# Schur Complement-based Substructuring of Stiff Multibody Systems with Contact

ALBERT PEIRET*, McGill University, Canada
SHELDON ANDREWS*, École de technologie supérieure, Canada
JÓZSEF KÖVECSES, McGill University, Canada
PAUL G. KRY, McGill University, Canada
MAREK TEICHMANN, CM Labs Simulations, Canada

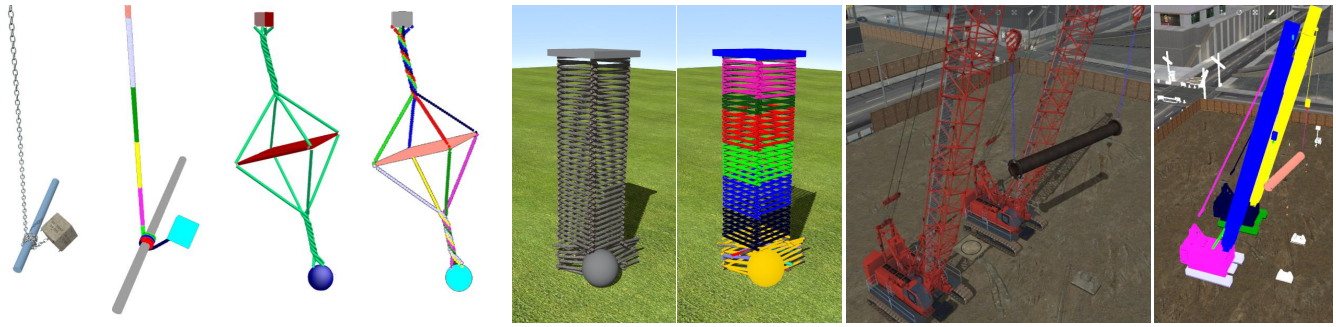


Fig. 1. Challenging scenarios simulated with our method, from left-to-right: a stiff chain with attached mass (chain wrap); an intertwined system of cables (cable spinner); a large stack of logs with heavy plate atop (log tower); a dual crane training simulation (tandem cranes). Each example is modeled as a multibody system with hundreds or thousands of constraints, contacts, and mass ratios up to 40, 000 : 1. The corresponding decomposition of the simulation is shown, where bodies of the same color belong to the same subsystem. Subsystems are coupled by our Schur complement solver for efficient simulation.

Substructuring permits parallelization of physics simulation on multi-core CPUs. We present a new substructuring approach for solving stiff multibody systems containing both bilateral and unilateral constraints. Our approach is based on non-overlapping domain decomposition with the Schur complement method, which we extend to systems involving contact formulated as a mixed bounds linear complementarity problem. At each time step, we alternate between solving the subsystem and interface constraint impulses, which leads to the identification of the active constraints. By using the active constraints to compute the effective mass of subsystems within the interface solve, we obtain an exact solution. We demonstrate that our simulations have preferable behavior compared to standard iterative solvers and substructuring techniques based on the exchange of forces at interface bodies. We observe considerable speedups for structured simulations where a user-defined partitioning can be applied, and moderate speedups for unstructured simulations, such as piles of bodies. In the latter case, we propose an automatic partitioning strategy based on the degree of bodies in the constraint graph. Because our method makes use of direct solvers, we are able to achieve interactive and real-time frame rates for a number of challenging scenarios involving large mass ratios, redundant constraints, and ill-conditioned systems.

CCS Concepts: • **Computing methodologies** → **Physical simulation**; *Concurrent algorithms*;

Additional Key Words and Phrases: Schur complement, unilateral constraints, LCP, multibody dynamics

**ACM Reference Format:**
Albert Peiret, Sheldon Andrews, József Kövecses, Paul G. Kry, and Marek Teichmann. 2019. Schur Complement-based Substructuring of Stiff Multibody Systems with Contact. *ACM Trans. Graph.* 38, 5, Article 150 (October 2019), 17 pages. https://doi.org/10.1145/3355621

## 1 INTRODUCTION

The Schur complement method permits the solution of a large linear system through a number of smaller system solves. While early uses of the method were important for systems that did not fit in memory, now that multi-core processors are prevalent in all consumer hardware, the method has become more interesting for fast parallel computation. Furthermore, since Schur complement domain decomposition works with direct solvers, it can deal with *stiff* systems more gracefully than iterative methods. In contrast, slow convergence can be a common problem with iterative methods

*Both authors contributed equally to this work.

Authors' addresses: Albert Peiret, McGill University, Montreal, Québec, Canada, alpeiret@cim.mcgill.ca; Sheldon Andrews, École de technologie supérieure, Montreal, Canada, sheldon.andrews@etsmtl.ca; József Kövecses, McGill University, Montreal, Canada, jozsef.kovecses@mcgill.ca; Paul G. Kry, McGill University, Montreal, Canada, kry@cs.mcgill.ca; Marek Teichmann, CM Labs Simulations, Montreal, Canada, marek@cm-labs.com.


0730-0301/2019/10-ART150 $15.00
https://doi.org/10.1145/3355621

when dealing with the stiffness that frequently arises in physics-based simulations. Specifically, we use the term *stiff* to describe poor conditioning that can easily arise from a number of situations. In the case of multibody systems, examples include large mass ratios, redundant constraints from contacts or loops, and the small compliances often used to regularize constraints. While iterative algorithms have been very popular in computer graphics compared to many other fields, and while preconditioning can help in dealing with stiff systems, there is nevertheless a number of examples where previous work has recognized the benefits and applied direct solvers, such as in the solution of stiff rods [Deul et al. 2018] and for adaptive discretization in cloth simulation [Narain et al. 2012].

Our work addresses the use of direct solvers for interactive simulation of stiff multibody systems with contact. Typical systems include both bilateral and unilateral constraints, as well as friction at the contact interfaces. Bilateral constraints typically model joints and actuators, while unilateral constraints are necessary to represent direct contact interactions between rigid bodies. Bilateral constraints can also have bounds that introduce further inequalities into the formulation. A common modeling approach for these problems leads to a dynamics formulation where the core mathematical model takes the form of a *mixed linear complementarity problem* (MLCP). Numerous rigid body dynamics platforms employ MLCP formulations to model physical problems. Unfortunately, the solution of an MLCP is more involved than solving a linear system and does not permit a direct application of Schur complement domain decomposition to speed up computation.

The novelty of our work is the development of an algorithm to speed up the solution of the MLCP representing the rigid body dynamics problem. The main idea is to split the overall system into smaller subsystems and solve each one separately. Individual subsystems are then coupled to the others using reduced-order models that rely on the concept of effective mass. This is based on the Schur complement domain decomposition approach, with important adjustments to identify and correctly handle unilateral constraints. The effective mass representation includes the effect of both bilateral and unilateral constraints within a subsystem. This is a novel use of the concept, which requires proper algorithmic considerations. Our approach provides an efficient parallel computation of the full system, without the convergence issues of iterative methods.

In this work, we target simulation for interactive and real-time applications. Interactive simulation is particularly important for training simulators, virtual reality, and video games, which are application areas that can benefit from an improved performance without sacrificing solution accuracy. Figure 1 shows a collection of challenging stiff systems that we are able to simulate efficiently with our method. In comparison to standard formulations and solvers, such as those used in industry, we can observe more than 20 times speedup while maintaining accuracy equivalent to a full direct solve.

## 2 RELATED WORK

Modeling contact dynamics in multi-rigid-body systems can generally lead to complementarity problems [Acary and Brogliato 2008; Baraff 1994; Moreau 1988]. In such systems, the rigid bodies are connected to each other through joints represented by bilateral constraints and direct contacts given by unilateral non-penetration constraints, which generally leads to an MLCP model [Stewart and Trinkle 1996]. Such a model can also be formulated as a convex, quadratic optimization problem [Moreau 1966].

The consideration of Coulomb friction at the contacts complicates the mathematical formulation and turns it into a nonlinear complementarity problem [Moreau 1988; Pfeiffer et al. 2006]. This is specifically due to the physical nature of friction whereby the friction force between a pair of bodies in contact depends on non-interpretation forces, and it is limited by the friction cone constraint [Pfeiffer et al. 2006]. However, with a proper approximation of the friction cone [Anitescu and Potra 1997; Glocker 2001; Stewart and Trinkle 1996], the core algorithmic problem leads back to an MLCP. The solution of such MLCP contact formulations has significant applications in computer graphics, various engineering fields, animation, and interactive simulation of virtual environments. In this regard, Bender et al. [2014] provide an excellent survey of the models, numerical methods, and algorithms for interactive simulation of multibody systems in computer graphics.

While fast linear time dynamics is possible for multibody systems with tree-structured constraints [Baraff 1996], parallel computing can offer possibilities to further accelerate the solutions in multi-core CPU environments. For this, the multibody system has to be partitioned into subsystems that interface each other. This problem has received significant attention for various problems where the rigid bodies interact only through joints represented by bilateral constraints with no bounds. For such cases, the problem is a system of linear equations rather than the more difficult general MLCP problem. A divide and conquer algorithm was proposed for such systems by Featherstone [1999]. In this approach, the subsystems are termed articulated bodies and the interfaces are called handles. The algorithm essentially imposes a sequential application of constraints and makes a parallel implementation possible. Several enhanced variants of this algorithm have also been reported, which target both CPU and GPU implementations [Critchley et al. 2009; Laflin et al. 2014]. However, the main disadvantages of the algorithm are that it requires the explicit availability of inverse mass matrices, it only works for bilateral joints with no bounds, and even for such cases it cannot handle Coulomb friction.

The Schur complement is a common tool in the decomposition of systems. Substructuring for vehicle dynamics analysis is one example addressing the special topology of vehicles [Kang et al. 2015; Kim 2002]. Here, the chassis serves as the main subsystem, with other components (suspension etc.) branching out from, and only interacting with, the main subsystem. These formulations have been developed only for bilateral constraints without bounds and cannot handle friction. They also strongly depend on a specific topology. However, there is a useful feature that implicitly arises in them: the derivation of the interface forces between the core subsystem and the branches leads to the expression of effective mass matrices associated with the interface variables. These are obtained in a form that is essentially the Schur complement of the original coupled mass matrix. We use this effective mass expression, or rather the inverse effective mass, in our algorithm. These kinds of expressions have also proved useful in fluid simulation, where the Poisson pressure solve can be seen as involving a large number

of bilateral constraints [Chu et al. 2017; Liu et al. 2016]. Here, the partitioning of subsystems benefits from the regular grid structure of fluid simulations, whereas partitioning in our work is based on the constraint graph whose topology changes at each time step.

When unilateral constraints such as contact are present, a general form of the MLCP problem is needed and the subsystem analysis becomes much more challenging. Tonge et al. [2012] propose an iterative Jacobi-based method that uses the concept of mass splitting in combination with a graph coloring algorithm to simulate large piles of bodies. Their simulations are partitioned into blocks of contacts involving sub-bodies, and these blocks are solved separately and then later combined by considering the fixed joints between the sub-bodies. Furthermore, although graph coloring approaches such as parallel PGS facilitate simulating large scale problems on the GPU, these methods are slow to converge for stiff and ill-conditioned systems and this severely mitigates performance gains from parallelization. In comparison, our approach produces results for simulations involving both bilateral and unilateral constraints, which are modeled as connected subsystems that are solved in parallel. The resulting constraint error and interpenetration is much lower thanks to the use of a direct solver.

Tomcin et al. [2014] solve complex multi-body problems in the presence of loops and contacts. Such highly over-constrained problems are poorly conditioned or have redundant constraints, and are in many respects similar to the systems addressed by our work. Tomcin et al. use a mixed iterative and direct approach which leads to very low constraint violation error through the use of carefully regularized linear solves to close loops in what could be called a pre-stabilization approach. Parallel solution of their sparse direct systems are possible with nested dissection. We distinguish our work in that our focus is on the MLCP error, though we could also use our formulation for a parallel position level post-step stabilization solve [Cline and Pai 2003].

Early work by Baraff and Witkin [1997] consider partitioned dynamics for interleaved simulations, with conjugate gradient iterations used to compute constraint forces between partitions. Lacoursière [2007] partition multibody systems at points where they are weakly coupled, and resolves the interface with block projected Gauss-Seidel iterations. The method needs warm-starting to achieve faster convergence. For $N$ subsystems, it solves $N + 1$ MLCPs every iteration, but the system matrices are taken directly as blocks of the main problem matrix. Although the constraint forces can be solved by a direct method, this formulation only takes into account the dynamics of the bodies directly connected at the interface, thus ignoring the internal dynamics of each subsystem while solving for the interface constraints. Because the subsystems are only weakly coupled, interface forces are resolved in an iterative manner and many iterations are required for stiff problems. Otherwise, the interface constraints appear *soft* and produce large constraint violations. We refer to this approach as *force based interface coupling* (FBIC) and make specific comparisons to it throughout this paper.

An important aspect of our work is that we form the inverse effective mass via the Schur complement based on the index set of the subsystems constraints. Our approach accounts for coupling of the entire subsystem, incorporating effects due to contact detachment and frictional sliding, and uses this to compute the effective mass and force/impulse terms. While we typically need only a few coupling iterations to identify the active constraints, we can likewise obtain the solution in one iteration if the active constraints do not change from the previous time step. We compute the effective mass using a modified Schur complement that only considers the active constraints in each subsystem; this is an important contribution of our work. Therefore, we not do not iterate over the values of constraint forces, but rather the index set of the constraints, using a pivoting approach.

