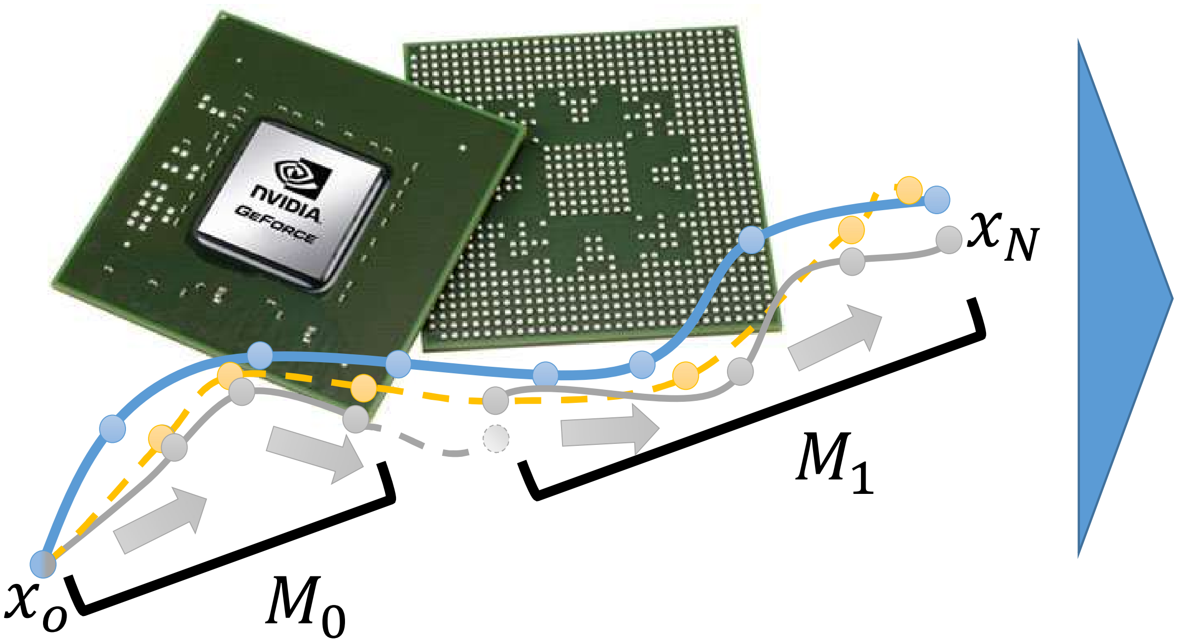
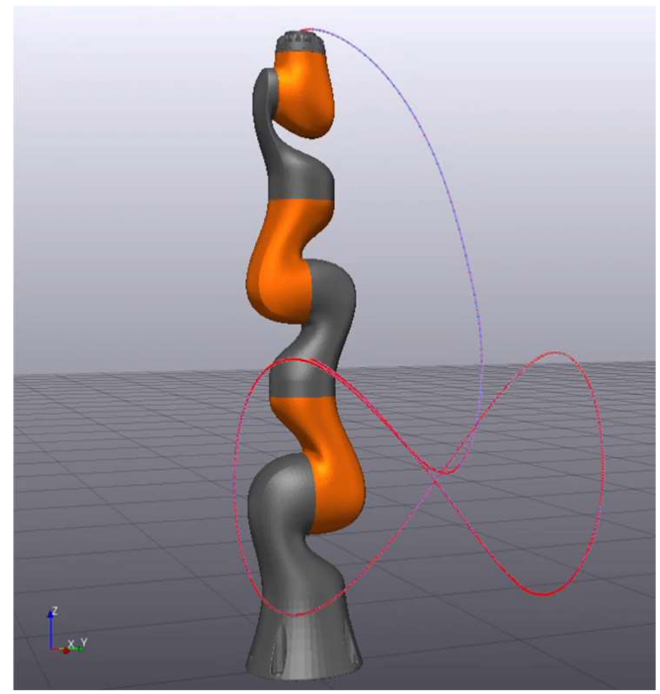
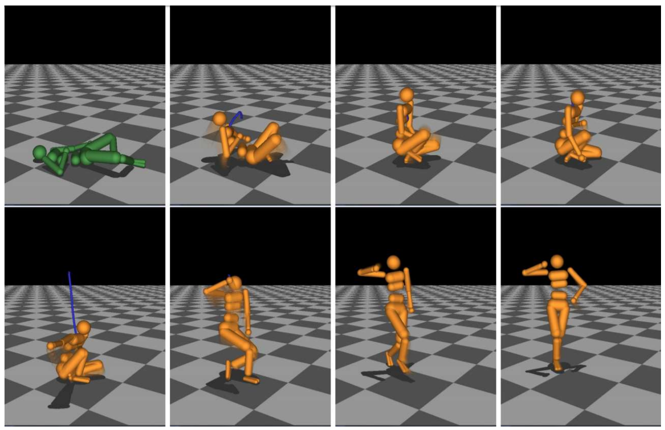
A Performance Analysis of Parallel Differential Dynamic Programming on a GPU



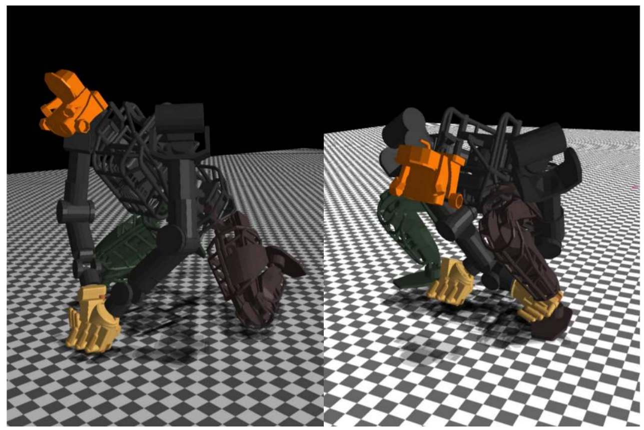


Brian Plancher and Scott Kuindersma
Harvard Agile Robotics Lab

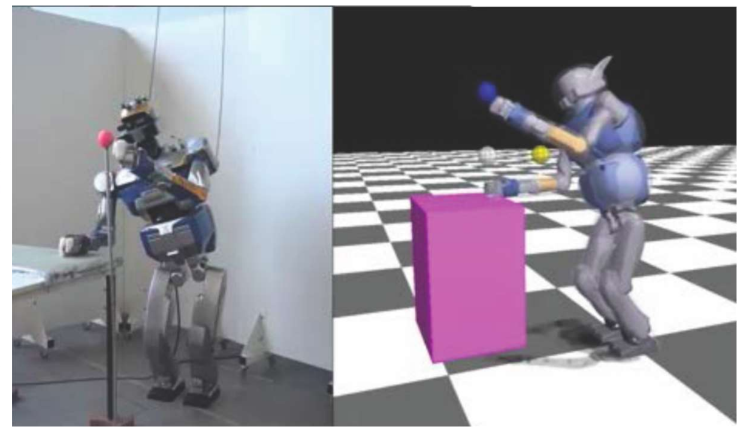
Differential Dynamic Programming (DDP) has shown great promise for Model Predictive Control (MPC)



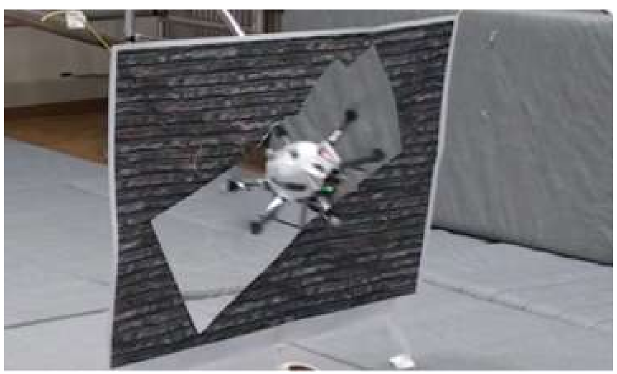
[Tassa et. al. IROS 2012]



[Erez et. al. Humanoids 2013]

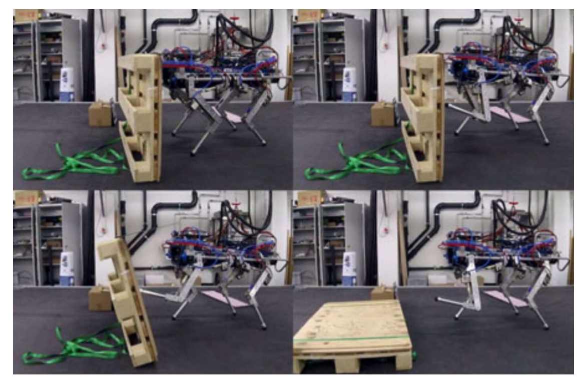


[Koenemann et. al. IROS 2015]



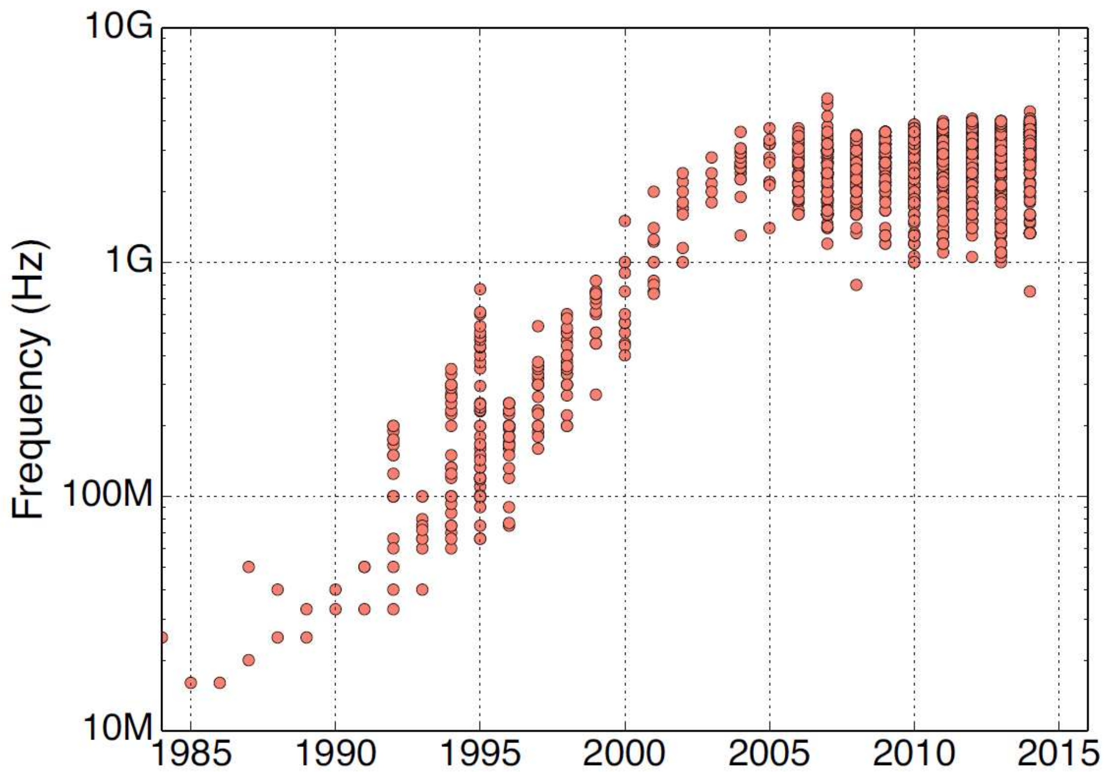
[Neunert et. al. ICRA 2016]

[Neunert et. al. Humanoids 2017]



[Farshidian et. al. IEEE RAL 2017]

Differential Dynamic Programming (DDP) has shown great promise for Model Predictive Control (MPC)



[Shao and Brooks Synthesis Lectures on Computer Architecture 2015]

- Frequency scaling is ending (CPUs aren't getting faster)
- Massive parallelism on GPUs and FPGAs may be a solution for trajectory optimization

A Performance Analysis of Parallel Differential Dynamic Programming on a GPU

- Systematically analyze the **algorithm level** and **instruction level** parallelism in DDP
- Discuss the benefits and trade-offs of **higher degrees of parallelization** (GPU) versus a higher clock rate (CPU)
- Demonstrate **real-time model predictive control** using a GPU based implementation

