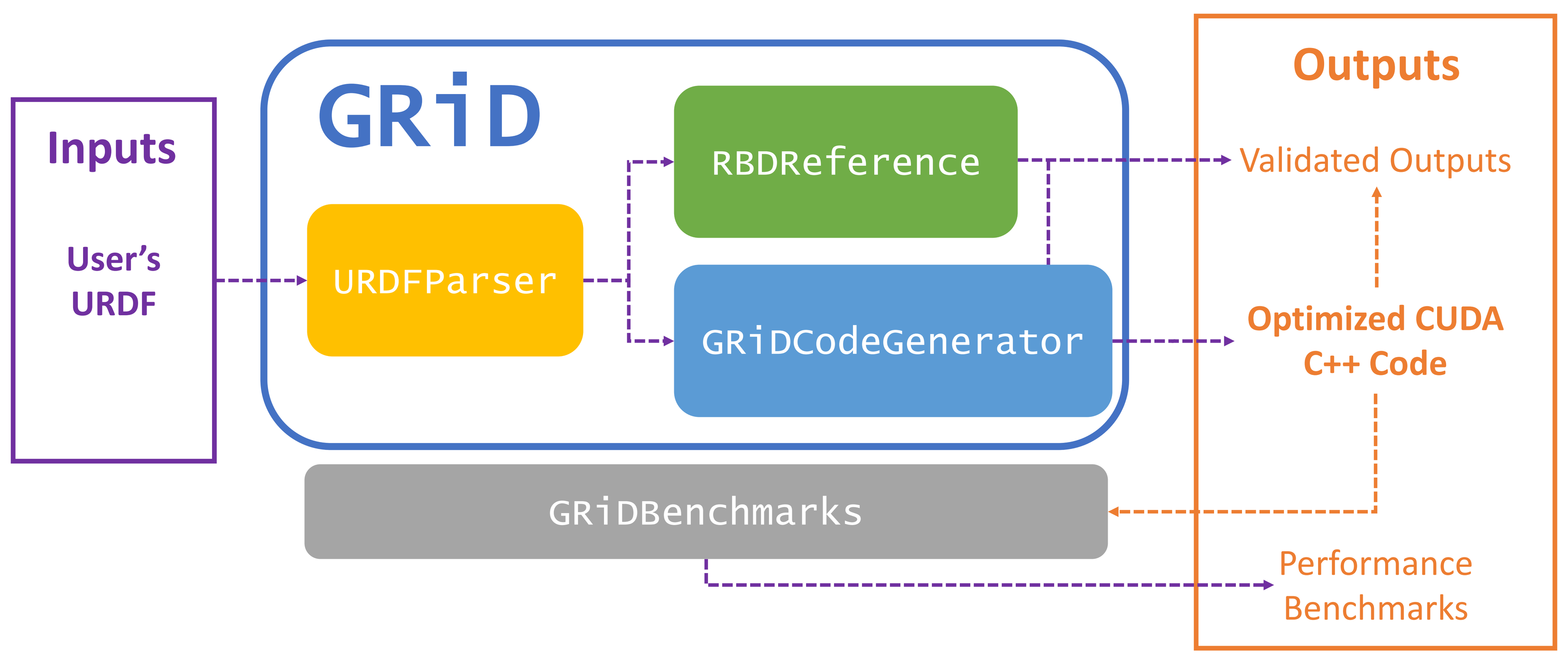
GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

Brian Plancher[1], Sabrina M. Neuman[1], Radhika Ghosal[1], Scott Kuindersma[1,2], Vijay Janapa Reddi[1]

1: Harvard University John A. Paulson School of Engineering and Applied Sciences, 2: Boston Dynamics

The Big Picture:



github.com/robot-acceleration/GRiD

GRiD is a header-only, modular, open-source, GPU-accelerated library for rigid body dynamics with analytical gradients. Key features include:

- **URDF parsing & code generation** to deliver optimized dynamics kernels that expose GPU-friendly computational patterns
  - E.g., Leverages both fine-grained parallelism within each computation & coarse-grained parallelism between computations
- Delivers **end-to-end computational speedups** through algorithmic refactoring
- **Modular, open-source, and header-only**

GRiD currently supports:

- Prismatic, fixed, and revolute joints
- **ID, FD, M^{-1}**
- **∇ID, ∇FD** with respect to $\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}$

This material is based upon work supported by the National Science Foundation (under Grant DGE1745303 and Grant 2030859). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and may not reflect those of the funding organizations.

