GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

Brian Plancher¹, Sabrina M. Neuman¹, Radhika Ghosal¹, Scott Kuindersma¹², Vijay Janapa Reddi¹

1: Harvard University John A. Paulson School of Engineering and Applied Sciences, 2: Boston Dynamics
GRiD makes it easy to use the GPU with robotics algorithms that use rigid body dynamics and provides up to a 7.2x speedup and maintains a 2.5x speedup with I/O.
GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

1. Why GPU Rigid Body Dynamics?

2. GRiD’s Modular Design

3. GRiD’s Optimizations

4. Results
Rigid Body Dynamics Gradients are a bottleneck for planning and control (e.g., nonlinear MPC)

[1] J. Carpentier and N. Mansrud, “Analytical Derivatives of Rigid Body Dynamics Algorithms,” RSS 2018

[2] M. Neunert, et al., “Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking,” ICRA 2016

[3] Best end-to-end [C]PU and [G]PU option from B. Plancher and S. Kuindersma, “A Performance Analysis of Parallel Differential Dynamic Programming,” WAFR 2018

Dynamics Gradient as a Percent of Computation

[1] [2] [3C] [3G]
CPUs aren’t getting faster

- Frequency scaling is ending (CPUs aren’t getting faster)
- Massive parallelism on GPUs may be a solution for hardware acceleration

[Shao and Brooks “Synthesis Lectures on Computer Architecture” 2015]
1. Why GPU Rigid Body Dynamics?

2. GRiD’s Modular Design

3. GRiD’s Optimizations

4. Results
GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

https://github.com/robot-acceleration
GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

https://github.com/robot-acceleration
GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

https://github.com/robot-acceleration

**GRiD** currently supports:

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

We are actively working to expand these features and welcome community support in this effort!
GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

1. Why GPU Rigid Body Dynamics?

2. GRiD’s Modular Design

3. GRiD’s Optimizations

4. Results
Algorithm 1 \( \nabla \text{RNEA-F}(\dot{q}, v, a, f, X, S, I) \rightarrow \partial c / \partial u \)

for frame \( i = 1 : N \) do

\[
\frac{\partial v_i}{\partial u} = i X_{\lambda_i} \frac{\partial v_{\lambda_i}}{\partial u} + \left\{ \begin{array}{l}
(i X_{\lambda_i} v_{\lambda_i}) \times S_i \\
S_i
\end{array} \right\} \quad u \equiv \dot{q}
\]

3: \[
\frac{\partial a_i}{\partial u} = i X_{\lambda_i} \frac{\partial a_{\lambda_i}}{\partial u} + \frac{\partial v_{\lambda_i}}{\partial u} \times S_i \dot{q} + \left\{ \begin{array}{l}
(i X_{\lambda_i} a_{\lambda_i}) \times S_i \\
v_i \times S_i
\end{array} \right\}
\]

4: \[
\frac{\partial f_i}{\partial u} = I_i \frac{\partial a_i}{\partial u} + \frac{\partial v_i}{\partial u} \times* I_i v_i + v_i \times* I_i \frac{\partial v_i}{\partial u}
\]
GRiD exploits the structure of each robot to minimize memory and optimize latency.

Refactor algorithms to expose parallel loops of unified operations.

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

1: for frame \( i = 1 : n \) in parallel do
2: \( \alpha_i = i X_{\lambda_i} v_{\lambda_i} \quad \beta_i = i X_{\lambda_i} a_{\lambda_i} \quad \gamma_i = I_i v_i \)
3: \( \alpha_i = \alpha_i \times S_i \quad \beta_i = \beta_i \times S_i \quad \delta_i = v_i \times S_i \)
4: for level \( l = 0 : l_{\text{max}} \) do
5: for frame \( i \in l \) in parallel do
6: \( \frac{\partial u_i}{\partial u} = i X_{\lambda_i} \frac{\partial v_{\lambda_i}}{\partial u} + \begin{cases} \alpha_i & u \equiv q \\ S_i & u \equiv \dot{q} \end{cases} \)
7: for frame \( i = 1 : n \) in parallel do
8: \( \rho_i = \frac{\partial v_{\lambda_i}}{\partial u} \times S_i \dot{q}_i + \begin{cases} \beta_i \\ \delta_i \end{cases} \)
9: for level \( l = 0 : l_{\text{max}} \) do
10: for frame \( i \in l \) in parallel do
11: \( \frac{\partial a_{\lambda_i}}{\partial u} = i X_{\lambda_i} \frac{\partial a_{\lambda_i}}{\partial u} + \rho_i \)
12: for frame \( i = 1 : n \) in parallel do
13: \( \frac{\partial f_i}{\partial u} = \frac{\partial u_i}{\partial u} \times \gamma_i \quad \eta_i = v_i \times \dot{I}_i \)
14: \( \frac{\partial f_i}{\partial u} = \frac{\partial f_i}{\partial u} + I_i \frac{\partial a_{\lambda_i}}{\partial u} + \eta_i \frac{\partial v_i}{\partial u} \)
GRiD exploits the structure of each robot to minimize memory and optimize latency

