# Parallel Real-Time Reinforcement Learning

Wouter Caarls

Department of Biomechanical Engineering
Delft University of Technology

6th Barbados Workshop on Reinforcement Learning
March 21, 2011

# Why real-time reinforcement learning?

## Use real system dynamics

- No model
- Low fidelity simulation

## Characteristics

- Execute learned policy on real system
  - (short) Action selection deadlines
- Update policy while executing
  - Only soft deadlines, but learning is slow if policy changes slowly

Both require fast computation.

# Why real-time reinforcement learning?

## Use real system dynamics

- No model
- Low fidelity simulation

## Characteristics

- Execute learned policy on real system
  - (short) Action selection deadlines
- Update policy while executing
  - Only soft deadlines, but learning is slow if policy changes slowly

Both require fast computation.

# Why real-time reinforcement learning?

## Use real system dynamics

- No model
- Low fidelity simulation

## Characteristics

- Execute learned policy on real system
  - (short) Action selection deadlines
- Update policy while executing
  - Only soft deadlines, but learning is slow if policy changes slowly

Both require fast computation.

# Why real-time reinforcement learning?

## Use real system dynamics

- No model
- Low fidelity simulation

## Characteristics

- Execute learned policy on real system
  - (short) Action selection deadlines
- Update policy while executing
  - Only soft deadlines, but learning is slow if policy changes slowly

Both require fast computation.

# Outline

1. Computation
   - Complexity
   - Moore's law
   - The many-core era
2. Parallel reinforcement learning
   - Environment
   - Updates
   - Model parallelism
3. Preliminary results
   - Model learning
   - Value function decomposition
4. Summary

# Outline

# Complexity of reinforcement learning

- Sample complexity
  - Time
    - Real time
    - Expensive simulation
  - Damage, hard to set initial condition, etc.
- Computational complexity
  - (batch) Updates
  - Model (construction, readout)

Samples are scarce, but computation is abundant. Data-efficient algorithms trade off sample complexity for computational complexity.

# Complexity of reinforcement learning

- Sample complexity
  - Time
    - Real time
    - Expensive simulation
  - Damage, hard to set initial condition, etc.
- Computational complexity
  - (batch) Updates
  - Model (construction, readout)

Samples are scarce, but computation is abundant. Data-efficient algorithms trade off sample complexity for computational complexity.

# Complexity of reinforcement learning

- Sample complexity
  - Time
    - Real time
    - Expensive simulation
  - Damage, hard to set initial condition, etc.
- Computational complexity
  - (batch) Updates
  - Model (construction, readout)

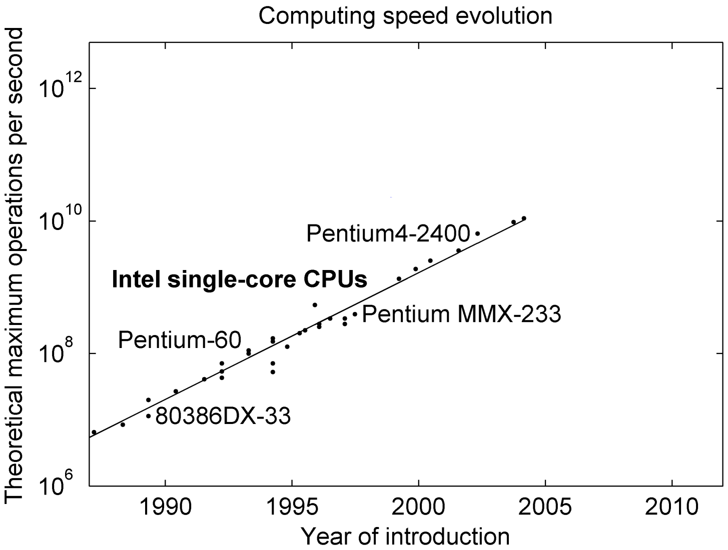
Samples are scarce, but computation is abundant. Data-efficient algorithms trade off sample complexity for computational complexity.

