GPU-Accelerated Subsystem-Based ADMM for Large-Scale Interactive Simulation

Harim Ji, Hyunsu Kim, Jeongmin Lee, Somang Lee,
Seoki An, Jinuk Heo, Youngseon Lee, Yongseok Lee, and Dongjun Lee

***Abstract*— In this paper, we implement the GPU-accelerated subsystem-based Alternating Direction Method of Multipliers (SubADMM) for interactive simulation. The challenging objective for interactive simulations is to deliver realistic results under tight performance, even for large-scale scenarios. We aim to achieve this by exploiting the parallelizable nature of SubADMM to the fullest extent. We introduce a new subsystem division strategy to make SubADMM 'GPU friendly' along with custom kernel designs and optimization regarding efficient memory access patterns. We successfully implement the GPU-accelerated SubADMM and show the accuracy and speed of the framework for large-scale scenarios, highlighted with an interactive 'Hand demo' scenario. We also show improved robustness and accuracy compared to other state-of-the-art interactive simulators with several challenging scenarios that introduce large-scale ill-conditioned dynamics problems.**

I. INTRODUCTION

Interactive physics simulation has been essential for the fields of haptics [33] and virtual reality (VR) [26], with its importance also recognized in diverse fields such as interactive computer animation [11], digital twins, and mechanical components design [1]. More recently, this interactive physics simulation is considered the key enabler for the framework of Learning from Demonstration (LfD) since it can drastically improve sample efficiency, particularly for contact-involving and long-horizon tasks [32], for which it is difficult even to finish the task autonomously.

Arguably, the three most essential requirements for this interactive physical simulation would be speed, accuracy, and scalability. First, the simulation should deliver at least 60 frames per second (fps) speed [3] with the ratio of simulation time step size to computation time near or larger than one. Second, its accuracy to the ground truth (or real physical experiment results [34]) should be small enough for believable realism or good sim-to-real performance. Third, it should allow for implementing a large-scale environment with not much compromise of speed and accuracy. All these become particularly challenging when contacts and constraints are involved among the objects.

For this, various off-the-shelf simulators have been proposed over the past decades. PhysX [10] is a state-of-the-art simulator which is widely used as an interactive simulator in both robotics [25] and haptics & VR [39], [15], [22], [27], [31]. MuJoCo physics engine [38] shows the ability to be used as an interactive simulator with a reasonable level of accuracy for successful manipulation with a simulated human hand model from a simulation platform, HAPTIX [18]. This platform is also utilized to collect human manipulation policies for imitation learning [32]. CHAI3D [17] is designed for interactive simulation with diverse algorithms for haptic feedback. This platform is widely used in medical simulation research [14], [13], [7].

Fig. 1: Overview of the 'Hand demo' scenario. The scenario contains a 281 number of rigid bodies dynamically coupled with an average number of 1773 contacts and 33 joint constraints.

However, there are limitations to these simulators. These simulators relax the actual dynamics problems to avoid tackling the non-linear complementarity problem (NCP) originating from constrained dynamics problems with contact conditions. PhysX relies on a position-based dynamics (XPBD [24]) approach, which often results in incorrect simulation even under extensive iterations [23]. Mujoco approximates the NCP to a relaxed cone complementarity problem (CCP), which introduces sudden bounces and contact softening [34]. Many works using CHAI3D adopt quasi-static dynamics [6], [40], which can handle only minimal scenarios [12]. These simulators also have limited scalability since the underlying solvers for these simulators are not directly parallelizable. While XPBD can utilize parallelizable constraint projection, Jacobi solve, this introduces a slow convergence.

In this context, we present a novel framework that tackles the exact NCP while still being interactive and scalable for large-scale scenarios. This framework is based on a GPU implementation of the recently proposed subsystem-based Alternating Direction Method of Multipliers (SubADMM [20]) algorithm. Since SubADMM can solve the exact dynamics problem and naturally handles operations for each subsystem and constraint in parallel, we exploit this parallelizable nature to the fullest extent by accelerating the SubADMM with GPU. We explain how parallelized computation can be equally distributed among threads with technical details to achieve accurate and interactive behavior, including the exact handling of the non-linear complementarity-based contact model, diverse joint constraints, and a tailored penalty parameter tuning strategy. We demonstrate the 'Hand demo'

*This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) (RS-2022-00144468, RS-2023-00208052) and the Ministry of Trade, Industry & Energy (MOTIE) of Korea (RS-2024-00419641).

The authors are with the Department of Mechanical & Aerospace Engineering, IAMD and IOER, Seoul National University, Seoul, Republic of Korea. {jiharim0911, hyunsu.kim, ljmlgh, hopelee, seoki97s, jinukheo, yslee1765, yongseoklee, djlee}@snu.ac.kr

scenario, shown in Fig. 1, and other challenging scenarios, showing that our framework can manage diverse large-scale scenarios accurately at interactive rates that are not achievable with current simulators.

This work shares similarities with [37] in the context of GPU acceleration and with [5] in the context of tackling the exact NCP using ADMM. However, [37] tackles the CCP and thus shares the same limitations with Mujoco, and [5] tackles the dual formulation of the NCP, which introduces a dense Delassus matrix into the process, making it hard to parallelize the algorithm.

The rest of the paper is organized as follows. Some preliminaries for constrained rigid body dynamics and the SubADMM framework are introduced in Sec. II. Then, implementation details for GPU-accelerated SubADMM are described in Sec. III, followed by experiments for interactive scenarios and challenging scenarios in Sec. IV. Finally, conclusions are made in Sec. V.

