Parallel DYNA

Real-time model learning RL

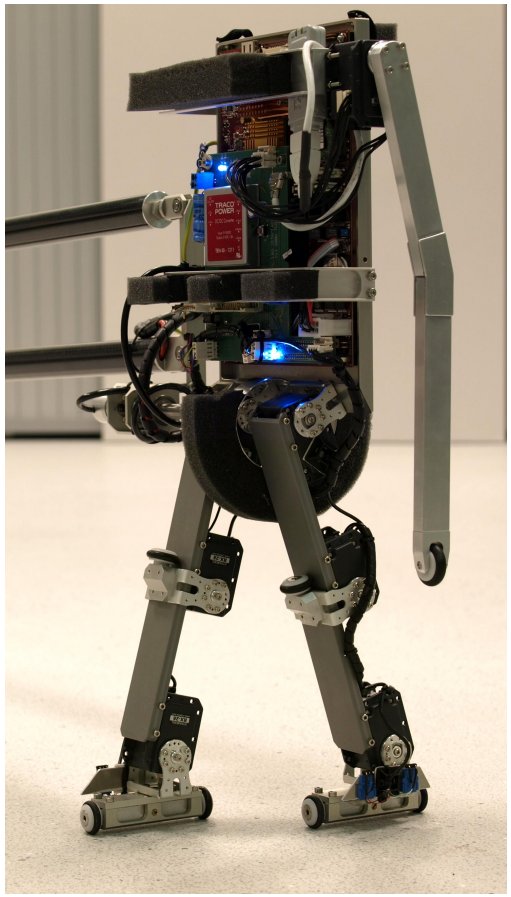
Wouter Caarls, Delft University of Technology
May 2nd, 2013

TU Delft

Reinforcement learning for robots

- Sample complexity of traditional RL algorithms is too high
  - Robots break
  - Too slow
- Computational complexity of sample-efficient algorithms is too high
  - System is idle while computing
  - Controlled by outdated control policy

  This may increase total learning time

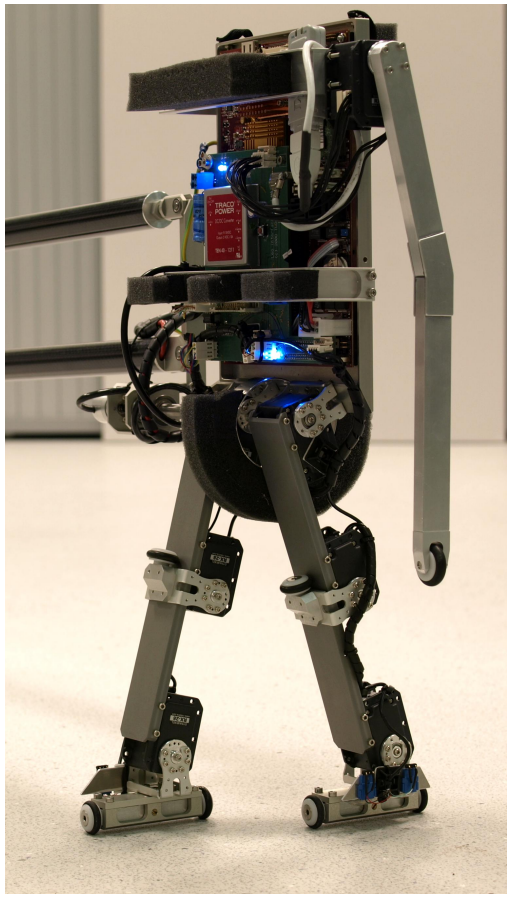


[Schuitema et al., 2010]

Reinforcement learning for robots

- Sample complexity of traditional RL algorithms is too high
  - Robots break
  - Too slow
- Computational complexity of sample-efficient algorithms is too high
  - System is idle while computing
  - Controlled by outdated control policy

This may increase total learning time

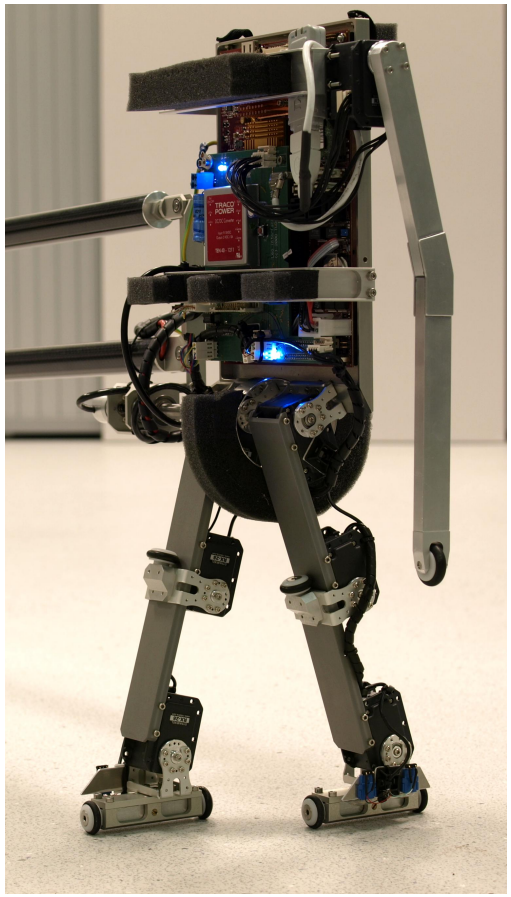


[Schuitema et al., 2010]

Reinforcement learning for robots

- Sample complexity of traditional RL algorithms is too high
  - Robots break
  - Too slow
- Computational complexity of sample-efficient algorithms is too high
  - System is idle while computing
  - Controlled by outdated control policy

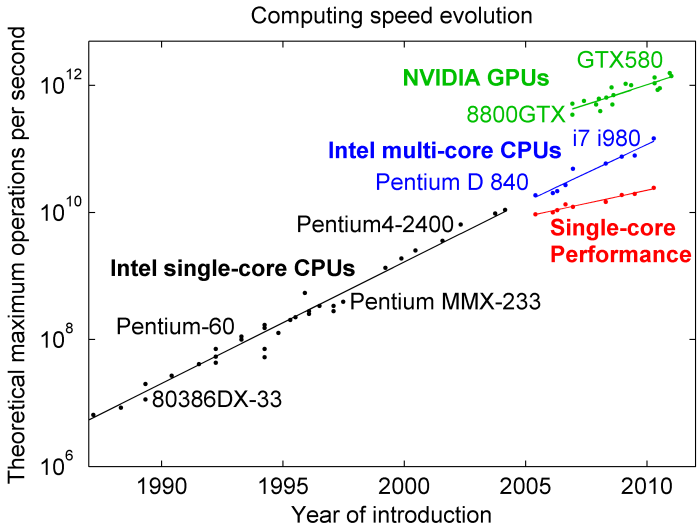
  This may increase total learning time



[Schuitema et al., 2010]

Why parallel reinforcement learning?

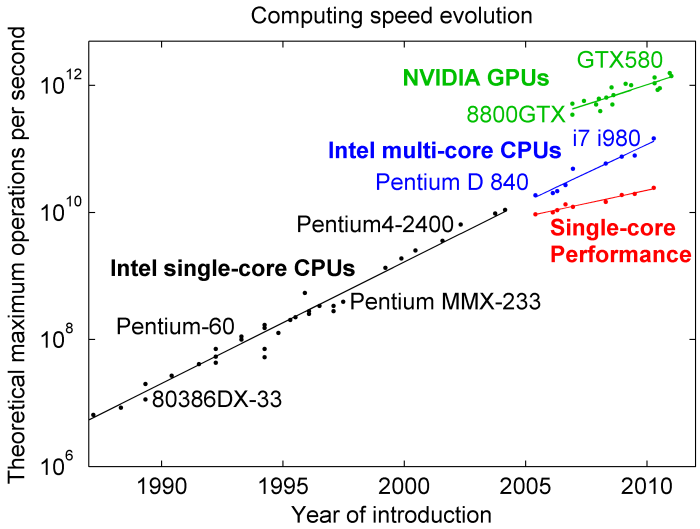
- Sequential CPU speed is not increasing to compensate for increased computational complexity
  - Moore's Law is about # of transistors, not clock frequency
  - More transistors can only do so much for single-threaded applications
  - Multi-core is the primary vectory of speed increase



Parallelism is the way to reduce computation time

Why parallel reinforcement learning?

