Decomposed Optimization Time Integrator for Large-Step Elastodynamics

MINCHEN LI, University of Pennsylvania & Adobe Research
MING GAO, University of Pennsylvania
TIMOTHY LANGLOIS, Adobe Research
CHENFANFU JIANG, University of Pennsylvania
DANNY M. KAUFMAN, Adobe Research

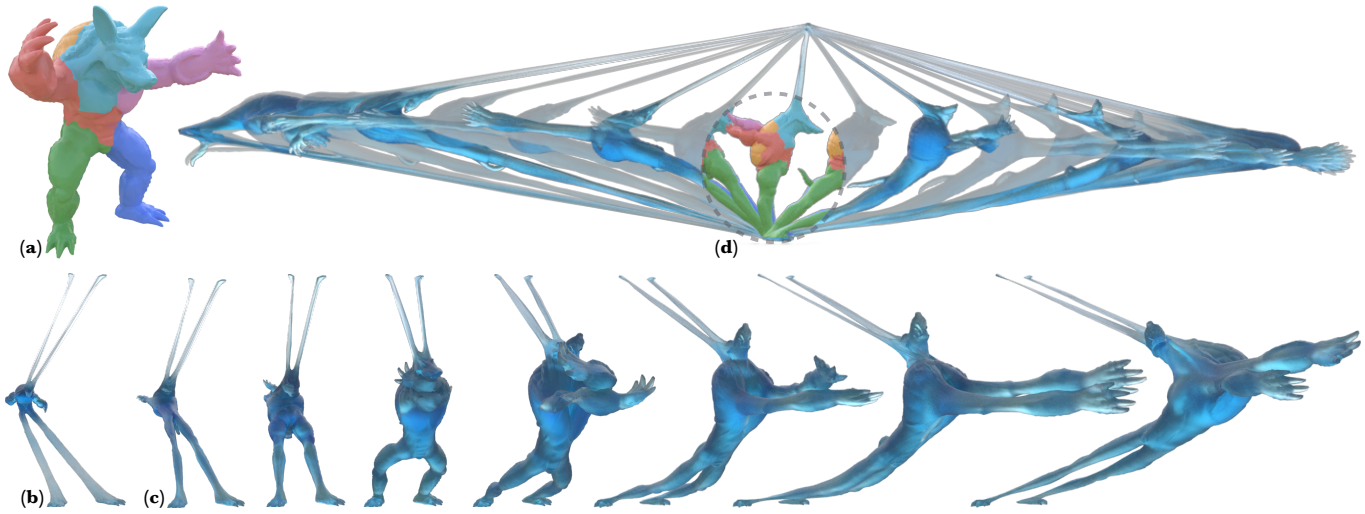

Fig. 1. ***Severe deformation dynamics simulated with large steps.*** **(a)** The Decomposed Optimization Time Integrator (DOT) decomposes spatial domains to generate high-quality simulations of nonlinear materials undergoing large-deformation dynamics. In **(b)** we apply DOT to rapidly stretch and pull an Armadillo backwards. We render in **(c)** a few frames of the resulting slingshot motion right after release. DOT efficiently solves time steps while achieving user-specified accuracies — even when stepping at frame-rate size steps; here at 25 ms. In **(d)** we emphasize the large steps taken by rendering all DOT-simulated time steps from the Armadillo's high-speed trajectory for the first few moments after release.

Simulation methods are rapidly advancing the accuracy, consistency and controllability of elastodynamic modeling and animation. Critical to these advances, we require efficient time step solvers that reliably solve all implicit time integration problems for elastica. While available time step solvers succeed admirably in some regimes, they become impractically slow, inaccurate, unstable, or even divergent in others — as we show here. Towards addressing these needs we present the Decomposed Optimization Time Integrator (DOT), a new domain-decomposed optimization method for solving the per time step, nonlinear problems of implicit numerical time integration. DOT is especially suitable for large time step simulations of deformable bodies with nonlinear materials and high-speed dynamics. It is efficient, automated, and robust at large, fixed-size time steps, thus ensuring stable, continued progress of high-quality simulation output. Across a broad range of extreme and mild deformation dynamics, using frame-rate size time steps with widely varying object shapes and mesh resolutions, we show that DOT always converges to user-set tolerances, generally well-exceeding and always close to the best wall-clock times across all previous nonlinear time step solvers, irrespective of the deformation applied.

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

Additional Key Words and Phrases: Computational Optimization, Domain Decomposition

ACM Reference Format:
Minchen Li, Ming Gao, Timothy Langlois, Chenfanfu Jiang, and Danny M. Kaufman. 2019. Decomposed Optimization Time Integrator for Large-Step Elastodynamics. *ACM Trans. Graph.* 38, 4, Article 70 (July 2019), 10 pages. https://doi.org/10.1145/3306346.3322951

Authors' addresses: Minchen Li, University of Pennsylvania & Adobe Research, minchernl@gmail.com; Ming Gao, University of Pennsylvania, ming.gao07@gmail.com; Timothy Langlois, Adobe Research, tlangloi@adobe.com; Chenfanfu Jiang, University of Pennsylvania, cffjiang@seas.upenn.edu; Danny M. Kaufman, Adobe Research, kaufman@adobe.com.

0730-0301/2019/7-ART70 $15.00
https://doi.org/10.1145/3306346.3322951

1 INTRODUCTION

Simulation of deformable-body dynamics is a fundamental and critical task in animation, physical modeling, and design. Algorithms for computing these simulations have rapidly advanced the accuracy, consistency, and controllability of generated elastodynamic trajectories. Key to these advances are a long-standing range of powerful implicit time stepping models to numerically time-integrate semi-discretized PDEs [Ascher 2008; Butcher 2016; Hairer et al. 2008; Hairer and Wanner 1996]. With a few restrictions, *incremental potentials* [Ortiz and Stainier 1999] can be constructed for these models. These are energies whose local minimizers give an implicit time step model's forward map at each time step [Hairer et al. 2006; Kane et al. 2000; Kharevych et al. 2006]. Adopting this variational perspective has enabled the application and development of powerful optimization methods to minimize these potentials and efficiently forward step dynamic simulations [Chao et al. 2010; Liu et al. 2017; Martin et al. 2011; Overby et al. 2017].

For deformable nonlinear materials, the incremental potential is the sum of a convex, quadratic discrete kinetic energy and a generally nonconvex, strongly nonlinear deformation energy weighted by time step size; see e.g. (3) below. Then, as step size and/or simulated distortions increase, the influence of this latter deformation energy term dominates. This, in turn, requires the expensive modeling of nonlinearities. As we will soon see, too weak an approximation of nonlinearity leads to large errors, artifacts, and/or instabilities, while too large or frequent an update makes other methods impractically slow — reducing progress and imposing significant memory costs.

To address these challenges we propose the Decomposed Optimization Time Integrator (DOT), a new domain-decomposed optimization method for minimizing per time step incremental potentials. DOT builds a novel quadratic matrix penalty decomposition to couple non-overlapping subdomains with weights constructed from missing subdomain Hessian information. We use this decomposition's Hessian, evaluated once at start of time step, as an inner-initializer to perform undecomposed L-BFGS time step solves on a domain mesh with a single copy of the simulation vertices. Advantages of DOT are then:

No vertex mismatch. While the decomposed Hessian is built per subdomain it is evaluated with a consistent set of vertices taken from the full, undecomposed mesh. Penalty weights thus add missing second-order Hessian data to subdomain vertices from neighbors across decomposition boundaries. This gives an initializer to our global L-BFGS solve. We use it to perform descent on the full mesh coordinates. This means vertices on interfaces are never separated, by construction.

Convergence. DOT adds no penalty forces nor gradients. DOT penalty terms only supplement our preconditioning matrix. Descent steps then precondition the undecomposed and unaugmented incremental potential's gradient and converge directly to the underlying undecomposed system's solution.

Efficiency. Subdomain Hessians are parallel evaluated and factorized once per time step. We show that they can also be applied (via small backsolves) in parallel as initializer at each iteration. Results then simply need to be blended together. This process is inserted between first and second efficient, low-rank updates of each quasi-Newton step. This couples individual, per-domain backsolves together for a global descent step. Resulting iterations are then more effective than L-BFGS approaches and yet faster than Newton.