II. PRELIMINARY

A. Constrained Dynamics

Consider the discretized dynamics with constraints of a system, which can be divided into N number of subsystems (e.g., rigid body, articulated rigid body, deformable body, etc.). The dynamics can be written as follows:

$$A_i\hat{v}_i = b_i + \sum_{k\in\mathcal{I}_i} J_{i,k}^T\lambda_k \quad i = 1,\dots,N \tag{1}$$

where i denotes the subsystem index, $\hat{v}_i$ denotes the representative velocity, $A_i \in \mathbb{R}^{n_i\times n_i}$, $b_i \in \mathbb{R}^{n_i}$ are the mass matrix and the dynamics vector including the momentum, Coriolis force and external forces where n_i is the degrees of freedom of the ith subsystem, k denotes the constraint index, $\mathcal{I}_i$ denotes the set of constraint indices the subsystem is under, $J_{i,k} \in \mathbb{R}^{n_{c,k}\times n_i}$, $\lambda_k \in \mathbb{R}^{n_{c,k}}$ are the time-scaled constraint Jacobian and the constraint impulse where $n_{c,k}$ denotes the constraint dimension. In this paper, we consider three classes of constraints: soft, hard, and contact constraints. By dealing with the constraints at the velocity level, we can express N_c number of these classes of constraints with a unified expression:

$$(J_{f_k,k}\hat{v}_{f_k} + J_{s_k,k}\hat{v}_{s_k}, \lambda_k) \in \mathcal{C}(e_k)\ k = 1\dots N_c \tag{2}$$

where f_k, s_k are the indices of each first and second subsystems which are involved in the kth constraint, and $\mathcal{C}(e_k) \in \mathbb{R}^{2n_k}$ is the set defined by the constraint error $e_k \in \mathbb{R}$ and the constraint type. The set $\mathcal{C}(e)$ for each constraint is defined below.

1) Soft constraint: Soft constraints connect two subsystems with compliance via a spring and a damper.

$$\mathcal{C}(e) = \{(x,\lambda) \mid \lambda + ke + cx = 0\} \tag{3}$$

where k and c are the gain and the damping parameters.

2) Hard constraint: Hard constraints can introduce both holonomic and non-holonomic constraints.

$$\mathcal{C}(e) = \{(x,\lambda) \mid x + \Phi(e) = 0\} \tag{4}$$

where $\Phi(e)$ is the error fixed by Baumgarte stabilization [2].

3) Contact constraint: In this paper, we treat contact constraints using the Signorini-Coulomb condition (SCC) by expressing the set $\mathcal{C}(e)$ as follows:

$$\mathcal{C}(e) = \left\{ (x,\lambda) \;\middle|\; \begin{array}{l} 0 \le \lambda_n \perp x_n + \Phi(e) \ge 0, \\ 0 \le \delta \perp \mu\lambda_n - \|\lambda_t\|_2 \ge 0, \\ \delta\lambda_t + \mu\lambda_n x_t = 0, \\ \delta \in \mathbb{R} \end{array} \right\} \tag{5}$$

where the subscripts n and t denote the row corresponding to each contact normal and contact tangents, and $\mu \in \mathbb{R}$ denotes the friction coefficient.

Solving the constrained dynamics becomes solving (1) while satisfying (2). From the SCC in (5), this problem becomes a non-linear complementarity problem (NCP).

B. Subsystem-Based ADMM

In this paper, we utilize the SubADMM framework for its ability to tackle the NCP from Sec. II-A while handling operations for each subsystem and constraint in parallel. Here, we explain the details of the SubADMM framework.

SubADMM formulates the discrete constrained dynamics problem as an optimization problem based on augmented Lagrangian by introducing slack variables $x_k \in \mathbb{R}^{n_{c,k}}$, $z_k \in \mathbb{R}^{n_{c,k}}$ and Lagrange multiplier $u_k \in \mathbb{R}^{n_{c,k}}$ for each kth constraint. This optimization problem is solved using the ADMM [4] by executing the following iterative steps:

Step 1 Velocity update:
The representative velocity of the ith subsystem is updated as follows:

$$\begin{aligned} \forall i,\ \left(A_i + \beta_i \sum_{k\in\mathcal{I}_i} J_{i,k}^T J_{i,k}\right)\hat{v}_i^{l+1} = \\ b_i + \sum_{k\in\mathcal{I}_i} J_{i,k}^T\left(\beta_i z_{i,k}^l - u_{i,k}^l\right) \end{aligned} \tag{6}$$

where the superscript l denotes the iteration step and $\beta_i \in \mathbb{R}$ is the penalty weight of the ith subsystem.

Step 2 x-update:
For ith subsystem with kth constraint, $x_{i,k}$ represents the constraint space velocity, which is updated as below:

$$\forall k,\ x_{i,k}^{l+1} = J_{i,k}\hat{v}_i^{l+1} \tag{7}$$

Step 3 z-update:
Suppose the kth constraint involves subsystem i and j, that is, $f_k = i,\ s_k = j$. This step updates $z_{i,k}^{l+1}$ and $z_{j,k}^{l+1}$ to satisfy the constraint condition described in (2):

$$\forall k,\ (z_{i,k}^{l+1} + z_{j,k}^{l+1}, \lambda_k^{l+1}) \in \mathcal{C}(e_k) \tag{8}$$

This can be done with the following update rules:

$$\begin{aligned} \beta_i z_{i,k}^{l+1} &= \underbrace{\beta_i x_{i,k}^{l+1} + u_{i,k}^l}_{y_{i,k}^{l+1}} + \lambda_k^{l+1} \\ \beta_j z_{j,k}^{l+1} &= \underbrace{\beta_j x_{j,k}^{l+1} + u_{j,k}^l}_{y_{j,k}^{l+1}} + \lambda_k^{l+1} \end{aligned} \tag{9}$$

where $\lambda_k^{l+1} \in \mathbb{R}^{n_{c,k}}$ can be obtained by simple scalar operations for each of three types of constraints :

