CSC 36000:
Modern Distributed Computing *NextGen with AI Agents*

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at City University of New York

Introduction to JAX for Distributed Computing

What is JAX?

JAX is a Python library for high-performance numerical computing and machine learning research.

- NumPy on Accelerators: JAX
- Composable Function Transformations: Apply transformations to your Python functions:
    - jit: Just-in-time compilation for speed.
    - grad: Automatic differentiation for optimization.
    - vmap: Automatic vectorization for batching.
    - pmap: Parallelization across multiple devices.

Example

```
import jax import jax.numpy as jnp
# A simple Python function
def predict(params, inputs):
    return jnp.dot(inputs, params)

# Let's transform it!
grad_fn = jax.grad(predict)
fast_grad_fn = jax.jit(grad_fn)
```

Just in time Compilation

```
from jax import jit

# A slow Python loop
def slow_function(x):
  for _ in range(5):
    x = x * 0.5 + 1.0
  return x

# A fast, compiled version
fast_function = jit(slow_function)

# The first run is slow (compilation), but subsequent runs are lightning fast!
```

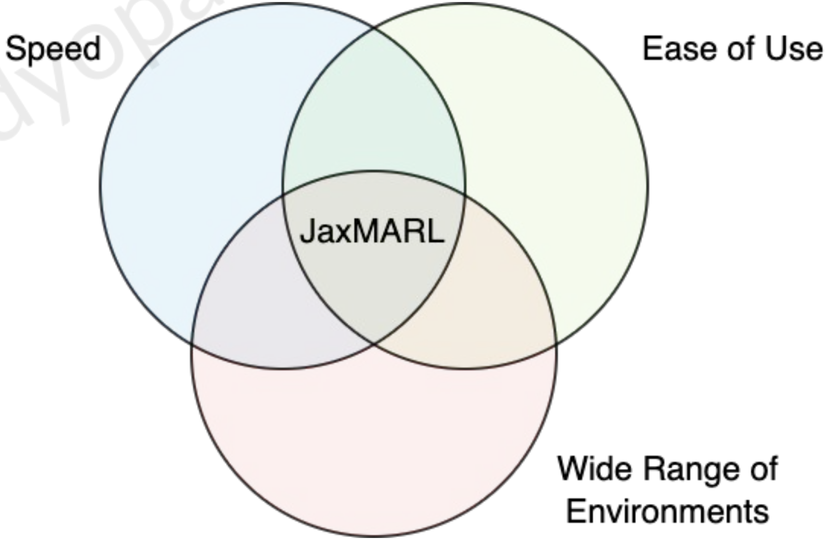
pmap

```
from jax import pmap

# Function to run on each device
def device_fn(x):
  # Each device gets a CHUNK of x
  return x * 2

# Create data and split it across devices
# Let's assume we have 8 devices
data = jnp.arange(8 * 3).reshape((8, 3)) # 8 rows, one for each device

# `pmap` compiles the function and runs one copy on each device
parallel_fn = pmap(device_fn)

# Run it! JAX handles sending each row to a different device.
result = parallel_fn(data)
print(result.shape) # Output: (8, 3)
```

Addressing Speedup with JAXMARL

Introduction

- Multi-Agent RL can be pretty slow!
- JAX-enabled hardware acceleration can make it 12500x faster!
- This means that experiments that once took days now take hours or minutes