Contributions. In summary, DOT converges at large, fixed-size time steps, ensuring stable continued progress of high-quality simulation output. DOT is an automated and robust optimization method especially suited for simulations with nonlinear materials, large deformations, and/or high-speed dynamics. By automated we mean users need not adjust algorithm parameters or tolerances to obtain good results when changing simulation parameters, conditions, or mesh sizes. By robust we mean a method should solve every reasonable time step problem to any requested accuracy given commensurate time, and only report success when the accuracy has been achieved. To achieve these goals we

- construct a quadratic penalty decomposition to couple non-overlapping subdomains with weights constructed from missing subdomain Hessian information;
- propose a resulting method using this domain-decomposition as inner initializer for undecomposed, full mesh, quasi-Newton time step solves;
- develop a line-search initialization method for reduced evaluations;
- extend Zhu et al.'s [2018] characteristic norm for consistent elastodynamic simulations; and
- perform extensive comparisons of recent performant nonlinear methods for solving large-deformation time stepping.

2 PROBLEM STATEMENT AND PRELIMINARIES

We focus on solving one-step numerical time-integration models with variational methods. Minimizing an incremental potential, E, we update state from time step t to $t+1$ with a local minimizer

$$x^{t+1} = \underset{x \in \mathbb{R}^{dn}}{\operatorname{argmin}}\, E(x, x^t, v^t), \tag{1}$$

for n vertex locations in d-dimensional space stored in vector x, with corresponding velocities v. Here $E(x, x^t, v^t)$ is a combined local measure of deformation and discrete kinetic energy, and x is potentially subject to boundary and collision constraints.

The deformation energy, $W(x)$, is (e.g. with linear finite elements) expressed as a sum over elements e in a triangulation T (triangles or tetrahedra depending on dimension),

$$W(x) = \sum_{e \in T} v_e w\big(F_e(x)\big), \tag{2}$$

where $v_e > 0$ is the area or volume of the rest shape of element e, w is an energy density function taking the deformation gradient as its argument, and F_e computes the deformation gradient for element e.

Concretely, as recent papers on solvers for time stepping in graphics [Bouaziz et al. 2014; Gast et al. 2015; Liu et al. 2013, 2017; Narain et al. 2016; Overby et al. 2017] have almost exclusively focused on the implicit Euler time stepping model, our results in the following sections will do so as well in order to provide side-by-side comparisons. For implicit Euler the incremental potential is

$$E(x, x^t, v^t) = \frac{1}{2} x^T M x - x^T M x^p + h^2 W(x). \tag{3}$$

Here $x^p = x^t + hv^t + h^2M^{-1}f$, f collects external and body forces, M is the finite element mass matrix, and velocity is updated by the implicit Euler finite difference stencil $v^{t+1} = \frac{x^{t+1}-x^t}{h}$. In the next sections we restrict our attention to solving a single time step and so, unless otherwise indicated, we simplify by specifying the incremental potential as $E(x) = E(x, x^t, v^t)$.

Notation. Throughout we will continue to apply superscripts t to indicate time step, while reserving subscripts incrementing i for inner solver iteration indices, and correspondingly subscripts with increments of j for quantities associated with subdomains.

3 RELATED WORK

Domain Decomposition. Domain decomposition strategies have long been an effective means of efficiently parallelizing large-scale linear problems [Quarteroni et al. 1999]. Direct connections with the Schur complement method and block-decomposed iterative solvers for linear algebra are then easily established for faster solves. These methods offer exciting opportunities for scalable performance that have also been applied in graphics [Huang et al. 2006; Kim and James 2012; Liu et al. 2016; Sellán et al. 2018], but can also suffer from slower convergence or gapping between imperfectly joined interfaces [Kim and James 2012]. Domain decompositions, including the classic Schwartz methods, have also been extended to the nonlinear regime [Dolean et al. 2015; Xiao-Chuan and Maksymilian 1994]. Here parallelization is easily obtained; however, methods generally offer linear convergence rates [Dolean et al. 2015]. While speeds can potentially be further improved by incorporating ADMM-type strategies [Parikh and Boyd 2012], overall application is still hampered by ADMM's underlying first-order convergence and the overhead of working with additional dual variables [Boyd et al. 2011]. For DOT, we design a domain decomposition without dual variables that employs energy-aware coupling penalty terms in the subdomain Hessian proxy that can be efficiently and easily parallelized for evaluation. We then integrate this model as an inner component of a customized limited memory BFGS to gain higher-order coupling across domains and so regain super-linear convergence with a small, additional fixed linear overhead.

Optimization-based time integrators in graphics. Position-based dynamics methods have become an increasingly attractive option in animation, being fast and efficient, but not controllable nor consistent [Müller et al. 2007]. Both Projective Dynamics (PD) [Bouaziz et al. 2014] and extended position-based dynamics [Macklin et al. 2016] observe that with small but critical modifications, iterated local and global solves can be brought more closely into alignment with implicit Euler time stepping, although various limitations in terms of consistency, controllability, and/or materials remain. Liu and colleagues [2017] observe that, in the specific case of the ARAP energy density function, PD is exactly a Sobolev-preconditioning, or in other words, an inverse-Laplacian-processed gradient descent of the incremental potential for implicit Euler. Note that beyond ARAP this analogy breaks down. Following on this observation they propose extending PD's Sobolev-preconditioning of the implicit Euler potential for models with a range of hyperelastic materials and thus varying distortion energies. They further improve convergence of this implicit-Euler solver by wrapping the Sobolev-preconditioner as an initializer for the limited-memory BFGS algorithm (L-BFGS) to minimize the incremental potential. In the following we refer to this algorithm as LBFGS-PD.

Concurrently, Overby et al. [2016; 2017] observe that PD can alternately be interpreted as a particular variant of ADMM with local variables formulated in terms of the deformation gradient. Overby and colleagues then show that an algorithm constructed in this way can likewise be extended to minimize the implicit Euler potential over a range of hyperelastic materials. In the following we will refer to this algorithm as ADMM-PD. Both ADMM-PD and LBFGS-PD improve upon PD both in extending to a wide range of hyperelastic materials and to increasing efficiency for ARAP-based deformation [Liu et al. 2017; Overby et al. 2017].

Optimization-based time integration. In brief, albeit by a circuitous path, time stepping solvers in computer graphics, starting from position-based methods, come full circle back to minimizing the standard incremental potential for fully nonlinear materials. In this setting, traditional time stepping in computational mechanics generally focuses on preconditioned gradient-descent methods, often augmented with line-search [Ascher 2008; Deuflhard 2011]. These methods find local descent directions p_i, at each iterate i, by preconditioning the incremental potential's gradient with a Hessian proxy H_i, so that $p_i = -H_i^{-1}\nabla E(x_i)$. For example, Sobolev-preconditioning [Neuberger 1985] for implicit Euler gives a fixed efficient preconditioner built with the Laplacian L so that $H_i = M + h^2L$ for all time steps [Liu et al. 2017].

Newton-type methods. Preconditioning with the Hessian $H_i = \nabla^2 E(x_i)$, gives Newton-stepping. To gain descent for large-time stepping this generally requires both line search and a positive-definite fix of the stiffness matrix which otherwise can make the total Hessian indefinite [Nocedal and Wright 2006]. Many closely related positive-definite corrections are suggested [Nocedal and Wright 2006; Shtengel et al. 2017; Teran et al. 2005]. Here, following Liu and colleagues, we employ Teran et al.'s per-element projection and refer to this method as Projected Newton (PN). Newton-type methods like these have rapid convergence but solution of the Hessian, either directly or via Newton-Krylov variations, are generally reported as too costly per-iteration to be competitive with less-costly preconditioners and scale poorly [Brown and Brune 2013; Liu et al. 2017; Overby et al. 2017]. We revisit and analyze this assumption in §5 with a suite of well-optimized solvers. We observe that in some ranges PN with a direct solve is actually competitive and can even outperform LBFGS-PD and ADMM-PD, while in others the situation is indeed often reversed. In order to retain Newton's super-linear convergence with improved efficiency, lagged updates of the Hessian preconditioner every few time steps have long been proposed [Deuflhard 2011; Hecht et al. 2012; Shamanskii 1967]. While in some cases this can be quite effective, the necessary frequency of these updates can not be pre-determined as it depends on local simulation state, e.g., transitions, collisions, etc, and so generally leads to unsightly artifacts and inaccuracies, including ghost forces and instabilities [Brown and Brune 2013; Liu et al. 2017].