Domain decomposition in elastic problems can exploit local model reduction, providing opportunities for fast solutions in addition to parallel computing [Barbič and Zhao 2011; Kim and James 2011]. Parker and O'Brien [2009] describe partitioning and parallelization in elastic systems with contact, where the system is divided into domains to parallelize the solve (a common technique in modern multibody engines) and the matrix multiplication operation in conjugate gradient solves are also parallelized for very large systems. Blocked Gauss-Seidel approaches are often preferred for large scale parallelization on the GPU, for instance, using a graph coloring algorithm to partition the system [Fratarcangeli and Pellacini 2015; Fratarcangeli et al. 2016]. Related to this is the staggered projection approach, which places contact normal and tangential forces into separate blocks [Kaufman et al. 2008]. Solvers for elastic systems can struggle to handle stiff multibody systems with LCP contact constraints. Approximate solutions to stiff constraint problems may need an additional solve to stabilize the constraints [Cline and Pai 2003]. While stabilization is generally unavoidable, stiff problems and those with large mass ratios are scenarios where it is still desirable to use methods that can produce accurate solutions.

Various techniques can be used to speed up convergence and improve solutions to multibody simulation problems. Warm starting and shock propagation can help projected Gauss-Seidel and impulse based methods [Erleben 2007; Guendelman et al. 2003]. Long range constraints can likewise be very beneficial in special cases, such as chains [Müller et al. 2017]. In contrast, our approach does not require any modifications to the model, nor do we make assumptions that would otherwise limit the applicability of our solution.

In the context of off-line simulations, there are other interesting methods, such as accelerated projected gradient descent with applications in the simulation of granular materials [Mazhar et al. 2015]. Visseq et al. [2012] also propose a domain decomposition algorithm for granular materials based on an approach they refer to as *gluing*. Contact is present within the domains, which are made of grains, and the grains between domains define the interface. However, the interface problem consists of the gluing constraints, which are essentially equality constraints that only take into account the dynamics of the pairs of glued grains, and it does not consider the internal dynamics of the domains.

Finally, we note that domain decomposition and substructuring are related to graph partitioning algorithms, such as METIS [Karypis and Kumar 1999] and SCOTCH [Pellegrini and Roman 1996], which produce reduced fill-in orderings of sparse matrices. However, these algorithms are not well-suited for the real-time and interactive applications we target in this paper. A comparison of matrix reordering methods for multibody simulation by Torres-Moreno et al. [2013] found that simple reordering strategies, specifically the column

approximate minimum degree method (COLAMD), outperformed graph partitioning algorithms for medium systems (i.e., 50 to 2000 degrees of freedom). We also propose an automatic partitioning algorithm (see Section 6) that requires very little computational overhead and operates directly on the constraint graph of similarly-sized systems.

## 3 CONSTRAINED MULTIBODY SYSTEMS

We consider multibody systems composed of rigid bodies that interact with each other through constraints that restrict their motion bilaterally or unilaterally. Direct contact between the bodies is commonly represented by such unilateral constraints, allowing the dynamics to be formulated as a complementarity problem. Here, we first introduce the general formulation for systems with bilateral constrains, and then add contact to formulate an MLCP.

The motion of the system for a certain configuration is defined by the velocity of $n$ bodies, which can be arranged in an array of $6n$ generalized velocities $\mathbf{v}$. For each body, the velocity of its center of mass with respect to an absolute frame can be represented by three linear components and three angular components. The interactions between bodies are described through constraints. The $m$ constraint velocities representing these modes of motion can be written as a linear combination of the generalized velocities,

$$\mathbf{J}\mathbf{v} = \mathbf{w}_0 \tag{1}$$

where $\mathbf{J}$ is the $m \times 6n$ constraint Jacobian matrix, and $\mathbf{w}_0$ contains the $m$ constraint velocities. Eq. (1) is obtained through a linearization of the constraint functions at a position level and can be used, for example, to constrain the relative position and orientation between bodies.

The dynamic equations that govern the motion of the system relate the time derivatives of the generalized velocities to the forces acting on the system. By introducing a finite-difference approximation, the change in the generalized momentum from one time step to the next can be related to the generalized impulse as

$$\mathbf{M}(\mathbf{v}^+ - \mathbf{v}) = h\mathbf{f} + \mathbf{J}^{\mathrm{T}}\boldsymbol{\lambda}^+, \tag{2}$$

where $\mathbf{v}$ and $\mathbf{v}^+$ are the velocity at the beginning and at the end of the time step, respectively, and $h$ is the time step size [Acary and Brogliato 2008]. Here, $\mathbf{M}$ is the $6n \times 6n$ symmetric positive-definite mass matrix, which has a block-diagonal structure with a bandwidth of 3 due to the inertia matrices associated with the angular velocity components, and $\mathbf{f}$ is the $6n$-dimensional array of generalized applied forces containing the Coriolis and centrifugal terms and external forces, such as gravity. The $m$ unknown constraint impulses are arranged in $\boldsymbol{\lambda}^+$ and are related to the constraint forces and moments through the time step.

The constraints in Eq. (1) may not be independent (i.e., the Jacobian matrix $\mathbf{J}$ is rank deficient), leading to a constraint redundancy problem. In such a case, the impulses $\boldsymbol{\lambda}^+$ cannot be uniquely determined. To cope with this, it is common to relax the constraints and regularize the force using a constitutive relation such as

$$\mathbf{J}\mathbf{v}^+ + \mathbf{C}\boldsymbol{\lambda}^+ + h^{-1}\boldsymbol{\phi} = \mathbf{w}_0\,, \tag{3}$$

where $\mathbf{C}$ is the $m \times m$ symmetric positive-definite matrix associated with the compliance of the regularized constraints, which is usually diagonal, and $\boldsymbol{\phi}$ accounts for the constraint violation at the position level in the current time step.

By combining Eq. (2) and Eq. (3), the augmented form of the *impulse-momentum* level formulation can be written as

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}^{\mathrm{T}} \\ \mathbf{J} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{v}^+ \\ \boldsymbol{\lambda}^+ \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ -\mathbf{d} \end{bmatrix}, \tag{4}$$

where $\mathbf{p} = \mathbf{M}\mathbf{v} + h\mathbf{f}$ and $\mathbf{d} = h^{-1}\boldsymbol{\phi} - \mathbf{w}_0$ are used to simplify the notation. In these kinds of formulations, the mass matrix $\mathbf{M}$ can be modified in order to account for the discretization of gyroscopic and constraint forces. For instance, by using the geometric stiffness tensor, which has been shown to improve stability when simulating stiff mechanical systems [Andrews et al. 2017; Tournier et al. 2015].

Furthermore, it is possible to formulate the problem using just the constraint impulses $\boldsymbol{\lambda}^+$ by taking the Schur complement of the mass matrix $\mathbf{M}$ in Eq. (4)

$$\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathrm{T}} + \mathbf{C}\right)\boldsymbol{\lambda}^+ = -\mathbf{d} - \mathbf{J}\mathbf{M}^{-1}\mathbf{p}\,, \tag{5}$$

where the inverse of the mass matrix $\mathbf{M}^{-1}$ can be computed in closed form because of its block-diagonal structure. This formulation gives a linear system in which fewer variables need to be solved at each step, since typically $m < 6n$. Furthermore, the velocities can be calculated at the end of the time step after having computed the constraint impulses.

It is worth noting that the matrix in Eq. (5) can be ill-conditioned for stiff systems that have low compliance $\mathbf{C}$. Specifically, this occurs when the system has large mass ratios, when the constraints are redundant, or a combination of both. In such cases, the condition number of the inverse effective mass $\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathrm{T}}$ can be quite large.

The position of all the bodies in the system is described by a set of generalized coordinates $\mathbf{q}$, and the dimensionality depends on the representation. Here, we use a 3D vector and a quaternion, respectively, to represent the position and orientation of each body with respect to the absolute frame, which amounts to $7n$ generalized coordinates. The position update at the end of the time step can then be computed using the generalized velocities $\mathbf{v}^+$ as

$$\mathbf{q}^+ = \mathbf{q} + h\mathbf{T}\mathbf{v}^+, \tag{6}$$

where $\mathbf{q}$ and $\mathbf{q}^+$ are the coordinates at the beginning and the end of the time step, respectively, and $\mathbf{T}$ is the $7n \times 6n$ transformation matrix, which is computed using the known coordinates $\mathbf{q}$. The quaternions of each body are normalized after the update to ensure numerically consistent rotations.

Although constraint violations can occur after the position update in Eq. (6), the Baumgarte stabilization used in Eq. (4) will correct it in subsequent time steps.

### 3.1 The Mixed Linear Complementarity Problem

We are primarily interested in simulating multibody systems containing a mixture of unilateral and bilateral constraints. Figure 2 illustrates the different types of constraint forces that arise in such simulations. The impulses $\boldsymbol{\lambda}^+$ can be limited to certain values depending on the kind of constraint, e.g., unilateral constraints have non-negative impulses. For a given time step, the inequalities can be written

$$\mathbf{l} \leqslant \boldsymbol{\lambda}^+ \leqslant \mathbf{u}\,, \tag{7}$$

where $\mathbf{l}$ and $\mathbf{u}$ contain the lower and upper bounds for the constraint impulses, respectively, which are set to $-/+$ infinity for bilateral constraints. The formulation in Eq. (5) can then be written as an MLCP, such that

$$\left.\begin{aligned} \underbrace{\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathrm{T}}+\mathbf{C}\right)}_{\mathbf{A}}\boldsymbol{\lambda}^{+}+\underbrace{\mathbf{d}+\mathbf{J}\mathbf{M}^{-1}\mathbf{p}}_{\mathbf{b}}=\mathbf{w}^{+} \\ \mathbf{l}\leqslant\boldsymbol{\lambda}^{+}\leqslant\mathbf{u} \end{aligned}\right\} \tag{8}$$

where $\mathbf{w}^{+}=\mathbf{w}_{\mathrm{l}}^{+}-\mathbf{w}_{\mathrm{u}}^{+}$ contain the *slack velocities* associated with the bounds of the constraint impulses. These are related to the constraint velocities, and need to satisfy the complementarity conditions

$$\begin{aligned} 0\leqslant\mathbf{u}-\boldsymbol{\lambda}^{+}\perp\mathbf{w}_{\mathrm{u}}^{+}\geqslant 0 \\ 0\leqslant\boldsymbol{\lambda}^{+}-\mathbf{l}\perp\mathbf{w}_{\mathrm{l}}^{+}\geqslant 0 \end{aligned} \tag{9}$$

where $\perp$ denotes component-wise complementarity. To simplify the notation, we only use the inequalities in Eq. (7) to denote an MLCP with bounded impulses $\boldsymbol{\lambda}^{+}$ and slack velocities $\mathbf{w}^{+}$. In this form, it is also known as a *bounded linear complementarity problem* (BLCP) with box constraints.

Contact is often represented by unilateral constraints, in which the relative velocity of each contact point is decomposed in one normal and two tangential components, so that the constraint impulses in $\boldsymbol{\lambda}^{+}$ represent the normal and friction force components. To allow contact detachment, the normal impulse must be non-negative, that is,

$$\lambda_{\mathrm{n}}^{+}\geqslant 0. \tag{10}$$

Geometric primitives, such as spheres, cylinders, and boxes, are used to represent the geometry of the bodies, and a collision detection algorithm determines the contact points and normal directions.

On the other hand, Coulomb friction defines a constraint on the tangential velocity with a limit for the resultant friction force. The friction constraint can be enforced component-wise by defining upper and lower bounds for the two friction force components as

$$-\mu\lambda_{\mathrm{n}}\leqslant\lambda_{\mathrm{t}_j}^{+}\leqslant+\mu\lambda_{\mathrm{n}}, \tag{11}$$

where $j=1,2$ denotes the two directions in the tangent plane, and $\lambda_{\mathrm{n}}$ is taken from the previous time step. This is also known as the *box friction* model, which likewise replaces the original nonlinear inequalities of the Coulomb model by a linear approximation of the friction cone [CM Labs Simulations 2017].

For a contact point, the slack velocity $\mathbf{w}^{+}$ in Eq. (8) is related to the normal velocity component, and it must be zero while the bodies are in contact. However, when the contact detaches, the impulse must be zero and the slack velocity becomes positive to satisfy the complementarity conditions. For the case of friction, when the impulse is within the bounds, the slack velocity must be zero, and so the contact is sticking. Otherwise, when it reaches a bound, the contact is sliding, and so the slack velocity is non-zero.

In general, the complementarity conditions in Eq. (9) can be defined as

$$\begin{cases} \mathbf{w}^{+}\geqslant 0 & \text{if} \quad \boldsymbol{\lambda}^{+}=\mathbf{l} \\ \mathbf{w}^{+}=0 & \text{if} \quad \boldsymbol{\lambda}^{+}\in(\mathbf{l},\mathbf{u}) \\ \mathbf{w}^{+}\leqslant 0 & \text{if} \quad \boldsymbol{\lambda}^{+}=\mathbf{u} \end{cases} \tag{12}$$

where the impulses within bounds $\boldsymbol{\lambda}^{+}\in(\mathbf{l},\mathbf{u})$ are known as *free*, and those at the bounds, $\boldsymbol{\lambda}^{+}=\mathbf{l}$ or $\boldsymbol{\lambda}^{+}=\mathbf{u}$, are known as *tight*. The labels tight and free on all constraints together determines the index set of the system.

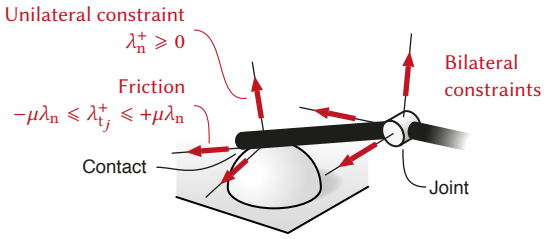



Fig. 2. Illustration of the constraint forces in a multibody system with contact, and definition of the normal and tangential impulses.

In this section, we have summarized a common approach for simulating a constrained multibody system modeled as an MLCP. The following sections explain our primary contribution, which is how domain decomposition can be applied to such systems by partitioning them into smaller subsystems and solving these in parallel.

