded information for Perception, Planning, and Control (PPC) and sends it to the `PPCModule`, which controls the various parameters exposed by the ECU units. The `Radar` pipeline provides similar information at a faster sampling rate to handle emergency braking situations. The encoders for the different sensor actors, as well as the `BEVProcessor`, rely on neural network inference for their operation. We modeled their corresponding reactions as multilayer perceptron (MLP) networks offloaded to the IMC units in ALPINE.

VI. EVALUATION

We evaluate LFCIM on the ADAS application described in the previous section. To quantify the runtime impact by LF, we compare

the LF application against a functionally equivalent application implemented without LF (hereafter, the standalone application). Constructing a perfectly fair comparison is inherently difficult, as any implementation choice may favor one design over the other. We therefore bias the comparison in favor of the standalone application; it is given the same optimistic schedule derived from LF, uses the same optimized kernels generated by the CINM compiler toolchain, and executes via a simple scheduler that follows this fixed, hardcoded schedule. This design gives the standalone application a runtime advantage, since it realizes only one of the many possible execution traces, whereas the LF application continues to handle deadlines and out-of-order inputs at runtime. The reported overhead is thus an upper bound on the cost of the LF runtime.

A. Experimental Setup

Our experiments are conducted on a `gem5`-based full-system simulator built on the ALPINE framework [12], which targets low-power ARM-based systems and tightly integrates AIMC accelerators with the CPU pipeline. The AIMC units are non-volatile, so the network weights are programmed onto the crossbars prior to execution and persist across runs. AIMC tiles are accessed via ISA extensions to ARMv8 that allow the CPU to issue MVM operations directly to the tile. Reactions are bound to specific cores through thread pinning, following the procedure used in ALPINE. Table II summarizes the full system configuration. CPU cycle count and latency are obtained from the statistics reported by the `gem5-X` simulator [27], while energy consumption is estimated using the ALPINE power model.

To provide insights into the mapping and scheduling design space, we evaluate a set of candidate placements of the ADAS application onto two cores, each with a dedicated AIMC unit. The seven main components of the ADAS use case (three camera encoders, LiDAR encoder, radar encoder, BEV preprocessor, and BEV processor) may each be placed on either core. Considering the two cores are homogeneous, so complementary assignments are equivalent and the design space reduces to 64 unique configurations.

To account for measurement bias [28], we take several precautions in our experimental methodology. Full-system simulation in `gem5` is subject to variability arising from OS-level non-determinism, including kernel scheduling decisions, interrupt timing, and background daemon activity, which can affect measurements even for a fixed workload. To control for this, the evaluated applications are restored from an identical checkpoint state prior to execution, ensuring that both applications begin from the same OS, cache, and memory state

TABLE II
SIMULATED SYSTEM CONFIGURATION

Parameter	Configuration
CPU Model	MinorCPU (in-order)
CPU Clock	1 GHz
ISA	ARMv8
Number of Cores	2
L1 I/D-Cache	32 kB
L2 Cache	1 MB, 2-way set-associative
Memory Type	DDR4-2400
Memory Ranks	4
Memory Size	4 GB
System Clock	1600 MHz
AIMC Tile Size	2000 rows × 2000 columns

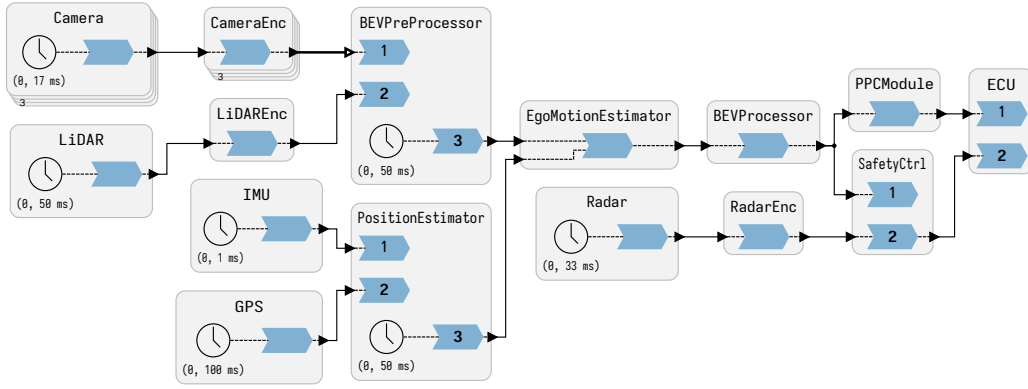


Fig. 3. ADAS level 3 application modelled with Lingua Franca

and eliminating variability due to boot-time and cold-start effects. In addition, each configuration is executed 60 times, and we report the median across runs for CPU cycle count, latency and energy consumption.

