Received 22 July 2024, accepted 6 August 2024, date of publication 9 August 2024, date of current version 2 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3441242

RESEARCH ARTICLE

Massively High-Throughput Reinforcement Learning for Classic Control on GPUs

XUAN SHA[1,2] AND TIAN LAN[3]
[1]School of Civil and Transportation Engineering, Southeast University Chengxian College, Nanjing, Jiangsu 210088, China
[2]College of Mechanics and Engineering Science, Hohai University, Nanjing, Jiangsu 211100, China
[3]Salesforce AI Research, Palo Alto, CA 94301, USA

Corresponding authors: Xuan Sha (x.sha@foxmail.com) and Tian Lan (tian.lan@salesforce.com)

This work was supported in part by the Research and Development Fund for Young Teachers of Southeast University Chengxian College under Grant Z0057, and in part by Jiangsu Provincial University Philosophy and Social Science Research Project under Grant 2022SJYB0707.

ABSTRACT This study presents a novel massively high-throughput reinforcement learning (RL) framework specifically designed for addressing classic control problems, leveraging our proposed architecture and algorithms optimized for efficient concurrent computations on GPUs. Our research demonstrates the effectiveness of our methods in efficiently training RL agents across various classic control problems, encompassing both discrete and continuous domains, while achieving rapid and stable performance up to 10K concurrent environment instances. Furthermore, we observe that RL exploration with a large number of parallel instances significantly enhances the stability of updating a shared model. For instance, we show that the stability of Deep Deterministic Policy Gradient (DDPG) training can be achieved without requiring experience replay, as evidenced in our study.

INDEX TERMS Classic control, GPU acceleration, high-throughput, reinforcement learning.

I. INTRODUCTION

In the vast landscape of artificial intelligence (AI), reinforcement learning (RL) stands as a fundamental approach to learning and decision-making in dynamic environments [1]. Rooted in the principles of trial-and-error learning and operant conditioning, RL algorithms enable agents to adapt and improve their decision-making strategies over time. By formulating learning as a sequential decision-making process, reinforcement learning agents learn from experience and adapt their behavior to achieve desired outcomes. Through techniques such as value iteration, policy improvement, and deep neural networks, RL algorithms can tackle complex problems spanning from gaming, robotics, and unmanned aerial vehicle to economics and scientific research [2], [3], [4], [5], [6], [7], [8], [9], [10].

Recent advancements in RL have brought about a resurgence of interest in the intersection between AI and classical mechanics. Researchers have begun exploring how reinforcement learning algorithms can be applied to solve problems in classical mechanics, such as optimal control and trajectory planning [11], [12], [13], [14]. Classical mechanics, classic control, and RL represent interconnected pillars in the fields of control theory and engineering. Classical mechanics provides the foundational principles governing the behavior of physical systems, including principles of motion dynamics, force interactions, and energy conservation. Classic control leverages these principles to design deterministic controllers that regulate the behavior of dynamic systems to achieve desired objectives. Meanwhile, RL offers a data-driven approach to control, enabling systems to learn optimal control policies directly from experience through interaction with their environments. The relationship between these disciplines is symbiotic: classical mechanics provides the theoretical framework, classic control offers rule-based strategies, and RL introduces adaptive and intelligent techniques for control. Integrating RL with classic control principles allows for the development of more robust, adaptive, and intelligent control systems capable of handling complex and nonlinear dynamics across a wide range of applications, from robotics and autonomous vehicles to industrial automation and beyond.

However, RL encounters challenges when applied to classic control problems, including learning instability, sample efficiency issues, difficulty in generalization and exploration-exploitation trade-offs. For instance, in classic

The associate editor coordinating the review of this manuscript and approving it for publication was Jianxiang Xi.

control problems, RL agents may struggle to converge to optimal policies due to the high dimensionality and continuity of action spaces, as well as the sparse nature of rewards [15], [16]. Moreover, the strong correlations often present in trajectory sequences in classical mechanics, can hinder effective policy exploration in RL [15]. Additionally, RL algorithms typically demand extensive training data to learn proficient policies, posing a significant challenge in classical mechanics where data collection can be costly or time-consuming. Enhancing sample efficiency is essential for RL algorithms to effectively learn from limited interactions with the environment.

Overcoming these challenges requires the development of tailored algorithms and techniques that address the specific characteristics of classic control domains, such as improved exploration strategies, enhanced sample efficiency, and robust generalization capabilities. By addressing these challenges, we can tackle complex control and optimization problems in classic control more effectively.

In this paper, we utilize a computational framework specifically designed to achieve massively high-throughput RL simulation, built upon the foundation of WarpDrive [17], which is accessible at https://github.com/salesforce/warp-drive. The high-throughput capability enables algorithmic advancements across various policy gradient methods and has exhibited considerable enhancements in training stability and speed across diverse environments in classic control. This enhancement facilitates more efficient exploration of optimal policies and leads to improved convergence rates. Finally, we explore future directions for high-throughput RL, which have the potential to foster innovation and discovery in both fields.