## 4 SUBSTRUCTURING OF MULTIBODY SYSTEMS

The central idea behind substructuring of constrained multibody dynamics is to define non-overlapping subsystems of bodies that only interact through a set of interface constraints ($\Gamma$) that couple the subsystems. In such a case, the internal constraints ($\Omega$) of each subsystem can be solved in parallel, which can lead to a significant reduction in computational time. Many methods determine the interface forces iteratively, by alternating between the interface solve and internal subsystems solve. However, if we are not careful when solving for the interface constraints, coupling between subsystems is weakened and poorly conditioned systems will converge slowly. We therefore propose a substructuring solver that formulates the interface dynamics by taking into account the internal dynamics of each subsystem.

Our formulation of the interface constraints is based on the Schur complement method, and uses the effective mass of the subsystems to account for their internal behavior. With this approach, it is not necessary to perform any iteration if all the constraints in the systems are bilateral. However, multiple iterations may be needed if the subsystems contain unilateral constraints, such as contact or any constraint with bounds. This is because the effective mass depends on the internal topology of the subsystems, which changes based on the index set of the constraint variables (e.g., if a contact is detaching).

Figure 3 illustrates this idea using a chain example. Intuitively, if the chain is pulled to the right or left, the mass of the links is transferred along the chain, and any load on one end would be perceived from the other end. Similarly, we take the effective mass of every subsystem into account when solving for the interface constraints.

However, in the presence of a constraint with bounds on the impulses, the effective mass must be computed using only the free constraints. For instance, if the chain in Figure 3 is pulled down, it will make contact with the obstacle whose mass will be perceived

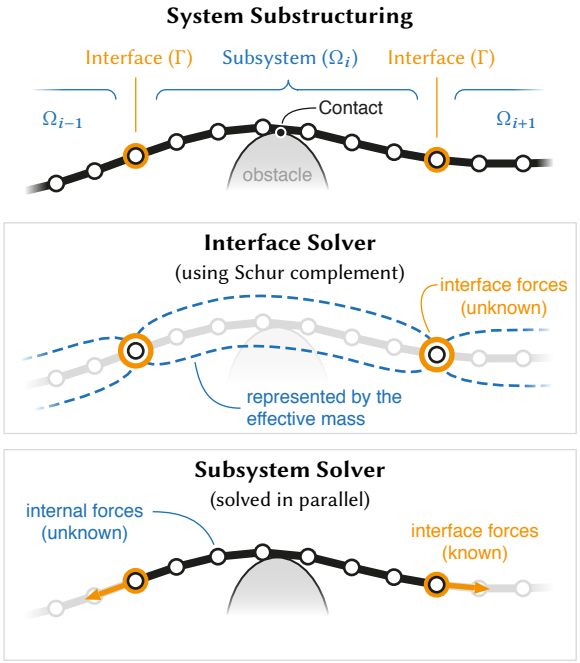



Fig. 3. Substructuring of a multibody system illustrated with a chain example in contact with an obstacle. The two solves performed by the Schur complement method are described: *interface solver* and *subsystem solver*.

by the rest of the chain. On the other hand, if the chain is pulled up and the contact detaches, it will not perceive the mass of the obstacle. Similar effects occur as the chain is pulled over the surface of the obstacle and friction forces transition from static to sliding.

Therefore, tight constraints are not considered in the computation of the effective mass, but rather as applied impulses with a fixed value determined by the bounds. As we explain using our illustrative example, this notably occurs when a contact is detaching or sliding. However, the set of free and tight constraints are not known before solving the subsystems, thus some iterations are needed in order to determine the index set of subsystem constraints. At each iteration, the free and tight sets of the internal constraint impulses are determined by the subsystem solve, which are then used to formulate the interface dynamics using the Schur complement method and solve for the interface impulses.

## 4.1 The Schur Complement Method

Consider a system partitioned into $N$ subsystems with a block diagonal mass matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_1 & & \\ & \ddots & \\ & & \mathbf{M}_N \end{bmatrix}, \tag{13}$$

where $\mathbf{M}_i$ is the mass matrix associated with the $i$-th subsystem, and the constraint Jacobian matrix can be reordered to group the constraints as internal ($\Omega$) and interface ($\Gamma$) as follows:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_\Omega \\ \mathbf{J}_\Gamma \end{bmatrix} = \left[\begin{array}{ccc} \mathbf{J}_1 & & \\ & \ddots & \\ & & \mathbf{J}_N \\ \hline \mathbf{J}_{\Gamma_1} & \cdots & \mathbf{J}_{\Gamma_N} \end{array}\right]. \tag{14}$$

Here, diagonal block $\mathbf{J}_i$ contains only the constraints inside the $i$-th subsystem, while $\mathbf{J}_{\Gamma_i}$ contains the interface constraints that couple the $i$-th subsystem with the others. The subsystems are only coupled through the interface constraints in $\mathbf{J}_\Gamma$, and each interface constraint is split among the blocks $\mathbf{J}_{\Gamma_i}$ of the subsystems it couples. Additionally, there are corresponding compliance matrices for all these constraints, which we denote $\mathbf{C}_i$ for the internal constraints of subsystem $i$, and $\mathbf{C}_\Gamma$ for the interface constraints.

The MLCP in Eq. (8) can be rewritten using this grouping as

$$\left.\begin{array}{l} \left[\begin{array}{ccc|c} \mathbf{A}_1 & & & \mathbf{G}_1^\mathrm{T} \\ & \ddots & & \vdots \\ & & \mathbf{A}_N & \mathbf{G}_N^\mathrm{T} \\ \hline \mathbf{G}_1 & \cdots & \mathbf{G}_N & \mathbf{A}_\Gamma \end{array}\right] \left[\begin{array}{c} \boldsymbol{\lambda}_1^+ \\ \vdots \\ \boldsymbol{\lambda}_N^+ \\ \hline \boldsymbol{\lambda}_\Gamma^+ \end{array}\right] + \left[\begin{array}{c} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_N \\ \hline \mathbf{b}_\Gamma \end{array}\right] = \left[\begin{array}{c} \mathbf{w}_1^+ \\ \vdots \\ \mathbf{w}_N^+ \\ \hline \mathbf{w}_\Gamma^+ \end{array}\right] \\ \mathbf{l}_i \leqslant \boldsymbol{\lambda}_i^+ \leqslant \mathbf{u}_i \qquad \forall\, i = 1 \ldots N \\ \mathbf{l}_\Gamma \leqslant \boldsymbol{\lambda}_\Gamma^+ \leqslant \mathbf{u}_\Gamma \end{array}\right\} \tag{15}$$

where the diagonal blocks $\mathbf{A}_i = \mathbf{J}_i \mathbf{M}_i^{-1} \mathbf{J}_i^\mathrm{T} + \mathbf{C}_i$ represent the inverse effective mass of the internal constraints of each subsystem. The interface constraint impulses are arranged in $\boldsymbol{\lambda}_\Gamma^+$, and the internal subsystem constraint impulses in $\boldsymbol{\lambda}_i^+$.

The block $\mathbf{A}_\Gamma = \mathbf{J}_\Gamma \mathbf{M}^{-1} \mathbf{J}_\Gamma^\mathrm{T} + \mathbf{C}_\Gamma$ represents the inverse effective mass of the bodies at the interface, and it does not include any information about the internal topology of the subsystems. Moreover, the coupling block $\mathbf{G}_i = \mathbf{J}_{\Gamma_i} \mathbf{M}_i^{-1} \mathbf{J}_i^\mathrm{T}$ maps the internal impulses of subsystem $i$ to the interface velocities, but only includes the bodies at the interface (i.e., the bodies that are simultaneously constrained by $\mathbf{J}_{\Gamma_i}$ and $\mathbf{J}_i$). From the $N$ top block-rows in Eq. (15), an MLCP for the internal constraints can be formulated for each subsystem as

$$\left.\begin{array}{r} \mathbf{A}_i \boldsymbol{\lambda}_i^+ + \mathbf{G}_i^\mathrm{T} \boldsymbol{\lambda}_\Gamma^+ + \mathbf{b}_i = \mathbf{w}_i^+ \\ \mathbf{l}_i \leqslant \boldsymbol{\lambda}_i^+ \leqslant \mathbf{u}_i \end{array}\right\} \qquad \forall\, i = 1 \ldots N \tag{16}$$

so that the internal impulses $\boldsymbol{\lambda}_i^+$ can be determined in parallel. However, an estimate of the interface impulses $\boldsymbol{\lambda}_\Gamma^+$ is needed. In order to do so, the last block-row of Eq. (15) can be used to formulate the following MLCP

$$\left.\begin{array}{r} \mathbf{A}_\Gamma \boldsymbol{\lambda}_\Gamma^+ + \displaystyle\sum_{i=1}^{N} \mathbf{G}_i \boldsymbol{\lambda}_i^+ + \mathbf{b}_\Gamma = \mathbf{w}_\Gamma^+ \\ \mathbf{l}_\Gamma \leqslant \boldsymbol{\lambda}_\Gamma^+ \leqslant \mathbf{u}_\Gamma \end{array}\right\}, \tag{17}$$

where an estimate of the internal constraint impulses of the subsystems $\boldsymbol{\lambda}_i^+$ is also needed.

One approach is to iterate between Eq. (16) and Eq. (17) until the error in the impulses satisfies a certain tolerance; such strategies have been previously proposed [Lacoursière 2007; Visseq et al. 2012].

However, the formulation of the interface dynamics in Eq. (17) only considers the coupling between bodies at the interface and the forces acting on them. This results in a weak coupling between subsystems since their internal dynamics are not represented in Eq. (17). Therefore, force based interface coupling approaches often require a large number of iterations to converge. *Warm starting* can be used to reduce the number of iterations by using impulses computed at the previous time step as an initial solution, although this may not be helpful when constraints change regularly across time steps, for instance, when new contacts are established or detach.

Instead, we formulate the interface dynamics using the Schur complement, so that the internal dynamics of the subsystems is taken into account when solving for the interface impulses $\boldsymbol{\lambda}_\Gamma^+$. This is equivalent to substituting the solution of the internal impulses $\boldsymbol{\lambda}_i^+$ from Eq. (16) into Eq. (17). When all subsystem constraints are free, $\mathbf{w}_i^+$ will be zero, and the MLCP for the interface impulses becomes

$$\left.\begin{aligned} \mathbf{S}_\Gamma \boldsymbol{\lambda}_\Gamma^+ + \mathbf{z}_\Gamma &= \mathbf{w}_\Gamma^+ \\ \mathbf{l}_\Gamma \leqslant \boldsymbol{\lambda}_\Gamma^+ &\leqslant \mathbf{u}_\Gamma \end{aligned}\right\}, \tag{18}$$

where

$$\mathbf{S}_\Gamma = \mathbf{A}_\Gamma - \sum_{i=1}^{N} \mathbf{G}_i \mathbf{A}_i^{-1} \mathbf{G}_i^{\mathrm{T}} \quad \text{and} \quad \mathbf{z}_\Gamma = \mathbf{b}_\Gamma - \sum_{i=1}^{N} \mathbf{G}_i \mathbf{A}_i^{-1} \mathbf{b}_i. \tag{19}$$

Here, $\mathbf{S}_\Gamma$ is the Schur complement of the subsystems block in Eq. (15) and it includes the inverse effective mass matrix of the subsystems $\mathbf{A}_i$, $\forall\ i = 1 \dots N$. This makes the formulation of the interface dynamics consistent with the subsystem dynamics, and thus we do not need all the coupling iterations that FBIC methods require for convergence.

If the internal impulses do not have limits (i.e., they are bilateral constraints), Eq. (18) only needs to be solved once. In that case, the internal impulses $\boldsymbol{\lambda}_i^+$ are linear in the interface impulses $\boldsymbol{\lambda}_\Gamma^+$, and the subsystems effective mass can be directly computed with matrix $\mathbf{A}_i$. Therefore, the computed interface impulses $\boldsymbol{\lambda}_\Gamma^+$ are also the solution of the original problem in Eq. (15), and can be used to solve the internal impulses $\boldsymbol{\lambda}_i^+$ in Eq. (16) in one iteration. However, in the general case, the existence of limits for the internal constraint impulses makes it necessary to recompute the Schur complement if the index set changes.

### 4.2 The Schur complement for constraints with bounds

Knowing if the internal constraints are free or tight is the key to formulating the correct interface dynamics. The Schur complement in Eq. (18) assumes that the internal impulses can take any value, as if they were unbounded. This can be a fair assumption for the free constraints because their impulses have some room to change before they reach a bound. In contrast, tight constraints have a well defined value and we therefore consider them as known.