# Complexity of reinforcement learning

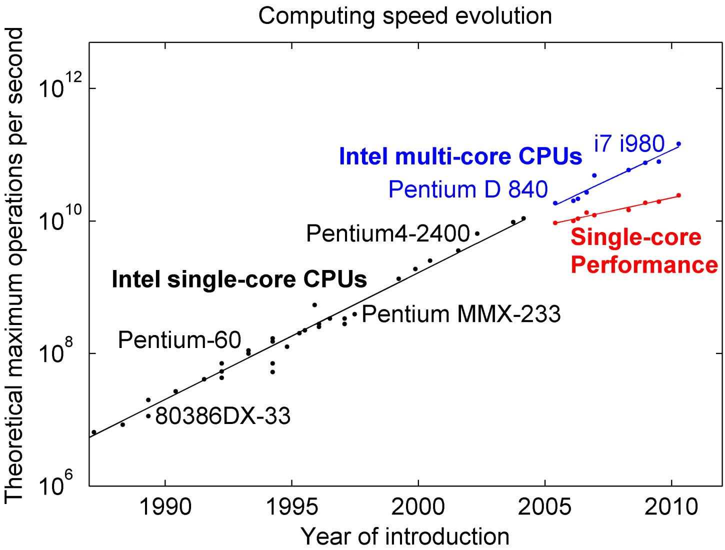
- Sample complexity
  - Time
    - Real time
    - Expensive simulation
  - Damage, hard to set initial condition, etc.
- Computational complexity
  - (batch) Updates
  - Model (construction, readout)

Samples are scarce, but computation is abundant. Data-efficient algorithms trade off sample complexity for computational complexity.

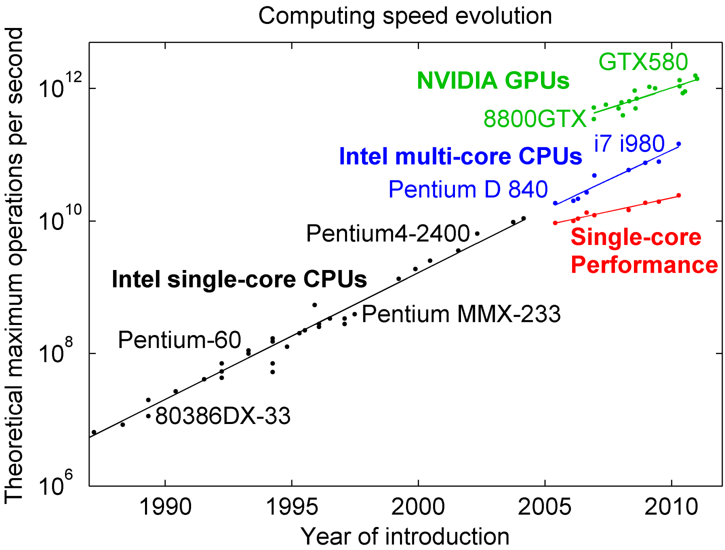
# Moore's law


Computing speed evolution
Theoretical maximum operations per second
$10^{12}$
$10^{10}$
$10^{8}$
$10^{6}$
1990
1995
2000
2005
2010
Year of introduction
Intel single-core CPUs
Pentium4-2400
Pentium MMX-233
Pentium-60
80386DX-33

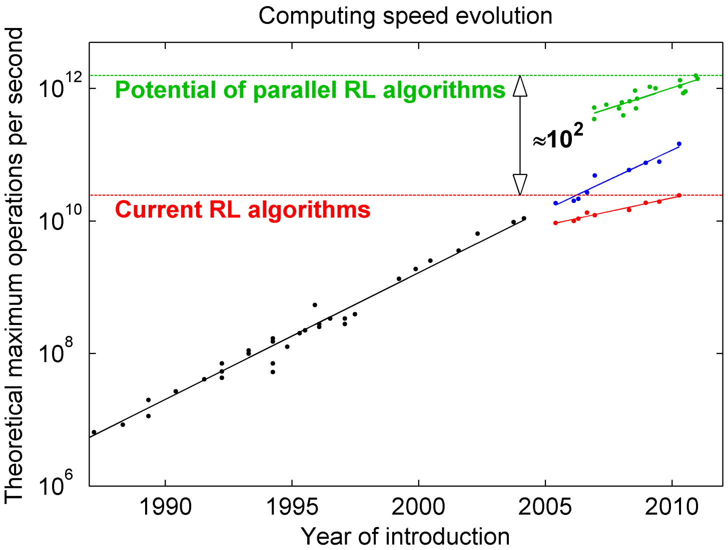
# Moore's law


Computing speed evolution
Theoretical maximum operations per second
Year of introduction
10^12
10^10
10^8
10^6
1990
1995
2000
2005
2010
Intel multi-core CPUs
i7 i980
Pentium D 840
Pentium4-2400
Single-core Performance
Intel single-core CPUs
Pentium MMX-233
Pentium-60
80386DX-33

# Moore's law


Computing speed evolution
Theoretical maximum operations per second
Year of introduction
$10^{6}$
$10^{8}$
$10^{10}$
$10^{12}$
1990
1995
2000
2005
2010
NVIDIA GPUs
GTX580
8800GTX
Intel multi-core CPUs
i7 i980
Pentium D 840
Single-core Performance
Pentium4-2400
Intel single-core CPUs
Pentium MMX-233
Pentium-60
80386DX-33

# Moore's law


Computing speed evolution
Theoretical maximum operations per second
$10^{12}$
$10^{10}$
$10^{8}$
$10^{6}$
Potential of parallel RL algorithms
≈$10^2$
Current RL algorithms
1990
1995
2000
2005
2010
Year of introduction

# Abundance of computation

Moore's law no longer speeds up sequential processes. In order to take advantage of the abundance of computation, algorithms must be parallel. Massively parallel.