II. REINFORCEMENT LEARNING BACKGROUND

A. GENERAL METHODOLOGIES

We explore a reinforcement learning scenario where an agent engages with an environment across discrete time steps. At each time step t, the agent perceives a state s_t and chooses an action a_t from a set of actions A based on its policy π, where π is a mapping from states s_t to actions a_t. In response, the agent transits to the next state s_{t+1} and receives a scalar reward r_t. The process continues until the agent reaches a terminal state, upon which it restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the total discounted accumulated reward from time step t with discount factor $\gamma \in (0, 1]$. The goal of the agent is to maximize the expected return from each state s_t.

The action value $Q^{\pi}(s, a) = E[R_t|s_t = s, a]$ represents the expected return for selecting action a in state s and following policy π. The optimal value function $Q^{*}(s, a) = max_{\pi} Q^{\pi}(s, a)$ gives the maximum action value for state s and action a attainable by any policy. Similarly, the value of state s under policy π is defined as $V^{\pi}(s) = E[R_t|s_t = s]$ and is the expected return following policy π from state s.

In value-based model-free reinforcement learning methods, let $Q(s, a; \theta)$ be an action-value function approximator with parameters θ. The updates to θ are derived from various reinforcement learning algorithms. For instance, Q-learning seeks to directly approximate the optimal action value function: $Q^{*}(s, a) \approx Q(s, a; \theta)$. In one-step Q-learning, the parameters θ of the action value function $Q(s, a; \theta)$ are learned by iteratively minimizing a sequence of loss functions, where the loss function is defined as $L_i(\theta_i) = E(r + \gamma maxQ(s', a'; \theta_g) - Q(s, a; \theta_i))$. Here s' is the state encountered after state s and θ_g represents the target network in general. In practice, the target network $Q(.; \theta_g)$ may differ from $Q(.; \theta_i)$ but is synchronized at specific intervals.

Different from value-based methods, policy-based methods directly parameterize the policy $\pi(a|s; \theta)$ and update the parameters θ by performing gradient ascent on $E[R_t]$ in the direction $\nabla_{\theta} log\pi(a_t|s_t; \theta)R_t$, which is an unbiased estimate of $\nabla_{\theta} E[R_t]$. To reduce the variance of this estimate while maintaining unbiasedness, a learned function of the state $b_t(s_t)$, known as a baseline, is subtracted from the return. The resulting gradient is $\nabla_{\theta} log\pi(a_t|s_t; \theta)(R_t - b_t(s_t))$. Typically, the value function $V(s_t)$ is adopted as the baseline and $A_t(s_t) = (R_t - V(s_t))$ is referred to as the advantage A_t at time step t.

B. SCALABLE REINFORCEMENT LEARNING

Scalable reinforcement learning (RL) systems commonly incorporate distributed rollout and trainer workers. Rollout workers execute the environment, generating rollouts by employing actions sampled from policy models, which can be situated on either rollout workers or trainer workers. Typically, rollout workers operate on CPU machines, occasionally utilizing GPU machines for more complex environments. Trainer workers gather rollout data asynchronously from rollout workers and iteratively optimize policies, utilizing either CPU or GPU machines [18], [19], [20].

Although this distributed architecture is scalable, it incurs high costs in worker communication and data transfer, while individual machine utilization remains suboptimal. This limits the throughput to the concurrency level of a few dozen worker machines at most [15]. To enhance performance, there are GPU- and TPU-based RL frameworks available, albeit primarily focused on gaming or robotics applications [17], [21], [22], [23], [24], [25]. Constructing efficient RL pipelines and algorithms for control simulations involving intricate agent interactions, significant data consumption, and diverse environments remains a challenging task.

III. COMPUTATIONAL ARCHITECTURE

Utilizing the WarpDrive architecture [17], [26], as depicted in Fig. 1, we seamlessly execute the entire RL workflow on a single GPU. This is facilitated by a unified data storage system hosted within the GPU, catering to simulation rollouts, action inference, reset, and training. By adopting this approach, we minimize CPU-GPU data communication and eliminate the need for additional data transfer within the GPU. Consequently, we observe a significant reduction in both simulation and training times.

