COLORADO SCHOOL OF MINES

Paper Presentation

CSCI-582 —— Dr. Mehmet Belviranli

Group 1 —— Shaun Kannady
TQ Bill Huynh

Project Overview

Project Summary

- Stage 1
  - Project + paper selection and presentation
  - Architecture - Istanbul
    - 2x AMD EPYC 7402, 4x NVIDIA RTX 3090
  - Application - CUDA programming in Robotics simulation
- Stage 2
  - Project implementation, presentation, and report

Domain

1. Soft Body Robotics
2. Multi Agent Robotics
3. Simulation
4. Reinforcement Learning

Key Point

- Apply GPU acceleration to robotics simulation, where parallelization brings benefits.

Paper Overview

Paper Title

Titan: A Parallel Asynchronous Library for Multi-Agent and Soft-Body Robotics using NVIDIA CUDA

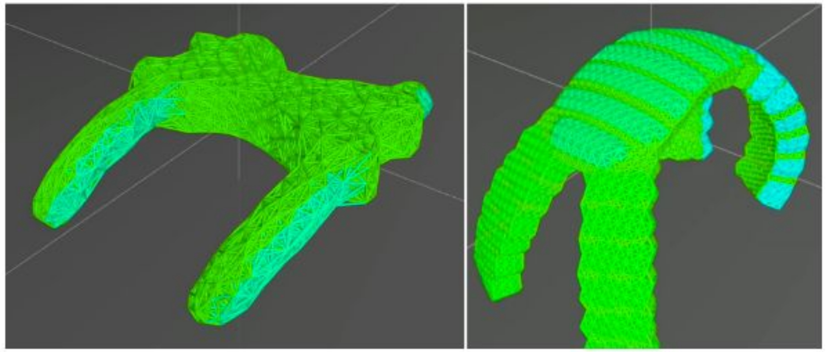


Fig. 1. Two four-legged soft robot simulated in Titan with hundreds of thousands of independent internal joints.

Venue

2020 ICRA (IEEE International Conference on Robotics and Automation)

One-Sentence Summary

A GPU-accelerated simulation library for multiple interacting robot bodies (e.g. multi-agent robotics)

Motivation

Why do we care?

Robotics

Soft robotics - complex structure, 1000s of components

Multi-agent robots – complex interactions as a team

Simulation

Useful for design, planning

Reinforcement Learning - parallel simulations can improve learning rates

Performance

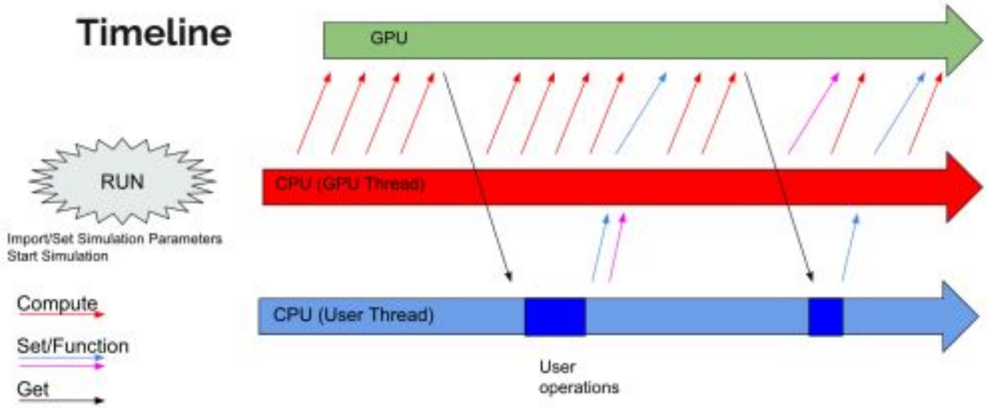
Many calculations can be done asynchronously, in parallel

Less reliance on serial computations

Problem Statement: The paper presents a CUDA-based library for high-performance asynchronous simulation that is capable of running multiple parallel simulations on GPU while performing optimization on CPU simultaneously.