- Sequential CPU speed is not increasing to compensate for increased computational complexity
  - Moore's Law is about # of transistors, not clock frequency
  - More transistors can only do so much for single-threaded applications
  - Multi-core is the primary vectory of speed increase



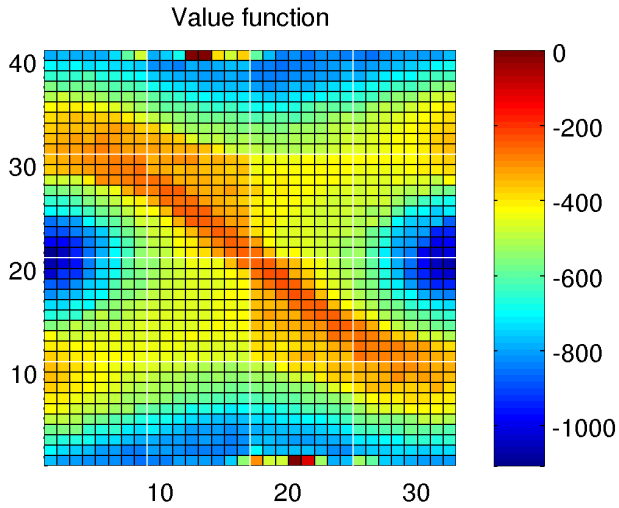
Parallelism is the way to reduce computation time

Outline

Outline

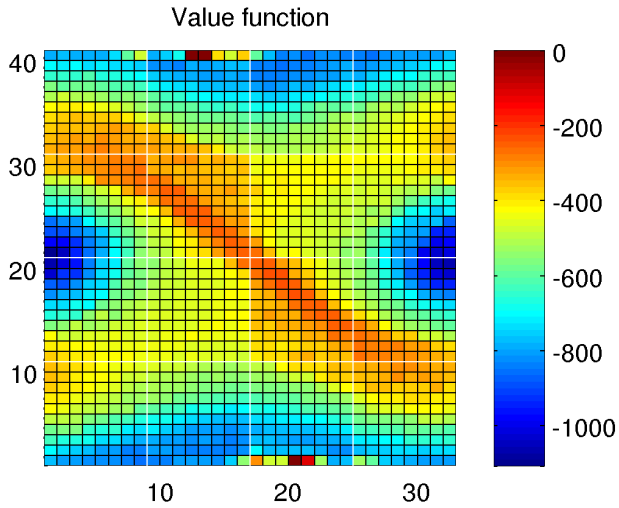
Traditional parallel RL

- State-space decomposition
  - Split up state space
  - Each processor learns in one subdomain
  - Value exchange across borders
- Multiple learners
  - Many agents learning the same task
  - Shared or replicated value function
  - Periodic or prioritized value exchange



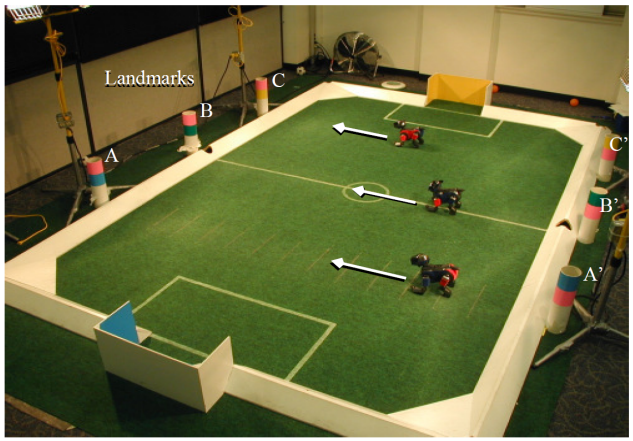
Traditional parallel RL

- State-space decomposition
  - Split up state space
  - Each processor learns in one subdomain
  - Value exchange across borders
- Multiple learners
  - Many agents learning the same task
  - Shared or replicated value function
  - Periodic or prioritized value exchange



Problems

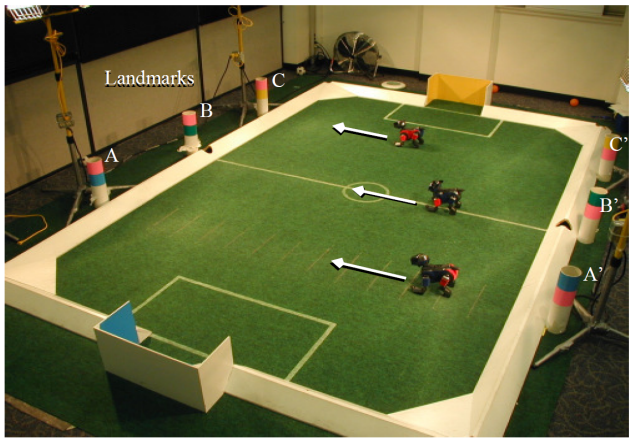
- Requires multiple systems
  - Robots
    May differ between each other
  - Simulations/models
    Differ from real system
- Limited by control frequency
  - Typically 20-100 Hz on real systems
  - Not a problem with simulations



[Kohl & Stone, 2004]

Problems

- Requires multiple systems
  - Robots
    May differ between each other
  - Simulations/models
    Differ from real system
- Limited by control frequency
  - Typically 20-100 Hz on real systems
  - Not a problem with simulations



[Kohl & Stone, 2004]

Other approaches

- Model learning
  - Multiple agents reading out single model
  - Model readout is faster than control frequency
- Experience replay
- Parallelization of function approximator
  - ANN
  - LSTD-Q
- Learning multiple tasks
  - Horde architecture

Other approaches

- Model learning
  - Multiple agents reading out single model
  - Model readout is faster than control frequency
- Experience replay
- Parallelization of function approximator
  - ANN
  - LSTD-Q
- Learning multiple tasks
  - Horde architecture

Other approaches

- Model learning
  - Multiple agents reading out single model
  - Model readout is faster than control frequency
- Experience replay
- Parallelization of function approximator
  - ANN
  - LSTD-Q
- Learning multiple tasks
  - Horde architecture

Other approaches

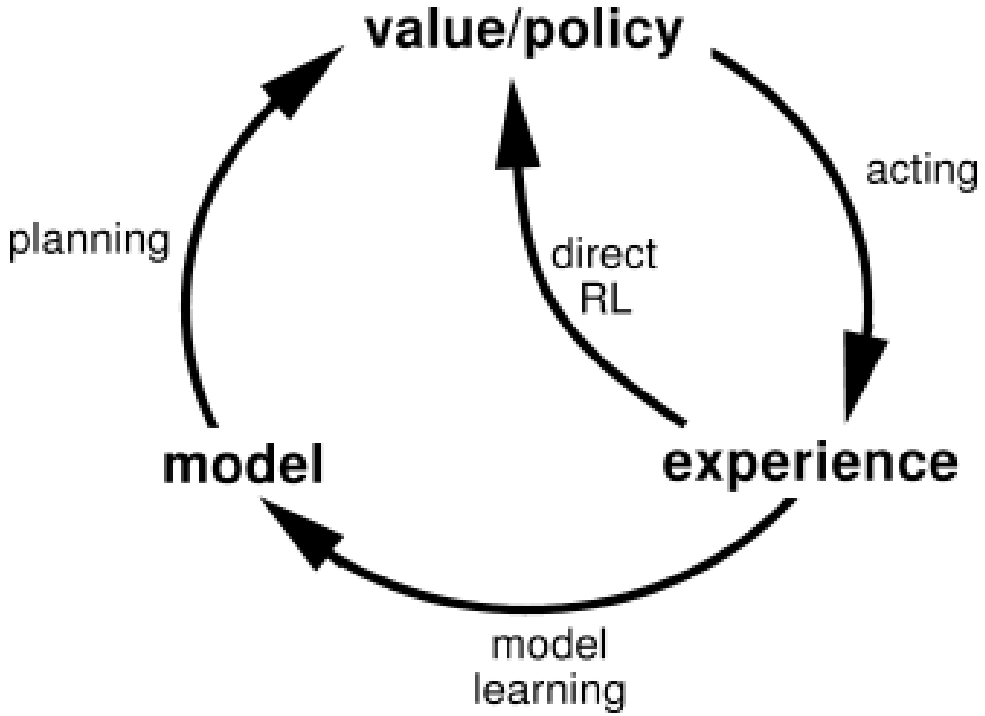
- Model learning
  - Multiple agents reading out single model
  - Model readout is faster than control frequency
- Experience replay
- Parallelization of function approximator
  - ANN
  - LSTD-Q
- Learning multiple tasks
  - Horde architecture

