Parallel Reinforcement Learning
Final Presentation

team07: Sam DePaolo, Michael Kielstra, Manqing Liu, Xiaohan Wu

Refresher: Reinforcement Learning

- At each time t, the agent receives current **state** S_t and **reward** R_t
- The agent then choose **action** A_t
- The action is sent to the environment , which moves to a new state S_{t+1} and reward R_{t+1}
- Goal is to learn a **policy** π which maximizes the cumulative reward

$$\pi(a, s) = Pr\left(A_t = a \mid S_t = s\right)$$

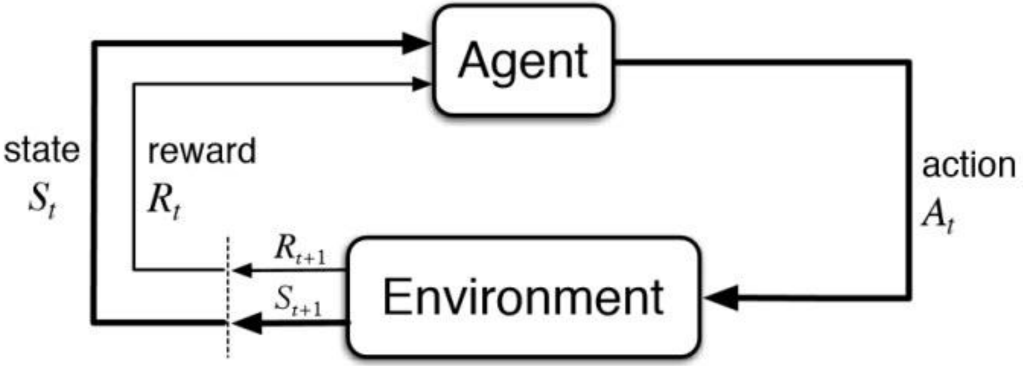


Image credit: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html
Reference: https://en.wikipedia.org/wiki/Reinforcement_learning

Refresher: Bandits

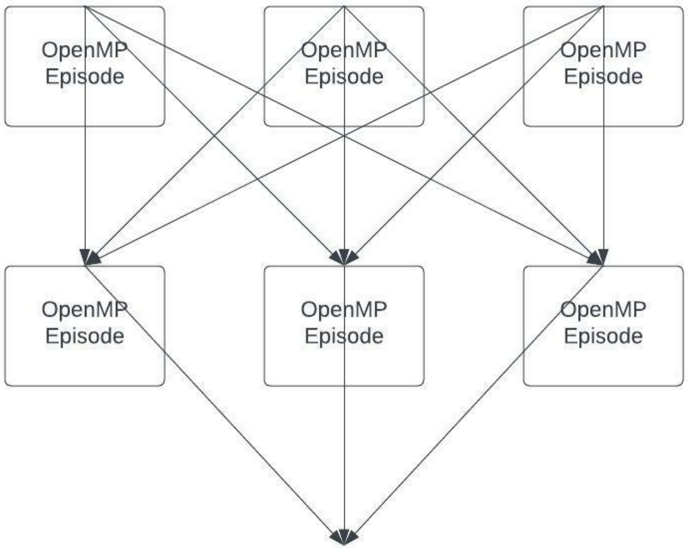
- A *bandit* is a Reinforcement Learning environment without any state at all
- We provide N=10 actions (“arms”) on our bandit
- Each reward is “mangled” with a repeated hyperbolic tangent function to avoid it following a simple normal distribution

Profiling analysis

Name of function called	% of time	Cumulative seconds	Self seconds
Mangle() function in Bandit::take_action()	95.00	9.79	9.79
Random number generator	4.95	10.31	0.51
normal_distribution()	0.19	10.33	0.02

