Modular and Parallelizable Multibody Physics Simulation via Subsystem-Based ADMM

Jeongmin Lee, Minji Lee, and Dongjun Lee

Abstract— In this paper, we present a new multibody physics simulation framework that utilizes the subsystem-based structure and the Alternating Direction Method of Multiplier (ADMM). The major challenge in simulating complex high degree of freedom systems is a large number of coupled constraints and large-sized matrices. To address this challenge, we first split the multibody into several subsystems and reformulate the dynamics equation into a subsystem perspective based on the structure of their interconnection. Then we utilize ADMM with our novel subsystem-based variable splitting scheme to solve the equation, which allows parallelizable and modular architecture. The resulting algorithm is fast, scalable, versatile, and converges well while maintaining solution consistency. Several illustrative examples are implemented with performance evaluation results showing advantages over other state-of-the-art algorithms.

I. INTRODUCTION

Physics simulation enables synthetic data acquisition in a virtual environment to reduce the cost, time, and risk of data-driven methods that are increasingly emerging in robotics [1]–[4]. Further, in terms of finding a solution to the modeled system dynamics equation, it can be directly utilized in various problems such as trajectory optimization [5], system identification [6], etc. As such, the importance of simulation is increasingly being emphasized, with a plethora of open-source software [7]–[11].

One of the most important concerns in robotic simulation research is how to obtain data that is accurate and efficient in terms of computation time. This is a challenging problem and implies the question of how to formulate the dynamics of systems, and which algorithms to use to solve them. Since it includes many factors such as discrete-time integration, various types of constraints, friction, system-induced sparsity, numerical algorithms, etc., various methods have been proposed for decades. However, simulation of a high degree of freedom (DOF) system with many constraints is still a difficult problem [12]. This is because, fundamentally, all system DOFs are dynamically coupled, so a constraint force acting on a part of the system in general affects the entire system. This leads many algorithms to use large-size matrix operations (e.g., factorization) or possibly excessive numerical iterations.

In this paper, we attempt to solve this challenge by developing a novel subsystem-based simulation approach, that is simple, modular, and suitable for parallelization while ensuring the solution consistency and accuracy. For this, we first split the multibody system into several subsystems and reformulate the conventional expressions of discrete-time constrained dynamics into a subsystem perspective. Then inspired by the structure of the Alternating Direction Method of Multiplier (ADMM [13]), we present a novel variable splitting scheme and solution process on the reformulated dynamics equation. This then reduces the solution process to iterations of 1) block-decomposed linear solving of the subsystem dynamics equation (allowing for complete parallelization) and 2) parallel resolution of all the constraint interfaces (with scalar operation only), ensuring low per-iteration computation time and scalability. Moreover, our method can handle with various types of constraints and also exhibits stable convergence properties, rendering itself as an appealing alternative for robot simulation. Several multibody simulation examples are then implemented and demonstrated to show the validity of our framework.

The conventional approach to dealing with constrained dynamics equations is applying pivoting algorithms [14] after formulating a linear complementarity problem [15]. However, since these direct methods require high computational complexity and polygonal friction cone approximation, iteration-based methods have been more widely used in recent studies. One of the popular approaches is using Gauss-Seidel type iteration per constraint [16]–[19]. These methods scale well for particle-based systems, but not well for systems with generalized coordinate representation (e.g., robot joint angles) and complex internal constraints (e.g., finite element). Several researches tackle this issue [20]–[22] by taking an operator splitting type method. However, their applicability to rigid-deformable objects with various constraint types is limited and they still have to deal with the full system size matrices. Another direction is to apply a Newton-type iteration over the cost including the constraint [23]–[25]. Despite their good convergence property, their second-order nature could be problematic for large-sized problems as they require multiple linear problem resolutions.

Our subsystem-based ADMM algorithm may be regarded as an opportunistic compromise between the two directions described above. By properly separating primal-dual relationships based on subsystems, we circumvent the burdens of handling both with many constraints and large-sized matrices. In this context, [26]–[28] share some conceptual similarities with our framework proposed here. However,

their applicability is much limited as compared to our framework, since 1) they need factorization to construct coupling interface equation, which is costly especially as the size of the subsystems grows, and 2) their constructed coupling dynamics is dense, therefore only a small number of interconnection between subsystems is permitted for reasonable performance. In contrast, by utilizing the structural peculiarity of ADMM, our proposed framework can handle all the constraints in a decoupled manner for each iteration phase, thereby not only substantially improving the algorithmic efficiency but also allowing for its extension to a wide range of multibody systems. We also note that [21], [29], [30] employ ADMM structure in simulation. However, their full system level approaches still require large-sized matrix operations. In contrast, our subsystem-based variable splitting gives a rise to small-sized and parallelized structures, making our scheme much more efficient and scalable.

The rest of the paper is organized as follows. Some background materials about constrained dynamics simulation and ADMM will be explained in Sec. II. Then our simulation framework using subsystem-based ADMM will be described in Sec. III. Some illustrative examples and performance evaluation will be presented in Sec. IV. Finally, discussions and concluding remarks are given in V.

