High-performance CPU Cloth Simulation Using Domain-decomposed Projective Dynamics

ZIXUAN LU*, University of Utah, USA
ZIHENG LIU*†, University of Utah, USA
LEI LAN, University of Utah, USA
HUAMIN WANG, Style3D Research, China
YUKO ISHIWAKA, SoftBank, Japan
CHENFANFU JIANG, UCLA, USA
KUI WU, LIGHTSPEED, USA
YIN YANG, University of Utah, USA

Fig. 1. **Efficient multi-domain cloth simulation on CPU.** This paper introduces a CPU-based high-performance cloth simulation framework based on domain decomposition. The core of our algorithm is a parallel scheme that fits the hardware architecture of the multicore CPU. Unlike existing GPU algorithms, CPU parallelization should focus more on convergence as the total number of available cores is limited. We show how this high-level idea is integrated with the projective dynamics pipeline at both local and global stages. Our method is able to deliver good runtime performance that is comparable to the state-of-the-art GPU counterparts and high-quality animations. The teaser figure highlights some results of our algorithm, featuring a virtual character dressed in various garments performing a diverse range of actions, including handstands, tossing, dancing, and walking. Those garment models are of high resolution, and they are divided into domains as visualized in the figure. When the time step size is set $h = 1/120$, our method only uses 300 - 400 ms to simulate one frame, which is more than one order faster than existing CPU simulators.

Whenever the concept of high-performance cloth simulation is brought up, GPU acceleration is almost always the first that comes to mind. Leveraging immense parallelization, GPU algorithms have demonstrated significant success recently, whereas CPU methods are somewhat overlooked. Indeed, the need for an efficient CPU simulator is evident and pressing. In many scenarios, high-end GPUs may be unavailable or are already allocated to other tasks, such as rendering and shading. A high-performance CPU alternative can greatly boost the overall system capability and user experience. Inspired by this demand, this paper proposes a CPU algorithm for high-resolution cloth simulation. By partitioning the garment model into multiple (but not massive) sub-meshes or domains, we assign per-domain computations to individual CPU processors. Borrowing the idea of projective dynamics that breaks the computation into global and local steps, our key contribution is a new parallelization paradigm at domains for both global and local steps so that domain-level calculations are sequential and lightweight. The CPU has much fewer processing units than a GPU. Our algorithm mitigates this disadvantage by wisely balancing the scale of the parallelization and convergence. We validate our method in a wide range of simulation problems involving high-resolution garment models. Performance-wise, our method is at least one order faster than existing CPU methods, and it delivers a similar performance compared with the state-of-the-art GPU algorithms in many examples, but without using a GPU.

*joint first authors
†Part of this work was done when Ziheng Liu was an intern at LIGHTSPEED.

Authors' Contact Information: Zixuan Lu, University of Utah, USA, birdpeople1984@gmail.com; Ziheng Liu, University of Utah, USA, ziheng.liu@utah.edu; Lei Lan, University of Utah, USA, lanlei.virhum@gmail.com; Huamin Wang, Style3D Research, China, wanghmin@gmail.com; Yuko Ishiwaka, SoftBank, Japan, yuko.ishiwaka@g.softbank.co.jp; Chenfanfu Jiang, UCLA, USA, chenfanfu.jiang@gmail.com; Kui Wu, LIGHTSPEED, USA, walker.kui.wu@gmail.com; Yin Yang, University of Utah, USA, yangzzzy@gmail.com.

ACM 1557-7368/2025/8-ART51
https://doi.org/10.1145/3731182

CCS Concepts: • **Computing methodologies → Physical simulation**.

Additional Key Words and Phrases: Cloth simulation, Parallel computation, GPU algorithm, CPU algorithm

ACM Reference Format:
Zixuan Lu, Ziheng Liu, Lei Lan, Huamin Wang, Yuko Ishiwaka, Chenfanfu Jiang, Kui Wu, and Yin Yang. 2025. High-performance CPU Cloth Simulation Using Domain-decomposed Projective Dynamics. *ACM Trans. Graph.* 44, 4, Article 51 (August 2025), 17 pages. https://doi.org/10.1145/3731182

1 Introduction

Cloth simulation stands as a cornerstone of realism for digital content generation [House and Breen 2000], making garments flow with smoothness, dresses cascade with elegance, and fabrics fold with precision. It remains a challenging problem for high-resolution garment models. This is because two-way coupled unknown degrees of freedom (DOFs) lead to a large-scale nonlinear system, and the computation complexity grows super-polynomially w.r.t. the DOF count. GPGPU, due to its excellent throughput and parallelizability, has become the mainstream solution for performance improvement [Bolz et al. 2003; Zeller 2005]. By solving unknown DOFs in parallel, recent GPU methods deliver remarkable results even for complex garment models [Lan et al. 2024; Li et al. 2023; Wu et al. 2022]. However, reliance on GPUs can introduce practical obstacles. GPUs are also the key hardware for other important computing tasks e.g., rendering and shading, for which they were originally designed. Moreover, many commodity computers may not have high-end GPUs capable of enabling meaningful performance improvements. On the other hand, efficient simulation using the CPU is a relatively unexplored problem. In contrast to GPUs, a CPU core excels in faster clock cycles, abundant high-speed cache, and more well-rounded ISA (instruction set architecture). Multicore CPUs also have a substantial potential for improving simulation performance. Nevertheless, one should never "copy-and-paste" an existing GPU algorithm for a CPU platform, whose parallelism architecture is better suited for handling fewer but more sophisticated computing tasks.

This paper proposes a cloth simulation framework that can be greatly accelerated on a multicore CPU. We opt for projective dynamics (PD) [Bouaziz et al. 2014] as the backbone for our pipeline. Being a variation of the quasi-Newton method [Liu et al. 2017], PD offers decent convergence and good potential for parallelization. In the meantime, we observe that the design/manufacture of clothing normally starts with patternmaking, after which various parts, such as the sleeves, front/back panels, hems etc., are sewn together. Such structure-wise composition of a garment model naturally matches the domain decomposition methods (DDMs) [Smith 1997]. Our method synergizes with those two ingredients algorithmically with an efficient CPU-based parallelization.

Concretely, during the global solve, we keep a small set of key DOFs two-way coupled while decoupling less essential DOFs at domains. This allows the domain-level parallelization of forward/back substitution. Our domain-decomposed PD global solver *exactly* solves the global matrix, and it is one order faster than the off-the-shelf libraries like MKL [Wang et al. 2014] or Eigen [Guennebaud et al. 2010] using a multicore CPU. When the constraint set varies, e.g., due to collisions and self-collisions, we devise a compact dual formulation for relaxing colliding DOFs and exploit the domain-parallelized global solver as a strong pre-conditioner. The local stage of the vanilla PD is GPU-friendly as each constraint is processed independently. Such massive parallelization slows the overall convergence and becomes less effective on the CPU. Instead, we switch to a Gauss-Seidel (GS) projection strategy, making this operation sequential within the domain but parallelizable across domains. Our framework also includes a new CCD processing algorithm using Halley's method, which converges $10-15\%$ faster than Newton's method for solving cubic polynomials.

Previous work, such as PARFES [Fialko 2019], introduced parallel forward/backward substitution algorithms for linear FEM that could potentially accelerate PD global steps. However, our global step solving approach differs fundamentally from PARFES. PARFES performs block $\mathbf{LSL}^\top$ decomposition of sparse symmetric matrices and identifies parallelizable independent tasks through an elimination tree converted to a supernodal structure. Our method leverages geometric and topological properties inherent to the problem. Specifically, we permute degrees of freedom (DOFs) based on their types to create predictable sparse patterns in the factorization. This enables parallel forward/backward substitution for redundant DOFs without requiring the construction and analysis of an elimination tree, resulting in a more direct parallelization approach tailored to our specific problem structure.

We have tested our method in a wide range of complex garment simulation scenes, and our CPU-based simulation produces high-quality results with a strong runtime performance that matches the state-of-the-art GPU algorithms. A concrete example is shown in the teaser (Fig. 1), where the dressed virtual avatar performs several interesting motions with different garments. Our CPU simulator only needs a few hundred milliseconds to simulate each frame (with the time step size 1/120). This is more than an order faster than vanilla PD-based cloth simulation using multi-threading.

2 Related Work

Being one of the core problems of computer graphics and animation, there exists a vast number of excellent contributions on the topic of cloth simulation. This section briefly reviews a few representative studies that are most relevant to our work.

Cloth simulation. Given a piece of cloth model, a common practice is to discretize its geometry with a mass-spring network [Choi and Ko 2002; Liu et al. 2013; Provot et al. 1995] or a triangle mesh [Etzmuß et al. 2003; Volino et al. 2009]. Early techniques choose to use explicit integration with small time steps [Provot et al. 1995]. The stability is improved by switching to the implicit integration [Baraff and Witkin 1998], at the cost of assembling and solving the resulting linearized systems. Kim [2020] later explained the fundamental connection between the Baraff-Witkin formulation [Baraff and Witkin 1998] and the anisotropic finite element methods (FEM) [Bathe 2006].

Cloth dynamics concerns in-plane stretching and out-off-plane bending. A wide range of material models have been proven effective for capturing the in-plane resistance, including Kirchhoff-Love [Chen et al. 2018], corotational [Etzmuß et al. 2003], orthotropic formulations [Volino et al. 2009] or even data-driven materials [Wang

et al. 2011]. Handling inextensible fabrics necessitates additional nonlinear penalties, such as Neo-Hookean [Li et al. 2021; Lu et al. 2024] or spline-based models [Xu et al. 2015]. Strain-limiting techniques [Thomaszewski et al. 2009; Wang et al. 2010] provide an easy fix for strong length and area preservation. Bending behavior is typically formulated on hinge elements through dihedral angles between adjacent triangles [Bridson et al. 2005; Grinspun et al. 2003; Wang et al. 2023]. Leveraging in-plane inextensibility, Bergou et al. [2006] developed an efficient quadratic bending model based on mesh's mean curvature.

GPU-based simulation. The high computational cost of solving nonlinear systems arising from implicit integration has long been a major challenge in cloth simulation. A widely adopted strategy is to re-formulate the force equilibrium into its variational counterpart [Gast et al. 2015; Kharevych et al. 2006]. This formulation brings an optimization perspective, enabled a more efficient implicit integration through sophisticated optimization techniques, especially when handling nonlinear constraints. Apart from force-based method and its variational energy form, constraint-based methods formulate the equilibrium configuration with a set of predefined constraints. Within this framework, constraints can be addressed locally and inexactly through techniques like constraint projection [Goldenthal et al. 2007], offering computational advantages and flexibility in handling the simulation. For instance, position-based dynamics (PBD) [Macklin et al. 2016; Müller et al. 2007] projects the position of a set of points directly, respected to the constraint groups. Projective dynamics or PD [Bouaziz et al. 2014] presents a global and local alternation scheme to approximately solve the nonlinear quadratic system. PD quickly becomes a popular simulation modality because its local projections are trivially parallelizable. Instead of solving the global system exactly e.g., using Cholesky factorization, iterative linear solvers can be used, such as Jacobi [Lan et al. 2024, 2022; Wang 2015], GS [Fratarcangeli et al. 2016] and preconditioned conjugate gradient (PCG) [Tang et al. 2013]. For more general and nonlinear models, sophisticated GPU algorithms can both accelerate the per-stencil computation and global system solving. In these methods, per-stencil computation is lightweight and independent and the linear solving can be accelerated through traditional GPU operators such as SpMV. However, such a massive parallelization is less suitble for CPU platform, given its distinct hardware architecture where each computation unit is equipped with sophisticated control logic and high computational capability, instead of sheer core quantities.

Multigrid & domain decomposition method. The multigrid method boosts simulation efficiency when a large number of DOFs are present [Bornemann and Deuflhard 1996; Trottenberg et al. 2001]. It has been extensively employed to solve Poisson systems in fluid simulation [McAdams et al. 2010; Molemaker et al. 2008]. While both incorporate meshes at different resolutions (geometrical meshes as input or algebraically logical meshes) to hierarchically eliminate oscillations or smooth errors, the geometric multigrid (GMG) [Georgii and Westermann 2006] utilizes spatial discretization (e.g., meshes or uniform grids) of different resolutionsas input to construct the restriction and prolongation operator. Xian et al. [2019] used a sparse sampling scheme to sparsify the coarse level matrices that effectively addresses the matrix density problem in Galerkin multigrid methods. The algebraic multigrid (AMG), on the other hand, approaches the construction by generating a subspace of the low-frequency dynamics regardless of real hierarchical geometry inputs, which shares a similar nature of model reduction [O'Brien et al. 2003; Pentland and Williams 1989]. For example, Li et al. [2023] used a B-spline subspace, and Tamstorf et al. [2015] built the subspace by QR decomposition on near-kernel components. Nonlinear multigrid is also an effective method e.g., as in [Wang et al. 2018], which updates the residue and system matrix periodically to incorporate the nonlinearity. Zhang et al. [2022] developed a progressive simulation method, which calculates high-resolution cloth deformation given input coarse poses. This method was later generalized to dynamic shell and cloth simulations [Zhang et al. 2024].