DDP computes the optimal control via the recursive Bellman equation

$$\min_{x,u} l_f(x_N) + \sum_{k=1}^{N-1} l(x_k, u_k)$$

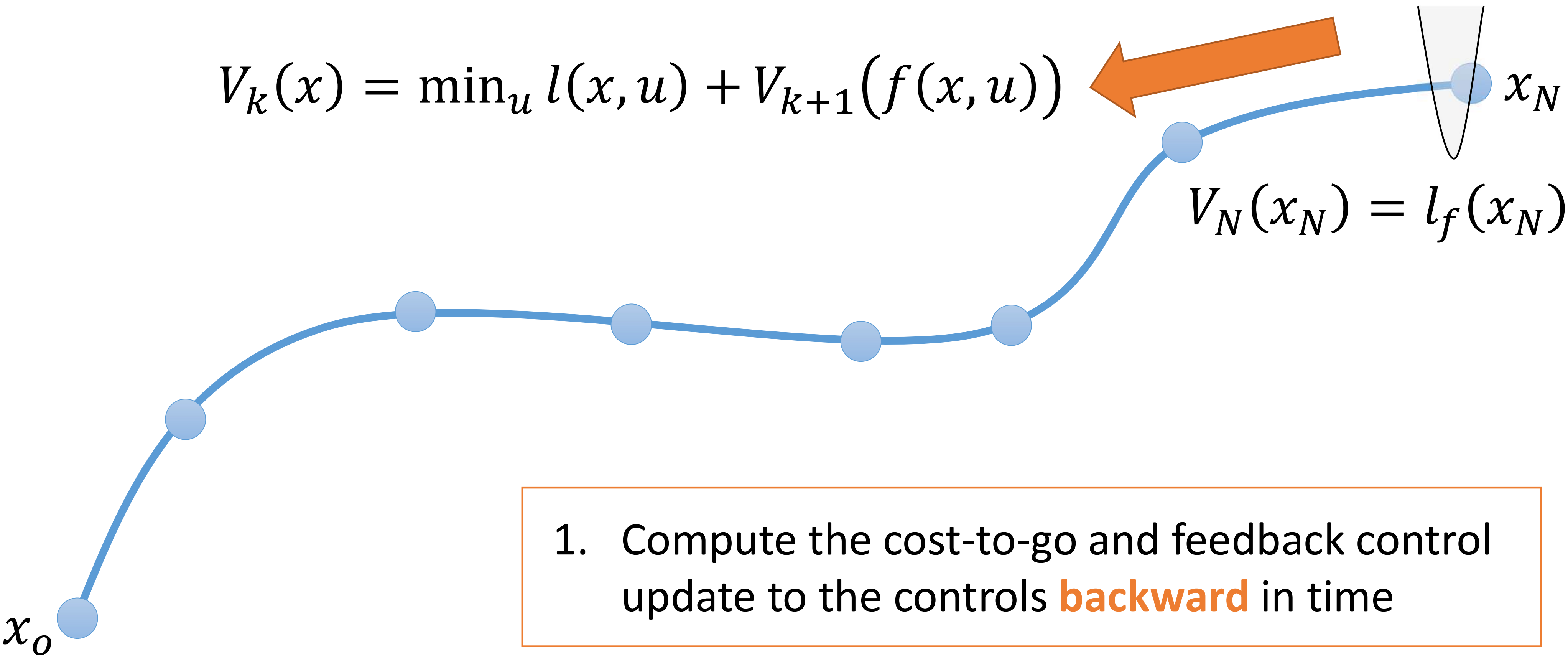
$$\text{s.t. } x_{k+1} = f(x_k, u_k)$$

$$V_k(x) = \min_u l(x,u) + V_{k+1}\big(f(x,u)\big)$$

$$V_N(x_N) = l_f(x_N)$$

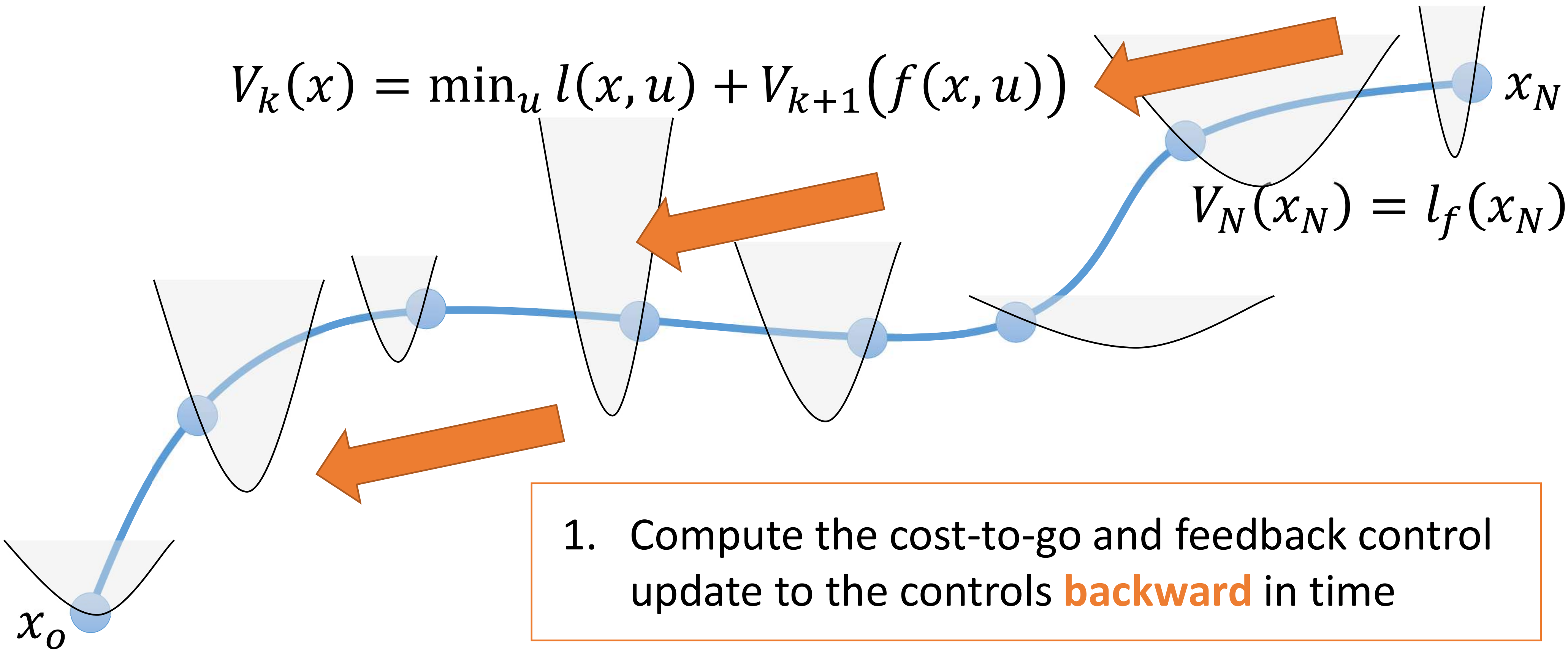
DP methods (DDP/SLQ/iLQR) use quadratic approximations around a nominal trajectory

$$V_k(x) = \min_u l(x,u) + V_{k+1}\big(f(x,u)\big)$$



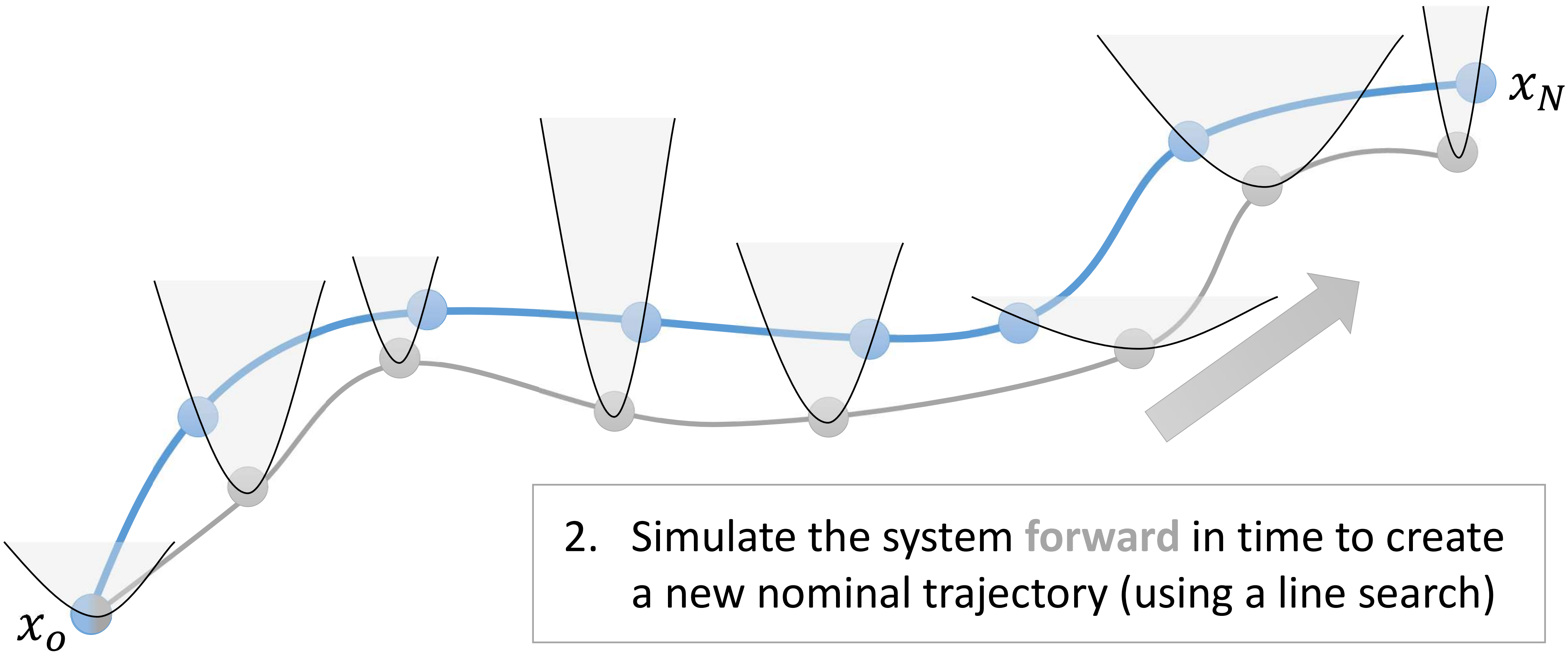
1. Compute the cost-to-go and feedback control update to the controls **backward** in time

DP methods (DDP/SLQ/iLQR) use quadratic approximations around a nominal trajectory



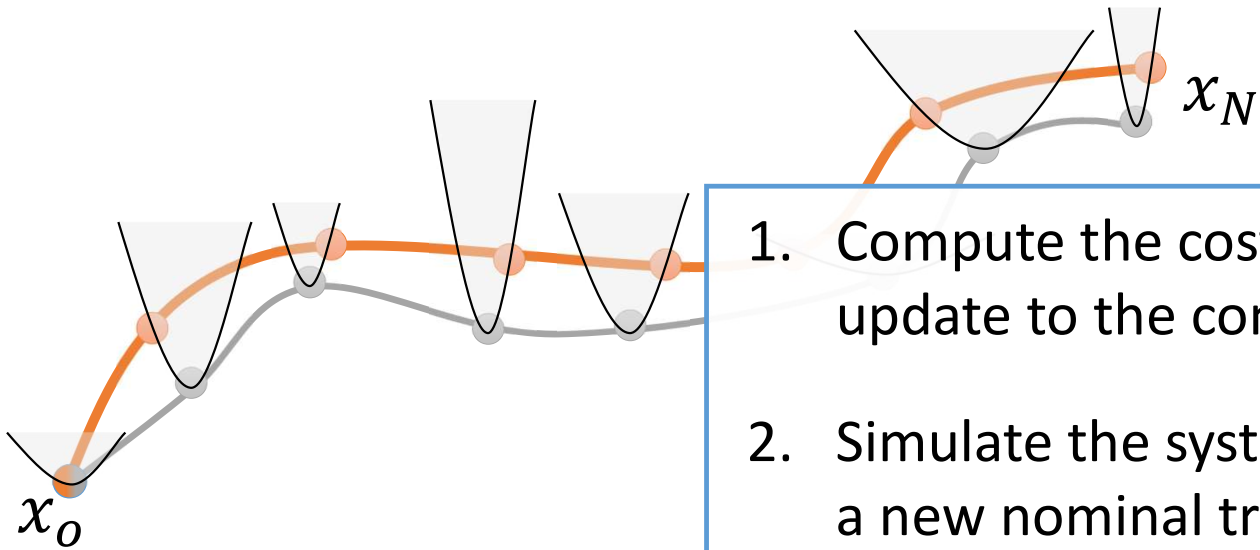
1. Compute the cost-to-go and feedback control update to the controls **backward** in time

DP methods (DDP/SLQ/iLQR) use quadratic approximations around a nominal trajectory



2. Simulate the system **forward** in time to create a new nominal trajectory (using a line search)

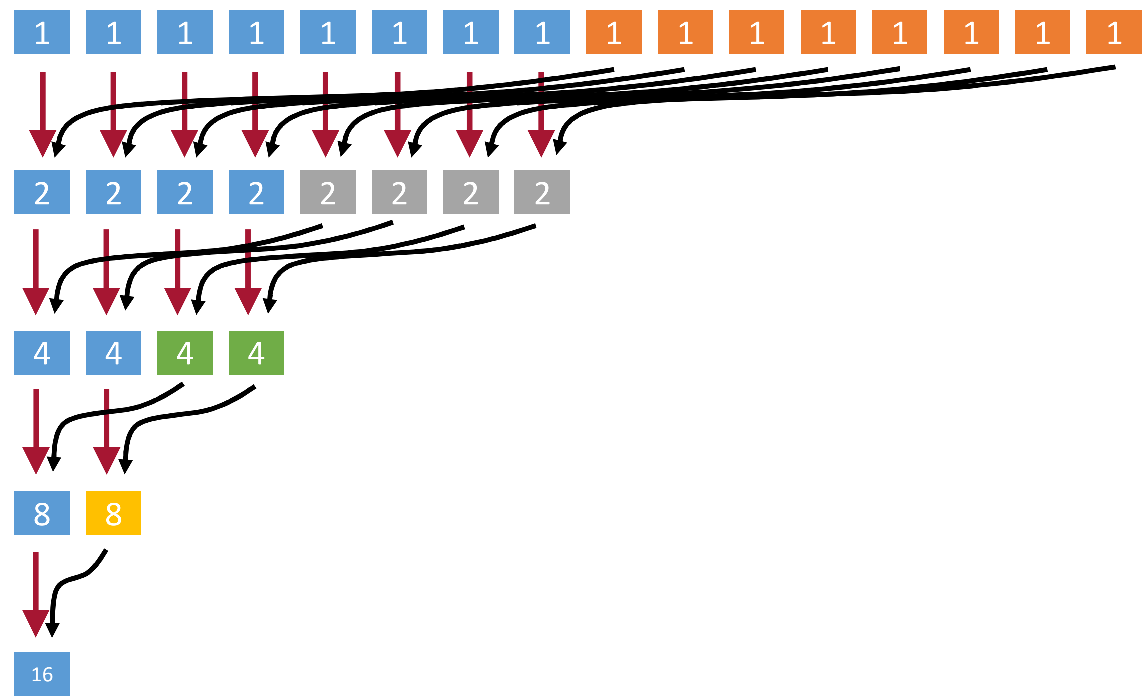
DP methods (DDP/SLQ/iLQR) use quadratic approximations around a nominal trajectory



1. Compute the cost-to-go and feedback control update to the controls **backward** in time
2. Simulate the system **forward** in time to create a new nominal trajectory (using a line search)
3. Taylor approximate the dynamics and cost to **setup for the next iteration**
4. Repeat this process until convergence

