A Resource-efficient Task Scheduling System using Reinforcement Learning

Chedi Morchdi[1], Cheng-Hsiang Chiu[2], Tsung-Wei Huang[2], Yi Zhou[1]

[1]Department of ECE, University of Utah

[2]Department of ECE, University of Wisconsin-Madison

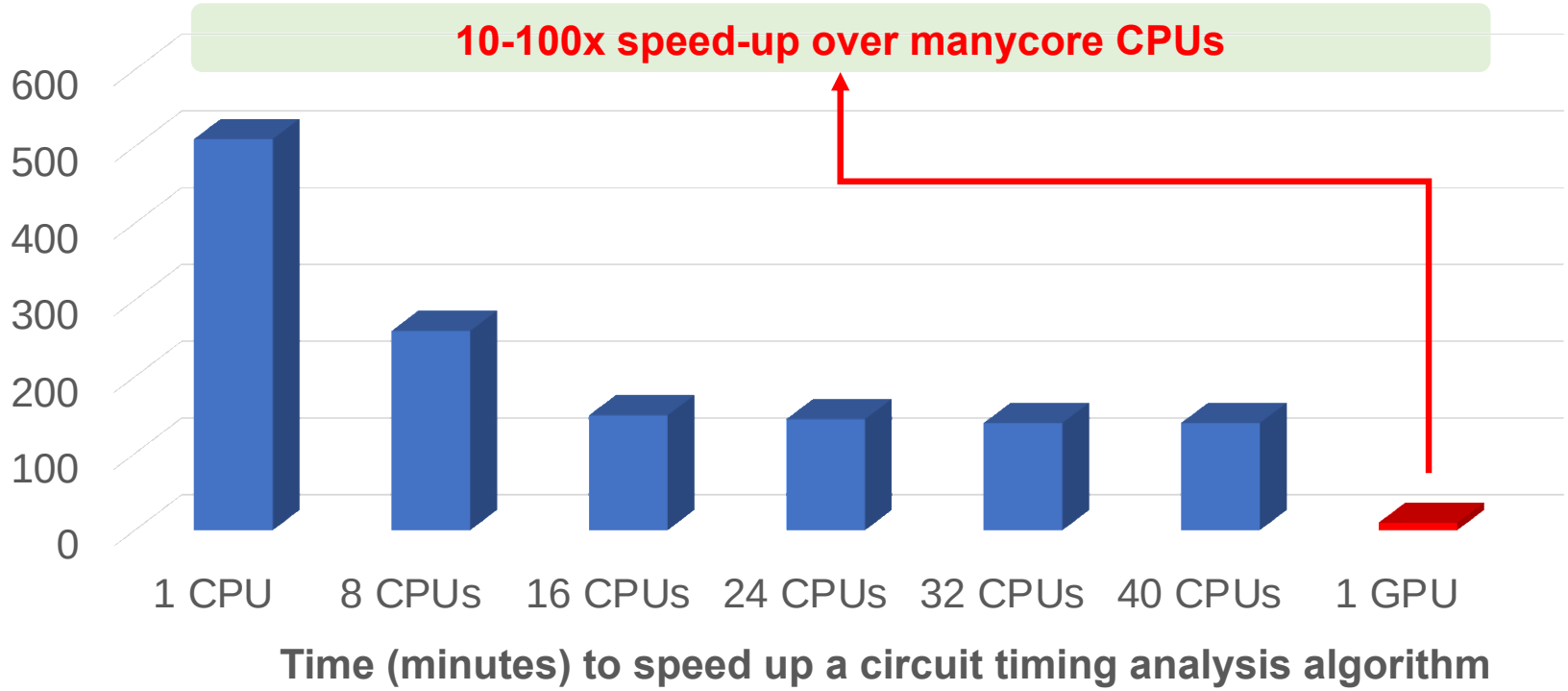
Content

- Introduction
- System Overview
- RL-Based Scheduling
- Experiments and Results
- Conclusion

