A Scalable and Reproducible System-on-Chip Simulation for Reinforcement Learning

1st Tegg Taekyong Sung  
EpiSys Science, Inc.  
Poway, CA  
tegg@episysscience.com

2nd Bo Ryu  
EpiSys Science, Inc.  
Poway, CA  
boryu@episysscience.com

Abstract—Deep Reinforcement Learning (DRL) underlies in a simulated environment and optimizes objective goals. By extending the conventional interaction scheme, this paper proffers gym-ds3, a scalable and reproducible open environment tailored for a high-fidelity Domain-Specific System-on-Chip (DSSoC) application. The simulation corroborates to schedule hierarchical jobs onto heterogeneous System-on-Chip (SoC) processors and bridges the system to reinforcement learning research. We systematically analyze the representative SoC simulator and discuss the primary challenging aspects that the system (1) continuously generates indefinite jobs at a rapid injection rate, (2) optimizes complex objectives, and (3) operates in steady-state scheduling. We provide exemplary snippets and demonstrate the run-time performances on different schedulers that successfully mimic results achieved from the standard DS3 framework and real-world embedded systems.

Index Terms—resource allocation, system-on-chip simulation, heterogeneous resource, real-world simulation

I. INTRODUCTION

Deep reinforcement learning (deep RL or DRL) has breakthrough performances in tactical games [24], [27], [34] and robotics [2], [15], [21]. To these successes, a systematized RL-perspective environment is essential to proceed with sequential interactions. The prior works in environment developments for various domains are robotic manipulation [13], [29] and vehicle maneuver [25], [35]. The critical mechanism underlying systems is that the agent must continuously interact with the simulation and receive the necessary information straightforwardly [12].

As a universal problem, resource allocation is associated with various problems, including clustering and wireless communication [20], [22], [32], [33]. Despite the successes in various scheduling applications, previous research has overlooked heterogeneous many-core systems. As a representative, Domain-Specific System-on-chip Simulation (DS3) is a high-fidelity discrete-event simulator targetted to the Domain-Specific System-on-Chip (DSSoC) application and faithfully mimics the real-world hardware performance. [4].

To facile RL agents’ design tailored to real-world systems, this paper introduces a gym-ds3 environment built upon the DS3 framework accessible to the RL research community. We systematically analyze the standard DS3 framework and elaborate fundamental challenging standpoints in designing RL agents to the DS3 framework. Contrast to available scheduling applications, the agent in the DS3 system requires to tackle various joint action sets in complex dynamics due to the hierarchical task dependency and fast job injection rate. Furthermore, we experimentally demonstrate that the scheduling performances operated in the proposed gym-ds3 framework equivalent to the performances in the standard DS3 framework. We publicly release code at https://github.com/EpiSci/gym-ds3.

II. RELATED WORK

The most relevant approach with this paper is the Park platform, which is a unified open Gym framework [12] for ten types of real-world simulators, including cluster scheduling, video streaming, network congestion, memory caching, and circuit design [16]. Park categorizes environments into different challenging problems in systems and provides exemplary RL algorithms.

Concerning scheduling applications, various existing simulations are developed by real-time execution. Formerly, Sparrow builds on a decentralized design that concurrently operates scheduler decisions for cluster jobs [20]. Borg manages large-scale clusters and aims to minimize the fault-recovery time in run-time failures for scheduling decisions [33]. Bose et al. summarize a secure and resilient embedded SoCs applicable in the power-efficient autonomous vehicles domain [10].

As a representative simulator in the SoC application, the DS3 framework positions a de facto benchmarking simulator in active research. DeepSoCS is the first deep RL hybrid scheduler that outperforms run-time performances over standard schedulers provided in the DS3 framework [28]. Krishnakumar et al. apply imitation learning technique to maintain competitive run-time performances and optimize power dissipation and energy efficiency [14]. HiLITE is dynamic power management incorporated into the DS3 framework that utilizes imitation learning to optimize energy efficiency [23].