- Soft constraint :

$$\lambda_k^{l+1} = -\frac{ke_k + c\left(\beta_i^{-1}y_i^{l+1} + \beta_j^{-1}y_j^{l+1}\right)}{1 + c\left(\beta_i^{-1} + \beta_j^{-1}\right)}$$

– Hard constraint :

$$\lambda_k^{l+1} = -\frac{e_k + \beta_i^{-1} y_i^{l+1} + \beta_j^{-1} y_j^{l+1}}{\beta_i^{-1} + \beta_j^{-1}}$$

– Contact constraint :

$$\lambda_k^{l+1} = \Pi_{\mathcal{C}} \left(-\frac{e_k + \beta_i^{-1} y_i^{l+1} + \beta_j^{-1} y_j^{l+1}}{\beta_i^{-1} + \beta_j^{-1}} \right)$$

where $\Pi_{\mathcal{C}}$ denotes a strict projection on the friction cone [19].

Step 4 u-update:
The Lagrangian multiplier for the ith subsystem with the kth constraint, $u_{i,k}$, is updated with a simple computation:

$$\forall k,\ u_{i,k}^{l+1} = u_{i,k}^{l} + \beta_i \left(x_{i,k}^{l+1} - z_{i,k}^{l+1} \right) \tag{10}$$

Notice that **Step 1** can be done subsystem-wise parallel while **Step2~Step4** can be done constraint-wise parallel. The work in [20] exploits these advantages by implementing the framework using a parallelization library, C++ OpenMP, and achieves linear scalability, showing significant improvements in both speed and accuracy compared to existing solvers in large-scale scenarios. However, the subsystem division in [20] is based on system classes (rigid body, articulated rigid body, and deformable body). This introduces a varying size of matrices (e.g., mass matrix, constraint Jacobian, etc.) and vectors (e.g., dynamics vector, Lagrangian multiplier, etc.) between subsystems and constraints, making it challenging to distribute equal computation between threads. Due to these limitations, the original SubADMM framework cannot directly use GPUs. In this paper, we adopt a new subsystem division strategy to resolve these limitations and implement GPU-accelerated SubADMM using the CUDA Toolkit [9]. By utilizing the massive computation power of modern GPUs, we achieve sub-linear scalability and handle large scenarios (a few thousand bodies) at an interactive rate. In Sec. III, we explain details of subsystem division strategy and details on the implementation.

III. GPU-Accelerated SubADMM

A. Subsystem Division

The overall parallelization scheme of GPU-accelerated SubADMM is shown in Fig. 2. We set every subsystem as an individual rigid body with its origin in the center of mass. This lets every subsystem have an equal 6 degrees of freedom (DoF) and makes all matrices and vectors within the same category (e.g., contact Jacobian, slack variables, etc.) equally sized and computed with a unified rule. Thus, we can distribute every parallelizable computation, explained in Sec. II-B, equally along the threads with minimized divergence (each thread doing the same computations) between threads and straightforwardly design a global memory layout for coalesced memory access patterns (consecutive threads access consecutive memories). This enables SubADMM to handle large-scale scenarios with sub-linear scalability with modern GPUs. From the N number of subsystems, we introduce a mass matrix $A_i \in \mathbb{R}^{6\times6}$ and a dynamics vector $b_i \in \mathbb{R}^6$ for $i = 1 \dots N$. Notice that the mass matrix, A_i, is constant, diagonal, and only 4 values can be stored: m, I_{xx}, I_{yy}, and I_{zz}. The dynamics vector b_i includes the effect of momentum, Coriolis force, and external wrench from user input such as virtual coupling [8].

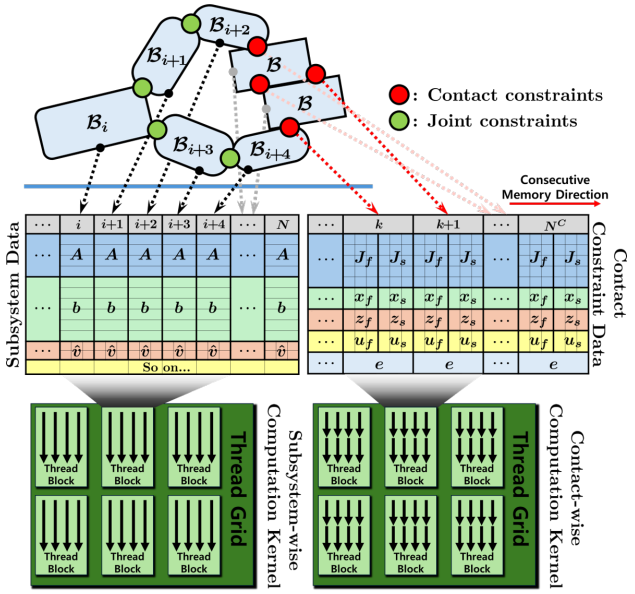


Fig. 2: GPU-accelerated SubADMM overview under a scenario with N number of subsystems, N^C number of contact constraints, and N^J number of joint constraints. Each subsystem and contact constraint is designated a memory region with the same layout pattern. GPU threads are assigned to each memory region for parallelized computation. This structure is the same for joint constraints, which is not shown in the figure. Notice that the contact-wise computation kernel is generally assigned more threads than the subsystem-wise computation kernel in large-scale scenarios.