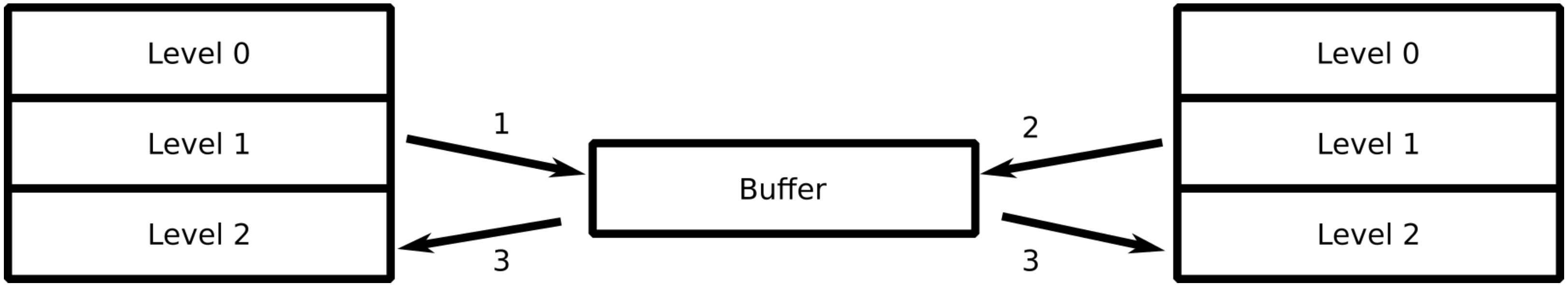
Parallel Agents

- 32 agents/core with OpenMP
  - OpenMP agents perform blocking synchronization per-episode
- Cores synchronize with MPI
  - Non-blocking reduction before episode
  - Local synchronization after episode

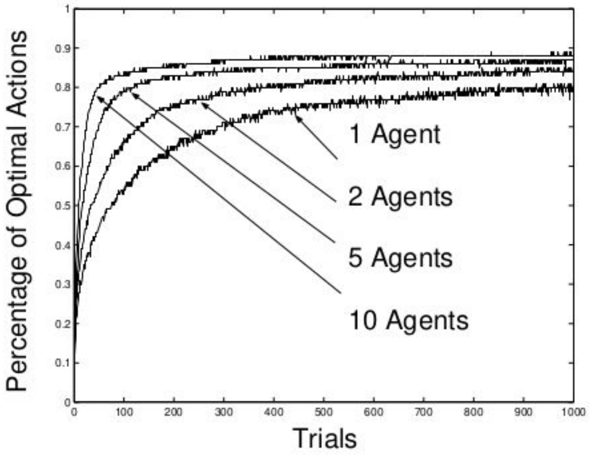


Keeping Synchronization Levels Apart

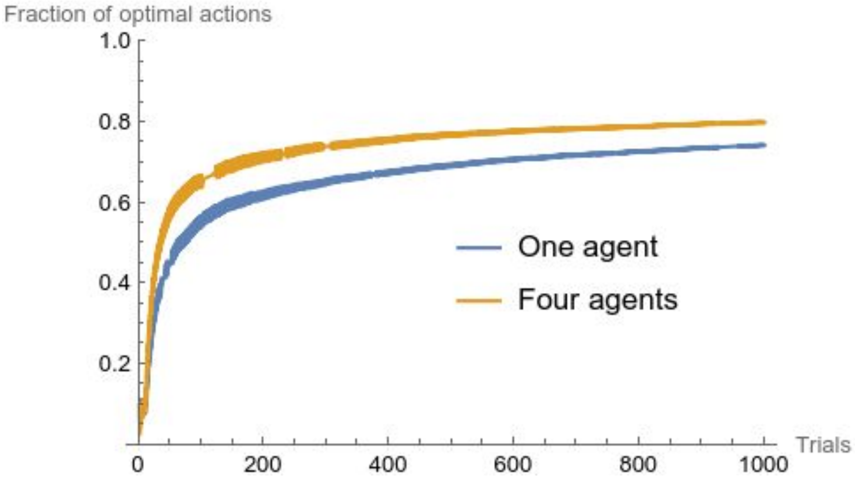
- All data stored as totals, not averages, so it combines additively
- Level 0 stores data from current episode
- Level i stores data from level i - 1 that has completed some synchronization process



Validation



Percentage of Optimal Actions
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
100
200
300
400
500
600
700
800
900
1000
Trials
1 Agent
2 Agents
5 Agents
10 Agents