Quasi-Newton methods. Alternately, quasi-Newton BFGS methods have long been applied for simulating large-deformation elastodynamics [Deuflhard 2011]. L-BFGS can be highly effective for

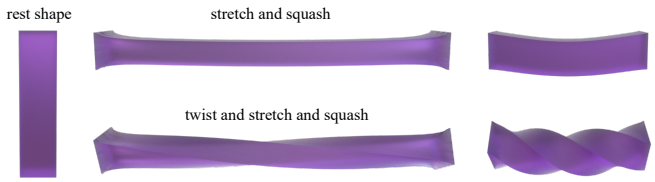

Fig. 2. ***Uniform deformation examples.***

minimizing potentials. However, an especially good choice of initializer is required and makes an enormous difference in convergence and efficiency [Nocedal and Wright 2006]; e.g., Liu et al.'s [2017] Sobolev-initializer. Directly applying the Hessian is of course clearly an ideal initializer for L-BFGS; unfortunately, it is generally a too costly option. Lazy updates of the Hessian, for example at start of time step, have also been considered [Brown and Brune 2013; Liu et al. 2017]; but again have been discarded as too costly and limiting in terms of scalability [Liu et al. 2017]. In the following we will refer to this latter method as LBFGS-H. Our development of DOT leverages these start of time step Hessians at the beginning of each incremental potential solve, applying a new, inner domain decomposition strategy to build an efficient, per-time step re-initialized method that outperforms or closely matches best-per-example prior methods across all tested cases.

Trade-offs across time-integration solvers. As both LBFGS-PD and ADMM-PD appeared concurrently and, to our knowledge, have not previously been analyzed side-by-side, we do so here for the first time along with PN and LBFGS-H, across a range of examples. We show how all these methods alternately excel or fall-short in varying criteria and examples; see §5. ADMM-PD has fast initial convergence that rapidly trails off, characteristic of ADMM methods, and thus is generally slowest, often failing to meet even moderate accuracy tolerances. LBFGS-PD on the other hand performs admirably when a global, uniform deformation is applied on a uniform mesh such as in bar stress-tests, see e.g., Fig. 2. For these cases the Laplacian preconditioner effectively smoothes global error generating rapidly converging simulations [Zhu et al. 2018]. However, for everyday non-uniform deformations on unstructured meshes common in many application, LBFGS-PD will rapidly lose efficiency. Often, as deformation magnitude grows, LBFGS-PD becomes slower than a direct application of PN and LBFGS-H, despite LBFGS-PD's per-iteration efficiency; see §5.

Based on our analysis of where these prior methods face difficulties we propose DOT which efficiently, robustly, and automatically converges to user-designated accuracy tolerances with timings that outperform or closely match these previous methods across a wide range of practical stress-test deformation scenes on unstructured meshes. DOT combines the advantages of per-time step updates of second-order information with an efficient quasi-Newton update of per-iteration curvature information. The two are integrated together by a novel domain-decomposition we construct that avoids slow coupling convergence challenges faced by traditional domain decomposition methods. Together in DOT these components form the basis for a scalable, highly effective, domain-decomposed, limited memory quasi-Newton minimizer of large time step incremental potentials with super-linear convergence; see §5.

4 METHOD

We solve each time step by iterated descent of the incremental potential. At each inner iteration i, we calculate a descent direction p_i and a line search step, $\alpha_i \in \mathbb{R}_+$. We then update our current estimate of position at time $t+1$ by $x_{i+1}^{t+1} \leftarrow x_i^{t+1} + \alpha p_i$.

4.1 Limited memory quasi-Newton updates

We begin with a quasi-Newton update. At each iterate i, the standard BFGS approach [Nocedal and Wright 2006] exploits the secant approximation from the difference in successive gradients, $y_i = \nabla E(x_{i+1}) - \nabla E(x_i)$, compared to the difference in positions $s_i = x_{i+1} - x_i$. This approximation is applied as low-rank updates to an inverse proxy matrix $D_i = H_i^{-1}$ (roughly approximating $\nabla^2 E^{-1}$) so that $D_{i+1} y_i = s_i$. Limited memory BFGS (L-BFGS) then stores for each iteration i just an initial, starting matrix, D_1 and the last m $\{s, y\}$ vector pairs (we use $m = 5$). Joint update and application of D_{i+1} is then applied *implicitly* with just a few, efficient vector dot-products and updates, with the application of the matrix D_1 sandwiched in between the first and second low-rank updates of each step.

Given iterate i's implicitly stored proxy D_i, and the initial proxy, D_1, we set $Q_i : \mathbb{R}^{dn \times dn} \times \mathbb{R}^{dn} \rightarrow \mathbb{R}^{dn}$ to apply the limited-memory quasi-Newton update and application of D_{i+1}. Then

$$p_i = Q_i(D_1, x_i) = -D_{i+1} \nabla E(x_i). \tag{4}$$

L-BFGS's performance depends greatly on choice of initializer D_1 [Nocedal and Wright 2006]. Setting D_1 to a weighted diagonal, for example, offers some improvement, as do various inverses of block diagonals taken from the Hessian. Similarly we can, as discussed above, apply Liu et al.'s [2017] inverse Laplacian or even, as proposed by Brown and Bune [2013], could use lagged updates of the Hessian inverse itself — just once every few time steps so as to not perform too many expensive updates and solves.

4.2 Initalizing L-BFGS

To consider the relative merits of potential initializers for time stepping we adopt a simple perspective. We observe that each of the above initializers corresponds to the application of a single iteration of a different, nonlinear method as an inner step in the L-BFGS loop [Zhu et al. 2018]. The better the convergence of the inner method generally the better performance of the outer L-BFGS method. From this perspective Liu et al. [2017], for example, can be seen to apply an inner iterate of inverse-Laplacian preconditioned descent and so gain the well-appreciated smoothing of the parent Sobolev Gradient Descent method [Neuberger 1985]. Or alternately we could consider applying the start of time step inverse Hessian $D_1 = \nabla^2 E(x^t)^{-1}$. The resulting optimization applies an iteration of the inexact Newton method within L-BFGS that could potentially augment quasi-Newton curvature with second-order information via the tangent stiffness matrix.

While this latter strategy could be highly effective to improve convergence [Liu et al. 2017], it is expensive both to factorize at every time step and to backsolve at every iteration. Moreover, we observe that initializing with the full Hessian inverse may be an unnecessarily global approach. In many cases large deformations are concentrated locally and their effects are only communicated across the entire material domain over multiple time steps. Likewise,

initializing with the full inverse Hessian limits scalability as we must work with its factors in every quasi-Newton iterate; see §5.

Motivated by these observations we instead design a method to update L-BFGS with Hessian information using a decomposition that allows us to efficiently store and apply local second-order information at each iterate. To do so we first construct a simple quadratic penalty decomposition that connects non-overlapping subdomains with automatically determined stiffness weights. Our decomposed optimization time integrator is then formed by applying a single inexact-Newton descent iterate of this method as an inner initializer of an undecomposed, full mesh L-BFGS step. The resulting method, as we see §5, then obtains well-improved, scalable convergence for large time step, large deformation simulations.