Refactor algorithms to expose parallel loops of unified operations

Compute remaining serial operations in parallel across levels of the rigid body tree
GRiD exploits the structure of each robot to minimize memory and optimize latency.

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

1: for frame \( i = 1 : n \) in parallel do
2: \[ \alpha_i = iX_{\lambda_i}v_{\lambda_i}, \quad \beta_i = iX_{\lambda_i}a_{\lambda_i}, \quad \gamma_i = I_i v_i \]
3: \[ \alpha_i = \alpha_i \times S_i, \quad \beta_i = \beta_i \times S_i, \quad \delta_i = v_i \times S_i \]

4: for level \( l = 0 : l_{\text{max}} \) do
5: for frame \( i \in l \) in parallel do
6: \[ \frac{\partial u_i}{\partial u} = iX_{\lambda_i} \frac{\partial v_{\lambda_i}}{\partial u} + \begin{cases} \alpha_i & u \equiv q \\ S_i & u \equiv \dot{q} \end{cases} \]

7: for frame \( i = 1 : n \) in parallel do
8: \[ \rho_i = \frac{\partial v_{\lambda_i}}{\partial u} \times S_i \dot{q}_i + \begin{cases} \beta_i \\ \delta_i \end{cases} \]

9: for level \( l = 0 : l_{\text{max}} \) do
10: for frame \( i \in l \) in parallel do
11: \[ \frac{\partial a_i}{\partial u} = iX_{\lambda_i} \frac{\partial a_{\lambda_i}}{\partial u} + \rho_i \]
12: for frame \( i = 1 : n \) in parallel do
13: \[ \frac{\partial f_i}{\partial u} = \frac{\partial u_i}{\partial u} \times* \gamma_i, \quad \eta_i = v_i \times* I_i \]
14: \[ \frac{\partial f_i}{\partial u} = \frac{\partial f_i}{\partial u} + I_i \frac{\partial a_i}{\partial u} + \eta_i \frac{\partial v_i}{\partial u} \]

The branch structure also determines sparsity in the columns of \( \partial v, \partial a, \) and \( \partial f \).
1. Why GPU Rigid Body Dynamics?

2. GRiD’s Modular Design

3. GRiD’s Optimizations

4. Results
GRiD improves both computational latency and scalability

As found in previous work, the GPU is faster and performs better as natural parallelism grows.
GRiD improves both computational latency and scalability

| N  | IIWA CPU | IIWA GPU | HyQ CPU | HyQ GPU |
|----|----------|----------|---------|---------|
| 16 | 20       | 20       | 20      | 20      |
| 32 | 32       | 32       | 32      | 32      |
| 64 | 64       | 64       | 64      | 64      |
| 128| 128      | 128      | 128     | 128     |
| 256| 256      | 256      | 256     | 256     |

We show that the GPU does even better as robot complexity grows as well!
GRiD improves both computational latency and scalability.

Although there are limits!
GRiD improves both computational latency and scalability.

And I/O is Problematic!
GRiD is a **URDF to optimized CUDA C++ library** designed to provide **GPU acceleration** for rigid body dynamics algorithms and their analytical gradients. GRiD provides up to a **7.2x speedup** and maintains a **2.5x speedup with I/O**.

[https://github.com/robot-acceleration](https://github.com/robot-acceleration)

**GRiD makes it easy to use the GPU** with robotics algorithms that use rigid body dynamics!

brian_plancher@g.harvard.edu