Reinforcement learning approach for mapping applications to dataflow-based Coarse-Grained Reconfigurable Array

Abstract

NDC architecture contains a Coarse Grain Reconfigurable Array (CGRA) Streaming Engine (SE). It is a custom hardware architecture that provides programming flexibility that enables high-performance and power savings. A program is represented as a computation graph and its operations need to be properly mapped into the SE device to ensure correct execution and dataflow. This creates an optimization problem with a vast and sparse search space. Creating mappings manually or using brute force algorithms mapping takes time and lots of effort. It also adds assumptions that reduce search-space trading off optimization possibilities. Programing SE manually requires expertise on how all the SE operates, which adds entry barrier to use SE device. In this work we propose a learning method to explore and optimize in an unsupervised manner using reinforcement learning (RL) framework. This provides an automated method that can produce programs quickly and searches for optimal mappings without programmers' interference. This tool also improves the usability of the SE device by encapsulating device configuration details. Proximal Policy Optimization (PPO) training a model to place operations into the SE tiles based on a reward function that models the SE device and its constraints. Graph neural networks are added to create embeddings to represent the computation graph. A transformer block is used to model sequential operation placement mode. The trained model was able to create valid mappings for SE. The implemented method is compared against evolutionary search and baseline.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Keywords: reinforcement learning, data-flow mapping, coarse grain reconfigurable array

ACM Reference Format:
. 2018. Reinforcement learning approach for mapping applications to dataflow-based Coarse-Grained Reconfigurable Array . In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

As Dennard scaling ends, big-data applications such as real-time image processing, graph analytics, and deep learning continue to push the boundaries of performance and energy efficiency requirements for compute system. One solution to this challenge is to move compute closer to memory or storage for substantial energy savings of data movement. We have developed an innovative Compute Near Memory (CNM) architecture that leverages the dramatic opportunities provided by the new CXL protocol. CNM incorporates heterogenous compute elements in the memory/storage subsystem to accelerate various computing tasks near data. One of these compute elements is the Streaming Engine (SE). The SE is a Coarse-Grained Reconfigurable Array (CGRA) of interconnected compute tiles.

The tiles in the CGRA are interconnected with both a synchronous fabric (SF) and an asynchronous fabric (AF) as shown in Fig. 1. SF connects each tile with neighboring tiles that are one or two clocks away. It also interconnects tile memory, multiplexers, and Single Instruction Multiple Data (SIMD) units within each tile. Tiles can be pipelined through SF to form a synchronous data flow (SDF) through multiply/shift or add/logical SIMD units. AF connects a tile with all other tiles, dispatch interface (DI), and memory interfaces (MIs). It bridges SDFs through asynchronous operations, which include SDF initiation, asynchronous data transfer from one SDF to another, system memory accesses, and branching and looping constructs. Together, SF and AF allow the tiles to efficiently execute high-level programming language constructs. Simulation results of hand-crafted SE kernels have shown orders-of-magnitude better performance per watt on data-intensive applications than existing computing platforms.

Mapping the instructions from the computation graph of a program onto the compute elements of the SE while adhering to architectural constraints is an NP-complete problem with a vast and sparse search space. Constraints related to tile memory and synchronous dataflow, use of delays to match timing requirements and more are necessary to ensure correct execution. Creating the mappings manually or using brute force algorithms takes time and lots of effort. This process also adds assumptions that reduce the search space, trading off optimization possibilities.

In this work we propose a Deep Reinforcement Learning (RL) method to explore and find optimal mappings in an unsupervised manner. Using the Proximal Policy Optimization (PPO) method we train a neural network model to place instructions onto the SE tiles guided by a reward function that models the SE device and its constraints. The trained model was able to create valid mappings for the SE by learning about the problem domain. By using a learning methodology, we are able to reuse what the neural network learned to obtain mappings for computation graphs that were not seen during training.

This line of research is inspired by recent work that used RL for chip placement. Our problem requirement of mapping nodes of a computation graph to available compute elements is similar to the problem of placing nodes of a chip netlist on a chip canvas.

2 Method

In the computation graph to be mapped, each node represents an instruction that needs to be placed onto a SE tile. The edges of the graph represent the data dependencies of the instructions. The nodes also contain information about the variables that need to be present in tile memories during instruction execution. All data needed for the instruction must be present at the tile for the instruction to be executed. The tiles can pass data to their neighbors and each tile can be configured with a different number of initiation intervals (II). Each II can be allocated to run a single instruction during program execution. After one cycle, the tiles will move on to the next II, with execution returning to first II after the last. All tiles run the instruction in the same II in parallel. Thus, whenever possible, independent instructions should be mapped to different tiles and same II to exploit parallelism. The first operation of each disconnected component of the computation graph (one component representing a synchronous flow) needs to be placed on different tiles. The SE device configuration and its constraints are modeled in a simulation environment and the reward function. Violating a constraint or placing instructions sub-optimally leads to a reduced reward whereas placements that optimally reduce total execution time of the graph are rewarded.