The domain decomposition method or DDM is another closely related method [Toselli and Widlund 2006]. Similar to multigrid, DDM aims to handle very large-scale simulation problems for HPC [Yamazaki et al. 2014]. In graphics, DDM is often used with reduced-order models to enrich local dynamics. Barbič and Zhao [2011] designed a substructuring algorithm assuming the interfaces among domains are small and nearly rigid. It is particularly effective for plant simulation [Zhao and Barbič 2013]. Yang et al. [2013] combined modal warping [Choi and Ko 2005] and component mode synthesis (CMS) [MacNeal 1971] to build local subspaces from the interface deformation. Kim and James [2011] coupled domains with springs to avoid inter-domain locking. Wu et al. [2015] also utilized a spring-based domain coupling with Cubature sampling [An et al. 2008]. Recently, Li et al. [2019] developed a domain decomposition method that achieves subdomain coupling through quadratic penalty potentials at interfaces, eliminating the requirement for dual variables in formulation. Peiret et al. [2019] introduces a novel Schur complement-based substructuring approach for efficiently simulating stiff multibody systems with contact. A key ingredient in DDM is the coupling mechanism among domains. Classic DDM is designed for large-scale FEM, where direct solvers should not be used. When decomposing the mesh into domains, interface DOFs are duplicated, leading to finite element tearing and interconnecting (FETI) method [Farhat and Roux 1991]. Many variations have been proposed, aiming to improve the convergence of the linear system e.g., see [Farhat et al. 2000]. FETI-DP [Farhat et al. 2001] is a PCG based DDM, which solves a coarse problem as the pre-conditioner of multi-domain PCG iterations.

We argue that DDM naturally fits the multicore CPU. As a co-dimensional problem, cloth domain partition tends to yield fewer boundary DOFs, which eases the coupling issue. Garments are often designed and fabricated via patches, meaning they have been decomposed into domains already. On the downside, cloth dynamics is highly nonlinear, and existing DDMs are not applicable directly. In this paper, we show how to integrate PD with DDM to enable highly efficient cloth simulation on the CPU.

3 Vanilla Projective Dynamics

Given a time integration scheme such as implicit Euler, PD aims to solve a variational optimization for each time step:

$$\arg\min_{\boldsymbol{x}} E = I(\boldsymbol{x}, \dot{\boldsymbol{x}}) + \Psi(\boldsymbol{x}),\ I = \frac{1}{2h^2}\|\boldsymbol{M}^{\frac{1}{2}}(\boldsymbol{x} - \boldsymbol{z})\|^2. \tag{1}$$

Here, $\boldsymbol{x}$ is the unknown variable we need to compute for the next time step i.e., the position of all the cloth vertices. $\boldsymbol{z} = \boldsymbol{x}^* + h\dot{\boldsymbol{x}}^* + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_{ext}$ is a known vector depending on the previous position $\boldsymbol{x}^*$, velocity $\dot{\boldsymbol{x}}^*$, and an external force $\boldsymbol{f}_{ext}$. $\boldsymbol{M}$ is the mass matrix, and h is the time step size. The objective function E consists of the inertia potential (I) penalizing accelerated movements, and the elasticity potential (Ψ) characterizing the deformation of the cloth.

PD splits the optimization of Eq. (1) into two steps, namely the local step and the global step. For the i-th constraint C_i, the local step is in the form of:

$$\arg\min_{\boldsymbol{y}_i} \frac{w_i}{2}\|\boldsymbol{A}_i\boldsymbol{S}_i\boldsymbol{x} - \boldsymbol{B}_i\boldsymbol{y}_i\|^2, \text{ s.t. } C_i(\boldsymbol{y}_i) = 0. \tag{2}$$

Here, $\boldsymbol{S}_i$ is a selection matrix picking DOFs pertaining to C_i from $\boldsymbol{x}$ i.e., $\boldsymbol{x}_i = \boldsymbol{S}_i\boldsymbol{x}$. $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ map the positional DOFs of $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ to the specific coordinate that the constraint C_i measures. $\boldsymbol{y}_i$ refers to the so-called *target position* of C_i — the position closest to the current value of $\boldsymbol{x}_i$ that keeps C_i satisfied. Intuitively, the local step aims to lower Ψ in a Jacobi-like manner. The target position is essentially the local optimum at C_i for minimizing Ψ.

The global step is a standard linear solve in the form of $\boldsymbol{K}\boldsymbol{x} = \boldsymbol{b}$:

$$\underbrace{\left(\frac{\boldsymbol{M}}{h^2} + \sum_i w_i\boldsymbol{S}_i^\top\boldsymbol{A}_i^\top\boldsymbol{A}_i\boldsymbol{S}_i\right)}_{\boldsymbol{K}}\boldsymbol{x} = \underbrace{\frac{\boldsymbol{M}}{h^2}\boldsymbol{z} + \sum_i w_i\boldsymbol{S}_i^\top\boldsymbol{A}_i^\top\boldsymbol{B}_i\boldsymbol{y}_i}_{\boldsymbol{b}}. \tag{3}$$

Eq. (3) relaxes the inertia potential I while averaging duplicated DOFs $\boldsymbol{y}_i$ to generate a global solution of $\boldsymbol{x}$ since a DOF could have multiple replicates if it is involved in multiple constraints. PD takes several alternations between the local step and the global step, and we refer to each full cycle of local and global steps a L-G iteration.

4 Domain-decomposed Global Solve

PD is often considered a fast simulation algorithm because 1) the local step is parallelizable, and 2) if the constraint set does not change, the global matrix becomes constant and can be pre-factorized. However, these advantages are significantly diminished on a CPU platform, and we need a re-designed scheme, which deploys the parallelization at the domain level.

We first set aside collision constraints and assume the global matrix $\boldsymbol{K}$ is constant. This assumption allows us to pre-factorize $\boldsymbol{K}$ e.g., using Cholesky decomposition as $\boldsymbol{K} = \boldsymbol{L}\boldsymbol{L}^\top$, and Eq. (3) can then be solved with one forward substitution and one backward substitution of $\boldsymbol{L}$. Substitution is costly with the complexity of $O(N^2)$, where N is the size of the global matrix. More importantly, forward/backward substitution is inherently sequential, and multiple CPU cores hardly help. Therefore, even with $\boldsymbol{K}$ being constant, global solve remains the bottleneck of the pipeline. By decomposing Eq. (3) into multiple sub-systems or *domains*, we show that it is possible to parallelize the forward/backward substitution in a primal-dual manner. This parallel global solve lays the foundation of our CPU-based simulation method.

4.1 DOF Classification

We start with a subdivision of an input cloth mesh. Each sub-mesh forms a domain, which consists of a set of edge-connected triangles, as shown in Fig. 2. For each domain, we can assemble the counterpart of Eq. (3) such that: $\boldsymbol{K}^j\boldsymbol{x}^j = \boldsymbol{b}^j$. The superscript j suggests the domain index.

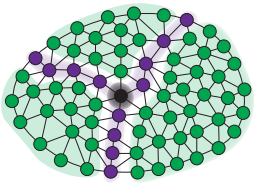

Fig. 2. **DOF types.** We categorize vertices as internal vertices (●) and boundary vertices (● + ●). The latter is further grouped into corner vertex (●) and duplicate vertices (●). The superset of internal vertices and duplicate vertices are collectively named as remainder vertices (● + ●).

The decomposition naturally classifies domain vertices into *internal vertices* and *boundary vertices*. As the name suggests, an internal vertex is exclusively owned by a single domain, while a boundary vertex is shared among adjacent domains. Subscripts I and B are used to denote the corresponding vertex category i.e., $\boldsymbol{x}_I^j$ and $\boldsymbol{x}_B^j$ are DOF values of internal and boundary vertices. Let N_B be the total number of boundary DOFs of the whole mesh and N_B^j be the number of boundary DOFs of the j-th domain. It should be noted that $N_B < \sum_j N_B^j$ because boundary vertices are overcounted at domains.

Among all the boundary DOFs, we designate a sub-group of N_C DOFs as *corner DOFs*. By default, corner DOFs are those shared by more than two domains i.e., they are at corners. The user may also manually include more boundary DOFs as corners. The other boundary DOFs are named as *duplicate DOFs* — they are duplicated by two neighbor domains. In other words, $N_B^j = N_D^j + N_C^j$, where N_D^j and N_C^j stand for the numbers of duplicate and corner DOFs of domain j. All of the N_R^j non-corner DOFs of the domain are called *remainder DOFs*. It is easy to verify that $N^j = N_I^j + N_B^j = N_R^j + N_C^j$. We then re-write $\boldsymbol{K}^j\boldsymbol{x}^j = \boldsymbol{b}^j$ block-wisely as:

$$\begin{bmatrix}\boldsymbol{K}_{RR}^j & \boldsymbol{K}_{RC}^j \\ \boldsymbol{K}_{RC}^{j\top} & \boldsymbol{K}_{CC}^j\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_R^j \\ \boldsymbol{x}_C^j\end{bmatrix} = \begin{bmatrix}\boldsymbol{b}_R^j \\ \boldsymbol{b}_C^j\end{bmatrix}, \tag{4}$$

with subscripts R and C denoting the corresponding vertex category.

4.2 Primal-dual Formulation

The interface compatibility constraints require that boundary DOFs have the same values at different domains. We use different strategies to enforce this constraint for corner vertices and duplicate vertices. Specifically, we impose a set of equality constraints for the duplicate vertices such that:

$$\sum_j \pm\boldsymbol{S}_D^j\boldsymbol{x}_R^j = \boldsymbol{0}, \tag{5}$$

where $\boldsymbol{S}_D^j \in \mathbb{R}^{(N_B-N_C)\times N_R^j}$ is the selection matrix, which picks out the domain's duplicate DOFs $\boldsymbol{x}_D^j$, and re-indexes it globally for all the $N_B - N_C$ non-corner boundary DOFs. On the other hand, the interface constraint is implicitly enforced at corner DOFs using the minimal coordinate $\boldsymbol{x}_C$. That said, $\boldsymbol{x}_C$ contains all the N_C corner DOFs of the mesh *without duplication*, and we use another selection matrix $\boldsymbol{S}_C^j \in \mathbb{R}^{N_C^j\times N_C}$ to map between $\boldsymbol{x}_C^j$ and $\boldsymbol{x}_C$ such that $\boldsymbol{x}_C^j = \boldsymbol{S}_C^j\boldsymbol{x}_C$.

Next, we expand the first row of Eq. (4) at the domain's remainder DOFs and maintain the interface compatibility constraint by the

dual variable $\boldsymbol{\lambda}$ at duplicate DOFs. The residual at corner DOFs is summed over all the domains and solved collectively at $\boldsymbol{x}_C$. This leads to:

$$\begin{cases} \boldsymbol{K}^j_{RR}\boldsymbol{x}^j_R + \boldsymbol{K}^j_{RC}\boldsymbol{S}^j_C\boldsymbol{x}_C = \boldsymbol{b}^j_R - \boldsymbol{S}^{j^\top}_D\boldsymbol{\lambda}, \\ \sum_j \boldsymbol{S}^{j^\top}_C\boldsymbol{K}^j_{RC}\boldsymbol{x}^j_R + \sum_j \boldsymbol{S}^{j^\top}_C\boldsymbol{K}^j_{CC}\boldsymbol{S}^j_C\boldsymbol{x}_C = \sum_j \boldsymbol{B}^j_C\boldsymbol{b}^j_C = \boldsymbol{b}_C. \end{cases} \tag{6}$$

Here, $\boldsymbol{\lambda} \in \mathbb{R}^{N_B - N_C}$ is the globally-indexed constraint force, i.e., the Lagrange multiplier corresponding to the interface constraint of Eq. (5).

From the first line of Eq. (6) we have:

$$\boldsymbol{x}^j_R = \boldsymbol{K}^{j^{-1}}_{RR}\left(\boldsymbol{b}^j_R - \boldsymbol{S}^{j^\top}_D\boldsymbol{\lambda} - \boldsymbol{K}^j_{RC}\boldsymbol{S}^j_C\boldsymbol{x}_C\right), \tag{7}$$

which implies $\boldsymbol{x}^j_R$ can now be calculated in parallel at domains as long as we can compute the dual variable $\boldsymbol{\lambda}$. To this end, we substitute Eq. (7) back to Eq. (5) to build the primal-dual version of the global system:

$$\begin{bmatrix} \boldsymbol{G}_{RR} & \boldsymbol{G}_{RC} \\ \boldsymbol{G}^\top_{RC} & -\boldsymbol{G}_{CC} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{x}_C \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}^*_R \\ -\boldsymbol{b}^*_C \end{bmatrix}, \tag{8}$$

where

$$\boldsymbol{G}_{RR} = \sum_j \boldsymbol{S}^j_D\boldsymbol{K}^{j^{-1}}_{RR}\boldsymbol{S}^{j^\top}_D \in \mathbb{R}^{(N_B-N_C)\times(N_B-N_C)},$$

$$\boldsymbol{G}_{RC} = \sum_j \boldsymbol{S}^j_D\boldsymbol{K}^{j^{-1}}_{RR}\boldsymbol{K}^j_{RC}\boldsymbol{S}^j_C \in \mathbb{R}^{(N_B-N_C)\times N_C},$$

$$\boldsymbol{G}_{CC} = \sum_j \boldsymbol{S}^{j^\top}_C\boldsymbol{K}^j_{CC}\boldsymbol{S}^j_C - \sum_j \boldsymbol{S}^{j^\top}_C\boldsymbol{K}^{j^\top}_{RC}\boldsymbol{K}^{j^{-1}}_{RR}\boldsymbol{K}^j_{RC}\boldsymbol{S}^j_C \in \mathbb{R}^{N_C\times N_C},$$

$$\boldsymbol{b}^*_R = \sum_j \boldsymbol{S}^j_D\boldsymbol{K}^{j^{-1}}_{RR}\boldsymbol{b}^j_R \in \mathbb{R}^{N_B-N_C},$$

$$\boldsymbol{b}^*_C = \boldsymbol{b}_C - \sum_j \boldsymbol{S}^{j^\top}_C\boldsymbol{K}^{j^\top}_{RC}\boldsymbol{K}^{j^{-1}}_{RR}\boldsymbol{b}^j_R \in \mathbb{R}^{N_C}.$$

Expanding the first row of Eq. (8) allows us to solve $\boldsymbol{\lambda}$ via:

$$\boldsymbol{\lambda} = \boldsymbol{G}^{-1}_{RR}\left(\boldsymbol{b}^*_R - \boldsymbol{G}_{RC}\boldsymbol{x}_C\right). \tag{9}$$

The r.h.s. of Eq. (9) needs the information of $\boldsymbol{x}_C$, which can be solved by substituting Eq. (9) back into the second line of Eq. (8):

$$\boldsymbol{G}^*_{CC}\boldsymbol{x}_C = \boldsymbol{b}^*_C - \boldsymbol{G}^\top_{RC}\boldsymbol{G}^{-1}_{RR}\boldsymbol{b}^*_R, \text{ for } \boldsymbol{G}^*_{CC} = \boldsymbol{G}_{CC} - \boldsymbol{G}^\top_{RC}\boldsymbol{G}_{RR}\boldsymbol{G}_{RC}. \tag{10}$$

After obtaining the corner DOFs $\boldsymbol{x}_C$, and the dual variable $\boldsymbol{\lambda}$, the remainder DOFs at each domain can then be solved in parallel.