B. LFCIM evaluation on ALPINE

Fig. 4 shows the energy and performance overhead of the LF implementation relative to the standalone application across the 64 configurations. The lowest latency is achieved by placing *CameraEnc* for camera 3 on one core and the remaining six reactions on the other, incurring an overhead of 20% over the standalone implementation at the same placement. However, a better energy utilization is obtained by placing *CameraEnc* for camera 1 and 2 on the same core and the remaining five reactions on the other. This motivates a multi-objective formulation, in which mappings must trade off energy and latency while meeting deadline constraints.

The lower overhead of the standalone application does not come without cost. Table III reports how often the BEV preprocessor merges frames that arrive out of order, for each core assignment of the standalone implementation. For readability, only a representative subset of the configurations is shown, with all three camera encoders placed on core 0. Neither implementation employs explicit thread synchronization, but the LF runtime enforces ordering between events by construction, whereas the standalone application relies on the components relative execution times matching those assumed by its fixed schedule. This ordering breaks under small variations in sensor arrival times and in multi-core execution, with up to 22% of runs merging frames from different logical timestamps. While one of the

evaluated assignments is free of out-of-order merges, it is not the one that minimizes energy or latency.

VII. FUTURE WORK AND CONCLUSION

In this paper, we presented LFCIM, a compiler that lowers LF programs onto emerging heterogeneous architectures incorporating IMC units, bridging deterministic reactive programming with the performance and energy benefits of memory-centric hardware. Our evaluation on a ADAS system shows that the deterministic guarantees provided by the LF runtime introduce an overhead of 20% relative to a standalone, non-LF implementation that offers no such guarantees and reflecting an idealized, best-case coordination scenario. This work opens several directions for future research. First, closing the observed performance gap motivates further research into scheduling techniques that leverage the information captured by the reactor programming model to guide offloading and scheduling decisions more efficiently. One direction is to extend the quasi-static scheduling framework of Lin *et al.* [29], whose DAG partitioning stage currently optimizes for workload balance and schedulability, to heterogeneous architectures. Secondly, we see an opportunity for hardware-software co-optimization, exploring lower bounds on hardware resources, such as clock frequency, memory bandwidth, and core count, that are sufficient to meet an application’s timing requirements while minimizing hardware footprint and energy consumption.

Existing benchmark suites for PIM systems predominantly emphasize GEMM-based computations [30], while existing benchmarks for the Reactor framework do not adequately address PIM-based execution [4]. As part of future work, we intend to develop a comprehensive benchmark suite that captures a representative range of reactive applications suitable for such heterogeneous architectures.

TABLE III
FREQUENCY OF OUT-OF-ORDER FRAMES FOR CORE ASSIGNMENTS IN THE STANDALONE IMPLEMENTATION, MEASURED OVER 60 RUNS PER CONFIGURATION. ALL THREE CAMERA ENCODERS ARE PLACED ON CORE 0 IN THE CONFIGURATIONS SHOWN.

Core assignment			Frequency of out-of-order frames (%)
Camera Encoder	LiDAR Encoder	BEV Preprocessor	
0	0	0	3
0	0	1	22
0	1	0	20
0	1	1	0

ACKNOWLEDGMENT

The authors used Claude (Anthropic) for grammar and spelling corrections throughout the manuscript. The tool was not used for any other purpose, and the authors take full responsibility for the final content. This work was supported by the AI competence center ScaDS.AI Dresden/Leipzig (01IS18026A-D), by the European Horizon MSCA Doctoral Network REACT “Self-AwaRe Neuro-morphic ArChiTectures: Security, Reliability and Energy-Efficiency” (101226463), and by the EU Horizon Europe Programme under grant agreement No 101135183 (MYRTUS).