II. PRELIMINARY

A. Constrained Dynamics

Consider following continuous-time dynamics:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + d\psi^T = f + J(q)^T \lambda \quad (1)$$

where $q \in \mathbb{R}^n$ is the generalized coordinate variable of system, $M(q), C(q,\dot{q}) \in \mathbb{R}^{n \times n}$ are the mass, Coriolis matrix, $d\psi^T \in \mathbb{R}^n$ is the potential action, $f \in \mathbb{R}^n$ is the external force, and $\lambda \in \mathbb{R}^{n_c}, J(q) \in \mathbb{R}^{n_c \times n}$ are the constraint impulse and Jacobian with n, n_c being the system/constraint dimension. The discretized version of the dynamics is

$$M_k \frac{v_{k+1} - v_k}{t_k} + C_k v_k + d\psi_k^T = f_k + J_k^T \lambda_k$$
$$\hat{v}_k = \frac{v_k + v_{k+1}}{2}, \quad q_{k+1} \leftarrow \text{update}(q_k, \hat{v}_k, t_k) \quad (2)$$

where k denotes the time step index, $M_k = M(q_k)$, $C_k = C(q_k, v_k)$, t_k is the step size, and $v_k, \hat{v}_k \in \mathbb{R}^n$ are the current, representative velocity [31] of each time step. Although we use the midpoint rule here, it can be transformed into other integration rules. From now on, time step index k will be omitted for simplicity but note that all components are still time(step)-varying.

In this paper, we deal with the constraints at the velocity level as in many other works [7]–[9], which is stable but is based on linearization. Issues that may arise from linearization can be suppressed by adopting multiple-linearization as in [21] or re-linearization [32], and these will be integrated into our future implementation. We classify the system constraints into three categories: soft, hard, and contact constraints:

1) Soft constraint: Soft constraints are originated from the elastic potential energy of the system (e.g., finite element). If the j-th constraint is soft, impulse can be written as

$$\lambda_j = -k_j(e_j + \alpha_j J_j \hat{v}) \quad (3)$$

where $e_j \in \mathbb{R}$ and $J_j \in \mathbb{R}^{1 \times n}$ are the (t-scaled) error and Jacobian for soft constraint, k_j is the gain parameter, and $\alpha_j > 0$ is the variable that includes an implicit term with constraint-space damping. The value of α_j is associated with system energy behavior, see [31], [33] for more details.

2) Hard constraint: Hard constraints ensure that equations and inequalities for the system are strictly satisfied (e.g., joint limit), including holonomic and non-holonomic types. If the j-th constraint is hard, it has the form of

$$J_j \hat{v} + e_j \geq 0 \quad (4)$$

where $e_j \in \mathbb{R}$ and $J_j \in \mathbb{R}^{1 \times n}$ denote the error and Jacobian for hard constraint. Here, the error can be determined by methods such as Baumgarte stabilization [34].

3) Contact constraint: Contact condition is typically the most demanding type since it includes non-linear complementarity relation between primal (i.e., velocity) and dual (i.e., impulse) variables. We take Signorini-Coulomb condition [35], which is the most universal expression for frictional contact. If the j-th constraint is contact, the relation is

$$0 \leq \lambda_{j,n} \perp J_{j,n}\hat{v} + e_{j,n} \geq 0$$
$$0 \leq \delta_j \perp \mu_j \lambda_{j,n} - \|\lambda_{j,t}\| \geq 0 \quad (5)$$
$$\delta_j \lambda_{j,t} + \mu_j \lambda_{j,n} J_{j,t}\hat{v} = 0$$

where $\perp$ denotes complementarity, $e_{j,n} \in \mathbb{R}$ and $J_{j,n} \in \mathbb{R}^{1 \times n}$ denote the error and Jacobian for contact normal, $J_{j,t} \in \mathbb{R}^{2 \times n}$ is the Jacobian for contact tangential, and μ_j is the friction coefficient and δ_j is the auxiliary variable. There are three situations induced by the condition - open ($\lambda_{j,n} = 0$), stick ($\lambda_{j,n} > 0, \delta_j = 0$), and slip ($\lambda_{j,n} > 0, \delta_j > 0$).

B. Alternating Direction Method of Multiplier

Alternating direction method of multiplier (ADMM [13]) is the method to solve the following optimization problem:

$$\min_{x,z} f(x) + g(z) \quad \text{s.t.} \quad Px + Qz = r$$

Based on the augmented Lagrangian defined as,

$$\mathcal{L} = f(x) + g(z) + u^T(Px + Qz - r) + \frac{\beta}{2}\|Px + Qz - r\|^2$$

where u is the Lagrange multiplier and $\beta > 0$ is the penalty weight. ADMM iteratively performs alternating minimization of $\mathcal{L}$ with respect to each variable. The iteration process of ADMM can be summarized as follow:

$$x^{l+1} = \arg\min_x \left(f(x) + \frac{\beta}{2}\|Px + Qz^l - r + \frac{1}{\beta}u^l\|^2 \right)$$
$$z^{l+1} = \arg\min_z \left(g(z) + \frac{\beta}{2}\|Px^{l+1} + Qz - r + \frac{1}{\beta}u^l\|^2 \right)$$
$$u^{l+1} = u^l + \beta(Px^{l+1} + Qz^{l+1} - r)$$