Design Optimizations:

We **re-factored** algorithms to better leverage the GPU's strengths by:

- Exposing more **natural parallelism** (especially across branches)
- Reducing work done in serial loops
- Leveraging topology driven **sparsity patterns** in matrices
- Unifying computational operations

Algorithm 1 ∇RNEA-F$(\dot{q}, v, a, f, X, S, I) \rightarrow \partial c/\partial u$

for frame $i = 1 : N$ **do**

$$\frac{\partial v_i}{\partial u} = {}^iX_{\lambda_i}\frac{\partial v_{\lambda_i}}{\partial u} + \begin{cases} ({}^iX_{\lambda_i}v_{\lambda_i}) \times S_i & u \equiv q \\ S_i & u \equiv \dot{q} \end{cases}$$

$$\frac{\partial a_i}{\partial u} = {}^iX_{\lambda_i}\frac{\partial a_{\lambda_i}}{\partial u} + \frac{\partial v_{\lambda_i}}{\partial u} \times S_i\dot{q}_i + \begin{cases} ({}^iX_{\lambda_i}a_{\lambda_i}) \times S_i \\ v_i \times S_i \end{cases}$$

$$\frac{\partial f_i}{\partial u} = I_i\frac{\partial a_i}{\partial u} + \frac{\partial v_i}{\partial u} \times^* I_i v_i + v_i \times^* I_i\frac{\partial v_i}{\partial u}$$

Algorithm 2 ∇RNEA-F-GRiD$(\dot{q}, v, a, f, X, S, I) \rightarrow \partial f/\partial u$

for frame $i = 1 : n$ **in parallel do**

$\alpha_i = {}^iX_{\lambda_i}v_{\lambda_i} \quad \beta_i = {}^iX_{\lambda_i}a_{\lambda_i} \quad \gamma_i = I_i v_i$

$\alpha_i = \alpha_i \times S_i \quad \beta_i = \beta_i \times S_i \quad \delta_i = v_i \times S_i$

for level $l = 0 : l_{max}$ **do**

for frame $i \in l$ **in parallel do**

$$\frac{\partial v_i}{\partial u} = {}^iX_{\lambda_i}\frac{\partial v_{\lambda_i}}{\partial u} + \begin{cases} \alpha_i & u \equiv q \\ S_i & u \equiv \dot{q} \end{cases}$$

for frame $i = 1 : n$ **in parallel do**

$$\rho_i = \frac{\partial v_{\lambda_i}}{\partial u} \times S_i\dot{q}_i + \begin{cases} \beta_i \\ \delta_i \end{cases}$$

for level $l = 0 : l_{max}$ **do**

for frame $i \in l$ **in parallel do**

$$\frac{\partial a_i}{\partial u} = {}^iX_{\lambda_i}\frac{\partial a_{\lambda_i}}{\partial u} + \rho_i$$