Other approaches

- Model learning
  - Multiple agents reading out single model
  - Model readout is faster than control frequency
- Experience replay
- Parallelization of function approximator
  - ANN
  - LSTD-Q
- Learning multiple tasks
  - Horde architecture

Outline

DYNA architecture



[Sutton & Barto, 1998]

Options

- Model approximator
  - Grid of state transition probabilities
  - Grid of feature transition probabilities
  - ANN, LWR, GP, etc.
- Value approximator
  - Tile coding
  - Basis functions
  - ANN, LWR, etc.
- Choice of model updates
  - Random
  - Trajectories (from current state)
  - Prioritized sweeping
- Updates per control step (K)

Options

- Model approximator
  - Grid of state transition probabilities
  - Grid of feature transition probabilities
  - ANN, LWR, GP, etc.
- Value approximator
  - Tile coding
  - Basis functions
  - ANN, LWR, etc.
- Choice of model updates
  - Random
  - Trajectories (from current state)
  - Prioritized sweeping
- Updates per control step (K)

Options

- Model approximator
  - Grid of state transition probabilities
  - Grid of feature transition probabilities
  - ANN, LWR, GP, etc.
- Value approximator
  - Tile coding
  - Basis functions
  - ANN, LWR, etc.
- Choice of model updates
  - Random
  - Trajectories (from current state)
  - Prioritized sweeping
- Updates per control step (K)

Options

- Model approximator
  - Grid of state transition probabilities
  - Grid of feature transition probabilities
  - ANN, LWR, GP, etc.
- Value approximator
  - Tile coding
  - Basis functions
  - ANN, LWR, etc.
- Choice of model updates
  - Random
  - Trajectories (from current state)
  - Prioritized sweeping
- Updates per control step (K)

Options

- Model approximator
  - Grid of state transition probabilities
  - Grid of feature transition probabilities
  - ANN, LWR, GP, etc.
- Value approximator
  - Tile coding
  - Basis functions
  - ANN, LWR, etc.
- Choice of model updates
  - Random
  - Trajectories (from current state)
  - Prioritized sweeping
- Updates per control step (K)

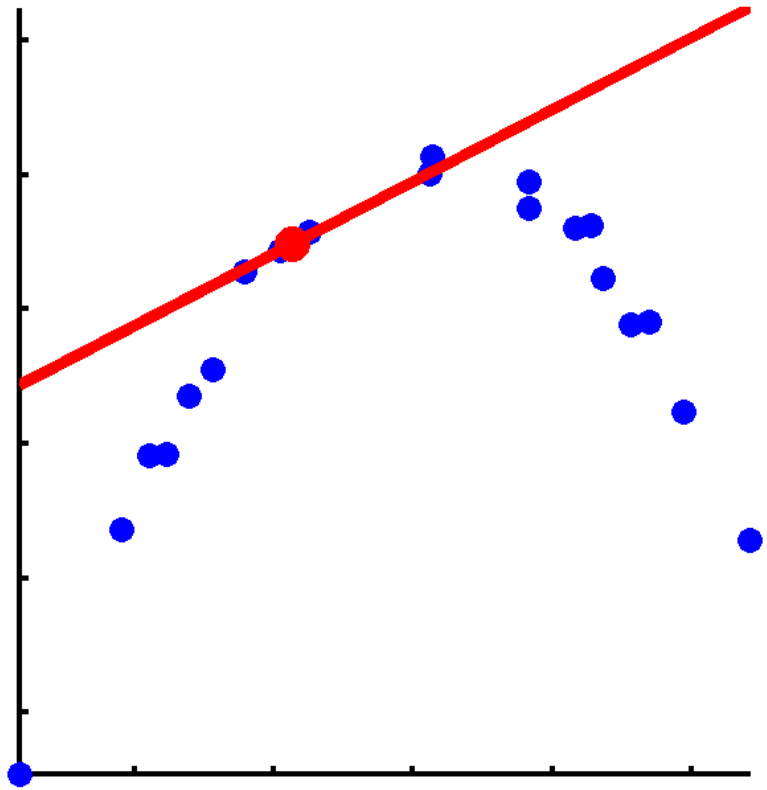
Locally weighted regression

- Find k nearest neighbors
  - Approximate nearest neighbor search
- Weigh according to distance

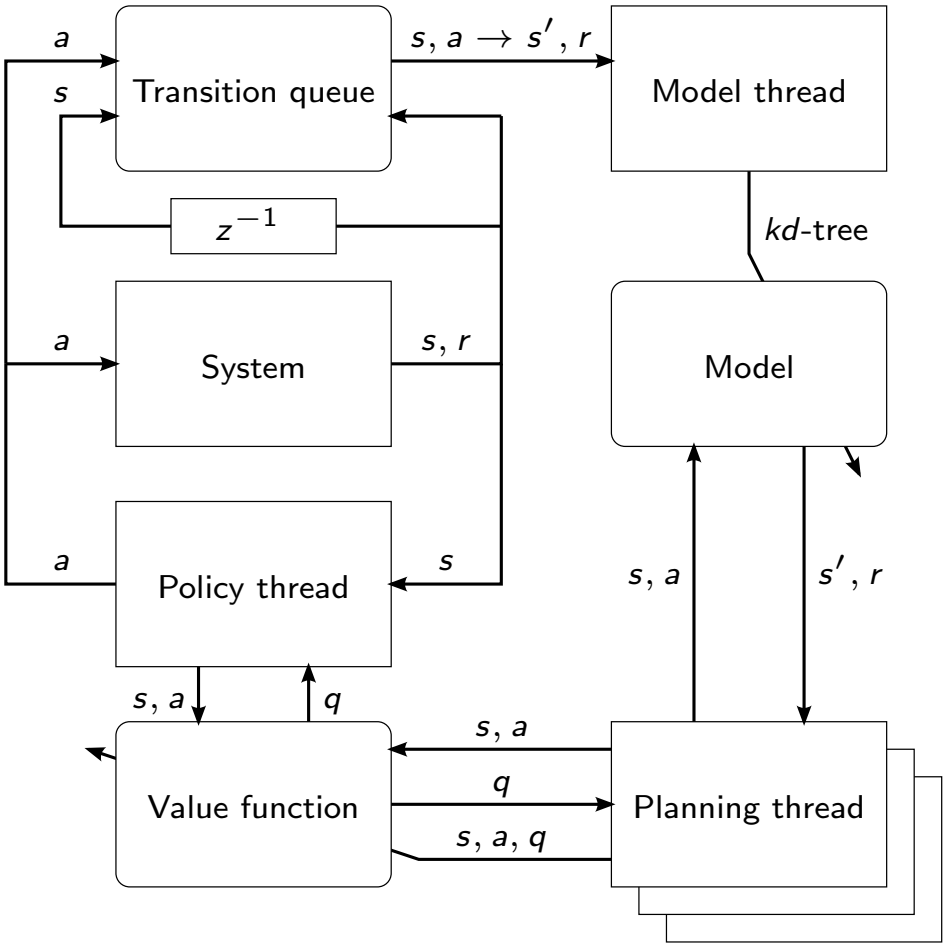
$$w(p) = e^{-\left(\frac{|p-q|_2}{h}\right)^2}$$