4.3 Decomposition

An illustration of our decomposition is in Fig. 3. We decompose our full domain Ω into s non-overlapping subdomains $\{\Omega_1, ..., \Omega_s\}$ partitioned by interfaces $\Gamma = \{\gamma_1, ..., \gamma_{|\Gamma|}\}$ along element boundaries with duplicate copies of each interface vertex assigned to each subdomain participating at that interface. We construct decompositions that both approximately minimize number of edges along interfaces (and so cross-subdomain communication) and balance numbers of nodes per subdomain [Karypis and Kumar 1998]. For each subdomain j we then likewise store its interfaces in $\Gamma_j \subseteq \Gamma$.

Each subdomain j then has n_j vertices stored in vector $x_j = (y_j^T, z_j^T) \in \mathbb{R}^{dn_j}$, formed by the concatenation of a copy of its interface vertices z_j, and its remaining subdomain vertices, y_j. Concatenation of all subdomain vertices $\hat{x} = (x_1^T, ..., x_s^T)^T \in \mathbb{R}^{d\hat{n}}$ then includes unshared interior vertices and duplicated interface vertices so that $\hat{n} > n$. We then collect interface vertices from the fully connected mesh as $x_\Gamma \subset x$. Note that x_Γ are then distinct from their duplicated interface copies in the decomposition and can serve as consensus variables for subdomain boundaries. Finally, per subdomain j, we construct interface restriction matrices R_{Γ_j} that extract vertices from x_Γ participating in that subdomain's interfaces Γ_j.

4.4 Penalty Potential

Building time stepping incremental potentials on this decomposed system we get a nicely separable sum $\sum_{j\in[1,s]} E_j(x_j)$, where $E_j = E|_{\Omega_j}$ is the restriction of the incremental potential to subdomain j. This separable potential could be efficiently optimized as it allows us to minimize each subdomain independently. Doing so, however, would erroneously decouple subdomains.

One possibility for reconnecting subdomains would be to add explicit coupling constraints. However, this would also require dual variables in the form of Lagrange-multipliers. As we are designing

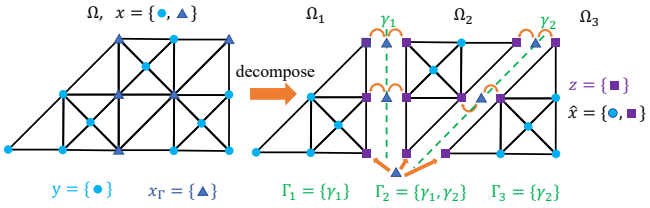

Fig. 3. ***DOT's decomposition.*** Layout and notation for a mesh (left) which is decomposed into three subdomains (right).

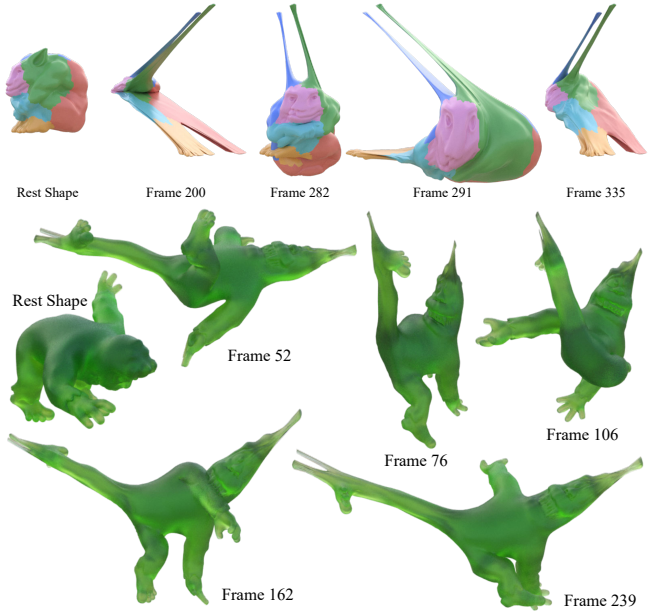

Fig. 4. ***Deformation stress-tests.*** DOT simulation sequences of hollow cat (top) and monkey (bottom) large-deformation, high-speed, stress-tests.

our decomposition for insertion within a quasi-Newton loop, additional dual variables are undesirable. Instead, we adopt a simple quadratic penalty for each interface in Γ to pull subdomains together, with consensus variables x_Γ as the bridge. Augmenting the above separable potential with this penalty gives

$$L(\hat{x}, x_\Gamma) = \sum_{j\in[1,s]} \left(E_j(x_j) + \frac{1}{2}\|z_j - R_{\Gamma_j} x_\Gamma\|^2_{K_j} \right). \tag{5}$$

Here each K_j is a penalty stiffness matrix that pulls subdomain j's interface vertices towards globally shared consensus positions stored in x_Γ. By driving $|K_j|$ towards ∞ as we optimize L we could hypothetically remove interface mismatch. However, finding practical values for K_j that sufficiently pull interface edges together, while avoiding ill-conditioning, is generally challenging. For example, if we choose a penalty that is too stiff it would pull interface domains together too tightly at the expense of overwhelming elasticity and inertia terms in the potential. On the other hand, if we choose a penalty that is too soft, subdomain potentials dominate and interface consensus would be underestimated.

4.5 Interface Hessians

We instead seek a penalty that automatically balances out the missing elastic and inertial information along interfaces. In our setting we observe that this missing information is directly available and forms a natural, automated weighting function for the augmented potential in (5) above. Consider that each subdomain's penalty term is effectively just a proxy for the missing energies from neighboring subdomains. Specifically, the Hessian for subdomain Ω_j is

$$\frac{\partial^2 L}{\partial {x_j}^2} = \begin{bmatrix} \frac{\partial^2 E_j(x_j)}{\partial {y_j}^2} & \frac{\partial^2 E_j(x_j)}{\partial y_j \partial z_j} \\ \frac{\partial^2 E_j(x_j)}{\partial z_j \partial y_j} & \frac{\partial^2 E_j(x_j)}{\partial {z_j}^2} + K_j. \end{bmatrix} \tag{6}$$

Here, the upper and off-diagonal (symmetric) terms nicely match the unrestricted Hessian on the full domain Ω,

$$\frac{\partial^2 E_j(x_j)}{\partial {y_j}^2} = \frac{\partial^2 E(x)}{\partial {y_j}^2} \text{ and } \frac{\partial^2 E_j(x_j)}{\partial y_j \partial z_j} = \frac{\partial^2 E(x)}{\partial y_j \partial (R_{\Gamma_j} x_\Gamma)}. \tag{7}$$

However, our lower diagonal does not as

$$\frac{\partial^2 E_j(x_j)}{\partial {z_j}^2} \neq \frac{\partial^2 E(x)}{\partial (R_{\Gamma_j} x_\Gamma)^2}, \tag{8}$$

irrespective of whether or not we have agreement on subdomain interface vertex locations. This is due to the absence of element stencils connecting interface nodes to neighboring subdomains. While this missing information is not currently present in $\partial^2 L/\partial {x_j}^2$, it gives us a natural definition for penalty weights. We assign weights K_j so that they generate the missing interfacial Hessian information from adjacent elements bridging across neighboring subdomains:

$$K_j(x) = \frac{\partial^2 E(x)}{\partial (R_{\Gamma_j} x_\Gamma)^2} - \frac{\partial^2 E_j(x_j)}{\partial {z_j}^2}. \tag{9}$$

Our quadratic penalty then recovers otherwise missing components of kinetic and elastic energy stencils across interface boundaries. Note that, as in PN, we project all computed Hessian stencils.

4.6 Discussion

To minimize (5) we could apply alternating iterations solving first for optimal $\hat{x}$ and then for x_Γ. Indeed, when we add a constraint Lagrangian term, $\sum_{j\in[1,s]} \lambda_j^T (z_j - R_{\Gamma_j} x_\Gamma)$ to (5), this alternating process generates a custom ADMM [Boyd et al. 2011] algorithm. We initially considered this approach and find that it significantly outperforms classic ADMM. We observe that automatic weighting with (9) nicely smooths error at boundaries; see e.g. Fig. 8. However, we also find that this strategy is still not competitive with undecomposed methods like PN, as too much effort is exerted to close gaps between subdomains. Instead, we apply a single iteration of our penalty decomposition as an efficient, inner initializer with second-order information. We then insert it within an outer, undecomposed quasi-Newton step solved on the full mesh. This ensures unnecessary effort is not spent pulling interfaces together, and updates our decomposition with global, full-mesh, curvature information.