where l is the loop index. ADMM is known as robust, simple to implement, and able to attain independent resolution with respect to each variable [13], [36].
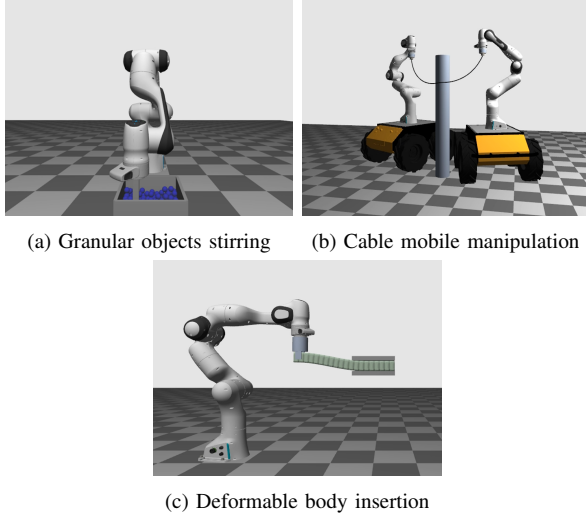
(a) Granular objects stirring   (b) Cable mobile manipulation

(c) Deformable body insertion

Fig. 1: Motivating examples and implementations of our subsystem-based ADMM framework.

III. SIMULATION VIA SUBSYSTEM-BASED ADMM

A. Subsystem Division

Our approach starts by dividing the entire system into several subsystems. See Fig. 1 for our motivational examples. We assume that objects in typical robotics simulation can be broadly classified into three main classes: rigid body, deformable body, and robot manipulator. In many cases, each rigid body and manipulator is treated as a single subsystem (as in Fig. 1a). This is intuitive and allows for preserving modularity for each class (e.g., constant 6 DOF inertia for a rigid body, articulated structure of manipulator). However, for the situations in which a large number of rigid bodies are connected through soft coupling (e.g., cable modeling as in Fig. 1b), we find that defining a new subsystem by assembling several rigid body instances can give better performance. In the case of a deformable body, its dimension is often so high to conveniently treat it as a single subsystem and causes an imbalance with other objects. Thus, we split each deformable object into several pieces and consider each as a subsystem, while they are jointly connected using hard constraints (as in Fig. 1c).

B. Subsystem-Based Dynamics Reformulation

Now consider that the whole system is divided as described in Sec. III-A. If all the subsystems are completely independent (i.e., no coupling), we can formulate each subsystem dynamics using the structure of (2) and write in the following compressed form:

$$A_i \hat{v}_i = b_i + J_{in,i}^T \lambda_{in,i} \quad (6)$$

for $i = \{1, \cdots, N\}$ where N is the number of subsystem, $A_i \in \mathbb{R}^{n_i \times n_i}, b_i \in \mathbb{R}^{n_i}$ are the subsystem dynamics matrices/vectors, and $\lambda_{in,i} \in \mathbb{R}^{n_{in,i}}, J_{in,i} \in \mathbb{R}^{n_{in,i} \times n_i}$ are the *intra*-subsystem constraint impulse/Jacobian while $n_i, n_{in,i}$ are the dimension of subsystem/intra-subsystem constraint. Here, each A_i is a symmetric positive definite from the mass matrix and energy Hessian approximation [21], [23], [35].

Remark 1: Since (3) is in closed-form of $\hat{v}$, it can be directly included in A_i, b_i, or still be remained in $\lambda_{in,i}$ of (6). Currently, this is optional, as both these schemes work fine in our framework.

Now to take into account the *coupling* constraints between the subsystems, we must add a coupling impulse and the dynamics of the entire system can be written as

$$\begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_N \end{bmatrix} \begin{bmatrix} \hat{v}_1 \\ \vdots \\ \hat{v}_N \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix} + \begin{bmatrix} J_{in,1}^T \lambda_{in,1} \\ \vdots \\ J_{in,N}^T \lambda_{in,N} \end{bmatrix} + J_{cp}^T \lambda_{cp} \quad (7)$$

where $\lambda_{cp} \in \mathbb{R}^{n_{cp}}$ and $J_{cp} \in \mathbb{R}^{n_{cp} \times n}$ are the inter-subsystem coupling impulse and Jacobian. Then (7) can be rewritten as

$$A\hat{v} = b + J_{in}^T \lambda_{in} + J_{cp}^T \lambda_{cp} \quad (8)$$

Note that this new subsystem-based dynamics formulation (8) does not relax any physical condition, while still allowing to utilize the block-diagonal structure of A, even for complex multibody scenarios.