The computational procedure of our domain decomposed global solve is outlined in Fig. 3, which includes three major steps of computing corner DOFs $\boldsymbol{x}_C$, dual variables $\boldsymbol{\lambda}$, and the remainder DOFs at each domain $\boldsymbol{x}^j_R$ respectively. As $\boldsymbol{K}$ is assumed constant at this point, all the matrices can be pre-assembled and pre-factorized (as colored in red in the figure). Instead of performing global forward/backward substitution of $\boldsymbol{K}^{-1}$, our formulation only needs forward/backward substitutions at domains' remainder DOFs i.e., $\boldsymbol{K}^{j^{-1}}_{RR}$, which are processed in parallel. In addition, we need to solve $\boldsymbol{G}^*_{CC}$ and $\boldsymbol{G}_{RR}$ for corner DOFs and the multiplier. They correspond to two linear systems of $N_C \times N_C$ and $(N_B - N_C) \times (N_B - N_C)$. Fortunately, it is reasonable to assume that N_C is a small quantity, and solving $\boldsymbol{x}_C$ using pre-factorized $\boldsymbol{G}^*_{CC}$ is efficient. Suppose there are D domains

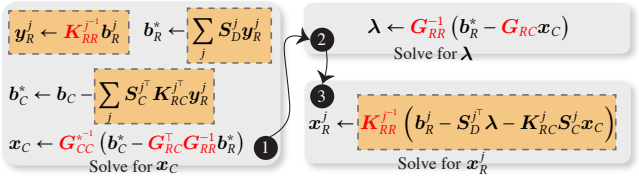

Fig. 3. **Domain-decomposed global solve.** Our primal-dual global system consists of three steps, namely solving for $\boldsymbol{x}_C$, $\boldsymbol{\lambda}$, and $\boldsymbol{x}^j_R$. This algorithm allows most forward/backward substitution to be carried out in parallel at domains' remainder DOFs, highlighted with light orange boxes. As a result, the performance of global solve can be accelerated linearly w.r.t. the number of CPU cores.

of roughly the same size. We have the DOFs of domain j as $N^j_D \approx \frac{N}{D}$, which suggests the time complexity of the domain-decomposed global solve is $O\left(D \cdot \frac{N^2}{D^2} + \max\left(N^2_B, N^2_C\right)\right)$.

Discussion. Strictly speaking, Eq. (8) is not a conventional primal-dual version of the original PD global problem. Firstly, the multiplier is only activated at duplicate DOFs. Corner DOFs, because they are globally indexed as $\boldsymbol{x}_C$, do not generate interface constraints. This strategy allows a straightforward formulation of $\boldsymbol{\lambda}$ because duplicate DOFs are shared by exactly two domains, and we do not need extra safeguards for the vertex shared by more than two domains to make sure dual DOFs are linearly independent. The primal variable $\boldsymbol{x}_I$, on the other hand, is condensed to corner DOFs. The condensation reduces the size of the primal part of the global system from N to N_C and makes $\boldsymbol{G}$ a dense matrix.

4.3 Domain Decomposition

The performance of our method is closely relevant to how domains are decomposed. Intuitively, we would like to make each $\boldsymbol{K}^j_{RR} \in \mathbb{R}^{N^j_R \times N^j_R}$ as small as possible so that the complexity of solving $\boldsymbol{x}^j_R$ can be effectively suppressed. Load balance is another important aspect. Since solving $\boldsymbol{x}^j_R$ is parallelized at domains, it is preferred that domains are of similar sizes so that the computation at each thread completes roughly at the same time.

We also need to solve $\boldsymbol{x}_C$ and $\boldsymbol{\lambda}$, which must be processed one after the other. With pre-factorized $\boldsymbol{G}^*_{CC}$ and $\boldsymbol{G}_{RR}$, the performance depends on N_C and $N_B - N_C$. Unfortunately, reducing N^j_R and reducing N_B are a pair of conflicting objectives — smaller domains have more boundary DOFs and fewer internal DOFs. Therefore, it is helpful to manually pick more corner vertices to lower $N_B - N_C$.

In our implementation, we utilize the graph partitioning tool METIS library [Karypis and Kumar 1997] to decompose the input garment mesh. METIS includes a multilevel graph cut algorithm minimizing boundary vertice count while balancing domain sizes. It allows explicit control over the number of domains and the maximum allowable ratio between the largest and smallest domain sizes.

4.4 Non-conforming Decomposition

As shown in Fig. 4, many digital garment models are not formed as a single monolithic triangle mesh. Instead, they are constructed by piecing together from multiple patches. The boundaries of those

patches are often not conforming. Practically, they are seamed via barycentric interpolations on interfacing edges or triangles.

Our method can be conveniently generalized to tackle non-conforming domains. An example is shown in Fig. 5 (a), where two domains are coupled via a non-conforming interface. The interface compatibility constraints apply to three orange and four green vertices — nonconforming domains lead to mismatching $\boldsymbol{x}_D^j$. We solve this ambiguity by always applying the constraint to the domain with fewer duplicate DOFs i.e., the orange domain in this exmaple. This strategy avoids potential overconstraining so that $\boldsymbol{G}_{RR}$ is well-conditioned. $\boldsymbol{S}_D^j$ becomes an interpolation matrix for the green domain — at the corresponding rows, $\boldsymbol{S}_D^j$ contains barycentric coordinates of local vertices for the neighboring duplicated vertices, which are being constrained. In this example, we have a nine-dimension dual variable $\boldsymbol{\lambda}$.

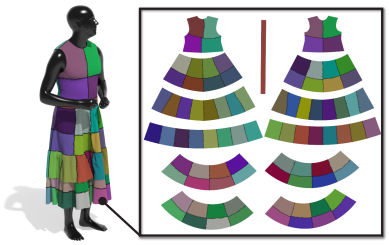

Fig. 4. **Non-conforming domain decomposition.** Our method can be generalized to handle non-conforming domains.

A more generic setup is illustrated in Fig. 5 (b), where multiple patches intersect. Unlike in conforming domain decomposition, the intersection point generally does not coincide with a mesh vertex, and the involving corner vertices are not full-rank. Therefore, $\boldsymbol{G}_{CC}$ in Eq. (8) becomes singular. The remedy is to further condense $\boldsymbol{x}_C$ to a set of linearly independent freedoms $\tilde{\boldsymbol{x}}_C$ i.e., the actual intersecting locations among multiple domains (plus other user-selected corner vertices and/or conforming corner vertices) such that $\tilde{\boldsymbol{x}}_C = \boldsymbol{B}_C\boldsymbol{x}_C$, where $\boldsymbol{B}_C$ the barycentric interpolation matrix. By projecting the second line of Eq. (8) into the column space of $\boldsymbol{B}_C$, the condensed primal variable $\tilde{\boldsymbol{x}}_C$ is computed via:

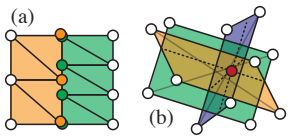

Fig. 5. **Non-conforming domains.** Our domain-decomposed global solver accommodates non-conforming domains. Duplicate DOFs (a) and corner DOFs (b) are processed by condensing the global step solve to a subset of independent DOFs to avoid overconstraining.

$$\tilde{\boldsymbol{x}}_C = \tilde{\boldsymbol{G}}_{CC}^{-1}\left(\tilde{\boldsymbol{b}}_C^* - \tilde{\boldsymbol{G}}_{RC}^\top \boldsymbol{G}_{RR}^{-1}\boldsymbol{b}_R^*\right), \tag{11}$$

where

$$\tilde{\boldsymbol{G}}_{CC} = \sum_j \boldsymbol{B}_C^{j\top}\boldsymbol{S}_C^{j\top}\boldsymbol{K}_{CC}^j\boldsymbol{S}_C^j\boldsymbol{B}_C^j - \sum_j \boldsymbol{B}_C^{j\top}\boldsymbol{S}_C^{j\top}\boldsymbol{K}_{RC}^{j\top}\boldsymbol{K}_{RR}^{j-1}\boldsymbol{K}_{RC}^j\boldsymbol{S}_C^j\boldsymbol{B}_C^j,$$

$$\tilde{\boldsymbol{G}}_{RC} = \sum_j \boldsymbol{S}_D^j\boldsymbol{K}_{RR}^{j-1}\boldsymbol{K}_{RC}^j\boldsymbol{S}_C^j\boldsymbol{B}_C^j,$$

$$\tilde{\boldsymbol{b}}_C^* = \boldsymbol{B}_C^\top\boldsymbol{b}_C - \sum_j \boldsymbol{B}_C^{j\top}\boldsymbol{S}_C^{j\top}\boldsymbol{K}_{RC}^{j\top}\boldsymbol{K}_{RR}^{j-1}\boldsymbol{b}_R^j.$$

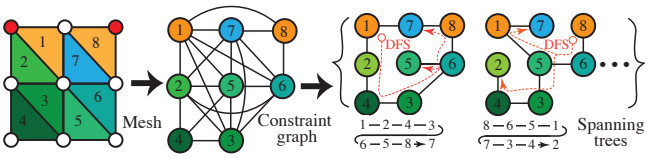

Fig. 6. **Sequential local projection.** We construct a constraint graph for each domain and extract several spanning trees. Our Gauss-Seidel local projection follows a DFS traversal of a spanning tree. DFS traversal always starts with a constraint with vertices whose positions are prescribed (e.g., fixed vertices), as highlighted in orange.

Here, $\boldsymbol{B}_C^j$ is the local interpolating matrix of the j-th domain. The dual variable $\boldsymbol{\lambda}$ and per-domain remainder variable $\boldsymbol{x}_R^j$ are solved afterwards.

5 Domain-decomposed Local Projection

The vanilla PD employs a Jacobi-like scheme at the local step to maximize the capacity of parallelization. This strategy is particularly well-suited for the GPUs as it handles a large number of constraints concurrently. Given the disparity in the number of processing cores, the CPU is clearly at a disadvantage. Being a CPU procedure, our method trades off some parallelizability to achieve better convergence.

Concretely, we downscale the local step parallelization from the constraint level to the domain level. Intuitively, this approach treats Ψ^j i.e., the elasticity potential of a domain, a type of *generalized constraint* and computes its target position sequentially in a Gauss-Seidel manner. The procedure of computing a constraint's target position remains unchanged per Eq. (2). Departing from the vanilla PD, after $\boldsymbol{y}_i$ is computed we further solve a small-size linear system:

$$\left(\frac{\boldsymbol{M}_i}{h^2} + \sum_i w_i\boldsymbol{A}_i^\top\boldsymbol{A}_i\right)\tilde{\boldsymbol{x}}_i = \frac{\boldsymbol{M}_i}{h^2}\boldsymbol{z}_i + \sum_i w_i\boldsymbol{A}_i^\top\boldsymbol{B}_i\boldsymbol{y}_i. \tag{12}$$

It is easy to see that Eq. (12) is simply a constraint-level global system that extracts columns and rows from Eq. (3) pertaining to constraint i. The l.h.s. is pre-factorized, and the solving is highly efficient given its low-rank nature. $\tilde{\boldsymbol{x}}_i$ offers a good estimation of the newly updated position without solving the full global system exactly. As we move forward to the next constraint $i+1$, vertices shared with constraint i will fetch the values from $\tilde{\boldsymbol{x}}_i$, which reflects the most updated local information, for calculating its own target position $\boldsymbol{y}_{i+1}$.

It is known that such a Gauss-Seidel-like relaxation scheme is biased toward a specific traversal order. We avoid this issue by pre-computing several paths of constraint iteration. As shown in Fig. 6, we construct a constraint graph of the domain to encode the topological connectivity of all the constraints. Each node on the graph represents a constraint involving several mesh vertices, i.e., a triangle in this example. Two constraints are connected by an edge if they share mesh vertices. Since we do not consider collision/self-collision at this point, the constraint graph can be pre-built. We then build several spanning trees of the graph. Ideally, any graph edge should be present in at least one of the spanning trees. A DFS (depth-first search) is performed for each spanning tree, and the

sequence of the node visits during the DFS traversal gives the order of our local projection. It is worth noting that the position of a vertex may be prescribed as an imposed boundary condition.