4.7 Initializer

We now have all necessary ingredients to construct DOT. We initialize an L-BFGS update with a single Newton iteration of our quadratic penalty in (5) as follows. At start of time step $t+1$ we define subdomain variables from current positions x^t as $\hat{x}^t = Sx^t$. Here $S \in \mathbb{R}^{d\hat{n}\times dn}$ is a separation matrix that maps the n full-mesh vertices x to subdomain coordinates with duplicated copies of interfacial vertices. Application of the inverse Hessian is then

$$D_1^{t+1} = BS^T \left(\partial^2 L(\hat{x}^t, x_\Gamma^t)/\partial \hat{x}^2\right)^{-1} S \in \mathbb{R}^{dn\times dn}. \tag{10}$$

Here $B \in \mathbb{R}^{dn\times dn}$ is a diagonal averaging matrix with diagonal entries corresponding to vertex v set to $1/n_v$ where n_v is the number of duplicate copies for the corresponding vertex v in the decomposition. Iteration i of DOT is then applied by application of

$$p_i = Q_i(D_1^{t+1}, x_i), \tag{11}$$

followed by our custom line search and update detailed below.

4.8 Construction

Construction and application of the per time step DOT is efficient. At start of solve we first compute and store factors of the augmented Hessians per subdomain,

$$H_j = \nabla^2 E_j(x_j^t) + K_j(x_j^t). \tag{12}$$

We then observe that

$$D_1^{t+1} = BS^T \text{diag}(H_1^{-1}, ..., H_s^{-1})S, \tag{13}$$

where diag$(\cdot)$ constructs a block diagonal $\mathbb{R}^{d\hat{n}\times d\hat{n}}$ matrix. Application of D_1^{t+1} as initializer in L-BFGS is then applied in parallel computation to any global vector $q \in \mathbb{R}^{dn}$. We first separate q to repeated subdomain coordinates $(q_1^T, ..., q_s^T)^T = Sq \in \mathbb{R}^{d\hat{n}}$. Then we independently backsolve subdomains with their factors to obtain $r_j = H_j^{-1} q_j$. Finally, we lift all r_j back to full mesh coordinates by blending with $r = BS^T(r_1^T, ..., r_s^T)^T$ to average duplicate coordinates appropriately. See Algorithm 1 below for the full DOT pseudocode.

4.9 Line Search

After our quasi-Newton update we next perform backtracking line search on p_i to ensure sufficient descent. For quasi-Newton methods rule-of-thumb [Nocedal and Wright 2006] is to always initialize line search with unit step length, i.e. $\alpha_{\text{start}} = 1$, to ensure large steps will take advantage of rapid convergence near solutions. In the large time step, large deformation setting however, we observe that the situation is nonstandard. We often start far from solutions and so need to balance large, initial step estimates against costs of repeated energy evaluations. For this purpose we apply an alternative initializer for line search. For each search direction p_i, DOT initializes with the optimal length of the one-dimensional quadratic model,

$$\alpha_{\text{start}} = \max\left(10^{-1}, \frac{-p_i^T \nabla E(x_i)}{p_i^T \nabla^2 E(x^t) p_i}\right). \tag{14}$$

Here the lower bound handles the rare instances where this local fit is too conservative.

4.10 Algorithm

Algorithm 1 contains the full DOT algorithm in pseudocode. The dominant costs for runtime are energy costs: evaluations, gradients, Hessians, and SVDs; and subdomain augmented Hessian costs: assembly, factorization and backsolves. Otherwise, costs for our quasi-Newton loop itself are linear (dot products, vector updates, etc). Memory cost is primarily the once per-timestep Cholesky factorization of the subdomain augmented Hessians and their corresponding backsolves per iteration.

5 EVALUATION

5.1 Implementation and Testing

We implemented a common test-harness code to enable the consistent evaluation of all methods with the same optimizations for common tasks. In comparisons of our ADMM-PD and LBFGS-PD implementations with their release codes [Liu et al. 2017; Overby et al. 2017] we observe an overall 2-3X speed-up across examples.

ALGORITHM 1: Decomposed Optimization Time Integrator (DOT)

Given: x^t, E, S, B, ϵ
Initialize and Precompute:
  $i = 1$
  $H_j^{-1} \leftarrow \left(\nabla^2 E_j(x_j^t) + K_j(x^t)\right)^{-1}, \forall j \in [1, s]$ // get Cholesky factors
  $g_1 \leftarrow \nabla E(x_1)$
// quasi-Newton loop to solve time step $t + 1$:
while $\|g_i\| > \epsilon h^2 \langle W \rangle \|\ell\|$ **do** // termination criteria (§5.2)
  $q \leftarrow -g_i$
  for $a = i - 1, i - 2, .., i - m$
    $s_a \leftarrow x_{a+1} - x_a, \quad y_a \leftarrow g_{a+1} - g_a, \quad \rho_a \leftarrow 1/(y_a^T s_a)$
    $\alpha_a \leftarrow \rho_a s_a^T q$
    $q \leftarrow q - \alpha_a y_a$
  end for
  $(q_1^T, \ldots, q_s^T)^T \leftarrow Sq$
  $r_j \leftarrow H_j^{-1} q_j, \forall j \in [1, s]$
  $r \leftarrow BS^T (r_1^T, \ldots, r_s^T)^T$
  for $a = i - m, i - m + 1, .., i - 1$
    $\beta \leftarrow \rho_a y_a^T r$
    $r \leftarrow r + (\alpha_a - \beta) s_a$
  end for
  $p_i \leftarrow r$
  $\alpha_{\text{start}} \leftarrow \max\left(10^{-1}, -\left(p_i^T \nabla E(x_i)\right)/\left(p_i^T \nabla^2 E(x^t) p_i\right)\right)$
  $\alpha \leftarrow \text{LineSearch}(x_i, \alpha_{\text{start}}, p_i, E)$
  $x_{i+1} \leftarrow x_i + \alpha p_i$
  $g_{i+1} \leftarrow \nabla E(x_{i+1})$
  $i \leftarrow i + 1$
end while

Our code is implemented in C++, parallelizing assembly and evaluations with Intel TBB, applying CHOLMOD [Chen et al. 2008] with AMD reordering for all linear system solves, and METIS [Karypis and Kumar 1998] for all decompositions. Note that for LBFGS-PD and ADMM-PD we perform their global Laplacian backsolves in parallel, per-dimension, and likewise factorize only the scalar Laplacian, one-time as a precompute. Following Overby et al. [2017], ADMM-PD's per-element energy minimizations are performed in diagonal space for efficiency. Common energy evaluations and gradients are optimized with AVX2 parallelization to achieve roughly 4× speedup. To do so we extended the open-source SIMD SVD library [McAdams et al. 2011] to support double precision for our framework.

Unless otherwise indicated, all experiments below were performed on a six-core Intel 3.7GHz CPU and were simulated with times step sizes of either 10, 25 (majority), or 40 ms (as indicated). For consistent comparison with prior work we focus our analysis on examples with the implicit Euler using the fixed co-rotational material [Stomakhin et al. 2012]. Below in §5.6 we also explore DOT with the Stable Neo-Hookean material [Smith et al. 2018].

We compile CHOLMOD with MKL LAPACK and BLAS, supporting multi-threaded linear solves, and set the number of threads per linear solver to take full advantage of multi-core architecture per method. Specifically, the single, full linear systems in each PN and LBFGS-H iteration are solved with 12 threads per solver, while the multiple smaller systems in each LBFGS-PD, ADMM-PD, and DOT iteration are solved simultaneously with 1 thread per solver.

Finally, note that we summarize detailed statistics from all of our experiments in tables in our supplemental. All tables referred to in the following are found there.