where h is the distance to the kth nearest neighbor

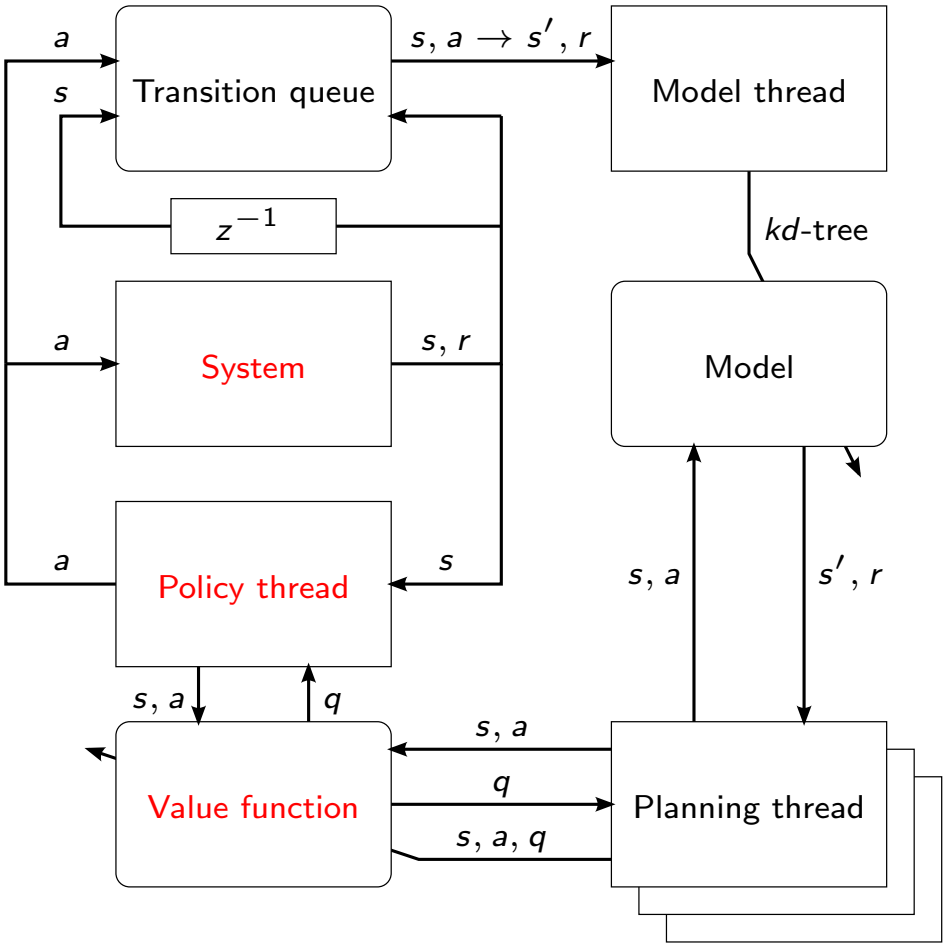
- Fit linear model using least-squares regression