III. BACKGROUND

The presented gym-ds3 is built on the Gym framework, which is an open interface connecting from the simulation and reinforcement learning algorithms [12]. The system is mainly comprised of reset function to warm up the environment to initialize relevant components and step function to receive an agent action and respond by immediate reward and the next state acquired by the system dynamics. Thereby, the
provided action-and-response methodology is suitable for RL agent workflow.

We begin by elaborating the standard DS3 framework specialized in scheduling hierarchical jobs to heterogeneous resources for an SoC application. The system processes in a real-time operation running by flops (floating-point operations per second) and designed with non-preemptive execution. An overall workflow is illustrated in Figure 1. The simulation comprises a job generator, distributed processing elements (PEs), scheduling policy, and simulation kernel governing task statuses.

A. Jobs and resources

The DS3 framework proceeds the kernel using a job (workload application) and resource (processing element) profiles originally targeted to wireless communication and the radar processing domain. As shown in Figure 2 a job is depicted by a Directed Acyclic Graph (DAG) $G = (V, E)$, where vertices correspond to tasks and edges to communication cost. The node values denote the task number, and the edge values entail data transmission delay accrued to the resource switching. The graph topology represents the dependencies between the tasks. Unlike numerous studies designed for cluster applications that specifying the job duration in the profiles, the SoC workload spanning time is defined by the assigned processing elements’ functionalities from the scheduling policy. The resources or processing elements (PEs)\(^1\) are structured with different functionalities in task run-time and energy consumption. Each PE also contains a list of Operating Performance Points (OPP), which characterize the supported frequency and voltage points. Thereby, the scheduling policy directly reflects the overall performance.

B. Job generator

One key property in DS3 is that the job generator continuously generates an indefinite number of jobs based on the input profile. The job queue holds $C$ jobs, where $C \leq T$. Here, $C$ denotes the number of jobs and $T$ the length of the job queue. A scale value denotes the frequency of job generation. The smaller scale entails high frequency and reflects the faster injection rate.

Once the job is generated, the system identifies job topology and distributes tasks into separate lists of statuses. As shown in Figure 1 the tasks free from dependency load to the ‘ready list’ and those having child dependencies remain in the ‘outstanding list’. As of multiple jobs awaited in the job queue, tasks with different job instances are arbitrarily overlapped. The more overlapping workloads derive the system more dynamic and stochastic.

C. Scheduling policy

The scheduler executes in run-time and essentially allocates each task in the ready list to available PE. The scheduling policy irregularly interacted with the constant clock signal (flops) due to the task dependency and overlapped task traces. After all assignments, the tasks move to the ‘executable list’ and await the PE execution. If the designated PE is idle, the distributed PEs concurrently run the assigned tasks in parallel, and the task transits to the ‘running list’. After execution, the task finally moves to the ‘complete list’.

Assume that the job is composed of $N$ tasks. The individual task run-time is the cumulative sum of PE run-time and data transmission delay, which is calculated with the job edge and PE performance as depicted in Figure 2. The task has waiting time when remaining time in the executable list and response time to the cumulative waiting time. The average response time (ART) denotes the rate between task execution time and task waiting time for all tasks. The scheduler operating in small ART generally indicates adequate performances. The scheduling objective is varied by the designing goals, for

\begin{table}[h]
\begin{center}
\begin{tabular}{ccc}
\hline
Task & $P_0$ & $P_1$ & $P_2$ \\
\hline
$T_0$ & 14 & 16 & 9 \\
$T_1$ & 13 & 19 & 18 \\
$T_2$ & 11 & 13 & 19 \\
$T_3$ & 13 & 8 & 17 \\
$T_4$ & 12 & 13 & 10 \\
$T_5$ & 13 & 16 & 9 \\
$T_6$ & 7 & 15 & 11 \\
$T_7$ & 5 & 11 & 14 \\
$T_8$ & 18 & 12 & 20 \\
$T_9$ & 21 & 7 & 16 \\
\hline
\end{tabular}
\end{center}
\caption{Task execution time & Power consumption}
\end{table}

