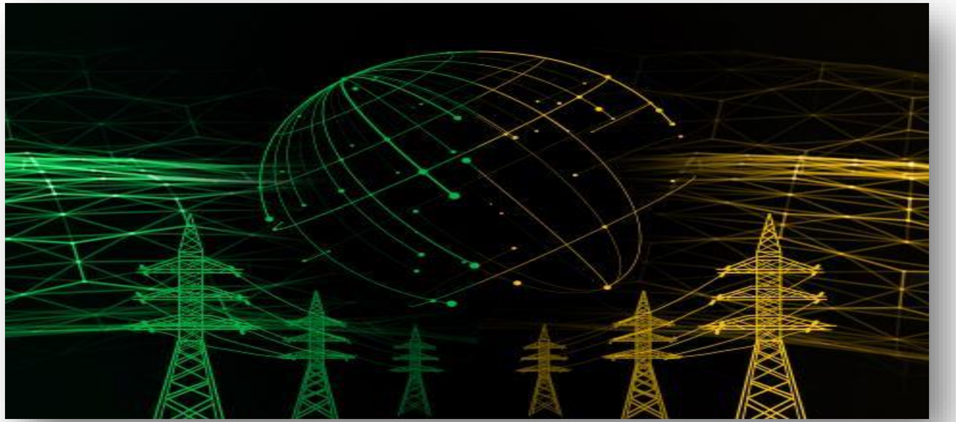
GMI-DRL: Empowering Multi-GPU DRL with Adaptive-Grained Parallelism