# Many-core architectures

- Computation
  - Hundreds to thousands of processors
  - Individually less capable
  - Partial SIMD
- Data access
  - Not significantly more memory than single processors
  - Distributed
  - Non-local access has high latency
- Sequential operation count is a poor predictor for performance
  - $O(n^2)$ may be faster at $n = 1000$ than $O(n)$

# Many-core architectures

- Computation
  - Hundreds to thousands of processors
  - Individually less capable
  - Partial SIMD
- Data access
  - Not significantly more memory than single processors
  - Distributed
  - Non-local access has high latency
- Sequential operation count is a poor predictor for performance
  - $O(n^2)$ may be faster at $n = 1000$ than $O(n)$

# Many-core architectures

- Computation
  - Hundreds to thousands of processors
  - Individually less capable
  - Partial SIMD
- Data access
  - Not significantly more memory than single processors
  - Distributed
  - Non-local access has high latency
- Sequential operation count is a poor predictor for performance
  - $O(n^2)$ may be faster at $n = 1000$ than $O(n)$

# Outline

# Parallelism in the environment

- Multiple systems
  - Evolutionary robotics
  - Single value function
  - Multiple value functions with exchange
- Concurrent simulation
  - Integrate heterogeneous updates
- Parallel simulation
  - GPU physics acceleration


Figure: Kohl and Stone, 2004.

# Parallelism in the environment

- Multiple systems
  - Evolutionary robotics
  - Single value function
  - Multiple value functions with exchange
- Concurrent simulation
  - Integrate heterogeneous updates
- Parallel simulation
  - GPU physics acceleration


Figure: Kohl and Stone, 2004.

# Parallelism in the environment

- Multiple systems
  - Evolutionary robotics
  - Single value function
  - Multiple value functions with exchange
- Concurrent simulation
  - Integrate heterogeneous updates
- Parallel simulation
  - GPU physics acceleration


Figure: Kohl and Stone, 2004.

# Parallel updates

- Single value function
  - Domain decomposition
  - Needs multiple update sources
    - Experience replay
    - Multiple systems
    - Model
- Multiple value functions
  - Learn different things
- Batch
  - Parallel backpropagation
  - Parallel LSPI

# Parallel updates

- Single value function
  - Domain decomposition
  - Needs multiple update sources
    - Experience replay
    - Multiple systems
    - Model
- Multiple value functions
  - Learn different things
- Batch
  - Parallel backpropagation
  - Parallel LSPI

# Parallel updates

- Single value function
  - Domain decomposition
  - Needs multiple update sources
    - Experience replay
    - Multiple systems
    - Model
- Multiple value functions
  - Learn different things
- Batch
  - Parallel backpropagation
  - Parallel LSPI

# Parallel model readout

Depending on the model, construction or readout can be parallel.

Locally linear regression

- Find k nearest neighbors
- Fit linear least-squares model

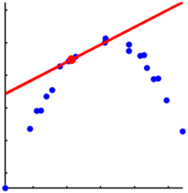
Both can be efficiently parallelized on GPUs.

# Parallel model readout

Depending on the model, construction or readout can be parallel.

## Locally linear regression

- Find k nearest neighbors
- Fit linear least-squares model



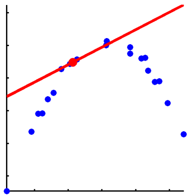
Both can be efficiently parallelized on GPUs.

# Parallel model readout

Depending on the model, construction or readout can be parallel.

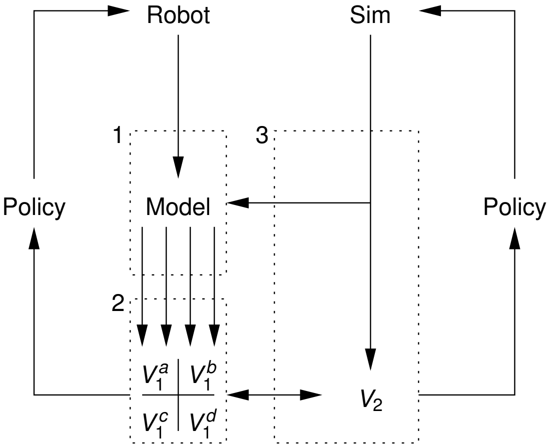
### Locally linear regression

- Find k nearest neighbors
- Fit linear least-squares model



Both can be efficiently parallelized on GPUs.

