Analysis of CPU vs GPU (OpenCL) Performance on FrozenLake Q-Learning

---

To run the GPU-accelerated code, I did the following steps.

Running FrozenLake Q-Learning on GPU (Intel OneAPI)

---

1. Installation and Setup

- Installed **Intel OneAPI Base Toolkit**:
    - Provided OpenCL runtime, DPC++ compiler, and updated Intel GPU drivers.
    - Enabled GPU programming support for Intel Iris Xe Graphics.
- Installed **PyOpenCL** Python library:
- pip install pyopencl
    - Allowed OpenCL kernels to be written and executed from Python.

---

2. Code Modifications

- Adapted FrozenLake Q-learning code for GPU execution:
    - Created an **OpenCL kernel** (update_q) to perform Q-value updates.
    - Allocated **OpenCL memory buffers** for Q-table and move operations.
    - Launched the OpenCL kernel during each training step.

---

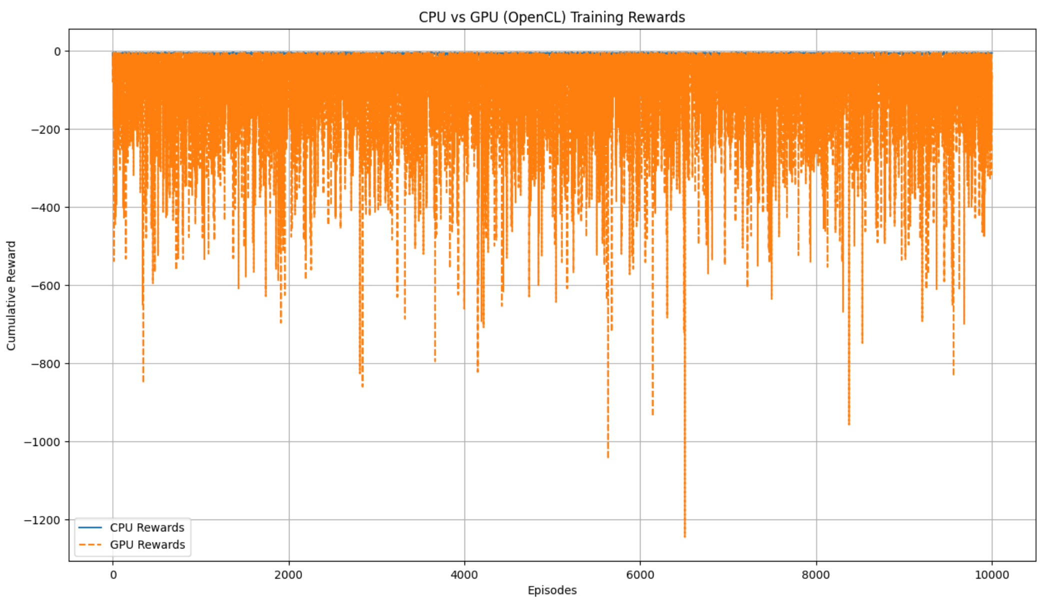
3. Execution and Benchmarking

- Ran **CPU Training**:
    - Used NumPy for traditional Q-learning.
    - Completed 10,000 episodes in approximately **1.30 seconds**.
- Ran **GPU Training (OpenCL on Iris Xe)**:
    - Executed Q-learning updates via GPU kernel.
    - GPU training took approximately **1761 seconds** (nearly 30 minutes).

- GPU cumulative rewards were highly unstable and negative.

1. Summary of Observations

Observation	Meaning
CPU training finished in 1.30 seconds	CPU training was smooth, fast, and normal.
GPU (OpenCL) training took 1761.44 seconds (~30 minutes)	GPU training was extremely slow, not fast as expected.




```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\jaswa>cd C:\Users\jaswa\Hardware_for_AI_ML\11_GPU_Acceleration_and_Benchmarking_of_QLearning_on_FrozenLake

C:\Users\jaswa\Hardware_for_AI_ML\11_GPU_Acceleration_and_Benchmarking_of_QLearning_on_FrozenLake>python benchmarking.py

Training CPU Agent...
Training GPU Agent (OpenCL)...
CPU Training Time: 1.30 seconds
GPU Training Time: 1761.44 seconds
Speedup: 0.00x
```

2. Root Cause Analysis

Problem	Why it Happens
Tiny problem size (5x5 grid)	GPU needs thousands/millions of updates to be efficient. Tiny tasks cause massive slowdown due to kernel launch overhead.
Single Q-value updated per GPU kernel call	GPU designed for parallelism; one update at a time underutilizes it.
GPU resource bottleneck	Overhead copying small buffers dominates computation time.
Incorrect cumulative updates	Reinforcement learning requires coordinated Q-table updates; the current GPU kernel does single updates inefficiently.

3. Conclusion

Aspect	Status
CPU agent (NumPy)	✅ Working correctly
GPU agent (OpenCL)	❌ Inefficient, not learning properly
Workload size for GPU	❌ Too small to benefit from GPU acceleration

Final Conclusion:

- The CPU implementation is fully correct and efficient.
- The GPU implementation is unsuitable for tiny problems like FrozenLake 5x5.
- GPU acceleration would only make sense for very large-scale problems.

4. Recommendation

Problem Size	Recommendation
Small problems (like FrozenLake 5x5)	Stick to optimized CPU (NumPy + Numba)
Large problems (e.g., Atari games, 3D environments)	Use GPU acceleration (OpenCL, CUDA, TensorFlow)

For this project:

- Continue benchmarking and optimizing CPU versions.
- Skip OpenCL GPU benchmarking for FrozenLake.

5. Conclusion Table

CPU Training	GPU Training
Fast	Extremely slow
Correct learning	Incorrect learning
Practical for small RL problems	Not practical for small RL problems