# Work-in-Progress: Ultra-fast yet Accurate Performance Prediction for Deep Neural Network Accelerators

Konstantin Lübeck, Alexander Louis-Ferdinand Jung, Felix Wedlich, and Oliver Bringmann, *Member, IEEE*

***Abstract*—We present an automatic methodology to accurately predict the performance of Deep Neural Network (DNN) accelerators using abstract descriptions of accelerator architectures and DNNs with a high degree of flexibility. By mapping partially unrolled neural network layers onto accelerator architectures, we automatically construct an analytical performance model, exploiting the dataflow-driven nature of DNNs that allows us to evaluate only a few loop iterations to determine the performance of a whole DNN layer.**

***Index Terms*—Accelerator architectures, deep neural networks, analytical models, prediction, neural network hardware.**

## I. Introduction

In recent years deploying Deep Neural Networks (DNNs) for Internet of Things applications has moved from data centers to edge devices. This shift sparked a new market of hardware vendors providing various parameterizable DNN accelerator architectures. To evaluate the performance of these accelerators, vendors often supply simulators that are several magnitudes slower than actual hardware. This fact makes it extremely challenging to compare different architectures as well as parameter sets for these DNN accelerators.

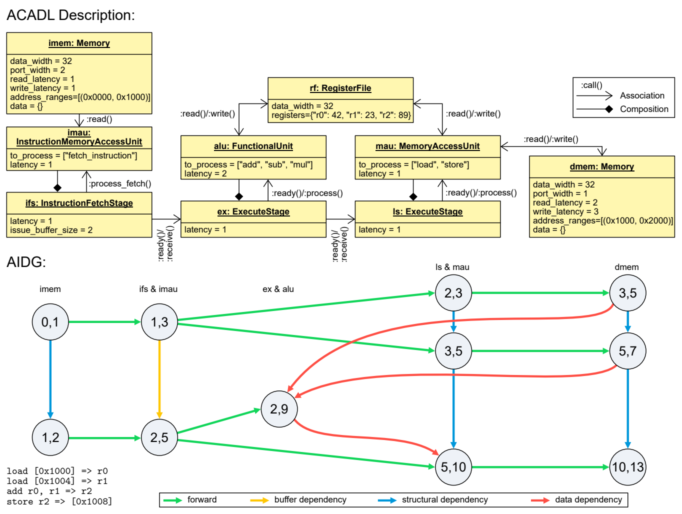
Therefore, we propose the Abstract Computer Architecture Description Language (ACADL) that provides a high degree of flexibility to cover a wide range of parameterizable accelerator architectures on different levels of abstraction and an accurate latency semantic. By mapping partially unrolled DNN layers onto architectures described in ACADL, we generate custom instructions, ranging from scalar to complex tensor operators, that are propagated through an accelerator architecture — tracking the structural and data dependencies of these instructions results in an Architectural Instruction Dependency Graph (AIDG), which serves as an analytical performance model for arbitrary accelerator architectures with complex memory hierarchies, including but not limited to parallel and pipelined accesses as well as local buffers.

## II. Abstract Computer Architecture Description Language

Computer/accelerator architectures are almost exclusively communicated using block diagrams. Each block describes the function of the architecture module it represents, while

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16ES0876 (GENIAL!).

The authors are with the Department of Computer Science, University of Tübingen, 72074 Tübingen, Germany (email: {konstantin.luebeck, a.jung, felix-tobias.wedlich, oliver.bringmann}@uni-tuebingen.de).



```
load [0x1000] => r0
load [0x1004] => r1
add r0, r1 => r2
store r2 => [0x1008]
```

Fig. 1. Example of an architecture described in ACADL and an AIDG that represents the dependencies of custom instructions that are propagated through that architecture.

arrows are used to depict how data is exchanged between different modules. However, modeling computer/accelerator architectures at system level is mainly done using HDLs or ESL languages such as SystemC.

ACADL is an object-oriented language that defines eleven classes that describe the basic building blocks of computer architectures. Those classes are: ACADLObject (base class), Memory, RegisterFile, PipelineStage, ExecuteStage, InstructionFetchStage, FunctionalUnit, MemoryAccessUnit, InstructionMemoryAccessUnit.

## III. Architectural Instruction Dependency Graph

To accurately predict the performance of DNNs mapped onto an accelerator architecture, we propose the Architectural Instruction Dependency Graph (AIDG), capturing when an instruction occupies a hardware module (ACADL object), in which order an instruction propagates through an accelerator architecture, resource conflicts, data dependencies, and buffer fill levels, extending the Execution Graph proposed by Li et al. [1].

An AIDG is a directed acyclic graph. Nodes of an AIDG represent that an instruction occupies an ACADL object. There are four different dependency types an edge can represent: forward, structural dependency, data dependency, and a buffer fill level dependency.

For each node there is a $t_{\text{enter}}$ and $t_{\text{leave}}$ which denote the time in clock cycles when an instruction enters an architecture module (ACADL object) and the time when the instruction leaves the architecture module after all dependencies have been resolved. In Fig. 1 an AIDG for an architecture executing four instructions is depicted while the numbers inside each node denote $t_{\text{enter}}$, $t_{\text{leave}}$.