Technical Details

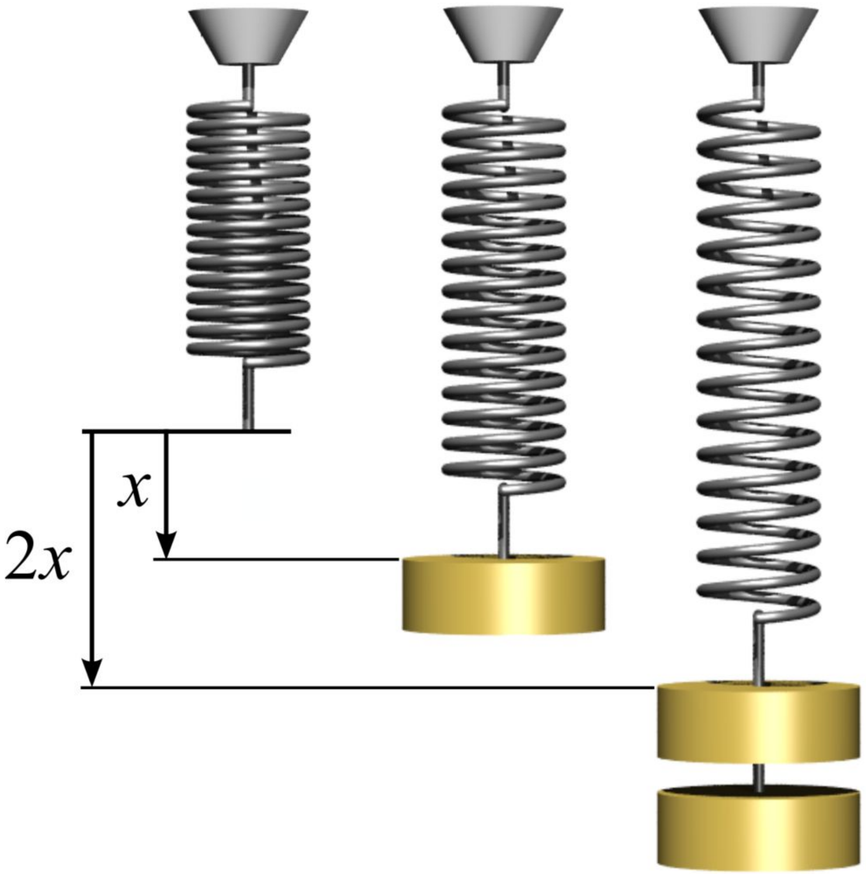
- Written in C++ (CUDA)
- Spring-mass lattices to represent soft robot structures
- Amortized constant insertion and deletion operations (lazily) on the GPU array data structure storing the spring and mass objects
- Parallel Euler (or Runge Kutta 4) integration to update soft body dynamics (instead of constraint-based approach)
- Asynchronous: minimized copying between CPU and GPU. A thread (on CPU) constantly launches GPU operations, which runs until a specified thread condition is met.



Execution Details

Goal: Alternate between simple physics calculations.

1. Make a thread for each spring and mass in the lattice for a particular time step
2. Use Hooke's Law ($F = k\Delta x$) to calculate force of spring on mass
   - Spring updates are mostly independent -> parallel
3. Sync Threads
4. Use Euler solver to update kinematics of each mass (position, velocity, acceleration)
   - Mass updates are independent of one another -> parallel
5. Sync Threads
6. Repeat for the next time step



