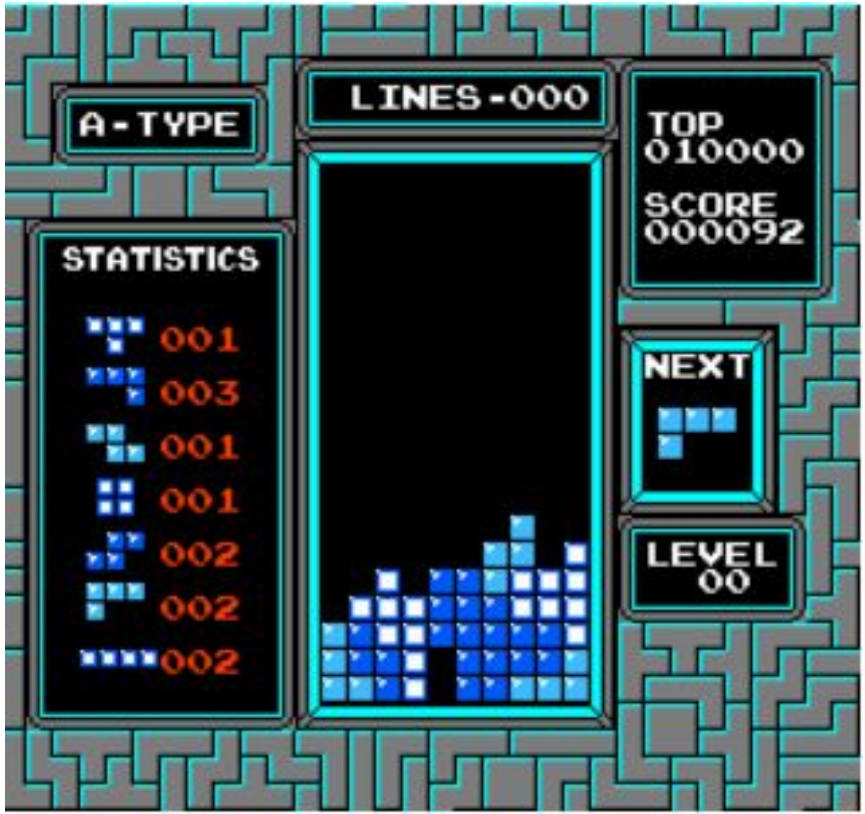
Parallelizing Deep RL

Alex Leonardi, Michael Fein, Christopher Gilmer-Hill, Owen Schafer

Task: Tetris

- Goal: Speed-up the training of our Deep RL Tetris-playing agent
  - Big Compute
  - Focus on weak scaling (Gustafson)

- gym-tetris: OpenAI Gym environment for Tetris
  - Gym environment is hosted on a server (Python script) that communicates with a client (C++)
  - Observation Space: 3 x 256 x 240 (Image)
  - Action Space: 6 (“Simple Joystick Movements”)



Source: https://pypi.org/project/gym-tetris/

Deep Reinforcement Learning Architecture

OpenAI Proximal Policy Optimization (PPO)

- PPO schema uses a clipped objective function to search over a trust region
- Simplifies the typical reinforcement learning by removing the KL penalty from the objective function
- Uses an estimated advantage (A) weighted by the ratio of probabilities, r, of achieving certain actions under different policies (parameterized by some theta)

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\ ]$$

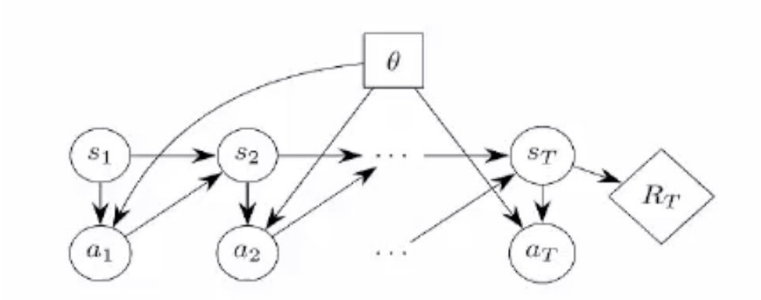
- θ is the policy parameter
- $\hat{E}_t$ denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively
- $\hat{A}_t$is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2

Source: Schulman, Et al., OpenAI, Abbeel, Et Al.

Deep Reinforcement Learning Architecture

Convolutional NN Observations

- CNN steps over the resulting observation space to send the state back to the optimization algorithm with total trainable weights of 1,548,352
- Messages are sent through ZeroMQ library between python gym server and C++ pytorch implementation



Source: Schulman, Et al., OpenAI, Abbeel, Et Al.