C. ADMM-Based Solver

To solve (8) using ADMM, we start by defining the following function:

$$f_i(\hat{v}_i, x_i) = \frac{1}{2}\hat{v}_i^T A_i \hat{v}_i - b_i^T \hat{v}_i + \mathcal{I}(J_{c,i}\hat{v}_i = x_i) \quad (9)$$

where $x_i \in \mathbb{R}^{n_{c,i}}$ is the auxiliary variable, $\mathcal{I}$ is the indicator function, and $J_{c,i} \in \mathbb{R}^{n_{c,i} \times n_i}$ is the row stack of $J_{in,i}$ and $J_{cp,i}$ while $n_{c,i}$ is the summation of intra- and inter-subsystem constraint dimension. The function (9) is defined independently for each subsystem and includes the cost for the dynamics (A_i, b_i) and the mapping into the constraint space $(J_i \hat{v}_i = x_i)$, but does not yet concern with constraint satisfaction. For the constraint satisfaction, we define the following function:

$$g(z) = g(z_1, z_2, \cdots, z_{n_s}) = \sum_{j=1}^{n_{in}+n_{cp}} g_j \quad (10)$$

where each $z_i \in \mathbb{R}^{n_{c,i}}$ is actually interpreted as a duplicated variable of x_i for the g function to enforce the constraints. The function g can be better understood in constraint-wise, i.e., summation of g_j where j index denotes each constraint. Each g_j is a function of only the variables corresponding to the j-th constraint i.e.,

$$\{z_{i,j} \mid i \in \mathcal{S}_j\}$$

where $z_{i,j}$ is the segment of z_i corresponds to the j-th constraint and $\mathcal{S}_j$ is the set of subsystem indexes related to the j-th constraint. For the intra-subsystem constraint, the cardinality of S_j (i.e., $|\mathcal{S}_j|$) is 1; if the constraint is inter-subsystem coupling, then $|\mathcal{S}_j| \geq 2$. Based on the functions (9) and (10) defined above, solving (8) can be reformulated as the following optimization problem:

$$\min_{\hat{v},x,z} \sum_{i=1}^{n_s} f_i(\hat{v}_i, x_i) + g(z) \quad (11)$$
$$\text{s.t.} \quad x = z$$

Now applying ADMM iteration on (11), we can obtain the following iteration sequence:

$$\hat{v}_i^{l+1}, x_i^{l+1} = \arg\min_{\hat{v}_i, x_i} \left(f_i + \frac{\beta_i}{2} \|x_i - z_i^l + \frac{1}{\beta_i} u_i^l\|^2 \right) \quad (12)$$

$$z^{l+1} = \arg\min_z \left(g + \sum_i \frac{\beta_i}{2} \|x_i^{l+1} - z_i + \frac{1}{\beta_i} u_i^l\|^2 \right) \quad (13)$$

$$u_i^{l+1} = u_i^l + \beta_i(x_i^{l+1} - z_i^{l+1}) \quad (14)$$

where (12) and (14) are actually computed $\forall i$ in parallel and the weight parameter $\beta_i \in \mathbb{R}$ is utilized for each subsystem for better numerical conditions (see also Sec. III-D.2). Note that the fixed-point of above iteration will satisfy $\forall i$ $J_{c,i}\hat{v}_i = x_i = z_i$, therefore it will exactly satisfy (8) and all constraints (i.e., (3), (4), (5) $\forall j$) without any relaxation. Since Lagrange multiplier update (14) is a trivial step, the main consideration here is how to solve (12) and (13) in an efficient manner.

1) Solving (12)*:* By using an auxiliary variable x_i, it can be seen that the dimension of the problem (12) is expanded to $\dim(\hat{v}_i) + \dim(x_i)$ from the original subsystem dimension $\dim(\hat{v}_i)$. Consider the following KKT conditions of (12):

$$A_i \hat{v}_i^{l+1} = b_i + J_{c,i}^T \gamma$$
$$\beta_i x_i^{l+1} = \beta_i z_i^l - u_i^l - \gamma$$
$$J_{c,i}\hat{v}_i^{l+1} = x_i^{l+1}$$

where γ is the Lagrange multiplier. Here, combining these three equations, we can obtain $\hat{v}_i$ by solving the following linear equation:

$$(A_i + \beta_i J_{c,i}^T J_{c,i})\hat{v}_i^{l+1} = b_i + J_{c,i}^T(\beta_i z_i^l - u_i^l) \quad (15)$$

where the equation is always solvable from the positive definite property of the left-most matrix. By this procedure, the problem size can be brought back to $\dim(\hat{v}_i)$, therefore the concern about increased computation time due to the inclusion of x_i can be obliviated. Note that this trick is not possible if we attempt to directly solve the minimization of non-smooth function f_i. This rather becomes possible as (12) in ADMM procedure uses the quadratic augmented term with scalar weight. In conclusion, the process for solving (12) is simply obtaining a *subsystem size* linear solution for each subsystem in *parallel*.

2) Solving (13)*:* As described earlier, g is the summation of all the g_j defined for each constraint. Accordingly, the problem (13) can be independently decomposed according to all the constraints as

$$\min_{\substack{z_{i,j} \\ i \in \mathcal{S}_j}} \left(g_j + \sum_{i \in \mathcal{S}_j} \frac{\beta_i}{2} \|x_{i,j}^{l+1} - z_{i,j} + \frac{1}{\beta_i} u_{i,j}^l\|^2 \right) \quad (16)$$

therefore can be solved $\forall j$ in parallel. Now consider solving (16) for bilateral case (i.e., $|\mathcal{S}_j| = 2$), which is one of the most frequently appearing in practice. For simplicity, let us assume $\mathcal{S}_j = \{1, 2\}$.