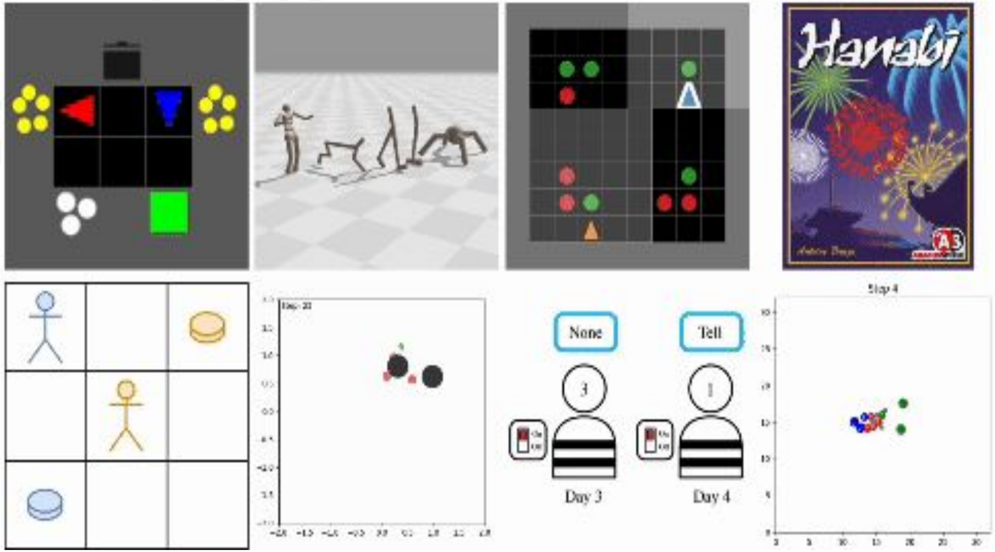
JAXMARL

- JAX is a Python library that makes writing hardware accelerated code easy
- JAXMARL co-locates the AI agent and Multi-Agent RL environment on a GPU
- JAXMARL vectorizes environment rollouts using a single function call

Multi-Agent Environments

- We implement 8 popular MARL environments
- We provide Q-learning and PPO baseline

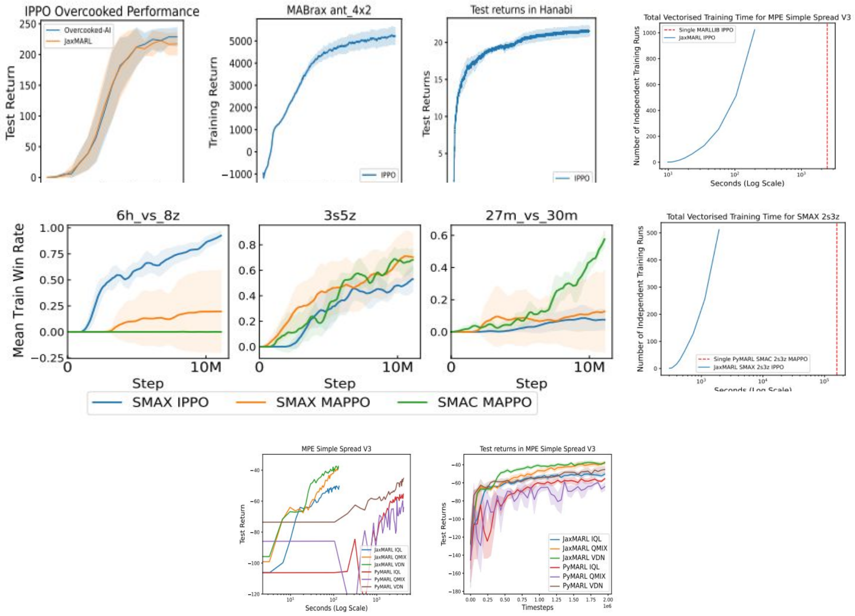
```python
import jax
from jaxmarl import make

key = jax.random.PRNGKey(0)
key, key_reset, key_act, key_step = jax.random.split(key, 4)

# Initialise and reset the environment.
env = make('MPE_simple_world_comm_v3')
obs, state = env.reset(key_reset)

# Sample random actions.
key_act = jax.random.split(key_act, env.num_agents)
actions = {agent: env.action_space(agent).sample(key_act[i]) \
           for i, agent in enumerate(env.agents)}

# Perform the step transition.
obs, state, reward, done, infos = env.step(key_step, state, actions)
```



Baseline and Speed Results

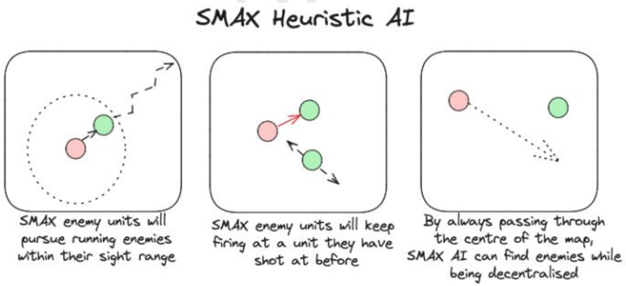
- The Jax Environments are much much faster than CPU baselines!
- Our evaluations demonstrate that our baselines are correct and can train quickly!