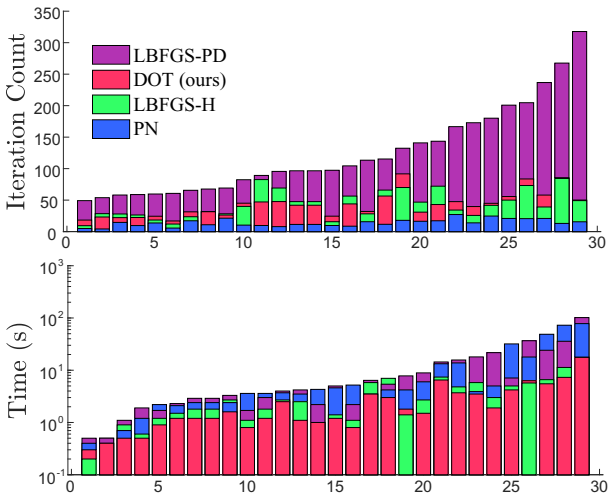

Fig. 5. ***Convergence and Timings.*** Per-example comparisons of iteration (top) and timing (bottom) costs, per frame, achieved by each time step solver. Note that the bars are overlaid, not stacked.

5.2 Termination Criteria

We next focus on the important question of when to stop iterating an individual time step solve. Clearly, for most simulation applications, it is not reasonable to manually monitor the quality of each individual iterate, within every individual time step solve, in order to decide when to stop. Analogously, applying the same fixed number of iterations for all time steps, no matter the method, will not suffice as different steps in a simulation will have more or less nonlinearity involved and so will require correspondingly different amounts of work to maintain consistent simulation quality. Without this simulations can and will accrue inconsistencies, artifacts and instabilities. Similarly, relative error measures for termination are not satisfactory because they are fulfilled when an algorithm is simply unable to make further progress and so has stagnated far

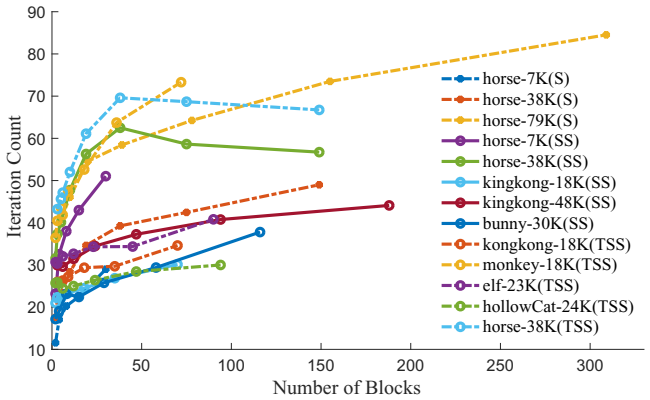

Fig. 6. ***Subdomain to iteration growth.*** We plot iterations per DOT simulation example as the size of the decomposition increases. We observe a trend of sublinear-growth in iteration count with respect to the number of subdomains, revealing promising opportunities for parallelization.

from a working solution. On the other hand, the gradient norm provides an excellent measure of termination for scientific computing problems where we seek high-accuracy. However, for animation and many other applications we seek a way to reliably stop at stable, good-looking, consistent simulations with reasonably low error, *but* not necessarily with vanishingly small gradients. As observed by Zhu et al [2018], in these cases even gradient norm tolerances are not suitable for choosing a termination criteria.

To address this problem for minimizing deformation energies in statics problems, Zhu and colleagues propose a dimensional analysis of the deformation energy gradient. They derive a characteristic value for the norm of the deformation gradient over the mesh. Then, applying it as a scaling factor to the gradient norm obtains consistent quality solutions across a wide range of problems, object shapes and energies. Here we observe that a small modification of this analysis gives a corresponding scaling factor for the incremental potential. We begin with the base case of the characteristic value for the norm of the deformation gradient over the mesh as $\langle W \rangle ||\ell||$. Here $\langle W \rangle$ is the norm of the deformation energy Hessian of a single element at rest, ℓ is a vector in $\mathbb{R}^n$ with each entry the surface area of each simulation node's one-ring stencil. For dynamic time stepping we are then minimizing with the incremental potential (3) and so have a weighted sum of discrete kinetic and deformation energies. At stationarity of (3) we then have $M(x - x^t - x^p) + h^2 \nabla W(x) = 0$. We then have proportional measures $-\frac{1}{h^2} M(x - x^t - x^p)$ at corresponding scale with $\nabla W(x)$, and similarly $\frac{1}{h^2} M(x - x^t - x^p) + \nabla W$. Then, for consistent convergence checks across examples and methods at each iteration we simply check the rescaled incremental potential

$$||\nabla E|| < \epsilon h^2 \langle W \rangle ||\ell||. \tag{15}$$

In the following evaluations we refer to this measure as the characteristic gradient norm (CN). After extensive experimentation across a wide range of examples, using Projected Newton (PN) as a baseline, we find consistent quality solutions across methods using (15) at increments of ϵ; see our supplemental videos. Moreover, we find that solutions satisfying $\epsilon = 10^{-5}$ are the first to avoid visual artifacts that we consistently observe below this tolerance, including variable material softening, damping, jittering and explosions. For all experiments in the remainder of this section, unless otherwise indicated, we thus set our termination criteria with $\epsilon = 10^{-5}$ using (15). All CN convergence measures thus apply rescaling of the gradient norm. Convergence in CN per method then confirms convergence to the same (reference) solution provided, e.g., by Newton's method.

5.3 Iteration Growth with Domain Size

We next investigate the scalability of DOT as the number of subdomains in the decomposition grows. We apply DOT across thirteen simulation examples ranging in mesh resolution from 7K to 136K vertices with a range of moderate to extreme deformation test scenes. Each scene is solved to generate ten simulated seconds, time stepped at $h = 25$ms. For each simulation example we create a sequence of decompositions by requesting target subdomain sizes starting at 16K vertices (where possible) down by halves to 256 vertices. We then simulate all of the resulting decompositions, spanning from 2 to 309 subdomains, with DOT. In figure 6 we plot the the number of DOT iterations per simulation example as the size of its decomposition increases. We observe a trend of sublinear-growth in iteration count (per simulation) as the number of subdomains increases, revealing promising opportunities for parallelization of DOT.

5.4 Performance

Fig. 6 suggests parallelization opportunities for DOT across a wide range of decomposition sizes. Of course, how to best exploit these opportunities will vary greatly with platform. Here we begin with two modest exercises starting with a six-core Intel 3.7GHz CPU, 64GB memory. We script a set of increasingly challenging dynamic deformation stress-test scenarios across a range of mesh shapes and resolutions. See, for example, Fig.s 1 and 4, our supplemental and video for example details. For each simulation we target DOT to simply utilize available cores and so set the number of subdomains in all simulations in this first exercise to six. In Fig. 5 and Tables 1-3 (supplemental) we summarize runtime statistics for these examples with DOT, PN, LBFGS-H, and LBFGS-PD across the full set of these examples. We also test ADMM-PD on the full set of examples but find it unable to converge on any in the set. See our convergence analysis below for more details on ADMM-PD's behavior here.

Timings. Across this set we observe DOT has the fastest runtimes, for all but three examples (see below for discussion of these), over the best timing for each example across all converging methods: PN, LBFGS-H, and LBFGS-PD. In general DOT ranges from 10X to 1.1X faster than PN, from 2.5X to 1X faster than LBFGS-H, and from 11.4X to 1.6X faster than LBFGS-PD. The one exception we observe is in the three smallest meshes of the horse stretch scalability example where the deformation is slow, so that the solves are close to statics. Here we observe that LBFGS-H is, on average 1.3X faster than DOT on smaller meshes up to 79K vertices. Then, as mesh size increases to 136K and beyond, here too DOT becomes faster. Finally, for even larger meshes LBFGS-H can not fit in memory; see *Scaling* below. Importantly, across examples, we observe that PN, LBFGS-PD and LBFGS-H alternate as fastest as we change simulation example. Here PN ranges from 2X slower to 4.6X faster than LBFGS-PD, while LBFGS-H often seems to be a best choice among the three, but also can be slowest, i.e., ranging from 1.1X to 1.7X slower. Trends here suggest that LBFGS-PD tends to do better for more moderate deformations while PN and LBFGS-H often pull ahead for more extreme deformations but this is not entirely consistent and it is challenging to know which will be the better method per example, a priori. Finally, as we see below, both PN and LBFGS-H do not scale well to larger systems.