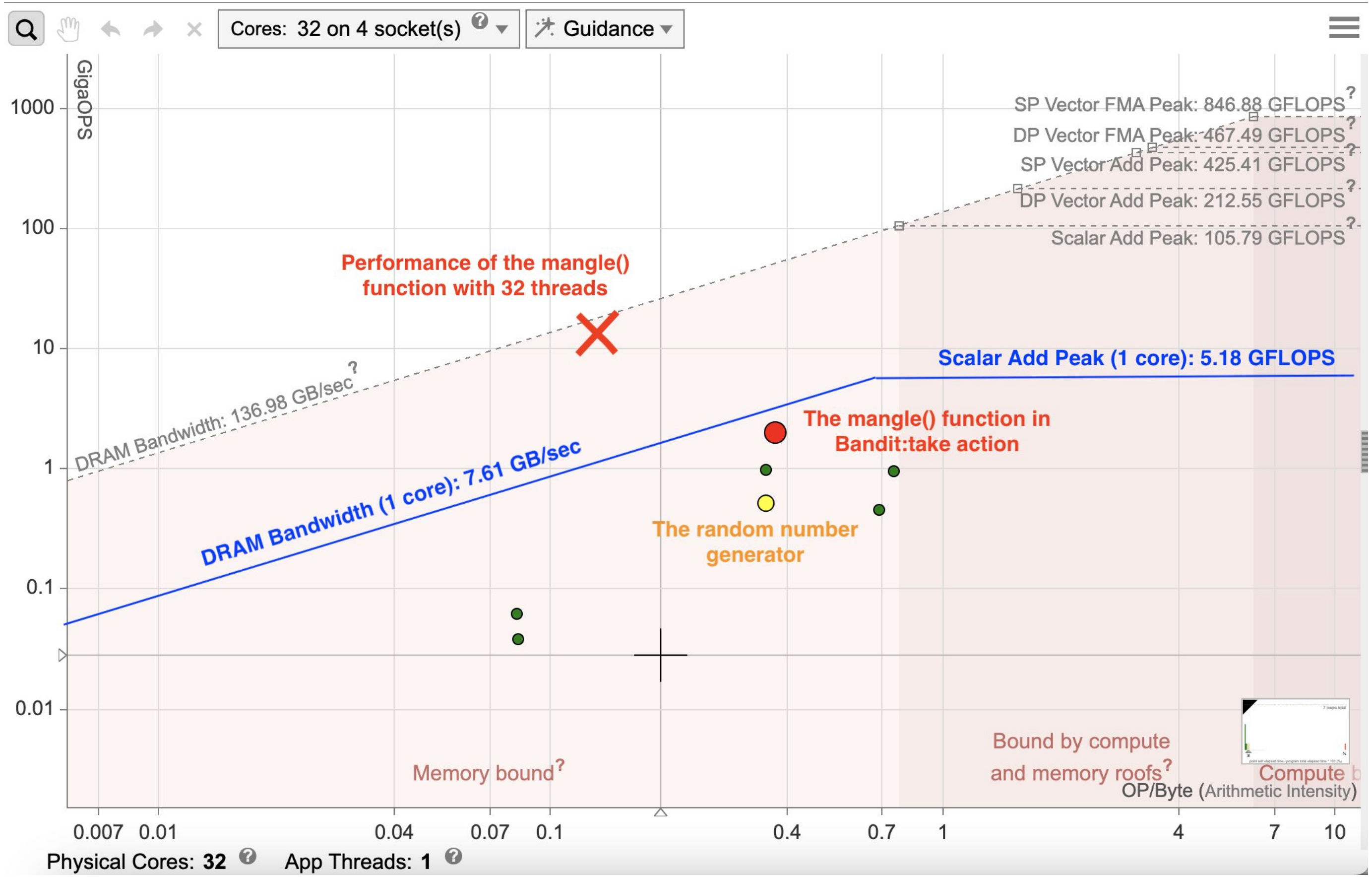


Fraction of optimal actions
1.0
0.8
0.6
0.4
0.2
0
200
400
600
800
1000
Trials
One agent
Four agents

Roofline

Sequential code roofline analysis performed with Intel Advisor:

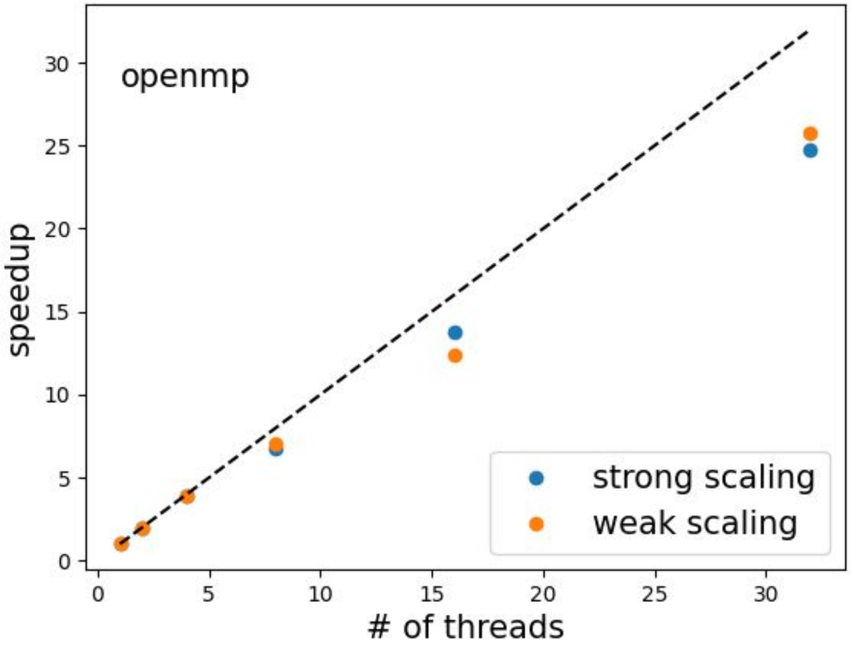
Justifies parallelization



Scaling

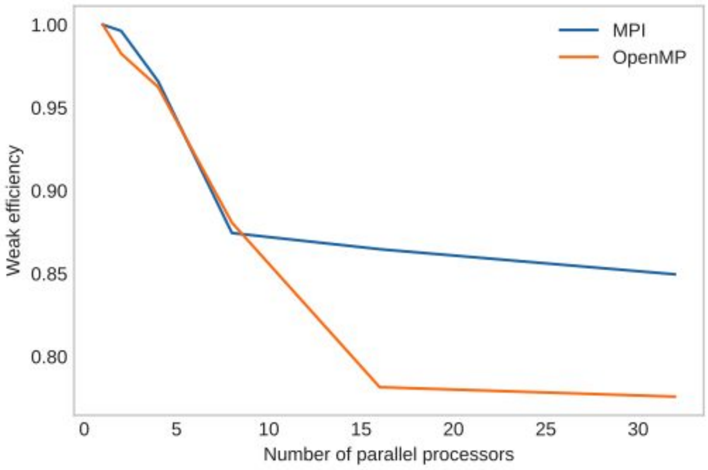
Strong and weak scaling analysis performed for both OpenMP and hybrid MPI-OpenMP code;

Hybrid code leads to very similar good scaling results



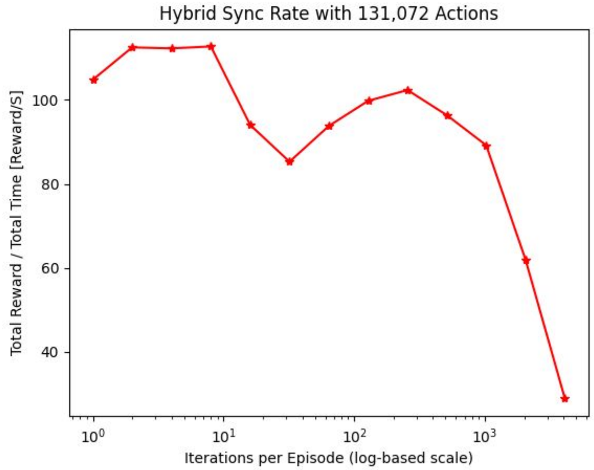
Weak Efficiency

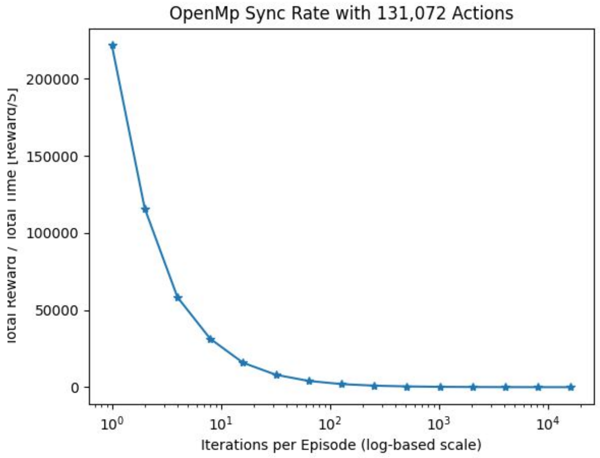
- Efficiency improves for MPI vs. OpenMP
- Reasons for improvement
    - Synchronizing between OpenMP threads requires more blocking
    - MPI can be done in a non-blocking way
    - Less blocking synchronizing and more non-blocking in hybrid code



Optimal Sync Frequency

- More synchronization means faster learning and higher total reward but also slower time due to communication between agents
- By looking at reward per second with a fixed action size we can determine the optimal frequency of communication (length of episode)
- Both the hybrid and OpenMP versions favor frequent communication





Discussion/Takeaways

- Parallelization in RL holds promise and should be further explored
- Hybrid parallelization for RL is feasible and effective
- Latency can be well hidden and frequent synchronization favored
- There is further potential to explore more complicated RL models with an increased number of arms