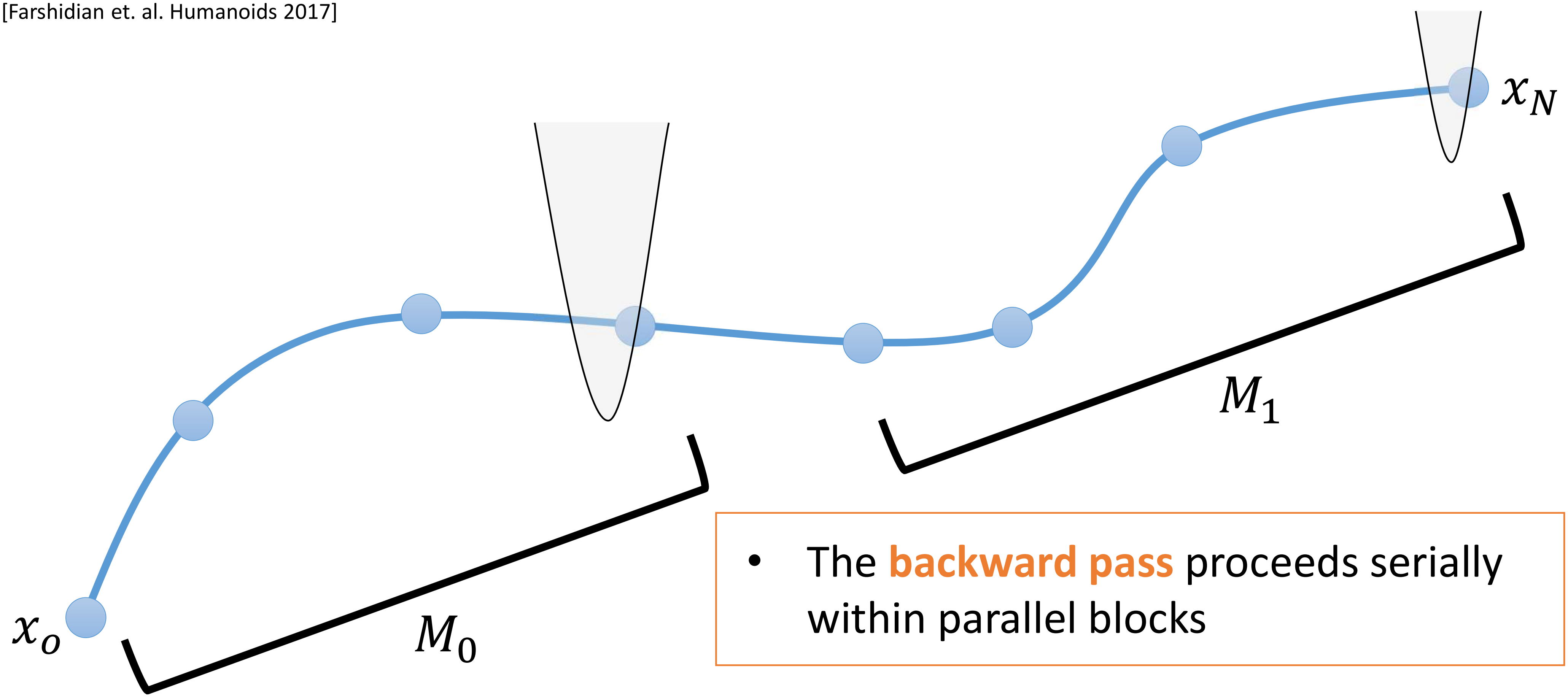
Instruction Level Parallelism parallelizes the standard computations in DDP

1. Taylor Approximations of the Dynamics and Cost
2. Line Search
3. Cost Computation



Breaking the trajectory into M blocks exposes Algorithm Level Parallelism in DDP

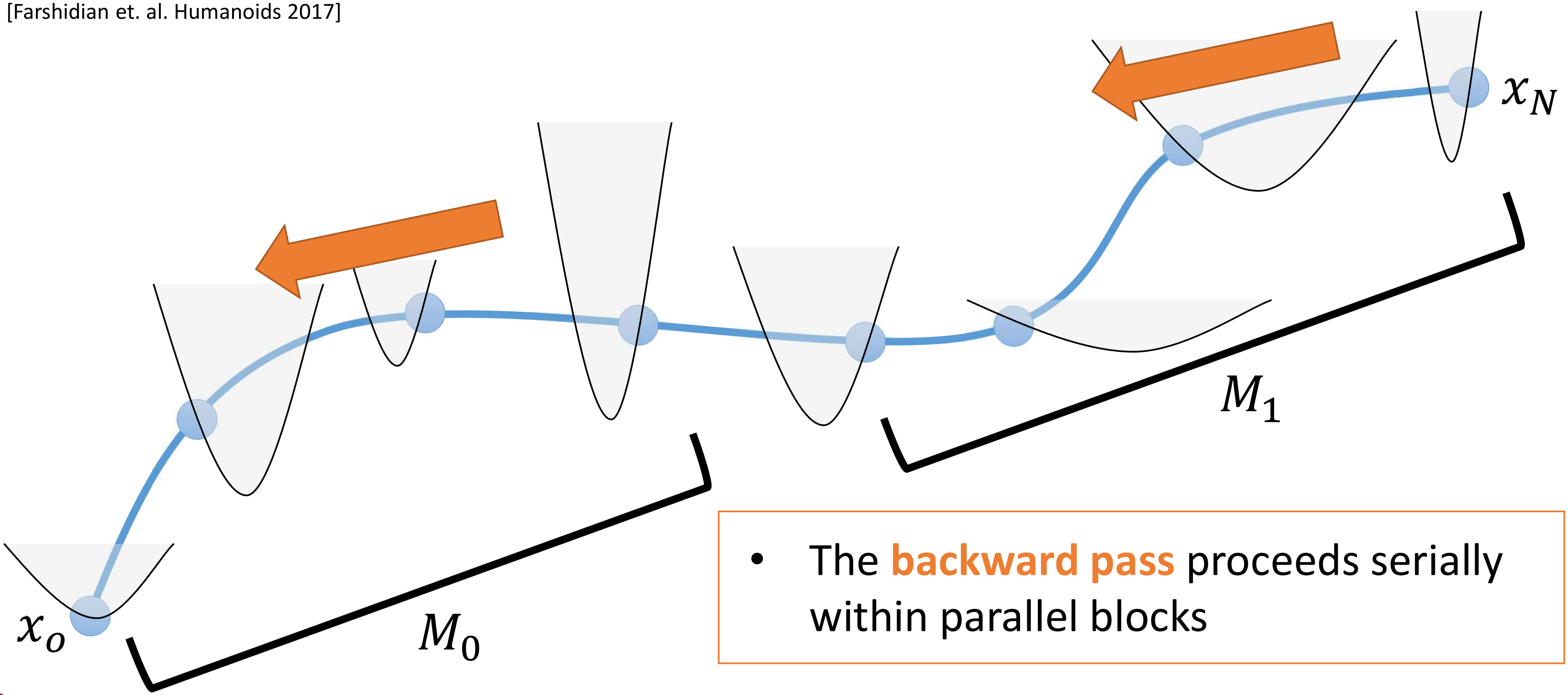
[Farshidian et. al. Humanoids 2017]



- The **backward pass** proceeds serially within parallel blocks

Breaking the trajectory into blocks exposes Algorithm Level Parallelism in DDP

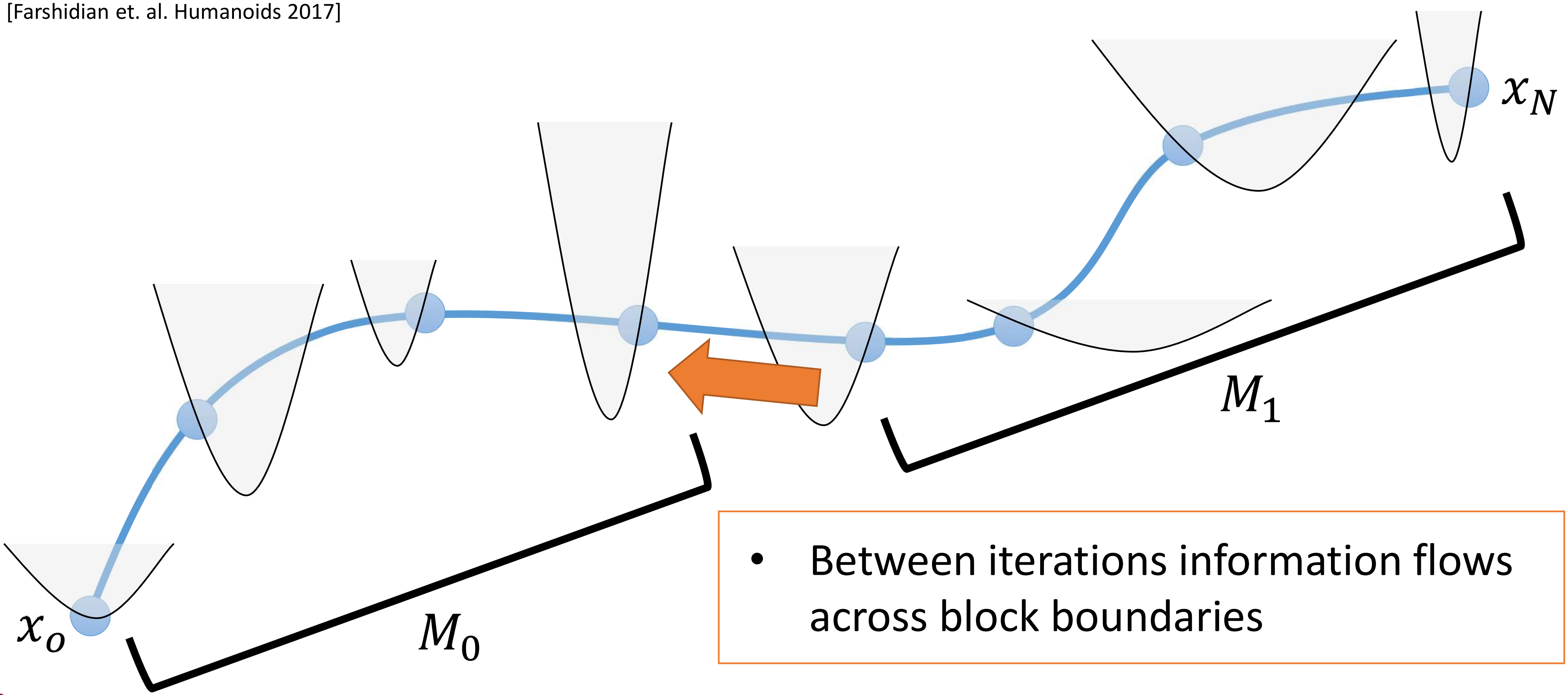
[Farshidian et. al. Humanoids 2017]



- The **backward pass** proceeds serially within parallel blocks

Breaking the trajectory into blocks exposes Algorithm Level Parallelism in DDP

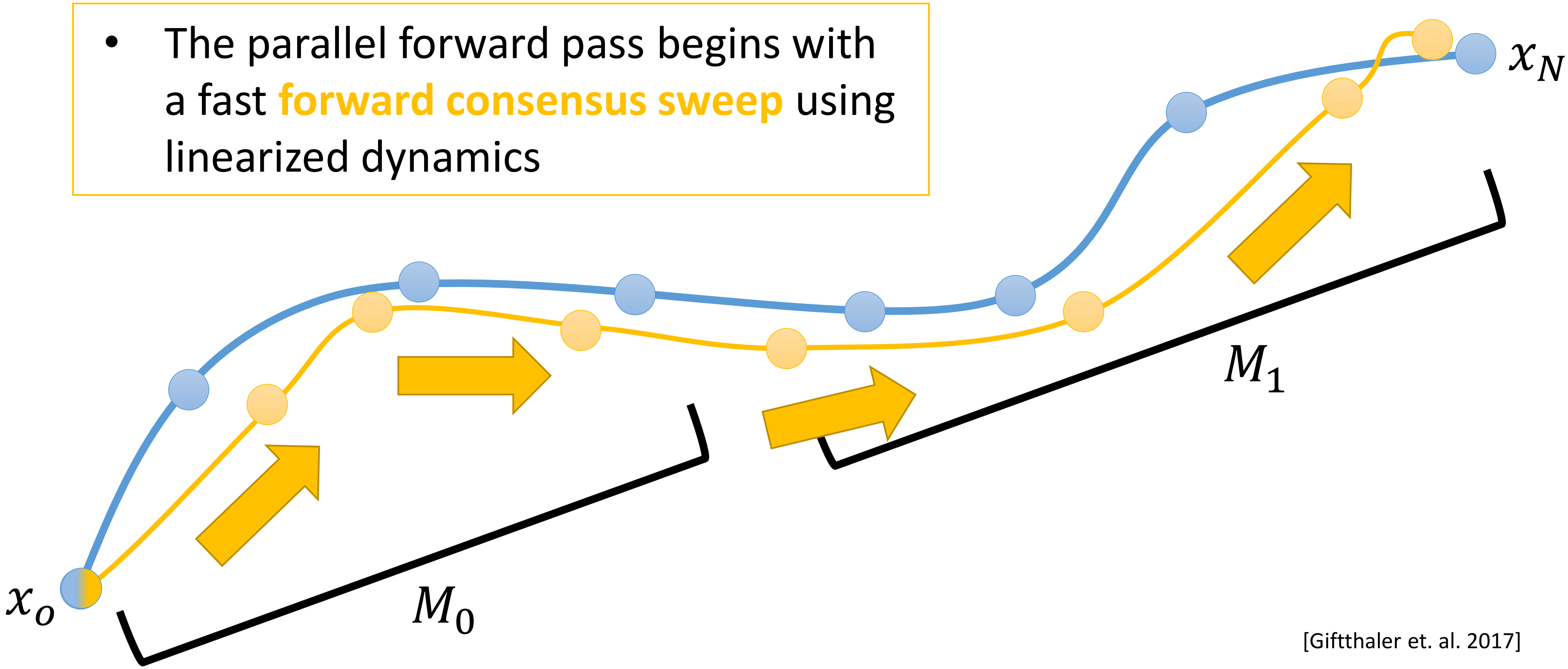
[Farshidian et. al. Humanoids 2017]