Introduction

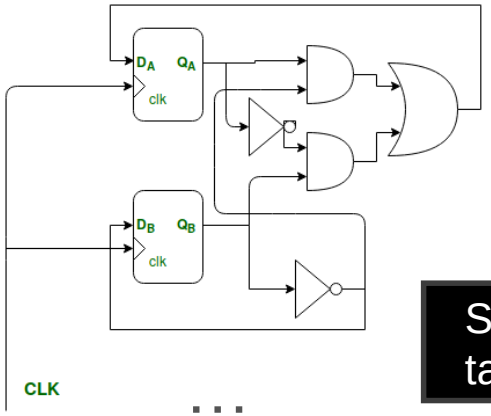
Why Parallelizing CAD?

- Advance the performance to a new level

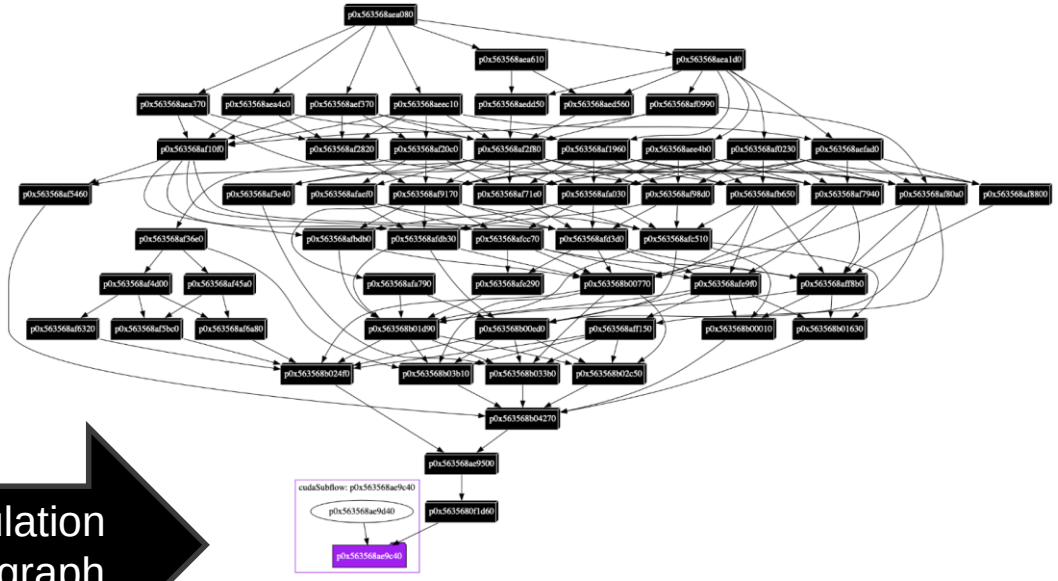


Today's Parallel CAD Workload is Complex

- **GPU-accelerated circuit analysis on a design of 500M gates[1]**
  - >100 kernels
  - >100 dependencies
  - >500s to finish
  - >10hrs turnaround



