CSCI 596 Course project
A parallel reinforcement learning framework

Kishore Ganesh, Qiongao Liu, Yusheng Jiao, Chenchen Huang, Haotian Hang

Instructor: Professor Aiichiro Nakano

Department of Aerospace and Mechanical Engineering,
University of Southern California, 854 Downey way, Los Angeles, California 90089, USA

May 20, 2020

1 Introduction

1.1 Motivation

Reinforcement learning has been widely used in robotics and control [15, 11, 6, 18, 8]. In recent years, it has also been introduced to solving fluid problems, such as flow control [16, 9, 12, 2, 3], soaring of birds [13, 14] and energy-efficient station keeping of balloon [1]. Recently, there are also several fascinating and important studies applied reinforcement learning technic in the field of fish swimming. In a potential flow, [7] used reinforcement learning to train a three-link fish to swim in a given direction efficiently. Using a Navier-Stokes solver combined with reinforcement learning algorithm, Koumoutsakos *et al.* have discovered the efficient strategy for fish's schooling behavior [4, 17] and C-start [10]. [5] found that using reinforcement learning, a low velocity swimmer can fulfill efficient point-to-point navigation in a Kármán vortex street.

Nowadays, parallel computing is extremely important in scientific computation. In using reinforcement learning to solve fluids problems, since evaluating each episode requires solving a an expensive fluid simulation, parallel computing is extremely important.

In this project, we are going to develop a parallel reinforcement learning framework which is extensible to different algorithms and supports multi-agent training.

1.2 General introduction of Reinforcement learning (RL)

1.3 Importance and challenges of parallel RL

2 Algorithm

In this project, we are mainly focused on Policy Proximal Optimization (PPO) algorithm, but the framework can support the implementation of any RL algorithms.

3 Code structure

3.1 Communication

Since we are trying to build a parallel framework running on multiple nodes with distributed memory, communication, namely MPI communication is most important thing we need to consider.

As shown in Figure 1, we are building the framework to allow communication between each nodes running the training environment and the nodes which running neural network training and inference. This communication is essential since the communication of observation and action is happening at each timestep of the updating of environment.

In Figure 1(a), on the left hind side, there are n sets of Nodes (I will explain later why there are sets of instead of n nodes) which are running environments. Generally speaking, we can run different environments on different sets of nodes in order to gather data for training, but commonly they are the same environment running on these n sets of nodes. These nodes will run the environments episode by episode

with randomized initialization as described in Sec 1.2. On the right hand side of Figure 1(a), there is a set of nodes running inference and training of neural network, and Figure 1(b) shows the communication between a set of environment nodes and the NN nodes.

As implied before, there can be also another two kinds of parallelization happening. However, we are not taking care of any of them based on the concept of "capsulation". Firstly, a single implementation of environment can be running on multiple nodes or GPUs. We are supposing that this should be implemented inside the environment and it need to has a single "master" used for communicate with the NN nodes. And what we need to do is to allocate the required nodes/GPUs to it. Secondly, since the training of neural network is costly, it also need to be run in parallel. And we are using the internal functionality of libtorch to do this.

For the Nodes 0 which holds the neural network, for on policy reinforcement learning algorithms, such as PPO, we don't let the update of the neural network happens simultaneously with the data gathering. Thus, there are two stages, one stage is the training of network, which can be done on GPU or multi-thread using the internal functionality of libtorch. The other stage is continuously receiving observation from the n environments, inferring action from it, and sending action to the environments. It also needs to organize and store the observation, action, reward in the memory, which will be used for training. For communication with different nodes simultaneously, we use a multi-thread Openmp session. Each thread are used to communicate and respond the observation-action request to each environment nodes.

The neural network should only be updated when all environments have reached the terminal state. For this, some synchronization is needed, which is achieved through the use of MPI_Barrier. One environment node has a barrier prior to sending the terminal observation and one after. Other environment nodes have two barriers after sending the terminal observation. The first barrier effectively results in one environment node only sending the terminal observation when all other environments have reached the terminal state. The second barrier prevents the other environments from progressing till the request of the environment node is acknowledged.

In addition, to enable this, asynchronous communication using MPI_Irecv, MPI_Test and MPI_Isend is used. Asynchronous communication is required because some nodes may not be ready to respond to a message (when they're waiting in the barrier).

