Accelerating Multi-Agent DDPG Training on Multi-GPU Platforms

Samuel Wiggins, Viktor Prasanna
Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California
Contact: {wigginss, prasanna}@usc.edu

***Abstract*—Multi-Agent Reinforcement Learning (MARL) is a crucial technology in artificial intelligence applications. Multi-Agent Deep Deterministic Policy Gradient (MADDPG) is a state-of-the-art MARL algorithm that has gained widespread adoption and is considered a popular baseline for comparing against novel MARL algorithms. The training process of MADDPG systems involves a high volume of data communication and computation when scaling to larger problem sizes. While state-of-the-art CPU-GPU systems provide the necessary computing power for high-throughput training, they do not efficiently utilize underlying platform resources and are unable to facilitate training using multiple GPU devices. We propose a novel accelerated system for MADDPG training that leverages multiple GPU devices while additionally increasing the utilization of CPU resources. We access our MADDPG system on multiple benchmarks on a multi-GPU platform, resulting in up to a 1.5× higher system throughput compared to state-of-the-art CPU-GPU systems.**

***Index Terms*—Multi-Agent Reinforcement Learning, Multi-GPU, MADDPG**

I. Introduction

Multi-agent reinforcement learning (MARL) is an emerging technology that has seen success in various applications such as wireless networks [1], Unmanned Aerial Vehicle (UAV) deployment [2], power distribution networks [3], etc. Compared to single-agent Reinforcement Learning (SARL), MARL introduces partial observability and non-stationarity challenges when multiple agents learn in a shared environment [4]. Inter-agent communication and coordination help alleviate these challenges, enabling agents (or sub-sets of agents) to learn together in a dynamic environment. Thus, MARL algorithms differ in how they manage inter-agent dependencies and how effectively they balance exploration in a multi-agent setting.

Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [5] is a state-of-the-art (SOTA) MARL algorithm that alleviates the communication learning problem through the exchange of actions and observations between agents in an all-to-all manner during the training process. It is considered a Centralized-Training Decentralized-Execution (CTDE) MARL algorithm, where all agents train centrally (using this all-to-all exchange) while executing their own determined action on an environment using their individual policy. MADDPG is one of the earliest successful MARL algorithms, demonstrating effectiveness in various fields, including smart grids [6], Internet of Things (IoT) applications [7], satellite-UAV networks [8], intelligent radar anti-jamming decision-making [9], etc.

This work was supported by the U.S. National Science Foundation (NSF) under grant SaTC-2104264.

We conduct a comprehensive analysis of current GPU-enabled MADDPG systems and observe several limitations and inefficiencies that impact their overall speed performance. *(1) Sequential Execution of MARL Training Tasks:* Current CPU-GPU MADDPG systems partition the entire environment sampling and agent training phases on different devices, performing each stage sequentially, effectively underutilizing the heterogeneous platform's compute resources. Such a task allocation is suboptimal since MADDPG is an off-policy MARL algorithm with no strict dependency on using only newly generated experiences. *(2) Inability to train using Multiple GPUs:* State-of-the-art implementations of MADDPG are unable to be trained on multi-GPU platforms. Using the vast data-parallel compute resources of multiple GPU devices should be considered when training large-scale MARL scenarios.

Motivated by the above limitations, we propose a novel system for MADDPG training. Our system can launch multiple processes that collaboratively train on more than one GPU device. We further optimize our system by concurrently executing tasks of off-policy MADDPG. We justify the use of a CPU-multi-GPU platform for MADDPG training. (1) Environment Sampling on CPU: Most open-source environments are developed on CPU devices for easy plug-and-play between different application-specific software simulations. (2) Model Training on GPUs: Large-scale MARL training scenarios, which involve many agents, large model sizes, and large batch sizes, can take advantage of the data-parallel compute cores of multiple GPU devices, which allows for handling larger models and datasets that exceed the memory capacity of a single GPU. Each GPU processes a portion of the input batch in parallel, leading to reduced training time and increased utilization of underlying platform resources.