Considering the free and tight constraints separately, the internal constraint impulses of the subsystems can be rearranged as

$$\boldsymbol{\lambda}_i^+ = \begin{bmatrix} \boldsymbol{\lambda}_{\mathcal{F}_i}^+ \\ \boldsymbol{\lambda}_{\mathcal{T}_i}^+ \end{bmatrix}, \quad \forall\ i = 1 \dots N, \tag{20}$$

where $\boldsymbol{\lambda}_{\mathcal{F}_i}^+ \in (\mathbf{l}_{\mathcal{F}_i}, \mathbf{u}_{\mathcal{F}_i})$ are the free impulses and $\boldsymbol{\lambda}_{\mathcal{T}_i}^+ = \mathbf{l}_{\mathcal{T}_i}$ or $\mathbf{u}_{\mathcal{T}_i}$ are the tight impulses at the lower or upper bound. Likewise, the constraint Jacobian matrix can be rearranged as

$$\mathbf{J}_i = \begin{bmatrix} \mathbf{J}_{\mathcal{F}_i} \\ \mathbf{J}_{\mathcal{T}_i} \end{bmatrix}, \quad \forall\ i = 1 \dots N, \tag{21}$$

where $\mathbf{J}_{\mathcal{F}_i}$ and $\mathbf{J}_{\mathcal{T}_i}$ are the constraint Jacobian matrices of the free and tight constraints, respectively. We can then rearrange the MLCP in Eq. (16) for all the subsystems, $i = 1 \dots N$, as

$$\left.\begin{aligned} \begin{bmatrix} \mathbf{A}_{\mathcal{F}\mathcal{F}_i} & \mathbf{A}_{\mathcal{F}\mathcal{T}_i} \\ \mathbf{A}_{\mathcal{T}\mathcal{F}_i} & \mathbf{A}_{\mathcal{T}\mathcal{T}_i} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}_{\mathcal{F}_i}^+ \\ \boldsymbol{\lambda}_{\mathcal{T}_i}^+ \end{bmatrix} + \begin{bmatrix} \mathbf{G}_{\mathcal{F}_i}^{\mathrm{T}} \\ \mathbf{G}_{\mathcal{T}_i}^{\mathrm{T}} \end{bmatrix} \boldsymbol{\lambda}_\Gamma^+ + \begin{bmatrix} \mathbf{b}_{\mathcal{F}_i} \\ \mathbf{b}_{\mathcal{T}_i} \end{bmatrix} &= \begin{bmatrix} \mathbf{w}_{\mathcal{F}_i}^+ \\ \mathbf{w}_{\mathcal{T}_i}^+ \end{bmatrix} \\ \mathbf{l}_i \leqslant \boldsymbol{\lambda}_i^+ &\leqslant \mathbf{u}_i \end{aligned}\right\}, \tag{22}$$

where all the free constraints satisfy $\mathbf{w}_{\mathcal{F}_i}^+ = 0$ by Eq. (12). On the other hand, for the tight constraints we can only say that $\mathbf{w}_{\mathcal{T}_i}^+ \geqslant 0$ or $\mathbf{w}_{\mathcal{T}_i}^+ \leqslant 0$, depending on the bound. Therefore, the equations of the free constraints can be written as

$$\mathbf{A}_{\mathcal{F}\mathcal{F}_i} \boldsymbol{\lambda}_{\mathcal{F}_i}^+ + \mathbf{A}_{\mathcal{F}\mathcal{T}_i} \boldsymbol{\lambda}_{\mathcal{T}_i}^+ + \mathbf{G}_{\mathcal{F}_i}^{\mathrm{T}} \boldsymbol{\lambda}_\Gamma^+ + \mathbf{b}_{\mathcal{F}_i} = 0. \tag{23}$$

By substituting the solution of the free impulses $\boldsymbol{\lambda}_{\mathcal{F}_i}^+$ from Eq. (23) into Eq. (17), we obtain the modified expression of the Schur complement in Eq. (18), such that

$$\mathbf{S}_\Gamma = \mathbf{A}_\Gamma - \sum_{i=1}^{N} \mathbf{G}_{\mathcal{F}_i} \mathbf{A}_{\mathcal{F}\mathcal{F}_i}^{-1} \mathbf{G}_{\mathcal{F}_i}^{\mathrm{T}} \tag{24}$$

and

$$\mathbf{z}_\Gamma = \mathbf{b}_\Gamma + \sum_{i=1}^{N} \left( \mathbf{G}_{\mathcal{T}_i} \boldsymbol{\lambda}_{\mathcal{T}_i}^+ - \mathbf{G}_{\mathcal{F}_i} \mathbf{A}_{\mathcal{F}\mathcal{F}_i}^{-1} \left( \mathbf{b}_{\mathcal{F}_i} + \mathbf{A}_{\mathcal{F}\mathcal{T}_i} \boldsymbol{\lambda}_{\mathcal{T}_i}^+ \right) \right). \tag{25}$$

Here, the effective inverse mass of the subsystems $\mathbf{A}_{\mathcal{F}\mathcal{F}_i}$ is computed using only the set of free constraints. Moreover, the tight impulses $\boldsymbol{\lambda}_{\mathcal{T}_i}^+$ are assumed to be known after the subsystem solve, which must be at either the lower or upper bound. This is the case of contacts that detach, where the impulse must be zero. Therefore, tight constraints do not transfer the mass like free constraints. Instead, their impulse is defined by the bounds, and they can simply be considered as applied impulses acting on the bodies.

We note that the product inside the summation in Eq. (24) is computed efficiently using a sparse Cholesky factorization of $\mathbf{A}_{\mathcal{F}\mathcal{F}_i}$. We first solve $\mathbf{A}_{\mathcal{F}\mathcal{F}_i} \mathbf{K}_{\mathcal{F}_i} = \mathbf{G}_{\mathcal{F}_i}^{\mathrm{T}}$ for $\mathbf{K}_{\mathcal{F}_i}$, and then compute $\mathbf{G}_{\mathcal{F}_i} \mathbf{K}_{\mathcal{F}_i}$. The matrix $\mathbf{G}_{\mathcal{F}_i}$ is stored using a sparse data structure, which further improves our solver performance.

## 5 SUBSTRUCTURING SOLVER

Our approach uses iterations to determine the index sets $\mathcal{I}_\Omega$ of the internal constraints $\Omega$, which determine if they are free or tight. These sets are used to formulate the Schur complement and solve for the interface impulses $\boldsymbol{\lambda}_\Gamma$. Then, the $N$ subsystems can be solved in parallel for the internal impulses $\boldsymbol{\lambda}_\Omega$, which makes up for most of the performance boost.

Algorithm 1 describes the iterative process performed by our substructuring solver. First, we initialize the index sets $\mathcal{I}_\Omega^0$ to free for all constraints. Alternatively, it is possible to warm start the solver by using the index sets of the previous time step, as we

**ALGORITHM 1:** Substructuring Solver

**function** SubstructuringSolver $(\mathbf{A}, \mathbf{b}, \mathbf{u}, \mathbf{l})$
  $k \leftarrow 0$
  Initialize Index Sets $\mathcal{I}_\Omega^0$ // all free or warm start
  **do**
    $k \leftarrow k + 1$
    $\mathbf{S}_\Gamma^k, \mathbf{z}_\Gamma^k \leftarrow$ SchurComp $(\mathbf{A}, \mathbf{b}, \mathcal{I}_\Omega^{k-1})$ // Eqs. (24) and (25)
    $\boldsymbol{\lambda}_\Gamma^k \leftarrow$ InterfaceSolve $(\mathbf{S}_\Gamma^k, \mathbf{z}_\Gamma^k, \mathbf{u}_\Gamma, \mathbf{l}_\Gamma)$ // Eq. (18)
    **for all** Subsystems $i = 1 \dots N$ **do in parallel**
      $\mathbf{b}_i^k \leftarrow \mathbf{G}_i^{\mathrm{T}} \boldsymbol{\lambda}_\Gamma^k + \mathbf{b}_i^0$
      $\boldsymbol{\lambda}_i^k \leftarrow$ SubsystemSolve $(\mathbf{A}_i^0, \mathbf{b}_i^k, \mathbf{u}_i, \mathbf{l}_i)$ // Eq. (16)
    **end**
    $\mathcal{I}_\Omega^k \leftarrow$ IndexSets $(\boldsymbol{\lambda}_\Omega^k, \mathbf{u}_\Omega, \mathbf{l}_\Omega)$ // Eq. (12)
  **while** $\mathcal{I}_\Omega^k \neq \mathcal{I}_\Omega^{k-1}$ and $k < k_{\max}$
  **return** $\boldsymbol{\lambda}^k$
**end**

later demonstrate (see Section 7.5). Then, at each iteration $k$, the algorithm uses the index sets of the previous iteration $\mathcal{I}_\Omega^{k-1}$ to form the Schur complement with the method SchurComp, i.e., $\mathbf{S}_\Gamma$ and $\mathbf{z}_\Gamma$ in Eqs. (24) and (25). The MLCP in Eq. (18) is then solved using the method InterfaceSolve, which produces the interface forces $\boldsymbol{\lambda}_\Gamma^k$. Next, the MLCP of each subsystem in Eq. (16) is solved in parallel with the method SubsystemSolve, which returns the internal impulses $\boldsymbol{\lambda}_i^k, \forall\, i = 1 \dots N$. Finally, the new index sets $\mathcal{I}_\Omega^k$ are determined and the process repeats until the index sets no longer change, or a maximum number of iterations has been reached.

If the index sets are the same as in the previous iteration, i.e., $\mathcal{I}_\Omega^k = \mathcal{I}_\Omega^{k-1}$, the algorithm terminates with the solution given by the last computed value of the impulses $\boldsymbol{\lambda}^k$. This is because performing an iteration using the same index sets would give the exact same solution. However, if the index sets are different, the next iteration starts, and the Schur complement is reformulated using the index sets found in the previous iteration $\mathcal{I}_\Omega^{k-1}$. The algorithm will also terminate with an inexact solution after a maximum number of coupling iterations.

It is important to note that the impulses computed in one iteration are not used in the next one; only the index sets. Moreover, once the correct index sets are determined, the solver terminates at the end of the coupling iteration with an accuracy that is comparable to a solution obtained using a direct method.

## 5.1 Interface Solver

The cornerstone of our substructuring approach is the method used to solve for the interface dynamics in the function InterfaceSolve. Recall that if there are contacts at the interface between subsystems, the interface dynamics formulates an MLCP, as shown in Eq. (18). There are many available solvers in the literature for such a problem, however not all of them are suitable for all applications, especially when dealing with stiff systems that can become ill-conditioned. We therefore investigate two MLCP solvers for the interface dynamics that are well-suited to this type of problem: *block principal pivoting* (BPP) [Júdice and Pires 1994] and *projected Gauss-Seidel with subspace minimization* (PGS-SM) [Silcowitz et al. 2011].

To better discuss the solution of Eq. (18), let us notice that the MLCP can be rewritten as

$$\left.\begin{bmatrix} \mathbf{S}_{\mathcal{F}\mathcal{F}_\Gamma} & \mathbf{S}_{\mathcal{F}\mathcal{T}_\Gamma} \\ \mathbf{S}_{\mathcal{T}\mathcal{F}_\Gamma} & \mathbf{S}_{\mathcal{T}\mathcal{T}_\Gamma} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}_{\mathcal{F}_\Gamma}^+ \\ \boldsymbol{\lambda}_{\mathcal{T}_\Gamma}^+ \end{bmatrix} + \begin{bmatrix} \mathbf{z}_{\mathcal{F}_\Gamma} \\ \mathbf{z}_{\mathcal{T}_\Gamma} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_{\mathcal{F}_\Gamma}^+ \\ \mathbf{w}_{\mathcal{T}_\Gamma}^+ \end{bmatrix} \right\}, \quad \mathbf{l}_\Gamma \leqslant \boldsymbol{\lambda}_\Gamma^+ \leqslant \mathbf{u}_\Gamma \tag{26}$$

where $\boldsymbol{\lambda}_{\mathcal{F}_\Gamma}^+$ and $\boldsymbol{\lambda}_{\mathcal{T}_\Gamma}^+$ are interface impulses in the free and tight set, respectively. Each solver uses different techniques to determine the index sets of the solution, and once they are determined, the value of $\boldsymbol{\lambda}_{\mathcal{T}_\Gamma}^+$ is known. Then, since $\mathbf{w}_{\mathcal{F}_\Gamma}^+ = 0$ by Eq. (12), a linear system for the free variables can be formulated as

$$\mathbf{S}_{\mathcal{F}\mathcal{F}_\Gamma} \boldsymbol{\lambda}_{\mathcal{F}_\Gamma}^+ = -\mathbf{z}_{\mathcal{F}_\Gamma} - \mathbf{S}_{\mathcal{F}\mathcal{T}_\Gamma} \boldsymbol{\lambda}_{\mathcal{T}_\Gamma}^+, \tag{27}$$

which can be solved by factorizing the matrix $\mathbf{S}_{\mathcal{F}\mathcal{F}_\Gamma}$. Here, we briefly describe the two solvers we use for the interface dynamics.

**Block principal pivoting.** BPP is a direct solver and it uses pivoting to move blocks of variables between free and tight sets, which has the advantage of producing exact solutions. The algorithm tries different sets and solves for the free impulses in Eq. (27) by factorizing the matrix $\mathbf{S}_{\mathcal{F}\mathcal{F}_\Gamma}$. In each iteration, the impulses $\boldsymbol{\lambda}_{\mathcal{F}_\Gamma}^+$ that exceed their bounds are moved to the tight set. Moreover, the complementarity between tight variables $\boldsymbol{\lambda}_{\mathcal{T}_\Gamma}^+$ and slack variable $\mathbf{w}_{\mathcal{T}_\Gamma}^+$ is checked in case they need to be moved to the free set.

**PGS with subspace minimization.** PGS-SM method uses a small number of PGS iterations to determine the free and tight sets. Then, a direct linear solve of Eq. (27) is performed in order determine the exact value of the free impulses $\boldsymbol{\lambda}_{\mathcal{F}_\Gamma}^+$. A Cholesky factorization of $\mathbf{S}_{\mathcal{F}\mathcal{F}_\Gamma}$ is again used here. If any impulses exceed the bounds, they are projected to the bounds and a new series of PGS iterations are performed. This method shows good convergence for determining the free and tight sets, and it is able to obtain the exact value of the free impulses $\boldsymbol{\lambda}_{\mathcal{F}_\Gamma}^+$.

## 5.2 Subsystem Solver

Once the interface impulses $\boldsymbol{\lambda}_\Gamma^+$ are computed, we use them to solve for the internal constraints of each subsystem in parallel. The subsystem dynamics is formulated in Eq. (16) as $N$ separate MLCPs. Since we are interested in stiff systems that are highly ill-conditioned, we use BPP as direct method to solve for the internal impulses.