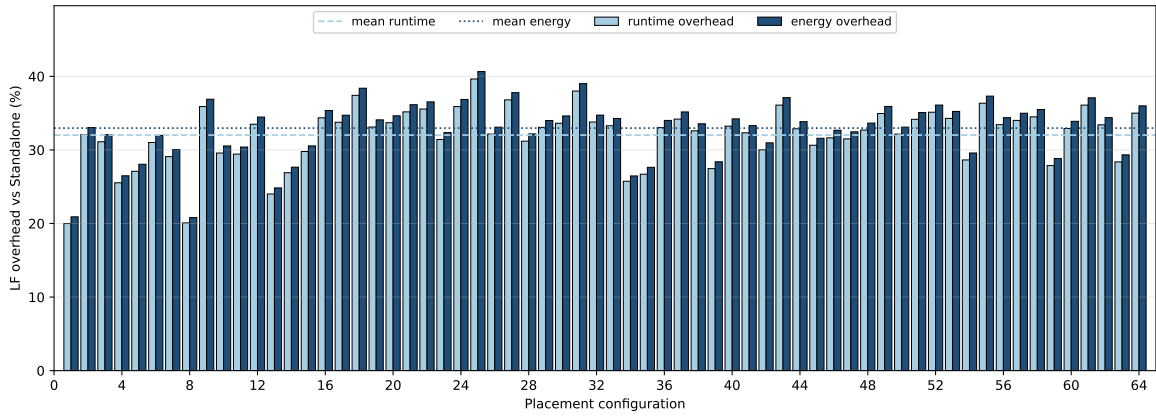


Fig. 4. LF energy and performance overhead relative to the baseline across the 64 unique component-to-core partitions of the ADAS use case. Values represent the median of 60 runs. Latency is reported by the gem5-X simulator and energy is estimated from the ALPINE power model.

REFERENCES

- [1] R. Xu, H. Lin, W. Jeon, H. Feng, Y. Zou, L. Sun, J. Gorman, K. Tolstaya, S. Tang, B. White *et al.*, “Wod-e2e: Waymo open dataset for end-to-end driving in challenging long-tail scenarios,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026, pp. 3709–3718.
- [2] C. Menard, A. Goens, M. Lohstroh, and J. Castrillon, “Achieving determinism in adaptive autosar,” in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 822–827.
- [3] M. Lohstroh, C. Menard, S. Bateni, and E. A. Lee, “Toward a lingua franca for deterministic concurrent systems,” *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 4, May 2021.
- [4] C. Menard, M. Lohstroh, S. Bateni, M. Chorlian, A. Deng, P. Donovan, C. Fournier, S. Lin, F. Suchert, T. Tanneberger, H. Kim, J. Castrillon, and E. A. Lee, “High-performance deterministic concurrency using lingua franca,” *ACM Trans. Archit. Code Optim.*, vol. 20, no. 4, Oct. 2023.
- [5] H. Teper, S. Lin, S. Li, E. A. Lee, and J.-J. Chen, “Deterministic execution of ros 2 applications via lingua franca,” 2026.
- [6] A. A. Khan, J. P. C. D. Lima, H. Farzaneh, and J. Castrillon, “The Landscape of Compute-near-memory and Compute-in-memory: A Research and Commercial Overview,” Jan. 2024.
- [7] H. Esmailzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, “Dark silicon and the end of multicore scaling,” in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 365–376.
- [8] R. Clapp, M. Dimitrov, K. Kumar, V. Viswanathan, and T. Willhalm, “Quantifying the performance impact of memory latency and bandwidth for big data workloads,” in *2015 IEEE International Symposium on Workload Characterization*, 2015, pp. 213–224.
- [9] P. Houshmand, G. M. Sarda, V. Jain, K. Ueyoshi, I. A. Papistas, M. Shi, Q. Zheng, D. Bhattacharjee, A. Mallik, P. Debacker, D. Verkest, and M. Verhelst, “Diana: An end-to-end hybrid digital and analog neural network soc for the edge,” *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 203–215, 2023.
- [10] C. Zhou, F. G. Redondo, J. Büchel, I. Boybat, X. T. Comas, S. R. Nandakumar, S. Das, A. Sebastian, M. Le Gallo, and P. N. Whatmough, “MI-hw co-design of noise-robust tinymml models and always-on analog compute-in-memory edge accelerator,” *IEEE Micro*, vol. 42, no. 6, pp. 76–87, 2022.
- [11] I. Boybat, T. Boesch, M. Allegra, M. Baldo, J. Bertolini-Agnoletto, G. W. Burr, A. Buschini, A. Cabrini, E. Calvetti, C. Cappetta, F. Conti, E. Ferro, E. F. Scarselli, A. Garofalo, F. Girardi, G. Islamoglu, V. P. Jonnalagadda, G. Karunaratne, C. Lammie, M. Le Gallo, C. Li, R. Massa, A. C. Ornstein, H. Pang, M. Pasotti, B. Rajendran, A. Redaelli, I. Sanli, W. A. Simon, A. Singh, S.-P. Singh, G. Urlini, A. Vasilopoulos, R. Zurla, G. Desoli, and A. Sebastian, “Heterogeneous embedded neural processing units utilizing pcm-based analog in-memory computing,” in *2024 IEEE International Electron Devices Meeting (IEDM)*, 2024, pp. 1–4.
- [12] J. Klein, I. Boybat, Y. M. Qureshi, M. Dazzi, A. Levisse, G. Ansaloni, M. Zapater, A. Sebastian, and D. Atienza, “Alpine: Analog in-memory acceleration with tight processor integration for deep learning,” *IEEE Transactions on Computers*, vol. 72, no. 7, pp. 1985–1998, 2023.
- [13] H. Ahmed, P. C. Santos, J. P. Lima, R. F. Moura, M. A. Alves, A. C. Beck, and L. Carro, “A compiler for automatic selection of suitable processing-in-memory instructions,” in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 564–569.
- [14] A. Siemieniuk, L. Chelini, A. A. Khan, J. Castrillon, A. Drebes, H. Corporaal, T. Grosser, and M. Kong, “Occ: An automated end-to-end machine learning optimizing compiler for computing-in-memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 6, pp. 1674–1686, 2021.
- [15] S. Qu, S. Zhao, B. Li, Y. He, X. Cai, L. Zhang, and Y. Wang, “Cim-mlc: A multi-level compilation stack for computing-in-memory accelerators,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 185–200.
- [16] A. A. Khan, H. Farzaneh, K. F. A. Friebe, C. Fournier, L. Chelini, and J. Castrillon, “Cinnm (cinnamon): A compilation infrastructure for heterogeneous compute in-memory and compute near-memory paradigms,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, 2024, pp. 31–46.
- [17] M. Lohstroh, Í. Í. Romeo, A. Goens, P. Derler, J. Castrillon, E. A. Lee, and A. Sangiovanni-Vincentelli, “Reactors: A deterministic model for composable reactive systems,” in *Cyber Physical Systems. Model-Based Design*. Cham: Springer International Publishing, 2020, pp. 59–85.
- [18] M. Lohstroh, E. A. Lee, S. A. Edwards, and D. Broman, “Logical time for reactive software,” in *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*, ser. CPS-IoT Week ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 313–318.
- [19] E. Khodadad, L. Pezzarossa, and M. Schoeberl, “Towards Lingua Franca on the Patmos Processor,” in *2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC)*. Tunis, Tunisia: IEEE, May 2024, pp. 1–5.
- [20] A. Siemieniuk, L. Chelini, A. A. Khan, J. Castrillon, A. Drebes, H. Corporaal, T. Grosser, and M. Kong, “Occ: An automated end-to-end machine learning optimizing compiler for computing-in-memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 6, pp. 1674–1686, 2022.
- [21] Y. Shin, J. Park, S. Cho, and H. Sung, “Pimflow: Compiler and runtime support for cnn models on processing-in-memory dram,” in *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*, ser. CGO ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 249–262.
- [22] Y. Xu, Y. Ni, Y. Ma, and A. Zou, “MiRTS: Scheduling of Mixed Real-Time Tasks on Accelerator-based Heterogeneous Architectures,” *IEEE Transactions on Computers*, no. 01, pp. 1–14, Aug. 5555.
- [23] A. Zou, Y. Xu, Y. Ni, J. Chen, Y. Ma, J. Li, C. Gill, X. Zhang, and Y. Jin,