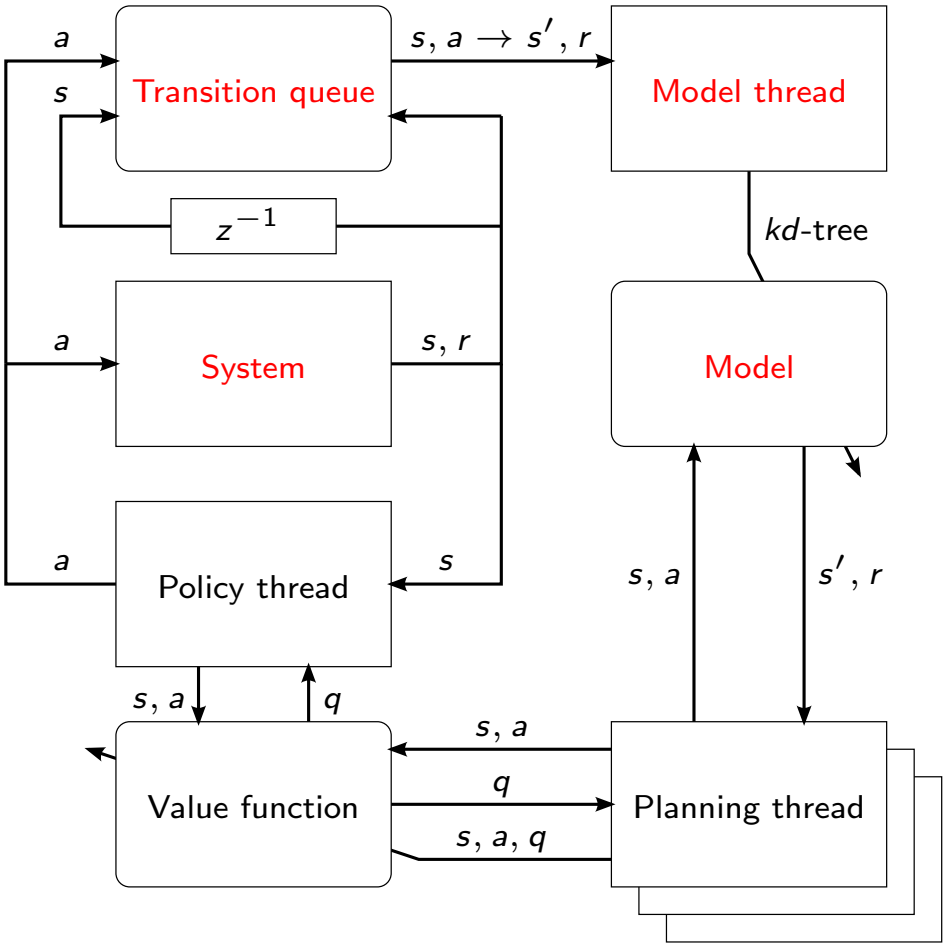
Parallel DYNA



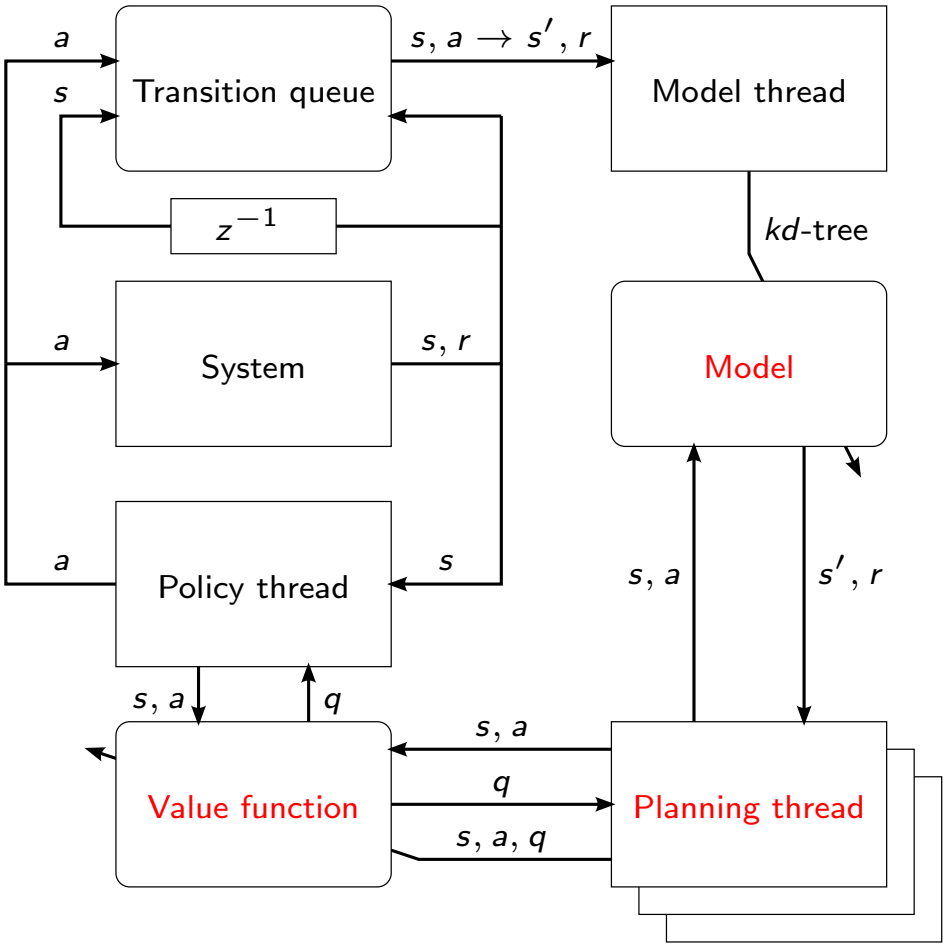
Parallel DYNA



Parallel DYNA

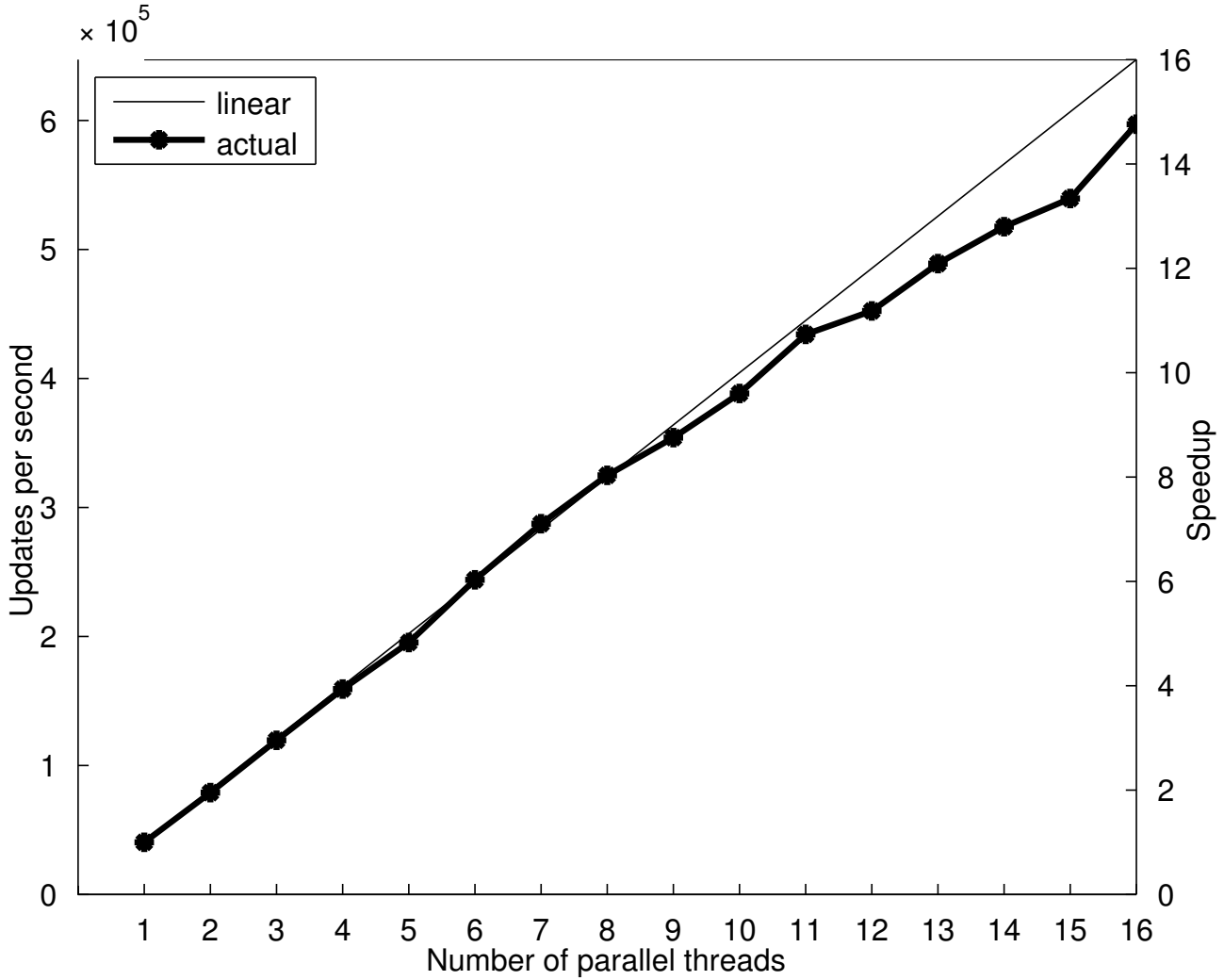


Parallel DYNA