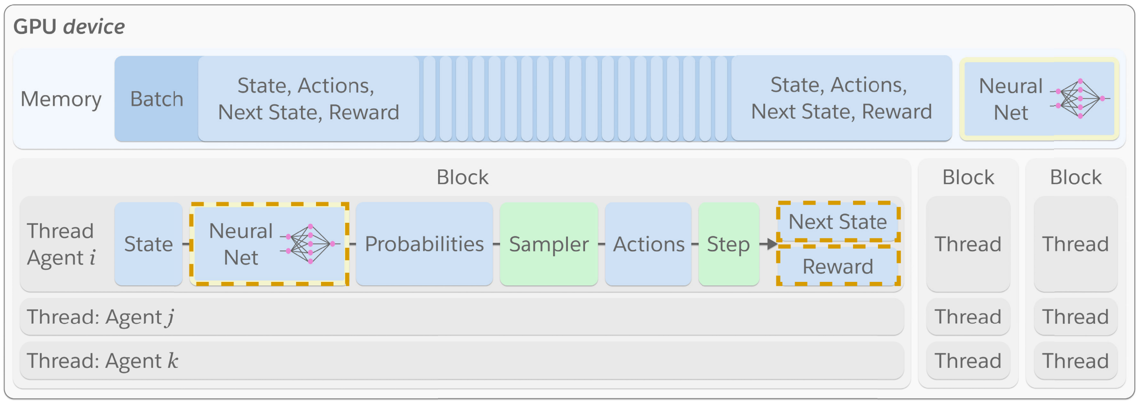

FIGURE 1. A flow chart depicting our computational framework. Computations within this framework are organized into GPU blocks, each comprising multiple threads to facilitate concurrent environment rollouts. Each thread is responsible for operating an agent that samples actions and computes rewards. These blocks have access to the global GPU memory, which houses the RL environment with local variations, and deep policy models. Additionally, they store in-place rollout data for training purposes. The dashed brown boxes represent references (not copies) of the policy model objects and data placeholders managed by blocks and hosted in the global memory. Users have the flexibility to compose and upload their custom environment setups to finalize the environment construction.

Moreover, our framework achieves parallelization at a low cost by concurrently running thousands of single-agent or multi-agent simulations, leveraging the inherent parallel processing capabilities of GPUs. Each environment instance operates independently within a dedicated GPU block, with individual agents running on unique GPU threads, facilitating interactions across threads. Notably, this framework is designed to be memory efficient. For example, each instance maintains a reference (rather than a copy) to the environment, incorporating local variations or random configurations. The neural network models are shared references among all instances and data tuples are updated in place in the global memory. This significantly mitigates the storage overhead associated with environment setup. These design choices enable running thousands of concurrent simulations and training on extremely large batches of experience.

Our framework also provides utility tools to simplify developing and running simulations on a GPU with light-weight wrapper classes that work with the Python service and environment modules. It enables automatic building of gym-style environment objects and runs them on the GPU using just a few lines of code [26]. As such, this framework is flexible and accommodates environments with a wide assortment of interactions, models, and learning algorithms.

This high-throughput, cost-effective architecture presents distinct advantages, especially in classic control problems. Such problems typically necessitate substantial data consumption, involve complex agent interactions, and encompass diverse environments. We have improved several RL algorithms tailored specifically for classic control scenarios, designed to take full advantage of this high-throughput architecture running end-to-end on GPUs.

IV. HIGH-THROUGHPUT POLICY GRADIENT ALGORITHMS

We now propose massively high-throughput variants of advantage actor-critic (A2C) and deep deterministic policy gradient (DDPG) leveraging unified data storage hosted within the GPU and the inherent concurrent processing capability of GPU.

A. HIGH-THROUGHPUT A2C

Outlined in the pseudocode provided in Algorithm 1, similar to A2C [15], high-throughput A2C (HA2C) maintains a globally shared policy $\pi(a|s;\theta_p)$ and an estimated value function $V(s;\theta_v)$. These functions are updated after every B steps on N concurrent environment replicas, with updates performed using the policy gradient $\nabla_{\theta_p}\log\pi(A|O;\theta_p)(Ret - V(O;\theta_v))$, where Ret denotes the accumulated return from the initial to the terminal state. Notably, in HA2C, observations O and actions A are represented in high-dimensional batches with the outermost shape of $[B,N]$, thus denoted by capital letters.

It is important to highlight the key distinctions between HA2C and conventional A2C. Firstly, all computations and data are contained within the GPU device, eliminating the need for CPU-GPU data transfer and communication. Secondly, as illustrated in Line 16 of Algorithm 1, rollouts of observations, actions, and rewards are concurrently collected by individual environment instances and are directly saved in-place to the corresponding data batches with dimensions matching those of the environment instances. Thirdly, as demonstrated in Line 25 of Algorithm 1, each individual environment instance resets independently without affecting the trajectory updates of other environments. The environment reset is executed by the *EnvResetManager*, which generates local variations or random configurations for the initial states of individual environment instances independently.