B. Contact Constraint

From N^C number of contact features (contact points and contact normals), we can construct contact Jacobians $J_{f_k,k}^C,\ J_{s_k,k}^C \in \mathbb{R}^{3\times6}$ for $k = 1 \dots N^C$. Here, the superscript C stands for 'Contact.' Each contact constraint introduces fixed size slack variables $x_{f_k,k}^C,\ x_{s_k,k}^C,\ z_{f_k,k}^C,\ z_{s_k,k}^C \in \mathbb{R}^3$ and fixed size Lagrange multipliers $u_{f_k,k}^C,\ u_{s_k,k}^C \in \mathbb{R}^3$ as shown in Fig. 2. Contact Jacobians can be constructed in parallel before the ADMM iteration phase.

Notice that contact wrenches generated on ith body by kth contact, $F_{i,k}^C \in \mathbb{R}^6$, can be easily obtained from the relation derived from SubADMM iteration steps:

$$\lambda_{i,k}^l = -u_{i,k}^l \tag{11}$$

Thus, we only perform the following calculation after the iteration phase, which can be done in parallel for each contact:

$$F_{i,k}^C = -\frac{{J_{i,k}^C}^T u_{i,k}^C}{\delta t} \tag{12}$$

where δt is the size of the time step.

C. Joint Constraint

The original SubADMM [20] uses generalized coordinates to simulate articulated rigid bodies as single subsystems. However, in order to divide each rigid body into subsystems while still handling articulated rigid bodies, we use joint constraints. Every type of joint in our framework is configured using the local joint frame transformation matrices relative to the body frame. This kind of method for configuring the joints is also used in PhysX [30]. Although each type of joint involves a different number of constraints, we always introduce fixed-size joint Jacobians, $J_{f_k,k}^J,\ J_{s_k,k}^J \in \mathbb{R}^{6\times6}$, slack variables, $x_{f_k,k}^J,\ x_{s_k,k}^J,\ z_{f_k,k}^J,\ z_{s_k,k}^J \in \mathbb{R}^6$, Lagrangian

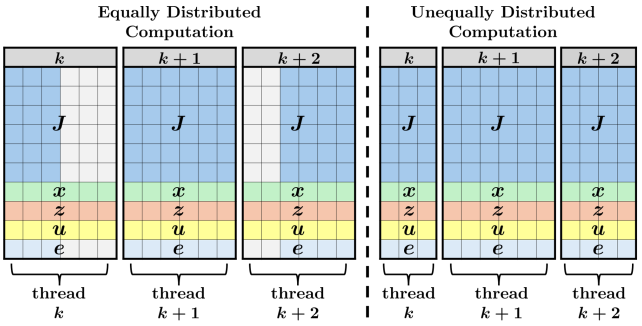


Fig. 3: Computation distribution for fixed-sized (left) and varying-sized (right) joint constraint data. Notice that each row of J is laid vertically in the memory region. The grayed-out memory region is always zero.

multipliers, $u^J_{f_k,k},\ u^J_{s_k,k} \in \mathbb{R}^6$, and joint errors, $e^J_k \in \mathbb{R}^6$. This introduces six constraints that rigidly fix the two bodies to each other. By setting the rows of the joint Jacobians and joint errors corresponding to the free constraints among those six constraints to zero, we can implement various types of joints, such as fixed, linear, planar, Cartesian, spherical, and cylindrical. At first glance, this may seem inefficient since redundant calculations such as multiplying and adding zero vectors occur. However, this approach enables each GPU thread to handle equally sized data, as shown in Fig 3, leading to minimized thread divergence, which is crucial for performance in GPU computing. If we instead introduce varying-sized Jacobains and variables, each thread should compute those with different rules, which causes divergences between threads as shown in Fig 3.

D. Penalty Weight Selection and Update

Each penalty weight parameter for the ith subsystem, β_i, is selected similarly as in the original SubADMM:

$$\beta_i = \frac{M\mathrm{Tr}(A_i)}{\mathrm{Tr}(C_i)} + c \tag{13}$$

where $M,\ c \in \mathbb{R}$ are the hyperparamters and $C_i \in \mathbb{R}^{6\times 6}$ are defined for the ith subsystem as follows:

$$C_i = \sum_{k\in\mathcal{I}_i^C} {J^C_{i,k}}^T J^C_{i,k} + \sum_{k\in\mathcal{I}_i^J} {J^J_{i,k}}^T J^J_{i,k} \tag{14}$$

The original SubADMM framework uses the pre-computed penalty weights throughout the entire iterations. However, this can cause imbalanced convergence between primal and dual residuals [35]. This imbalance can be handled by updating the penalty weights for every m iteration step. We choose the penalty weights update scheme similar to that used in [36]:

$$\beta_i \leftarrow \Pi_{[\frac{1}{\alpha},\ \alpha]} \left(\frac{\|r_{\text{primal}}\|_\infty}{\|r_{\text{dual}}\|_\infty} \right) \beta_i \tag{15}$$

where $\alpha \in \mathbb{R}$ is the hyperparameter, $r_{\text{primal}} \in \mathbb{R}^{6N^C+12N^J}$ and $r_{\text{dual}} \in \mathbb{R}^{6N}$, are the primal and the dual residuals defined as follows:

$$r_{\text{primal}} = \begin{bmatrix} x^{C,l+1} - z^{C,l+1} \\ x^{J,l+1} - z^{J,l+1} \end{bmatrix} \tag{16}$$