\begin{table}[h]
\begin{center}
\begin{tabular}{ccc}
\hline
PE & $P_0$ & $P_1$ & $P_2$ \\
\hline
$P_0$ & 100 & 100 & 100 \\
$P_1$ & 100 & 100 & 100 \\
$P_2$ & 100 & 100 & 100 \\
\hline
\end{tabular}
\end{center}
\caption{Communication bandwidth}
\end{table}

\(^1\)Resources represent the raw profile information, and processing elements represent essential operating components execute tasks.
instance, minimizing average latency\(^2\), power dissipation, or energy consumption.

IV. PROPOSED APPROACH

The prototypical DS3 framework is developed by SimPy \[18\] that is a process-based discrete-event simulation framework. SimPy built-in simulation can be operated in real-time, and the simulation kernel with multiple instances can be executed in parallel. This notion, however, is difficult to directly match the interaction time step in the reinforcement learning perspective. We thereby derive a representative environment that supports the standard Gym scheme upon the DS3 framework.

The scheduling agent can be designed as a global control or distributed control, depending on the design perspective. For the latter case, independent PE can be referred to multiple agents collaborating to complete tasks quickly. Moreover, gym-ds3 supports Ray, a distributed operating framework manageable to scalable training \[19\]. For the current version of gym-ds3, the state indicates the simulation information (i.e., task/job/PE statuses, relevant task time), the action space, a joint set on which task allocates to which PEs. The reward function is stated with the average job duration. Users can modify state, action, and reward statements afterward. For the following sections, we pose the challenging critical standpoints for RL agents in DS3 simulation.

A. Challenges

We highlight the main challenging standpoints arising from designing RL agents in the gym-ds3 environment. First, the agent must tackle a varying number of actions. The action of a scheduling agent is a set of tuples composed of two primitive actions, task and PE selections. Systematically, DS3 fetches more than one specifically designed graph-structured job profile. One straightforwardly considers selecting actions for the tasks in the ready list. In that sense, for various task dependencies on different job profiles, the number of remaining tasks is constantly changing. Furthermore, the ready tasks are arbitrarily overlapped by the multiple injecting jobs. Hence, the action associates with a variable joint action problem. Second, the complexity in action spaces exponentially increases to the number of jobs and heterogeneous resources. The disparate functionalities from the distributed PEs and the combination of tasks in the mixed job topology increase complexity in sequential action selection. Third, the indefinite jobs generated at a fast injection rate cause the system dynamics more complex. One of the main differences between SoC and the clustering domain is that the SoC workloads essentially run in a short duration but much faster workload injection. Figure 3 depicts the analysis of job rates for DS3 and Spark simulations \[17\]. Here, we modify simulations with the same range of running flops and the number of injected jobs. We evaluate scheduling performance with heuristic schedulers. Due to innate development, DS3 completes jobs faster than Spark by a factor of 10.

\(^2\)The latency is proportional to the number of completed jobs and inversely proportional to the cumulative execution time.

B. Steady-state scheduling

Absolute makespan minimization of scheduling heterogeneous resource is NP-hard in most practical situations \[5\], \[26\]. The steady-state scheduling circumvents this difficulty by considering asymptotic optimality \[6\], \[7\]. In DS3 simulation, jobs are indefinitely generated, and the scheduling performance is evaluated starting from the steady-state that the jobs are fully stacked to the job queue.

Figure 4 illustrates the timeline that jobs arbitrarily injected into the system based on the scale values. The jobs after the last clock signal are discarded.

DS3 accounts warm-up period to attain a steady-state. The traces before the warm-up period referring to the initializing phase is neglected. The warm-up period essentially denotes additional duration to reach jobs fully stacked to the job queue. Designing a warm-up period requires domain-expert knowledge.
Essentially, the better scheduling policy takes a more extended warm-up period.