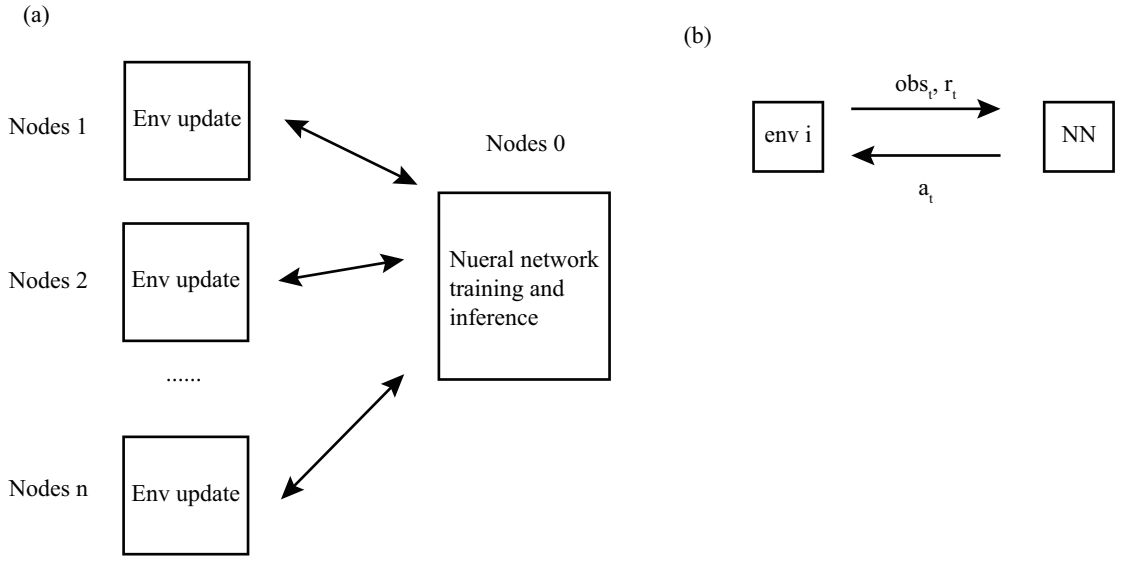

Figure 1: Illustration of general objective

4 Validation cases

5 Comparison

Using Proximal Policy Optimization (PPO) algorithm, on a same problem, train networks with four configurations, and compare their performance:

Activation function: tanh, sin

A separate actor and critic and combining actor and critic in a same neural network.

References

[1] Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

[2] Dixia Fan, Liu Yang, Zhicheng Wang, Michael S Triantafyllou, and George Em Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proceedings of the National Academy of Sciences*, 117(42):26091–26098, 2020.

[3] Paul Garnier, Jonathan Viquerat, Jean Rabault, Aurélien Larcher, Alexander Kuhnle, and Elie Hachem. A review on deep reinforcement learning for fluid mechanics. *Computers & Fluids*, 225:104973, 2021.

[4] Mattia Gazzola, Babak Hejazialhosseini, and Petros Koumoutsakos. Reinforcement learning and wavelet adapted vortex methods for simulations of self-propelled swimmers. *SIAM Journal on Scientific Computing*, 36(3):B622–B639, 2014.

[5] Peter Gunnarson, Ioannis Mandralis, Guido Novati, Petros Koumoutsakos, and John O Dabiri. Learning efficient navigation in vortical flow fields. *arXiv preprint arXiv:2102.10536*, 2021.

[6] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[7] Yusheng Jiao, Feng Ling, Sina Heydari, Nicolas Heess, Josh Merel, and Eva Kanso. Learning to swim in potential flow. *Phys. Rev. Fluids*, 6:050505, May 2021.

[8] Li-Ke Ma, Zeshi Yang, Tong Xin, Baining Guo, and KangKang Yin. Learning and exploring motor skills with spacetime bounds. *Computer Graphics Forum*, 40(2), 2021.

[9] Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(4):1–11, 2018.

[10] Ioannis Mandralis, Pascal Weber, Guido Novati, and Petros Koumoutsakos. Learning swimming escape patterns under energy constraints. *arXiv preprint arXiv:2105.00771*, 2021.

[11] Andrew Y Ng, H Jin Kim, Michael I Jordan, Shankar Sastry, and Shiv Ballianda. Autonomous helicopter flight via reinforcement learning. In *NIPS*, volume 16. Citeseer, 2003.

[12] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Réglade, and Nicolas Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of fluid mechanics*, 865:281–302, 2019.

[13] Gautam Reddy, Antonio Celani, Terrence J Sejnowski, and Massimo Vergassola. Learning to soar in turbulent environments. *Proceedings of the National Academy of Sciences*, 113(33):E4877–E4884, 2016.

[14] Gautam Reddy, Jerome Wong-Ng, Antonio Celani, Terrence J Sejnowski, and Massimo Vergassola. Glider soaring via reinforcement learning in the field. *Nature*, 562(7726):236–239, 2018.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[16] Russ Tedrake, Zack Jackowski, Rick Cory, John William Roberts, and Warren Hoburg. Learning to fly like a bird. In *14th International symposium on robotics research. Lucerne, Switzerland*. Citeseer, 2009.

[17] Siddhartha Verma, Guido Novati, and Petros Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23):5849–5854, 2018.

[18] Jie Xu, Tao Du, Michael Foshey, Beichen Li, Bo Zhu, Adriana Schulz, and Wojciech Matusik. Learning to fly: computational controller design for hybrid uavs with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.