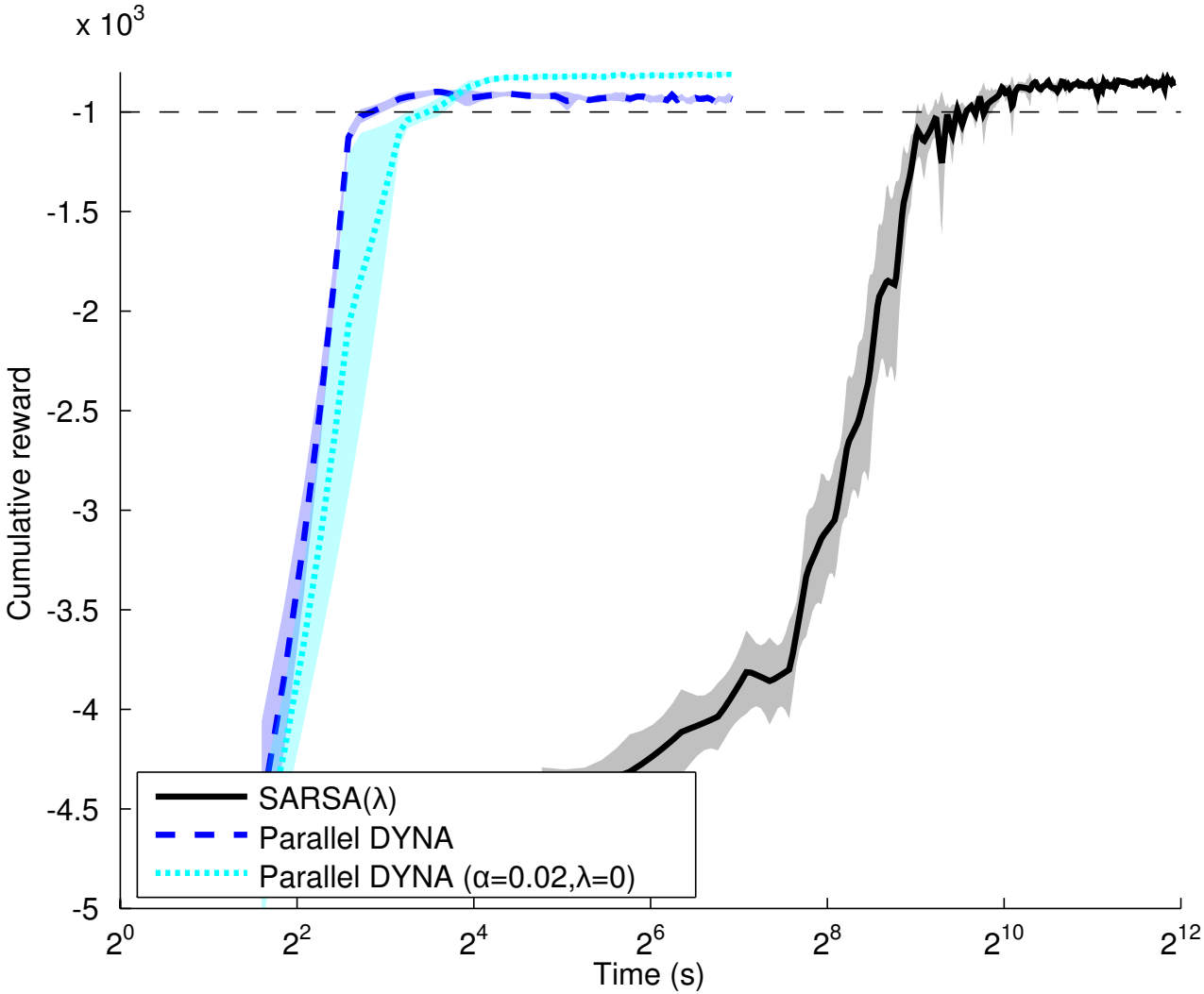


Outline

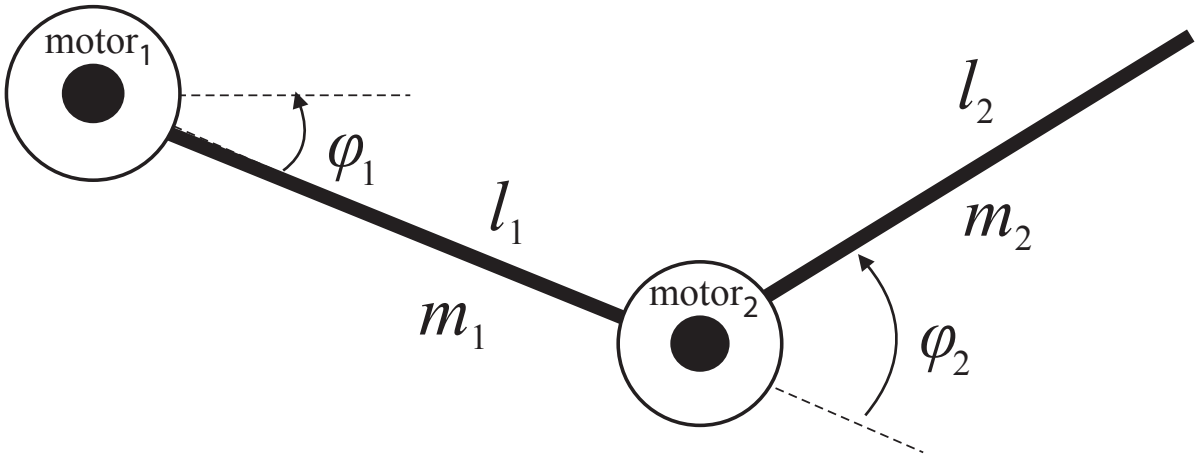
Computational efficiency



Inverted pendulum

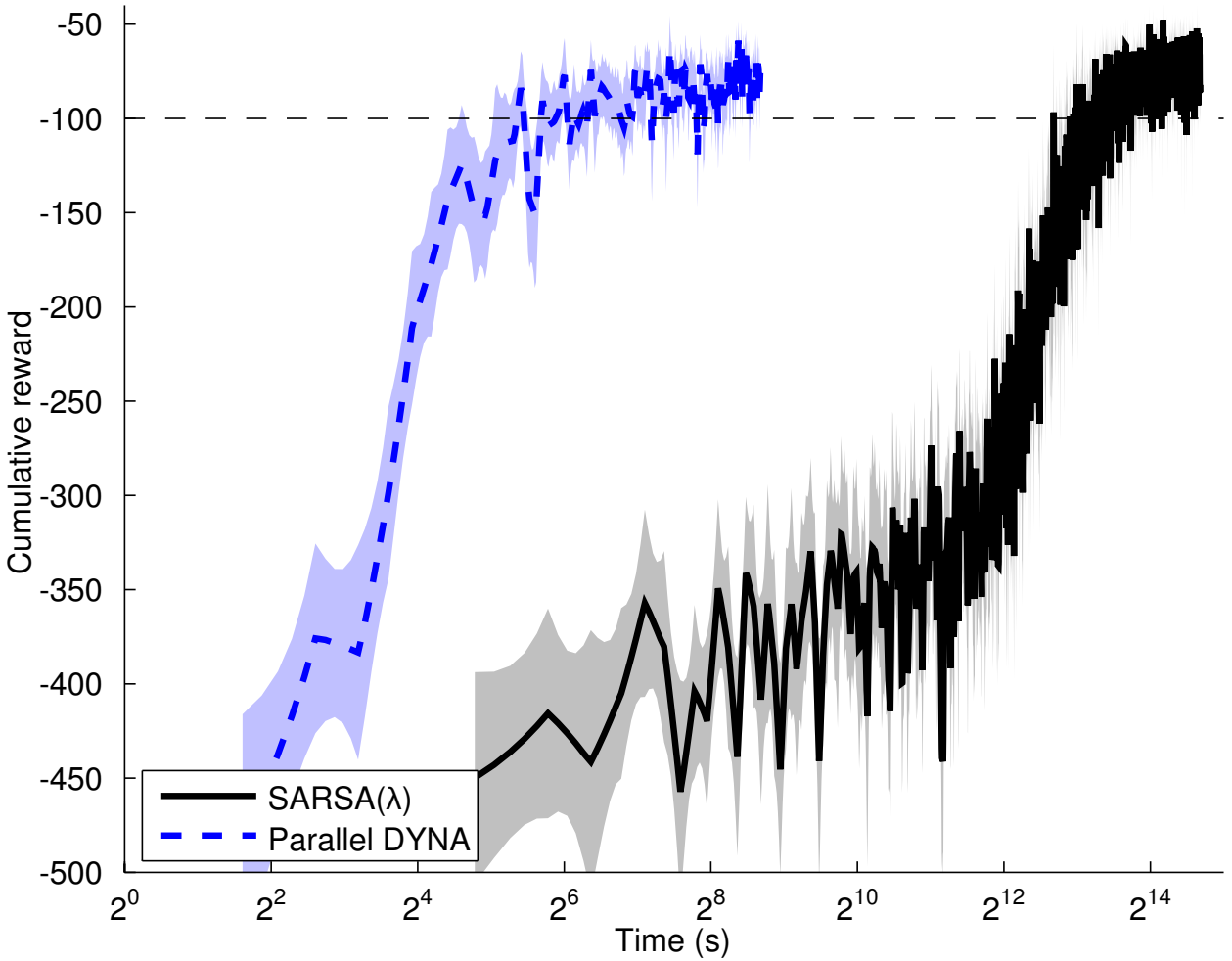


Two-link manipulator