Hard constraint: As z_i is the value already mapped into constraint space, g_j only needs to enforce the constraint on $z_{1,j} + z_{2,j}$. So in the case of hard constraint,

$$g_j = \mathcal{I}(z_{1,j} + z_{2,j} + e_j \geq 0) \quad (17)$$

and (17) can be interpreted as constraint impulse λ_j acting on the linear solution of the quadratic terms in (16) i.e.,

$$\beta_1 z_{1,j} = \underbrace{\beta_1 x_{1,j}^{l+1} + u_{1,j}^l}_{:=y_{1,j}^{l+1}} + \lambda_j$$
$$\beta_2 z_{2,j} = \underbrace{\beta_2 x_{2,j}^{l+1} + u_{2,j}^l}_{:=y_{2,j}^{l+1}} + \lambda_j \quad (18)$$

where we introduce the new variable y for conciseness. We can see from the structure of (16) that the relation (18) is *matrix-free*, and only consists of scalar weights. Thanks to this property, λ_j can be computed in a very simple manner as we combine (18) with the following complementarity condition:

$$0 \leq \lambda_j \perp z_{1,j} + z_{2,j} + e_j \geq 0 \quad (19)$$

the solution for λ_j can be obtained with the simple scalar operation:

$$\lambda_j = \Pi_{\geq 0} \left(-\frac{\beta_1^{-1} y_{1,j}^{l+1} + \beta_2^{-1} y_{2,j}^{l+1} + e_j}{\beta_1^{-1} + \beta_2^{-1}} \right)$$

where $\Pi_{\geq 0}$ denotes the projection on positive set.

The matrix-free relation (18) is the same for other types of constraints (soft, contact), while (19) to be replaced with other relation.

Soft constraint: From the structure of (3),

$$\lambda_j = -k_j(e_j + \alpha_j(z_{1,j} + z_{2,j})) \quad (20)$$

has to be satisfied. Then by substituting (18) to (20), we can obtain the impulse solution as

$$\lambda_j = -\frac{k_j(e_j + \alpha_j(\beta_1^{-1} y_{1,j}^{l+1} + \beta_2^{-1} y_{2,j}^{l+1}))}{(1 + (\beta_1^{-1} + \beta_2^{-1})\alpha_j k_j)}$$

which is also very simple to compute.

Contact constraint: Here the relation between $z_{1,j} + z_{2,j}$ and λ_j must follow (5), therefore

$$0 \leq \lambda_{j,n} \perp z_{1,j,n} + z_{2,j,n} + e_{j,n} \geq 0$$
$$0 \leq \delta_j \perp \mu\lambda_{j,n} - \|\lambda_{j,t}\| \geq 0 \quad (21)$$
$$\delta\lambda_{j,t} + \mu\lambda_{j,n}(z_{1,j,t} + z_{2,j,t}) = 0$$

Despite the complexity of (21), solution can be easily obtained from the simple scalar structure of (18):

$$\lambda_j = \Pi_{\mathcal{C}} \left(-\frac{\beta_1^{-1} y_{1,j}^{l+1} + \beta_2^{-1} y_{2,j}^{l+1} + e_j}{\beta_1^{-1} + \beta_2^{-1}} \right)$$

where $\Pi_{\mathcal{C}}$ denotes the projection on the friction cone. The process can be done through a few algebraic operations, while respecting all contact conditions [35].

Although we explain the process only for the bilateral case, it can be shown straightforwardly that such a simple solution form can be derived for other cases as well.

Algorithm 1 Simulation via Subsystem-Based ADMM

1: Subsystem division for given multibody (Sec. III-A)
2: **while** simulation **do**
3:     $\forall i$ construct A_i, b_i in parallel
4:     $\forall j$ construct e_j, J_j in parallel
5:     $\forall i$ factorize $A_i + \beta_i J_{c,i}^T J_{c,i}$ in parallel
6:     **while** loop **do**
7:         $\forall i$ update $\hat{v}_i^{l+1}$ from (15) in parallel
8:         compute residual θ from (22)
9:         **if** $\theta < \theta_{th}$ or $l = l_{max}$ **then**
10:            **break**
11:         **end if**
12:         $\forall j$ update z_j^{l+1} from (16) in parallel
13:         $\forall i$ update u^{l+1} from (14) in parallel
14:         $l \leftarrow l + 1$
15:     **end while**
16:     update each subsystem state using $\hat{v}_i^{l+1}$
17: **end while**

D. Convergence

It can be easily verified that each f_i and g_j for hard and soft constraints is convex in our formulation (11). For contact conditions, g_j may not be convex, but can be convexified by adopting the relaxed convex model [16]. In such cases, our method can guarantee convergence [13]. Although we have not encountered the convergence issue associated with non-convexity of (5), a more thorough analysis will be left for future work.