At each pivoting iteration of the BPP solver, the internal impulses are split into free and tight as in Eq. (22). Since the value of $\boldsymbol{\lambda}_{\mathcal{T}_i}^+$ is known, and $\mathbf{w}_{\mathcal{F}_i}^+ = 0$, a linear system for the free constraints can be formulated as

$$\mathbf{A}_{\mathcal{F}\mathcal{F}_i} \boldsymbol{\lambda}_{\mathcal{F}_i}^+ = -\mathbf{b}_{\mathcal{F}_i} - \mathbf{G}_{\mathcal{F}_i}^{\mathrm{T}} \boldsymbol{\lambda}_\Gamma^+ - \mathbf{A}_{\mathcal{F}\mathcal{T}_i} \boldsymbol{\lambda}_{\mathcal{T}_i}^+, \tag{28}$$

which is solved at each pivoting step. To solve this linear system, the matrix $\mathbf{A}_{\mathcal{F}\mathcal{F}_i}$ must be factorized, which is done by a Cholesky decomposition. Interestingly, when the subsystems are solved, the factorization of $\mathbf{A}_{\mathcal{F}\mathcal{F}_i}$ in the last pivoting iteration can be reused to compute the expressions of the Schur complement in Eqs. (24) and (25). This is because the free set found by the BPP solver is the same that is used to determine the effective mass at the interface in

Eq. (24). Therefore, we store the factorized matrix to be used in the next substructuring iteration.

### 5.3 Convergence

In our experience, the index sets typically converge within a small number of iterations of our algorithm (see results in Section 7). The key is to obtain an approximation of the interface impulses such that the solution of the subsystems yields the correct index sets. There exists a large set of possible interface impulses that will give the correct index sets of the internal constraints, and these are all equally useful as they will allow the algorithm to terminate in the next iteration. Our solver therefore qualifies as a direct solver for substructuring.

Unfortunately, like other direct methods that use a block pivoting strategy, it is not possible to provide a proof of convergence. This is due to possible cycling of the index sets, which is difficult to predict. One strategy to deal with cycling was proposed by Júdice and Pires [1994]. Their algorithm switches to a single pivoting strategy after a pre-specified number of pivoting steps, since single pivoting schemes are proven to converge if a solution exists.

We use a similar strategy in our robust BPP implementation. A hash table based on a checksum computed from the index sets is used to detect cycles, and if a cycle is detected we fall back to single pivoting for a small number of pivoting steps, $p$, before returning to block pivoting. A value of $p = 5$ is used in all of our experiments.

### 5.4 Solving ill-conditioned systems

A common approach when using the Schur complement method is to solve the interface variables using the conjugate gradient method, since it avoids ever having to form the matrix $\mathbf{S}_\Gamma$ and thus may be solved efficiently. An example of this approach can be found in the recent work by Chu et al. [2017].

Due to changing index sets and bounded variables, conjugate gradient and its common variants are not applicable to our problem. We performed preliminary experiments with the generalized conjugate gradient (GCG) method [O'Leary 1980], and found it to be suitable for some simulations. However, this algorithm gives poor convergence with the ill-conditioned problems that result from the simulation of stiff systems. In addition, using simple preconditioners, such as the diagonal Jacobi, did not noticeably improve convergence. Therefore, we choose not to show the results here.

The ill-conditioning is inconvenient for other iterative solvers such as PGS, which exhibits slow convergence. Then, iterative solvers become completely inefficient, since the amount of iterations needed in order to obtain an accurate solution is extremely large. We reinforce this argument with examples in Section 7 and the accompanying video. Direct linear solvers, on the other hand, behave well under such conditions.

## 6 SUBSYSTEM PARTITIONING

When using substructuring techniques, a very important aspect is how to partition the system into smaller subsystems. In the cases which involve systems with a known topology, such as mechanisms, the subsystems can easily be chosen as distinct parts of the system. For instance, a load lifted by a cable crane can be partitioned so that crane, cable, and load constitute three separate subsystems. However, contact makes partitioning more challenging because it changes the topology of the system by making and breaking contacts between the bodies. Therefore, a good partitioning strategy is required for systems where the constraint topology is only available at runtime, such as piles of bodies with contact.

### 6.1 Semantic partitioning

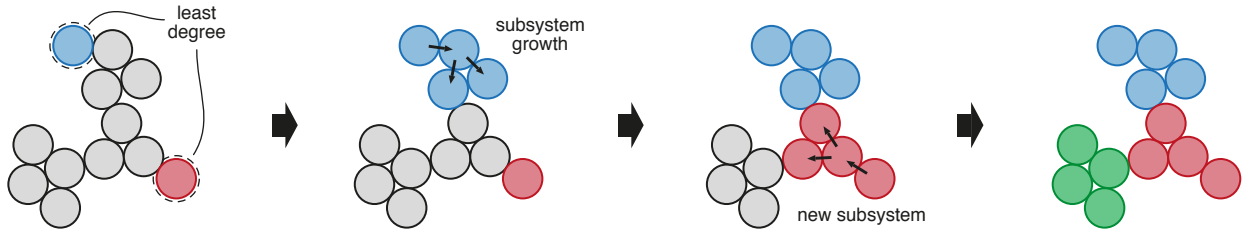
In some instances, the dominant coupling effects of a system are determined by bilateral constraints, with only a small number of contacts occurring at runtime. This presents a known topology that does not change much with time, and so fix-sized system partitioning can be very useful. In such cases, it is the task of the user to define the subsystems, which are usually grouped based on their function in the system or their similarity. This *semantic partitioning* benefits from considerations which are often intuitive to the user, which can later translate into better numerical performance. For instance, grouping the bodies so that the subsystems present a certain topology, such as a chain or a tree, can reduce the fill-in of the subsystem lead matrix, thus lowering the computational time for each subsystem solver. Moreover, in some cases, it is possible to reduce the number of contacts within the subsystems and have contacts only at the interface, which further simplifies subsystem dynamics. On the other hand, when there is a significant presence of contact in the system, such as a large pile of bodies, a user defined partitioning is not feasible because the constraint graph topology changes at each time step. Therefore, an algorithm to generate the subsystems based on the topology of the system can be very useful.

### 6.2 Minimum degree partitioning

In order to automatically partition our simulations, we propose a heuristic that searches the constraint graph and forms groups of connected bodies. Our *minimum degree partitioning* algorithm begins at a body on the periphery of an initial group and adds adjacent bodies based on their degree, or connectivity, in the constraint graph. This approach is inspired by the Cuthill-McKee algorithm [Cuthill and McKee 1969], which reduces fill-in and creates small bandwidth matrices. Bodies adjacent to a group are added based on their degree in ascending order, such that bodies with a minimum degree are prioritized at each step. The degree of each body is computed by only considering connections to other bodies in the initial group.

Once the number of bodies in a group reaches the prescribed maximum per subsystem, the algorithm begins again by starting at a periphery body using the same criterion and a new subsystem begins to grow. Since the connectivity changes dynamically (i.e., due to contact) the partitioning is reinitialized at each time step based upon the constraint graph. Figure 4 shows an illustrative example demonstrating our partitioning scheme.

We found that this partitioning strategy produces constraint orderings with reduced fill-in and small bandwidth in the subsystem matrices $\mathbf{A}_i$, and also helps to reduce connections between adjacent subsystems. The former characteristic is important for efficiently solving individual subsystems, whereas the latter is an important consideration when assembling $\mathbf{S}_\Gamma$ using Eq. (24).




Find the bodies that have less bodies in contact with them (least degree).

Grow the subsystem by adding the adjacent bodies with least degree.

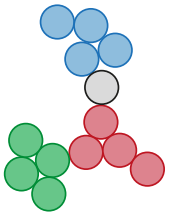
When the maximum number of bodies per subsystem is reached, start again.

Finish when the number of bodies left is small enough to form a subsystem.

Fig. 4. Minimum degree partitioning for an unstructured system with $n_{\max} = 4$.

Algorithm 2 provides pseudo-code for our minimum degree partitioning method. Initially, the algorithm creates a default group $\mathcal{G}_0$ containing all the bodies in the system. Then, the body $\beta \in \mathcal{G}_0$ with minimum degree is chosen to start a new group $\mathcal{G}_i$, and it is removed from the default group. The body group grows following the criterion of minimum degree, so that the next chosen body $\beta \in \mathcal{A}_i \cap \mathcal{G}_0$ is the adjacent body to the subsystem with minimum degree. Bodies are successively added to the group until the number of bodies in the group is equal to the maximum number $n_{\max}$, which can be defined by the user or based on the total number of bodies. Then, new groups are created following the same procedure, and the algorithm terminates when the default group contains less bodies than $n_{\max}$.

**Orphan partitions.** Sometimes the minimum degree algorithm will produce partitions consisting of just one body, such as the gray body shown in the example on the right. This results in additional constraints being added to the interface set and unnecessary work for our algorithm.



To avoid creating *orphan* partitions, we perform an additional check for any adjacent bodies with degree zero. In other words, bodies that are not connected via a constraint to any other bodies in the default group $\mathcal{G}_0$. If such bodies exists, we simply add them to the new partition $\mathcal{G}_i$. Although this produces a partition that exceeds the maximum body threshold, our algorithm benefits from reduced overhead due to fewer constraints in $\Gamma$.

**ALGORITHM 2:** Minimum Degree Partitioning

```
1:  function MinimumDegreePartitioning
2:      G_0 ← all bodies                          // initialize default group
3:      i ← 1                                     // number of groups
4:      while Size(G_0) > n_max do
5:          β ← MinDegreeBody(G_0)                // body with min. degree
6:          G_i ← {β}                             // create a new group
7:          G_0 ← G_0 − {β}
8:          while Size(G_i) < n_max do
9:              A_i ← AdjacentBodies(G_i)         // get adjacent bodies
10:             β ← MinDegreeBody(A_i ∩ G_0)      // get next body
11:             G_i ← G_i + {β}                   // add body to the group
12:             G_0 ← G_0 − {β}
13:         end while
            // check for orphan partitions
14:         A_i ← AdjacentBodies(G_i)
15:         O = {β | β ∈ A_i and Degree(β) is zero}
16:         G_i ← G_i + O
17:         G_0 ← G_0 − O
18:         i ← i + 1
19:     end while
20: end function
```

## 7 RESULTS

This section evaluates our proposed substructuring method using various challenging examples. Many of these involve high mass ratios, stiff constraints, and long kinematic chains. As demonstrated in the accompanying video, fixed-point iterative methods such as PGS usually fail to achieve the same quality as direct solvers and require many iterations.

All results were obtained using a six core Intel Core i7 3.3 GHz CPU with hyper-threading enabled. Simulations were performed using a time step of $h = 1/60$ s. We use the Vortex physics engine to perform collision detection and build constraint Jacobian matrices. The algorithms for substructuring, constraint solvers, and subsystem partitioning are implemented in C++. We use the Eigen linear algebra library to perform matrix and vector operations. A sparse Cholesky factorization is used to solve the linear systems in Eq. (16) and Eq. (18), and for the subsystem matrix $\mathbf{G}_i\mathbf{A}_i^{-1}\mathbf{G}_i^{\mathrm{T}}$.

We compare the performance of our substructuring method to the BPP solver without substructuring, which we consider as a baseline algorithm. Two implementations of this solver are included in the results: one using the sparse Cholesky decomposition provided by Eigen (Eigen-BPP), and another using the PARDISO direct sparse solver (PARDISO-BPP) [Intel 2019]. We also include a comparison to PGS, which is a popular method among real-time physics engines, and the FBIC technique proposed by Lacoursière [2007]. We implement our substructuring method using the two solver algorithms described in Section 5.1, that is, block principal pivoting (Schur-BPP), and projected Gauss-Seidel with subspace minimization (Schur-PGS-SM). We use a multithreaded implementation for our substructuring methods (Schur-BPP, Schur-PGS-SM) and for the FBIC and PARDISO-BPP solvers. We use a single threaded implementation for the Eigen-BPP and PGS solvers.

Table 1. Default parameters for each of the solvers used in our experiments, including: maximum number of coupling iterations ($k_{\max}$), the maximum iterations or pivoting steps ($\kappa_{\text{iter}}$), the tolerance when computing index set changes ($\eta$), stopping tolerance of iterative methods ($\epsilon$).

| Solver | $k_{\max}$ | $\kappa_{\text{iter}}$ | $\eta$ | $\epsilon$ |
|---|---|---|---|---|
| (Eigen/PARDISO)-BPP | - | 50 | $10^{-5}$ | - |
| PGS | - | 1000 | - | $10^{-9}$ |
| Schur-BPP | 10 | 50 | $10^{-5}$ | - |
| Schur-PGS-SM | 10 | 50 | $10^{-5}$ | $10^{-9}$ |
| FBIC | 250 | 50 | $10^{-5}$ | - |

Table 2. The partitioning scheme (and bodies per partition) and the average condition number of the matrix $\mathbf{S}_\Gamma$ for each example.

| Example | Partitioning (# of bodies) | cond($\mathbf{S}_\Gamma$) |
|---|---|---|
| Log tower | min.degree (16) | $2.2\times10^{6}$ |
| Cable spinner | semantic (36–48) | $3.9\times10^{7}$ |
| Chain wrap | semantic (16) | $5.6\times10^{4}$ |
| Chain drop | semantic (16) | $2.7\times10^{4}$ |
| Rock pile | min. degree (16) | $1.5\times10^{4}$ |
| Net with truck | semantic (48) | $2.5\times10^{12}$ |
| Tandem cranes | semantic (24–50) | $4.8\times10^{10}$ |

In order to provide a fair comparison of their runtime performance, each solver was tuned to produce behavior qualitatively similar to the baseline Eigen-BPP algorithm. Table 1 gives the parameters used for each solver, which are used across all of our experiments unless stated otherwise. In the case of the PGS and FBIC algorithms, reasonable behavior was often unachievable without allowing a very large number of Gauss-Seidel iterations or coupling iterations. These methods also use *warm starting*, which initializes the solution with constraint impulses from the previous time step. Substructuring was not used for the Eigen-BPP and PGS solvers, meaning that the entire system was solved using a single matrix, although in the case of PGS the matrix is never fully assembled.

Graph coloring was not used to parallelize the PGS solver since the convergence of this method was poor for the systems we tested. Likewise, other researchers have observed that parallel PGS may introduce jitter artifacts in the simulation Tonge et al. [2012].

A maximum iteration count of 1000 and 250 was used for PGS and FBIC, respectively. Warm starting was also used for both of these solvers, in which the constraints impulses at the previous time step are used as an initial solution for the current time step. However, even with many iterations, these methods struggled to produce results that match the baseline.