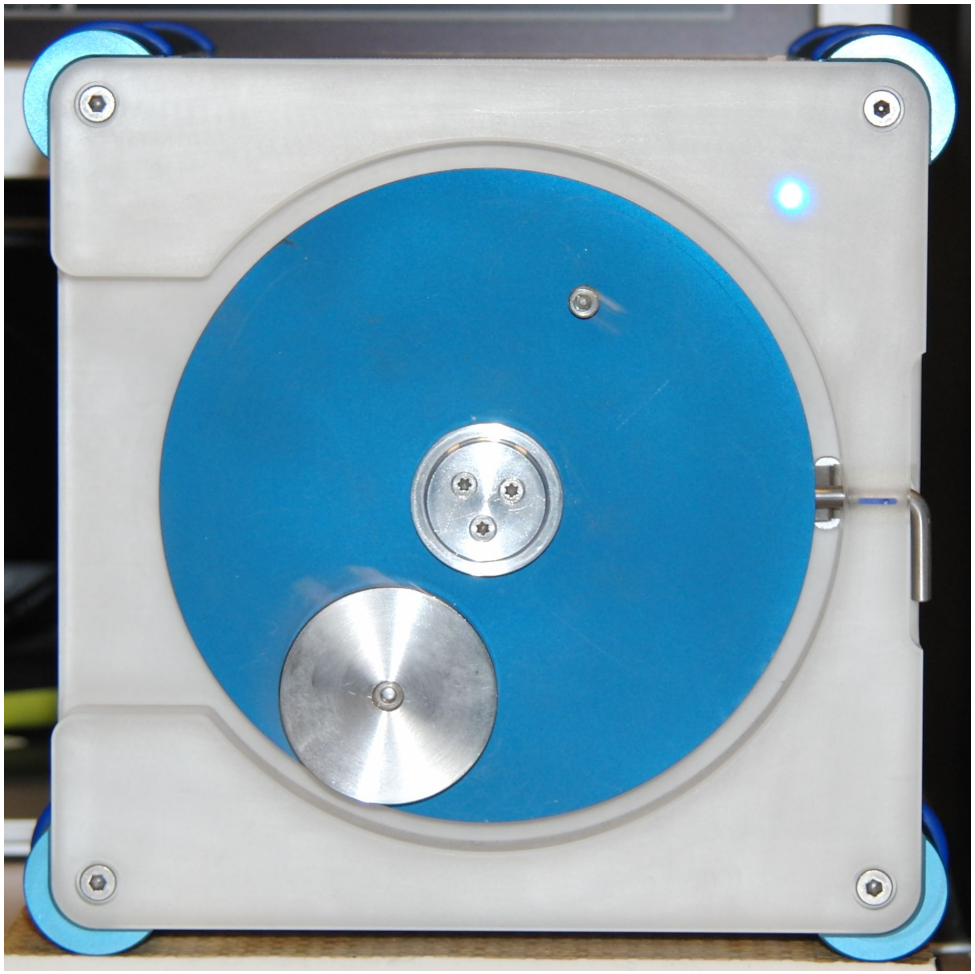


- Reward $-5\phi_1^2 - 0.05\dot{\phi}_1^2 - 5\phi_2^2 - 0.05\dot{\phi}_2^2$
- 4 state dimensions, 2 action dimensions, tile coding
- 3 discretized actions per dimension

