GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

Brian Plancher\textsuperscript{1}, Sabrina M. Neuman\textsuperscript{1}, Radhika Ghosal\textsuperscript{1}, Scott Kuindersma\textsuperscript{1,2}, Vijay Janapa Reddi\textsuperscript{1}

\textsuperscript{1}: Harvard University John A. Paulson School of Engineering and Applied Sciences, \textsuperscript{2}: Boston Dynamics
GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

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.
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)


Dynamics Gradient as a Percent of Computation

[1] [2] [3C] [3G]

30-90%
CPU performance has plateaued. One solution is massive parallelism on GPUs.

- Frequency scaling is ending (CPUs aren't getting faster)
- Massive parallelism on GPUs may be a solution for hardware acceleration
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
GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

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

[Diagram of GRiD system]

- **Inputs**
  - User's URDF

- **GRiD**
  - URDFParser
  - GRiDCodeGenerator

- **Outputs**
  - Validated Outputs
  - Optimized CUDA C++ Code
  - Performance Benchmarks

- **GRiDBenchmarks**

[Website Link]
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
• ID, FD, M⁻¹
• ∇ID, ∇FD with respect to q, ˙q, u

We are actively working to expand these features and welcome community support in this effort!
1. Why GPU Rigid Body Dynamics?

2. GRiD’s Modular Design

3. GRiD’s Optimizations

4. Results
GRiD exploits the structure of each robot to minimize memory and optimize latency.

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

**Very serial algorithm**

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

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

2. \( \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\{ \left( i X_{\lambda_i} a_{\lambda_i} \right) \times S_i \right. \\
\left. \left( i X_{\lambda_i} a_{\lambda_i} \right) \times S_i \right. \\
\left. v_i \times S_i \right\}
\]

3. \( \frac{\partial f_i}{\partial u} = I_i \frac{\partial a_i}{\partial u} + \frac{\partial v_i}{\partial u} \times^* I_i v_i + v_i \times^* I_i \frac{\partial v_i}{\partial u} \)
GRiD exploits the structure of each robot to minimize memory and optimize latency.

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: \[ \alpha_i = \dot{X}_\lambda v_{\lambda i} \quad \beta_i = \dot{X}_\lambda a_{\lambda i} \quad \gamma_i = I_i v_i \]
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: \hspace{1em} for frame \( i \in l \) in parallel do
6: \[ \frac{\partial v_i}{\partial u} = \dot{X}_\lambda \frac{\partial v_{\lambda i}}{\partial u} + \begin{cases} \alpha_i & u \equiv q \\ S_i & u \equiv \dot{q} \end{cases} \]
7: \hspace{1em} 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: \hspace{1em} for frame \( i \in l \) in parallel do
11: \[ \frac{\partial a_i}{\partial u} = \dot{X}_\lambda \frac{\partial a_{\lambda i}}{\partial u} + \rho_i \]
12: \hspace{1em} for frame \( i = 1 : n \) in parallel do
13: \[ \frac{\partial f_i}{\partial u} = \frac{\partial v_i}{\partial u} \times \gamma_i \quad \eta_i = v_i \times I_i \]
14: \[ \frac{\partial f_i}{\partial u} = \frac{\partial f_i}{\partial u} + I_i \frac{\partial a_i}{\partial u} + \eta_i \frac{\partial \gamma_i}{\partial u} \]

Refactor algorithms to expose parallel loops of unified operations.
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: \( \alpha_i = iX_{\lambda_i}v_{\lambda_i}, \quad \beta_i = iX_{\lambda_i}a, \quad \gamma_i = I_iv_i \)
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 \( l = 0 : l_{\max} \) do
5: for frame \( i \in l \) in parallel do
6: \( \partial v_i/\partial u = iX_{\lambda_i} \partial v_{\lambda_i}/\partial u + \begin{cases} \alpha_i & u = q \\ S_i & u = \dot{q} \end{cases} \)
7: for frame \( i = 1 : n \) in parallel do
8: \( \rho_i = \partial v_{\lambda_i}/\partial u \times S_i \dot{q_i} + \beta_i \delta_i \)
9: for level \( l = 0 : l_{\max} \) do
10: for frame \( i \in l \) in parallel do
11: \( \partial a_i/\partial u = iX_{\lambda_i} \partial a_{\lambda_i}/\partial u + \rho_i \)
12: for frame \( i = 1 : n \) in parallel do
13: \( \partial f_i/\partial u = \partial f_i/\partial u \times \gamma_i, \quad \eta_i = v_i \times \dot{I}_i \)
14: \( \partial f_i/\partial u + I_i \partial a_i/\partial u + \eta_i \partial v_i/\partial u \)

Refactor algorithms to expose parallel loops of unified operations.

Compute remaining serial operations in parallel across levels of the rigid body tree.

Level 0
Level 1
Level 2
Level 3
GRiD exploits the structure of each robot to minimize memory and optimize latency.

The branch structure also determines sparsity in the columns of $\partial v$, $\partial a$, and $\partial f$.

---

**Algorithm 2** $\nabla$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: \[ \alpha_i = iX_{\lambda_i}v_{\lambda_i}, \quad \beta_i = iX_{\lambda_i}a_{\lambda_i}, \quad \gamma_i = I_iv_i \]
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 $l = 0 : l_{max}$ do
5: for frame $i \in l$ in parallel do
6: \[ \frac{\partial v_i}{\partial u} = iX_{\lambda_i} \frac{\partial v_{\lambda_i}}{\partial u} + \begin{cases} \alpha_i & u \equiv q \\ S_i & u \equiv \dot{q} \end{cases} \]
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 $l = 0 : l_{max}$ 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: \[ \frac{\partial f_i}{\partial u} = \frac{\partial v_i}{\partial u} \times \gamma_i, \quad \eta_i = v_i \times I_i \]
14: \[ \frac{\partial f_i}{\partial u} = \frac{\partial f_i}{\partial u} + I_i \frac{\partial a_i}{\partial u} + \eta_i \frac{\partial v_i}{\partial u} \]
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
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.

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!

Forward Dynamics Gradient Multiple Computation Latency

- IIWA CPU
- IIWA GPU
- HyQ CPU
- HyQ GPU
- Atlas CPU
- Atlas GPU

Mean Computation Time ($\mu$s)

- $N = 16, 32, 64, 128, 256$

- $2.5x 5.3x$
- $1.5x 2.6x$
- $0.9x 1.3x$
- $1.6x 3.7x$
- $1.0x 1.7x$
- $1.8x 4.9x$
- $1.0x 1.6x$

Place Zoom Headshot Here
GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients

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

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

brian_plancher@g.harvard.edu

---

This material is based upon work supported by the National Science Foundation (under Grant DGE1745303 and Grant 2030859 to the Computing Research Association for the CIFellows Project). 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.