### 7.1 Examples

Here we provide a brief description of the examples used to evaluate our substructuring approach. Figure 5 shows screenshots from simulations of these examples, and the color coded partitioning of some examples is shown in Figure 1. Constraint compliances used to regularize the multibody system (i.e., diagonal values of $\mathbf{C}$) range from $10^{-7}$ to $10^{-10}$. The examples involve mass ratios up to $40,000:1$. Therefore, due to constraint redundancy and the large mass ratios, the MLCP matrices in the examples typically have high condition numbers. Table 2 gives the condition number of the interface matrix $\mathbf{S}_\Gamma$, as well as the partitioning scheme we use for each example.

**Log tower.** This example involves 160 logs (50 kg each) stacked vertically with a 2500 kg box dropped on top. Partitions are created using the minimum degree algorithm proposed in Section 6, and the partitions change at each time step. The stack is initially stable, but is perturbed when a rolling ball collides with the base of the stack that causes it to collapse.

**Cable spinner.** A complex system involving 8 flexible cables is used to suspend a 10 kg plate and 2500 kg ball. Each cable consists of Cable links (0.1 kg each) coupled to neighboring links using a joint with 6 degrees of freedom that is nearly inextensible, but allows bending and torsional motion.

**Chain wrap.** A stiff chain consisting of 100 links (0.25 kg each) is used to support a massive box (500 kg). Each neighboring link is coupled by a universal joint, with the last link coupled to the box. A user defined partitioning algorithm is used to create subsystems of each sequential grouping of 12 bodies.

**Chain drop.** A stiff chain consisting of 250 links (0.25 kg each) is dropped onto an inclined plane. Each link is connected by a universal joint, and the chain is partitioned into groups of 32 bodies each. The friction coefficient is low enough so that the chain slowly slides off the plane. This example evaluates the ability of our approach to handle coupling between subsystems when many friction variables are sliding (tight).

**Rock pile.** A large pile of rocks (masses ranging from $80-250$ kg) is pushed by a stiff, lightweight shovel (1 kg). The constraint connectivity of the system is not known *a priori*. Therefore, this highly unstructured simulation tests our partitioning algorithm. Partitions are created using the minimum degree algorithm proposed in Section 6.

**Net with truck.** A dump truck is dropped onto a web of cables. The cables have high stiffness, but very low mass (0.2 kg per body). Each section of cable is a partition consisting of $48-65$ bodies and more than 280 constraints per partition. The truck (8000 kg) is a separate partition consisting of 22 bodies and more than 120 constraints. A user defined partitioning is used to decompose the simulation so that the truck constitutes one subsystem, and each cable section is one subsystem. Exceptionally, a maximum iteration count of 1000 was needed with this example when using the FBIC method.

**Tandem cranes.** Two cranes (approx. $75,000$ kg each) suspend a heavy pipe (3500 kg) in the air. Each crane consists of several subsystems: the cable systems, vehicle track and drivetrain, boom and hoist, and hooking mechanisms. The pipe is given an initial velocity to produce a trajectory that results in sliding motion.

### 7.2 Performance

A summary of timing information and speedup factors for each solver for all examples can be found in Table 3. Figure 7 shows the time to solve the dynamical system for several of the examples shown in Figure 6.

We note that our method gave a significant speedup in all the examples, and in several cases was an order of magnitude faster. The

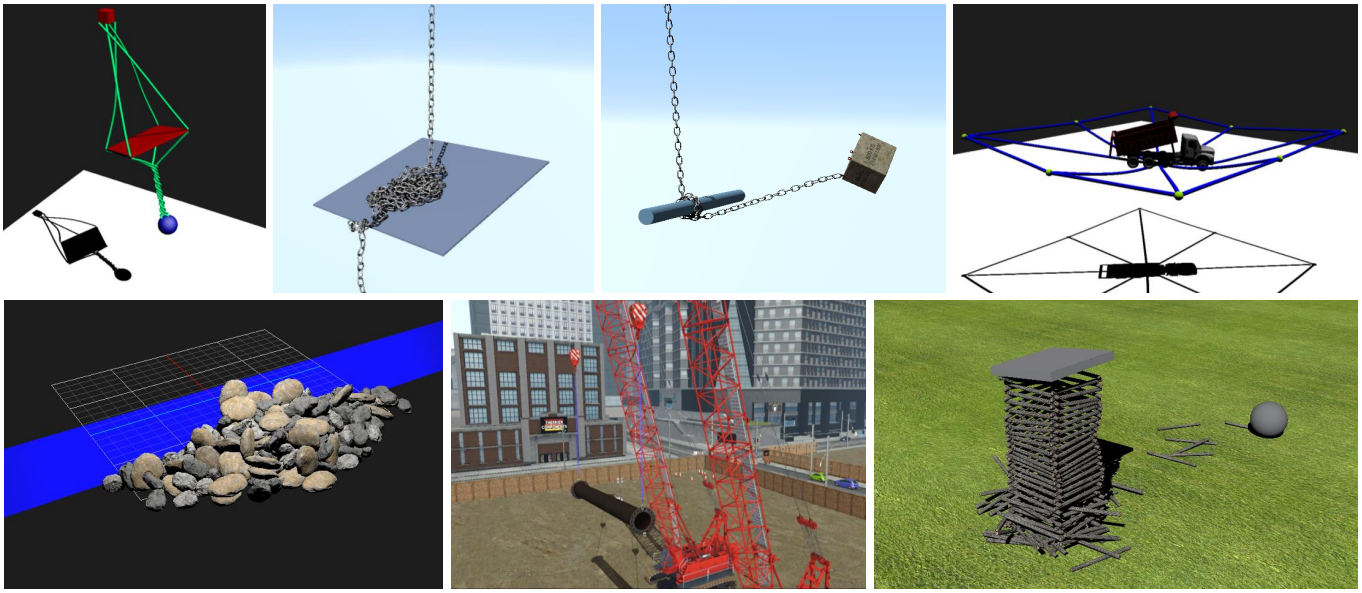


Fig. 5. The examples used in our performance evaluation. Top row (left-to-right): cable spinner, chain drop, chain wrap, net with truck. Bottom row (left-to-right): rock pile, tandem cranes, log tower.

Table 3. The average wall-clock time of the MLCP solver for all the time steps (and speedup factor) for each example. The average number of coupling iterations used for each example (avg. $k$) and the percentage of frames where the maximum number of coupling iterations is reached (max. $k$) are also provided. The fastest algorithm is highlighted in gray. Speedup factors are computed relative to the baseline algorithm, which is the Eigen-BPP method without substructuring. Examples with an asterisk (*) regularly reach the maximum number of coupling iteration.

| **Example** | Eigen-BPP | PGS | Schur-BPP | Schur-PGS-SM | FBIC | PARDISO-BPP | avg. $k$ | max. $k$ |
|---|---|---|---|---|---|---|---|---|
| Log tower* | 202.6 ms | 302.4 ms (0.7×) | 59.7 ms (3.4×) | 51.4 ms (3.9×) | 111.6 ms (1.8×) | 113.4 ms (1.8×) | 5.8 | 39.2 % |
| Cable spinner | 141.3 ms | 229.1 ms (0.6×) | 8.7 ms (16.2×) | 8.8 ms (16.0×) | 93.2 ms (1.5×) | 170.5 ms (0.8×) | 2.1 | 0 % |
| Chain wrap | 54.6 ms | 198.5 ms (0.3×) | 11.6 ms (4.7×) | 12.3 ms (4.4×) | 132 ms (0.4×) | 27.6 ms (2.0×) | 4.4 | 1.9 % |
| Chain drop | 46.9 ms | 150.5 ms (0.3×) | 9.1 ms (5.2×) | 8.7 ms (5.4×) | 20.5 ms (2.3×) | 20.4 ms (2.3×) | 3.2 | 2.1 % |
| Rock pile* | 311.7 ms | 284.4 ms (1.1×) | 101.5 ms (3.1×) | 98.6 ms (3.2×) | 373.9 ms (0.8×) | 102.5 ms (3.0×) | 6.3 | 10.2 % |
| Net with truck | 252.3 ms | 802.4 ms (0.3 ×) | 11.9 ms (21.1×) | 12.7 ms (19.9×) | 632.0 ms (0.4×) | 396.7 ms (0.6×) | 2.6 | 0 % |
| Tandem cranes | 180.6 ms | 222.9 ms (0.81×) | 12.9 ms (13.9×) | 13.0 ms (13.8×) | 134.3 ms (1.3×) | 100.4 ms (1.8×) | 2.4 | 0.2 % |

Schur-BPP implementation using our proposed method typically gave best performance. However, for the log tower and rock pile examples, Schur-PGS-SM gave slightly better performance. We note that these examples involve complex contact between a large number of bodies, which indicates that PGS-SM may be better suited to solve the interface constraints in such cases.

Our algorithm was faster than the PARDISO implementation of the BPP solver in all examples. Surprisingly, the PARDISO version performed slower than the Eigen implementation of the BPP solver in some instances, specifically, the net with truck, and the cable spinner. We hypothesize that this is due to the overhead of performing a sparsity analysis, which is done automatically by the PARDISO solver, and in certain cases this step is futile. For the rock pile example, we also note that the PARDISO solver performs similarly to the Schur-BPP and Schur-PGS-SM algorithms (see right panel in Figure 7). This is due to the diminishing returns offered by our algorithm for simulations requiring more coupling iterations.

### 7.3 MLCP error

We compute the error when solving Eq. (8) by the *natural residual* [Enzenhöfer et al. 2018]. The residual for each variable $i$ is computed as

$$r_i = \max\left(\left|\min\left(\lambda_i^+ - l_i,\ w_{\mathrm{l},i}^+\right)\right|,\ \left|\min\left(u_i - \lambda_i^+,\ w_{\mathrm{u},i}^+\right)\right|\right). \qquad (29)$$

We report the total MLCP error of the system as $\|\mathbf{r}\|$. Figure 8 shows the per frame error for the examples shown in Figure 6. Our approach produces error which is comparable to the baseline Eigen-BPP solver, and many orders of magnitude reduced compared to the other methods.

The noisy plots in Figure 8 indicate that the Rock pile example is one of the most challenging examples. As reported in Table 3, for a significant percentage of frames, our algorithm did not converge within the allotted number of coupling iterations $k_{\max}$. These correspond to time steps where a large error is reported in the plot, although the simulation remained stable and visually authentic.

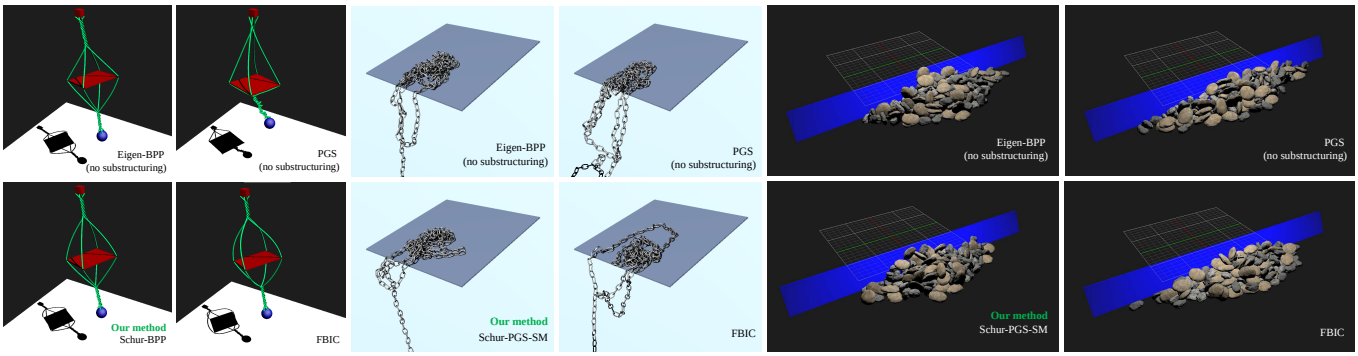



Fig. 6. Representative frames from the cable spinner (left), chain drop (middle), and rock pile (right) examples. Each panel provides a visual comparison of the baseline method (Eigen-BPP) with our method (Schur-BPP,Schur-PGS-SM) and other existing methods (PGS and FBIC).

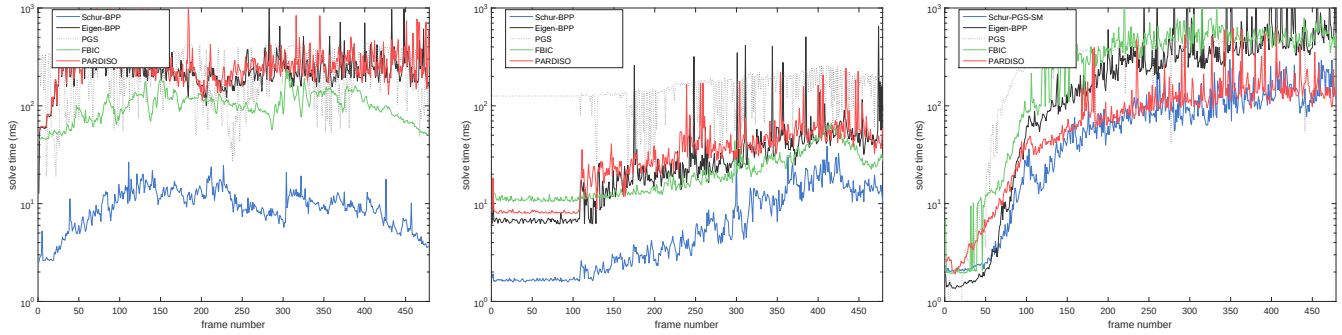



Fig. 7. Solve time per frame for the cable spinner (left), chain drop (middle), and rock pile (right) examples, showing the first 8 s of the simulation

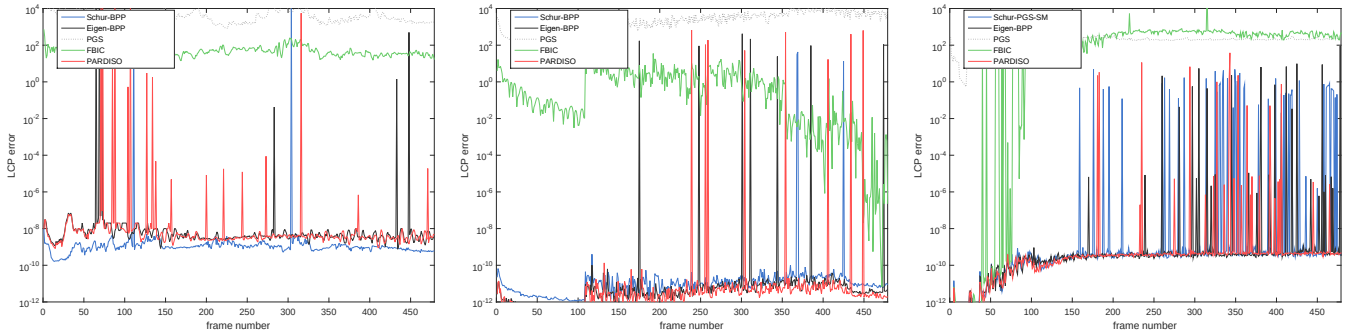



Fig. 8. LCP error per frame for the cable spinner (left), chain drop (middle), and rock pile (right) examples, showing the first 8 s of the simulation.