- Between iterations information flows across block boundaries

Breaking the trajectory into blocks exposes Algorithm Level Parallelism in DDP

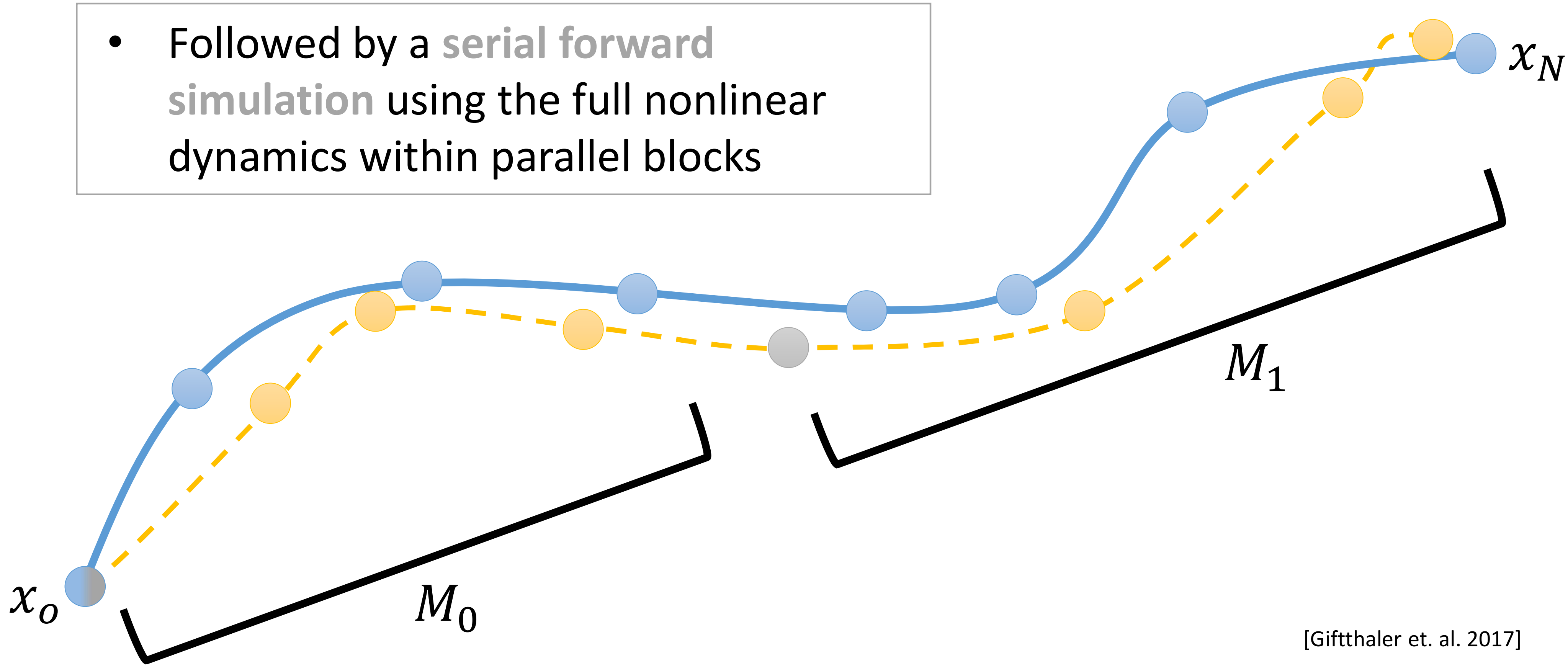
- The parallel forward pass begins with a fast **forward consensus sweep** using linearized dynamics



[Giftthaler et. al. 2017]

Breaking the trajectory into blocks exposes Algorithm Level Parallelism in DDP

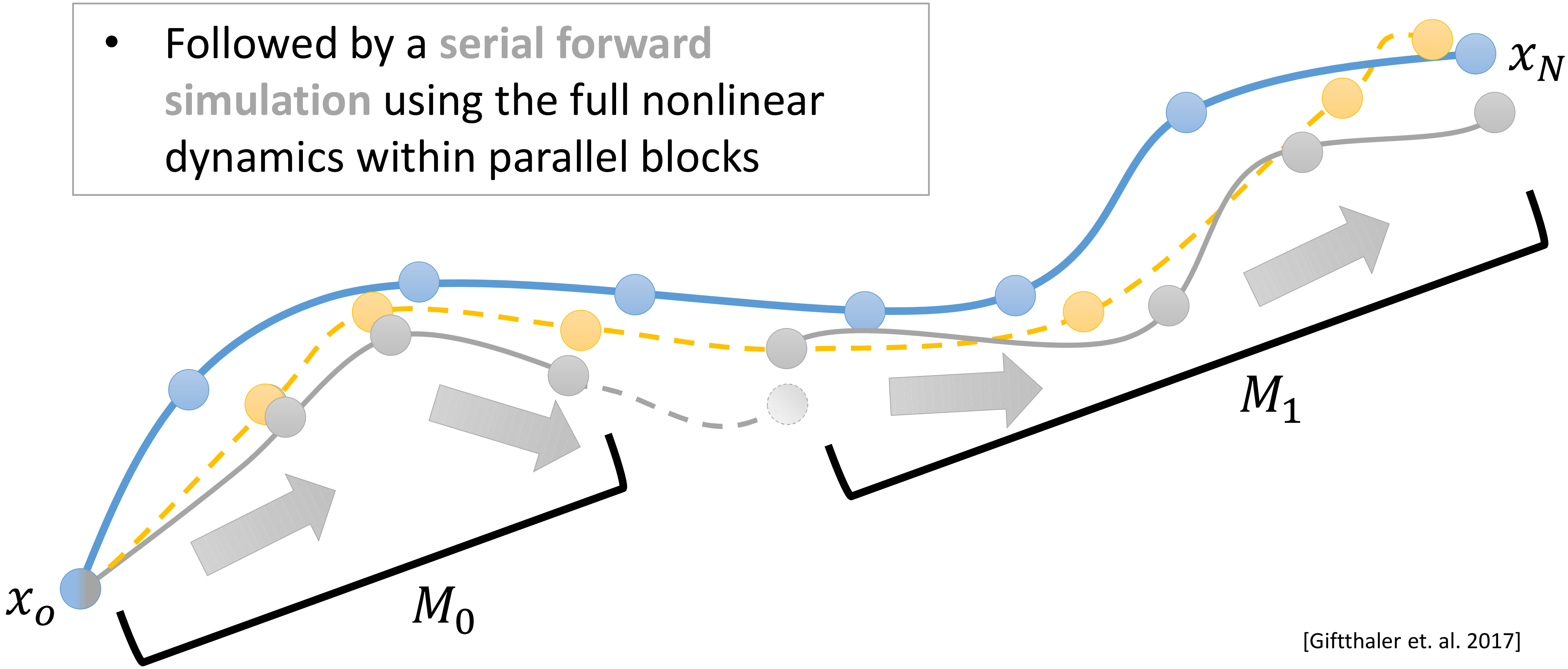
- Followed by a **serial forward simulation** using the full nonlinear dynamics within parallel blocks



[Giftthaler et. al. 2017]

Breaking the trajectory into blocks exposes Algorithm Level Parallelism in DDP

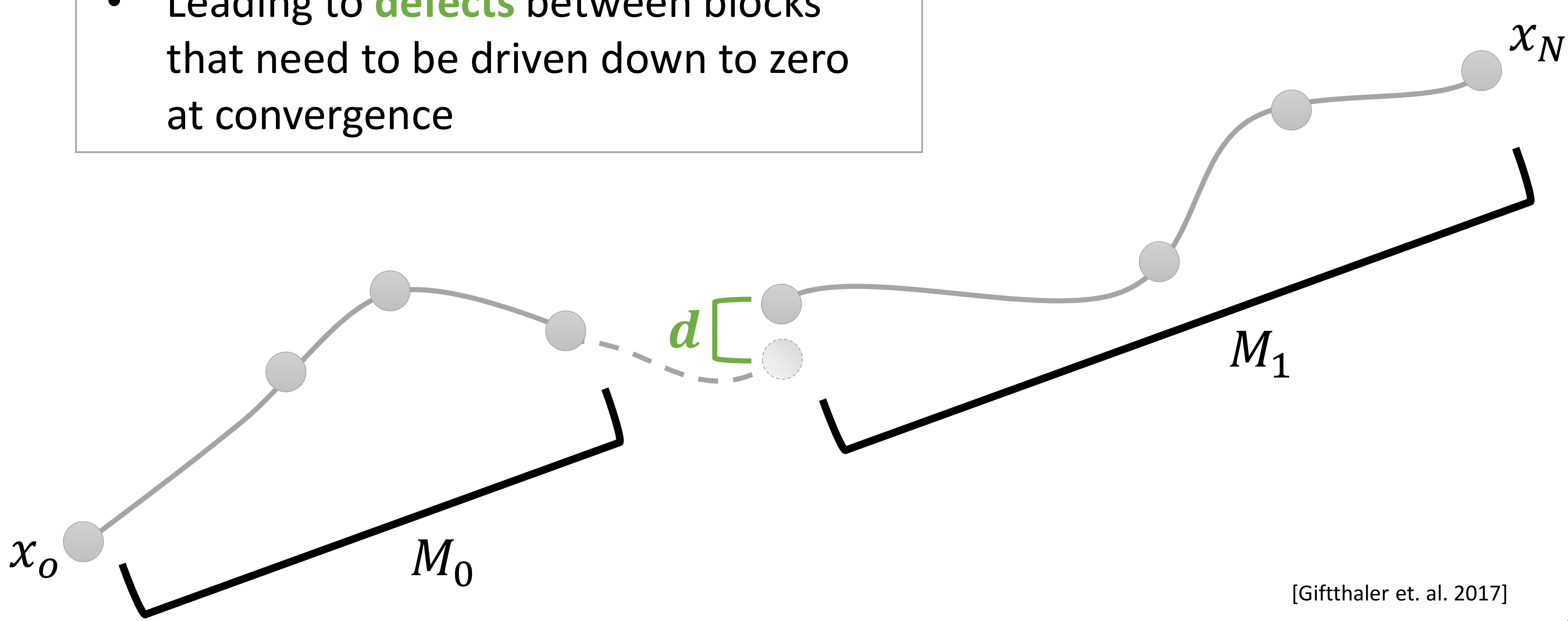
- Followed by a **serial forward simulation** using the full nonlinear dynamics within parallel blocks



[Giftthaler et. al. 2017]

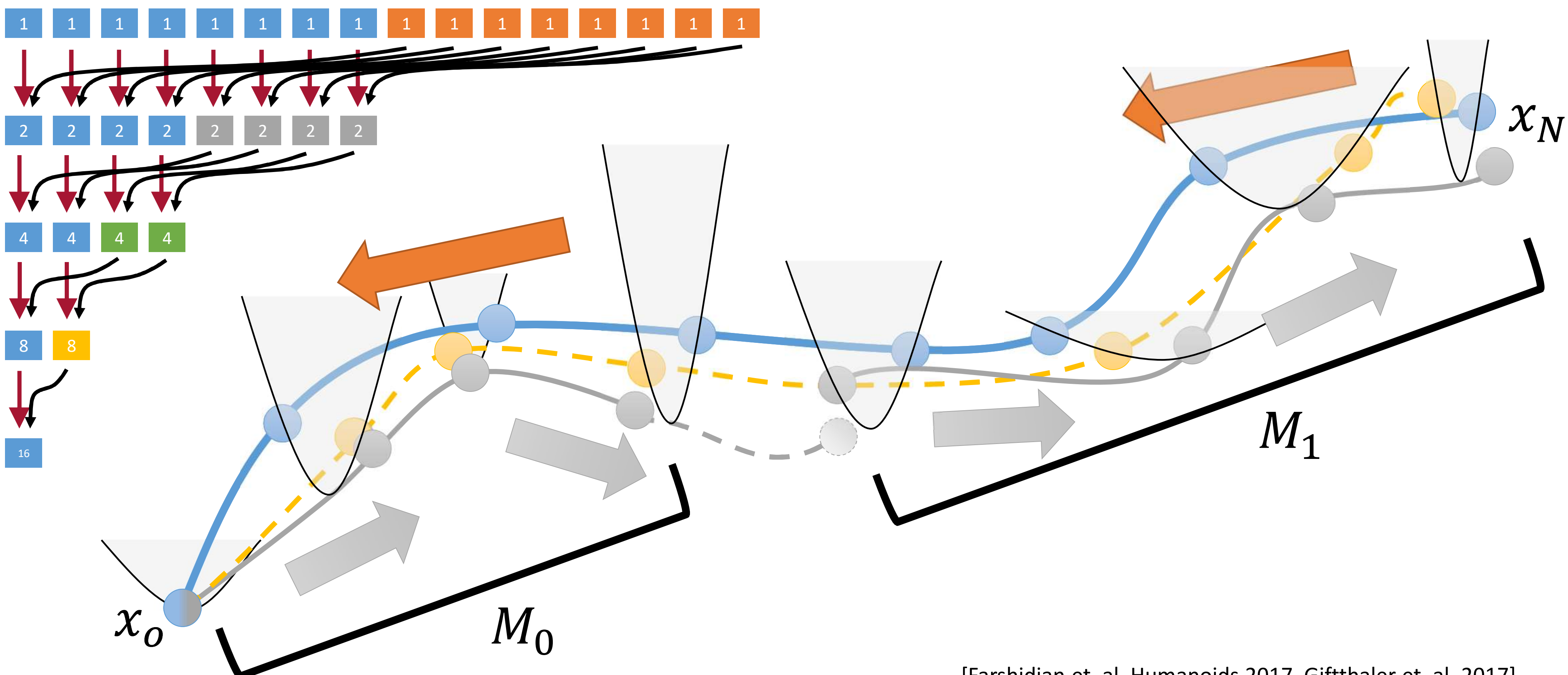
Breaking the trajectory into blocks exposes Algorithm Level Parallelism in DDP

- Leading to **defects** between blocks that need to be driven down to zero at convergence