Novel Integrations

OpenAI Gym is built for Python, so we combined a PyTorch C++ framework with Atari-style OpenAI Gym Environments by:

- Editing gym environment server implementations
- Restructuring convolutional layers to match Tetris game input/reward output
- Debugging ZMQ to communicate between new tetris gym server and pytorch

Novel Documentation

The process of setting up and running a gym environment using C++ is not well documented, so we noted all steps necessary to set up a C++ RL environment in Ubuntu 18.04:

- Installing dependencies
- Building executable files
- Launching the gym server
- Training the agent
- Implementing PyTorch parallelism

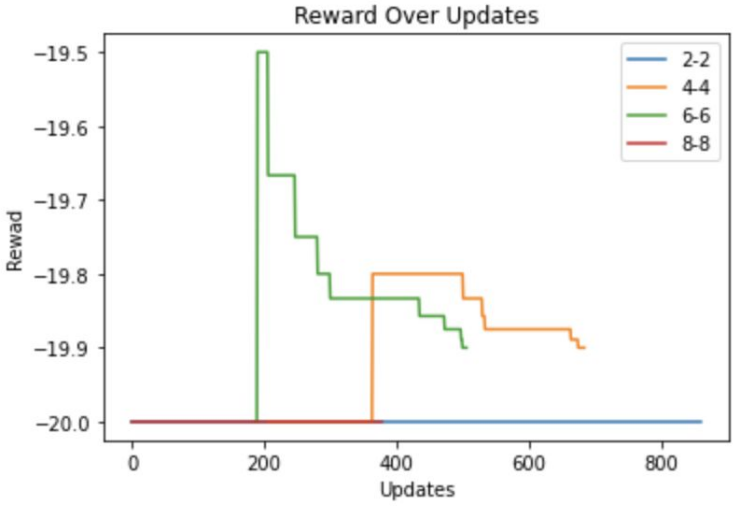
Parallelization Scheme: CPU (OpenMP) / GPU (CUDA)

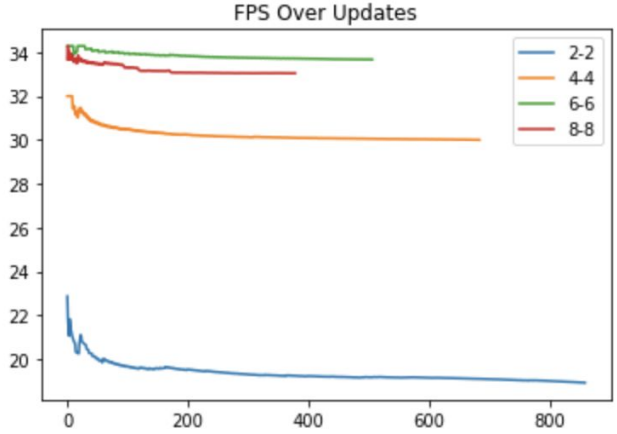
- OpenMP Usage:
  - Shared-Memory model
  - at::set_num_threads and control of num_envs allows access to multiple parallel OpenMP threads and Gym environments
  - Procedure and Loop-Level parallelism
- CUDA Usage:
  - GPU-Accelerated computing model
  - const bool use_cuda = true;
  - Loop- and Instruction-level parallelism
- Attempted MPI Usage:
  - CMake currently fails to correctly link the .cpp executable for ppo.cpp to ompi.h; however, the code should be otherwise correct in terms of MPI implementation

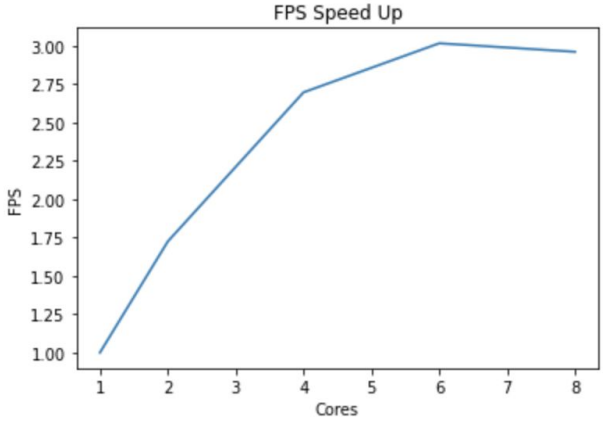
```
int main(int argc, char *argv[])
{
    spdlog::set_level(spdlog::level::debug);
    spdlog::set_pattern("%^[%T %7l] %v%$");

    at::set_num_threads(8);
    torch::manual_seed(0);
```