# Global scheme



# Outline

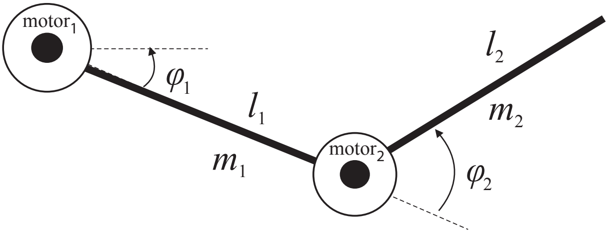
# Model learning

- Learn a model from interactions with the real world
- Use computational abundance to perform many model updates
  - Random
  - Prioritized sweeping
  - Sequential (imitate real trials)
- Direct updates become insignificant

# Model learning

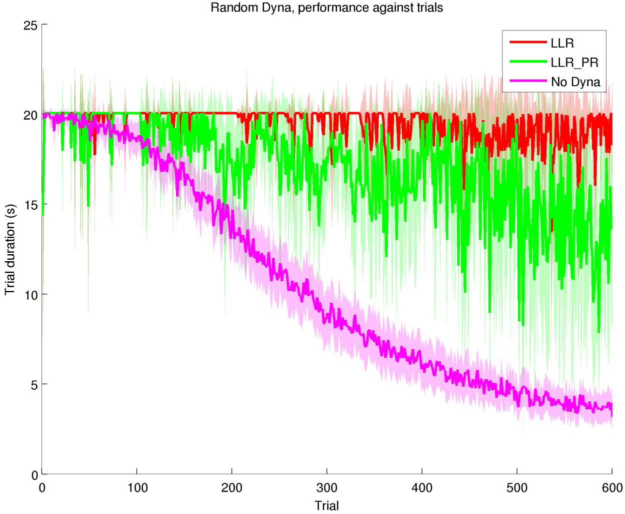
- Learn a model from interactions with the real world
- Use computational abundance to perform many model updates
  - Random
  - Prioritized sweeping
  - Sequential (imitate real trials)
- Direct updates become insignificant

# Two-link manipulator

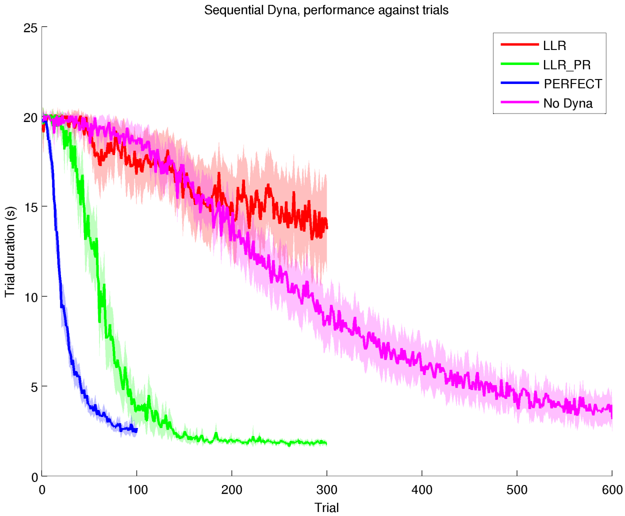



- Reward & termination when $\phi_1 = \phi_2 = \dot{\phi}_1 = \dot{\phi}_2 = 0$
- Constant time penalty
- 4 state dimensions, 2 action dimensions, tile coding
- 5 discretized actions per dimension
- SARSA