Yuke Wang
Assistant Professor
Rice University

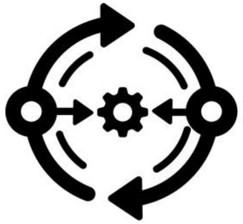
DRL is everywhere…



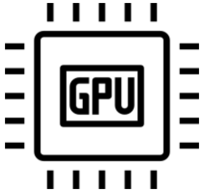
Credit: Google Image

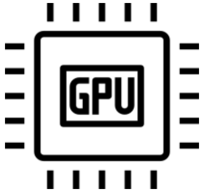
Background and Motivation

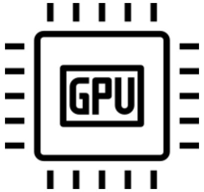
- DRL basics

- GPU-based DRL



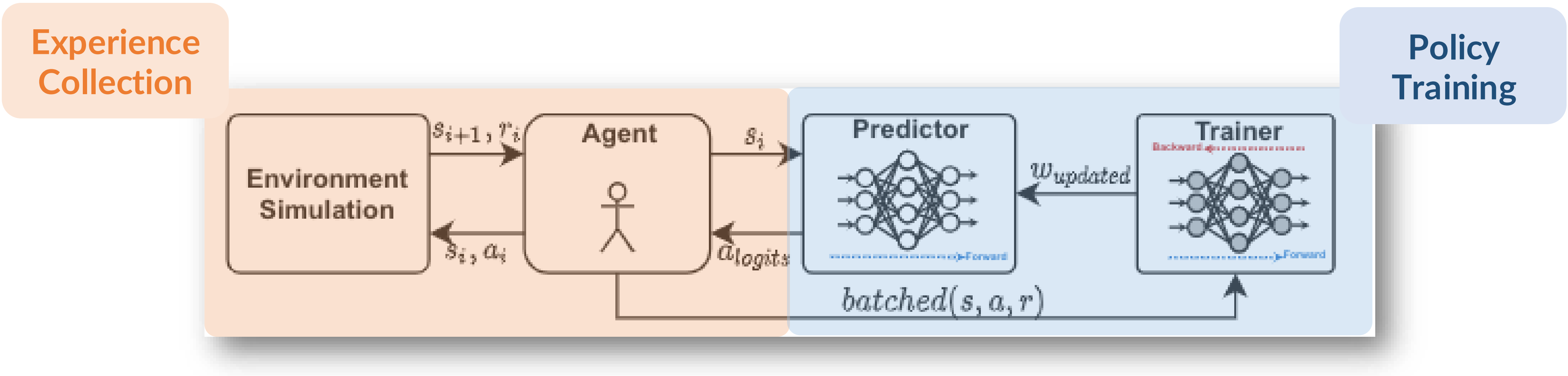




- GPU Spatial Multiplexing

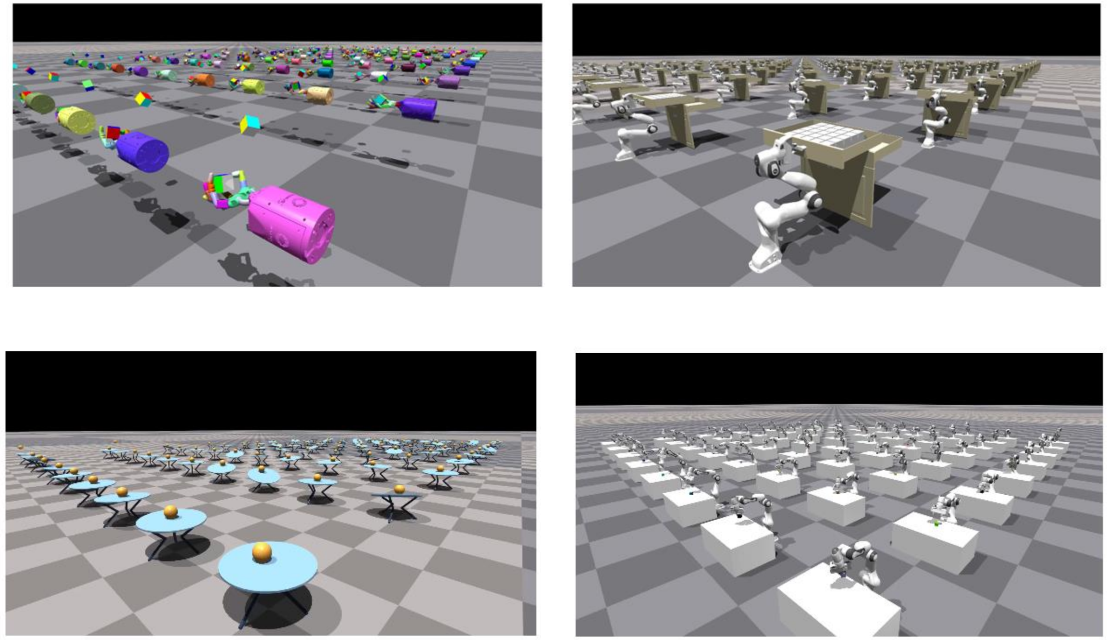

DRL Basics

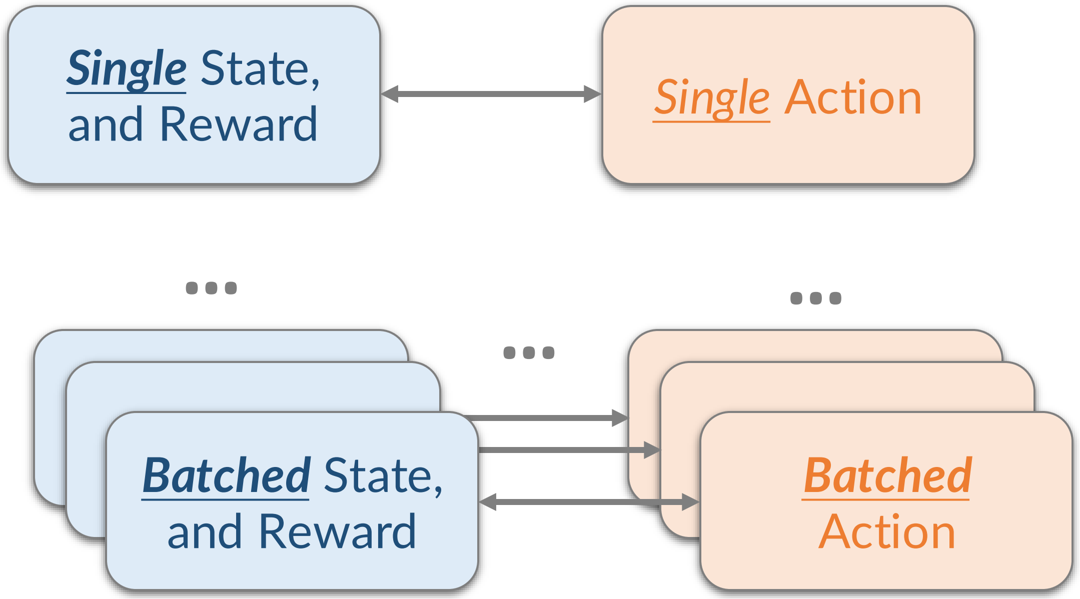
- Basic DRL Computation Flow.



Credict: NVIDIA Isaac Gy

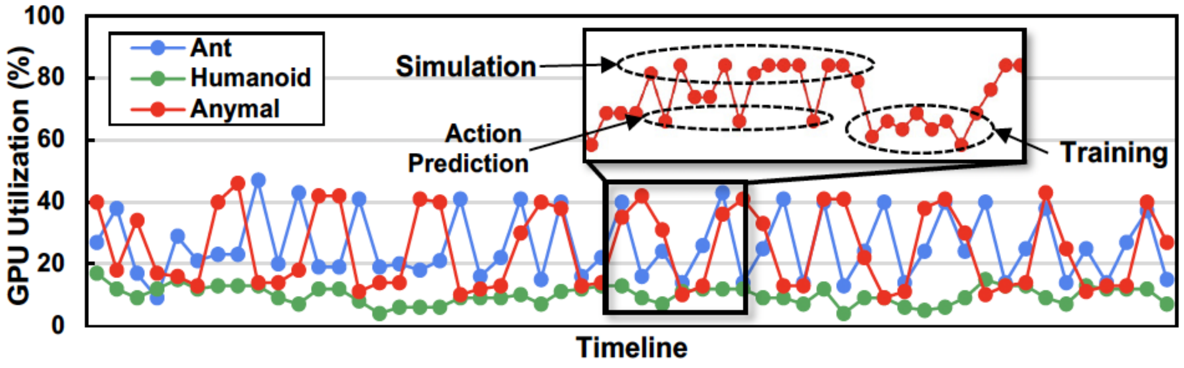
GPU-based DRL (Isaac Gym)