Taking Fig. 6 as an example, two red vertices of the top left and right are fixed. Our DFS always starts from a constraint (in orange) with such prescribed vertices (as they do not have dynamic freedoms) to track the strain propagation across the mesh. The per-domain local projection alternates among those pre-computed iteration paths. As shown in Fig. 7, this strategy effectively removes the bias of GS-based projection. After all the domain-level local projections are completed, the global step solve follows, and we move to the next L-G iteration.

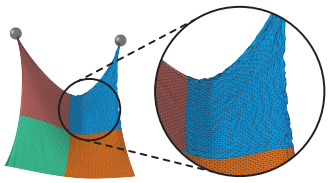

Fig. 7. **Biased residual with fixed GS relaxation.** GS iteration sequentially relaxes each constraint within a domain. If the order of the constraint transversal is fixed, GS tends produce patterned residual i.e., bias. We pre-compute multiple paths given the domain constraint graph and alternate GS iterations with different paths. This fully avoids the bias issue.

6 Collision-aware Global Solve

The discussion so far assumes the constraint set of the system does not change. This assumption does not hold in cloth simulation, where collisions and self-collisions are pervasive. Since PD handles collisions as a type of constraint, the global step matrix varies under different collision configurations. In this section, we show how to exploit our domain-decomposed pre-factorization to efficiently handle collision-in-the-loop global systems.

The presence of collisions and self-collisions alters Eq. (3) to:

$$(\boldsymbol{K} + \Delta\boldsymbol{K})(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b}. \tag{13}$$

That said, we explicitly label the change of global step matrix $\Delta\boldsymbol{K}$, vertex position adjustment $\Delta\boldsymbol{x}$, and the increment of r.h.s. vector $\Delta\boldsymbol{b}$ that are brought by the detected collision constraints. In practice, we often solve the collision-free global system $\boldsymbol{K}\boldsymbol{x} = \boldsymbol{b}$ first and use the corresponding $\boldsymbol{x}$ for collision detection. Therefore, the unknown we are looking to solve for is $\Delta\boldsymbol{x}$. Expanding Eq. (13) yields:

$$(\boldsymbol{K} + \Delta\boldsymbol{K})\Delta\boldsymbol{x} = \Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}. \tag{14}$$

We stack all the collision DOFs into a compact vector $\hat{\boldsymbol{x}} \in \mathbb{R}^K$, where K is the total number of collision DOFs. The notation $\hat{(\cdot)}$ suggests the variable is for collision DOFs. We also use the subscript k to specify the index of a collision constraint to differentiate it from other constraints, which are indexed with i.

Recall that the selection matrix $\boldsymbol{S}_k$ picks DOFs out of $\boldsymbol{x}$ for the k-th collision. We now split this operation into two steps — a collision selection matrix $\hat{\boldsymbol{S}} \in \mathbb{R}^{K\times N}$ first retrieves all the colliding DOFs i.e., $\hat{\boldsymbol{x}} = \hat{\boldsymbol{S}}\boldsymbol{x}$; after that $\hat{\boldsymbol{S}}_k$ extracts DOFs associated with the k-th collision from $\hat{\boldsymbol{x}}$. As $\hat{\boldsymbol{S}}$ is constant for all the collision constraints, this splitting reveals the structure of $\Delta\boldsymbol{K}$:

$$\Delta\boldsymbol{K} = \sum_k w_k \boldsymbol{S}_k^\top \boldsymbol{A}_k^\top \boldsymbol{A}_k \boldsymbol{S}_k = \sum_k w_k \hat{\boldsymbol{S}}^\top \hat{\boldsymbol{S}}_k^\top \boldsymbol{A}_k^\top \boldsymbol{A}_k \hat{\boldsymbol{S}}_k \hat{\boldsymbol{S}} = \hat{\boldsymbol{S}}^\top \Delta\hat{\boldsymbol{K}} \hat{\boldsymbol{S}}, \tag{15}$$

where we have:

$$\Delta\hat{\boldsymbol{K}} = \sum_k w_k \hat{\boldsymbol{S}}_k^\top \boldsymbol{A}_k^\top \boldsymbol{A}_k \hat{\boldsymbol{S}}_k \in \mathbb{R}^{K\times K}. \tag{16}$$

We follow the strategy as in [Lan et al. 2024] and set $\boldsymbol{A}_k$ as an identity matrix for collision constraints making $\Delta\hat{\boldsymbol{K}}$ diagonal.

Substituting Eq. (15) into Eq. (14) and applying the Woodbury identity [Hager 1989] yield:

$$\begin{aligned}\Delta\boldsymbol{x} = \left(\boldsymbol{K} + \hat{\boldsymbol{S}}^\top \Delta\hat{\boldsymbol{K}} \hat{\boldsymbol{S}}\right)^{-1} (\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}) = \boldsymbol{K}^{-1}(\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}) \\ - \boldsymbol{K}^{-1}\hat{\boldsymbol{S}}^\top \left(\Delta\hat{\boldsymbol{K}}^{-1} + \hat{\boldsymbol{S}}\boldsymbol{K}^{-1}\hat{\boldsymbol{S}}^\top\right)^{-1} \hat{\boldsymbol{S}}\boldsymbol{K}^{-1}(\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}).\end{aligned} \tag{17}$$

We then project Eq. (17) into the row space of $\hat{\boldsymbol{S}}$. Intuitively, doing so condenses the problem size and prioritizes the solving for colliding DOFs. Left-multiplying $\hat{\boldsymbol{S}}$ at its both sides of Eq. (17) with some manipulations results in a linear system of:

$$\hat{\boldsymbol{H}}\hat{\boldsymbol{\lambda}} = \hat{\boldsymbol{d}}, \tag{18}$$

where:

$$\begin{aligned}&\hat{\boldsymbol{H}} = \Delta\hat{\boldsymbol{H}} + \boldsymbol{I}, \quad \Delta\hat{\boldsymbol{H}} = \Delta\hat{\boldsymbol{K}}^{-1}\left(\hat{\boldsymbol{S}}\boldsymbol{K}^{-1}\hat{\boldsymbol{S}}^\top\right)^{-1}, \\ &\hat{\boldsymbol{d}} = \hat{\boldsymbol{S}}\boldsymbol{K}^{-1}(\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}), \quad \hat{\boldsymbol{\lambda}} = \hat{\boldsymbol{S}}\boldsymbol{K}^{-1}(\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}) - \hat{\boldsymbol{S}}\Delta\boldsymbol{x},\end{aligned}$$

and $\boldsymbol{I}$ is a K by K identity matrix.

Unlike other constraints, collisions embody a type of *hard constraint*. Hence, we always have $w_k \gg w_i$. It can be shown that

$$\left\|\Delta\hat{\boldsymbol{H}}\right\| \le \left\|\Delta\hat{\boldsymbol{K}}^{-1}\right\| \left\|\left(\hat{\boldsymbol{S}}\boldsymbol{K}^{-1}\hat{\boldsymbol{S}}^\top\right)^{-1}\right\| \le \frac{w_i}{w_k} \ll 1, \tag{19}$$

then we have:

$$0 < \epsilon = \left\|\Delta\hat{\boldsymbol{H}}\right\| \ll 1, \tag{20}$$

where $\epsilon := \|\Delta\hat{\boldsymbol{H}}\|$ is the spectral norm i.e., the maximum eigenvalue of $\Delta\hat{\boldsymbol{H}}$. Because

$$\left\|\hat{\boldsymbol{H}}\right\| = \left\|\Delta\hat{\boldsymbol{H}} + \boldsymbol{I}\right\| \le \left\|\Delta\hat{\boldsymbol{H}}\right\| + \|\boldsymbol{I}\| = \epsilon + 1,$$

we can then posit that the spectral radius of $\hat{\boldsymbol{H}}$ is bounded as:

$$1 < \rho(\hat{\boldsymbol{H}}) < \epsilon + 1 \approx \frac{w_i}{w_k} + 1. \tag{21}$$

In other words, $\rho(\widetilde{\boldsymbol{H}})$ is greater than but very close to one. This property inspires us to use the steepest descent (SD) method to solve Eq. (18) given the fact that the linear system can be solved with one SD iteration if its spectral radius equals one.

A reasonable initial guess of $\hat{\boldsymbol{\lambda}}$ for starting SD iteration is $\hat{\boldsymbol{\lambda}}_0 \leftarrow \hat{\boldsymbol{d}}$, which assumes $\Delta\hat{\boldsymbol{x}} = \boldsymbol{0}$. The corresponding residual $\hat{\boldsymbol{r}}$ of Eq. (18) then becomes:

$$\hat{\boldsymbol{r}} = \hat{\boldsymbol{d}} - \hat{\boldsymbol{H}}\hat{\boldsymbol{\lambda}}_0 = -\Delta\hat{\boldsymbol{K}}^{-1}\left(\hat{\boldsymbol{S}}\boldsymbol{K}^{-1}\hat{\boldsymbol{S}}^\top\right)^{-1} \hat{\boldsymbol{S}}\boldsymbol{K}^{-1}(\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}). \tag{22}$$

Even with our domain-wise pre-factorization, evaluating $\hat{\boldsymbol{r}}$ remains prohibitive. To avoid this difficulty, we ignore r.h.s DOFs for non-colliding vertices by replacing $\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}$ with $\hat{\boldsymbol{S}}^\top\hat{\boldsymbol{S}}(\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x})$. This strategy cancels out $(\hat{\boldsymbol{S}}\boldsymbol{K}^{-1}\hat{\boldsymbol{S}}^\top)^{-1}$, and the residual can be efficiently approximated as:

$$\hat{\boldsymbol{r}} \approx -\Delta\hat{\boldsymbol{K}}^{-1}\hat{\boldsymbol{S}}(\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}). \tag{23}$$

The SD iteration is then updated via:

$$\hat{\boldsymbol{\lambda}} \leftarrow \hat{\boldsymbol{\lambda}} + \frac{\hat{\boldsymbol{r}}^\top \hat{\boldsymbol{r}}}{\hat{\boldsymbol{r}}^\top \Delta\hat{\boldsymbol{H}} \hat{\boldsymbol{r}} + \hat{\boldsymbol{r}}^\top \hat{\boldsymbol{r}}} \cdot \hat{\boldsymbol{r}} \approx \hat{\boldsymbol{\lambda}} + \frac{\hat{\boldsymbol{r}}}{1 + \frac{w_i}{w_k}}, \tag{24}$$

by leveraging the spectral property of $\Delta\hat{\boldsymbol{H}}$ i.e., Eq. (21).

Multiplying $\hat{\boldsymbol{S}}^\top \hat{\boldsymbol{S}}$ to $\Delta\boldsymbol{b} - \Delta\boldsymbol{K}\boldsymbol{x}$ discards the influence from non-colliding DOFs to $\Delta\hat{\boldsymbol{x}}$. To recover this information, we finalize our collision-aware global solve with a couple of full PCG iterations over Eq. (14). At this point, it is expected that residual errors at collision DOFs $\Delta\hat{\boldsymbol{x}}$ are well relaxed with SD iterations for Eq. (18). Therefore, $\boldsymbol{K}^{-1}$ stands as a (very) strong pre-conditioner for CG iterations. While $\boldsymbol{K}$ is never factorized as a whole, our domain-decomposed global solver efficiently calculates $\boldsymbol{y} = \boldsymbol{K}^{-1}\boldsymbol{p}$ for any right vector $\boldsymbol{p}$ by solving $\boldsymbol{y} = \boldsymbol{K}\boldsymbol{p}$.

Discussion. In a nutshell, our collision-aware global step isolates the solve for $\Delta\boldsymbol{x}$ using the Woodbury formula. The key strategy is to build a dual version of this problem in a reduced space i.e., to solve $\hat{\boldsymbol{\lambda}}$ so that we can fully utilize the fact that collision constraints are much stiffer than other compliance constraints such as bending or edge length preserving. The unique structure of PD matrices helps extract important spectral information of $\hat{\boldsymbol{H}}$ so that the dual problem can be solved effectively with the SD method. The remaining residual error is smoothed with PCG using the domain-decomposed (collision-free) global matrix. We find that solving the reduced dual problem of Eq. (18) is critical to our performance gain. A representative user case is shown in Fig. 8. In this example, a piece of square tablecloth of 270K DOFs drapes over a teapot. The time step is $h = 1/120$. We examine the system convergence under conditions with the highest number of collisions ($\sim$ 70K). After five SD iterations, the relative error is lowered to 5%, which takes about 25 ms. Two extra PCG iterations are followed to converge the system. If we directly use PCG to solve Eq. (14) (with the pre-conditioner), we end up with over 20 iterations. In other words, several inexpensive SD iterations of Eq. (18) bring the total number of PCG iterations down from 20 to 5. The computation time is shortened by about 70%.

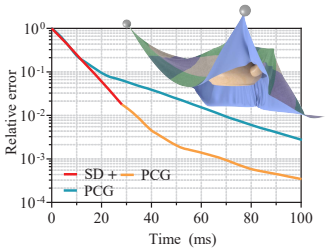

Fig. 8. **SD & PCG.** We drape a square tablecloth over the teapot, and plot the convergence curve w.r.t. computation time. There are 270k DOFs on the cloth, and 70K collision constraints. The time step size is set as 1/120. SD iterations for the dual problem of Eq. (24) effectively reduce the need for full PCG iterations at a very low cost. It saves about 70% computation time if one chooses to use PCG to solve the collision-aware global system directly.