B. HIGH-THROUGHPUT DDPG

Outlined in the pseudocode provided in Algorithm 2, High-Throughput Deep Deterministic Policy Gradient (HDDPG), akin to DDPG [12], learns a deterministic policy $\mu(s;\theta_\mu)$ instead of a policy distribution. HDDPG maintains both exploration and target networks. To enhance exploration,

Algorithm 1 Massively High-Throughput Advantage Actor-Critic

```
1  Require
2      Concurrent RL environment replicas $E_1$ to $E_N$, hosted individually by N computational blocks of a GPU
3  end
4  Require
5      All execution are within the GPU device
6  end
7  Initialize
8      global: policy network $\pi(a|s;\theta_p)$ with weights $\theta_p$ and value network $V(s;\theta_v)$ with weights $\theta_v$
9      global: environment resetting manager $EnvResetManager$
10     global: observation batch $O$, action batch $A$, reward batch $R$, done batch $D$ with the shape $[B, N, *]$ where $B$ is the
       batch size
11     global: shared counter $T = 0$
12 end
13 $EnvResetManager.reset([E_1, E_2, \ldots E_N])$;
14 while $T < T_{max}$ do
15     reset $d\theta_p \leftarrow 0$ and $d\theta_v \leftarrow 0$;
16     Do In Parallel ($\mathbf{E}_1$ to $\mathbf{E}_n$)
17         while $t < B$ do
18             Perform $a_t$ according to policy $\pi(a_t|s_t;\theta_p)$;
19             Receive reward $r_t$ and new state $s_{t+1}$;
20             $O_t[E_i] \leftarrow s_t$;
21             $R_t[E_i] \leftarrow r_t$;
22             $A_t[E_i] \leftarrow a_t$;
23             $s_t \leftarrow s_{t+1}$;
24             $t \leftarrow t + 1$;
25             if terminated or reaching maximum episode length for $E_i$ then
26                 $D_t[E_i] \leftarrow 1$;
27                 $EnvResetManager.reset(E_i)$;
28             else
29                 $D_t[E_i] \leftarrow 0$;
30             end
31         end
32     end
33     Calculate the accumulated returns $Ret$ from $R$ and $D$;
34     $d\theta_p \leftarrow d\theta_p + \nabla_{\theta_p} log\pi(A|O;\theta_p)(Ret - V(O;\theta_v))$;
35     $d\theta_v \leftarrow d\theta_v + \nabla_{\theta_v}(Ret - V(O;\theta_v))^2$;
36 end
```

noise is added to the exploration policy μ. Additionally, we incorporate N-step returns Ret_n, defined as $r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n Q(s_n, \mu(s_n; \theta_\mu))$ to account for rewards in multiple future steps, resulting in a more stable estimation of future return. Furthermore, HDDPG performs soft updates on the parameters using $\theta \leftarrow \tau\theta + (1-\tau)\theta'$.

Apart from the inherent differences between our high-throughput implementation and conventional methods detailed in the previous section on HA2C, it's important to highlight a key distinction between HDDPG and DDPG. Instead of employing an off-policy experience replay buffer to stabilize the learning of the Q function, our proposed HDDPG simply maintains an on-policy buffer to train the current rollouts exclusively, as demonstrated in Lines 20 to 22 of Algorithm 2. Although this method can be extended to off-policy learning by retaining more historical data, we find that the on-policy buffer suffices for our studies due to the vast number of concurrent environments, which provide sufficiently diverse trajectories.

V. ENVIRONMENTS

We use the set of classic control environments proposed by OpenAI Gym for assessing the capabilities and properties of the proposed framework [27]. These environments provide classic control scenarios that mimic real-world control problems with well-defined state and action spaces, as well as clear performance metrics. Therefore, they serve as standardized and foundational testbeds for evaluating reinforcement learning algorithms on classic control problems.

We reimplement the step functions of these environments using Numba [28], a just-in-time compiler for Python code, to convert them into CUDA kernels and device functions following the CUDA execution model. The behaviors of these environments are thoroughly validated for consistency