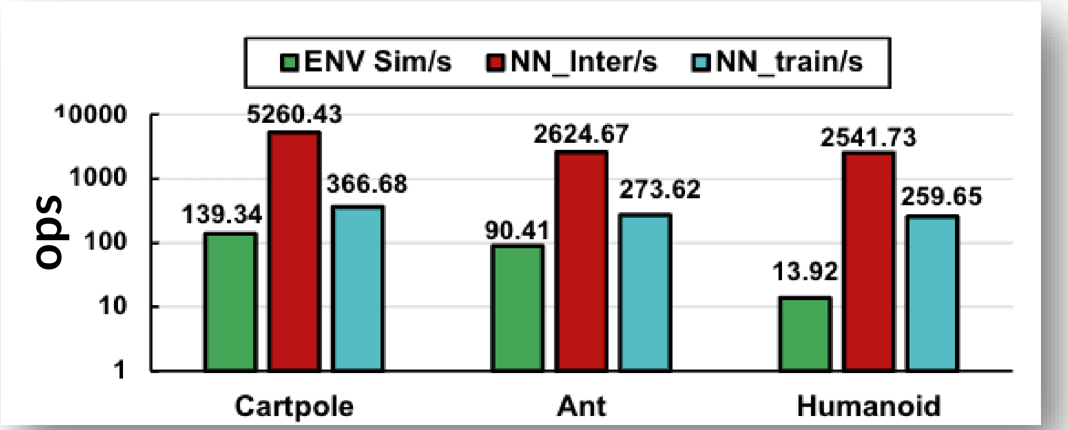




Thousands of environments to run in parallel on a single GPU

Observations of GPU-based DRL

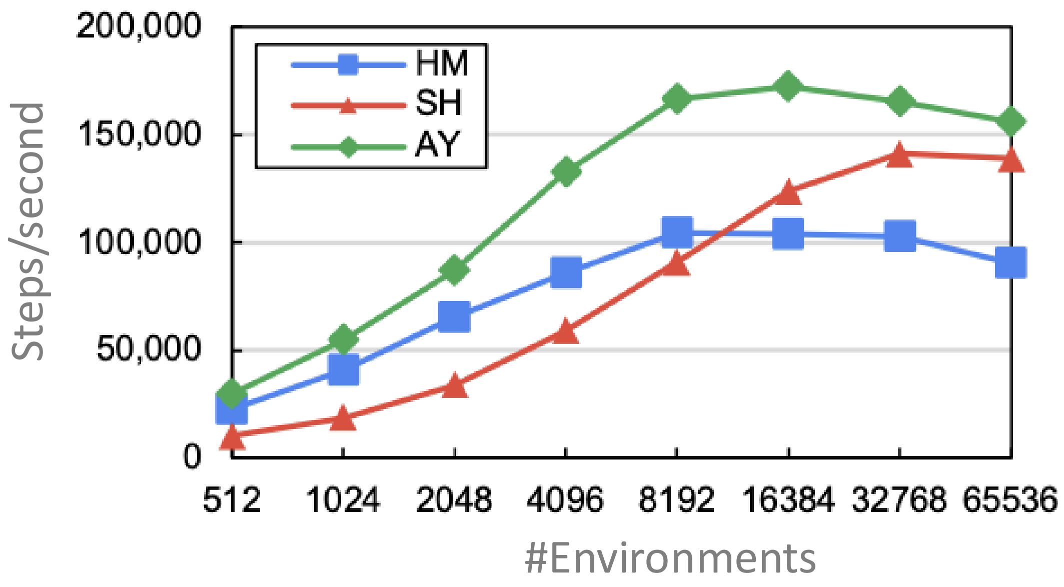


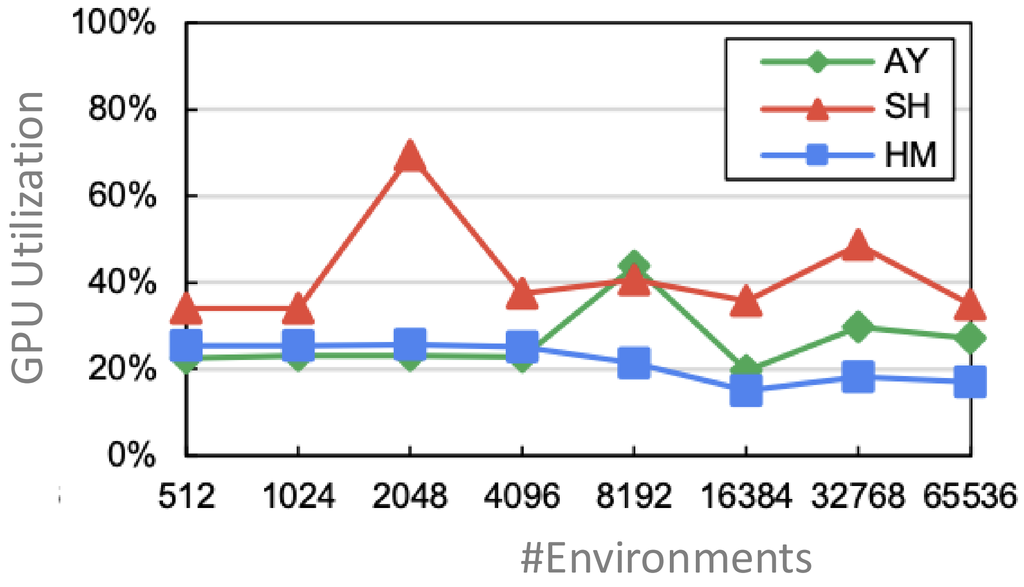


Cyclic GPU Utilization Pattern!

Simulation is the bottleneck!!

Observations of GPU-based DRL



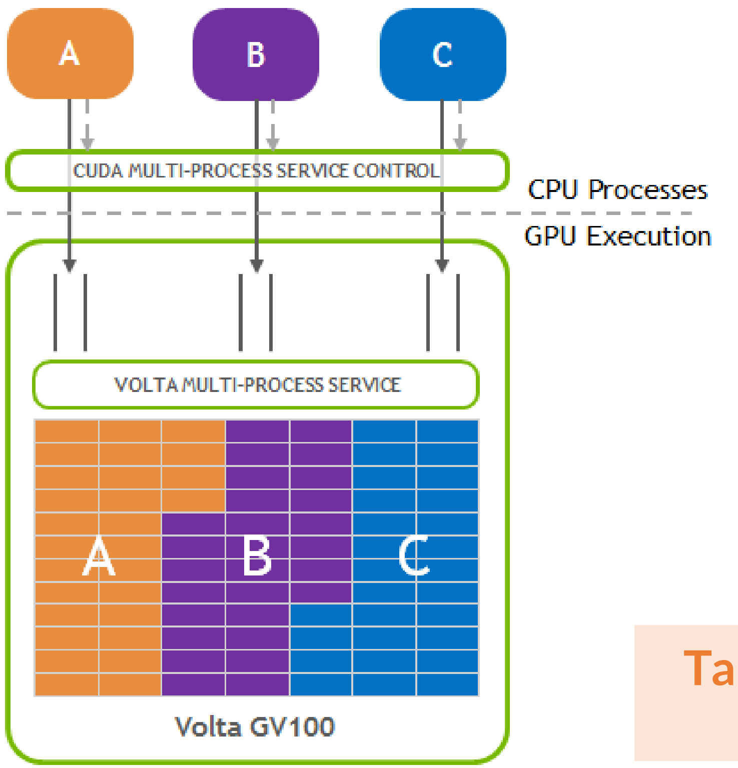


Large simulation batch could hardly work!