7 Experiments

We have tested our method in a wide range of simulation cases. The experiments are performed with both AMD and Intel platforms with an AMD Ryzen Threadripper PRO 5975WX 32-core CPU, and an Intel i9-13900K 24-core CPU. We use CHOLMOD [Chen et al. 2008] on the AMD platform and MKL on the Intel platform. The CPU parallelization is handled with TBB library [Pheatt 2008]. Simulation parameters and timing information are reported in Tab. 1, and the visualization of the time breakdown is shown in Fig. 10. We also analyze the per frame time with respect to the number of active contact pairs for the given example (Fig. 11). Our method shows relatively stable performance (235 – 341 ms) across a range of contact pair quantities (65 – 118K pairs) in this case. Unless specified, our default time step size is 1/120 running with 32 threads on the 5975WX CPU. We normalize the garment into a unit box, and we use $\|\Delta\boldsymbol{x}\| = 1\text{e}{-3}$ as the default convergence condition. Please refer to the accompanying video for more animation results.

7.1 Multi-domain Global Solve

We first report a detailed study on the performance of the proposed multi-domain global solver when collision is not taken into account. While the global step matrix can be pre-factorized, the forward/backward substitution is a sequential operation, and multi-threading does not help improve the performance. Our method is able to fully exploit CPU cores and parallelize domain-level computations. Fig. 9 visualizes the timing statistics for solving the global matrix for a square tablecloth of different resolutions using different numbers of domains and threads. We run the test on both AMD (32 cores) and Intel (24 cores) platforms. Our baseline is single domain global solve using the pre-factorized global matrix. In general, the solving time is quickly lowered as more CPU cores are used. We use different color bars to visualize total times used for solving $\boldsymbol{G}^*_{CC}$ and $\boldsymbol{G}_{RR}$ for Eqs (9) and (10), which is not parallelizable (in orange) and for solving $\boldsymbol{K}^j_{RR}$ at domains (in blue). The time statistics is consistent with our previous analysis. With the increase in the number of participating cores, the total time used for solving $\boldsymbol{K}^j_{RR}$ decreases. On the other hand, the time used for solving $\boldsymbol{G}^*_{CC}$ and $\boldsymbol{G}_{RR}$ remains stable as those two computations are sequential. The increase in domain count also lowers the solving time for $\boldsymbol{K}^j_{RR}$. When domains become smaller with fewer remainder DOFs, solving time of $\boldsymbol{K}^j_{RR}$ at each domain declines quadratically. However, when the domain decomposition gets denser, $\boldsymbol{G}_{RR}$ becomes a bigger matrix, and we can clearly observe a steady increase of orange bars w.r.t. the increase of the number of domains. In some cases, solving the dual variable $\boldsymbol{\lambda}$ takes more time than solving the primal variable $\boldsymbol{x}_R$. As a result, more domains negatively impact the runtime performance.

Another interesting notice is the efficiency improvement does not always perfectly align with the number of threads. We first need to have a sufficient number of domains to match the available number of computing cores. For instance, if the mesh is only decomposed into four domains, pushing the simulation to 16 or 32 threads is not helping. The performance gets worse as each thread is assigned with fewer hardware resources. If the mesh resolution is not fine enough, parallel solve also becomes less effective. A "sweet spot" is making the domain count similar to the number of processing cores. For instance, our AMD CPU has 32 cores, and we often find generating 32 domains gives good speedup. The Intel CPU used in this test has 24 cores. However, they are not equally powerful, with 8 performance cores and 16 efficiency cores. This hardware-level variation may explain the differences in large-scale multi-domain solves.

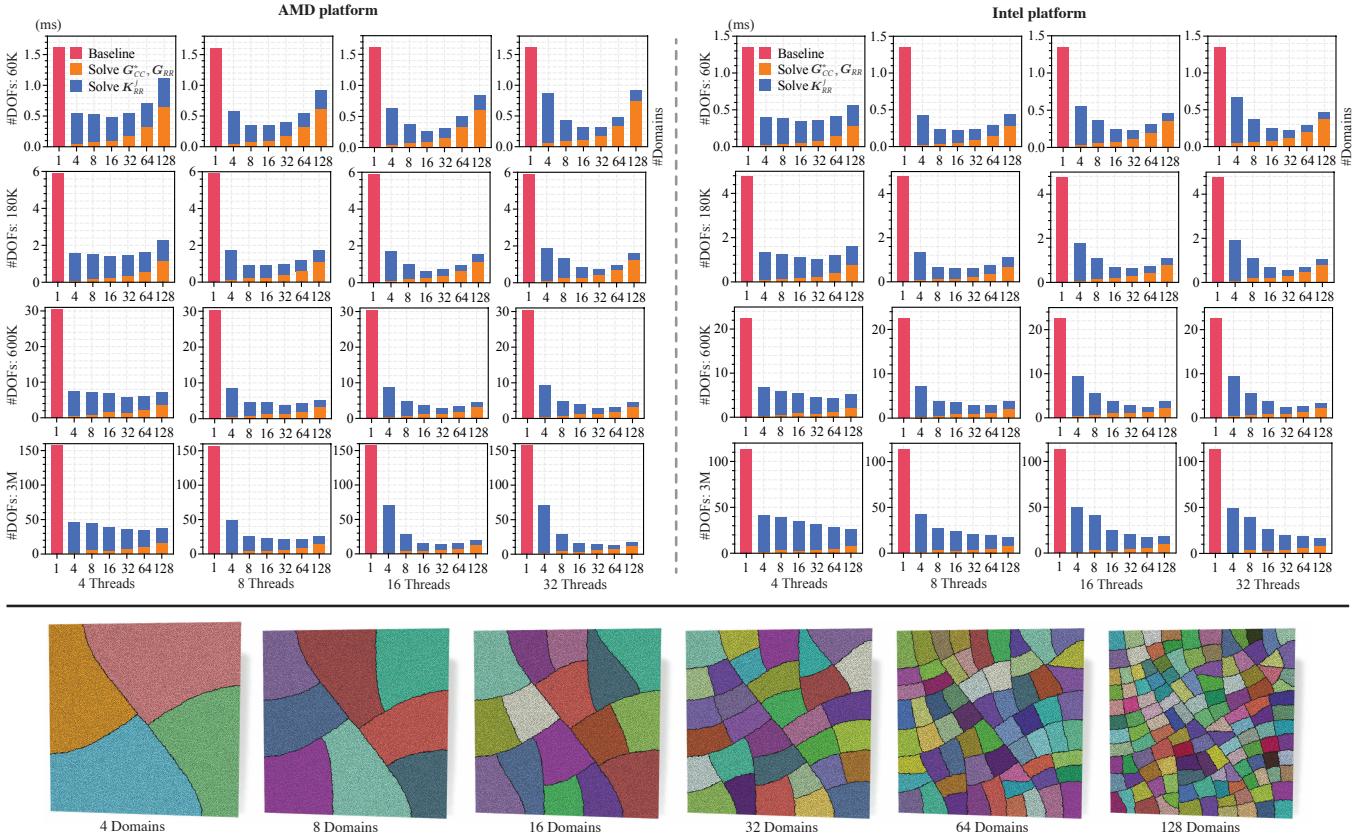

Fig. 9. **Multi-domain global solve with different numbers of domains & CPU threads.** We report the global solve time using domain-decomposed parallel solver on square cloth meshes of different resolutions with 60K, 180K, 600K, and 3M DOFs. The mesh is decomposed into 4, 8, 16, 32, 64 and 128 domains respectively. The domain partition is shown at the bottom. Compared with the baseline using the off-the-shelf numerical library i.e., the left bar in all the plots, which performs the global solve on the entire mesh, our method brings multifold speedups. The test runs on both the AMD CPU (left) and the Intel CPU (right) using different numbers (4 - 32) of threads.

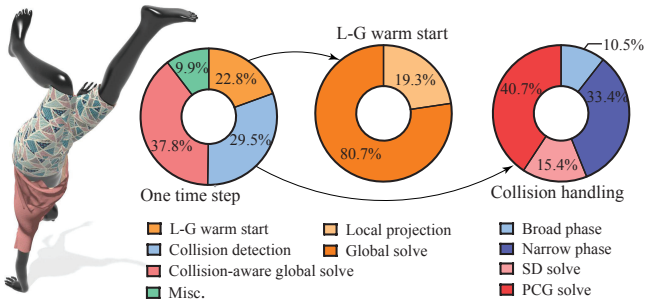

Fig. 10. **Time breakdown.** We visualize the breakdown of the computation time used for one time step of a typical animation frame, where the character in a t-shirt and shorts is performing a single-hand handstand. There are 376K DOFs in this example (91K collision constraints). The time step size is set as $h = 1/120$, and the total time used at this frame is 240 ms.

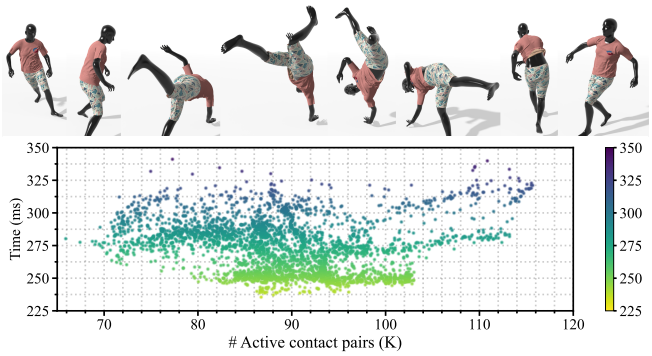

Fig. 11. **Relationship between active contact pairs and time.** We analyze the per frame time with respect to the number of active contact pairs. While a significant increase in active contact pairs typically slows down simulations, our method shows relatively stable performance across a range of contact pair quantities in the given example.

Nevertheless, our method offers substantial performance gain. In high-resolution simulations, the global solve becomes an order faster. This timing performance is nowhere close to GPU-based global

Table 1. **Experiment statistics.** The table below presents detailed timing statistics for all experiments discussed in the paper. # **D** gives the total number of domains in the example. # **DOF** is the total number of simulation DOFs. # **Ele.** indicates the total number of elements (triangles for cloth meshes and tetrahedrons for deformable objects). **C|B|R** reports the number of corner DOFs, boundary DOFs, and remainder DOFs. h is the time step size, which is set as 1/120 in most cases. The **Cull|Nrw.** column reports the timing information for collision handling: **Cull** is for broad-phase collision culling; and **Nrw.** is for the narrow phase identifying all the collision constraints. ρ is the mass density, measured in $\frac{\text{kg}}{\text{m}^2}$ for cloth and $\frac{\text{kg}}{\text{m}^3}$ for deformable objects. For simulations containing both (e.g., Fig. 22), the first value corresponds to cloth and the second to the deformable object. **Mats.** lists material parameters: stretching (in Pa) and bending stiffness for cloth, or Young's modulus (in Pa) for deformable objects. κ_s, κ_c denotes the weights (w_k) for self-collision and collider collision constraints. $\|\Delta x\|$ specifies the convergence criterion. # **Iter.** gives the average per-time-step total number of L-G iterations. **L|G|SD|PCG** gives the time consumed by the solver. **L** and **G** report timing for the collision-free warm start. **SD** and **PCG** give the computation time used for SD solve (i.e., Eq. (24)) and full PCG solve pre-conditioned with the collision-free global solver. **Misc.** is additional computational costs. All timing measurements presented above represent the average time for the complete sequence. The rightmost column displays the overall simulation time, encompassing collision detection, collision handling, local and global steps, and other processes. **Min.** and **Max.** indicate the minimum and maximum per-frame times, while **Med.** and **Avg.** represent the median and average per-frame times respectively. All timing measurements are in milliseconds.