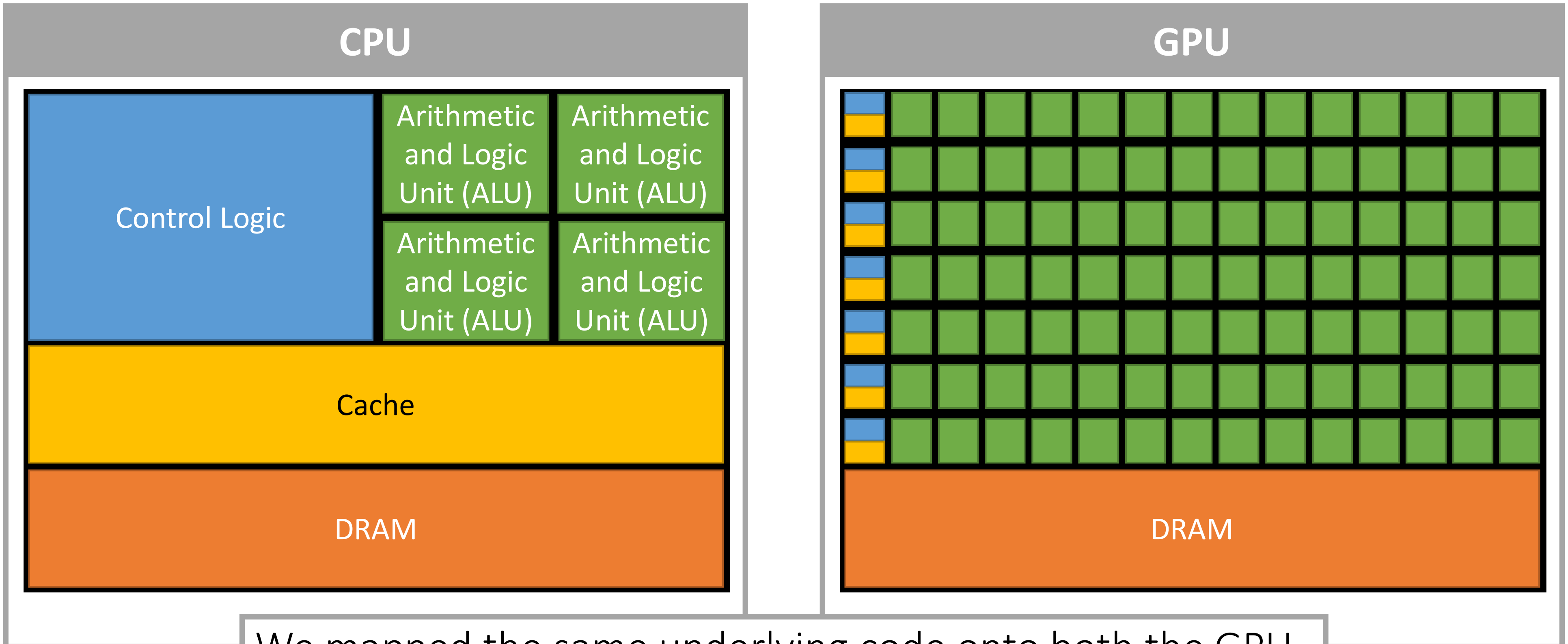
[Giftthaler et. al. 2017]

Using both instruction and algorithm level parallelism leads to the Parallel DDP Algorithm



[Farshidian et. al. Humanoids 2017, Giftthaler et. al. 2017]

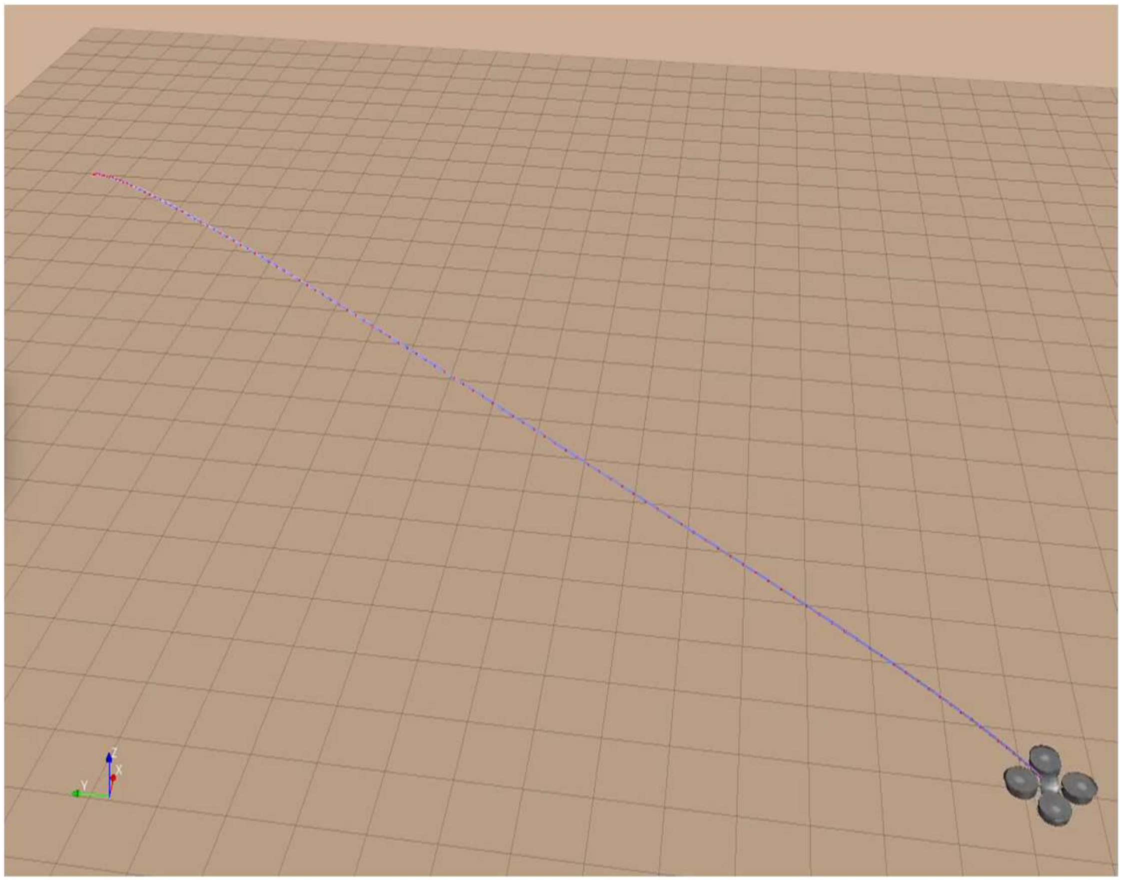
As compared to CPUs, GPUs trade off clock rate, control logic, and cache size for ALU operations

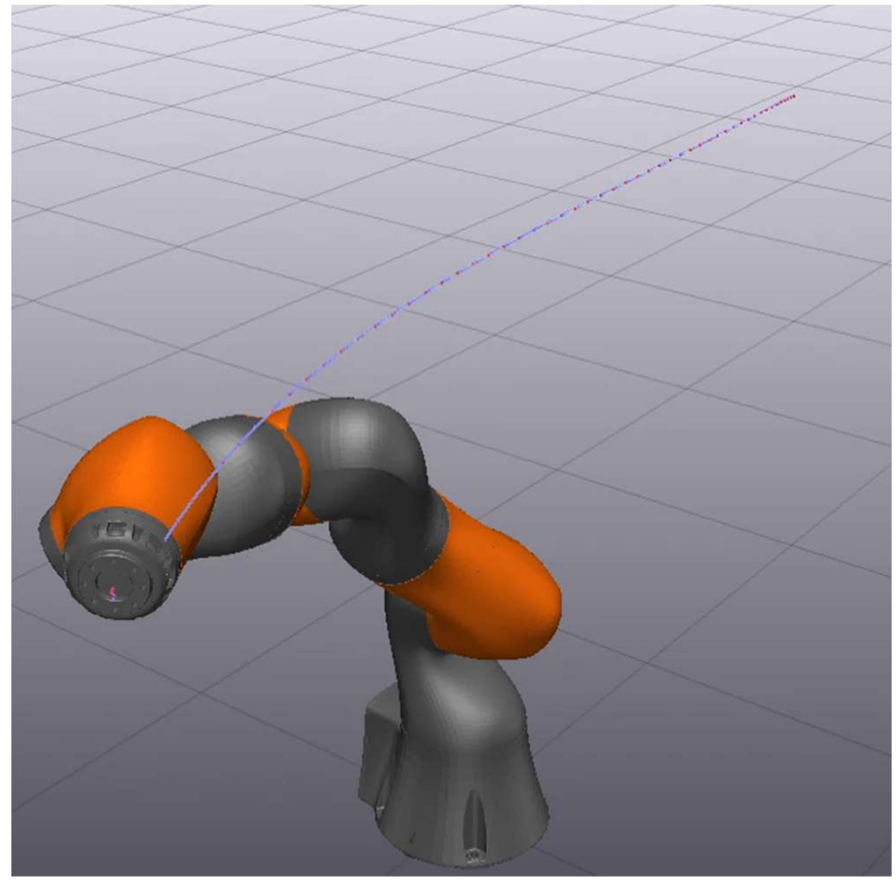


We mapped the same underlying code onto both the GPU and multi-threaded CPU (adding loops in the CPU case)

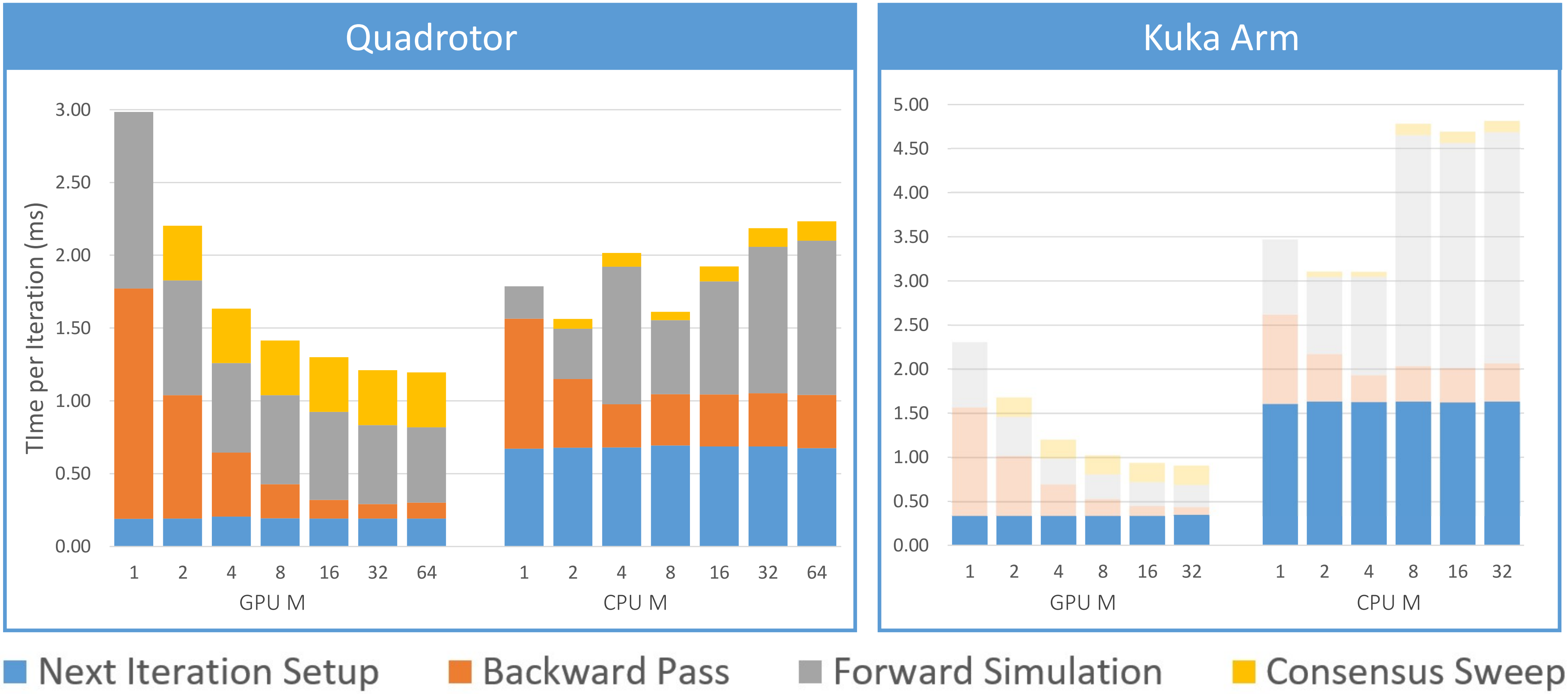
[NVIDIA]