In practice, reaching to initializing phase wastes time to run an episode, particularly in RL training. Sung et al. introduce ‘pseudo-steady-state' (PSS) that approximates steady-state to reduce the waiting time. As denoted in Figure 5 in PSS mode, the episode starts running from the complete jobs stacked in the job queue. Assume that the system holds up to \( N \) jobs, then the PSS starts simulation after generating \( N \) jobs in the job queue. In this paper, we evaluate scheduling algorithms with the PSS option.

C. Implementation

This section derives the code snippets of the proposed environment framework. First, we provide the GymDS3 class built upon the Gym framework [12].

```python
class GymDS3(gym.Env):
    def __init__(self):
        ...

    def reset(self):
        # reset environment

    def step(self, action):
        if ready task list is not empty:
            # select action
        else:
            # select no-action and run simulator
            # update immediate reward

        obs = self._get_observation()
        return obs, reward, done, {}  # step function

    def _get_observation(self):
        ...  # get observation function

        return (job_dags, action_map, env_storage, PE)  # return (job_dags, action_map, env_storage, PE)
```

In the `step` function, the action can be valid action sets (i.e., task-to-PE mappings) or ‘no-action’ in that the task not in the ready list. For the latter case, the immediate reward is updated. The `_get_observation` function returns environment instances. `job_dags` denotes the injected job data, `action_map` the take-to-PE mappings, `env_storage` the task statuses and performance statistics, `PE` the PE information. Next, we demonstrate the actual simulation processing code snippets.

```python
# Create a new environment and reset it.
env = GymDS3(simulation_length, scale)
state = env.reset()

# Create a scheduler
scheduler = get_scheduler(env, scheduler_name)

# If using ‘DeepSoCS' as scheduler
if scheduler_name == 'DeepSoCS':
    sess = tf.Session()
    actor_agent = ActorAgent(sess, **kwargs)
    scheduler.set_actor_agent(actor_agent)

    done = False

while not done:
    action = scheduler.schedule(state)
    state, reward, done, info = env.step(action)
```

The environment is generated with total simulation length and scale value. In particularly using neural scheduler (i.e., DeepSoCS [28]), TensorFlow session [1] and corresponding agent are initialized.

V. Experiments

This section demonstrates the feasibility of gym-ds3 to standard DS3 framework by conducting experiments and two orthogonal directions. (a) We compare average response time for gym-ds3 and DS3 in different job frequencies and provided schedulers. (b) We verify the latency performances in different schedulers with gym-ds3 and DS3 simulations.

A. Experimental Setup

Throughout the experiments, we use five different types of jobs that modified topology based on the Simple profile from Figure 2. The jobs are continuously generated to the exponential distribution controlled by the scale factor. To diminish time for initializing phase, we start evaluation on pseudo-steady-state condition. We set the job queue length to three and simulate for 5,000 clock time (flops). We conduct five trials using different random seeds to produce result data.

The gym-ds3 environment provides heuristic schedulers: Shortest Job First (SJF) [31], Minimum Execution Time (MET) [11], Earliest Task First (ETF) [9], and Heterogeneous Early Finish Time (HEFT) [5], [8], [30]. Also, we include DeepSoCS [28], which is a hybrid scheduler on Deep RL and heuristic approaches.

B. Average response time

We extrapolate average response time (ART) using the gym-ds3 environment and standard DS3 framework with different job generation frequencies and schedulers. Figure 6 highlights the task waiting time and task running time using MET scheduler. Considering that the DS3 successfully mimics customer hardware performance, we discover that gym-ds3 simulation demonstrates almost equivalent performance to the DS3 framework. The minor differences can be neglected due to the simulation variances. Next, we evaluate ART on different schedulers and job frequencies using gym-ds3 environment, as illustrated in Figure 7. The variances are marked with the
validate run-time performances using various schedulers and job frequencies in gym-ds3 and DS3 and extrapolate almost identical performances.