Scaling. To examine scaling we successively increase mesh resolution for the horse TSS example. On this machine (recall 64GB memory) both PN and LBFGS-H can not run models beyond 308K vertices while DOT and LBFGS-PD can continue for examples up to and including 754K vertices. We compare the performance among methods at these two extremes and find that DOT is 1.9X faster than LBFGS-H at 308K nodes, while it is 2.7X faster than LBFGS-PD at 754K nodes, where both PN and LBFGS-H can not run.

Changing Machines. Next, in Table 4, we report statistics as we exercise DOT on both our six-core machine and a sixteen-core Xeon

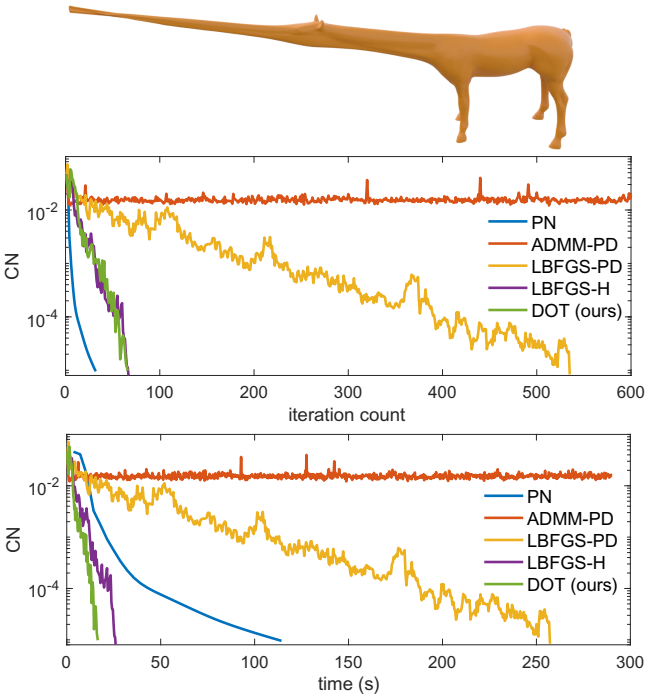

Fig. 7. ***Convergence comparisons. Top:*** We compare the convergence of methods for a single time step midway through a stretch script with a 138K vertex mesh; measuring error with the characteristic gradient norm (CN) see §5.2. ***Middle:*** We observe DOT's super-linear convergence matches LBGHS-H and closely approaches Project Newton's (PN), while LBFGS-PD lags well behind and ADMM-PD does not converge. ***Bottom:*** comparing timing, DOT pulls ahead of PN and LBFGS-H, with lower per-iteration costs.

2.4GHz CPU. Here we correspondingly set the number of subdomains in all simulations to six and sixteen respectively. Although overall timings of course change, we see that DOT similarly maintains the fastest runtimes across both machines, over the best timing for each example between PN, LBFGS-H and LBFGS-PD. Here these three latter methods again swap one another in speeds per example.

Changing Decompositions. Then, to confirm that there is a wide range of viable subdomains settings for DOT, we examine performance as we vary subdomain sizes using the same simulation example set from §5.3 above. In Table 5 we summarize statistics for these simulations and observe that all simulations match or out-perform the best result timing between PN, LBFGS-H and LBFGS-PD; the only exceptions being the smallest 7K mesh horse simulations are slightly outperformed by PN and LBFGS-H.

5.5 Convergence

DOT balances efficient, local second-order updates with global curvature information from gradient history. In Fig. 7 we compare convergence rates and timings across methods for a single time step midway through the large stretch example of the 138K vertex horse mesh. We observe super-linear convergence for DOT, matching LBFGS-H's and closely approaching PN's, while LBFGS-PD lags well behind, and ADMM-PD characteristically does not converge to even a much lower tolerance than the one requested. In turn, comparing timings, DOT out-performs PN and LBFGS-H with lower

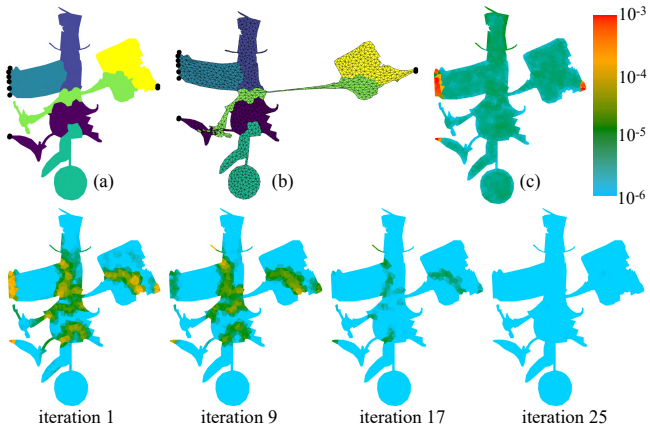

Fig. 8. ***Residual Visualization.*** With a decomposition (a), current deformation in (b), and starting error in (c), we visualize DOT's characteristic convergence process. In the first few iterations error is concentrated at interfaces and is then rapidly smoothed by the successive iterations.

per-iteration cost. In Fig. 8 we visualize DOT's characteristic process: in the first few iterations error is concentrated at interfaces and is then quickly smoothed out by the successive iterations.

In addition to PN, LBFGS-PD, LBFGS-H and ADMM-PD, we also investigated standard Jacobi and Gauss-Seidel decomposition methods [Quarteroni et al. 1999], and experimented with applying incomplete Cholesky as initializer for L-BFGS. Convergence rates for the former two methods are slow, making them impractical compared to the above methods. The latter method, interestingly, sometimes performs well, but is inconsistent as it also can be slow and even fails to converge in other cases; see our supplemental for details.

5.6 Varying Time Step, Material Parameters and Model

Here we compare behavior as we change material parameters and vary over a range of frame-size time steps. We apply a twist, stretch and squash (TSS) script to an 18K vertex monkey mesh. For the same example we apply time step sizes of 10, 25 and 40 ms respectively and vary material parameters comparing across a range of Young's modulus and Poisson ratio. As summarized in Table 2, across all examples DOT obtains the fastest runtimes with trends showing timings increasing for all methods as we increase time step size and Poisson ratio. For varying stiffness, timings for DOT stay largely flat, while here PN, LBFGS-H, and LBFGS-PD become slower with softer materials. Finally, we consider changing the material model. We apply the Stable Neo-Hookean model to run the monkey TSS example. Relative timings stays consistent as with the FCR model, where DOT ranges from 1.5X to 2.4X faster than all alternatives.

6 CONCLUSION

In this work we developed DOT, a new time step solver that enables efficient, accurate and consistent frame-size time stepping for challenging large and/or high-speed deformations with nonlinear materials. So far we have focused on CPU parallelization on moderate commodity machines with medium-scale meshes ranging from 31K to 4.2M tetrahedra. However, as we see in §5.3 above, DOT's sub-linear scaling of iterations for more decompositions makes extensions of DOT to large-scale systems exceedingly promising to

pursue. Concurrently for meshes at all scales we observe that while rule-of-thumb matching domain count to available cores already exposes significant speed-up and robustness we have also seen in §5 that across a wide range of decomposition sizes we maintain a significant and consistent advantage. Thus we are also excited to explore DOT with recent advances in batch-processed factorizations and solves on the GPU, e.g., with MAGMA [Abdelfattah et al. 2017], where L-BFGS low-rank updates can be efficiently performed via map reduce. Likewise, while DOT offers speed and robust convergence at large time step, decomposition also offers other promising opportunities. One exciting direction is applying recent mesh-adaptation strategies [Schneider et al. 2018] which can now be performed on-the-fly, independently per subdomain.