Experiments

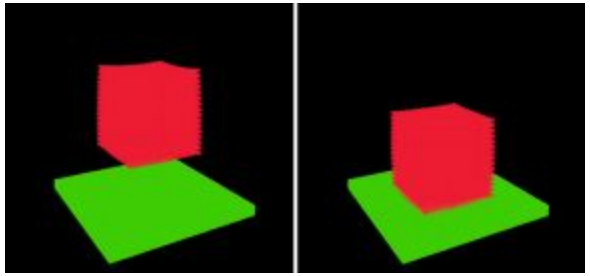
Platform

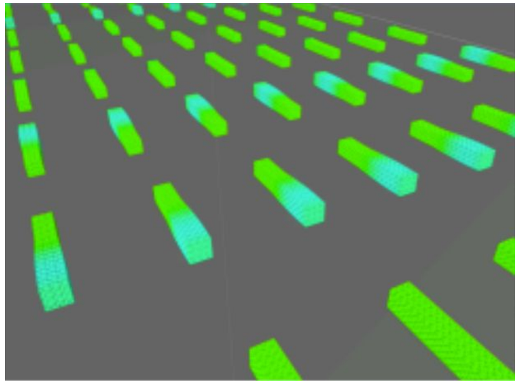
Simulations on the Titan library
GPU: qty (1) NVIDIA Titan X
CPU: qty (1) 3.7GHz Intel Core i7-8700K CPU

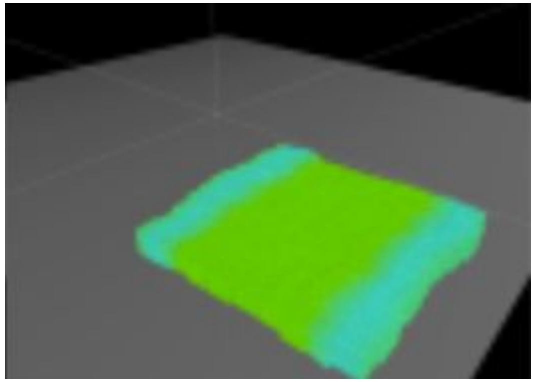
Benchmark: GPU version vs. CPU sequential version

Results

- Bouncing cube (lattice of masses and springs): 3900% perf increase on GPU
- Locomotive worm: acceptable locomotion accuracy, with 339,200 springs in parallel
- Multi-body swarms: flexible application (20 lines of C++) compared to custom simulators (likely thousands of lines)







Contributions

Main Contributions

- **A unified, highly-parallelized, GPU-accelerated simulation for large robotic systems**
- **Powerful CUDA kernel design for parallel simulations**
- **Asynchronous, between CPU and GPU**

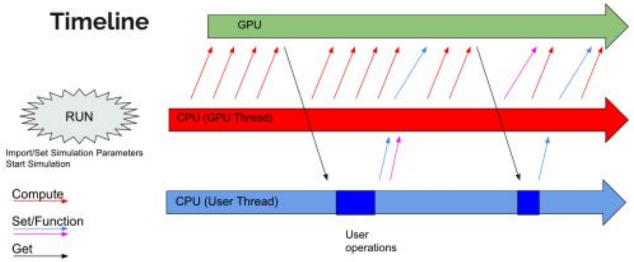
Key Ideas

- **Data structure (pointers) for fast, constant time operations**
- **Asynchronicity: only synchronize when needed between CPU and GPU, minimize bottlenecks**
- **Parallel simulations help design/optimization of large robot systems**
- **Especially beneficial for reinforcement learning robot agents**
- **Simple calculations to leverage parallelism**