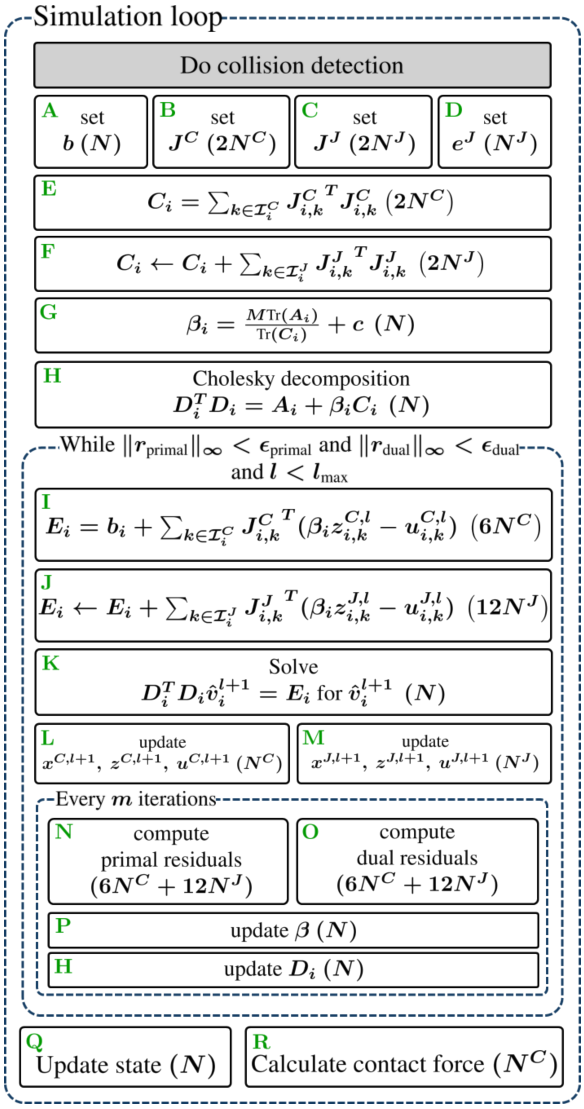


Fig. 4: Kernels and kernel scheduling for complete simulation loop including collision detection and GPU-accelerated SubADMM. Each block labeled with a bold alphabet letter represents a custom-built kernel.

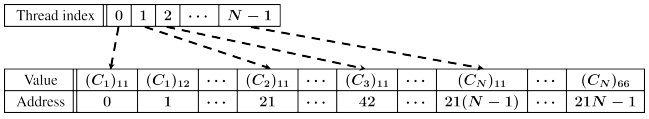
$$(r_{\text{dual}})_{6i-5:6i} = \beta_i \left(\sum_{k\in\mathcal{I}_i^C} {J^C_{i,k}}^T (z^{C,l}_{i,k} - z^{C,l+1}_{i,k}) + \sum_{k\in\mathcal{I}_i^J} {J^J_{i,k}}^T (z^{J,l}_{i,k} - z^{J,l+1}_{i,k}) \right) \tag{17}$$

Notice that we clamp the update ratio in the $[\frac{1}{\alpha},\ \alpha]$ interval to avoid excessive updates. This penalty weight update scheme balances the convergence of primal and dual residuals and suppresses a periodic surge of either residual. This can prevent stopping the iteration during these surges when applying a fixed number of iterations.

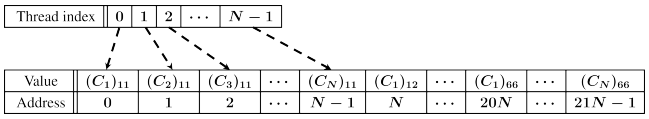
The hyper-parameters M, c, and α can be tuned, but we found that setting $M = 16,\ c = 10^5,\ \alpha = 4$ shows good performance, which is set as default.

E. GPU Kernel

The detailed algorithm is shown in Fig. 4. We design custom GPU kernels for every block labeled with a bold



(a) Uncoalesced global memory access



(b) Coalesced global memory access

Fig. 5: Global Memory access pattern for two kinds of memory layout.

alphabet appearing in Fig. 4, where the number in the parenthesis is the number of GPU threads assigned for the kernel call. When assembling the C_i matrices for each body, as shown in the kernels labeled as **E** and **F** in Fig. 4, we parallelize the execution at the constraint level rather than the body level. This can further exploit the computation power of the GPU since the number of constraints is generally much larger than the number of bodies, and it can also reduce thread divergences coming from different numbers of constraints between the bodies. However, naively using the add operation can cause race conditions (different threads writing on the same memory simultaneously) since different threads for different constraints may add to the same C_i matrix for the same body. We thus use the `atomicAdd` operation provided by the CUDA Toolkit [28] to avoid this issue. This style of kernel design is also used for assembling $E_i \in \mathbb{R}^6$ vectors for each ith subsystem as shown in the kernels labeled as **I** and **J** and constructing the dual residual, defined in (III-D), shown in the kernel labeled as **O**. Notice that some kernels that computes b_i, $J^C_{i,k}$, $J^J_{i,k}$, and e^J_k, each labeled as **A**, **B**, **C**, and **D**, are decoupled to each other. In this case, we can schedule the kernels to be called asynchronously to keep as many threads busy as possible. This asynchronous scheduling is also used to schedule kernels that update the slack variables and the Lagrangian multipliers for each contact constraint and joint constraint, labeled as **N** and **O**, and kernels that update the state of each body and calculate the contact forces, labeled as **Q** and **R**.

For acquiring the contact features, we use a custom-built collision detector implemented with the CUDA toolkit, appearing as a gray block in Fig. 4. Details for these are out of the scope of this paper.