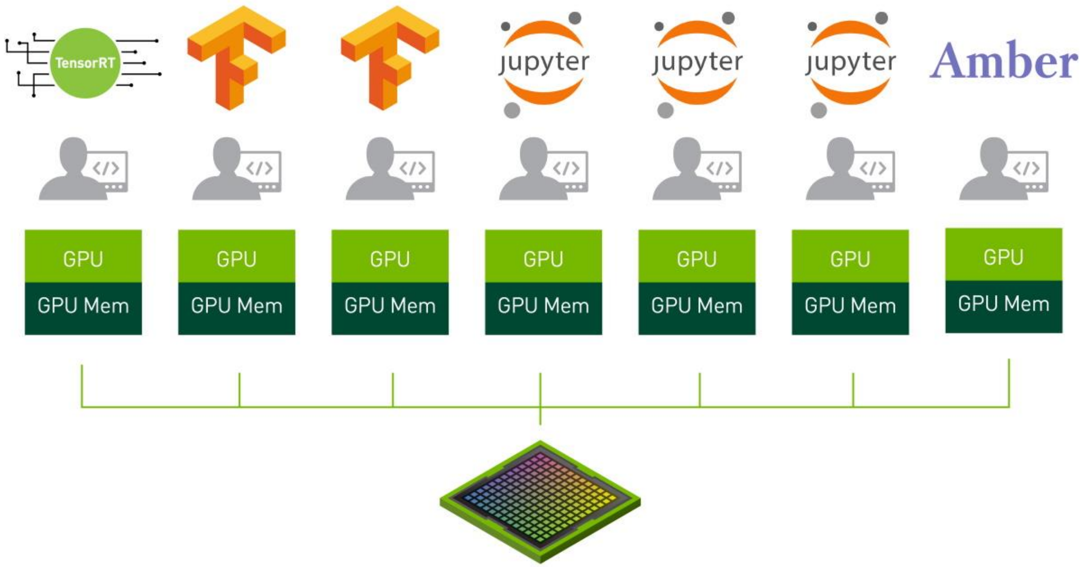
GPU utilization is low!

Adaptive-Grained Parallelism (AGP)

- **NVIDIA Multi-Processing Service (MPS).**

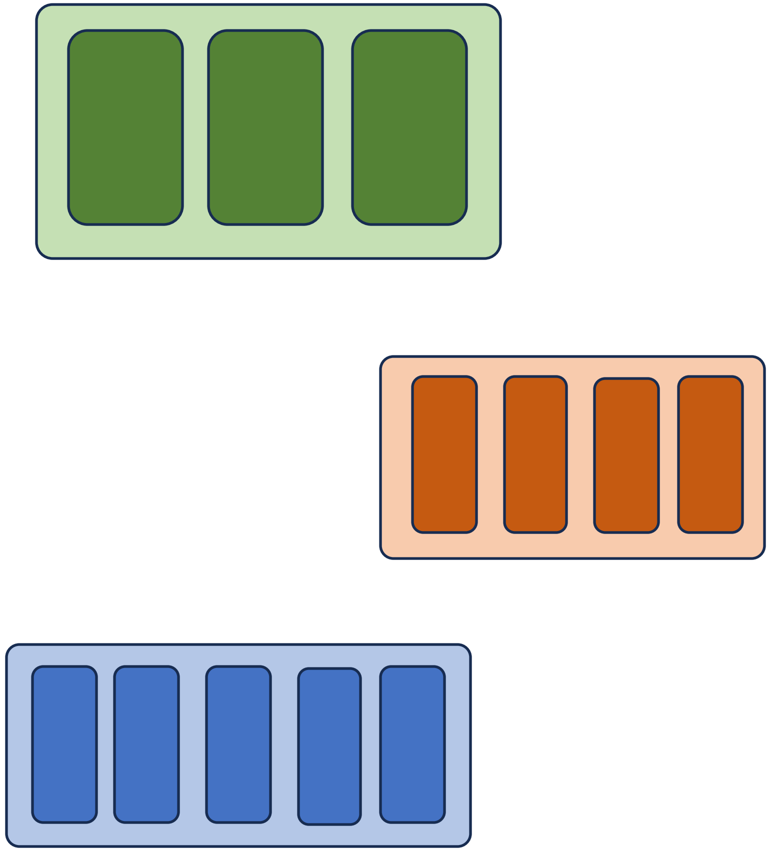


- **NVIDIA Multi-Instance GPU (MIG).**

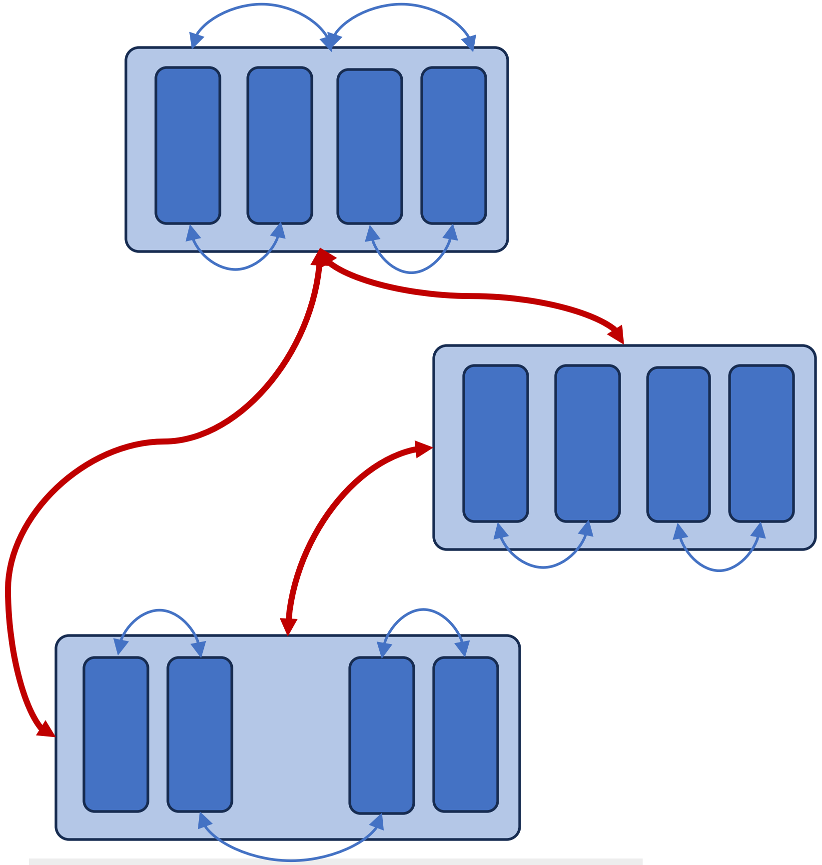


Tailor the Hardware Resources to fit the computation ! !