ACKNOWLEDGMENT

The authors would like to thank Hanbum Ko for experiment setup and Jeewoo Kim for valuable discussions.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In 12th {USENIX} symposium on operating systems design and implementation ( OSDI ’16), pages 265–283, 2016.

[2] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. The International Journal of Robotics Research, 39(1):3–20, 2020.

[3] Hamid Arabnejad and Jorge G Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Transactions on Parallel and Distributed Systems, 25(3):682–694, 2013.

[4] Samet E. Arda, Anish NK, A. Alper Goksoy, Nirnal Kumbhare, Joshua Mack, Anderson L. Sartor, Ali Akgolu, Radu Marculescu, and Umit Y. Ogras. Ds3: A level-domain specific system-on-chip simulation framework, 2020.

[5] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. Complexity and approximation: Combinatorial optimization problems and their approximability properties. Springer Science & Business Media, 2012.

[6] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Steady-state scheduling on heterogeneous clusters. International Journal of Foundations of Computer Science, 16(02):163–194, 2005.

[7] Dimitris Bertsimas and David Gamarnik. Asymptotically optimal algorithms for job shop scheduling and packet routing. Journal of Algorithms, 33(2):296–318, 1999.

[8] Luiz F Bittencourt, Rizos Sakellariou, and Edmund RM Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, pages 27–34. IEEE, 2010.

[9] James Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task scheduling strategies for workflow-based applications in grids. In CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005., volume 2, pages 759–767. IEEE, 2005.

[10] Pradip Bose, Augusto Vega, Sarita Adve, Vikram Adve, Sasa Misailovic, Luca Carloni, Ken Shepard, David Brooks, Vijay Janapa Reddi, and Gu-Yeon Wei. Secure and resilient socs for autonomous vehicles.
[11] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks to heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 61(6):810–837, 2001.

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[13] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

[14] Anish Krishnakumar, Samet E Arda, A Alper Goksoy, Sumit K Mandal, Umit Y Ogras, Anderson L Sartor, and Radu Marculescu. Runtime task scheduling using imitation learning for heterogeneous many-core systems. arXiv preprint arXiv:2007.09361, 2020.

[15] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research, 17(1):1334–1373, 2016.

[16] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Yang, Ryan Marcus, Ravichandra Addanki, Mehrdad Khani Shirkoohi, Songtao He, et al. Park: An open platform for learning-augmented computer systems. Advances in Neural Information Processing Systems 32 (NIPS 2019), 2019.

[17] Hongzi Mao, Malte Schwarzkopf, Shailesh Boja Venkatkrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In Proceedings of the ACM Special Interest Group on Data Communication, pages 270–288, 2019.

[18] Norm Matloff. Introduction to discrete-event simulation and the simpy.

[19] Alejandro Rico, Felipe Cabarcas, Antonio Quesada, Milan Pavlovic, Tegg Taekyong Sung, Jeongsoo Ha, Jeewoo Kim, Alex Yahja, Chae-Bong Sohn, and Bo Ryu. Deepsocs: A neural scheduler for heterogeneous system-on-chip (soc) resource scheduling. Electronics, 9(6):936, 2020.

[20] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 5026–5033. IEEE, 2012.

[21] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In Proceedings of the Tenth European Conference on Computer Systems, pages 1–17, 2015.

[22] Abhishek Verma, Luis Pedrosa, Madhukar Koropu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In Proceedings of the Tenth European Conference on Computer Systems, pages 1–17, 2015.

[23] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Mihaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Perko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature, 575(7782):350–354, 2019.

[24] Iker Zamora, Nestor Gonzalez Lopez, Victor Mayoral Vilches, and Alejandro Hernandez Cordero. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. arXiv preprint arXiv:1608.05742, 2016.