Reinforcement learning tackles optimization problems by using the REINFORCE algorithm. It trains actor and critic models (represented as a neural networks). The actor model outputs actions (node placements) given an observation of the current state of the SE. The critic model is trained to predict the expected reward given an action in the current state. Proximal Policy Optimization (PPO) is a RL method widely used for continuous and discrete action problems. Our SE mapping task is formulated as a discrete action problem. In PPO, samples of SE state, actions performed, and rewards obtained at each time step are collected after executing the actions provided by the actor model in simulation. These samples are stored in a buffer that is used as data to train the models. In our case, the state is represented by a concatenation of an array of features of placed nodes, a selected node to be placed next and an embedding of the whole computation graph. An action is a tile location and an II count. The reward obtained is based on the number of cycles taken to execute all nodes. The actor model is trained to produce actions from sample states obtained during simulation and the critic model is trained to match the sampled rewards from the actions produced by the actor using a surrogate loss function. This sample and train process is repeated over various iterations. In this manner, the problem search space is explored using the reward function as a heuristic.

In our neural network architecture, a graph neural network (GNN) is used to create embeddings from the computation graph, so that the RL models have information of the data dependencies of different nodes in the graph. An actor model with MLP layers and transformer encoder layers were evaluated in this work. Fig. 2 provides information about how the neural network is trained.

3 Related work

The RL mapper tool lowers SE usability barrier. The user doesn't need to be SE architecture expert, saving training cost. RL tool can also assist other tools or programmers in the program SE mapping creation by providing placement suggestions or tile configuration labels. The RL mapper also provides a faster method of compilation than the brute force algorithm. The RL mapper performs unsupervised learning and optimization allowing it to search in a wider search space. The implemented reinforcement learning approach also shows an advantage compared to baseline random search and evolutionary search (ES) method. The RL approach can learn from a collection of programs and reuse some data for new programs, while other methods will have

to perform a new search for every new program. PPO with graph embeddings showed that can search more optimizations by finding higher rewards than other RL approaches. This technique could be applied to other applications (e.g. chip placement).

In [1], a RL method for chip placement is presented. A graph neural network to create embedding from a netlist graph and passed through an actor model via PPO. The produced output is the whole chip placement. Their actor is composed of deconvolution layers which are more computationally intensive. [2] presents a combination of graph neural network and transformer-XL model to place operations on devices.

In our case, the input is a combination of computation graph, SE device state and node to be placed. Instead of only feeding our actor model with embedding from graph neural network, we combine the graph neural network embedding with information from the SE device and a representation of node that is going to be placed. A configuration is also added to guide the model to optimize different goals.

Another difference from [2] is that our strategy is to place one computation node at a time, instead of generating a whole assignment per iteration. This allows the model to break down the placement problem into sub-problems. This also allows the framework to start from a different start point. For example, if some nodes are already placed by some other algorithm, the RL mapper can place the remaining nodes. This approach also allows us to sample more data during sampling phase. It can save per node placement, instead of only saving a sample for an entire sequence.

4 Results

The implemented RL approach was able to successfully map the computation graphs for vector addition and multiply-add operations. Fig. 3 shows the increase in the value of the rewards obtained as the number of training iterations increase. As the reward increases, we obtain mappings with lesser total execution times.

This method also presents an advantage compared to baselines random search and evolutionary search (ES) methods which do not learn. The RL approach can also learn from a collection of computation graphs and reuse the learning for mapping previously unseen computation graphs. The other methods on the other hand will have to perform a new search and start from scratch for every new computation graph. PPO with graph embeddings showed that we can obtain more optimized placements by finding higher rewards than other RL approaches.

5 Conclusion

Our RL mapper improves and increments the capabilities of the toolset used to program the SE device. It can search for optimal mappings while using the learning to map previously unseen workloads. It improves on the total time required to get a mapping as compared to the existing brute force search approach and allows for automated search of mappings with different optimizations under different trade-offs. It also reduces the manual labor required to find mappings. The techniques presented could be applied to other applications such as chip placement, which is worth trying in future work. Our future work also includes:

- Optimizing training methods to obtain mappings for problems like IFFT which have a bigger search space than currently used computation graphs.
- Increasing sample efficiency of learning methods and improving the simulation environment for the SE by adding more constraints.
- Integration of RL mapper into the SE toolset

Acknowledgments

To Robert, for the bagels and explaining CMYK and color spaces.

References

[1] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. 2020. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746* (2020).

[2] Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C Ma, Qiumin Xu, Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, et al. 2019. Gdp: Generalized device placement for dataflow graphs. *arXiv preprint arXiv:1910.01578* (2019).

A Research Methods

A.1 Part One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi malesuada, quam in pulvinar varius, metus nunc fermentum urna, id sollicitudin purus odio sit amet enim. Aliquam ullamcorper eu ipsum vel mollis. Curabitur quis dictum nisl. Phasellus vel semper risus, et lacinia dolor. Integer ultricies commodo sem nec semper.

A.2 Part Two

Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper. Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.

B Online Resources

Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit congue. Quisque mattis elit a risus ultrices commodo venenatis eget dui. Etiam sagittis eleifend elementum.

Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam. Nunc pulvinar massa et mattis lacinia.