FGYM: Toolkit for Benchmarking FPGA based Reinforcement Learning Algorithms

Nathaniel Peura, Yuan Meng, Sanmukh Kuppannagari, Viktor Prasanna
School of Electrical and Computer Engineering, University of Southern California
Emails: {peura, ymeng643, kuppanna, prasanna}@usc.edu

***Abstract*—FPGA-based heterogeneous computing platforms are promising candidates to enable fast training of Reinforcement Learning (RL) agents. Typically, an RL agent for an environment is trained via interactions with a software that simulates the environment. While several toolkits exist to quickly deploy RL training on CPU or GPU, there lacks a similar toolkit for FPGAs. To ease the deployment process of RL using FPGAs, we demonstrate FGYM (FPGA-GYM) - a toolkit that generates an end-to-end interface between the simulation environments running on the CPU and agents running on the FPGA. FGYM supports a variety of environments and automatically generates the memory interface using PCIe. FGYM supports multiple levels of parallelism including vectorized agent-environment interactions and memory port aggregation. It also provides profiling results for users to identify the execution bottlenecks.**

I. Introduction

Reinforcement Learning (RL) involves the iterative process of an autonomous agent interacting with the environment by sensing the state, s, and choosing actions, a, in a temporal-spatial trajectory to maximize its rewards, r (based on its policy model) [1]. As a common practice in the RL community, OpenAI GYM [2] - a software simulation environment running on a CPU - is used to benchmark various RL algorithms. However, interfacing GYM with an FPGA based RL agent remains a manual and time-consuming process. To address this issue, we develop FGYM that automates the process of deploying RL training on FPGAs, thereby, enabling rapid benchmarking of novel RL algorithms and their FPGA implementations. Benefiting from the complete software development flow enabled by VITIS [3], FGYM targets developers and academic researchers in both FPGA and Deep Learning community.

II. Main Features

FGYM has two phases. In the pre-execution phase, the Host Code Generator takes high-level algorithm and device specifications as user inputs, and generates: (1) host executable, and (2) main memory port specifications that needs to be implemented in the RL FPGA kernel. Several template RL agents will be made available. In the post-execution phase, (3) profiling results are provided to identify execution bottlenecks.

Host Code Generator: It generates the host program and memory configuration file from user-provided high-level specifications such as algorithm hyper-parameters [1] (e.g. Rollout Number (N), Trajectory Length (T)), GYM benchmarking environment name [2] and FPGA device specification

This work is supported by Xilinx and NSF awards 2009057 and 1911229.

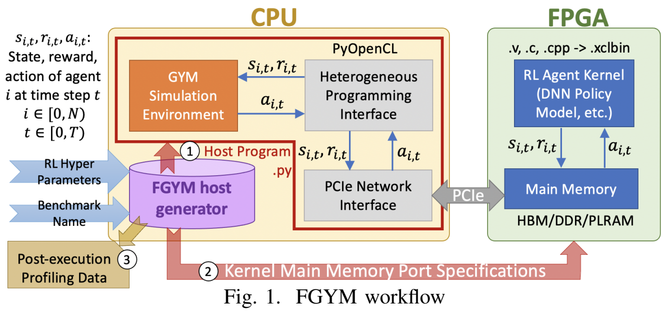

Fig. 1. FGYM workflow

as inputs. The generated program automates data communication between the GYM environment and the kernel bitstream. FGYM also computes the memory requirements for the particular GYM environment and generates a configuration file where the main memory is chosen (on-chip vs DRAM/HBM) to store observations and rewards to minimize communication latency.

Post-Execution Profiler: The generated host program is integrated with code that outputs profiling data after execution. The main profiling data include GYM step latency, kernel computation latency, the overhead and bandwidth utilization of communication through PCIe, and main memory read/write operations. These results can be utilized by RL researchers to fine-tune the performance of parallel Deep RL algorithms on heterogeneous platforms.

III. Demo

We demonstrate FGYM by deploying action evaluations on two environments, one that outputs a scalar value - Cartpole [2], and another that outputs an image - Atari-Pong [2] on a data center CPU-FPGA heterogeneous platform. The action-generation policies of the agents are implemented in pre-loaded FPGA bitstreams. For each environment, we deploy groups of RL agents by varying the algorithmic hyper-parameters. We show the output profiling results of different execution bottleneck scenarios (PCIe, FPGA kernel, etc.).

References

[1] Y. Meng, Y. Yang, S. Kuppannagari, R. Kannan, and V. Prasanna, "How to efficiently train your ai agent? characterizing and evaluating deep reinforcement learning on heterogeneous platforms," in *HPEC*. IEEE, 2020.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[3] V. Kathail, "Xilinx vitis unified software platform," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 173–174.