Challenges of AGP



How to determine the granularity of the instance?



How to connect different instances?

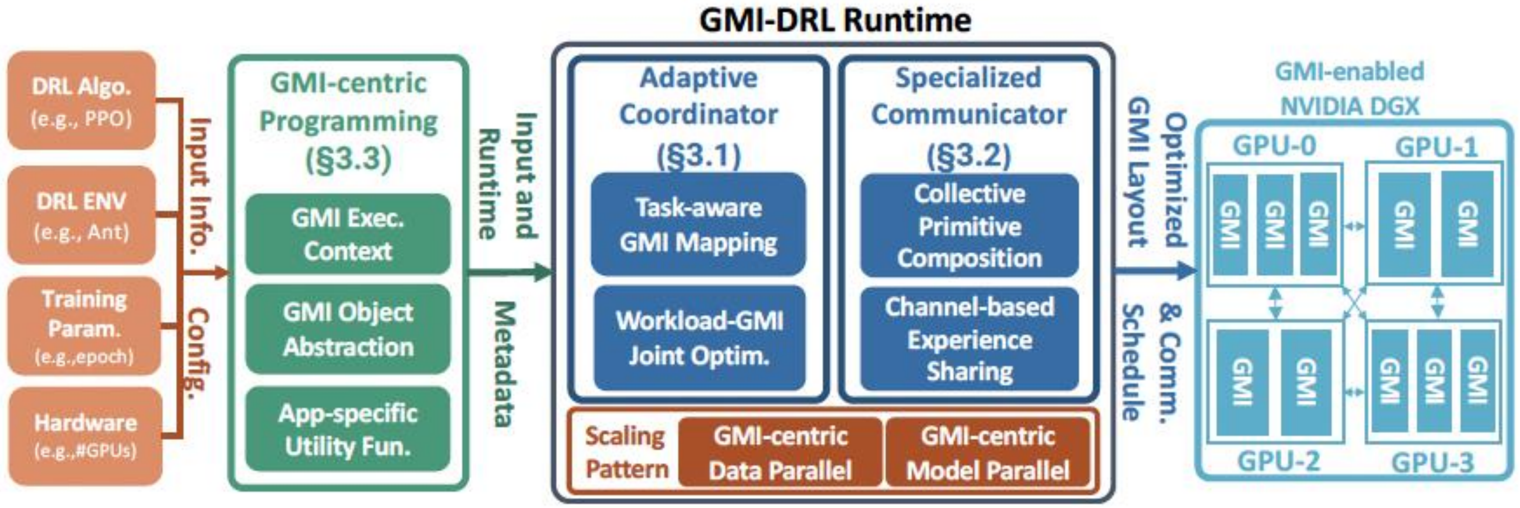
GMI-DRL Overview

- Process-based GMI programming.
- Adaptive GMI management.
- Specialized GMI communication.

Bridging users and GMIs

1) Matching RL tasks and GMIs
2) Mapping GMIs and GPUs

Connecting input/output among different GMIs



Process-based GMI Programming.

Listing 1: Example of GMI-based Programming.

```
import GMI_RL
import os
# import other packages ...
class RL_role(object):
    # Initize the base environment.
    def __init__(self, GMI_id, role, dev_id):
        self.GMI_id = GMI_id
        self.role = role
        self.GMI_mgr = GMI_RL.GMI_manager.add_GMI(GMI_id)
        self.GMI_mgr.set_GPU(dev_id)
        self.group = GMI_manager.get_group(GMI_id)
        # import other packages (e.g., pytorch)...
    def GMI_run(self, param1, param2, ...):
        while True:
            # major routine of send/receive data
            # or task processing, such as ENV,
            # AGENT and Trainer.
    def GMI_collective(self, data):
        # some data processing work ...
        proc_data = proc_fun(data)
        # allreduce data within a group of GMIs.
         self.GMI_mgr.allreduce(proc_data, self.group)
    def GMI_send(self, data, dst_GMI_id):
        # some data processing work ...
        proc_data = proc_fun(data)
        # Asynchronized send data to another GMI.
        self.GMI_mgr.send(proc_data, dst_GMI_id)
    def GMI_recv(self, src_GMI_id):
        # Synchronized receive data from another GMI.
        data = self.GMI_mgr.recv(src_GMI_id)
```

RL Simulator: Generate the environment states/observations for action prediction on agent and rewards for training.

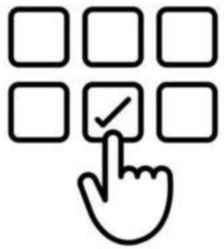
RL Agent: Make an action decision based on the environment states and observations.

RL Trainer: Update the policy and value NN model based on the collected experience data from RL agent.