We evaluated Parallel DDP on a GPU and CPU through two experiments in simulation

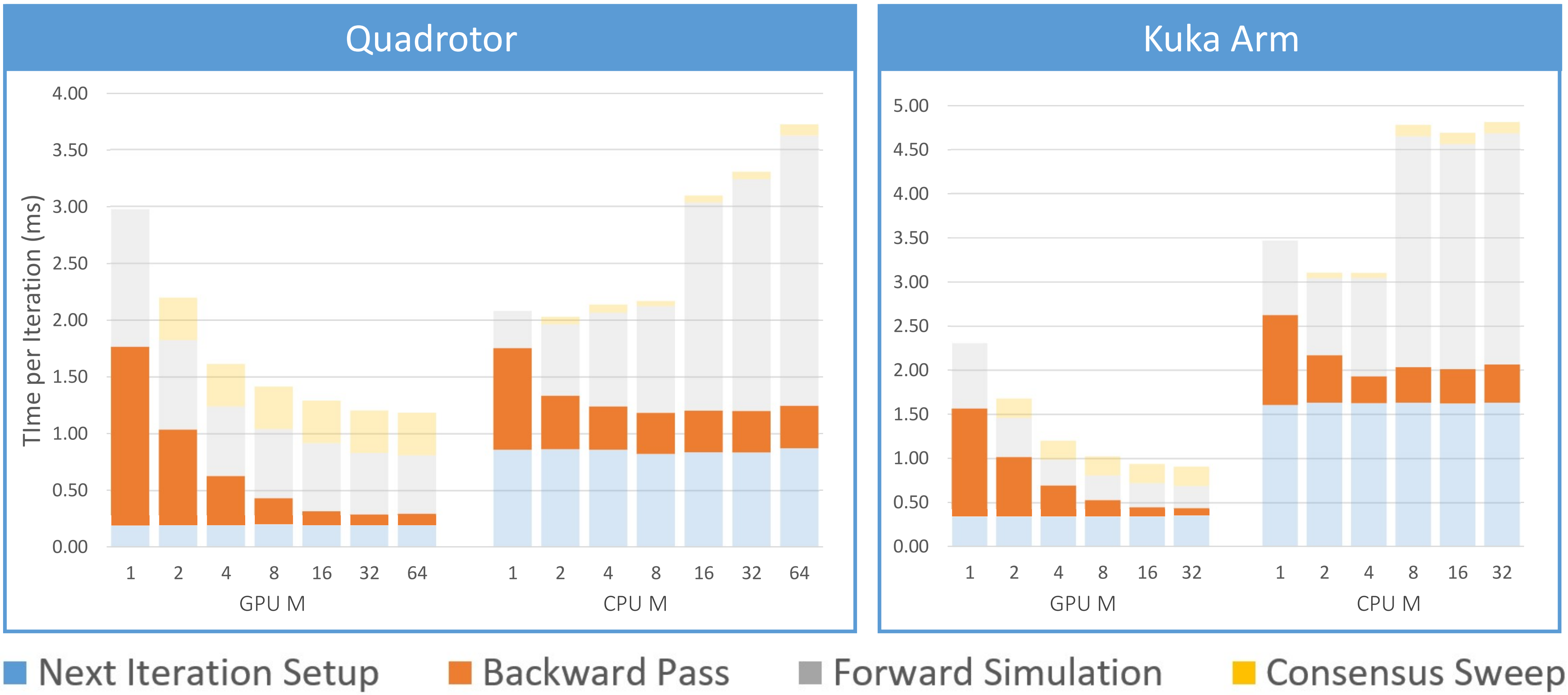




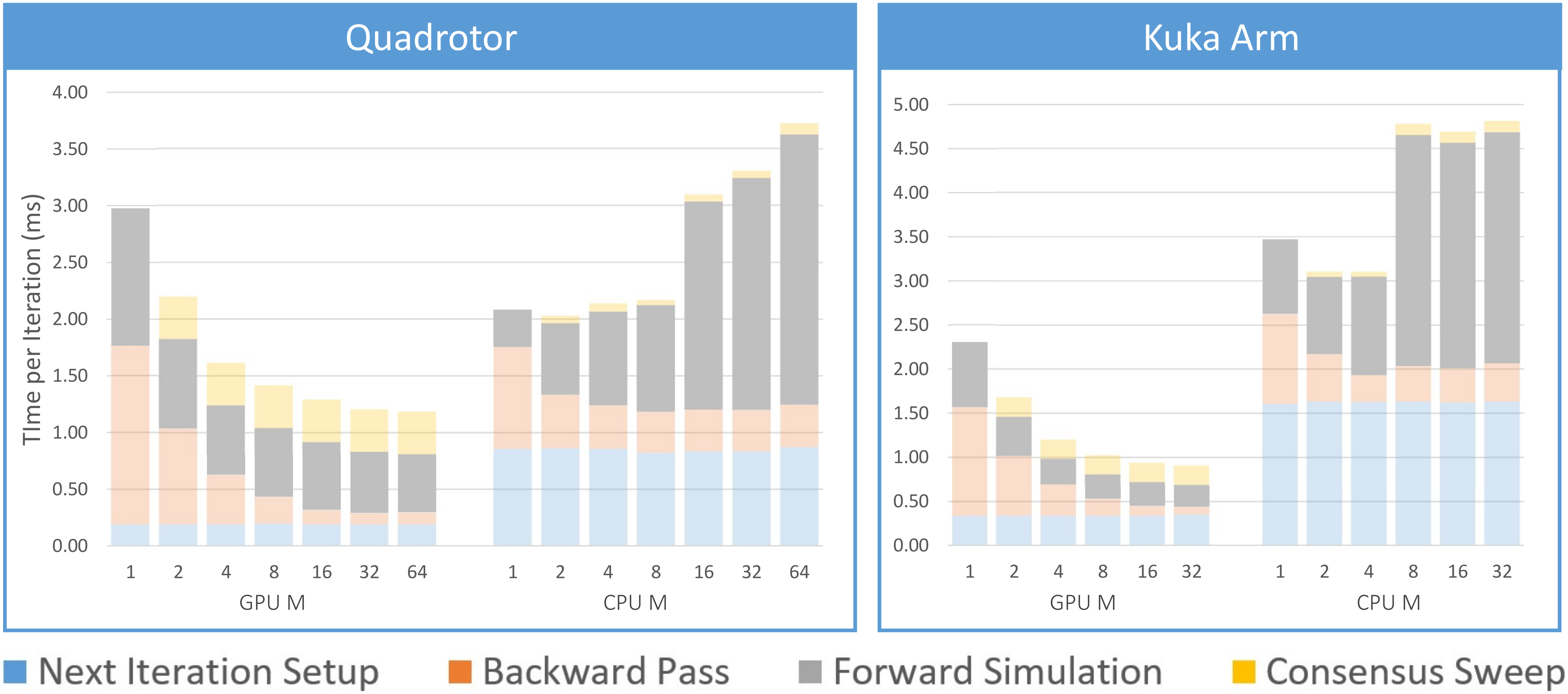
The GPU is able to exploit instruction level parallelism for a faster next iteration setup



Both the CPU and GPU are able to exploit algorithm level parallelism in the backward pass

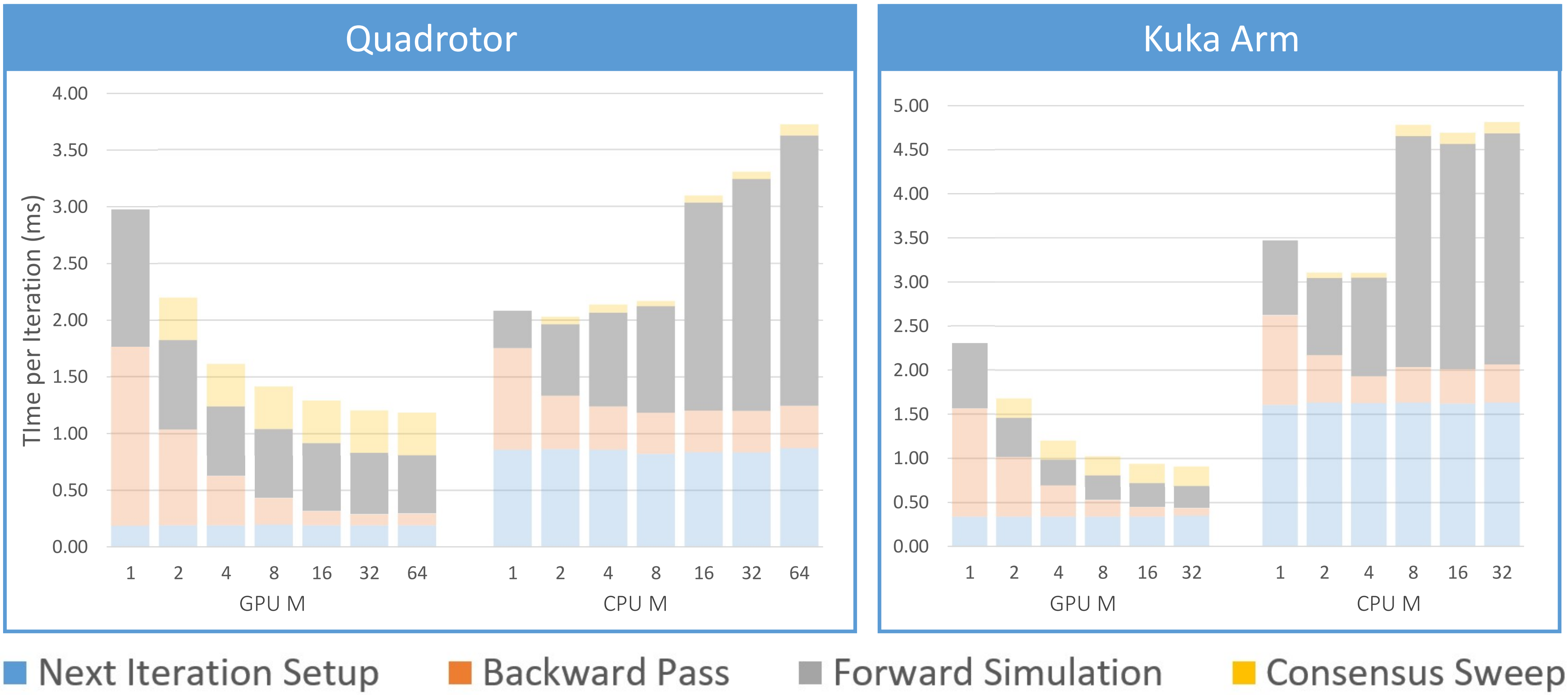


The forward simulation does not parallelize well on the CPU

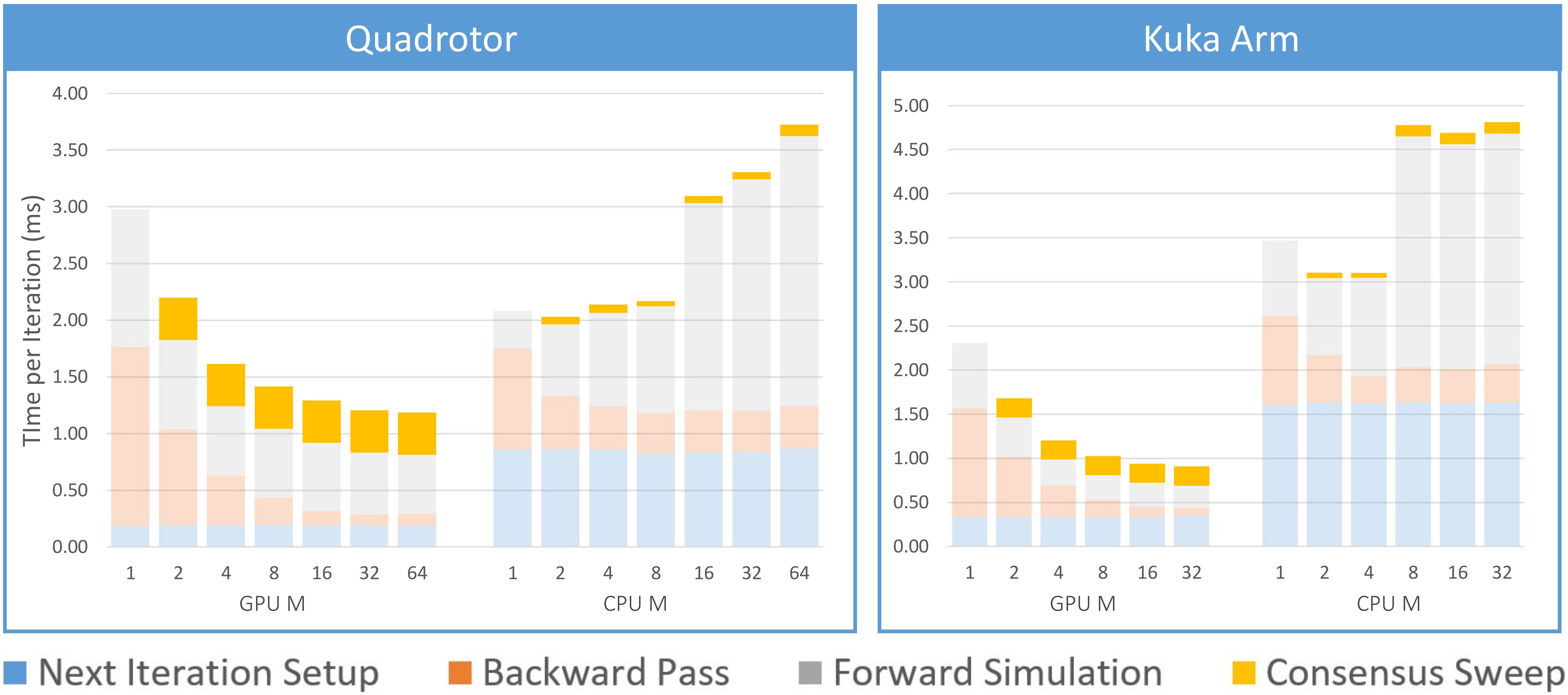


Algorithm level parallelism leads to delayed information and slower convergence

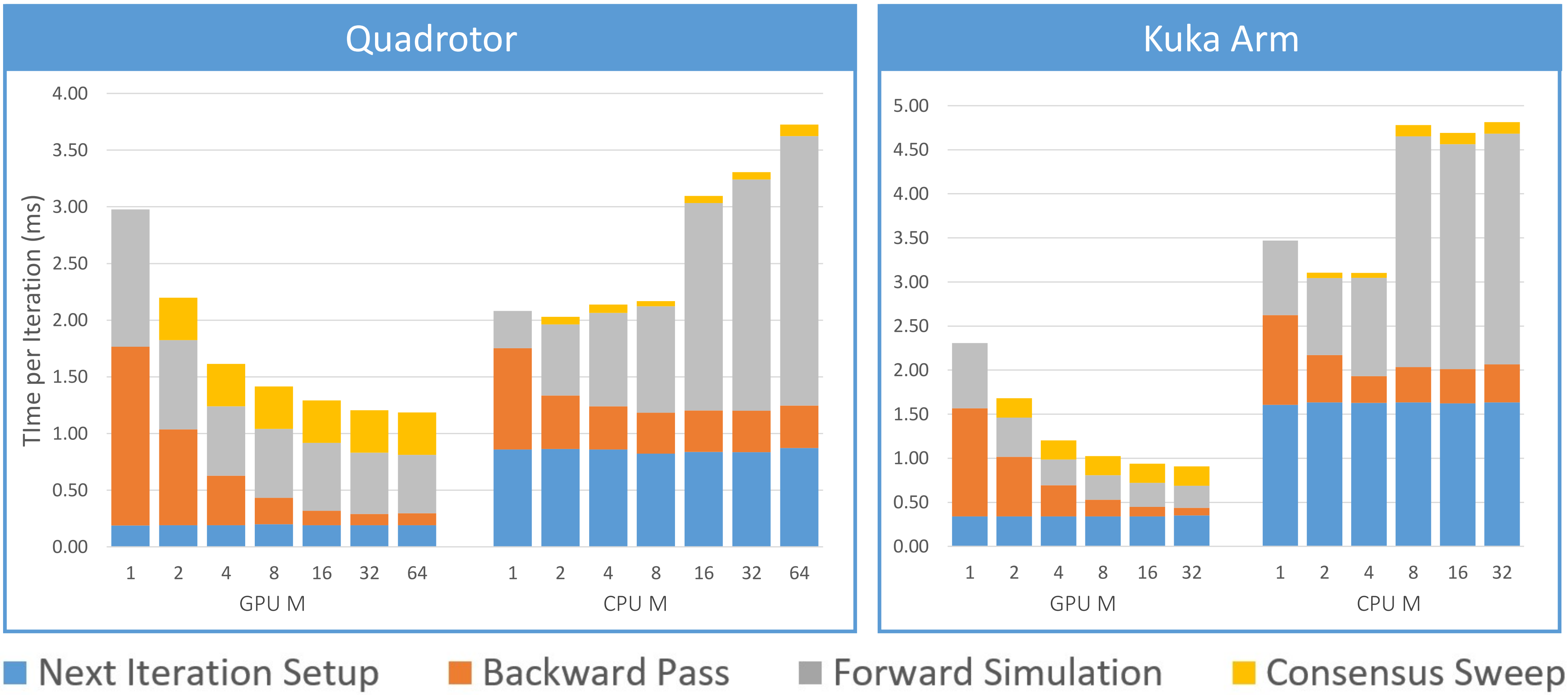
The forward simulation does not parallelize well on the CPU