Note that the algorithm converges most of the time in the other examples.

The rock pile and log tower are not only problematic for our approach, but the baseline algorithm too. This is due to the large number of non-interpenetration and friction constraints at the interface that alternate between tight and free index sets at each time step. In these cases, we found that the PGS-SM algorithm gave slightly better performance and error convergence, and so we report performance values for this algorithm for the rock pile and log tower example in Table 3.

We observed that with the PGS method it was often impossible to reduce the error below a minimum bound, even by increasing the number of iterations and lowering the stopping tolerance. In other words, convergence of the PGS method stagnates. Similar behavior could be observed for the FBIC method, although lower error could often be achieved.

## 7.4 Qualitative comparison

Since the interface in many examples typically consists of non-interpenetration and friction constraints, error often manifests as

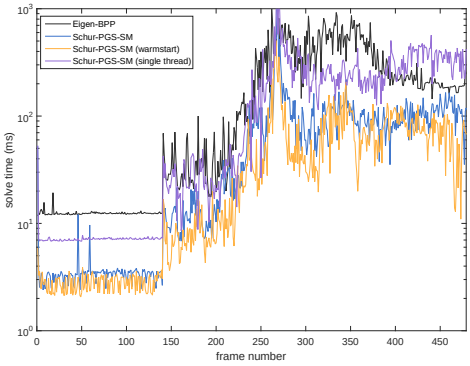



Fig. 9. Comparison of performance for the log tower example using a single threaded implementation of our substructuring solver (violet) versus a multithreaded implementation with warm starting (orange).

noticeable penetration and pronounced sliding. For instance, in the rock pile example, the PGS and FBIC methods produced behavior where the rocks spread out quickly compared to the baseline, whereas our approach better preserves the integrity of the pile. In the same example, we also observed that some rocks would occasionally slip underneath the blade when simulating with too few iterations for the PGS and FBIC algorithms, or if warm starting was not used.

The PGS algorithm also produced artifacts in the cable spinner example. Specifically, some of the cables pass through each other, and thus become permanently entangled. This demonstrates the inability of this method to deal with large forces produced by the tightly intertwined cables. The simulation with the baseline and our approach does not produce these artifacts.

With the log tower, it begins to collapse prematurely when simulated with PGS and FBIC. The final distribution of logs and location of the heavy box is most faithfully simulated using our method compared to the baseline.

In the chain drop example, we note that the chain slides off the inclined plane early for the PGS and FBIC solvers. In the chain wrap example, the PGS solver produces visible sliding and bouncing of the chain, whereas the FBIC solver results in noticeably more vibrations in the chain and a different wrapping pattern. Our approach produces a simulation most similar to the baseline.

With the net example, neither PGS or FBIC are able to produce adequate tensile forces to support the truck. However, our approach gives nearly identical behavior to the baseline. Further side-by-side comparisons with the examples and each solver method can be found in the supplementary video.

## 7.5 Single threaded and warm starting

A benefit of our method is that the workload of simulating constrained multibody systems can be shared across multiple processor units. However, single threaded implementations also stand to benefit from our method. Figure 9 shows the solve time per frame for the log tower for a single threaded implementation of our method using the Schur-PGS-SM solver. Simply decomposing the simulation into smaller subsystems gives a performance speedup. We attribute this to the Cholesky factorization, which has $O(m^3)$ computational

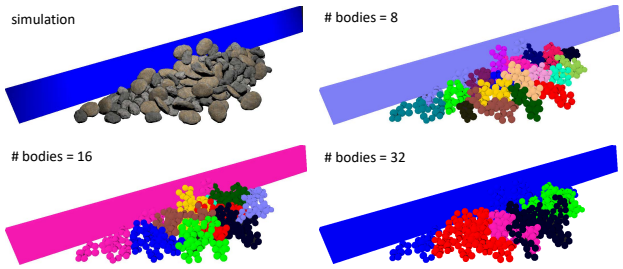



Fig. 10. Partitioning of the rock pile (upper left) using our minimum degree algorithm, showing 8 (upper right), 16 (lower left), and 32 (lower right) bodies per partition. Each color is a label indicating the partition for the body.

Table 4. The average solve time and average partition count for examples using the minimum degree partition algorithm. The performance of our substructuring algorithm varies with the number of bodies per partition. Parameters highlighted in gray were used in our evaluation.

| **Example** | # bodies | avg. solve time | avg. partitions |
|---|---|---|---|
| Log tower | 8 | 53.4 ms | 30.4 |
| | 16 | 51.4 ms | 19.1 |
| | 32 | 63.3 ms | 14.3 |
| | 64 | 88.8 ms | 12.8 |
| Rock pile | 8 | 169.1 ms | 21.2 |
| | 16 | 98.6 ms | 12.1 |
| | 32 | 129.7 ms | 6.8 |
| | 64 | 142.7 ms | 4.9 |

complexity where $m$ is the number of constraints. By breaking the system into smaller components, this reduces the computational complexity to $NO(p^3)$, where $N$ is the number of subsystems and $p$ is the average size of the reduced subsystem, which is typically much smaller than $m$.

Figure 9 also demonstrates that our approach can benefit from using the index set of constraints from the previous time step. This is a type of warm starting, though we note that the performance improvement is moderate (when contacts frequently detach, there is little benefit to warm starting the solution). However, if the active set changes infrequently, such as when the tower is static at the beginning of the simulation or when the logs are piled at the end, using data from the previous step helps improve performance.

## 7.6 Partitioning

We use the algorithm described in Section 6 to automatically create subsystems based on the constraint graph. Figure 10 shows the result of applying our minimum degree partitioning scheme to the rock pile example. Bodies of the same color are grouped into the same partition, and our algorithm traverses the constraint graph to form coherent subsystems. Although the colors change from frame-to-frame, the grouping of bodies is consistent. Occasionally, bodies become entirely disconnected from the rest of the simulation, creating *islands*, in which case they are arbitrarily added to one of the existing subsystems. The accompanying video also shows color

coded examples of our partitioning scheme across many simulation frames. The time required to perform partitioning is 0.1 ms for the rock pile example, and 0.2 ms for the log tower example.

Our partitioning algorithm is parameterized by the maximum number of bodies per partition. We evaluate the effect of this parameter by measuring the performance of the simulation and sweeping multiple values of the parameter. The results are shown in Table 4. It is clear that the size of the subsystems (and total number of subsystems) has an effect on the performance of our method. Ideally the number of partitions would match the number of available processing units. However, since our method computes a Cholesky factorization at each step, large subsystems can adversely affect the performance. This is therefore a parameter of our method that must be tuned for optimal performance.

We have noticed small changes in the solution which depend on the partitioning. These can be observed in the accompanying video for the chain wrap, log tower, and rock pile examples. This sensitivity is to be expected when solving such stiff systems involving unilateral constraints. However, we note that the motion produced by our approach is qualitatively similar to the baseline algorithm independent of the partitioning scheme.

### 7.7 Scalability analysis

We evaluate the scalability of our method using several experiments to assess the *strong scaling* and *weak scaling* of the approach. Strong scaling is measured by simulating a fixed-size problem and increasing the number of threads. Weak scaling is measured by increasing the size of the simulation along with the number of threads (e.g., if the number of bodies is doubled, so are the number of threads). For this analysis we use the log tower and chain wrap examples, since the former involves only unilateral contact constraints and the latter involves a mix of bilateral and unilateral. A bridge example is also introduced that involves only unbounded bilateral constraints. The log tower and bridge are shown in Figure 11 along with their partitions.

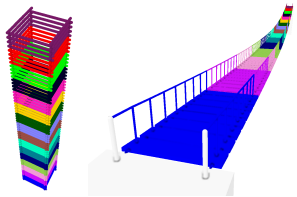


Fig. 11. The log tower with 256 logs (left); the bridge with 128 planks has a total of 893 bodies and 1278 bilateral constraints (right).

The scalability experiments were performed on a 16 core Intel Core i9 2.80 GHz CPU. Note that in all examples used for the scalability analysis, our substructuring technique was applied and the larger system was decomposed into smaller individual subsystems.

**Strong scaling.** Figure 12 shows the simulation speedup for a variety of examples using up to 16 threads. These plots highlight the portion of the solve time that can be parallelized. Performance is expected to increase with the number of cores due to the parallelization of solving individual subsystems. However, the speedup ratio diminishes as the number of threads increases. This is due to the inherent overhead of each coupling iteration, which is not parallelizable and thus becomes the bottleneck of our solver. The scaling of systems with a defined topology, such as the chain wrap and bridge examples, follows a monotonic increase in speedup. These examples

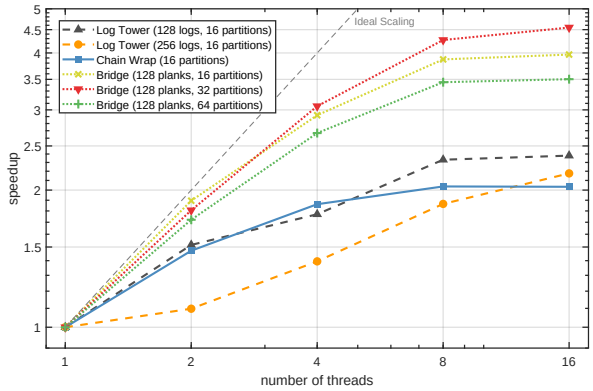



Fig. 12. Speedup of fixed-size simulations of the log tower, chain wrap, and bridge when using various numbers of threads. A plot of the ideal scaling is shown as reference.

use the semantic partitioning. Whereas for unstructured systems like the log tower example, the scaling is less effective. We attribute this to two main factors: (i) using the minimum degree partitioning does not guarantee that the number of partitions and their size are constant along the simulation, which might contribute to an uneven load of threads; (ii) a larger number of partitions translates into a larger interface, which further increases the iteration overhead.

**Weak scaling.** Figure 13 shows the solve time for multiple sizes of the log tower and bridge examples, with the number of threads increased proportional to the size of the simulation. We note that the solve time increases as the size of each example increases. This is due to the aforementioned overhead and uneven load distribution among cores.

Nevertheless, for some examples, the multithreaded implementation of our method using 16 threads achieves nearly a 5× speedup with respect to the single threaded implementation. Note that the bridge example only contains bilateral constraints, thus our method converges in one coupling iteration. In systems with contact, such as the chain and tower examples, the scaling is expected to be inferior since the solution of an MLCP is required for the interface constraints, which further increases the overhead. Therefore, the maximum scaling speedup is given by examples with fewer contacts and a small interface.

## 8 CONCLUSIONS

Solving multibody systems can be challenging when they involve large numbers of bodies, and complex systems of constraints. Unilateral contact, friction, and bilateral constraints with limits can make the problem particularly difficult. Additional numerical issues arise in systems with widely varying masses, singular configurations, and redundant constraints. For stiff, poorly conditioned systems, direct methods are desirable to ensure accurate solutions, but direct methods typically do not scale well as systems become large. Our contribution is a new substructuring method for the efficient solution of large multibody systems involving unilateral constraints. We use direct methods to solve both the subsystems and the interface, which produces high quality solutions. Subsystem solves are trivially parallelized, while solving the interface using the effective mass

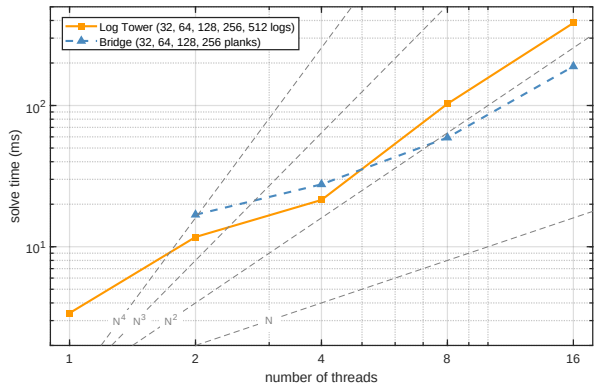



Fig. 13. Solve time using different number of threads for the log tower and bridge examples. As the size of each simulation is increased, so are the number of threads. We show plots of some simple polynomials as reference.

of subsystems provides an efficient means of resolving accurate subsystem interaction forces. Our algorithm typically only needs 3 to 4 coupling iterations to identify the active constraints and solve the system. We describe a simple yet effective partitioning method that can be applied to any multibody system to target a desired number of CPU cores. We demonstrate our method on a number of challenging simulation scenarios and show that we can compute solutions up to 20 times faster than traditional solvers used in industry without compromising solution quality.