F. Memory Layout

When performing computation with GPUs, coalesced global memory access is crucial for performance [29]. In this section, we explain how we design memory layout patterns and achieve coalesced global memory access. Consider storing the matrix C_i for each ith subsystem defined in (14) for all subsystems. Since each C_i is a 6×6 symmetric matrix, we only need to store 21 elements. A naive approach to store every matrix C_i is to assign a consecutive memory region of size $21N$ and store each matrix C_i serially as shown in Fig. 5a. This leads to uncoalesced memory access when performing body-wise parallel computation. For example, when computing β in (13), each ith thread computes $\mathrm{Tr}(C_i)$. Thus, each ith thread should fetch $(C_i)_{11}$ from memory, which causes an uncoalesced memory access pattern as shown in Fig. 5a. We instead store each C_i with a stride of N as shown in Fig. 5b, which leads to coalesced memory access. This also leads to coalesced memory access for other operations with the matrix C_i, such as kernel labeled as **H** in Fig. 4. This kind of optimization is possible since we divide the whole system into subsystems to have the same DoF as explained in Sec. II-A, enabling each thread to fetch the same amount of memories and to know what memory address to access based on its thread index and the number of subsystems in the scene. Similar strategies are used for storing every vector and matrix.

Fig. 6: Experiment setup and scene captures for 'Hand demo' scenario.

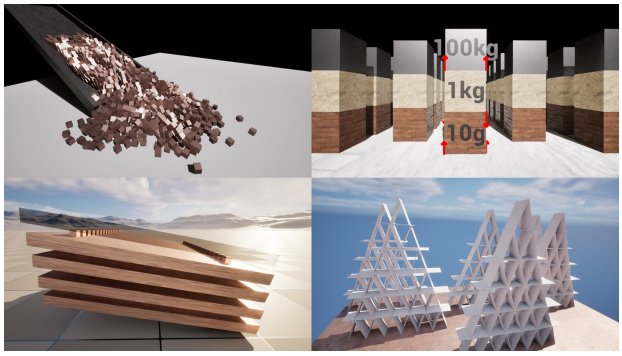


Fig. 7: Scene capture for stress test scenarios. From left to right, top to bottom: 'Sliding cubes,' 'Vertical stacks,' 'Oblique stacks,' and 'Card houses' after 912 seconds of simulation time. For 'Vertical stacks,' contact forces for the front boxes are rendered as red arrows.

IV. Experiments

In this section, we conduct several test scenarios. Every scenario is run with an Intel Core i9-13900K CPU and an RTX 4080 GPU. Implementation is done with a custom-built Unreal Engine 5.3 plugin, rendering quality set to 'High.' The hyperparameters from Sec. III-D are set as the default for every scenario. The input of the hand motion used in the 'Hand demo' scenario, which will be described in Sec. IV-A, is obtained using Visual Inertial Skeletal Tracking (VIST) [21]. The overall results and settings for each scenario are shown in Table. I. We also recommend that the readers see the supplemental video for more details.

A. Hand demo

To highlight our simulator's ability to handle large-scale scenarios accurately and interactively, we implement a 'Hand

TABLE I: Settings and performance of the solver for test scenarios described in Sec. IV. Residuals and time cost (GPU-accelerated SubADMM) are calculated as the average for the scenario's number of frames.

Scenario	DoFs	Iterations	δt (ms)	Time Cost (ms)	$\|r_{\text{primal}}\|_\infty$	$\|r_{\text{dual}}\|_\infty$	Contact Average (Max)	Joints
Hand demo ($m = 20$)	1686	100	4	5.00	1.48E-5	3.99E-5	1773(1901)	33
Sliding 100 cubes ($m = 30$)	600	120	5	3.82	2.26E-7	2.05E-7	488(456)	-
Sliding 500 cubes ($m = 30$)	3000	120	5	4.15	3.75E-6	3.62E-6	3864(3968)	-
Sliding 1000 cubes ($m = 30$)	6000	120	5	4.85	5.19E-6	5.10E-6	8961(9140)	-
Sliding 1500 cubes ($m = 30$)	9000	120	5	5.26	9.37E-6	9.16E-6	14813(15972)	-
Sliding 2000 cubes ($m = 30$)	12000	120	5	5.51	1.05E-5	1.72E-5	21156(22660)	-
Vertical stacks ($m = 50$)	2160	200	5	5.81	1.09E-5	7.47E-05	1438(1440)	-
Oblique stacks ($m = 30$)	840	150	5	4.52	6.97E-7	8.62E-6	1087(1088)	-
Card houses ($m = 100$)	2376	300	1	9.12	3.06E-8	6.64E-7	2551(2680)	-

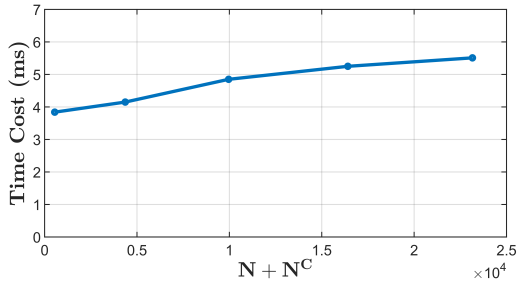


Fig. 8: Time cost of GPU-accelerated SubADMM for sliding cubes scenarios

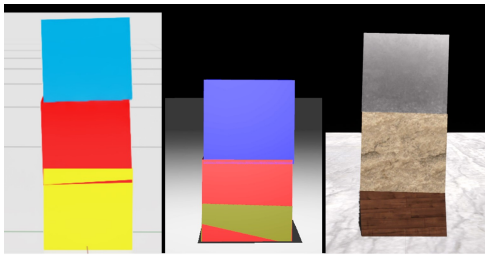


Fig. 9: Robustness test with odd mass ratios. From left to right: PhysX, Mujoco, and Chaos.