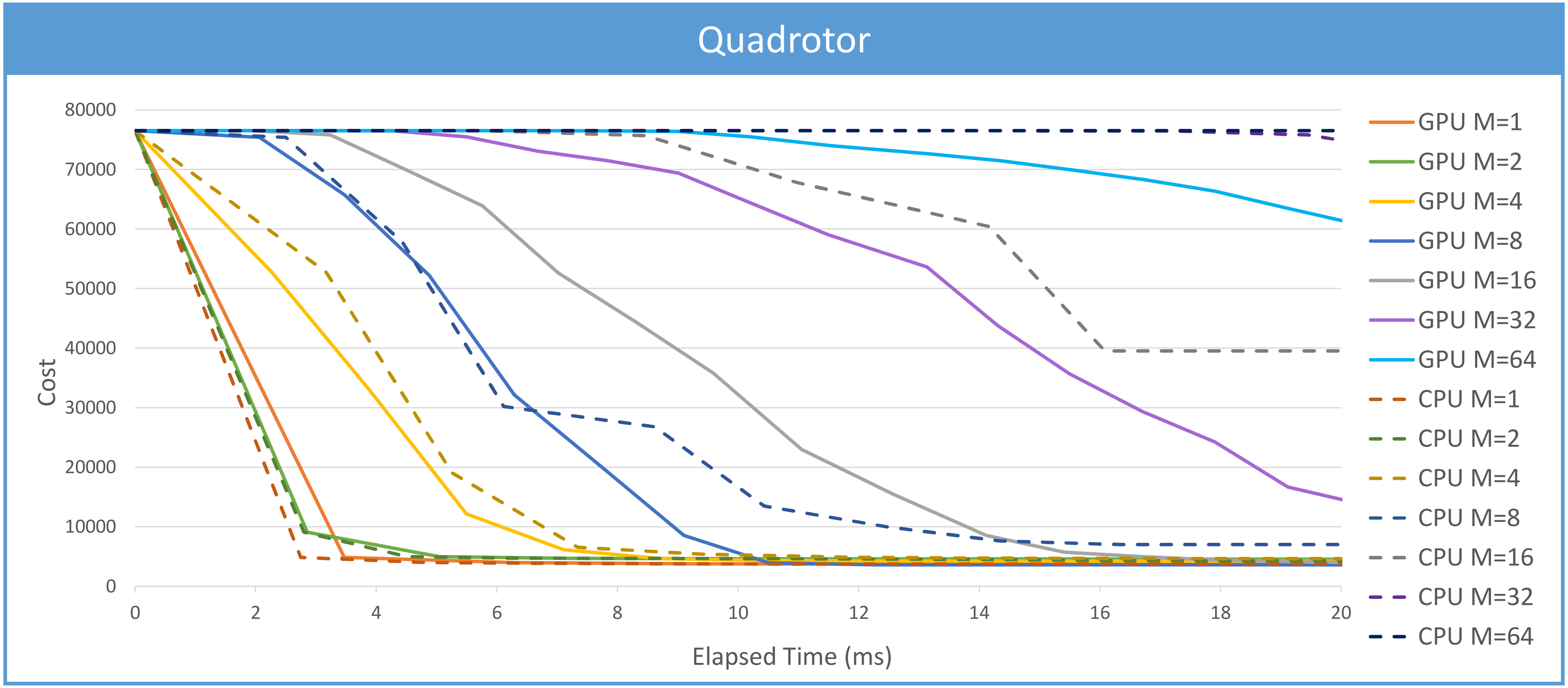
The CPU is able to compute the serial forward consensus sweep faster than the GPU



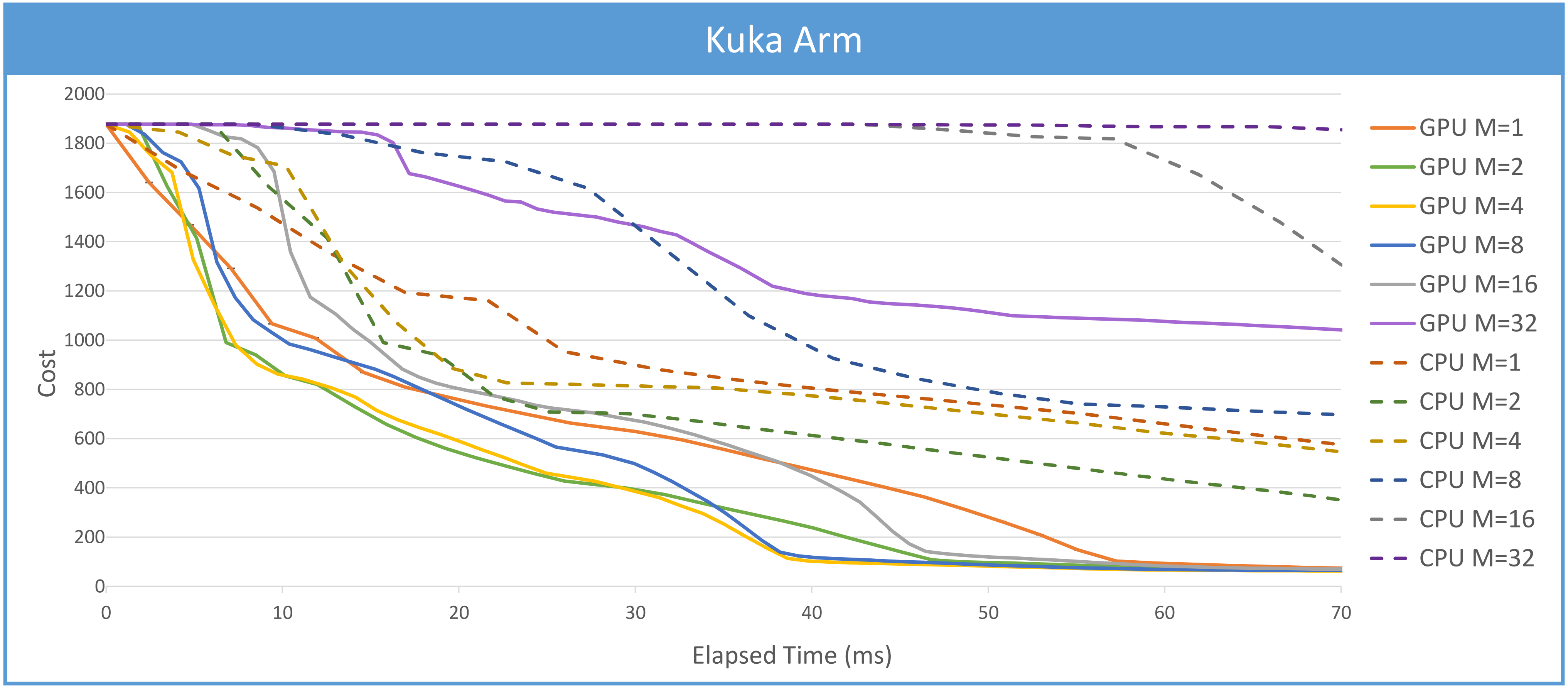
Parallelism has tradeoffs but when exploited carefully leads to improved performance



Parallelism has tradeoffs but when exploited carefully leads to improved performance

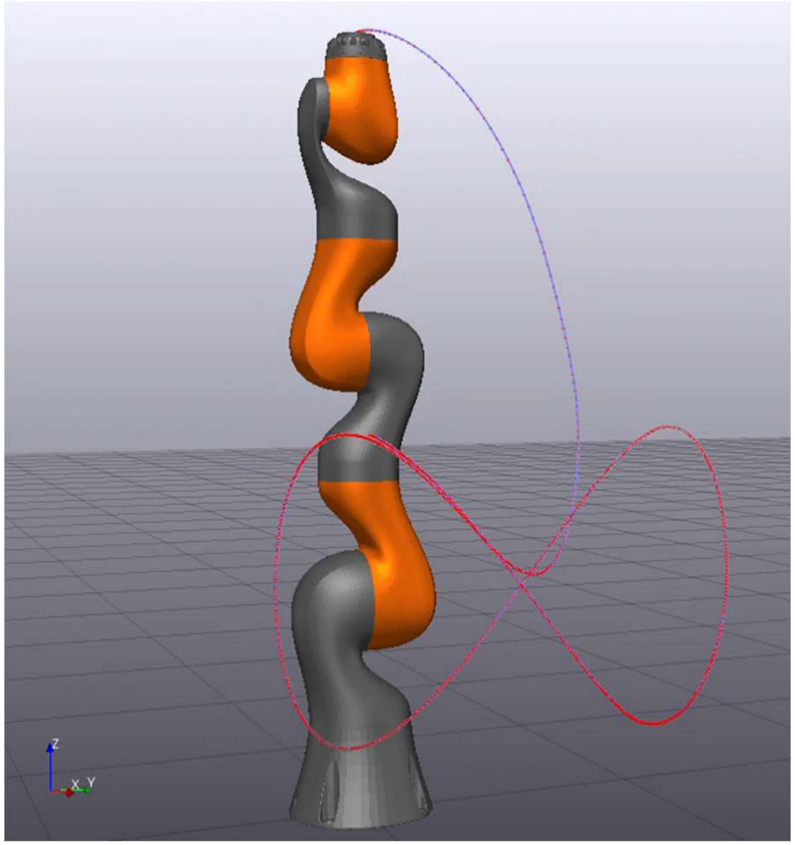


Parallelism has tradeoffs but when exploited carefully leads to improved performance



Parallel DDP on a GPU can be used for real time Model Predictive Control

- GPU M=4
- 10ms control loop
- Only given current goal position at each solve



Let’s air that dirty laundry

agile.seas.harvard.edu brian_plancher@g.harvard.edu

- Usual DDP sticking points still apply as it is sensitive to:
  - Cost function choices
  - Initial state and input trajectories
  - Regularization scheme
- Sensitivities are heightened for higher degrees of parallelization
- MPC results need to be validated on hardware (I am actively working on that)

This work was supported by a Draper Internal Research and Development grant and by the National Science Foundation Graduate Research Fellowship (under grant DGE1745303). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the funding organizations.

A Performance Analysis of Parallel Differential Dynamic Programming on a GPU

agile.seas.harvard.edu brian_plancher@g.harvard.edu

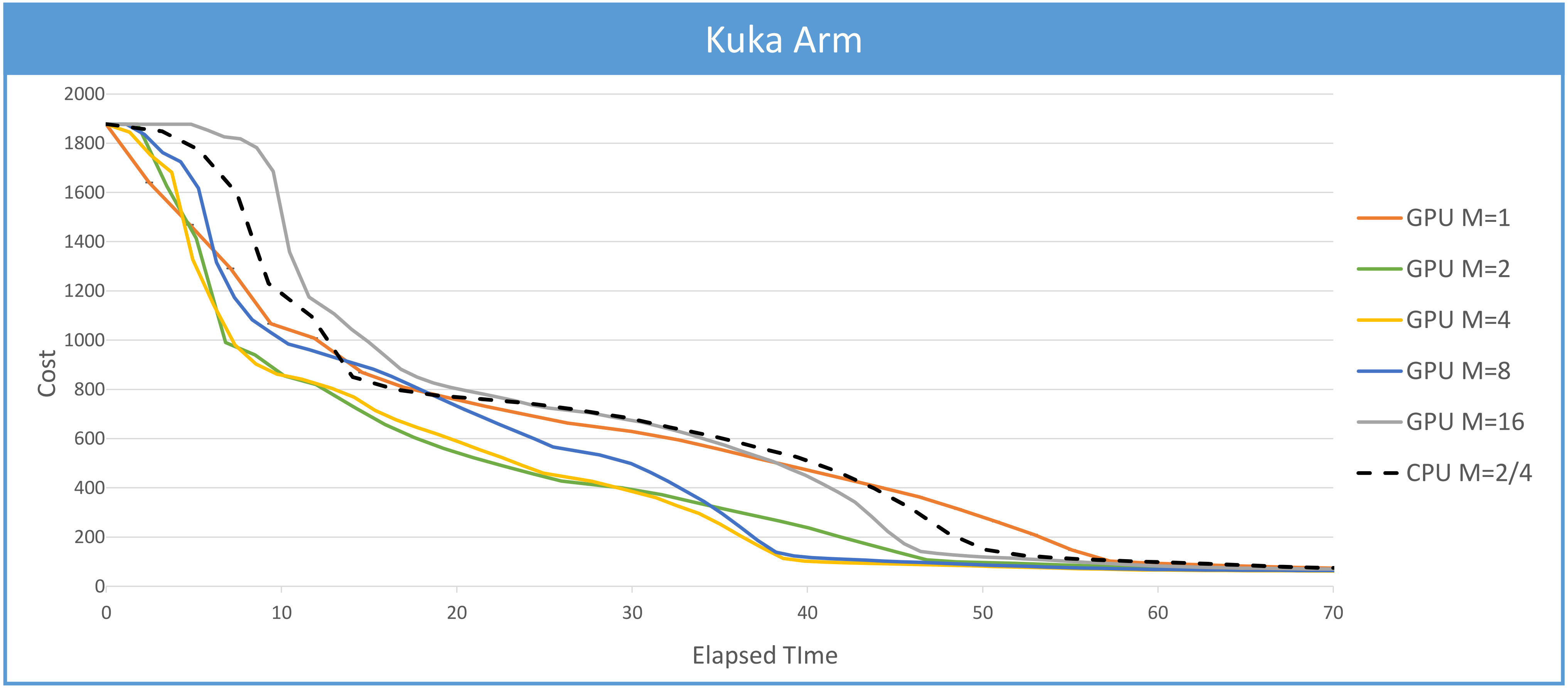
- Exploit **instruction level parallelism** for maximal performance
- Hardware specific tradeoffs exist for **algorithm level parallelism**
- If expensive operations can be parallelized then GPUs can provide higher performance
- DDP is not very parallelizable – potential for direct methods?
- More evidence for **real-time model predictive control**

This work was supported by a Draper Internal Research and Development grant and by the National Science Foundation Graduate Research Fellowship (under grant DGE1745303). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the funding organizations.