Adaptive GMI Management

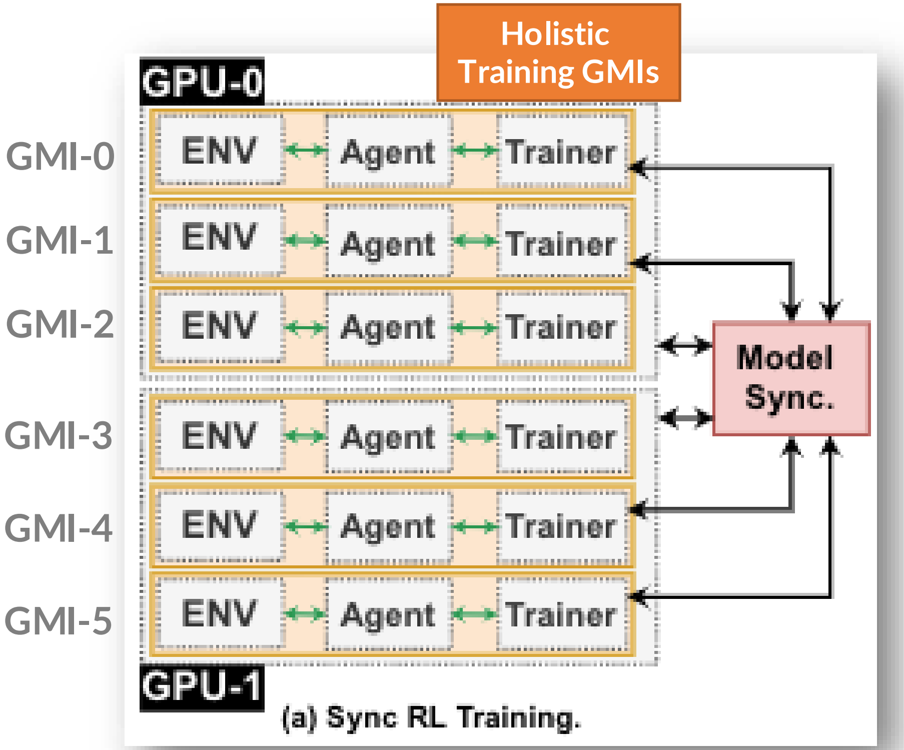
- Resource-aware GMI Mapping
- Workload-aware GMI Selection

1. Mapping of GMIs to GPUs.
2. Communication among GMIs.

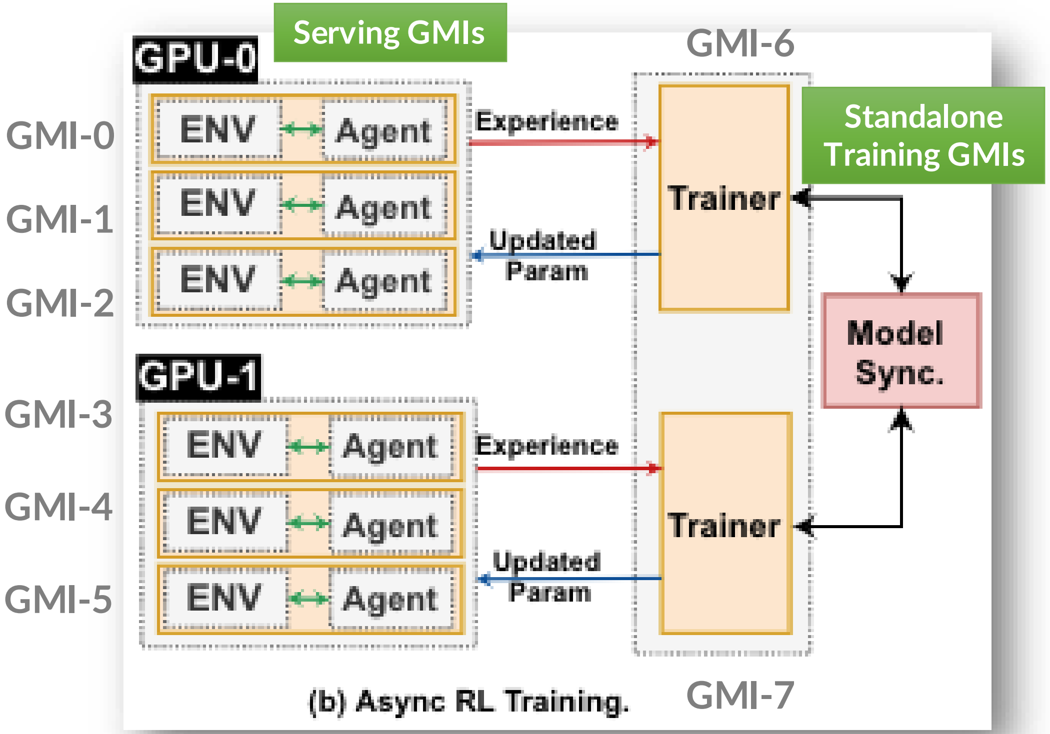


1. GMI-Resource (SM, Mem, etc.)
2. Runtime Performance

Resource-aware GMI Mapping



Co-locate ENV, AGENT, and Trainer on the same GMI



Co-locate ENV and AGENT on the same GMI while Trainer on other GMI

How many Serving/Training GMIs should be placed on each GPU?

Workload-aware GMI Selection

- **First,** performance profile of RL tasks on GMIs.
  - Accommodating Feasibility.
  - Performance Discount.
- Second, early stop detection with saturated performance.

Algorithm 1: Profiling-based GMI Exploration.

```
input  : DRL_bench, num_GPU
output : num_env, GMIperGPU
best_config = tuple(); max_top = −inf;
for GMIperGPU in 10 ... 1 do
    pre_top = 0; pre_mem = 0;
    for num_env in [128, 256, 512, ..., 16384, 32768] do
        ▷ Filter out the GMI OOM/crashing cases..
        if num_env>=512 && projMem(num_env) >
          GPU_mem/GMIperGPU then
            break;
        end
        ▷ Profile the performance of a GMI..
        top, mem = profile(DRL_bench, GMIperGPU, num_env);
        ▷ Initialize tracking variables..
        if pre_top == pre_mem == 0 then
            pre_top = top; pre_mem = mem;
            continue;
        end
        ▷ Compute performance/resource changes..
        R_top = (top − pre_top)/(pre_top);
        R_mem = (mem − pre_mem)/(pre_mem);
        Sat = R_top/R_mem;
        pre_top = top; pre_mem = mem;
        ▷ Check if the performance saturates..
        if Sat < α then
            break;
        end
        ▷ Project the overall system throughput..
        acc_top = estimate(GMIperGPU, num_GPU, top);
        if acc_top > max_top then
            max_top = acc_top;
            best_config = (num_env, GMIperGPU);
        end
    end
end
num_env, GMIperGPU = best_config[0], best_config[1];
```

Specialized GMI Communication.