Simulation task graph

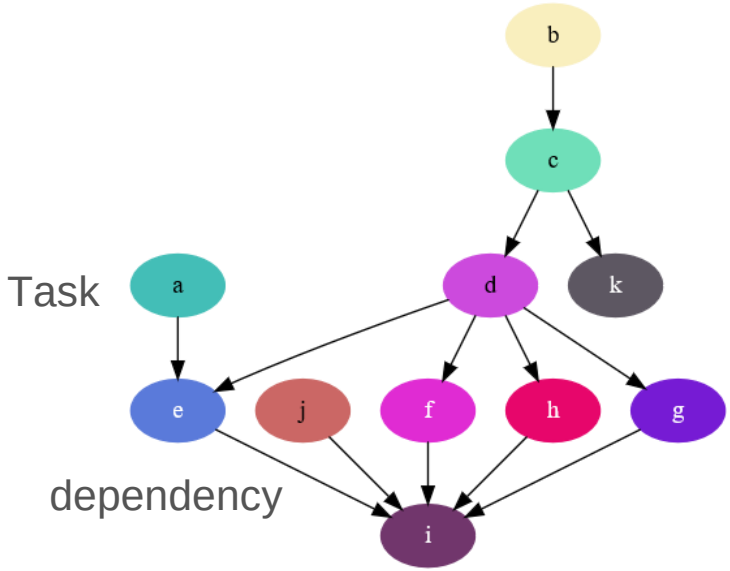


Task-parallel Programming Solution

- **Task graph parallelism scales the best for parallelizing CAD**
  - Capture programmers' intention in decomposing an algorithm into a top-down task graph
  - Runtime can scale these dependent tasks across a large number of processing units

OpenMP StarPU kokkos

TBB PaRSEC



Scheduling is a Big Challenges

- **Many CAD task graphs need multiple runs**
  - Both inside and outside the program (e.g., iterative optimization, multiple scenario timing analysis)
  - CAD task graphs are often very sparse (e.g., 17M tasks with 18M dependencies)
  - However, existing scheduling algorithms often count on general-purpose heuristics (e.g., random assignment, work stealing) that consume many workers to schedule sparse task graphs
    - Ex: OpenMP by default uses all available CPU cores, wasting a lot of CPU resources
- **Need a smart, adaptive, and resource-efficient scheduler**

Contribution

- **Scheduling Algorithm**: We introduced a RL-based task scheduling algorithm. Using fewer resources, we had comparable performance to existing solutions.

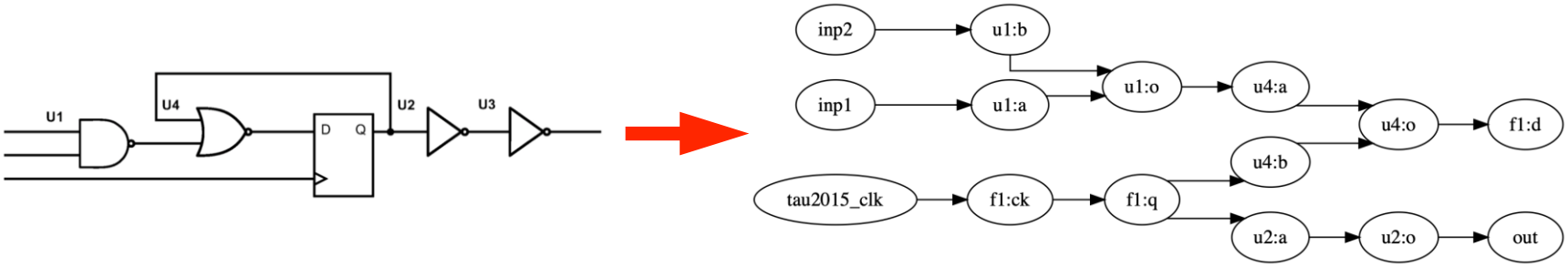
- **Generalizability**: Our proposed scheduling policy generalized well to a wide range of task graphs.

- **Extensible State Representations**: easy-to-extend state representation to accommodate new computing environment statistic.