Our main contributions are:

- We conduct an extensive analysis identifying limitations and inefficiencies of current MADDPG training systems.
- We propose a novel MADDPG training system that effectively utilizes system resources of a multi-GPU platform.
- We perform several ablation studies to confirm that our optimizations have little to no effect on reward convergence compared to baseline MADDPG implementations.
- We evaluate our multi-GPU MADDPG system on multiple large-scale benchmarks, resulting in up to 1.5× higher system throughput compared to CPU-single-GPU MADDPG systems.

II. Background

A. Multi-Agent Deep Deterministic Policy Gradient

We consider a partially observable N-agent Markov Game [10], a multi-agent extension of traditional Markov Decision Processes [11], composed of a state space $\mathcal{S}$, an action space for each agent $\mathcal{A}_i$ ($i \in 1...N$), a reward function for each agent $R_i : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, and transition probability $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto P(\mathcal{S})$ which defines the probability for agents to transition to other states given their current states and actions taken. For agent i, we denote its policy as a probability distribution over its action space $\pi_i : \mathcal{S} \rightarrow P(\mathcal{A}_i)$. Each agent i aims to find an optimal policy that maximizes its own expected cumulative reward until terminal time step T: $R_i = \sum_{t=0}^{T} \gamma^i r_i^t$, where γ is a discount factor. Multi-Agent Deep Deterministic Policy Gradient (MADDPG) follows an actor-critic training paradigm [12], using four Deep Neural Network (DNN) models for each agent. One DNN model is used to approximate the action-value function (critic network) and helps facilitate the training of an agent's policy model. Two additional DNN models, target critic and policy networks, are used for training stability [13] to train their non-target counterparts. These models are usually represented by Multi-Layer Perceptrons (MLPs).

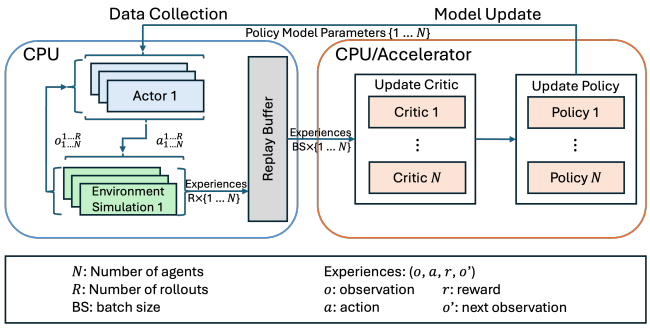

Fig. 1. Current state-of-the-art training of MADDPG systems

The training process of MADDPG can be viewed as two distinct phases, as shown in Figure 1. **(1) Data Collection**: N Actors, each holding an agent's policy network, interact with multiple (determined by the number of rollouts) environment simulations, taking actions based on received observations. Experience tuples containing data points for training are stored in a replay buffer. **(2) Model Update:** A Learner is responsible for agent critic and policy networks training through Stochastic Gradient Descent (SGD) [14] using a batch of experiences sampled from the replay buffer. In MADDPG, critic models are updated first, followed by policy networks. Updated policies are then used to interact with the environment in the following Data Collection phase. This phase in current SOTA MADDPG implementations can either happen on the CPU or offload to an accelerator device, such as a GPU, FPGA, etc., to use additional compute resources on those platforms.

B. Challenges of training large-scale MADDPG Systems

There are several limitations that impact the speed performance of current MADDPG systems. Figure 2 shows a single-training-iteration of MADDPG on an environment simulation Rover Tower [15] based on the popular Multi-Particle Environment (MPE) [5] using a CPU-GPU platform. This figure is representative of different environment simulations. While the ratio between Data Collection and Model Update execution times may differ between different simulations (since state space $\mathcal{S}$ and an agent's action space $\mathcal{A}$ may have different dimensions), increasing the number of agents (along with batch size, model size, etc.) will result in the Model Update phase continuing to dominate end-to-end MADDPG training. This motivates the need to accelerate the Model Update phase of MADDPG. A data-parallel multi-GPU approach can reduce execution time by concurrently processing a subset of the full batch of experiences on each GPU device. More information about our multi-GPU system is discussed in Section III.