- **[Sync]** Hierarchical Gradient Reduction.

Latency Driven

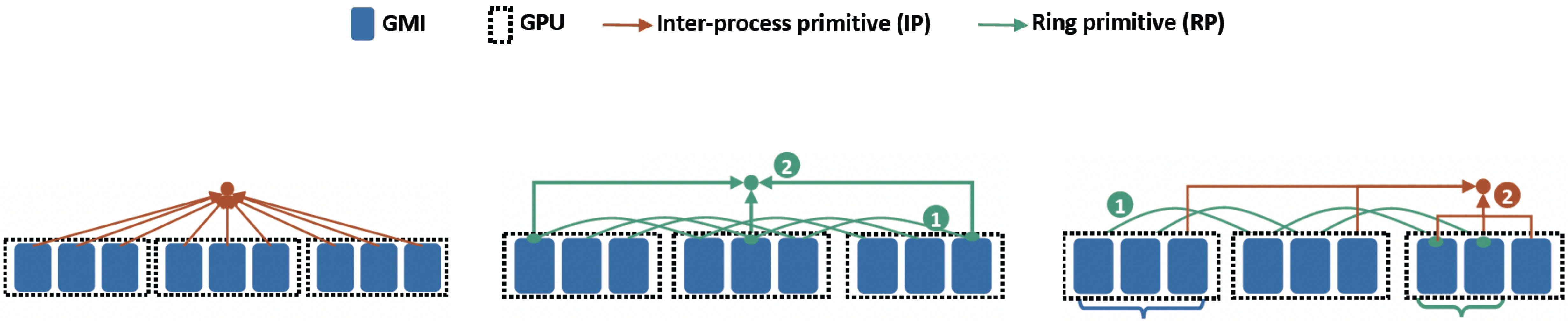
1) Intra-GPU inter-GMI reduce
2) Inter-GPU inter-GMI reduce

- **[ASync]** Channel-based Experience Sharing

Throughput Driven

1) Experience Decomposition
2) Experience Batching

Hierarchical Gradient Reduction (for Sync)



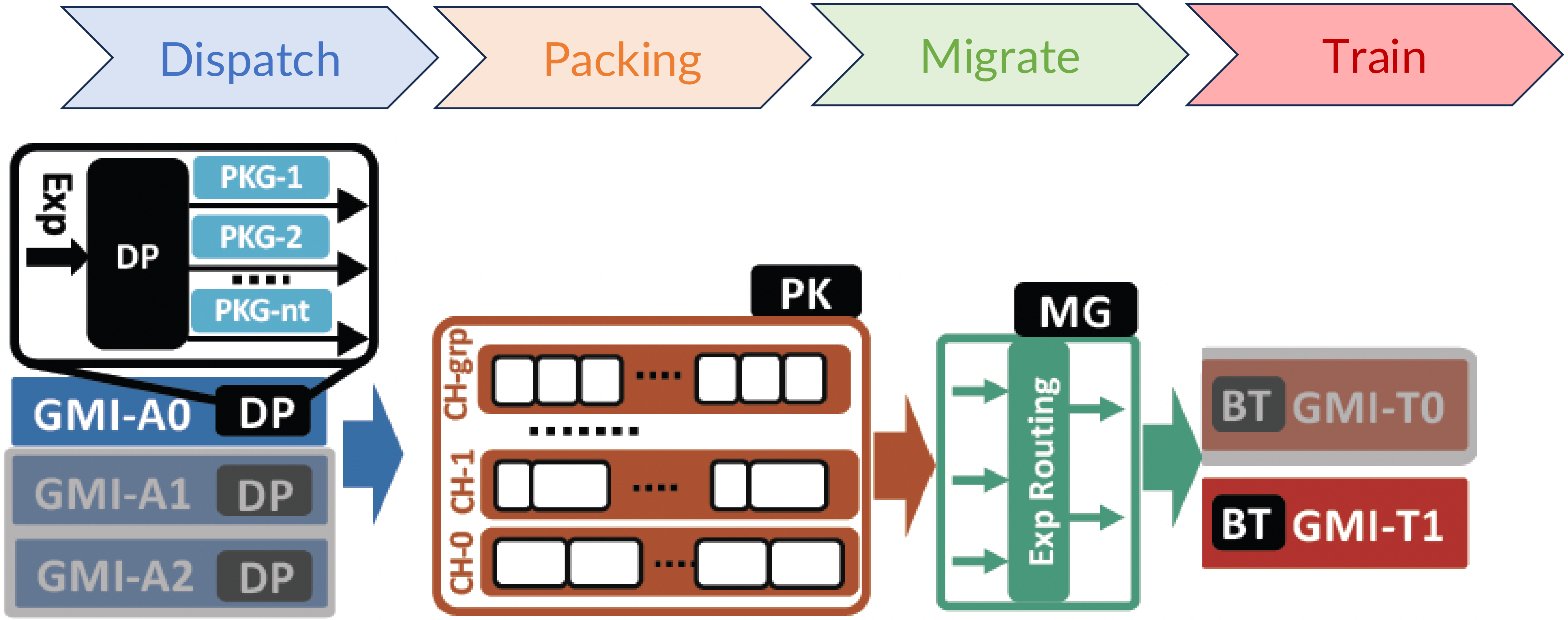
(a) IP only.

(b) RP only.

(c) Hybrid RP + IP.

1) Parallelized intra-GPU reduction.
2) Minimized Inter-GPU traffic.