System Overview

Problem Formulation

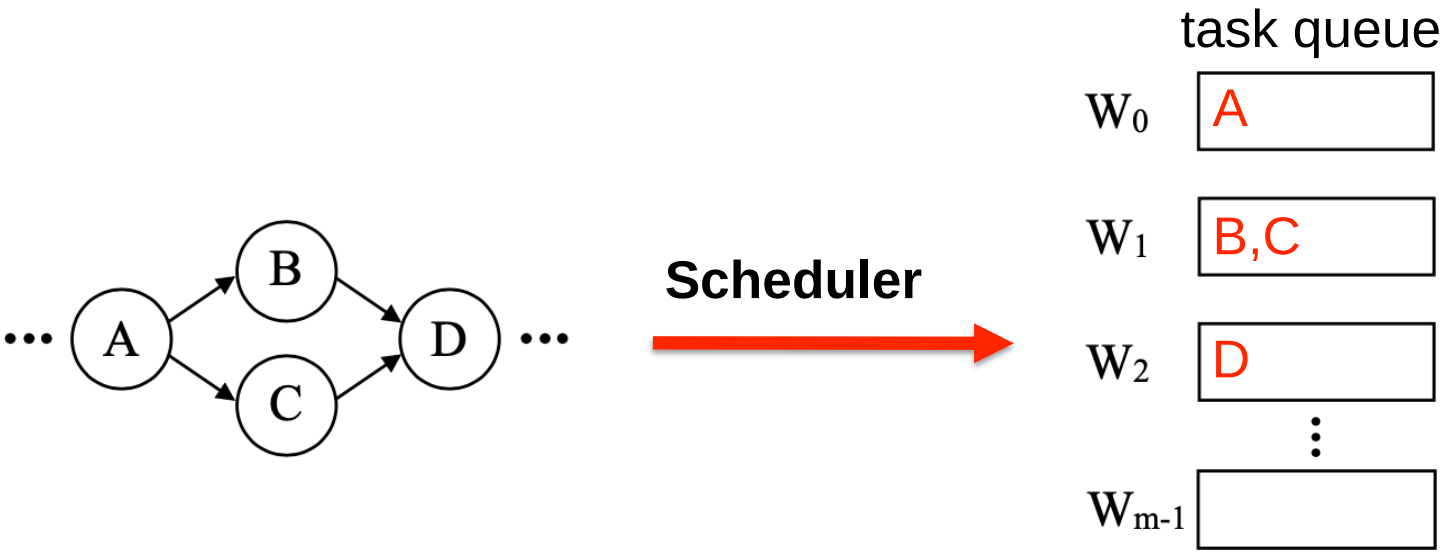
- Static timing analysis (STA): describe the timing analysis circuit as a task graph



- Nodes and edges represent tasks and their dependencies
- Dependencies constraint the execution order of tasks and denote the data flow

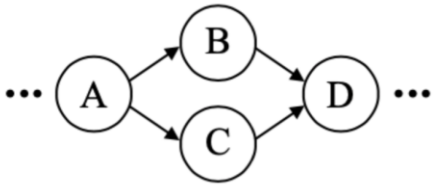
Scheduler

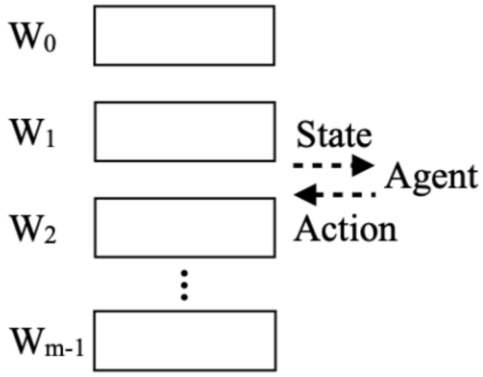
- Consider a system with multi-cores (called workers)
- Scheduler assigns tasks to workers



RL-based Scheduler

- We propose a reinforcement learning (RL)-based scheduler
- Formulate task-scheduling as Markov Decision Process
- Based on *State*, RL scheduler takes *Action* to assign tasks

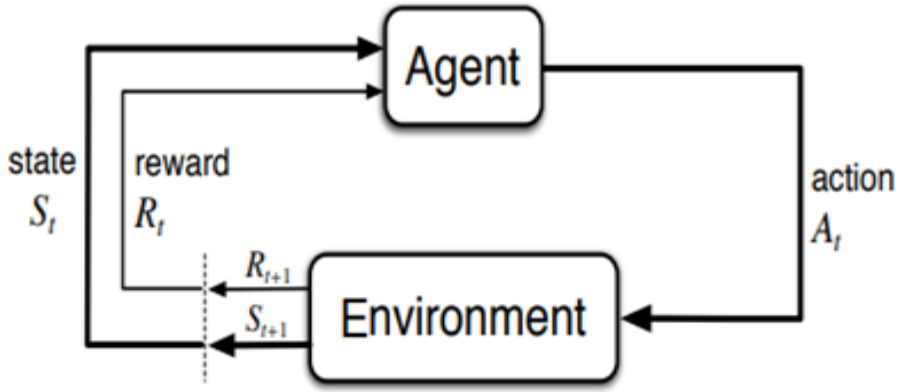




Reinforcement Learning (RL)