demo' scenario in which a user can interact with the simulation in real-time using a virtual hand. This scenario includes tasks such as dexterous manipulation with various objects (apple, small cube, solderer, thin spoon, wooden toy car, etc.), stacking objects (tricky cube), and grasping heavy objects (golden cube, Stanford bunny). To deliver physically realistic results, the simulator should solve the NCP with hundreds of objects and thousands of constraints in real-time. It also should be robust for ill-conditioned problems arising from the interaction between odd mass ratios, such as between the fingertip (10g) and the golden cube (1kg) or the Stanford bunny (1.5kg). As shown in Table. I, our simulator can achieve such tight requirements. We emphasize that contact wrenches and points between the finger and objects, explained in Sec. III-B, are available for use in other haptics devices. Such integration with these devices is left as a future work. We implement the same scenario using Chaos, a physics simulator for Unreal Engine 5, with the same time step as the 'Hand demo' scenario and 256 number of position, velocity, and projection iterations. We find that Chaos cannot deliver enough accuracy for the dexterous manipulation tasks. The detailed results are provided in the supplemental video.

B. Stress test

1) Scalability: We test the scalability of GPU-accelerated ADMM with a 'Sliding cubes' scenario. We test the scenario with 100, 500, 1000, 1500, and 2000 cubes. As shown in Fig. 8, GPU-accelerated SubADMM demonstrates sub-linear scalability regarding the number of bodies and contact constraints.

2) Robustness under odd mass ratio: As described in Sec. IV-A, the interaction between odd mass ratios should be robustly handled for interactive simulations. We further test the robustness under odd mass ratios with two scenarios: 'Vertical stacks' and 'Oblique stacks.' In 'Vertical stacks, ' a 100kg box, slightly perturbed with a pitch angle, is dropped on top of a 1kg box sitting on a 10g box. The scenario contains 120 number of these stacks. In 'Oblique stacks,' 10kg plates are stacked upon 50g cubes. The friction coefficient of each plate and cube is set to 3, which is enough to prevent sliding. These scenarios are simulated successfully, showing that our framework can handle such large-scale and ill-conditioned problems in real-time. We test the same scenarios using Chaos and other state-of-the-art interactive simulators, PhysX and Mujoco, explained in Sec. I, with an extensive number of iterations. For 'Vertical stacks,' every simulator fails to stack the heavy block, as shown in Fig. 9. For 'Oblique stacks,' Chaos exhibits springy motion but successfully maintains the stack, while PhysX and Mujoco show sliding even if we increase the friction coefficient to 10. To enable stacking, PhysX needs techniques such as 'sleeping' (ignoring consecutive small velocities), and Mujoco needs an additional 'No slip iteration' but with a large trade-off (more than 10ms) in computation time. The detailed settings and results are provided in the supplemental video.

3) Stacking stability: We test the stacking stability of our simulator with a 'Card houses' scenario. The challenging aspects of this scenario are well explained in [16]. The six card houses are initially shocked due to small gaps between each card in the first frame, and quickly stabilize. We observe stable behavior for more than 15 minutes of simulation time even though no stabilization methods, such as sleeping, are used.

V. CONCLUSIONS

In this paper, we implement the GPU-accelerated SubADMM, which exploits the parallelizable nature of SubADMM to the fullest extent. We introduce a new subsystem division strategy to make SubADMM 'GPU friendly' and apply optimization techniques with GPU programming. With an interactive 'Hand demo' scenario, we demonstrate the ability of this new framework to handle large-scale scenarios accurately at an interactive rate. We also show improved scalability compared to the original SubADMM and accuracy and robustness compared with several state-of-the-art simulators with challenging scenarios.

REFERENCES

[1] Inc Ansys. Ansys vrxperience hmi. https://www.ansys.com/content/dam/product/optical/vrxperience/ansys-vrxperience-hmi-datasheet.pdf, 2024. Accessed: 2024-09-15.

[2] Joachim Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer methods in applied mechanics and engineering*, 1(1):1–16, 1972.

[3] Jan Bender, Kenny Erleben, and Jeff Trinkle. Interactive simulation of rigid body dynamics in computer graphics. In *Computer Graphics Forum*, volume 33, pages 246–270. Wiley Online Library, 2014.

[4] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[5] Justin Carpentier, Quentin Le Lidec, and Louis Montaut. From compliant to rigid contact simulation: a unified and efficient approach. In *Proceedings of the 20th Robotics: Science and Systems (RSS) Conference*. HAL, July 2024. hal-04588906.

[6] Sonny Chan, François Conti, Nikolas H Blevins, and Kenneth Salisbury. Constraint-based six degree-of-freedom haptic rendering of volume-embedded isosurfaces. In *2011 IEEE World Haptics Conference*, pages 89–94. IEEE, 2011.

[7] Xiaojun Chen, Pengjie Sun, and Denghong Liao. A patient-specific haptic drilling simulator based on virtual reality for dental implant surgery. *International journal of computer assisted radiology and surgery*, 13:1861–1870, 2018.

[8] J Edward Colgate, Michael C Stanley, and J Michael Brown. Issues in the haptic display of tool use. In *Proceedings 1995 IEEE/RSJ international conference on intelligent robots and systems. Human robot interaction and cooperative robots*, volume 3, pages 140–145. IEEE, 1995.

[9] Nvidia Corporation. Cuda c++ programming guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 2024. Accessed: 2024-09-01.

[10] Nvidia Corporation. Nvidia physx. https://developer.nvidia.com/physx-sdk, 2024. Accessed: 2024-08-27.

[11] DeepMotion. Physics based multiple 3 point tracked vr avatar interaction. https://www.youtube.com/watch?v=SaGezfGzFQs, January 2018. YouTube video.