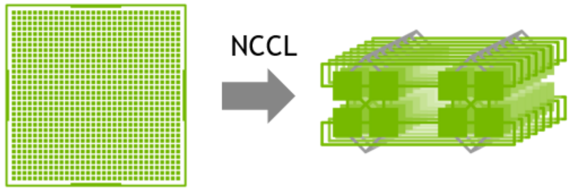
Channel-based Experience Sharing (for Async)



Experiments

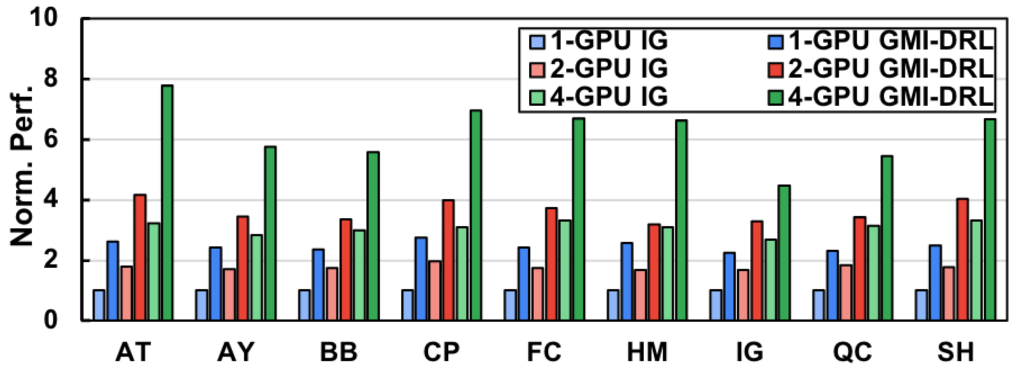
Benchmark	Abbr.	#ENV	#Dim.	Policy (Value) MLP
Ant	AT	1,024	60	60-256-128-64-8(1)
Anymal	AY	4,000	48	48-256-128-64-12(1)
BallBalance	BB	4,096	24	24-256-128-64-3(1)
Cartpole	CP	512	4	4-32-32-1(1)
FrankaCabinet	FC	2,048	23	23-256-128-64-9(1)
Humanoid	HM	4,096	108	108-200-400-100-21(1)
Ingenuity	IG	4,096	13	13-256-256-128-6(1)
Quadcopter	QC	8,192	21	21-256-256-128-12(1)
ShadowHand	SH	4,096	211	211-512-512-512-256-20(1)

Platform: NVIDIA DGX-A100 with 8xA100 connected with NVLinks.



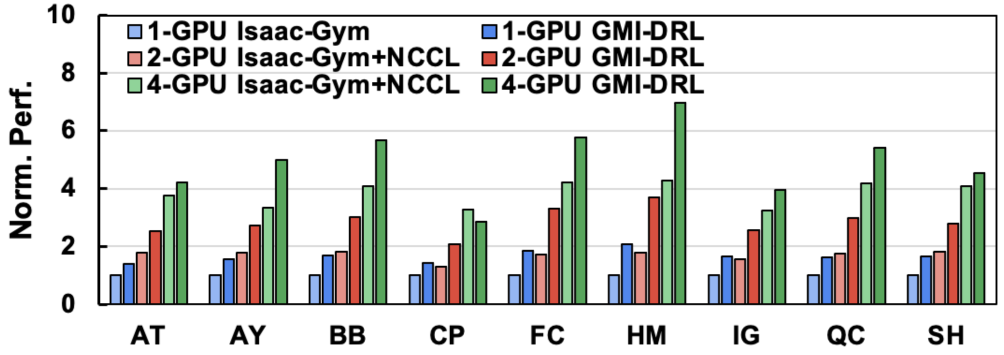
Performance

❖Policy Serving compared with **Isaac-Gym Serving**.



2.17x higher throughput on average

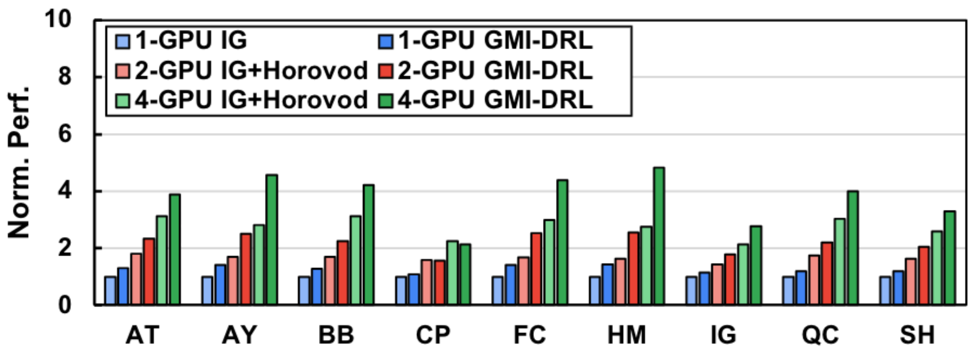
❖Policy Training compare with **Isaac-Gym w/ NCCL**.



1.54x higher throughput on average

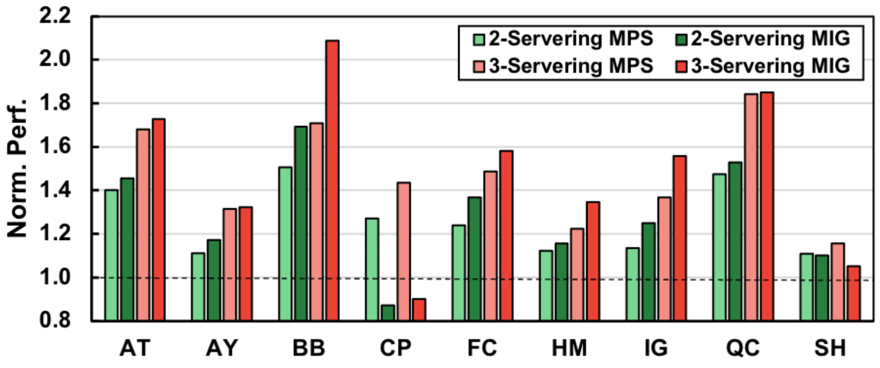
Performance (cont'd)

- Policy Training compared with **Isaac-Gym w/ Horovod**.



1.76x higher throughput on average

- Backend choice between MPS and MIG in Serving.



RAY

MindSpore

MIG could benefit on more complex and heavier RL benchmark

Thank You!

YK-Wang96/gmi-drl-ae

yuke.wang@rice.edu

https://wang-yuke.com