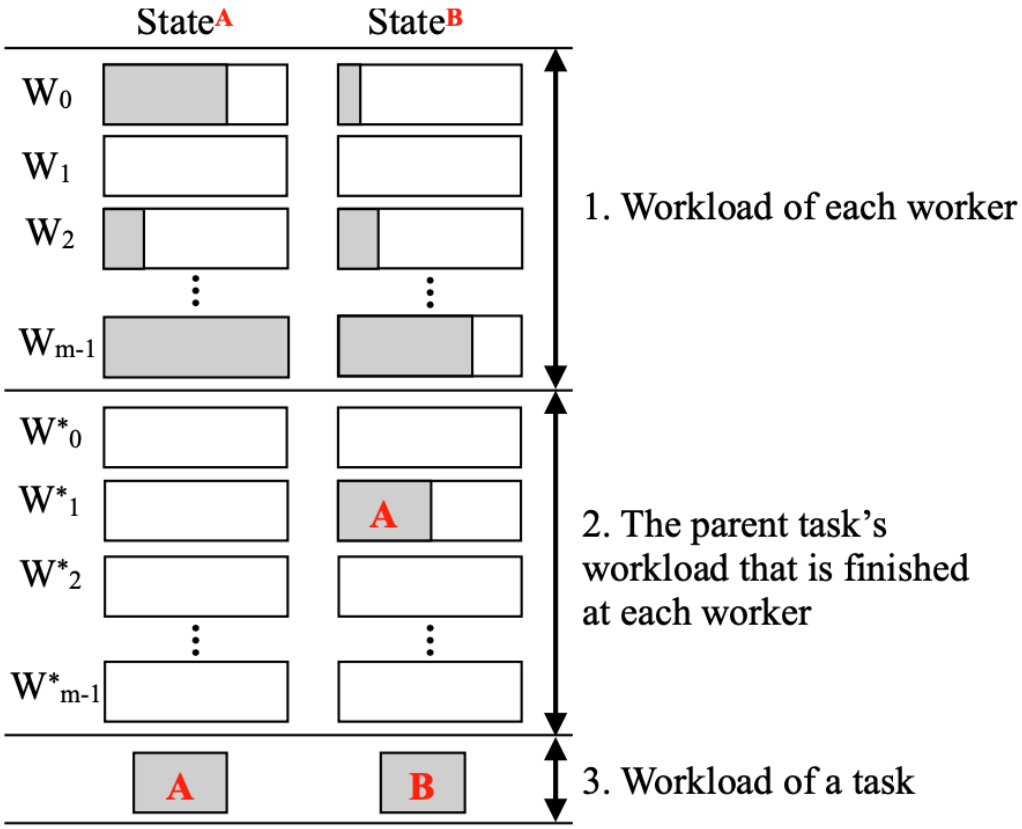
RL and Markov Decision Process (MDP)

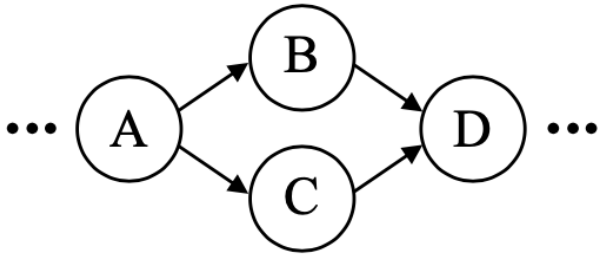
- MDP: describe how agent interacts with an environment