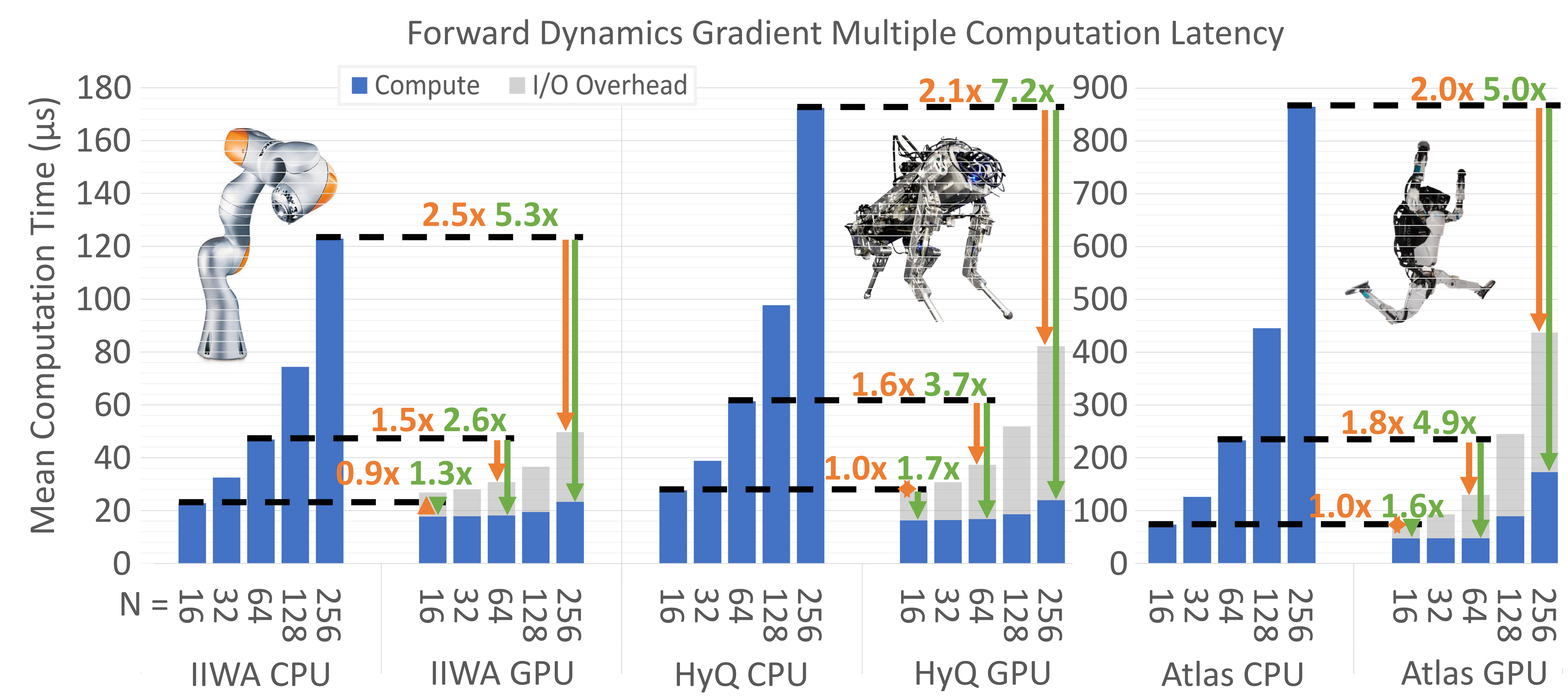
for frame $i = 1 : n$ **in parallel do**