Algorithm 2 Massively High-Throughput Deep Deterministic Policy Gradient

```
Require
    Concurrent RL environment replicas $E_1$ to $E_N$, hosted individually by N computational blocks of a GPU
end
Require
    All executions are within the GPU device
end
Initialize
    global: critic network $Q(s, a; \theta_Q)$ and actor network $\mu(s; \theta_\mu)$, target critic $Q'$ and target actor $\mu'$ with weights $\theta_{Q'} \leftarrow \theta_Q, \theta_{\mu'} \leftarrow \theta_\mu$
    global: environment resetting manager $EnvResetManager$
    global: random noise generator $RandomNoiseGenerator$
    global: observation buffer $O$, action buffer $A$, reward buffer $R$, done buffer $D$ with the shape $[B, N, *]$ where $B$ is the batch size. Here, $O, A, R, D$ are FIFO.
    global: shared counter T = 0
end
while $T < T_{max}$ do
    reset $d\theta_p \leftarrow 0$ and $d\theta_v \leftarrow 0$;
    Do In Parallel ($E_1$ to $E_n$)
        while $t < B$ do
            Perform $a_t = \mu(s_t; \theta_\mu) + noise$;
            Receive reward $r_t$ and new state $s_{t+1}$;
            $O[E_i] \leftarrow s_t$ by FIFO;
            $R[E_i] \leftarrow r_t$ by FIFO;
            $A[E_i] \leftarrow a_t$ by FIFO;
            $s_t \leftarrow s_{t+1}$;
            $t \leftarrow t + 1$;
            if $terminated\ for\ E_i$ then
                $D[E_i] \leftarrow 1$ by FIFO;
                $EnvResetManager.reset(E_i)$;
            else
                $D[E_i] \leftarrow 0$ by FIFO;
            end
        end
    end
    Calculate the n-step returns $Ret_n$ from $R$ and $D$;
    $d\theta_\mu \leftarrow d\theta_\mu + \nabla_a Q(O, a; \theta_Q)|_{a=\mu(S)} \nabla_{\theta_\mu} \mu(S; \theta_\mu)$;
    $d\theta_Q \leftarrow d\theta_Q + \nabla_{\theta_Q} (Ret_n - Q(O, A; \theta_Q))^2$;
    $\theta_{Q'} \leftarrow \tau\theta_Q + (1 - \tau)\theta_{Q'}$;
    $\theta_{\mu'} \leftarrow \tau\theta_\mu + (1 - \tau)\theta_{\mu'}$;
end
```

with the original Python implementation provided by OpenAI Gym, available at https://github.com/openai/gym. Subsequently, these environments are loaded and integrated into WarpDrive's CUDA back-ends for use in RL simulations. Additionally, we optimize the original Python step function with an optimized NumPy implementation to establish a fair CPU baseline. The following section provides a concise overview of these environments.

A. CARTPOLE

Cartpole environment is a classic reinforcement learning problem designed to test an agent's ability to balance a pole on a cart. In this environment, there is a cart that can move along a frictionless track, and it must balance a pole that is attached to it. The agent's goal is to prevent the pole from falling over by applying appropriate discrete actions (pushing the cart left or right). The state of the environment is typically represented by four variables: the cart's position, velocity, the angle of the pole, and the angular velocity of the pole. The episode ends if the pole tilts beyond a certain angle or if the cart moves too far from the center of the track. The objective for the agent is to keep the pole balanced for as long as possible, which is often measured by the number of time steps it can maintain balance before the episode terminates.

B. ACROBOT

Acrobot is another classic reinforcement learning problem. In this environment, there is a two-link pendulum system

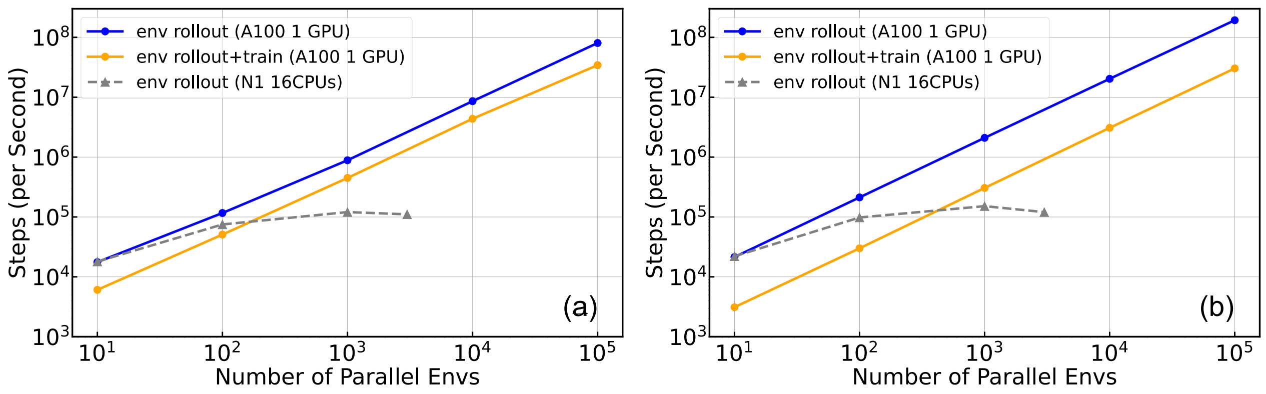

FIGURE 2. Rollout and training throughput versus the number of parallel environments (log-log scale) to 100K concurrent environments with random local initialization. The GPU simulation, running on a single Nvidia A100 GPU, scales throughput linearly for (a) Cartpole and Acrobot environments and (b) Pendulum environment. In (a), the throughput exhibits negligible differences between Cartpole and Acrobot and the presented value is an average over the two environments with a small relative standard deviation of 6 percent. For both (a) and (b), the CPU simulation, running on a 16-CPU node, has significantly lower throughput and cannot scale to more than a few thousand environments.