Scene	# D	# DOF	# Ele.	# C | B | R	h	Cull | Nrw.	ρ	Mats.	κ_s, κ_c	$\|\Delta x\|$	# Iter.	L | G SD | PCG	Misc.	Min. | Max. Med. | Avg.
Cloth on Armadillo (Fig. 14)	32	282K	187K	3K|6K|117K	$\frac{1}{120}$	15|32	0.5	2e4|2e−2	1e6, 2e6	1e−3	11	9|19 6|57	7	132 | 194 148 | 145
Kick (Fig. 15)	96	444K	291K	12K|24K|393K	$\frac{1}{120}$	40|129	1	2e4|2e−2	1e6, 2e6	1e−3	16	37|93 68|204	17	460 | 1292 546 | 588
Make a knot (Fig. 16)	64	324K	212K	4.5K|9K|319.5K	$\frac{1}{120}$	45|167	0.5	1e4|1e−2	1e6, 2e6	1e−3	14	28|78 66|163	17	233 | 851 544 | 564
Fashion show (Fig. 17)	73	1.1M	656K	18K|36K|966K	$\frac{1}{120}$	49|96	0.3	2e4|2e−2	1e6, 2e6	1e−3	15	39|268 157|433	21	863 | 1, 961 966 | 1, 063
Single handstand (Fig. 18a)	72	376K	234K	6K|12K|222K	$\frac{1}{120}$	31|76	0.5	1e4|2e−2	1e6, 2e6	1e−3	9	15|43 34|80	17	235 | 341 302 | 296
Hip-hop (Fig. 18b)	76	390K	256K	6K|12K|354K	$\frac{1}{120}$	44|71	1	1e4|2e−2	1e6, 2e6	1e−3	11	27|78 59|104	12	286 | 458 382 | 395
Multi-layered dress (Fig. 18c)	89	315K	206K	12K|24K|273K	$\frac{1}{120}$	39|69	0.5	1e4|5e−3	1e6, 2e6	1e−3	14	26|64 51|83	19	264 | 626 343 | 351
Robe dance (Fig. 18d)	84	519K	343K	12K|24K|381K	$\frac{1}{120}$	27|70	0.3	1e4|5e−3	1e6, 2e6	1e−3	8	19|66 48|125	19	301 | 647 381 | 374
Open the window (Fig. 19)	64	6M	4M	60K|120K|6M	$\frac{1}{120}$	711|1, 482	0.5	2e4|2e−2	1e6, 2e6	1e−3	28	466|1, 204 966|1, 772	56	3, 022 | 9, 867 6, 991 | 6, 657
Folding (Fig. 20)	64	492K	324K	12K|24K|480K	$\frac{1}{120}$	48|102	0.5	1e4|1e−2	1e6, 2e6	1e−3	11	45|98 46|202	16	231 | 991 566 | 557
Dressed Armadillo (Fig. 22)	48	150K	164K	6K|12K|144K	$\frac{1}{120}$	39|79	1 1e3	1e4|1e−2 1e5	1e6, 1e7	2e−3	9	34|11 11|21	9	185 | 236 201 | 204
Barbarian ship (Fig. 23)	64	402K	487K	15K|30K|386K	$\frac{1}{120}$	31|93	1e2	5e6	1e6, 1e6	1e−3	10	93|59 28|61	14	349 | 450 377 | 379

solvers. For instance, Wang [2015] used diagonally pre-conditioned Jacobi to solve the global matrix inexactly. Each Jacobi iteration uses less than 0.1 ms, even for high-resolution models. A key difference lies in the fact that our method fully (exactly) solves the global matrix, while GPU solvers only give an approximated global solution. As a result, our method needs much fewer L-G iterations for each time step. The exactness of the global solve is another hidden factor for our efficiency.

7.2 Multi-domain Local Solve

Our local solve is also specifically designed for the multicore CPU. As discussed in Sec. 5, we use a hybrid constraint relaxation scheme, with GS-based sequential update within the domain and a Jacobi-like update across domains. We note that local GS relaxation converges the system more effectively compared with the Jacobi method employed in the vanilla PD framework. To better illustrate this advantage, we show a side-by-side comparison between these two different constraint projection schemes in Fig. 12. The simulation is a standard draping test of a tablecloth with two corners fixed.

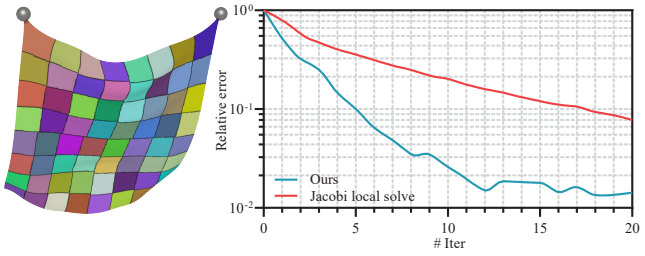

Fig. 12. **Multi-domain local solve.** Our local projection is different from the vanilla PD, which is sequential within a domain and parallelized across domains. At each domain, we relax constraint in a GS manner alternating among several pre-computed traversal orders. In this simple but representative example, our method uses much fewer L-G iterations compared with the Jacobi-based parallel projection scheme. In this example, the global matrix is always exactly solved.

We decompose the mesh into 64 domains, as shown in the figure. As the global step matrix is exactly solved, the total number

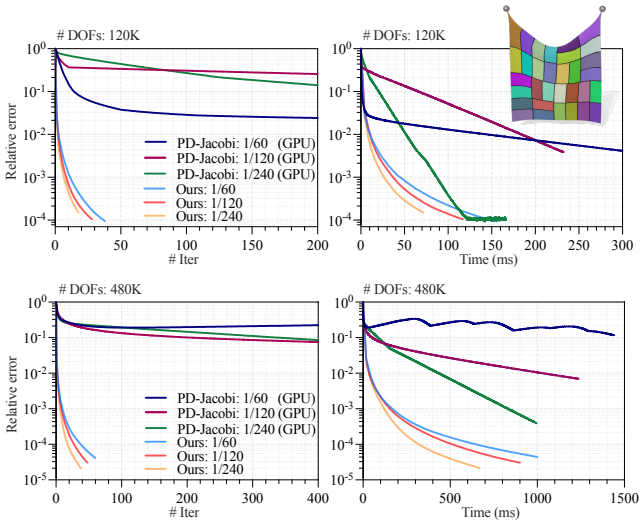

Fig. 13. **Comparison with PD-Jacobi (GPU).** We plot the convergence curves of our method and PD-Jacobi [Wang 2015] in a simple simulation, where the square tablecloth deforms under gravity with two top corners fixed. We use 32 domains for our method. Our method uses much fewer L-G iterations in all setups. Due to the GPU parallelization, PD-Jacobi becomes more efficient when $h = 1/240$. If the time step size is set more aggressively to $1/120$ or $1/60$, our method re-takes the performance advantage.

of L-G iterations is a good indicator reflecting the quality of the local projection. To this end, we plot the convergence curves using Jacobi local projection and our method. Our local projection shows a noticeable advantage compared with vanilla PD, which prioritizes parallelization.

7.3 Halley-based CCD processing

Our method is compatible with existing collision handling approaches such as the nonlinear penalty method [Wu et al. 2020], constraint-based barrier method [Lan et al. 2022], or exponential penalty [Lan et al. 2024]. For the broad-phase collision detection, we employ parallel linear BVH [Karras 2012] with AABB as leaf nodes to construct the scene acceleration structure and perform intersection queries. This structure is updated at each time step. The refitting and querying typically take less than 15% of the total runtime.

We use different strategies for garment-garment collisions and garment-collider collisions. Specifically, for garments and external colliders (e.g., floor, obstacle, or the body of an avatar), we simplify collision detection to pairs between garment vertices and collider primitives. For self-collisions, we consider all types of collision pairs between vertices and primitives after the broad-phase culling. We generate the collision constraints through projecting the target positions in the local step as in [Lan et al. 2022]. We set the weight of collision constraints w_k as a big constant, typically two orders bigger than other constraint weight w_i (as reported in Tab. 1). This makes the steepest descent highly effective for collision-aware global solving.

Being a co-dimensional model, CCD is always needed for narrow-phase collision detection for cloth animation. Each CCD for a pair

Fig. 14. **Comparison with PD-Coulomb (CPU).** Our method is faster than PD-Coulomb [Ly et al. 2020] by an order. In this experiment, a piece of cloth drops on an Armadillo and falls on the floor. While PD-Coulomb is also accelerated by CPU multithreading, the parallelization only applies at the local projection with OpenMP. Our method can be better accelerated by more CPU cores at both local and global steps, and it also converges faster than PD-Coulomb due to our novel projection scheme. In this experiment, there are 282K DOFs, and our method uses 145 ms to simulate one time step.

of colliding primitives needs to solve a cubic polynomial $f(t) = 0$. While analytic root finding for cubic equations is possible, it is not preferred because of the numerical stability issue. Instead, numerical root finding is commonly chosen, such as Newton's method. We follow the idea in [Yuksel 2022] that performs the root finding at intervals. However, we use Halley's method [Scavo and Thoo 1995] for its better convergence. Halley's method is slightly more expensive than Newton's method for cubic problems. Nonetheless, this complexity disparity is invisible on the CPU cores (given their fast clock cycles).

Since $f(t)$ is a cubic function. Its second derivative is constant. Halley iteration for finding the root i.e., the TOI (time of impact) of $f(t)$ is given as:

$$t \leftarrow t - \frac{2f(t)f'(t)}{2\left[f'(t)\right]^2 - f(t)f''(t)}. \tag{25}$$

Our experiment shows that it converges 10% to 15% faster than Newton-based root finding on average on the CPU.

7.4 Comparison with Existing GPU/CPU Algorithms

Next, we compare our method with several classic cloth simulation algorithms, including PD-Jacobi [Wang 2015] (GPU), PD-Coulomb [Ly et al. 2020] (CPU), C-IPC [Li et al. 2021] (CPU), PD-BFGS [Li et al. 2023] (GPU), PD-IPC [Lan et al. 2022] (GPU), and PD-EXP [Lan et al. 2024] (GPU). All the methods produce high-quality results when converged, and the visual differences between different methods are hardly discernable. Performance-wise, our method significantly outperforms CPU-based methods (e.g., 10× to 100× faster compared with the multithreaded PD/FEM methods) and achieves a runtime FPS comparable to many GPU solvers. It is unlikely for a CPU simulator to outperform all GPU-based algorithms in terms of speed. Nevertheless, we confidently regard our method as one of the best-performing CPU solvers to date.

Comparison with PD-Jacobi (GPU). PD-Jacobi [Wang 2015] uses Chebyshev acceleration to improve the convergence of the Jacobi method for the global solve. It has been a popular choice for high-performance cloth simulation due to its simplicity and convenient

Fig. 15. **Comparison with PD-BFGS (GPU) & C-IPC (CPU).** The character in a tired skirt performs a kicking action. Fast body movements generate interesting garment dynamics and rich self-collisions. There are 444K DOFs in the scene. All the methods produce high-quality animations. Our method uses 588 ms for solving one time step ($h = 1/120$), which is 2.6× faster than GPU-based PD-BFGS and 66× faster than CPU-based C-IPC.

Fig. 16. **Comparison with PD-IPC (GPU).** Two fabric strips are tangled and pulled in opposite directions to form a tight knot. The left figure illustrates the domain partition in the rest configuration. Similar to PD-IPC, we apply the CCD filtering after each iteration of the contact-aware global solve. There are 324K DOFs in the scene, and 57K collision constraints. Our method is comparable with PD-IPC, and runs at 564 ms per frame. The runtime performance of PD-IPC for this simulation is 544 ms per frame. The time step size is $h = 1/120$.

Fig. 17. **Comparison with PD-EXP (GPU).** In this example, we compare our method with PD-EXP [Lan et al. 2024] to simulate a virtual fashion show, where the character in a light midi skirt walks to the front and then turns around. We apply an external wind field to the left to generate more dynamic garment movements. There are 1.1M DOFs in the simulation. Due to the subspace pre-conditioning, PD-EXP is faster than our method. It uses 734 ms to simulate one time step ($h = 1/120$), while our method needs 1.06 seconds on average.

implementation, e.g., one does not even need to assemble the PD global matrix. PD-Jacobi uses a single Jacobi iteration to approximate the solution of the global solve. This parallelism-concentrate scheme works well for conservative time steps. When the time step is set more aggressively e.g., $1/120$ or even $1/60$, or the number of simulation DOFs is further increased, the number of L-G iterations needed to converge one time step goes up substantially. This comparison is based on a simple experiment of hanging a piece of tablecloth. We plot the convergence curves of our method and PD-Jacobi under three different time step sizes, and different mesh resolutions in Fig. 13. The domain decomposition of the mesh is also visualized in the figure. It can be clearly seen from this comparison that our method uses much fewer L-G iterations compared with PD-Jacobi in all situations. This is due to the combination of the exact global matrix solve and better-converging local projection. However, superior parallelization of the GPU makes Jacobi iteration and per-constraint projection highly efficient. This efficiency can compensate for the increased iteration count when $h = 1/240$ — we can see PD-Jacobi converges faster in terms of computation time. However, if the time step size increases to $1/120$ or $1/60$, our method becomes more efficient. When the resolution of the mesh is further increased, our method is faster than PD-Jacobi even for $h = 1/240$.

Comparison with PD-Coulomb (CPU). PD-Coulomb [Ly et al. 2020] is a CPU-based simulation algorithm. Its main contribution is novelly converting the classic Coulomb friction model to the form of constraint projection within the PD framework. PD-Coulomb uses OpenMP for parallelization. To make sure the comparison is objective, we implemented the same frictional constraint as in [Ly et al. 2020]. The snapshots of this experiment are shown in Fig. 14, where we drape a piece of cloth on an Armadillo. There are 282K DOFs and 56K collision constraints on average in this experiment. Our method is 11× faster under the same convergence condition and time step size.