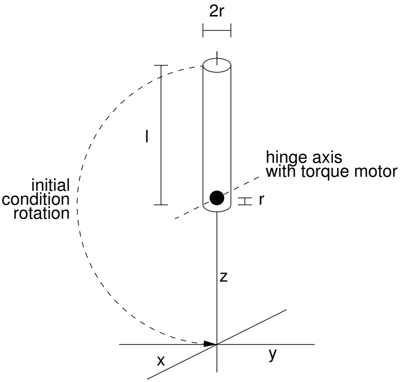
# Random dyna, n=10000

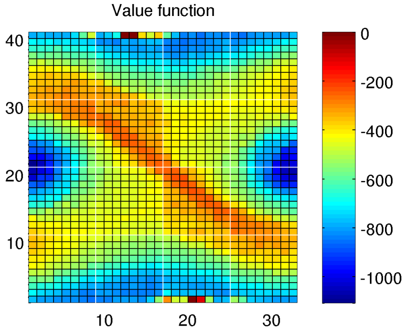


# Sequential dyna, n=10000

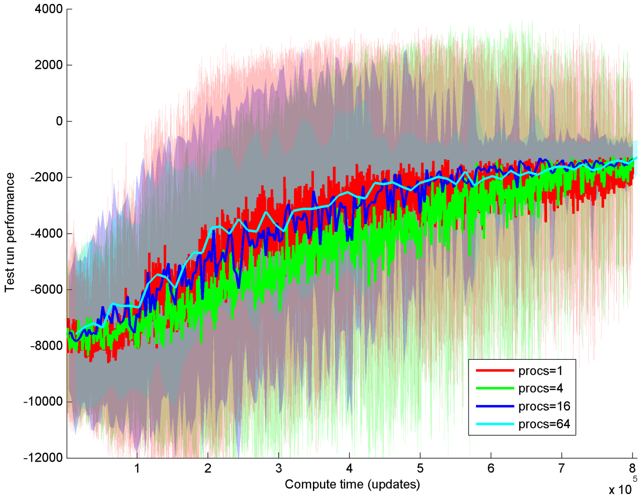


# Pendulum swing-up

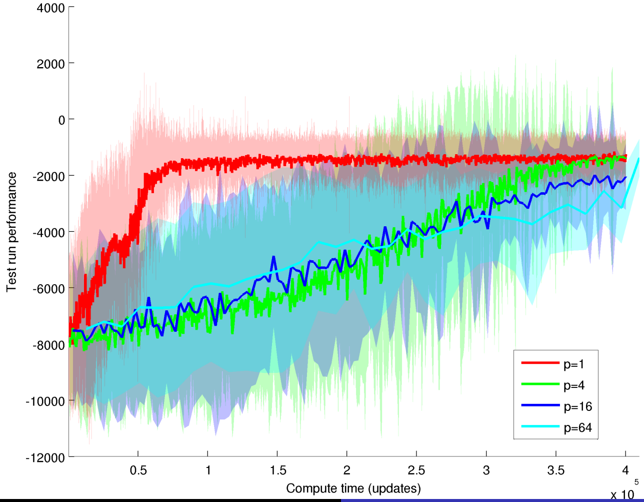


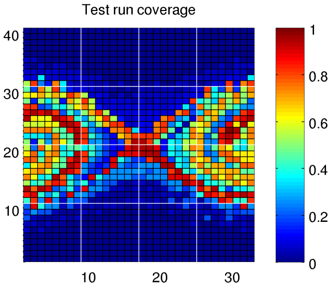
# Performance

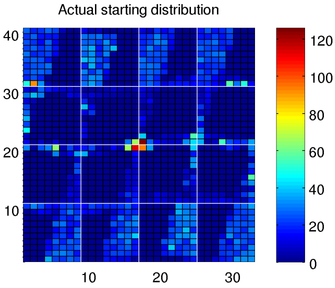


# Real-world performance

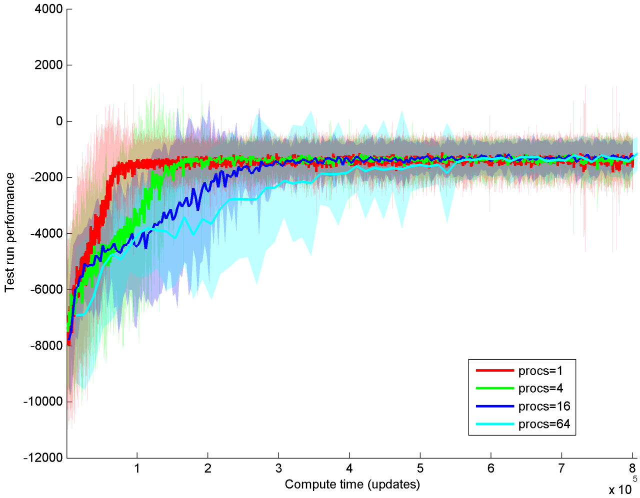


# Starting state distribution



# Testrun-guided starting states



# Outline

# Summary

- Moore's Law won't speed up our algorithms anymore unless they are parallel. Parallelism can be employed at many levels, but must do effective work. Conventional computational complexity is losing validity.
- Outlook
  - Find parallelism in current algorithms.
  - Design new algorithms with parallelism in mind.