Results: CPU

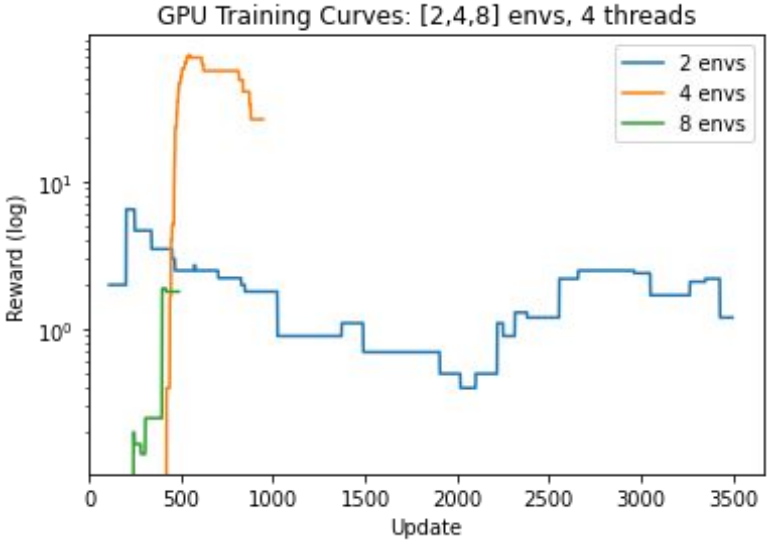
Labels are formatted as follows:
“num_cores-num_envs”

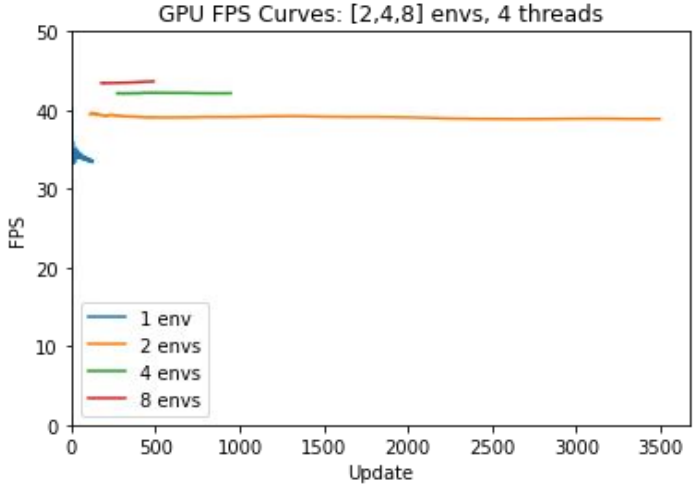


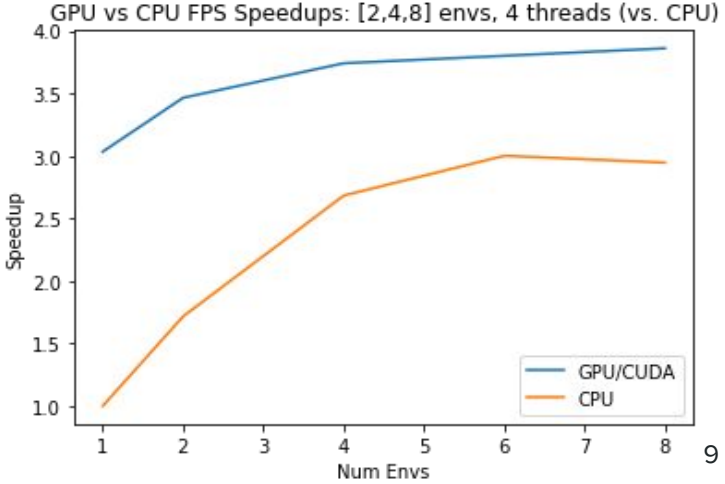




Results: GPU/CUDA







Future Work: MPI

- Potential combination of SMPP and DMPP across different parts of the problem
- (Currently non-functional due to CMake version compatibility/library linking issues; however, the file ppo_mpi.cpp contains the code for the below implementation scheme)
- MPI Implementation Scheme:
  - Partition batch of actions across MPI_COMM_WORLD nodes
  - Calculate gradient for each partition w/ loss.backward()
  - MPI_ALLreduce to average gradient across all nodes
  - Optimizer.step() to update NN based on *average* gradient

Partition Observation

ALLreduce gradient

Thank you!