1) Residual: Originally, our process (12), (13), (14) is the iteration of $(\hat{v}, x, z, u)$ and both primal and dual residual [13] are required to check the condition to terminate the iteration. Instead, for our framework, we use the variable y in (18) to define the residual as

$$\theta = \sum_{i=1}^{n_s} \|y_i^{l+1} - y_i^l\|^2 \tag{22}$$

where θ is the residual value. This means that the iteration can be reinterpreted in terms of the lower-dimensional variable y, and the process of calculating the residuals can be more concise. The following proposition provides the rationale of the statement:

Proposition 1: $(\hat{v}^{l+1}, x^{l+1}, z^{l+1}, u^{l+1})$ is the fixed-point of the iteration (12), (13) and (14), if and only if $\theta = 0$.

Proof: ($\Rightarrow$) This is trivial. ($\Leftarrow$) As $\theta = 0$ denotes $\forall y_i^{l+1} = y_i^l$, we can find that $z^{l+1} = z^l$ holds as $\forall \lambda_j$ are uniquely determined from y. Then as (14) is equivalent to $u_i^{l+1} = y_i^{l+1} - z_i^{l+1}$, $u^{l+1} = u^l$ also holds. Finally, $\hat{v}$ is determined from z and u (15), so we can conclude that the set value is in fixed-point of the iteration. ∎

2) Choice of β: We find that iteration has stable convergence regardless of β, but the value of β affects the convergence rate. We empirically confirm that the following β setting exhibits good performance:

$$\forall \beta_i = \frac{\text{Tr}(A_i)}{\text{Tr}(J_{c,i}^T J_{c,i})} \tag{23}$$

which suggests a balanced weight between dynamics-related term A_i and constraint-related term $J_{c,i}^T J_{c,i}$. A more in-depth theoretical analysis of the strategy will be discussed in future work.

E. Summary

Our physics simulation framework via subsystem-based ADMM is summarized in Alg. 1. As described earlier, the major part of the procedure is *subsystem-wise* parallel solving of (15) (line 7) and *constraint-wise* parallel solving of (16) (line 12). From these characteristics, the computational complexity of our algorithm is at least linear: $\mathcal{O}(n_s + n_{in} + n_{cp})$. If parallelization is taken into account, it will be lower.

IV. EXAMPLES AND EVALUATIONS

We use an Intel Core i7-8565 CPU 1.80GHz (Quad-Core), OpenGL as a rendering tool, C++ Eigen as a matrix computation library, and C++ OpenMP as a parallelization library in our implementation. Time step is set to 10 ms for all examples. See also our supplemental video.

A. Scenarios

We implement three high-DOF multibody manipulation scenarios. In general, they consist of a combination of high-gain controlled robotic arms and lightweight objects with multi-type constraints, resulting in numerically challenging situations. We employ Franka Emika panda [37] as a robot arm and Husky [38] as a ground vehicle.

1) Granular object stirring: The example is illustrated in Fig. 1a: the robot arm uses an end effector to stir the granular material contained in the box. The granular material consists of a total of 216 spheres with a radius of 1 cm and a weight of 4 g. The total system dimension is 1303, and since each rigid body and robot is treated as a subsystem, there are a total of 217 subsystems.

2) Collaborative cable manipulation: The example is illustrated in Fig. 1b: two mobile manipulator consist of a ground vehicle and a robot arm are transporting and winding a flexible cable. Cable is modeled by 640 rigid bodies and soft constraint from Cosserat model, with length 1.2 m, diameter 8 mm, Young modulus 0.1 MPa, and Poisson ratio 0.49. Each mobile manipulator is modeled as 10-DOF system while its movement constrained by non-holonomic constraint (no-slip). Total system dimension is 3840, and we treat each mobile manipulator and 4 cable segments as a subsystem, making a total of 162 subsystems.

3) FEM beam insertion: The example is illustrated in Fig. 1c: the robot arm inserts the deformable beam modeled with co-rotational FEM through narrow gap. The size of beam is $0.05 \times 0.05 \times 0.5$ m, with a Young modulus 10 MPa and a Poisson ratio 0.45. The FEM model consists of 1591 nodes and 6347 tetrahedral elements, therefore total dimension is 4780. We divide the model into 20 subsystems so the entire system consists of a total of 21 subsystems including the manipulator.

Solver		PGS			PJ			FADMM			NNewton			**SubADMM**		
Iteration		30	60	90	30	60	90	30	60	90	3	6	9	30	60	90
Stir	AT	14.50	23.91	32.27	3.235	5.688	9.425	28.41	41.40	56.95	24.35	46.99	75.44	3.489	5.940	8.705
	AA	4.427	4.928	5.248	3.033	3.349	3.562	4.107	4.644	5.009	3.565	4.429	5.324	4.069	4.579	5.023
Cable	AT	48.08	59.30	72.96	-	-	-	16.74	23.52	32.96	43.35	87.98	132.7	2.288	4.285	6.402
	AA	4.141	4.860	5.404	-	-	-	4.189	4.634	4.910	3.270	4.454	5.278	4.344	4.984	5.222
Beam	AT	231.4	241.2	251.2	-	-	-	50.33	92.11	130.3	188.5	360.2	525.7	13.41	24.67	35.50
	AA	3.895	4.194	4.255	-	-	-	4.220	4.478	4.756	2.445	3.494	4.945	4.326	4.743	4.925

TABLE I: Evaluation results for various solvers. AT: average compuatation time (ms), AA: average accuracy (constraint error value converted using $-\log(\cdot)$ before averaged, therefore bigger is better). Unmarked values (-) means that the simulation fails to run successfully (e.g., significant penetration).

