**Fall 2023: ME759 Final Project Proposal**

**Quick intro:**

* Name: Aidan Rigby
* Email: acrigby@wisc.edu
* Home department: Nuclear Engineering and Engineering Physics
* Status: PhD Student.
* Name of your teammate: Cole Dunbar

**Project Title**: A Parallel Reinforcement Learning Algorithm for Model Based Control of Transient Processes.

**Link to git repo for project**: https://github.com/acrigby/ALPACA/tree/main/ME759

**Problem statement**: This project looks to improve the execution speed of an existing sequential deep Q-learning algorithm by employing parallel computing. The result will be parallel solution algorithm written using python that trains an agent to control a black box executable model based on the framework developed by OpenAI’s CartPole model from the Gymnasium library. The solution will run the model on multiple threads (runs currently performed sequentially) on the high-performance computer Euler thus improving learning speed dramatically. The work undertaken will also attempt to perform the neural network update process in parallel with the parallel model evaluations and therefore gain speed performance benefits this way.

**Motivation/Rationale**: Prior research between UW Madison and Idaho National Laboratory has shown that deep Q-learning algorithms can offer improvements in safety and economics of nuclear transient process controls by generating feedforward signals for PID controllers (1). The transient based modelling and solver tool Dymola has a library of nuclear energy and integrated energy system models which can be exported as a (license locked) C++ executable. The aim of this project is to demonstrate that the existing reinforcement learning solution utilized with Dymola could be parallelized on a HPC to reduce the current run time (currently approximately 3 days). This run time is prohibitive to explore the range of feedforward signals required to allow operation across the full operating regime of a nuclear power plant. To demonstrate that a speed up can be achieved the solver algorithm will be parallelized to control a “dummy” C++ executable model in place of the license locked Dymola model so that this implementation can be tested on Euler.

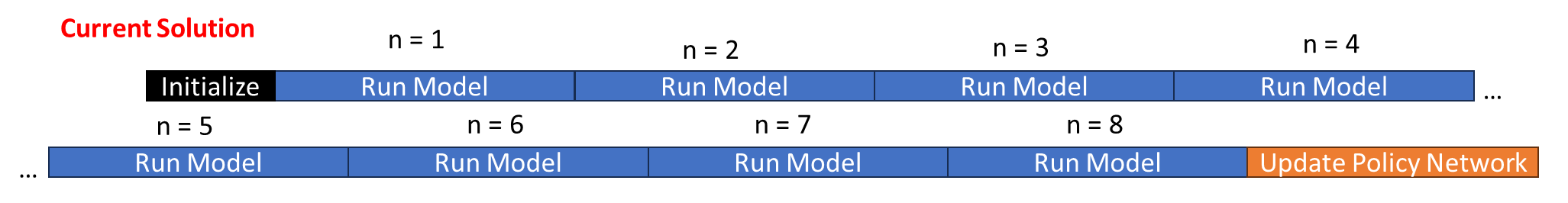
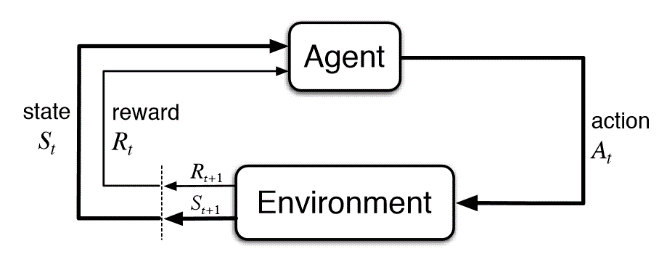
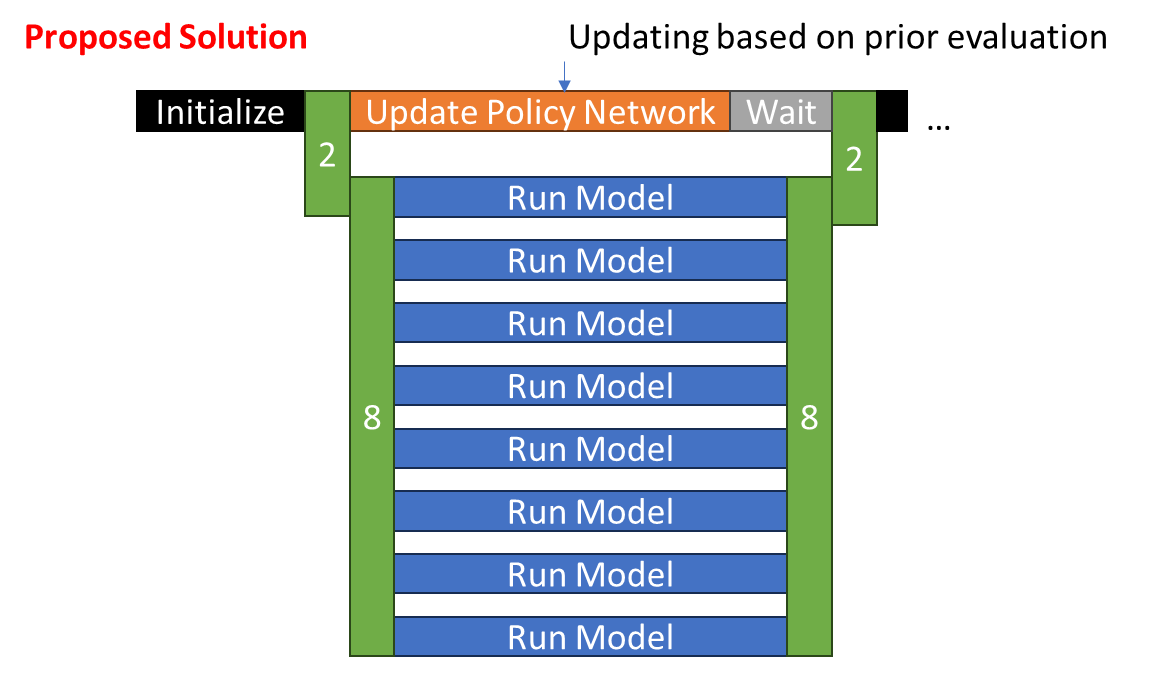
**Explain how you contemplate going about it**: To any RL solution there are two key processes: episodes (blue) and network updates (orange). Episodes consist of a series of environment evaluations based on the current policy network. In the algorithm investigated each episode generates around 100 combinations of action, states, and rewards with this process seen in Figure 1. The network update steps use these 100 combinations to update the policy network that is used to generate the agent’s actions. The current algorithm runs an episode and at every step pushes the output result to the memory. The update policy network runs after every 10 episodes and takes a random combination of these results to update the policy network. This is shown in time in figure 2, the reason for the long run time with the sequential algorithm is evident.