- MDP: state s_t, policy π, action a_t, reward r_t
- Goal: learn optimal policy that yields the highest reward

State


State^A
State^B
W0
W1
W2
Wm-1
W*0
W*1
W*2
W*m-1
A
A
B
1. Workload of each worker
2. The parent task's workload that is finished at each worker
3. Workload of a task


A
B
C
D

Action and Policy

- Action: m possible actions (for m workers)
- Policy: specified by state-action value table $Q(s, a)$
  - ❑ expected total reward of taking action a in state s
  - ❑ learned using deep Q-learning

$$\pi(a|s) = \arg\max_{a'} Q(s, a')$$

- State transition: after task is assigned, the state changes (worker's queue workload changed).

Reward

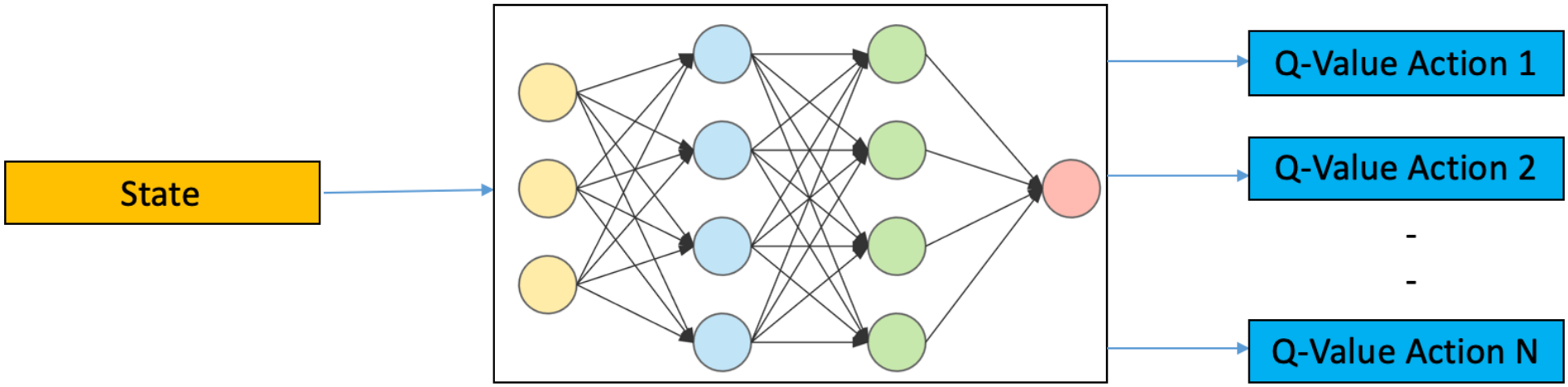
- r_t := $-\log_{10}$(*workload balance*) $-\ \alpha \log_{10}$(*transfer cost*)
  - Workload balance: max(queue loads)-min(queue loads)
  - Transfer cost: sum of workloads of parents that were not assigned to the same worker
  - $\alpha > 0$: hyperparameter.
  - Reward design penalizes actions that cause imbalanced queue workloads and high transfer cost.

Deep Q-learning algorithm

- The Q-function satisfies the following Bellman equation:

$$Q(s_t, a_t) = r_t + \gamma \mathbb{E}\left[\max_a Q(s_{t+1}, a)\right]$$

- We use a deep NN to parametrize the Q function.
- The NN takes the state as input and outputs the Q values of all possible actions:



- Step 1: Data Collection
  - Take action a_t (using ϵ-greedy policy)
  - Collect data (s_t, a_t, r_t, s_{t+1}), add to replay memory

- Step 2: Data Sampling
  - Sample batch B data from replay memory.
  - Compute target $y_T = r_T + \gamma \max_a Q(s_{T+1}, a)$

- Step 3: Update Q-network
  - Compute loss $L = \frac{1}{B}\sum_{T \in B}\left(y_T - Q_\theta(s_T, a_T)\right)^2$
  - Update Q_θ via backpropagation

Experimental Results

Experiments Setting

- Evaluated on static timing analysis (STA) application
- 9 task graphs generated using OpenTimer
- Compiled using gcc-12, -std=c++17 and -O3 enabled
- Experiments ran on a Ubuntu 19.10 (Eoan Ermine) machine with 80 Intel Xeon Gold 6138 CPU at 2.00GHz and 256 GB RAM

Baseline Scheduler

- Assigns tasks uniformly at random to the 40 workers
- Not adaptive to the dynamic computing environment
- Widely used to schedule STA workload.