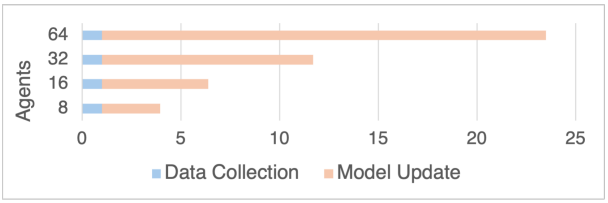

Fig. 2. Normalized Execution Time Breakdown

C. Related Work

There are few works that focus on accelerating MARL systems. RLlib [16] is a popular SARL library with support for a few decentralized MARL algorithms. However, multi-GPU support can only be applied to SARL algorithms. MARLlib [17] extends RLlib to include additional popular MARL algorithms, including MADDPG. However, MARLlib does not have any multi-GPU support. [18] introduces a CPU-FPGA heterogeneous system that accelerates MADDPG training. However, this work only explores smaller-scale MPE environments with few agents. All of the above implementations do not enable concurrent Data Collection and Model Update for their respective off-policy algorithms.

III. Multi-GPU MADDPG Training System

A. System Overview

Our MADDPG system, shown in Figure 3, consists of an Actor process and multiple Learner processes. An example notation: "Experience Sampling i (G.1)" refers to the Experience Sampling at iteration i for GPU 1. The Actor process is responsible for the Data Collection Phase, where agent policy models interact with the environment simulations to generate experiences. These experiences (observation, action, next observation, reward) are stored in a replay buffer that is created in a shared memory space available to all processes (Actor and Learners). The Actor process appends experiences to this shared replay buffer, while the Learner processes sample a random batch of experiences to perform Critic and Policy network updates using SGD. Unlike current state-of-the-art MADDPG implementations, our system can deploy multiple Learner processes on multiple GPU devices, addressing the limitation of the *Inability to train using Multiple GPUs*. We use a batched-data-parallel training scheme across

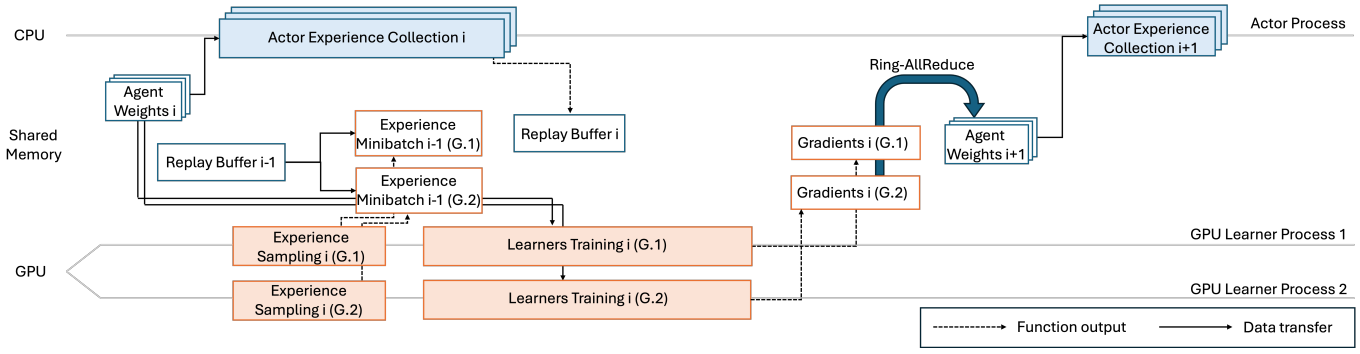

Fig. 3. MADDPG Training on a Multi-GPU Platform

GPU devices. Each GPU device stores copies of all agent neural networks in GPU global memory. We send a micro-batch (a subset of the whole batch; i.e., for a batch size of 1024, two GPUs will each get 512 samples) to each GPU Learner in order to perform SGD. Gradients are synchronized across Learner processes using a Ring AllReduce operation, where each process can then calculate the same final Critic and Policy weights with the aggregated gradients. Updated agent policy weights are then used in the next Data Collection phase to generate new experiences.