[12] Mathew Halm and Michael Posa. A quasi-static model and simulation approach for pushing, grasping, and jamming. In *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*, pages 491–507. Springer, 2020.

[13] Jianlong Hao, Xiaoliang Xie, Gui-Bin Bian, Zeng-Guang Hou, and Xiao-Hu Zhou. Development of a multi-modal interactive system for endoscopic endonasal approach surgery simulation. In *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 143–148. IEEE, 2016.

[14] Jan Hergenhan, Jacqueline Rutschke, Michael Uhl, Stefan Escaida Navarro, Bjorn Hein, and Heinz Worn. A haptic display for tactile and kinesthetic feedback in a chai 3d palpation training scenario. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 291–296, 2015.

[15] Markus Höll, Markus Oberweger, Clemens Arth, and Vincent Lepetit. Efficient physics-based implementation for realistic hand-object interaction in virtual reality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 175–182, 2018.

[16] Danny M Kaufman, Shinjiro Sueda, Doug L James, and Dinesh K Pai. Staggered projections for frictional contact in multibody systems. In *ACM SIGGRAPH Asia 2008 papers*, pages 1–11. 2008.

[17] O Khatib and K Salisbury. The chai libraries. In *Proceedings of Eurohaptics*, 2003.

[18] Vikash Kumar and Emanuel Todorov. Mujoco haptix: A virtual reality system for hand manipulation. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 657–663. IEEE, 2015.

[19] Jeongmin Lee, Minji Lee, and Dongjun Lee. Large-dimensional multibody dynamics simulation using contact nodalization and diagonalization. *IEEE Transactions on Robotics*, 39(2):1419–1438, 2022.

[20] Jeongmin Lee, Minji Lee, and Dongjun Lee. Modular and parallelizable multibody physics simulation via subsystem-based admm. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10132–10138. IEEE, 2023.

[21] Yongseok Lee, Wonkyung Do, Hanbyeol Yoon, Jinuk Heo, WonHa Lee, and Dongjun Lee. Visual-inertial hand motion tracking with robustness against occlusion, interference, and contact. *Science Robotics*, 6(58):eabe1315, 2021.

[22] Christos Lougiakis, Jorge Juan González, Giorgos Ganias, Akrivi Katifori, Ioannis-Panagiotis, and Maria Roussou. Comparing physics-based hand interaction in virtual reality: Custom soft body simulation vs. off-the-shelf integrated solution. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 743–753, 2024.

[23] Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Viktor Makoviychuk. Non-smooth newton methods for deformable multi-body dynamics. *ACM Transactions on Graphics (TOG)*, 38(5):1–20, 2019.

[24] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. Xpbd: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, pages 49–54, 2016.

[25] Malte Mosbach, Kara Moraw, and Sven Behnke. Accelerating interactive human-like manipulation learning with gpu-based simulation and high-quality demonstrations. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 435–441. IEEE, 2022.

[26] Syed T Mubarrat, Antonio Fernandes, and Suman K Chowdhury. A physics-based virtual reality haptic system design and evaluation by simulating human-robot collaboration. *IEEE Transactions on Human-Machine Systems*, 2024.

[27] Kiran Nasim and Young J Kim. Physics-based assistive grasping for robust object manipulation in virtual reality. *Computer Animation and Virtual Worlds*, 29(3-4):e1820, 2018.

[28] NVIDIA. *CUDA C Programming Guide - Atomic Functions*, 2024. Accessed: 2024-09-14.

[29] NVIDIA. How to access global memory efficiently in cuda c/c++ kernels. https://developer.nvidia.com/blog/how-access-global-memory-efficiently-cuda-c-kernels/, 2024. Accessed: 2024-09-14.

[30] NVIDIA. Physx sdk documentation - joints. https://nvidia-omniverse.github.io/PhysX/physx/5.1.0/docs/Joints.html, 2024. Accessed: 2024-09-14.

[31] Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, John A Castro-Vargas, Sergio Orts-Escolano, and Jose Garcia-Rodriguez. A visually realistic grasping system for object manipulation and interaction in virtual reality environments. *Computers & Graphics*, 83:77–86, 2019.

[32] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[33] Kenneth Salisbury, Francois Conti, and Federico Barbagli. Haptic rendering: introductory concepts. *IEEE computer graphics and applications*, 24(2):24–32, 2004.

[34] Dongwon Son, Hyunsoo Yang, and Dongjun Lee. Sim-to-real transfer of bolting tasks with tight tolerance. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9056–9063. IEEE, 2020.

[35] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[36] Alessandro Tasora, Dario Mangoni, Simone Benatti, and Rinaldo Garziera. Solving variational inequalities and cone complementarity problems in nonsmooth dynamics using the alternating direction method of multipliers. *International Journal for Numerical Methods in Engineering*, 122(16):4093–4113, 2021.

[37] Alessandro Tasora, Dan Negrut, and Mihai Anitescu. Gpu-based parallel computing for the simulation of complex multibody systems with unilateral and bilateral constraints: an overview. *Multibody Dynamics: Computational Methods and Applications*, pages 283–307, 2010.

[38] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[39] Mickeal Verschoor, Daniel Lobo, and Miguel A Otaduy. Soft hand simulation for smooth and robust natural interaction. In *2018 IEEE conference on virtual reality and 3D user interfaces (VR)*, pages 183–190. IEEE, 2018.

[40] Yanni Zou, Peter X Liu, Qiangqiang Cheng, Pinhua Lai, and Chunquan Li. A new deformation model of biological tissue for surgery simulation. *IEEE transactions on cybernetics*, 47(11):3494–3503, 2016.