When deformations are mild, uniformly distributed, at slow speeds and/or on small meshes we see that the win for DOT is sometimes not as significant. If such cases are to be expected then certainly alternatives such as PN, LBFGS-H, LBFGS-PD and ADMM-PD may be reasonable choices as well. Our experience suggests that most scenarios are not likely to limit the scope of a simulation tool to these cases. If so, then we propose DOT as a one-size-fits-all method that improves or closely matches performance in these easier and gentler cases and then shines for the challenging scenarios were previous methods become stuck and/or exceedingly slow.

Our evaluation of DOT has so far focused on invertible energies. Efficiency with noninverting energies, e.g., Neo-Hookean, may require custom-handling of the elasticity barrier, e.g, by line search curing [Zhu et al. 2018], and is an interesting future direction.

DOT solves elastodynamics for both rapidly moving and fixed boundary conditions over a wide range of simulation conditions. As in ADMM-PD and LBFGS-PD, integrating standard collision resolution methods using penalty or hard-constraints, can be directly encorporated in DOT. However, custom-leveraging DOT's structure for efficient contact processing remains exciting future work.

Finally, while our decompositions from METIS have been effective they are certainly not optimal for optimization-based time stepping. It would be interesting to explore custom decompositions which specifically take advantage of the incremental potential's structure and even adaptive decompositions per-time step. Likewise, while the simple strategy of matching subdomain count to cores already enables simple and easy-to-implement advantages, it is of course in no way optimal. An exciting future investigation is exploring per-task custom decompositions based on compute resources available.

ACKNOWLEDGMENTS

We thank Yixin Zhu for experiment assistance and Dan Ramirez for voice overs. This work was supported in part by the NSF (grant IIS-1755544,CCF-1813624).

REFERENCES

A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra. 2017. Fast Cholesky factorization on GPUs for batch and native modes in MAGMA. *J of Comp Sci* 20 (2017).
U. M Ascher. 2008. *Numerical methods for evolutionary differential equations.*
S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans Graph* 33, 4 (2014).
S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning* 3, 1 (2011).
J. Brown and P. Brune. 2013. Low-rank quasi-Newton updates for robust Jacobian lagging in Newton-type methods. In *Int Conf Math Comp Meth App Nucl Sci Eng.*
J. C. Butcher. 2016. *Numerical methods for ordinary differential equations.*
I. Chao, U. Pinkall, P. Sanan, and P. Schröder. 2010. A simple geometric model for elastic deformations. *ACM Trans Graph (SIGGRAPH)* 29, 4 (2010).
Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. 2008. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans on Mathematical Software (TOMS)* 35, 3 (2008).
P. Deuflhard. 2011. *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms.*
V. Dolean, P. Jolivet, and F. Nataf. 2015. *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation.*
T. Gast, C. Schroeder, A. Stomakhin, C. Jiang, and J. M Teran. 2015. Optimization integrator for large time steps. *IEEE Trans Vis Comp Graph* 21, 10 (2015).
E. Hairer, C. Lubich, and G. Wanner. 2006. *Geometric Numerical Integration.*
E. Hairer, S. P Nørsett, and G. Wanner. 2008. *Solving Ordinary Differential Equations I.*
E. Hairer and G. Wanner. 1996. *Solving Ordinary Differential Equations II.*
F. Hecht, Y. J. Lee, J. R. Shewchuk, and J. F. O'Brien. 2012. Updated Sparse Cholesky Factors for Corotational Elastodynamics. *ACM Trans Graph* 31, 5 (2012).
J. Huang, X. Liu, H. Bao, B. Guo, and H. Shum. 2006. An efficient large deformation method using domain decomposition. *Comp & Graph* 30, 6 (2006).
C. Kane, J. E Marsden, M. Ortiz, and M. West. 2000. Variational integrators and the Newmark algorithm for conservative and dissipative mechanical systems. *Int J for Numer Meth in Eng* 49, 10 (2000).
G. Karypis and V. Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J on Sci Comp* 20 (1998).
L. Kharevych, W. Yang, Y. Tong, E. Kanso, J. E Marsden, P. Schröder, and M. Desbrun. 2006. Geometric, variational integrators for computer animation. In *Symp Comp Anim.*
T. Kim and D. L James. 2012. Physics-based character skinning using multidomain subspace deformations. *IEEE Trans on visualization and Comp Graph* 18, 8 (2012).
H. Liu, N. Mitchell, M. Aanjaneya, and E. Sifakis. 2016. A scalable schur-complement fluids solver for heterogeneous compute platforms. *ACM Trans Graph* 35, 6 (2016).
T. Liu, A. W. Bargteil, J. F. O'Brien, and L. Kavan. 2013. Fast Simulation of Mass-Spring Systems. *ACM Trans Graph* 32, 6 (2013). Proc of ACM SIGGRAPH Asia.
T. Liu, S. Bouaziz, and L. Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Trans Graph* 36, 4 (2017).
M. Macklin, M. Müller, and N. Chentanez. 2016. XPBD: position-based simulation of compliant constrained dynamics. In *Proc of the 9th Int Conf on Motion in Games.*
S. Martin, B. Thomaszewski, E. Grinspun, and M. Gross. 2011. Example-based elastic materials. *ACM Trans Graph (SIGGRAPH)* 30, 4 (2011).
A. McAdams, A. Selle, R. Tamstorf, J. Teran, and E. Sifakis. 2011. Computing the singular value decomposition of 3× 3 matrices with minimal branching and elementary floating point operations. *University of Wisconsin Madison* (2011).
M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. 2007. Position based dynamics. *J. Vis. Commun. Imag Represent.* 18, 2 (2007).
R. Narain, M. Overby, and G. E Brown. 2016. ADMM ⊇ projective dynamics: fast simulation of general constitutive models.. In *Symp on Comp Anim.*
JW Neuberger. 1985. Steepest descent and differential equations. *J of the Mathematical Society of Japan* 37, 2 (1985).
J. Nocedal and S. Wright. 2006. *Numerical Optimization.*
M. Ortiz and L. Stainier. 1999. The variational formulation of viscoplastic constitutive updates. *Comp Meth in App Mech and Eng* 171, 3-4 (1999).
M. Overby, G. E Brown, J. Li, and R. Narain. 2017. ADMM ⊇ Projective Dynamics: Fast Simulation of Hyperelastic Models with Dynamic Constraints. *IEEE Trans Vis Comp Graph* 23, 10 (2017).
N. Parikh and S. Boyd. 2012. Block splitting for distributed optimization.
A. Quarteroni, A. Valli, and P.M.A. Valli. 1999. *Domain Decomposition Methods for Partial Differential Equations.*
T. Schneider, Y. Hu, J. Dumas, X. Gao, D. Panozzo, and D. Zorin. 2018. Decoupling simulation accuracy from mesh quality. *ACM Trans Graph* (2018).
S. Sellán, H. Y. Cheng, Y. Ma, M. Dembowski, and A. Jacobson. 2018. Solid Geometry Processing on Deconstructed Domains. *CoRR* (2018).
VE Shamanskii. 1967. A modification of Newton's method. *Ukrainian Mathematical J* 19, 1 (1967).
A. Shtengel, R. Poranne, O. Sorkine-Hornung, S. Z. Kovalsky, and Y. Lipman. 2017. Geometric Optimization via Composite Majorization. *ACM Trans Graph* 36, 4 (2017).
B. Smith, F. De Goes, and T. Kim. 2018. Stable Neo-Hookean Flesh Simulation. *ACM Trans Graph* 37, 2 (2018).
A. Stomakhin, R. Howes, C. Schroeder, and J. M Teran. 2012. Energetically consistent invertible elasticity. In *Symp Comp Anim.*
J. Teran, E. Sifakis, G. Irving, and R. Fedkiw. 2005. Robust Quasistatic Finite Elements and Flesh Simulation. In *Symp Comp Anim.*
C. Xiao-Chuan and D. Maksymilian. 1994. Domain Decomposition Methods for Monotone Nonlinear Elliptic Problems. In *Contemporary Math.*
Y. Zhu, R. Bridson, and D. M. Kaufman. 2018. Blended Cured Quasi-Newton for Distortion Optimization. *ACM Trans. on Graph (SIGGRAPH)* (2018).