MADDPG is an off-policy MARL algorithm. Actors interacting with simulations generate experiences during the Data Collection Phase and stores these experiences in a replay buffer. Learners samples a batch of experiences from the replay buffer in the following Model Update phase. It is possible that a Learner samples experiences that were not collected during the previous Data Collection iteration. Experience tuples that are multiple iterations old (meaning they were generated from outdated policy models) are still considered in off-policy MARL. Because of this, our system addresses the limitation of *Sequential Execution of MARL Training Tasks* by enabling concurrent execution of Data Collection and Model Update Phases (hence Experience Sampling from the replay buffer that is from the previous iteration: i-1 from Figure 3).

IV. Ablation Studies

A. Overlapping Data Collection and Model Update Phases

A key optimization of our system is leveraging the off-policy nature of MADDPG by overlapping the Data Collection and Model Update phases rather than executing each task sequentially. This means that Actors in the Data Collection phase will sample experiences from the environment with policy networks that are an iteration old. Figure 4 shows the mean episode reward using our optimization against the original synchronous training method for an example environment simulation Cooperative Communication from the MPE. We average across ten different runs and observe that this optimization has a negligible (0-2%) impact on reward convergence. This trend is consistent with other environment simulations. Overlapping these phases is particularly beneficial in scenarios with a fewer number of agents, where the execution times of both phases are relatively close. Our speedup benefit of overlapping both phases is bounded by the single-iteration-execution time of the slowest phase.

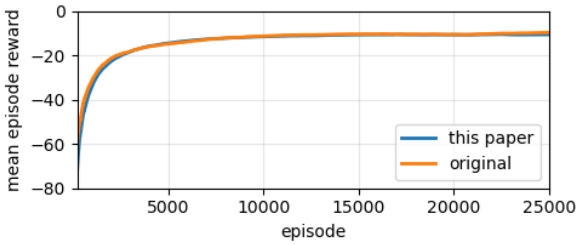

Fig. 4. Reward Convergence Comparison overlapping MADDPG Phases

B. Shared Replay Buffer among Training Processes

Our system can deploy multiple Learner processes depending on the number of available GPUs on a given platform. These GPU Learner processes need to be able to access the shared replay buffer that stores the experience tuples generated by the Actor process. In traditional single-Learner MADDPG training, a single CPU or GPU process randomly samples a batch of experiences from the replay buffer. In our system, each Learner process samples its own micro-batch of experiences, where it is possible to select duplicate or different samples because of the randomness of sampling from the replay buffer. Figure 5 shows the mean episode reward of a single Learner process compared to ours with multiple Learners averaged across 10 runs. Again, we see negligible convergence differences (slightly in the beginning) between the two (0-5%).