### A. Construction and Evaluation

To construct an AIDG for a partially unrolled loop kernel of a DNN layer, each instruction of the loop kernel is propagated through the given accelerator architecture in ACADL. For each architecture module an instruction passes through, a node is added to the AIDG. This node is then connected with edges from the four different dependency types to the appropriate nodes that were added before. We repeat this for all instructions in a loop kernel to get the AIDG for one loop kernel iteration.

To determine the end-to-end latency $\Delta t$ of a whole DNN layer with $n$ iterations, the nodes and edges of a single iteration are added $n_{\text{prolog}}$ times to the AIDG until the end-to-end latency $\Delta t_{\text{iteration}}$ and the overlap $\Delta t_{\text{overlap}}$ of two consecutive iterations do not change anymore. To determine $n_{\text{prolog}}$ we use a heuristic.

Afterward we can calculate $\Delta t_{\text{prolog}}$ by resolving all dependencies for each node of the AIDG to determine $t_{\text{enter}}$, $t_{\text{leave}}$ and calculate the end-to-end latency of a whole DNN layer with

$$\Delta t = \Delta t_{\text{prolog}} + (n - n_{\text{prolog}}) \cdot (\Delta t_{\text{iteration}} - \Delta t_{\text{overlap}}).$$

The AIDG construction and evaluation take linear time because each node is only visited once.

## IV. Results

We evaluated the presented methodology by describing a general systolic array, an Eyeriss v1 [2] derived, and a Plasticine [3] derived architecture in ACADL. Onto these, we mapped TC-ResNet8 [4], AlexNet [5], and EfficientNet [6], which cover a wide variety of DNN architectures and layers.

Additionally, we implemented an event-based simulator based on the ACADL latency semantic using the Python package SimPy [7] to validate the predicted cycles from the AIDG evaluation. The predicted cycles for each architecture by the AIDG evaluation matched the simulated cycles exactly. Table I presents how many iterations of a DNN have to be evaluated using an AIDG to get an accurate performance prediction compared to the actual number of iterations.

Fig. 2 shows that the AIDG construction and evaluation time depends linearly on the amount of nodes in an AIDG. This shows that our performance prediction framework is scalable even for large accelerator architectures.

TABLE I
COMPARISON OF THE AMOUNT OF DNN LAYER ITERATIONS ($\sum$ ITERS.) TO THE AIDG EVALUATED ITERATIONS AND THE AIDG EVALUATION RUNTIME $t$ FOR DIFFERENT ACCELERATOR ARCHITECTURES AND DNN MAPPINGS.

| Accelerator | DNN | $\sum$ iters. | eval. | iters. | $t$ [s] |
|---|---|---|---|---|---|
| Systolic Array | TC-ResNet8 | 30 004 | 468 | (1.56 %) | 106 |
| | AlexNet | 30 871 680 | 372 | (0.001 %) | 81 |
| | EfficientNet | 26 074 782 | 4134 | (0.016 %) | 948 |
| Eyeriss v1 derived | TC-ResNet8 | 8045 | 678 | (8.43 %) | 411 |
| | AlexNet | 10 116 488 | 370 | (0.004 %) | 344 |
| | EfficientNet | 14 080 613 | 2674 | (0.019 %) | 1841 |
| Plasticine derived | TC-ResNet8 | 8934 | 568 | (6.36 %) | 1451 |
| | AlexNet | 8 329 852 | 454 | (0.005 %) | 1122 |
| | EfficientNet | 21 526 106 | 3581 | (0.017 %) | 5965 |

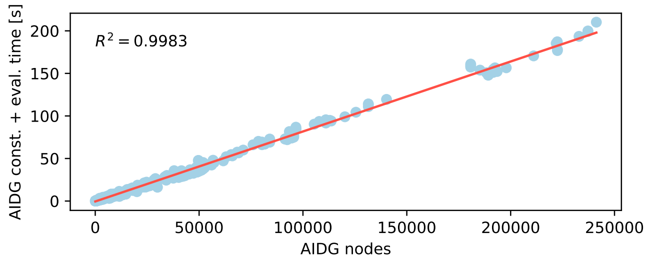


Fig. 2. Linear scaling of the AIDG construction and evaluation time depending on the amount of AIDG nodes.

## V. Conclusion

We presented a fast yet accurate automatic performance prediction framework for DNN accelerator architectures based on the Abstract Computer Architecture Description Language (ACADL). Mapping partially unrolled deep neural network (DNN) layers onto architectures described in ACADL, we construct and evaluate an Architectural Instruction Dependency Graph (AIDG) that allows us to evaluate as few as 0.001% of all loop iterations of a DNN while maintaining very high accuracy.

## References

[1] X. Li, A. Roychoudhury, and T. Mitra, "Modeling out-of-order processors for WCET analysis," *Real-Time Systems*, vol. 34, no. 3, pp. 195–227, Jun. 2006. [Online]. Available: https://doi.org/10.1007/s11241-006-9205-5

[2] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[3] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A Reconfigurable Architecture For Parallel Patterns," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, Jun. 2017.

[4] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha, "Temporal Convolution for Real-time Keyword Spotting on Mobile Devices," vol. abs/1904.03814, 2019. [Online]. Available: http://arxiv.org/abs/1904.03814

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[6] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 6105–6114.

[7] Team SimPy. (2022) SimPy: Discrete event simulation for Python. [Online]. Available: https://simpy.readthedocs.io/en/latest/