## 8.1 Discussion

Table 3 indicates that using the BPP and PGS-SM algorithms to solve for the interface constraints offer similar performance improvements. The algorithms differ fundamentally in how they estimate the index set. The BPP method performs a search in the space of possible index sets (active and tight). However, if a solution cannot be found in the allotted number of pivoting steps, it may return a solution which is far from a physically valid one. In such cases it is easy for the simulation to become unstable. The PGS-SM algorithm estimates the index set using Gauss-Seidel iterations. We found that it will often make progress towards a more stable solution, even if it is not an exact one. It also seems better suited to handle nearly degenerate problems, which can arise in scenarios involving complex contact at the interface.

In our experiments, we use only the BPP method to solve for individual subsystems. But a heterogeneous combination of solver algorithms is possible with our proposed technique. In other words, the solver could be selected based on the characteristics of each subsystem. For instance, PGS may be used if approximate solutions are suitable for a particular subsystem. Integration with our method would only require adding an extra step to compute the Cholesky decomposition of the $\mathbf{A}_{\mathcal{F}\mathcal{F}_i}$ matrix once the iterative algorithm has converged.

## 8.2 Limitations

A major limitation of our approach is that the performance benefits diminish as the number of coupling iterations increases. This can be seen in Table 3. The speedup factor appears to be inversely proportional to the average number of coupling iterations. We attribute this to the overhead of factorizing the matrix $\mathbf{S}_\Gamma$ at each coupling step. This serial portion of our algorithm must be done at each coupling iteration, even if only one of the subsystems has changed its index set. This is most problematic when there are a significant number of redundant contacts in the simulation, such as in the log tower and rock pile. In other words, complex and unstructured piles of bodies are problematic for our method. We observed that only a small percentage of variables (usually friction constraints) persistently pivot after the first few coupling iterations. Yet these prevent our algorithm from converging within the allotted number of iterations. Our BPP implementation is robust in that it detects index set cycling within each subsystem solve. However, we do not currently have a strategy to deal with cycling that can occur during coupling iterations. This occurs primarily when simulating the aforementioned contact scenarios and, in some cases, can prevent our algorithm from converging.

From a modeling standpoint, the use of box friction in our work presents some limitations since it assumes that the normal force is known based on the previous time step. This can be problematic, for instance, if contacts form and detach frequently and their correspondence to the previous time step is ambiguous. One way to resolve this is by performing an initial pass to estimate the normal force at each time step and then updating Coulomb friction bounds accordingly. However, our approach is agnostic to the underlying contact model, so long as it can be represented as a complementarity problem. For instance, we could use the formulation of Anitescu and Potra [1997].

## 8.3 Future work

There are several interesting avenues for future work. The heuristic we use for automatic partitioning can likely be improved by taking inspiration from the vast body of work on graph partitioning. For even larger systems, we believe that our substructuring technique can be applied recursively, with solver iterations resembling multigrid cycles. Recursive partitioning of subsystems could lead to interesting new results in out-of-core solves of massive systems of bodies with unilateral and bilateral constraints.

Currently, our method recomputes a Cholesky factorization whenever the index set of a subsystem changes. We believe the efficiency of our method could be improved by computing the factorization of $\mathbf{A}_i$ once, and then reusing the existing Cholesky factorization to obtain $\mathbf{A}_{\mathcal{F}\mathcal{F}_i}$. For instance, Enzenhöfer et al. [2019] recently proposed an efficient block pivoting method that successively applies low rank downdates to an initial factorization based on the current index set.

Finally, a possible solution to mitigate artifacts due to non convergence would be to perform a last step of our algorithm, with pivoting disabled. This would only require an additional linear solve and it would ensure consistency across the interface and subsystem solutions.

## ACKNOWLEDGMENTS

The authors would like to thank Daniel Holz and Andreas Enzhöfer at CM Labs Simulations for their feedback and for providing the Tandem Crane example. We gratefully acknowledge the support of CM Labs Simulations, the Natural Sciences and Engineering Research Council of Canada (NSERC), and Prompt Quebec.

## REFERENCES

Acary, V. and Brogliato, B. 2008. *Numerical Methods for Nonsmooth Dynamical Systems*. Springer-Verlag, Berlin, Heidelberg.

Andrews, S., Teichmann, M., and Kry, P. G. 2017. Geometric Stiffness for Real-time Constrained Multibody Dynamics. *Computer Graphics Forum* 36, 2. https://doi.org/10.1111/cgf.13122

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

Baraff, D. 1994. Fast contact force computation for nonpenetrating rigid bodies. *Computer Graphics Proceedings, Annual Conference Series* 23-24, 23–34.

Baraff, D. 1996. Linear-time Dynamics Using Lagrange Multipliers. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. ACM, New York, NY, USA, 137–146. https://doi.org/10.1145/237170.237226

Baraff, D. and Witkin, A. 1997. *Partitioned dynamics*. Technical Report. Carnegie-Mellon University Robotics Institute.

Barbič, J. and Zhao, Y. 2011. Real-time Large-deformation Substructuring. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11)*. ACM, New York, NY, USA, Article 91, 8 pages. https://doi.org/10.1145/1964921.1964986

Bender, J., Erleben, K., and Trinkle, J. 2014. Interactive Simulation of Rigid Body Dynamics in Computer Graphics. *Computer Graphics Forum* 33, 1, 246–270. https://doi.org/10.1111/cgf.12272

Chu, J., Zafar, N. B., and Yang, X. 2017. A Schur Complement Preconditioner for Scalable Parallel Fluid Simulation. *ACM Trans. Graph.* 36, 5, Article 163, 11 pages. https://doi.org/10.1145/3092818

Cline, M. B. and Pai, D. K. 2003. Post-stabilization for rigid body simulation with contact and constraints. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, Vol. 3. 3744–3751 vol.3. https://doi.org/10.1109/ROBOT.2003.1242171

CM Labs Simulations. 2017. *Theory Guide: Vortex Software's Multibody Dynamics Engine*. Technical Report. https://www.cm-labs.com/vortexstudiodocumentation/Vortex_User_Documentation/Content/Concepts/theoryguide.html

Critchley, J. H., Anderson, K. S., and Binani, A. 2009. An Efficient Multibody Divide and Conquer Algorithm and Implementation. *ASME Journal of Computational and Nonlinear Dynamics* 4, 021004–1–021004–10.

Cuthill, E. and McKee, J. 1969. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*. ACM, 157–172.

Deul, C., Kugelstadt, T., Weiler, M., and Bender, J. 2018. Direct Position-Based Solver for Stiff Rods. *Computer Graphics Forum* 37, 6, 313–324. https://doi.org/10.1111/cgf.13326

Enzenhöfer, A., Andrews, S., Teichmann, M., and Kövecses, J. 2018. Comparison of Mixed Linear Complementarity Problem Solvers for Multibody Simulations with Contact. In *Workshop on Virtual Reality Interaction and Physical Simulation*. The Eurographics Association. https://doi.org/10.2312/vriphys.20181063

Enzenhöfer, A., Lefebvre, N., and Andrews, S. 2019. Efficient block pivoting for multi-body simulations with contact. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D'19)*. ACM.

Erleben, K. 2007. Velocity-based Shock Propagation for Multibody Dynamics Animation. *ACM Trans. Graph.* 26, 2, Article 12, 20 pages. https://doi.org/10.1145/1243980.1243986

Featherstone, R. 1999. A Divide-and-Conquer Articulated-Body Algorithm for Parallel O(log(n)) Calculation of Rigid-Body Dynamics. Part 1 and 2. *The International Journal of Robotics Research* 18, 9, 867–892.

Fratarcangeli, M. and Pellacini, F. 2015. Scalable Partitioning for Parallel Position Based Dynamics. *Computer Graphics Forum* 34, 2, 405–413. https://doi.org/10.1111/cgf.12570

Fratarcangeli, M., Tibaldo, V., and Pellacini, F. 2016. Vivace: A Practical Gauss-seidel Method for Stable Soft Body Dynamics. *ACM Trans. Graph.* 35, 6, Article 214, 9 pages. https://doi.org/10.1145/2980179.2982437

Glocker, C. 2001. *Set-Valued Force Laws, Dynamics of Non-Smooth Systems*. Springer-Verlag, Berlin, Heidelberg.

Guendelman, E., Bridson, R., and Fedkiw, R. 2003. Nonconvex Rigid Bodies with Stacking. *ACM Trans. Graph.* 22, 3, 871–878. https://doi.org/10.1145/882262.882358

Intel. 2003–2019. Intel Math Kernel Library. https://software.intel.com/en-us/mkl.

Júdice, J. J. and Pires, F. M. 1994. A block principal pivoting algorithm for large-scale strictly monotone linear complementarity problems. *Computers & operations research* 21, 5, 587–596.

Kang, H. C., Kim, S.-S., and Lee, C.-H. 2015. Parallel processing with the subsystem synthesis method for efficient vehicle analysis. *Journal of Mechanical Science and Technology* 29, 7, 2663–2669.

Karypis, G. and Kumar, V. 1999. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing* 20, 1, 359–392.

Kaufman, D. M., Sueda, S., James, D. L., and Pai, D. K. 2008. Staggered Projections for Frictional Contact in Multibody Systems. *ACM Trans. Graph.* 27, 5, Article 164, 11 pages. https://doi.org/10.1145/1409060.1409117

Kim, S.-S. 2002. A Subsystem Synthesis Method for Efficient Vehicle Multibody Dynamics. *Multibody System Dynamics* 7, 189–207.

Kim, T. and James, D. L. 2011. Physics-based Character Skinning Using Multi-domain Subspace Deformations. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '11)*. ACM, New York, NY, USA, 63–72. https://doi.org/10.1145/2019406.2019415

Lacoursière, C. 2007. A Parallel Block Iterative Method for Interactive Contacting Rigid Multibody Simulations on Multicore PCs. In *Applied Parallel Computing. State of the Art in Scientific Computing*, Bo Kågström, Erik Elmroth, Jack Dongarra, and Jerzy Waśniewski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 956–965.

Laflin, J. J., Anderson, K. S., Khan, I. M., and Poursina, M. 2014. Advances in the Application of the Divide-and-Conquer Algorithm to Multibody System Dynamics. *ASME Journal of Computational and Nonlinear Dynamics* 9, 041003–1–041003–8.

Liu, H., Mitchell, N., Aanjaneya, M., and Sifakis, E. 2016. A Scalable Schur-complement Fluids Solver for Heterogeneous Compute Platforms. *ACM Trans. Graph.* 35, 6, Article 201, 12 pages. https://doi.org/10.1145/2980179.2982430

Mazhar, H., Heyn, T., Negrut, D., and Tasora, A. 2015. Using Nesterov's Method to Accelerate Multibody Dynamics with Friction and Contact. *ACM Trans. Graph.* 34, 3, Article 32, 14 pages. https://doi.org/10.1145/2735627

Moreau, J. J. 1966. Quadratic Programming in Mechanics: Dynamics of One-Sided Constraints. *SIAM Journal on Control and Optimization* 4, 1, 153–158.

Moreau, J. J. 1988. Unilateral Contact and Dry Friction in Finite Freedom Dynamics. *Nonsmooth Mechanics and Applications* 302, 1–82.

Müller, M., Chentanez, N., Macklin, M., and Jeschke, S. 2017. Long Range Constraints for Rigid Body Simulations. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '17)*. ACM, New York, NY, USA, Article 14, 10 pages. https://doi.org/10.1145/3099564.3099574

Narain, R., Samii, A., and O'Brien, J. F. 2012. Adaptive Anisotropic Remeshing for Cloth Simulation. *ACM Trans. Graph.* 31, 6, Article 152, 10 pages. https://doi.org/10.1145/2366145.2366171

O'Leary, D. P. 1980. A generalized conjugate gradient algorithm for solving a class of quadratic programming problems. *Linear Algebra Appl.* 34, 371–399.

Parker, E. G. and O'Brien, J. F. 2009. Real-time Deformation and Fracture in a Game Environment. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '09)*. ACM, New York, NY, USA, 165–175. https://doi.org/10.1145/1599470.1599492

Pellegrini, F. and Roman, J. 1996. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*. Springer, 493–498.

Pfeiffer, F., Foerg, M., and Ulbrich, H. 2006. Numerical aspects of non-smooth multibody dynamics. *Computer Methods in Applied Mechanics and Engineering* 195, 50-51, 6891–6908.

Silcowitz, M., Niebe, S., and Erleben, K. 2011. Interactive Rigid Body Dynamics Using a Projected Gauss–Seidel Subspace Minimization Method. In *Computer Vision, Imaging and Computer Graphics. Theory and Applications*, Paul Richard and José Braz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 218–229.

Stewart, D. E. and Trinkle, J. C. 1996. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction. *Internat. J. Numer. Methods Engrg.* 39, 15, 2673–2691.

Tomcin, R., Sibbing, D., and Kobbelt, L. 2014. Efficient enforcement of hard articulation constraints in the presence of closed loops and contacts. *Computer Graphics Forum* 33, 2, 235–244. https://doi.org/10.1111/cgf.12322

Tonge, R., Benevolenski, F., and Voroshilov, A. 2012. Mass Splitting for Jitter-Free Parallel Rigid Body Simulation. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2012* 31, 4, 1–8.

Torres-Moreno, J.-L., Blanco, J.-L., López-Martínez, J., and Giménez-Fernández, A. 2013. A comparison of Algorithms for Sparse Matrix Factoring and Variable Reordering aimed at Real-time Multibody Dynamic Simulation. In *ECCOMAS Multibody Dynamics*. 167–168.

Tournier, M., Nesme, M., Gilles, B., and Faure, F. 2015. Stable Constrained Dynamics. *ACM Trans. Graph.* 34, 4, Article 132, 10 pages. https://doi.org/10.1145/2766969

Visseq, V., Martin, A., Iceta, D., Azéma, E., Dureisseix, D., and Alart, P. 2012. Dense granular dynamics analysis by a domain decomposition approach. *Computational Mechanics* 49, 6, 709–723.