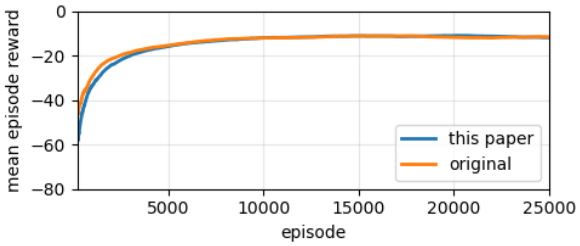

Fig. 5. Reward Convergence Comparison using a Shared Replay Buffer

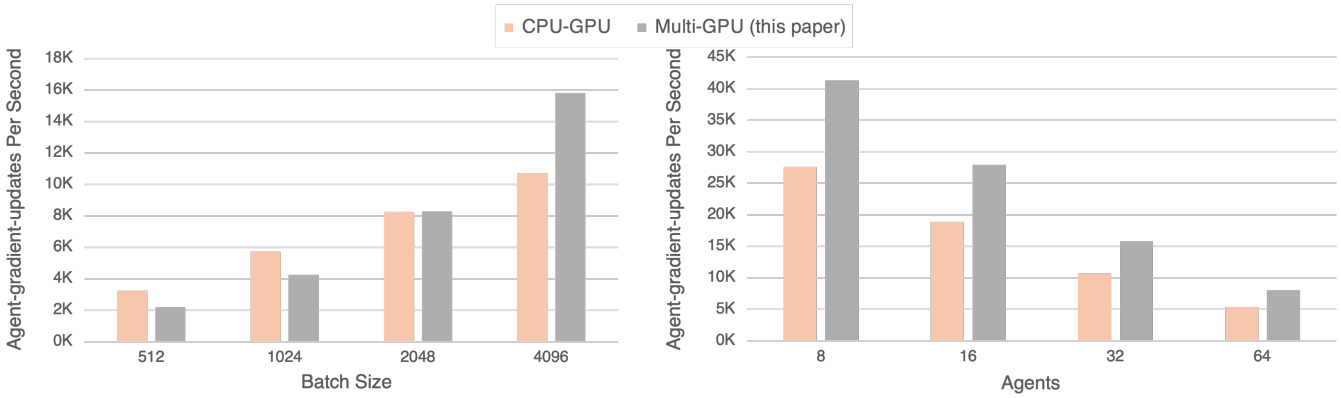

Fig. 6. MADDPG System Throughput Comparison with varying Batch Sizes (left) and number of Agents (right)

V. Evaluation

A. Experiment Setup

Performance Metrics: The main metric optimized by an acceleration system for MARL is the system throughput in terms of number of Agent-gradient-updates Per Second (APS):

$$APS = \frac{\text{number of agents} \times \text{batch size}}{T_{iteration}}, \quad (1)$$

where $T_{iteration}$ is the single-training-iteration execution time. Since we are overlapping the Data Collection and Model Update phases in our implementation, $T_{iteration} = \max(T_{DC}, T_{MU})$, where T_{DC} and T_{MU} are the single-training-iteration execution times of the Data Collection and Model Update phases, respectively.

TABLE I
Platform Specifications

Platform	CPU AMD EPYC 9754	GPU NVIDIA H100
Frequency	3.1 GHz	1.755 GHz
Memory Bandwidth	460.8 GB/s	2.04 TB/s
On-Chip Memory	256 MB L3 Cache	50 MB L2 Cache
Peak Performance	6.35 TFLOPS	51.2 TFLOPS

Platform and Environment Specifications: We evaluate our implementation using the Rover Tower environment from the MPE. Our results are representative of other benchmarks, which all share relatively similar simulation times with varying action and observation space dimensions. In our experiments, we vary the batch size and number of agents since GPUs greatly outperform CPUs when training on larger problem sizes. All DNNs for each agent are 3-layer MLPs with hidden dimensions of 1024. Other hyperparameters are consistent with the original MADDPG paper [5].

The specifications for our chosen devices are outlined in Table I. We use a two-socket AMD EPYC 9754 with two NVIDIA H100 GPUs connected via PCIe, where our baseline comparison is a CPU-GPU setup using the same devices. While our experimental setup only utilizes two GPU Learners on two GPU devices, our training system can be extended to accommodate any number of GPUs. We implement our design using PyTorch v2.3.1 with CUDA v12.1.0 and Python v3.10.14. We use PyTorch's DistributedDataParallel wrapper to enable multi-process MADDPG training. For our Multi-GPU implementation, we split the full batch of experiences into equal micro-batches for each GPU.