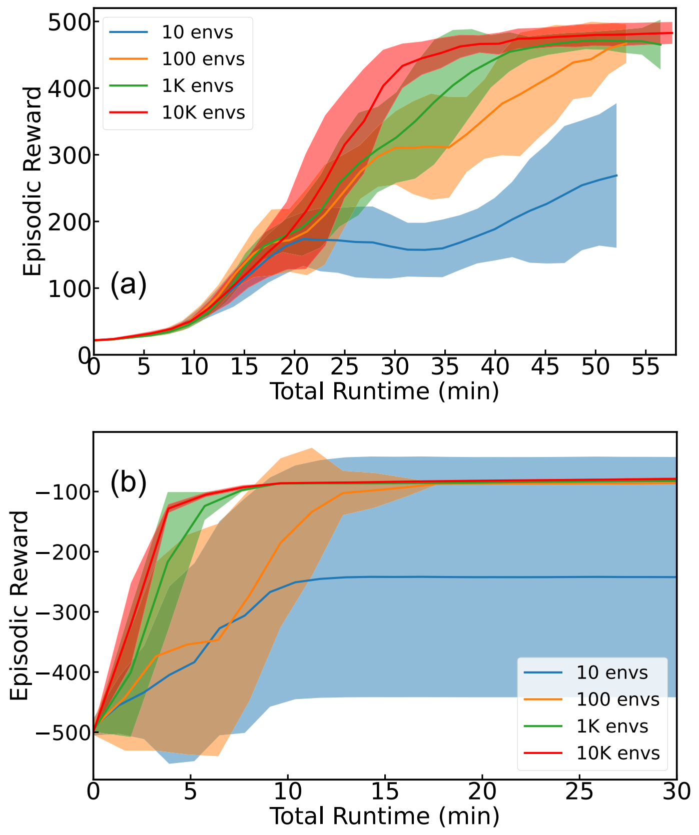

FIGURE 3. The average episodic reward (the accumulated total reward collected from the start to the terminal state) versus the training time (wall-clock minutes) for (a) Cartpole and (b) Acrobot running at various concurrency levels. For robustness, the depicted results are averaged over eight independent runs from scratch with different initialization seeds and the same hyperparameters. The shadow regions represent the error bars of eight independent runs.

known as the acrobot, which is attached to a fixed pivot point. The goal for the agent is to swing the bottom link of the acrobot up to a certain height by applying discrete torques to the joint between the two links. The state of the environment is typically represented by the angles and angular velocities of the two links. The episode ends when either the top link reaches a certain height or a maximum number of time steps is reached. The objective for the agent is to learn a policy that allows it to swing the bottom link up to the desired height using as few torques as possible.

C. CONTINUOUS PENDULUM

Continuous Pendulum is designed to simulate a physical pendulum. In this environment, there is a massless rod and a point mass at the end. The pendulum is attached to a pivot point and can swing freely in a two-dimensional plane. The state of the environment is typically represented by the angle of the pendulum relative to the vertical axis and its angular velocity. The action space consists of continuous torques that can be applied to the pendulum, allowing the agent to control its movement. The goal for the agent is to learn a policy that stabilizes the pendulum in an upright position by minimizing the angle deviation from the vertical axis with as little torque as possible.

VI. EXPERIMENTS

A. EFFICIENCY AND SCALABILITY

All experiments were conducted on a single Nvidia A100 GPU, specifically the *a2-highgpu-1g* instance, hosted on the Google Cloud Platform (GCP). Performance comparisons were made against a conventional distributed architecture [15], running on a 16-CPU node at GCP, specifically the *n1-standard-16* instance. As depicted in Fig. 2, our experiments achieved an impressive throughput of approximately 80.4 million environment steps per second and 34.3 million total (environment rollout + training) steps per second with 100,000 concurrent environment instances. While all three environments exhibit small differences for the throughput of environment rollout, there is 20-30% drop of the training throughput in Pendulum. This is primarily due to the more complex training algorithm of HDDPG, for example, the dual model updates between the target network and the on-policy network.

With 10 concurrent environment instances, our framework exhibited a throughput of approximately 17K environment steps per second, comparable to that of a conventional distributed system. However, such distributed systems can barely scale to a few thousand environment instances and suffer from serious throughput saturation. In contrast, our framework's performance in all our classic control environments scales linearly up to 100,000 environment instances with a single GPU. This scalability underscores the efficiency of our framework in classic control environments, showcasing near-perfect parallelism and indicating minimal data transfer and communication costs. This insight highlights the robustness and effectiveness of our approach in handling large-scale RL simulations with high throughput.

B. LEARNING SPEED AND STABILITY

We study the convergence speed of our framework as a function of the number of environment replicas running in parallel. The data reveal that, under consistent fixed hyperparameters, the simulations operating with an increased number of concurrent environments attain global convergence faster and more stably.