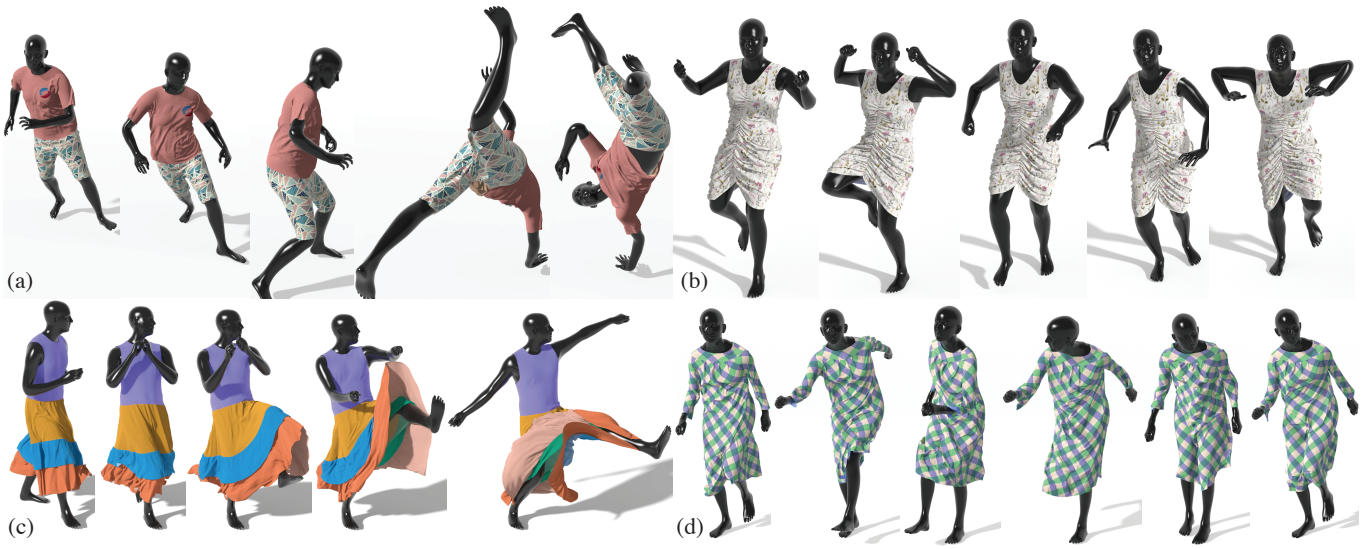

Fig. 18. **Garments on moving avatars.** Our method produces interesting and realistic garment dynamics on moving avatars. We show snapshots of different combinations of motions and garments. In sub-figure (a), the character in a fitted t-shirt and shorts performs a single handstand. In sub-figure (b), the character dresses with a tight white nightdress and performs a hip-hop dance on the spot. In sub-figure (c), the character wearing a loose, multi-layered long dress performs a throwing motion. In sub-figure (d), the character dressed in a robe is swaying and warling in place. There are 376K, 390K, 315K, 519K DOFs in sub-figures (a), (b), (c), and (d), and the simulation runs at 296 ms, 395 ms, 351 ms, 374 ms per frame, respectively.

Fig. 19. **Open the window.** We show a high-resolution simulation involving 6M DOFs. Each curtain has 32 domains as shown. They are pulled away from the middle to reveal the view of a city night. At this resolution, inexact global solve will significantly increase the total number of L-G iterations (e.g., as in [Wang 2015]). Our method converges more effectively and uses 6.6 s to simulate one frame under $h = 1/120$.

Comparison with PD-BFGS (GPU) & C-IPC (CPU). PD-BFGS is a GPU cloth simulation method [Li et al. 2023]. It uses a spline-based subspace to pre-condition the global matrix before the GPU Jacobi iteration. C-IPC is a CPU-based algorithm [Li et al. 2021] using full nonlinear FEM and Newton's method to solve the garment/thin-shell dynamics. It employs the incremental potential contact (IPC) as the major modality for collision processing [Li et al. 2020]. C-IPC is very expensive because of the use of full Newton solve, and it also needs CCD-based line search filtering after each Newton iteration to make sure all the triangles are separate. The experiment results are reported in Fig. 15. In this comparative example, the virtual character with a tiered skirt performs a kicking action. The domain decomposition of the garment is also provided in the figure. There are 444K DOFs in this example. Our method uses 16 L-G iterations and 588 ms on average to simulate one frame. While can be effectively accelerated on the GPU, PD-BFGS uses a high-dimension subspace to pre-condition the Jacobi iteration. Our method is 2.6× faster than the GPU-based PD-BFGS. C-IPC is not parallelization friendly, and our method is 66× faster than multicore accelerated C-IPC.

Comparison with PD-IPC (GPU). PD-IPC [Lan et al. 2022] incorporates IPC barrier in the framework of PD by using the IPC barrier as the weight of a collision constraint. Similarly to C-IPC, PD-IPC

Fig. 20. **Folding.** In this scene, four tablecloths fall on the table one by one with different orientations. When stacking on each other, complex self-collisions are generated. Our method can robustly handle this simulation. There are 492K DOFs in the system, and the simulation runs at 557 ms per frame.

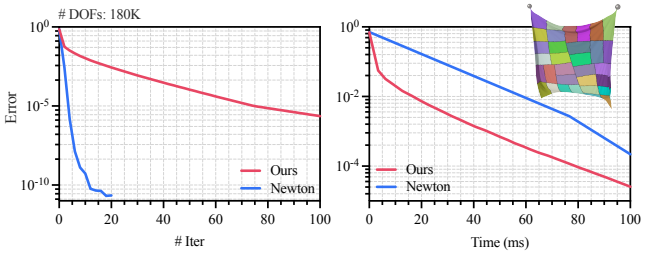

Fig. 21. **Comparison with Newton method (CPU).** We compare the convergence characteristics of our method against the Newton method in a simple simulation when a square tablecloth deformed under gravity with two top corners fixed. For our approach, we employ 32 domains. The time step size is 1/60. Using gradient magnitude as the error metric, we analyze the performance based on both iteration counts and runtime (in milliseconds). The results demonstrate that while the Newton method exhibits significantly faster convergence per iteration, our method requires less computational time overall, even with a relatively large time step.

guarantees that the resulting simulation is free of interpenetration. To compare the performance difference between our method and PD-IPC, we run a simulation involving extensive self-collisions. As shown in Fig. 16, we pull two strips in opposite directions to create a tight knot. There are 324K DOFs in the example. Our method uses 564 ms on average for one time step with 14 L-G iterations. In this comparison, we choose to use the rank-two aggregated Jacobi method for PD-IPC. On average, PD-IPC needs 92 L-G iterations. Similarly to the previous comparison with PD-Jacobi (Fig. 13), our method takes a longer computation time to complete one L-G iteration with fewer iterations. Therefore, our method has a comparable performance compared with PD-IPC (544 ms per frame).

Comparison with PD-EXP (GPU). PD-EXP [Lan et al. 2024] uses an exponential barrier to process collisions so that the barrier does not depend on the actual distance between primitives. More importantly, PD-EXP uses a modal subspace preconditioner for the global matrix solve, which eliminates most low-frequency residual errors. As a result, Jacobi or aggregated-Jacobi methods become highly effective. PD-EXP is the fastest simulation method we tested, and it is faster than our method. In this comparison, we simulate a fashion runway scene with a walking character using both our method and PD-EXP as shown in Fig. 17. There are over 1.1M DOFs in this scene. With $h = 1/120$, PD-EXP uses 734 ms for one time step while our method needs 1.06 seconds.

Comparison with Newton method. We also compare our method with Newton method. In this experiment, a 180K DOFs square tablecloth deforms under gravity with two corners fixed. We employ 32 domains for our method. Fig. 21 demonstrates that while the Newton method exhibits significantly fast convergence per iteration, our method requires less computational time overall, even with a relatively large time step (1/60).

7.5 More Results

Figs. 1 and 18 show a set of simulation results using our method. The virtual avatar performs four different motions, including hand standing, tossing, dancing, and walking. We simulate the dynamics of diverse types of garments: shorts, t-shirt, fit dress, robe, and multi-tiered skirt. Our method produces high-quality results in all examples. The simulated garment dynamics is interesting and realistic. The domain decomposition of each garment model is visualized in Fig. 1. Please refer to Tab. 1 for detailed timing information and simulation parameters.

Fig. 19 reports a high-resolution simulation to show the scalability of our method. This experiment simulates the dynamics of pulling a pair of curtains from the middle to sides, capturing their trajectories and the formation of wrinkles. There are 6M DOFs in the simulation. Each curtain has 32 domains making 64 domains in total, as visualized in the figure. It takes 6.6 seconds to simulate one time step ($h = 1/120$).

Our method is able to handle simulations with intensive self-collision. As shown in Fig. 20, we vertically drop four tablecloths on a desk. This simple setup is quite challenging for a numerical solver as the falling cloths involve a large number of self-collisions under high-velocity movement. We follow the strategy as in C-IPC [Li et al. 2021] to apply a CCD-based line search filtering by the end of each time step. There are 492K DOFs. Our method uses 478 ms to simulate one frame.

7.6 Deformable Objects

While our primary focus is high-resolution cloth simulation on the CPU, our method can also be used to simulate volumetric deformable models. Because of the reliance on PD framework, we choose the ARAP material that can be approximated with quadratic constraints. To this end, we report two more experiments. In Fig. 22, we dress the Armadillo in a skirt and drop two of such dressed Armadillos into a container. During the falling, the Armadillos interact with several cylinders as well as body-body, body-garment, and garment-garment collisions. Our method is able to seamlessly handle both volumetric models and garment meshes. There are 32 domains on the Armadillo and 16 on the skirt. Each time step takes 204 ms.

Fig. 23 shows the other deformable simulation result of a falling barbarian ship. The ship has a complex geometry, which facilitates our domain decomposition because it allows small-size domain interfaces. There are 487K elements on the ship, and our method simulates one frame using 379 ms.

8 Conclusion & Limitation

In this paper, we propose a cloth simulation framework that is specially tailored for the CPU platform. We algorithmically integrate two computational techniques, namely domain decomposition and projective dynamics to achieve this goal. The core of our method is to combine a primal-dual formulation of the global PD solve so that the forward/backward substitution can be parallelized at domains. This fast global solver, when coupled with a condensed steepest descent, can effectively handle varying global systems caused by garment collisions. The local step computation is also changed from a Jacobi-like relaxation to a hybrid scheme, where pre-computed GS relaxation is sequentially carried out at each domain, and per-domain computations are processed concurrently. Together with a better CCD solver, our method delivers a similar runtime performance as the state-of-the-art GPU-based algorithms.

However, modern processors often feature heterogeneous cores, such as performance cores and efficient cores [Jarus et al. 2013]. To fully leverage these heterogeneous architectures, the domain decomposition strategy must be adapted accordingly. Additionally, it is of great interest to deploy the proposed multi-domain solver on the GPU. A GPU thread can only efficiently process small-size $\boldsymbol{K}^j_{RR}$, necessitating a densely decomposed domain structure. This results in a high-dimensional dual problem for solving $\boldsymbol{\lambda}$ and a large-scale $\boldsymbol{G}_{RR}$ as described in Eq. (9). Fortunately, our primal-dual global solve algorithm can be recursively applied to address $\boldsymbol{G}_{RR}$, offering a potential solution to this challenge (Fig. 24). In the future, we will also explore the possibility of applying our algorithm on mobile devices like smartphones and tablets, where the hardware resource is constrained.

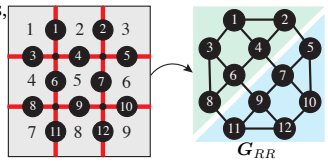

Fig. 24. **Recursive primal-dual solve.** When domains are densely decomposed, solving the dual variable $\boldsymbol{\lambda}$ at duplicate DOFs may be a large-scale problem as well. However, they are decoupled by the corner DOFs (small solid dots), and $\boldsymbol{G}_{RR}$ is a sparse block-wise matrix. Two sets of boundary DOFs (i.e., black circles) are coupled if and only if they are shared by a domain. The denser the decomposition is, the sparser boundary DOFs are coupled. Therefore, we can apply another domain decomposition to the graph representing the connectivity of boundary DOFs (as shown on the right) and solve $\boldsymbol{G}_{RR}$ in parallel.

Our method is not without limitations. First, it does not explicitly address collision handling. The CPU faces inherent challenges in managing a large number of collision pairs, making it less efficient than the GPU for collision processing. As reported in Tab. 1, collision detection represents a significant portion of the total computation time. Our method incorporates with PD framework, this local-global alternation scheme is only applicable to certain idealized material models, and its extension to more complex, real-world materials is less straightforward. We recommend [Liu et al. 2017] for more details. Furthermore, generalizing our method to other nonlinear solvers such as ADMM presents challenges. Unlike Projective Dynamics, which utilizes a quadratic energy formulation with a constant system matrix, ADMM accommodates nonlinear constitutive models and dynamic constraints. This nonlinearity fundamentally complicates the prefactorization strategy, as system matrices change between iterative steps. The problem of efficiently updating intermediate matrices in interative-time or even in real-time remains an open research question, with no straightforward extension of our current approach being immediately apparent. We will keep explore adapting the proposed multi-domain PD solver to handle a broader range of material behaviors in a more general optimization framework. Currently, our approach assumes that all CPU cores have similar computational capabilities and subdivides the mesh into domains of roughly equal sizes.

Acknowledgments

We thank reviewers for their detailed and constructive comments. Chenfanfu Jiang is partially supported by NSF 2153851 and TRI. Yin Yang is partially supported by NSF under grant number 2301040.

References

Steven S An, Theodore Kim, and Doug L James. 2008. Optimizing cubature for efficient integration of subspace deformations. *ACM transactions on graphics (TOG)* 27, 5 (2008), 1–10.

David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 43–54.

Jernej Barbič and Yili Zhao. 2011. Real-time large-deformation substructuring. *ACM transactions on graphics (TOG)* 30, 4 (2011), 1–8.

Klaus-Jürgen Bathe. 2006. *Finite element procedures*. Klaus-Jurgen Bathe.

Miklos Bergou, Max Wardetzky, David Harmon, Denis Zorin, and Eitan Grinspun. 2006. A quadratic bending model for inextensible surfaces. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (Cagliari, Sardinia, Italy) *(SGP '06)*. Eurographics Association, Goslar, DEU, 227–230.

Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. 2003. The GPU as numerical simulation engine. In *SIGGRAPH 2003*. Citeseer.

Folkmar A Bornemann and Peter Deuflhard. 1996. The cascadic multigrid method for elliptic problems. *Numer. Math.* 75 (1996), 135–152.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *Acm Transactions On Graphics* 33, 4 (2014), 154.

R. Bridson, S. Marino, and R. Fedkiw. 2005. Simulation of clothing with folds and wrinkles. In *ACM SIGGRAPH 2005 Courses on - SIGGRAPH '05*. ACM Press, Los Angeles, California, 3.

Hsiao-Yu Chen, Arnav Sastry, Wim M van Rees, and Etienne Vouga. 2018. Physical simulation of environmentally induced thin shell deformation. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–13.

Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans. Math. Softw.* 35, 3 (Oct. 2008), 22:1–22:14.

Kwang-Jin Choi and Hyeong-Seok Ko. 2002. Stable but responsive cloth. *ACM Trans. Graph.* 21, 3 (July 2002), 604–611.

Min Gyu Choi and Hyeong-Seok Ko. 2005. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics* 11, 1 (2005), 91–101.

Olaf Etzmuß, Michael Keckeisen, and Wolfgang Straßer. 2003. A fast finite element solution for cloth modelling. In *11th Pacific Conference onComputer Graphics and Applications, 2003. Proceedings.* IEEE, 244–251.

Charbel Farhat, Michel Lesoinne, Patrick LeTallec, Kendall Pierson, and Daniel Rixen. 2001. FETI-DP: a dual–primal unified FETI method—part I: A faster alternative to the two-level FETI method. *Numerical Meth Engineering* 50, 7 (March 2001), 1523–1544.

Charbel Farhat, Antonini Macedo, Michel Lesoinne, Francois-Xavier Roux, Frédéric Magoulés, and Armel de La Bourdonnaie. 2000. Two-level domain decomposition

Fig. 22. **Dressed Armadillo.** Two dressed Armadillos fall into a container. This example includes a rich set of interactions, including the self-collisions between deformable bodies, self-collisions between garments, body-garment collisions, body-collider collisions, and garment-collider collisions. Those interactions are unfirmly processed in our framework. There are 32 domains on the Armadillo and 16 domains on the garment. On average, each time step takes 204 ms, and there are 150K DOFs in the scene.

Fig. 23. **Barbarian ship on the stair.** In this example, the barbarian ship slides down on a spiral stair. There are 487K tetrahedron elements on the ship, which is decomposed into 64 domains. Our method uses 379 ms to simulate one time step, which on average needs 10 L-G iterations.

methods with Lagrange multipliers for the fast iterative solution of acoustic scattering problems. *Computer methods in applied mechanics and engineering* 184, 2-4 (2000), 213–239.

Charbel Farhat and Francois-Xavier Roux. 1991. A method of finite element tearing and interconnecting and its parallel solution algorithm. *Numerical Meth Engineering* 32, 6 (Oct. 1991), 1205–1227.

Sergiy Fialko. 2019. Parallel algorithms for forward and back substitution in linear algebraic equations of finite element method. *Journal of telecommunications and information technology* (2019).

Marco Fratarcangeli, Valentina Tibaldo, Fabio Pellacini, et al. 2016. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.* 35, 6 (2016), 214–1.

Theodore F. Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M. Teran. 2015. Optimization Integrator for Large Time Steps. *IEEE Transactions on Visualization and Computer Graphics* 21, 10 (2015), 1103–1115.

Joachim Georgii and Rüdiger Westermann. 2006. A Multigrid Framework for Real-Time Simulation of Deformable Bodies. *Comput. Graph.* 30, 3 (jun 2006), 408–415.

Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. 2007. Efficient simulation of inextensible cloth. In *ACM SIGGRAPH 2007 Papers* (San Diego, California) *(SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 49–es.

Eitan Grinspun, Anil N. Hirani, Mathieu Desbrun, and Peter Schröder. 2003. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '03)*. Eurographics Association, Goslar, DEU, 62–67.

Gaël Guennebaud, Benoit Jacob, et al. 2010. Eigen. *URl: http://eigen. tuxfamily. org* 3, 1 (2010), 8.

William W Hager. 1989. Updating the inverse of a matrix. *SIAM review* 31, 2 (1989), 221–239.

Donald House and David Breen. 2000. *Cloth modeling and animation.* CRC Press.

Mateusz Jarus, Sébastien Varrette, Ariel Oleksiak, and Pascal Bouvry. 2013. Performance evaluation and energy efficiency of high-density HPC platforms based on Intel, AMD and ARM processors. In *Energy Efficiency in Large Scale Distributed Systems: COST IC0804 European Conference, EE-LSDS 2013, Vienna, Austria, April 22-24, 2013, Revised Selected Papers*. Springer, 182–200.

Tero Karras. 2012. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics Conference on High-Performance Graphics*. 33–37.

George Karypis and Vipin Kumar. 1997. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. (1997). https://conservancy.umn.edu/handle/11299/215346

L. Kharevych, Weiwei Yang, Y. Tong, E. Kanso, J. E. Marsden, P. Schröder, and M. Desbrun. 2006. Geometric, Variational Integrators for Computer Animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Vienna, Austria) *(SCA '06)*. Eurographics Association, Goslar, DEU, 43–51.

Theodore Kim. 2020. A Finite Element Formulation of Baraff-Witkin Cloth. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 171–179.

Theodore Kim and Doug L James. 2011. Physics-based character skinning using multi-domain subspace deformations. In *Proceedings of the 2011 ACM SIGGRAPH/eurographics symposium on computer animation*. 63–72.

Lei Lan, Zixuan Lu, Jingyi Long, Chun Yuan, Xuan Li, Xiaowei He, Huamin Wang, Chenfanfu Jiang, and Yin Yang. 2024. Efficient GPU Cloth Simulation with Non-distance Barriers and Subspace Reuse. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–16.

Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022. Penetration-free projective dynamics on the GPU. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–16.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.* 39, 4 (Aug. 2020), 49:49:1–49:49:20.

Minchen Li, Ming Gao, Timothy Langlois, Chenfanfu Jiang, and Danny M Kaufman. 2019. Decomposed optimization time integrator for large-step elastodynamics. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–10.

Minchen Li, Danny M Kaufman, and Chenfanfu Jiang. 2021. Codimensional incremental potential contact. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–24.

Xuan Li, Yu Fang, Lei Lan, Huamin Wang, Yin Yang, Minchen Li, and Chenfanfu Jiang. 2023. Subspace-Preconditioned GPU Projective Dynamics with Contact for Cloth Simulation. In *SIGGRAPH Asia 2023 Conference Papers* (Sydney, NSW, Australia) *(SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 1, 12 pages.

Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. 2013. Fast simulation of mass-spring systems. *ACM Trans. Graph.* 32, 6 (Nov. 2013), 1–7.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-newton methods for real-time simulation of hyperelastic materials. *Acm Transactions on Graphics (TOG)* 36, 3 (2017), 1–16.

Zixuan Lu, Xiaowei He, Yuzhong Guo, Xuehui Liu, and Huamin Wang. 2024. Projective Peridynamic Modeling of Hyperelastic Membranes With Contact. *IEEE Transactions on Visualization and Computer Graphics* 30, 8 (2024), 4601–4614.

Mickaël Ly, Jean Jouve, Laurence Boissieux, and Florence Bertails-Descoubes. 2020. Projective dynamics with dry frictional contact. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 57–1.

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

Richard H MacNeal. 1971. A hybrid method of component mode synthesis. *Computers & Structures* 1, 4 (1971), 581–601.

A. McAdams, E. Sifakis, and J. Teran. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Madrid, Spain) *(SCA '10)*. Eurographics Association, Goslar, DEU, 65–74.

Jeroen Molemaker, Jonathan M. Cohen, Sanjit Patel, and Jonyong Noh. 2008. Low Viscosity Flow Simulations for Animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Dublin, Ireland) *(SCA '08)*. Eurographics Association, Goslar, DEU, 9–18.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.

James O'Brien, Kris Hauser, and Chen Shen. 2003. Interactive Deformation Using Modal Analysis with Constraints. *Graphics Interface* 3 (05 2003).

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 (Oct. 2019), 17 pages. doi:10.1145/3355621

A. Pentland and J. Williams. 1989. Good Vibrations: Modal Dynamics for Graphics and Animation. *SIGGRAPH Comput. Graph.* 23, 3 (jul 1989), 207–214.

Chuck Pheatt. 2008. Intel® threading building blocks. *Journal of Computing Sciences in Colleges* 23, 4 (2008), 298–298.

Xavier Provot et al. 1995. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics interface*. Canadian Information Processing Society, 147–147.

TR Scavo and JB Thoo. 1995. On the geometry of Halley's method. *The American mathematical monthly* 102, 5 (1995), 417–426.

Barry F Smith. 1997. Domain decomposition methods for partial differential equations. In *Parallel numerical algorithms*. Springer, 225–243.

Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed Aggregation Multigrid for Cloth Simulation. *ACM Trans. Graph.* 34, 6, Article 245 (nov 2015), 13 pages.

Min Tang, Ruofeng Tong, Rahul Narain, Chang Meng, and Dinesh Manocha. 2013. A GPU-based streaming algorithm for high-resolution cloth simulation. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 21–30.

Bernhard Thomaszewski, Simon Pabst, and Wolfgang Strasser. 2009. Continuum-based strain limiting. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 569–576.

Andrea Toselli and Olof Widlund. 2006. *Domain decomposition methods-algorithms and theory*. Vol. 34. Springer Science & Business Media.

Ulrich Trottenberg, Cornelius W. Oosterlee, Anton Schuller, and Achi Brandt. 2001. *Multigrid*. Academic Press.

Pascal Volino, Nadia Magnenat-Thalmann, and Francois Faure. 2009. A simple approach to nonlinear tensile stiffness for accurate cloth simulation. *ACM Trans. Graph.* 28, 4, Article 105 (Sept. 2009), 16 pages.

Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, Yajuan Wang, Endong Wang, Qing Zhang, Bo Shen, et al. 2014. Intel math kernel library. *High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures* (2014), 167–188.

Huamin Wang. 2015. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph.* 34, 6 (Nov. 2015), 246:1–246:9.

Huamin Wang, James O'Brien, and Ravi Ramamoorthi. 2010. Multi-resolution isotropic strain limiting. *ACM Transactions on Graphics (TOG)* 29, 6 (2010), 1–10.

Huamin Wang, James F O'Brien, and Ravi Ramamoorthi. 2011. Data-driven elastic models for cloth: modeling and measurement. *ACM transactions on graphics (TOG)* 30, 4 (2011), 1–12.

Zhendong Wang, Longhua Wu, Marco Fratarcangeli, Min Tang, and Huamin Wang. 2018. Parallel multigrid for nonlinear cloth simulation. *Computer Graphics Forum* 37, 7 (10 2018), 131–141.

Zhendong Wang, Yin Yang, and Huamin Wang. 2023. Stable Discrete Bending by Analytic Eigensystem and Adaptive Orthotropic Geometric Stiffness. *ACM Trans. Graph.* 42, 6 (Dec. 2023), 1–16. doi:10.1145/3618372

Botao Wu, Zhendong Wang, and Huamin Wang. 2022. A GPU-based multilevel additive schwarz preconditioner for cloth and deformable body simulation. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–14.

Longhua Wu, Botao Wu, Yin Yang, and Huamin Wang. 2020. A safe and fast repulsion method for GPU-based cloth self collisions. *ACM Transactions on Graphics (TOG)* 40, 1 (2020), 1–18.

Xiaofeng Wu, Rajaditya Mukherjee, and Huamin Wang. 2015. A unified approach for subspace simulation of deformable bodies in multiple domains. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–9.

Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A Scalable Galerkin Multigrid Method for Real-Time Simulation of Deformable Objects. *ACM Trans. Graph.* 38, 6, Article 162 (nov 2019), 13 pages.

Hongyi Xu, Funshing Sin, Yufeng Zhu, and Jernej Barbič. 2015. Nonlinear material design using principal stretches. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–11.

Ichitaro Yamazaki, Sivasankaran Rajamanickam, Erik G Boman, Mark Hoemmen, Michael A Heroux, and Stanimire Tomov. 2014. Domain decomposition preconditioners for communication-avoiding Krylov methods on a hybrid CPU/GPU cluster. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 933–944.

Yin Yang, Weiwei Xu, Xiaohu Guo, Kun Zhou, and Baining Guo. 2013. Boundary-aware multidomain subspace deformation. *IEEE transactions on visualization and computer graphics* 19, 10 (2013), 1633–1645.

Cem Yuksel. 2022. High-Performance Polynomial Root Finding for Graphics. *Proc. ACM Comput. Graph. Interact. Tech.* 5, 3, Article 27 (July 2022), 15 pages.

Cyril Zeller. 2005. Cloth simulation on the GPU. In *ACM SIGGRAPH 2005 Sketches* (Los Angeles, California) *(SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 39–es.

Jiayi Eris Zhang, Jérémie Dumas, Yun Fei, Alec Jacobson, Doug L James, and Danny M Kaufman. 2022. Progressive simulation for cloth quasistatics. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–16.

Jiayi Eris Zhang, Doug James, and Danny M Kaufman. 2024. Progressive Dynamics for Cloth and Shell Animation. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–18.

Yili Zhao and Jernej Barbič. 2013. Interactive authoring of simulation-ready plants. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.