B. Performance: System Throughput (APS)

The bar plots in Figure 6 show an APS comparison between the baseline CPU-GPU and our multi-GPU implementation. The left plot varies the batch size from 512 - 4096 samples in a 32-agent experiment. We observe that the CPU-GPU platform outperforms our multi-GPU implementation when training with smaller batch sizes, specifically sizes of 1024 and below. This is because each gradient update employs full-batch forward and backward propagation for both the critic and policy networks, allowing concurrent operation on multiple samples, which scales effectively with increasing batch sizes without significantly extending T_{MU}. In our multi-GPU system, the original batch is divided into micro-batches. When these micro-batches fail to fully utilize the extensive CUDA cores available on the GPU devices, distributing the training across multiple devices becomes inefficient, coupled with the additional synchronization overhead of gradient aggregation.

The right plot varies the number of agents using a batch size of 4096. Using our multi-GPU approach, we observe up to a 1.5× increased system throughput (maximum achieved APS at 64 agents). Our optimization of overlapping Data Collection and Model Update phases has less effect when training with more agents; thus, the reduced training time using multiple GPUs has more effect when scaling to a large number of agents. The empirical results from both plots suggest that using multiple GPUs should be considered when training with large batch sizes with many agents.

VI. Conclusion

We developed the first MADDPG system that can leverage multiple GPU devices for agent policy network training. We conclude that the usage of multiple GPUs can lead to increased

system throughput when training with larger batch sizes and number of agents. Our ideas can be applied to other off-policy algorithms with different model specifications.

REFERENCES

[1] X. Lin, Y. Tang, X. Lei, J. Xia, Q. Zhou, H. Wu, and L. Fan, "Marl-based distributed cache placement for wireless networks," *IEEE Access*, vol. 7, pp. 62 606–62 615, 2019.

[2] Z. Dai, Y. Zhang, W. Zhang, X. Luo, and Z. He, "A multi-agent collaborative environment learning method for uav deployment and resource allocation," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 8, pp. 120–130, 2022.

[3] J. Wang, W. Xu, Y. Gu, W. Song, and T. C. Green, "Multi-agent reinforcement learning for active voltage control on power distribution networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3271–3284, 2021.

[4] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. De Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," *arXiv preprint arXiv:1707.09183*, 2017.

[5] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

[6] W. Lei, H. Wen, J. Wu, and W. Hou, "Maddpg-based security situational awareness for smart grid with intelligent edge," *Applied Sciences*, vol. 11, no. 7, p. 3101, 2021.

[7] Y. Zhu, H. Yao, T. Mai, W. He, N. Zhang, and M. Guizani, "Multiagent reinforcement-learning-aided service function chain deployment for internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 15 674–15 684, 2022.

[8] S. Guo and X. Zhao, "Multi-agent deep reinforcement learning based transmission latency minimization for delay-sensitive cognitive satellite-uav networks," *IEEE Transactions on Communications*, vol. 71, no. 1, pp. 131–144, 2022.

[9] J. Wei, Y. Wei, L. Yu, and R. Xu, "Radar anti-jamming decision-making method based on ddpg-maddpg algorithm," *Remote Sensing*, vol. 15, no. 16, p. 4046, 2023.

[10] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[11] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[12] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[13] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[14] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[15] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *International conference on machine learning*. PMLR, 2019, pp. 2961–2970.

[16] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 3053–3062.

[17] S. Hu, Y. Zhong, M. Gao, W. Wang, H. Dong, X. Liang, Z. Li, X. Chang, and Y. Yang, "Marllib: A scalable and efficient multi-agent reinforcement learning library," *Journal of Machine Learning Research*, 2023.

[18] S. Wiggins, Y. Meng, R. Kannan, and V. Prasanna, "Accelerating multi-agent ddpg on cpu-fpga heterogeneous platform," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2023, pp. 1–7.