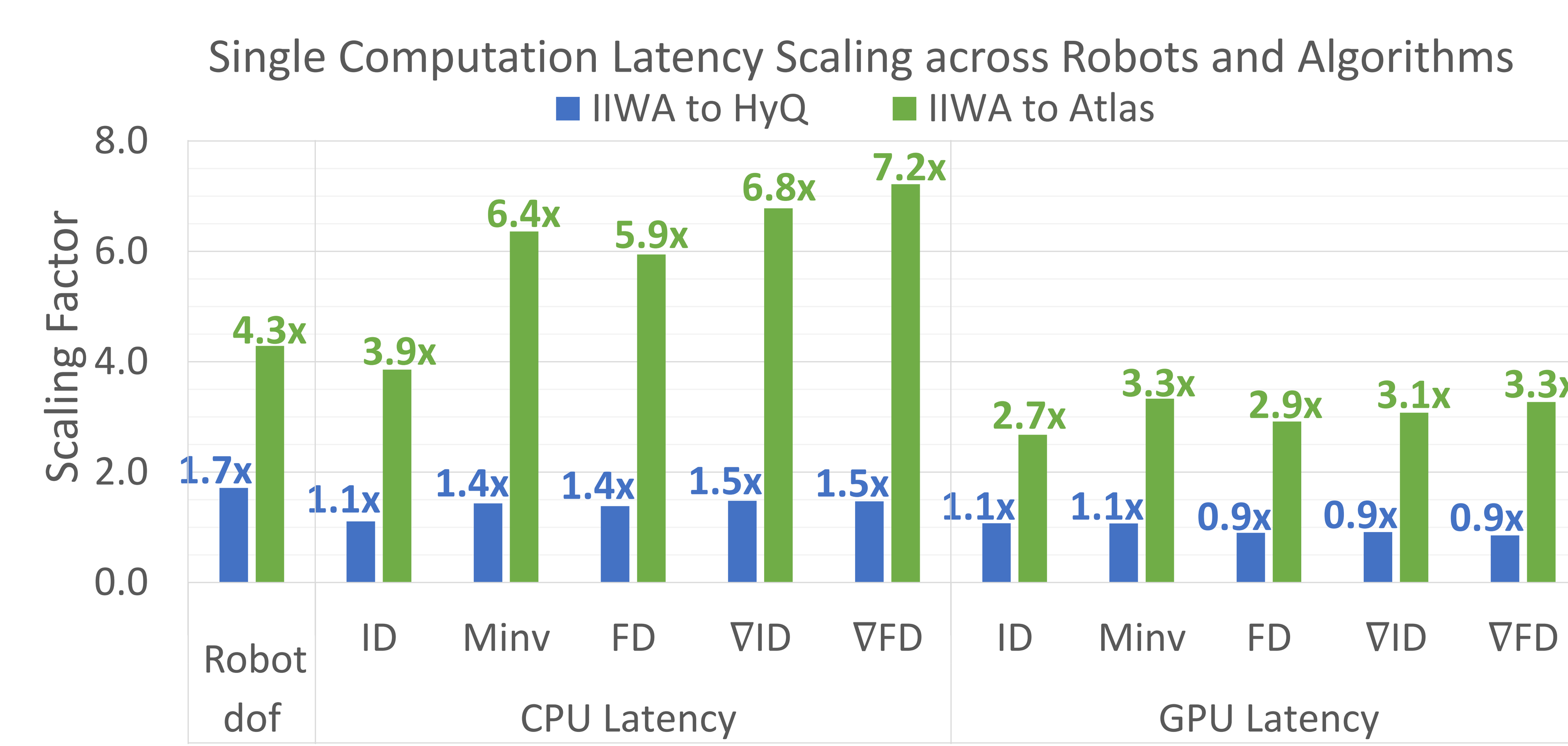
$$\frac{\partial f_i}{\partial u} = \frac{\partial v_i}{\partial u} \times^* \gamma_i \qquad \eta_i = v_i \times^* I_i$$

$$\frac{\partial f_i}{\partial u} = \frac{\partial f_i}{\partial u} + I_i\frac{\partial a_i}{\partial u} + \eta_i\frac{\partial v_i}{\partial u}$$

Performance Results:

- Benchmarked against Pinocchio, a state-of-the-art CPU library
  - Pinocchio supports optimized CPU code generation of rigid body dynamics & analytical gradients
- **GRiD scales well to complex robots and multiple computations**
  - As much as a 7.2x computational speedup over the CPU
  - As much as a 2.5x speedup when accounting for I/O overhead





Single Computation Latency (µs)

(ID = Inverse Dynamics, M^{-1} = Direct Inverse of the Mass Matrix, FD = Forward Dynamics, and ∇ indicates the gradient of that algorithm)

	Algorithm	IIWA	HyQ	Atlas
CPU	ID	0.3	0.3	1.1
	M^{-1}	0.5	0.8	3.4
	FD	0.9	1.2	5.3
	∇ID	1.4	2.1	9.8
	∇FD	2.9	4.3	20.9
GPU	ID	3.0	3.2	8.0
	M^{-1}	5.2	5.6	17.4
	FD	7.7	6.9	22.4
	∇ID	6.3	5.8	19.5
	∇FD	12.9	11.0	42.1

We used a high-performance workstation with a 3.8GHz eight-core Intel Core i7-10700K CPU and a 1.44GHz NVIDIA GeForce RTX 3080 GPU running Ubuntu 20.04 and CUDA 11.4.4 We compare timing results across three robot models: the 7 degrees-of-freedom (dof) Kuka LBR IIWA-14 manipulator, the 12 dof HyQ quadruped, and the 30 dof Atlas humanoid. For single computation and multiple computation latency, we took the average of one million, and one hundred thousand trials, respectively.