Titan Summary

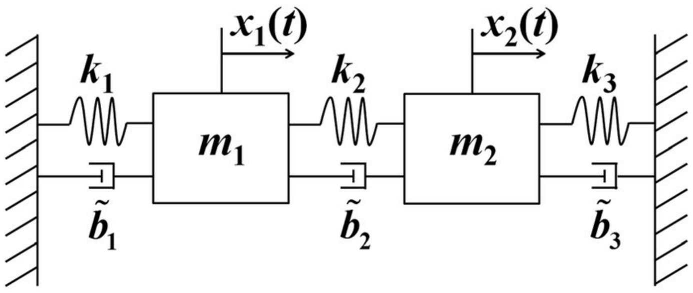
Step-Based Integration

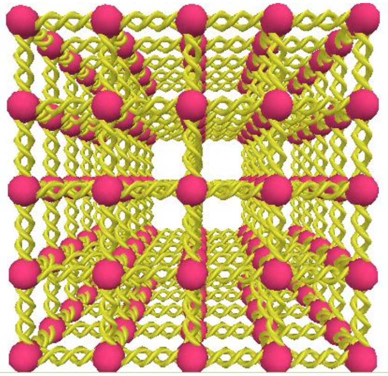
```
for i = 2:length(t)
    k1 = h * f(t(i - 1), w(i - 1, :));
    k2 = h * f(t(i - 1) + (h / 2), w(i - 1, :) + (k1 / 2));
    k3 = h * f(t(i - 1) + (h / 2), w(i - 1, :) + (k2 / 2));
    k4 = h * f(t(i - 1) + h, w(i - 1, :) + k3);
    w(i, :) = w(i - 1, :) + ((k1 + (2 * k2) + (2 * k3) + k4) /
6);
end

for i = 2:length(t)
    w(i, :) = w(i - 1, :) + h * f(t(i - 1), w(i - 1, :));
end
```



Breakpoints, Joins, and Events

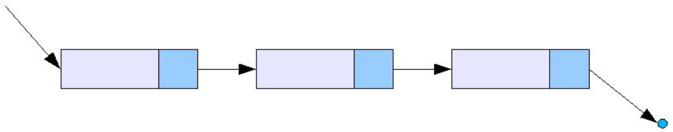
Spring Kinematics





Lattice Structures

Parallelized Computation



O(1) Insertion and Deletion

Takeaways

Things We Learned/Liked

- **Masses and springs - building blocks for any type of robot design**
- **Authors combined computer science and physics knowledge**
- **Open-source**

Disliked

- **Masses and springs - learning curve, to build a robot structure of choice**
- **Focuses a lot more on soft robotics than real-life mobile robot models**
- **Favors parallelism but sacrifices accuracy**

Room for Improvement

- **To add built-in robot designs for popular robots (Husky, Jackal, etc.)**
- **To compare with performance of other libraries**

Newer Work

- **Plancher et al. [1] - incorporates analytical gradients, better for gradient-based optimization methods, learning**

Strengths and Weaknesses

Strengths

- Uses simpler calculation methods that can be parallelized
- Independent GPU simulation and CPU analysis
- Allows topology optimization by updating robot body during runtime in constant time
- Allows input of complex material properties to simulate real materials (dynamic spring constant computation)

Weaknesses

- Too large/small time steps reduce simulation accuracy (non "well-behaved" functions causing loss of conservation of energy/momentum)
- Mitigates inaccuracy with alternative integration methods like RK4, which compromises performance compared to default Euler integration
- Experiments do not show performance improvement for multi-body systems

Future Directions/Solutions

- Add new CUDA kernels for better integration accuracy (ie: Gauss-Legendre)
- Add computation that optimizes time steps and method of integration (ie: Adaptive RK).
- Improve collision detection between meshes when necessary

Our Project Goals

Project Goals

1. Leverage CUDA programming and GPUs in robotics simulation
2. Gain practical/theoretical understanding of CUDA and GPU acceleration
3. Testing: characterize performance and efficiency of GPUs, compared to baseline processor

Potential Independent Variables

1. Spring-Mass Lattice
   a. Spring Count
   b. Mass Count
   c. Spring:Mass Ratio, Dependencies
2. Processing System
3. Integration Method

Potential Dependent Variables

1. Latency
2. Energy Efficiency

Our Idea

Use some combination of these experimental variables (control the rest) and create an experiment where we measure one (or both) of the dependent variables and compare the differences.

COLORADO SCHOOL OF MINES

Questions?

COLORADO SCHOOL OF MINES

Sources

[1] B. Plancher, S. M. Neuman, R. Ghosal, S. Kuindersma and V. J. Reddi, "GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients," 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 2022, pp. 6253-6260, doi: 10.1109/ICRA46639.2022.9812384. keywords: {Codes;Heuristic algorithms;Machine learning;Parallel processing;Libraries;Trajectory;Planning},

Pictures:

https://www.researchgate.net/figure/General-2-DOF-mass-spring-damper-system_fig2_304712628

https://www.researchgate.net/figure/A-mass-spring-model-that-represents-a-3D-lattice-of-mass-nodes-connected-by-springs_fig3_47714999

https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/

https://developer.nvidia.com/cuda/wsl

https://en.wikipedia.org/wiki/Hooke%27s_law#/media/File:Hookes-law-springs.png