Figure - Reinforcement learning step methodology (https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292)

Figure - Sequential Code algorithm schematic

The new solution is shown in Figure 3 it would run up to 16 models (shown for 8 different CPU threads) in parallel and then push all the action, states, and rewards to memory afterwards. The aim would be to run the evaluations concurrently with the updating of the policy network based on the previous set of episode results. This would allow an even greater speed up of the solution algorithm.

Figure - Proposed Parallel solution algorithm schematic

**ME759 aspects the proposed work draws on**:

* The parallelization would be achieved using the OpenMP based PyOMP using the learning from ME759 on OpenMP in C to run with the python-based solution algorithm. This solution will focus on the use of OpenMP sections/tasks and drawn on important lessons in the course on updating shared memory spaces in parallel computing.
* There is also a possibility of the use of library-based GPU functions (primarily PyTorch) to speed up execution but because this does not involve us writing our own parallel code the emphasis will be on the previous step.

**Deliverables**: The project will deliver a detailed project report. This will begin with the motivation and summary of previous research. It will also lay out the details of the current solution algorithms with an evaluation of speed. The report will document the process of parallelizing the code and difficulties encountered. The main emphasis will be given to a study of speed up improvements made by sequentially parallelizing more of the code. The report will end with a discussion of how the parallel Q-learning algorithm will be extended to run with Dymola on the INL HPC Sawtooth, work that is intended to begin next year.

The project will deliver a docker image containing the model executable, python function and run scripts, python environment set up, input files and Slurm scripts for running the code. Provided will also be the original file so that the speed up results can be verified.

The code will also be pushed regularly to the GitHub repositories ME759 folder with the raw code and commit history presented alongside the docker image.

**How you will demonstrate what you accomplished**:

The scaling study of the sequential Q-learning algorithm running on the dummy executable compared with the multiple levels of parallelization achieved will hopefully demonstrate that speed up improvements can be achieved based on the algorithm architecture shown in figure 3.

**Other remarks**: (1) The current report documenting this has not yet cleared the Idaho National Lab review system and been posted on OSTI but a link will be provided to it in the final project.