“A survey of real-time scheduling on accelerator-based heterogeneous architecture for time critical applications,” 2025.

- [24] S. Lin, E. Jellum, M. Theile, T. Tanneberger, B. Sun, C. Jerad, R. Xu, G. Feng, C. Menard, M. Lohstroh, J. Castrillon, S. Seshia, and E. Lee. PretVM: Predictable, Efficient Virtual Machine for Real-Time Concurrency. [Online]. Available: <http://arxiv.org/abs/2406.06253>
- [25] E. R. Jellum, M. Schoeberl, E. A. Lee, and M. Orlandic, “Codesign of Reactor-Oriented Hardware and Software for Cyber-Physical Systems,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 17, no. 4, pp. 1–30, Dec. 2024.
- [26] Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, D. L. Rus, and S. Han, “Bevfusion: Multi-task multi-sensor fusion with unified bird’s-eye view representation,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 2774–2781.
- [27] Y. M. Qureshi, W. A. Simon, M. Zapater, K. Olcoz, and D. Atienza, “Gem5-x: A many-core heterogeneous simulation platform for architectural exploration and optimization,” *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, Jul. 2021.
- [28] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, “Producing wrong data without doing anything obviously wrong!” in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIV. New York, NY, USA: Association for Computing Machinery, 2009, p. 265–276.
- [29] S. Lin, E. Jellum, M. Theile, T. Tanneberger, B. Sun, C. Jerad, Y. Xu, G. Feng, M. Mæhlum, J.-J. Chen *et al.*, “Quasi-static scheduling for deterministic timed concurrent models on multi-core hardware,” *ACM Transactions on Embedded Computing Systems*, vol. 24, no. 5s, pp. 1–25, 2025.
- [30] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, “Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware,” in *2021 12th International Green and Sustainable Computing Conference (IGSC)*, 2021, pp. 1–7.