B. Baselines

We implement the following algorithms for performance comparison, with our method being denoted as SubADMM.

1) Projected Gauss-Seidel (PGS): PGS is a representative algorithm in robotics and graphics fields [16]–[19] and software [8]–[10]. We implement an algorithm with conjugate gradient-based acceleration to improve its performance.

2) Projected Jacobi (PJ): PJ is similar to PGS, but they do not solve constraints sequentially, but rather solve and update them in parallel at once.

3) Full ADMM (FADMM): State-of-the-art implementations of ADMM algorithms [39] can be used to solve physics simulation, which is specified in [29]. The main difference with our algorithm is that they require solving of the full-system size matrix for each iteration.

4) Nonsmooth Newton (NNewton): We also implement a recently proposed algorithm that transform the constraints into non-smooth function and solve it using Newton iteration. We refer [23], [40] for details.

C. Performance Index

We apply the same number of iteration $(30, 60, 90)$ for all algorithms except NNewton and measure the average solver computation time and constraint error norm from the simulation results. In the case of NNewton, considering its second-order nature (cost per iteration is high but uses fewer iterations), the number of iterations is reduced by $1/10$ (i.e., $3, 6, 9$). Constraint error for contact is calculated using Fischer-Burmeister function [23].

D. Results

Evaluation results are summarized in Table I. For graunlar object strring scenario, PJ and SubADMM shows the fastest computation speed, and this is due to their structure suitable for parallelization. However, constraint error of PJ is signifcantly higher than SubADMM. This reflects the unstable and slow convergence of the Jacobi-style iteration. On the other hand, SubADMM shows comparable error with other methods and shows its validity in terms of accuracy. In the case of the cable and beam scenario, the computation performance of PGS and PJ becomes lower as the Delassus operator assembly is more complicated. As such, FADMM outperforms them, yet SubADMM is still the fastest. This is due to our special structure, which, as mentioned earlier, only requires parallelized resolution of the subsystem matrices
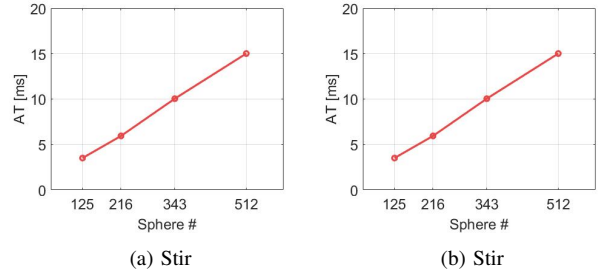

(a) Stir   (b) Stir

Fig. 2: Scalablity test results of SubADMM.

without dealing with large-sized matrices. In the similar vein, SubADMM also has an efficiency advantage over NNewton. Algorithms other than PJ showed valid accuracies, while PJ failed to generate an adequate simulation results. In summary, the results demonstrate all of the methodologically described advantages of SubADMM: 1) it avoids burdens on both many constraints and large-sized matrices, and 2) it does not use certain approximations on the model and has a good convergence property.

E. Scalability

To precisely evaluate the scalability of our method, we measure the computation time (iteration: 60) by increasing the number of spheres in the stir scenario and the number of segments in the cable scenario. Fig. 2 shows linear complexity of SubADMM (R-squared value: 0.9993, 0.9998).

V. DISCUSSIONS AND CONCLUSIONS

In this paper, we present a new physics simulation framework based on subsystem-based ADMM. Our approaches combines a novel subsystem-based formulation (7) and operator splitting (9) and (10), thereby achieve parallelizable and modular architecture for general multibody dynamics. Several examples are implemented and evaluations show the advantages of our framework against state-of-the-art algorithms. We believe that a generic implementation (similar to the open source form) will make a good contribution to the robotics community. We also believe that our work can be extended to the area of optimal control by exploiting the coupled structure of the large-size optimization problem (e.g., time correlation). Finally, similar to typical ADMM, the convergence property of our algorithm is stable but still linear. Therefore, combination with second-order acceleration schemes will be a promising research direction.

REFERENCES

[1] F. Agostinelli, S. McAleer, A. Shmakov, and P. Baldi. Solving the rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363, 2019.

[2] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020.

[3] Z. Ding, N. F. Lepora, and E. Johns. Sim-to-real transfer for optical tactile sensing. In *IEEE International Conference on Robotics and Automation*, pages 1639–1645, 2020.

[4] M. A. Lee, Y. Zhu, P. Zachares, M. Tan, K. Srinivasan, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg. Making sense of vision and touch: Learning multimodal representations for contact-rich tasks. *IEEE Transactions on Robotics*, 36(3):582–596, 2020.

[5] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *IEEE International Conference on Robotics and Automation*, pages 2536–2542, 2020.

[6] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier. Differentiable simulation for physical system identification. *IEEE Robotics and Automation Letters*, 6(2):3413–3420, 2021.

[7] Raisim physics engine. https://raisim.com/.

[8] Bullet physics engine. https://pybullet.org/.