RL-based Scheduler

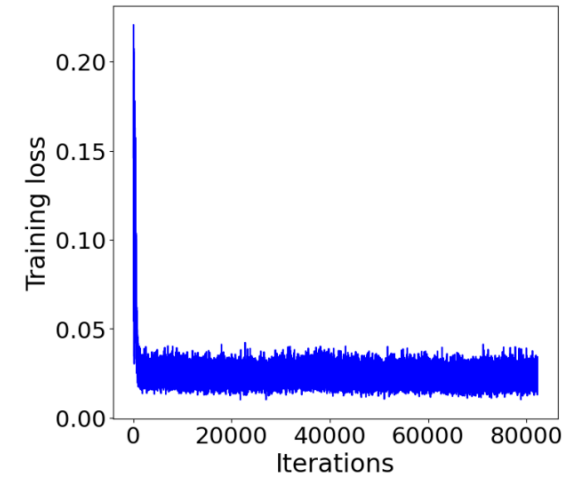
- Algorithm 1 implemented with hyper-parameters:
  - batch size B = 64
  - target network synchronization period K = 10
  - reward discount factor γ = 0.95
  - reward weight α = 0.01
  - experience replay memory size N = 10k

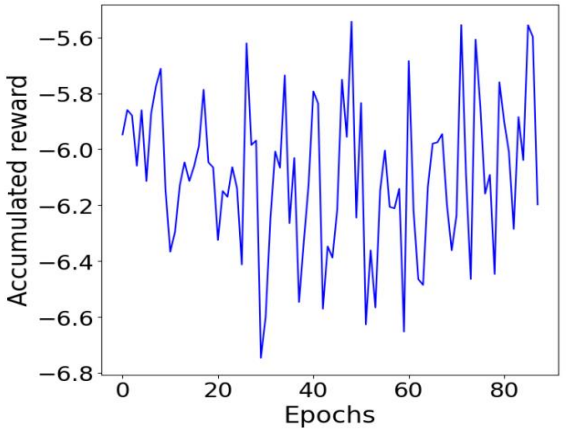
- Trained on mixed graph: aes_core, tv80 and c6288.

Training

- Policy parameters θ trained using Adam optimizer with learning rate η = 1e − 4.

- The training loss decays quickly, indicating that the learned policy performs well on the training data.