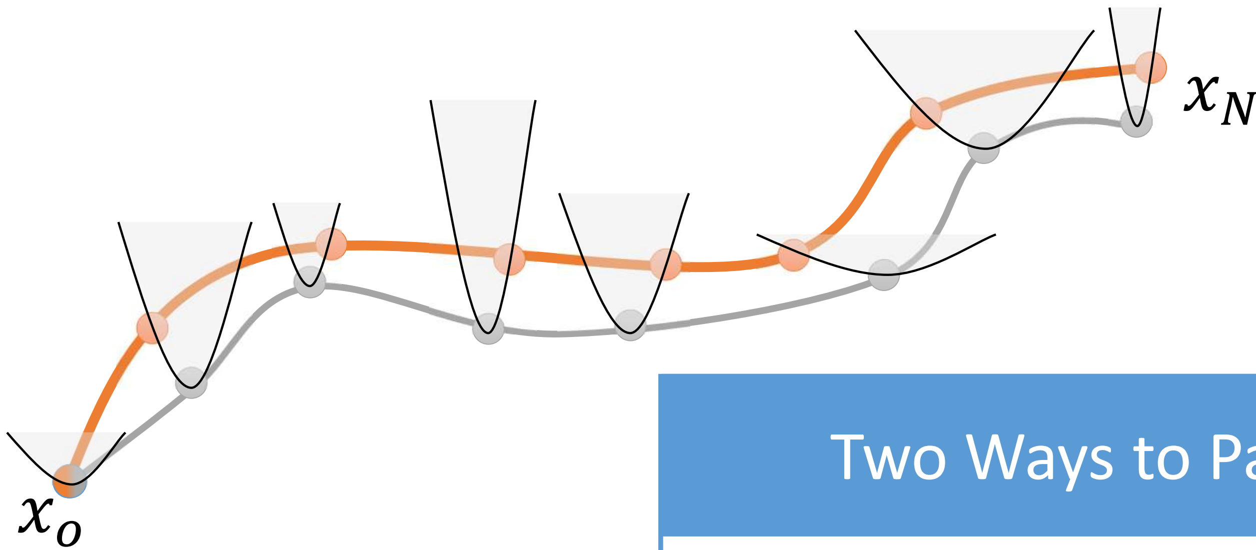
Thank you for listening!

Questions?

Parallelism has tradeoffs but when exploited carefully leads to improved performance



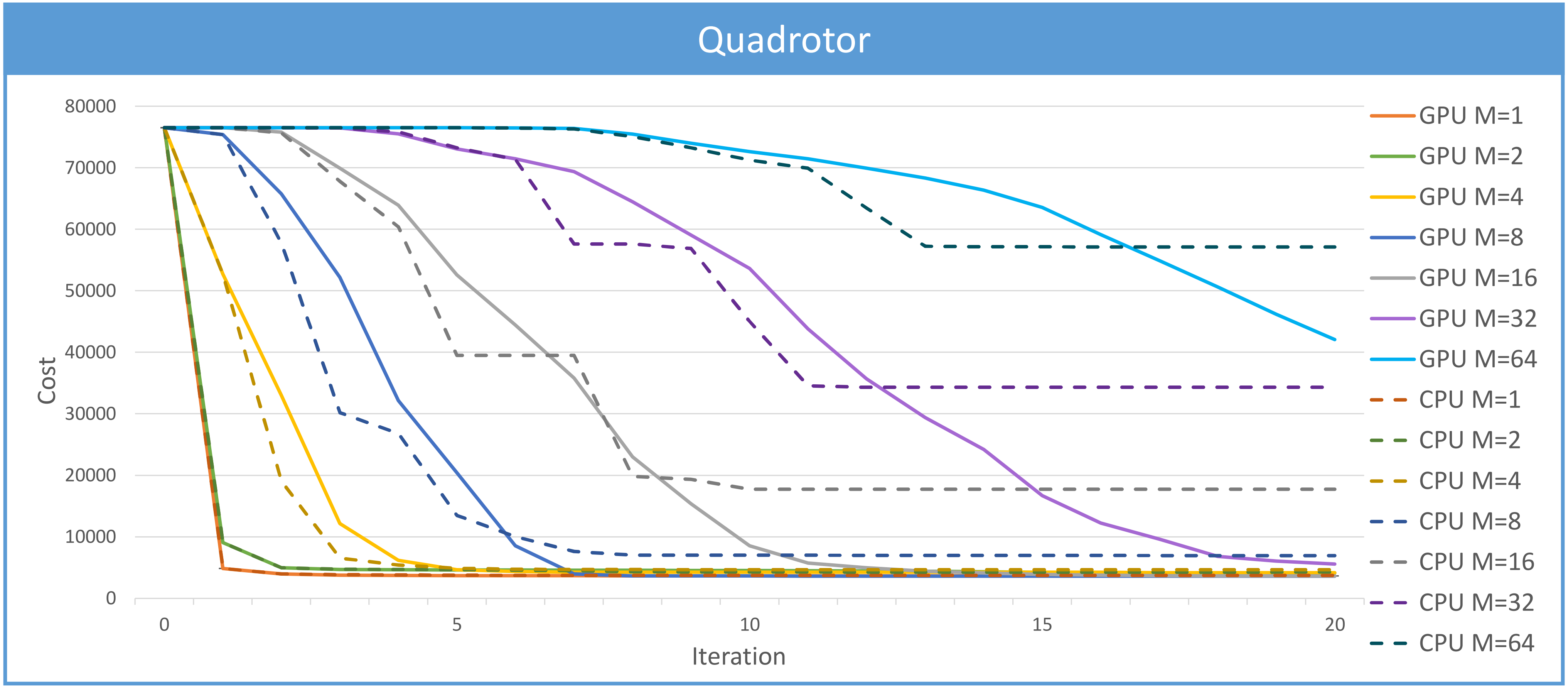
DP methods (DDP/SLQ/iLQR) use quadratic approximations around a nominal trajectory



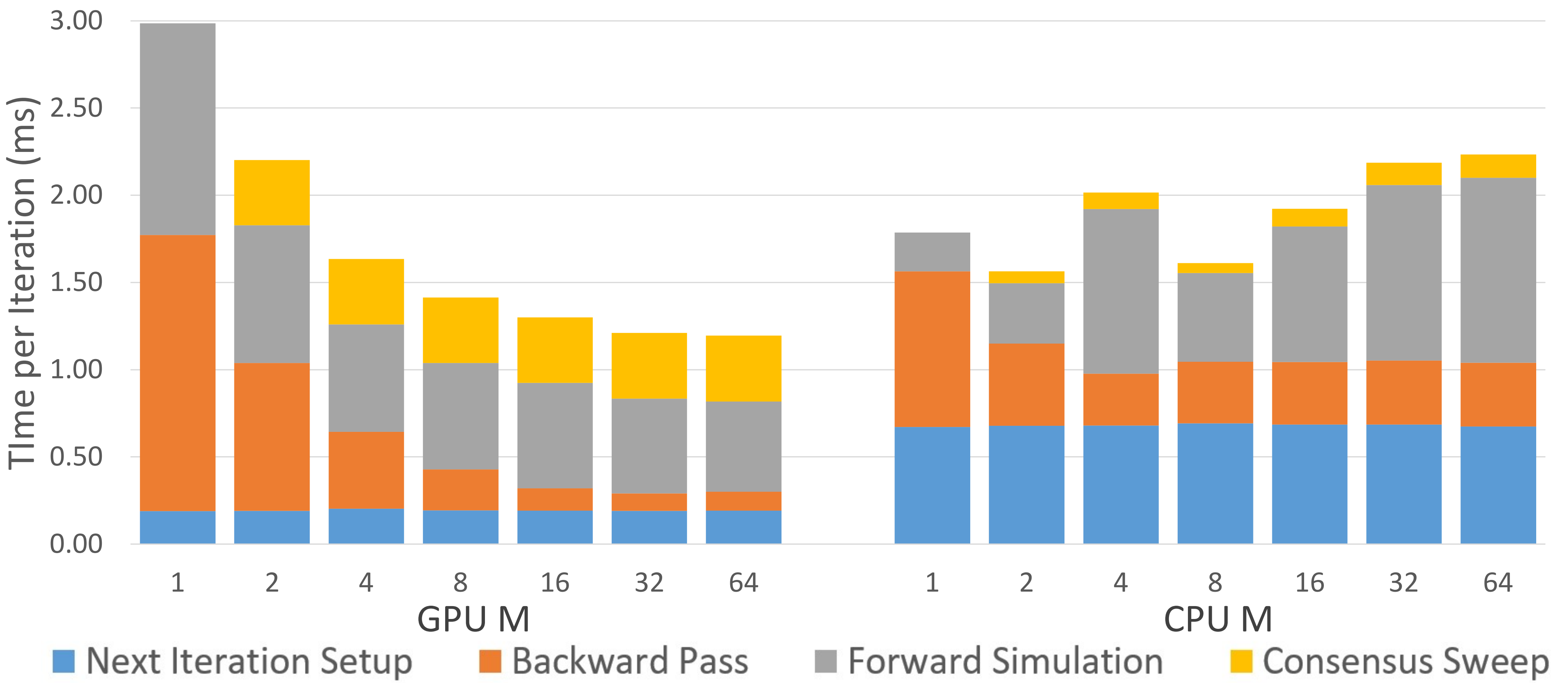
Two Ways to Parallelize Algorithms

1. Instruction Level Parallelism
2. Algorithm Level Parallelism

Algorithm level parallelism leads to delayed information and slower convergence

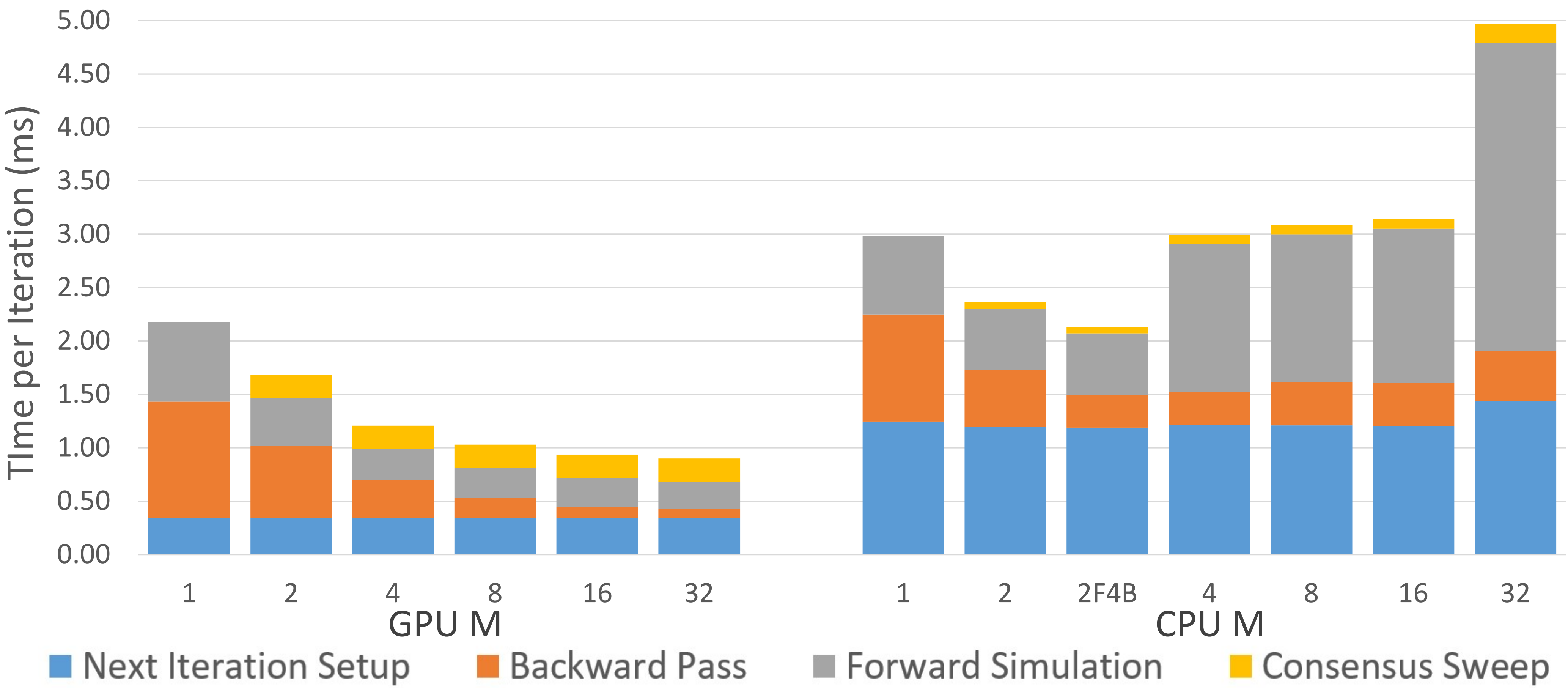


Quad



Time per Iteration (ms)
3.00
2.50
2.00
1.50
1.00
0.50
0.00
1
2
4
8
16
32
64
GPU M
1
2
4
8
16
32
64
CPU M
Next Iteration Setup
Backward Pass
Forward Simulation
Consensus Sweep

Arm



TIme per Iteration (ms)
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
1
2
4
8
16
32
GPU M
1
2
2F4B
4
8
16
32
CPU M
Next Iteration Setup
Backward Pass
Forward Simulation
Consensus Sweep