[9] Mujoco physics engine. http://www.mujoco.org/.

[10] Flex physics engine. https://developer.nvidia.com/flex.

[11] Sofa physics engine. https://www.sofa-framework.org/.

[12] H. Choi, C. Crump, C. Duriez, A. Elmquist, G. Hager, D. Han, F. Hearl, J. Hodgins, A. Jain, F. Leve, C. Li, F. Meier, D. Negrut, L. Righetti, A. Rodriguez, Tan. J., and Trinkle. J. On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences*, 118(1):e1907856118, 2021.

[13] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Foundations and Trends in Machine Learning, 2011.

[14] J. E. Lloyd. Fast implementation of lemke's algorithm for rigid body contact simulation. In *IEEE International Conference on Robotics and Automation*, 2005.

[15] M. Anitescu and F. A. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14:231–247, 1997.

[16] E. Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. In *IEEE International Conference on Robotics and Automation*, pages 6054–6061, 2014.

[17] M. Macklin, M. Müller, N. Chentanez, and T. Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics*, 33(4), 2014.

[18] M. Macklin, M. Müller, and N. Chentanez. Xpbd: Position-based simulation of compliant constrained dynamics. In *International Conference on Motion in Games*, pages 49–54, 2016.

[19] P. C. Horak and J. C. Trinkle. On the similarities and differences among contact models in robot simulation. *IEEE Robotics and Automation Letters*, 4(2):493–499, 2019.

[20] M. A. Otaduy, R. Tamstorf, D. Steinemann, and M. Gross. Implicit contact handling for deformable objects. *Computer Graphics Forum*, 28(2):559–568, 2009.

[21] G. Daviet. Simple and scalable frictional contacts for thin nodal objects. *ACM Transactions on Graphics*, 39(4), 2020.

[22] J. Carpentier, R. Budhiraja, and N. Mansard. Proximal and sparse resolution of constrained dynamic equations. In *Robotics: Science and Systems*, 2021.

[23] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and V. Makoviychuk. Non-smooth newton methods for deformable multibody dynamics. *ACM Transactions on Graphics*, 38(5):1–20, 2019.

[24] M. Li, Z. Ferguson, T. Schneider, T. Langlois, D. Zorin, D. Panozzo, C. Jiang, and D. M. Kaufman. Incremental potential contact: Intersection-and inversion-free, large-deformation dynamics. *ACM Transactions on Graphics*, 39(4), 2020.

[25] A. M Castro, F. N Permenter, and X. Han. An unconstrained convex formulation of compliant contact. *IEEE Transactions on Robotics*, 2022.

[26] A. Peiret, S. Andrews, J. Kövecses, P. G. Kry, and M. Teichmann. Schur complement-based substructuring of stiff multibody systems with contact. *ACM Transactions on Graphics*, 38(5):1–17, 2019.

[27] J. Lee, M. Lee, J. Yoon, and D. J. Lee. A parallelized iterative algorithm for real-time simulation of long flexible cable manipulation. In *IEEE International Conference on Robotics and Automation*, pages 12040–12046, 2021.

[28] M. Lee, J. Lee, J. Yoon, and D. Lee. Real-time physically-accurate simulation of robotic snap connection process. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5173–5180, 2021.

[29] A. Tasora, D. Mangoni, S. Benatti, and R. Garziera. Solving variational inequalities and cone complementarity problems in nonsmooth dynamics using the alternating direction method of multipliers. *International Journal for Numerical Method in Engineering*, 122(16):4093–4113, 2021.

[30] M. Overby, G. E. Brown, J. Li, and R. Narain. Admm $\supseteq$ projective dynamics: Fast simulation of hyperelastic models with dynamic constraints. *IEEE Transactions on Visualization and Computer Graphics*, 23(10):2222–2234, 2017.

[31] M. Kim, Y. Lee, Y. Lee, and D. J. Lee. Haptic rendering and interactive simulation using passive midpoint integration. *International Journal of Robotics Research*, 36(12):1341–1362, 2017.

[32] M. Verschoor and A. C. Jalba. Efficient and accurate collision response for elastically deformable models. *ACM Transactions on Graphics*, 38(2), 2019.

[33] S. Andrews, M. Teichmann, and P. G. Kry. Geometric stiffness for real-time constrained multibody dynamics. *Computer Graphics Forum*, 36(2):235–246, 2017.

[34] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1(1):1–16, 1972.

[35] J. Lee, M. Lee, and D. Lee. Large-dimensional multibody dynamics simulation using contact nodalization and diagonalization. *IEEE Transactions on Robotics*, 2022.

[36] J. Wang, F. Yu, X. Chen, and L. Zhao. Admm for efficient deep learning with global convergence. In *ACM International Conference on Knowledge Discovery & Data Mining*, pages 111–119, 2019.

[37] Franka emika. https://www.franka.de/.

[38] Clearpath robotics. https://clearpathrobotics.com/.

[39] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. Osqp: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12:637–672, 2020.

[40] S. Andrews, K. Erleben, and Z. Ferguson. Contact and friction simulation for computer graphics. In *ACM SIGGRAPH 2022 Courses*, 2022.