For Cartpole and Acrobot simulations with discrete action spaces, we employed the proposed HA2C algorithm across various levels of concurrency, ranging from 10 to 10,000 environment instances. As depicted in Fig. 3, with 10,000 Cartpole and Acrobot environment replicas, we observed attainment of the global optimum within 30 and 5 minutes,

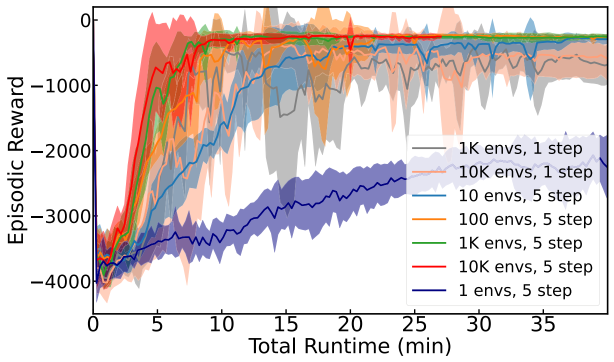

FIGURE 4. The average episodic reward (the accumulated total reward collected from the start to the terminal state) versus the training time (wall-clock minutes) for Continuous Pendulum running at various concurrency levels and N-step returns (N=1 or 5). For robustness, the depicted results are averaged over eight independent runs from scratch with different initialization seeds and the same hyperparameters. The shadow regions represent the error bars of eight independent runs.

respectively. In contrast, using only 10 environment replicas resulted in insufficient convergence within such a short timeframe. Notably, the convergence variance, as indicated by the error bars over eight independent runs, noticeably decreased with higher concurrency levels, suggesting significantly enhanced robustness with increased throughput.

For the Pendulum environment, which utilizes a continuous action space, we applied the proposed on-policy HDDPG algorithm without an experience buffer across various levels of concurrency, ranging from 1 to 10,000 environment replicas. As depicted in Fig. 4, with 10,000 environment replicas and 5-step returns, the agent reached the global optimum within 8 minutes. Conversely, lower throughput resulted in insufficient convergence and increased variance. It is worth noting that N-step returns, which incorporate rewards from more future steps, tend to stabilize the training process, as evidenced by the comparison of convergence trends between 5-step and 1-step returns for the same number of environment instances.

VII. CONCLUSION

We have introduced a massively high-throughput RL study for classic control problems, harnessing highly efficient concurrent computations through our proposed architecture and algorithms executed on GPUs. Our findings demonstrate that our methods can efficiently train RL agents across a range of classic control problems, spanning both discrete and continuous domains, achieving fast and stable performance. Notably, our approach surpasses the current state-of-the-art methods with significantly less training time and increased robustness. Additionally, we observed that RL exploration with a large scale of parallel instances has a remarkable stabilizing effect on updating a shared model. For instance, stable DDPG is achievable without the need for experience replay, as demonstrated in this study.

Overall, our aim is to contribute to the democratization of high-performance RL systems and drive advancements in classic control problems. We hope that our study encourages leveraging high-throughput GPU simulations and inspires further efforts to extend and integrate our methods with other tools for rapidly constructing simulations tailored for classic control workflows on GPUs or other accelerators.

VIII. FUTURE WORK

As high-throughput RL continues to evolve, there are several promising avenues for future research in the realm of classic control. One direction is the exploration of more complex and dynamic environments that better mimic real-world scenarios, such as autonomous vehicle control. These environments present unique challenges, including high-dimensional state and action spaces, as well as intricate dynamics that require sophisticated control strategies. Furthermore, investigating the transferability of learned policies across different environments and tasks could lead to more versatile and generalizable control algorithms. Finally, exploring the integration of domain knowledge and physical principles into RL algorithms could provide valuable insights into the design of more efficient and robust control strategies. Overall, the future of high-throughput reinforcement learning in classic control holds great promise for advancing our understanding of intelligent control systems and accelerating their practical applications.

REFERENCES

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. London, U.K.: Pearson, 2016.

[2] OpenAI. (2018). *OpenAI Five*. [Online]. Available: https://blog.openai.com/openai-five/

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, and P. Georgiev, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[4] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 3389–3396.

[5] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: Lessons we have learned," *Int. J. Robot. Res.*, vol. 40, nos. 4–5, pp. 698–721, Jan. 2021, doi: 10.1177/0278364920987859.

[6] X. Wang, Y. Wang, X. Su, L. Wang, C. Lu, H. Peng, and J. Liu, "Deep reinforcement learning-based air combat maneuver decision-making: Literature review, implementation tutorial and future direction," *Artif. Intell. Rev.*, vol. 57, no. 1, p. 1, Jan. 2024.

[7] S. Zheng, A. Trott, S. Srinivasa, D. C. Parkes, and R. Socher, "The AI economist: Taxation policy design via two-level deep multiagent reinforcement learning," *Sci. Adv.*, vol. 8, no. 18, p. eabk2607, May 2022.