Two-link manipulator

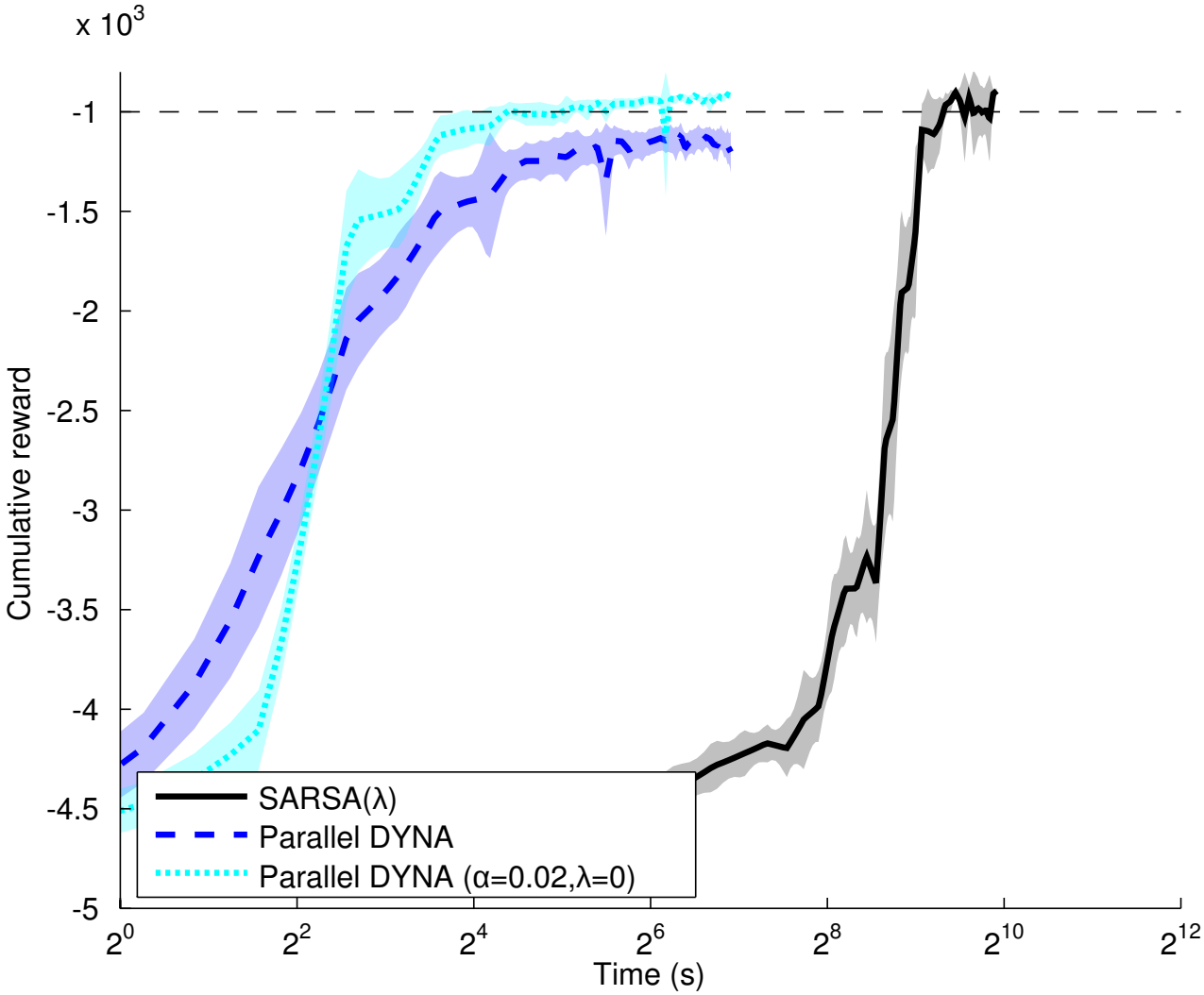


Real-world pendulum

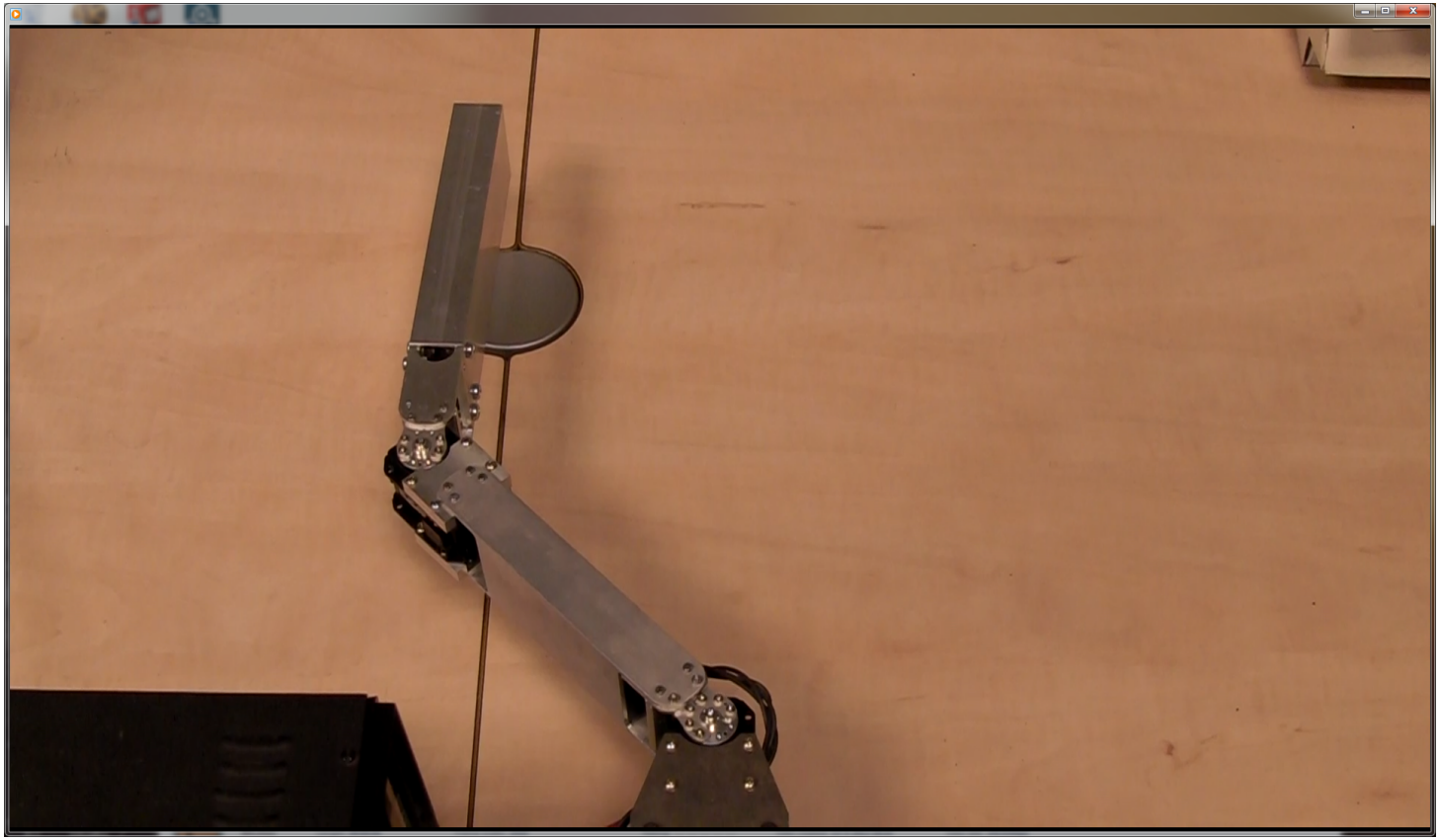


DCSC DC Motor Setup

Real-world pendulum

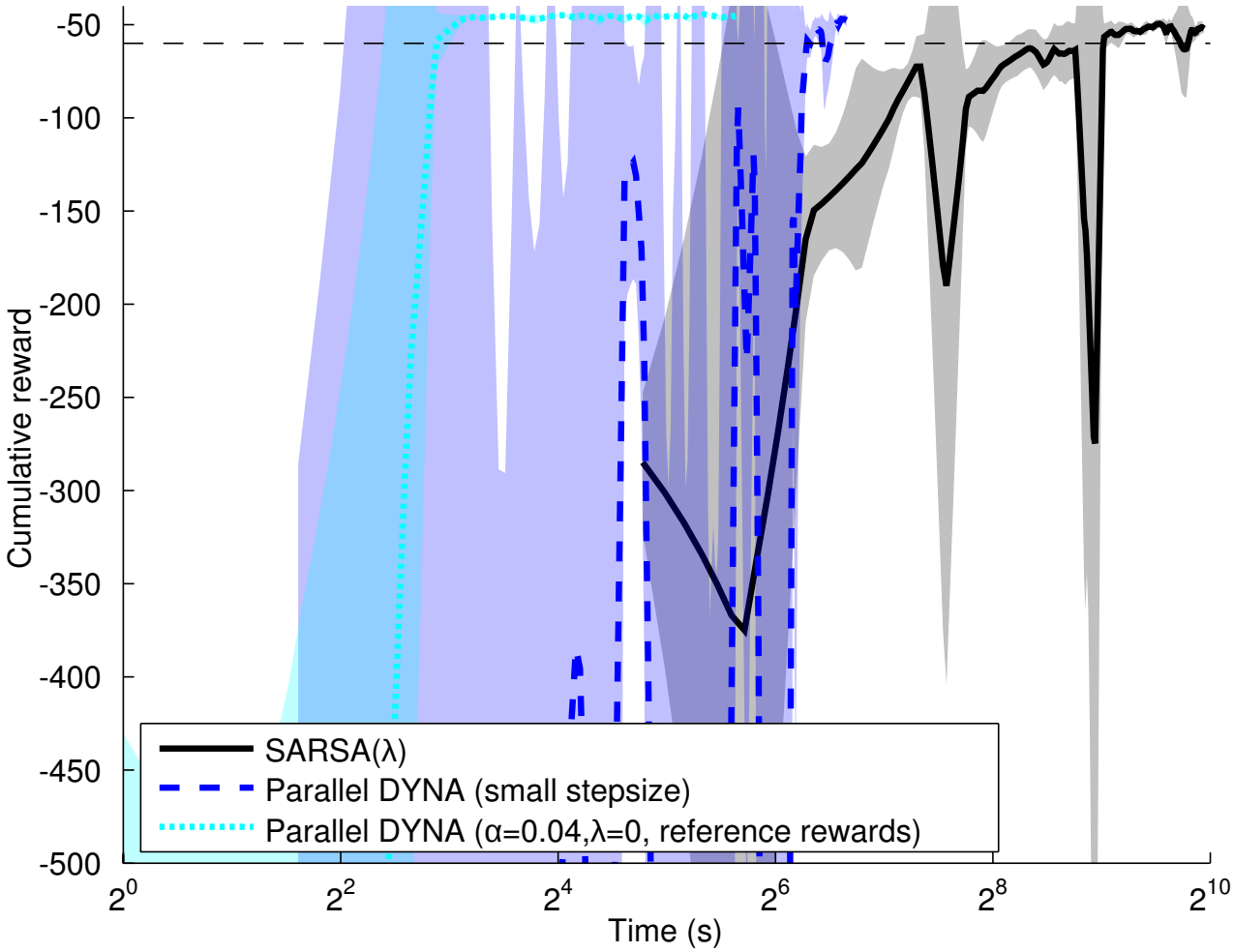


Real-world pick and place



- 2 Dynamixel RX-28 servos
- Move from pick to place

Real-world pick and place



Movie

Outline

Summary

- RL on robots requires reduced sample complexity without increasing computation time.
- Moore's Law won't speed up our algorithms anymore unless they are parallel.
- Traditional parallel reinforcement learning won't work on robots, because we have only one system and it is limited by the control frequency.
- Model learning (parallel DYNA) solves this problem, leading to significant speedups.

The bottleneck is now model learning speed.

Summary

- RL on robots requires reduced sample complexity without increasing computation time.
- Moore's Law won't speed up our algorithms anymore unless they are parallel.
- Traditional parallel reinforcement learning won't work on robots, because we have only one system and it is limited by the control frequency.
- Model learning (parallel DYNA) solves this problem, leading to significant speedups.

The bottleneck is now model learning speed.

Summary

- RL on robots requires reduced sample complexity without increasing computation time.
- Moore's Law won't speed up our algorithms anymore unless they are parallel.
- Traditional parallel reinforcement learning won't work on robots, because we have only one system and it is limited by the control frequency.
- Model learning (parallel DYNA) solves this problem, leading to significant speedups.

The bottleneck is now model learning speed.

Summary

- RL on robots requires reduced sample complexity without increasing computation time.
- Moore's Law won't speed up our algorithms anymore unless they are parallel.
- Traditional parallel reinforcement learning won't work on robots, because we have only one system and it is limited by the control frequency.
- Model learning (parallel DYNA) solves this problem, leading to significant speedups.

The bottleneck is now model learning speed.

Summary

- RL on robots requires reduced sample complexity without increasing computation time.
- Moore's Law won't speed up our algorithms anymore unless they are parallel.
- Traditional parallel reinforcement learning won't work on robots, because we have only one system and it is limited by the control frequency.
- Model learning (parallel DYNA) solves this problem, leading to significant speedups.

The bottleneck is now model learning speed.

Experimental data

Task	Setting	Rise time (s)	Perf	Speedup
Pendulum (sim)	SARSA(λ)	650	-848	
	PDYNA	9.1	-935	72x
	PDYNA[1]	12.6	-812	52x
Pendulum (real)	SARSA(λ)	561	-1027	
	PDYNA		-1153	
	PDYNA[1]	28.1	-932	20x
Manipulator (sim)	SARSA(λ)	4521	-76	
	PDYNA	41.8	-81	108x
Manipulator (real)	SARSA(λ)	417	-55	
	PDYNA[1]	73.9	-57	6x
	PDYNA[12]	6.9	-45	60x

[1]Decreased learning rate, no eligibility traces [2]Using reference model for rewards