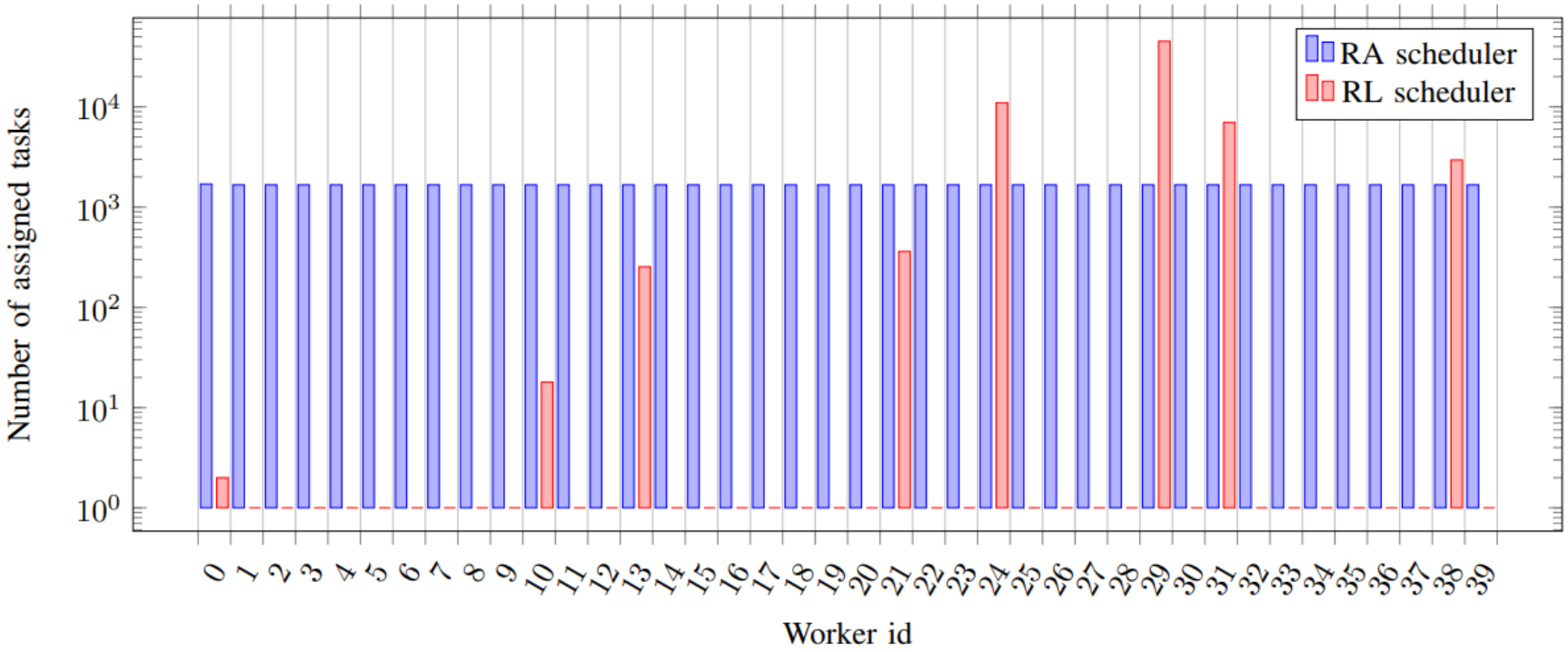
Performance Comparison

- Runtime of RL-based scheduler is consistently slightly lower than the RA scheduler for all the test graphs.
- The RL scheduler uses less workers compared to the RA scheduler (7-8 workers vs 40 workers).

Graph	$\|V\|$	$\|E\|$	Runtime (Seconds)			# Workers		
			RA	RL	Improvement	RA	RL	Improvement
mixed graph	88,626	115,777	38.44	**38.29**	0.39%	40	**7**	471%
aes_core	66,751	86,446	29.55	**28.89**	2.28%	40	**8**	400%
ac97_ctrl	42,438	53,558	18.92	**18.09**	4.59%	40	**8**	400%
tv_80	17,038	23,087	7.76	**7.04**	10.22%	40	**8**	400%
wb_dma	13,125	16,593	5.52	**5.33**	3.56%	40	**8**	400%
c6288	4,837	6,244	2.01	**1.98**	1.52%	40	**8**	400%
c7552_slack	3,802	4,791	1.75	**1.60**	9.38%	40	**7**	471%
usb_phy_ispd	2,447	2,999	1.11	**1.00**	11%	40	**7**	471%
s1494	2,292	2,925	1.04	**0.97**	7.22%	40	**7**	471%

Performance Comparison

- Histogram of tasks over workers on aes_core graph.



Performance Comparison

- Histogram of tasks over workers on mixed graph.

Conclusions

Conclusions

- Developed a resource-efficient RL-based task scheduling system adapted to the dynamic computing environment.
- Compared to the baseline scheduler, our RL-based scheduler achieved a lower runtime on all task graphs while using only 20% of workers.
- Future work: extend to a distributed environment and consider GPU task graphs into our state model.

Thank you!