[8] A. Trott, S. Srinivasa, D. van der Wal, S. Haneuse, and S. Zheng, "Building a foundation for data-driven, interpretable, and robust policy design using the AI economist," 2021, *arXiv:2108.02904*.

[9] T. Lan and Q. An, "Discovering catalytic reaction networks using deep reinforcement learning from first-principles," *J. Amer. Chem. Soc.*, vol. 143, no. 40, pp. 16804–16812, Oct. 2021. [Online]. Available: https://pubs.acs.org/doi/abs/10.1021/jacs.1c08794

[10] J. Nousiainen, C. Rajani, M. Kasper, T. Helin, S. Y. Haffert, C. Vérinaud, J. R. Males, K. Van Gorkom, L. M. Close, J. D. Long, A. D. Hedglen, O. Guyon, L. Schatz, M. Kautz, J. Lumbres, A. Rodack, J. M. Knight, and K. Miller, "Toward on-sky adaptive optics control using reinforcement learning-model-based policy optimization for adaptive optics," *Astron. Astrophys.*, vol. 664, p. A71, Aug. 2022, doi: 10.1051/0004-6361/202243311.

[11] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 5026–5033.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[13] K. R. Williams, R. Schlossman, D. Whitten, J. Ingram, S. Musuvathy, J. Pagan, K. A. Williams, S. Green, A. Patel, A. Mazumdar, and J. Parish, "Trajectory planning with deep reinforcement learning in high-level action spaces," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 59, no. 3, pp. 1–16, Jun. 2023, doi: 10.1109/TAES.2022.3218496.

[14] J. Rabault, F. Ren, W. Zhang, H. Tang, and H. Xu, "Deep reinforcement learning in fluid mechanics: A promising method for both active flow control and shape optimization," 2020, *arXiv:2001.02464*.

[15] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016, *arXiv:1602.01783*.

[16] K. Zhang, A. Koppel, H. Zhu, and T. Başar, "Global convergence of policy gradient methods to (almost) locally optimal policies," 2019, *arXiv:1906.08383*.

[17] T. Lan, S. Srinivasa, H. Wang, and S. Zheng, "WarpDrive: Fast end-to-end deep multi-agent reinforcement learning on a GPU," *J. Mach. Learn. Res.*, vol. 23, no. 316, pp. 1–6, 2022. [Online]. Available: http://jmlr.org/papers/v23/22-0185.html

[18] R. de Kock, O. Mahjoub, S. Abramowitz, W. Khlifi, C. R. Tilbury, C. Formanek, A. Smit, and A. Pretorius, "Mava: A research library for distributed multi-agent reinforcement learning in JAX," 2021, *arXiv:2107.01460*.

[19] M. W. Hoffman et al., "Acme: A research framework for distributed reinforcement learning," 2020, *arXiv:2006.00979*.

[20] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," 2018, *arXiv:1802.01561*.

[21] Y. Tang, Y. Tian, and D. Ha. (2022). *EvoJAX: Hardware-Accelerated Neuroevolution*. [Online]. Available: https://github.com/google/evojax

[22] M. Hessel, M. Kroiss, A. Clark, I. Kemaev, J. Quan, T. Keck, F. Viola, and H. van Hasselt, "Podracer architectures for scalable reinforcement learning," 2021, *arXiv:2104.06272*.

[23] S. Dalton, I. Frosio, and M. Garland, "Accelerating reinforcement learning through GPU Atari emulation," 2019, *arXiv:1907.08467*.

[24] C. Daniel Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax—A differentiable physics engine for large scale rigid body simulation," 2021, *arXiv:2106.13281*.

[25] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance GPU-based physics simulation for robot learning," 2021, *arXiv:2108.10470*.

[26] T. Lan, S. Srinivasa, H. Wang, and S. Zheng, "WarpDrive: Extremely fast end-to-end deep multi-agent reinforcement learning on a GPU," 2021, *arXiv:2108.13976*.

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

[28] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-based Python JIT compiler," in *Proc. 2nd Workshop LLVM Compiler Infrastruct. HPC*, Nov. 2015, pp. 1–6.

XUAN SHA received the M.S. degree in architecture and civil engineering from Jiangsu University, Zhenjiang, China, in 2016. She is currently pursuing the Ph.D. degree in mechanics with the College of Mechanics and Engineering Science, Hohai University, Nanjing, China. In 2016, she began teaching with the School of Civil Engineering and Transportation, Southeast University Chengxian College, where she is a Lecturer. Her research interests include dynamics and optimal controls.

TIAN LAN received the Ph.D. degree in applied physics (electrical engineering) from California Institute of Technology, Pasadena, USA, in 2014. He is currently a Lead Research Scientist with Salesforce AI Research. He has extensive experience building up large-scale and massively parallel computational simulation platforms for academia, automated trading, and high-tech industry.

• • •