Environment	Original, 1 Env	Jax, 1 Env	Jax, 100 Envs	Jax, 10k Envs	Maximum Speedup
MPE Simple Spread	8.34×10^4	5.48×10^3	5.24×10^5	3.99×10^7	4.78×10^2
MPE Simple Reference	1.46×10^5	5.24×10^3	4.85×10^5	3.35×10^7	2.29×10^2
Switch Riddle	2.69×10^4	6.24×10^3	7.92×10^5	6.68×10^7	2.48×10^3
Hanabi	2.10×10^3	1.36×10^3	1.05×10^5	5.02×10^6	2.39×10^3
Overcooked	1.91×10^3	3.59×10^3	3.04×10^5	1.69×10^7	8.85×10^3
MABrax Ant 4x2	1.77×10^3	2.70×10^2	1.81×10^4	7.62×10^5	4.31×10^2
Starcraft 2s3z	8.31×10^1	5.37×10^2	4.53×10^4	2.71×10^6	3.26×10^4
Starcraft 27m vs 30m	2.73×10^1	1.45×10^2	1.12×10^4	1.90×10^5	6.96×10^3
STORM	–	2.48×10^3	1.75×10^5	1.46×10^7	–
Coin Game	1.97×10^4	4.67×10^3	4.06×10^5	4.03×10^7	2.05×10^3



Star-Craft with SMAX

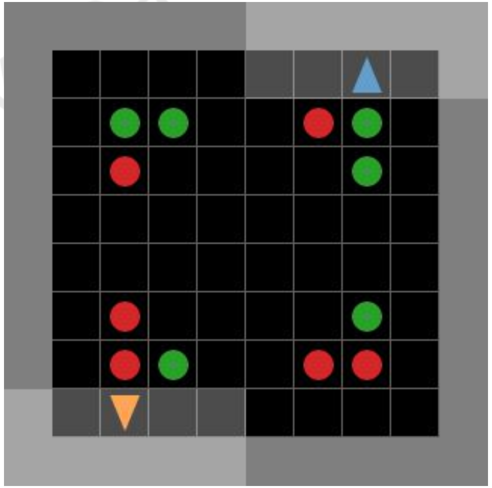
- We improve on SMAC and SMACv2's heuristics in SMAX
- The enemy heuristic is now fully decentralised meaning win-rates below 50% represent a concrete failure to learn
- SMAX is also faster and in pure Python so more customisable!



Scenario	Ally Units	Enemy Units	Start Positions
2s3z	2 stalkers and 3 zealots	2 stalkers and 3 zealots	Fixed
3s5z	3 stalkers and 5 zealots	3 stalkers and 5 zealots	Fixed
5m_vs_6m	5 marines	6 marines	Fixed
10m_vs_11m	10 marines	11 marines	Fixed
27m_vs_30m	27 marines	30 marines	Fixed
3s5z_vs_3s6z	3 stalkers and 5 zealots	3 stalkers and 6 zealots	Fixed
3s_vs_5z	3 stalkers	5 zealots	Fixed
6h_vs_8z	6 hydralisks	8 zealots	Fixed
smacv2_5_units	5 uniformly randomly chosen	5 uniformly randomly chosen	SMACv2-style
smacv2_10_units	10 uniformly randomly chosen	10 uniformly randomly chosen	SMACv2-style
smacv2_20_units	20 uniformly randomly chosen	20 uniformly randomly chosen	SMACv2-style

STORM

- STORM: Spatial-Temporal Representations of Matrix Games
- STORM allows to represent matrix games as grid-world scenarios
- Presents new challenges such as partial observability, multi-step agent interactions and longer time horizons



Questions?