NEAR LINEAR PERFORMANCE IN AN ARTICULATED BODY SOLVER WITH IMPLICIT ELASTICITY

A Thesis

by

MARGARET ANNE BAXTER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Shinjiro Sueda
Committee Members,	Dylan Shell
	Ergun Akleman
Head of Department,	Dima Da Silva

August 2019

Major Subject: Computer Science

ABSTRACT

The dynamics of articulated rigid bodies can be solved in O(n) time using a recursive method. When elasticity is added between the bodies, with linearly implicit integration, the stiffness matrix in the equations of motion breaks the tree topology of the system, making the recursive method inapplicable. The only alternative has been to form and solve the system matrix, which takes O(n^3) time. A new approach that can solve the linearly implicit equations of motion in near linear time, coined REDMAX, is built using a combined reduced/maximal coordinate formulation. This hybrid model enables direct flexibility to apply arbitrary combinations of forces in both reduced and maximal coordinates, while maintaining near linear performance in the number of bodies.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by Dr. Shinjiro Sueda, thesis committee chair, and committee members Associate Professor Dylan Shell of the Department of Computer Science and Professor Ergun Akleman of the Department of Vizualization.

RedMax was created by Professor Shinjiro Sueda and his students, Ying Wang, Nicholas Weidner, Yura Hwang, and myself.

Funding Sources

Graduate study was supported by a fellowship from the College of Engineering at Texas A&M University, and funding from Dr. Shinjiro Sueda.

NOMENCLATURE

REDMAX	Refers to the new Reduced Coordinate System
PCG	Preconditioned Conjugate Gradient Solve
DOF	Degree of Freedom
Rigid Body	Used synonymously with body, block, and link

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1 INTRODUCTION AND LITERATURE REVIEW

1.1 Introduction

Articulated rigid body dynamics has many applications in various disciplines, including biomechanics, robotics, aerospace, and computer graphics. It has been extensively studied starting in the 1960s (*e.g.,* [Roberson 1966]), but it was not until the 1980s that an $O(n)$ algorithm, where n is the number of joints or bodies, became widely known [Featherstone 1983]. This algorithm and its variants are based on a recursive formulation, where various quantities are computed recursively based on the tree structure of the mechanism. Around the same time, an alternative $O(n^3)$ method based on matrix factorization was also developed by [Walker and Orin 1982], which, according to De Jalon and Bayo [2012], can outperform the $O(n)$ recursive method when n is small (< 10). Although some important mechanisms, such as serial manipulators, have only a few joints, for many applications in computer graphics, n can be quite large—even a single hand has $n \geq 15$. Therefore, there are still many cases where $O(n)$ methods are still preferred over $O(n^3)$ methods.

The story changes when implicit elasticity is added between *arbitrary* bodies rather than only between immediate neighbors, which results in off-diagonal elements being added to the stiffness matrix. Examples of such scenarios include: architectural design with cables [Whiting et al. 2012; Deuss et al. 2014], deployable folding mechanisms [Demaine and O'Rourke 2008; Zhou et al. 2014], and simply attaching damped springs between pairs of bodies. Using the linearly implicit integrator commonly used in graphics [Baraff and Witkin 1998], the $O(n)$ recursive method no longer works because the stiffness matrix breaks the tree topology of the system matrix. To date, the only alternative has been to use the $O(n^3)$ factorization method. We address this issue by introducing a new approach that allows us to

solve the linearly implicit equations of motion in linear to subquadratic time. If the topology-breaking springs are not present, this method gracefully reverts back to the standard $O(n)$ recursive approach.

In the discussion so far, we have tacitly assumed that the dynamics are represented using "reduced" coordinates, where a minimal set of degrees of freedom (DOFs), such as joint angles for revolute joints and relative translations for prismatic joints, are used to represent the state of the system. An alternate approach that uses "maximal" coordinates has also been studied. For example, an $O(n)$ method for maximal coordinates was discovered by Baraff [1996]. However, constraints need to be applied to model joints, and these constraints must be stabilized to avoid drift [Baumgarte 1972; Cline and Pai 2003]. On the other hand, reduced coordinates do not require any stabilization, since reduced coordinates only allow configurations that satisfy the joint constraints. Loops are handled with constraints in either approach, but in practice, stabilizing a few loop constraints is much easier than stabilizing the whole structure. Furthermore, reduced coordinates are in general faster, as no additional joint constraints are required and the number of DOFs in a system is much smaller (typically $\frac{1}{6}$ the size) than its maximal counterpart. Also, Baraff [1996] notes that there is anecdotal evidence that larger time steps are possible using reduced coordinates.

One of the advantages of maximal coordinates is that it is more intuitive—it is easier to add various forces and to combine with other constraints (e.g. damped springs). To address this point, we show that our formulation of dynamics, which we call RedMax, is very flexible even though it uses reduced coordinates. Any combination of reduced/maximal forces can be handled.

To summarize, the contributions presented here are:

- A near linear time approach for articulated dynamics, even in the presence of a non-diagonal maximal stiffness matrix, based on our novel projected block Jacobi preconditioner.
- A formulation that exposes both maximal and reduced degrees of freedom, allowing the presence of forces and constraints in reduced or maximal coordinates.

1.2 Related Work

Articulated rigid body dynamics has been an active research area for many decades, especially in the field of robotics, where high-performance algorithms were required for low-power systems. For example, a great deal of effort has been spent on both $O(n)$ and $O(n^3)$ methods to refine the performance for tree configuration or for closed loop systems [Bae and Haug 1987a,b]. Although asymptotically worse, $O(n^3)$ methods have attracted significant attention because they are more intuitive and are more easily parallelizable [Walker and Orin 1982; Avello et al. 1993; Negrut et al. 1997]. However, these methods do not work in the presence of the maximal stiffness matrix from linearly implicit integration, one of the most common integration methods in graphics [Baraff and Witkin 1998].

There have also been a number of related works on the simulation of various phenomena using articulated rigid bodies, such as: trees [Quigley et al. 2018], hair [Hadap 2006], and characters [Hernandez et al. 2011]. Linear time methods for *flexible* multibody systems have also been studied for decades, as described in the detailed survey by Wasfy and Noor [2003]. Of particular importance to graphics, Bertails [2009] showed that the recursive linear time approach can be used to simulate the dynamics of elastic rods. These efficient methods can only be used in the special case when all of the implicit forces are between topologically neighboring bodies (*e.g.,* joint springs), since then the topology of the reduced stiffness matrix will be the same as that of the reduced mass matrix. However, in the general case, the implicit

forces are between *arbitrary* bodies, and so the recursive linear time approaches cannot be used.

It is important to note that REDMAX can easily be extended to enable the use of reduced/maximal coordinates with fully implicit two way coupling of articulated and deformable bodies, which has been of particular interest in computer graphics: [Shinar et al. 2008], [Kim and Pollard 2011], [Jain and Liu 2011], [Liu et al. 2013]. However, the work presented here is limited in scope to the optimization of REDMAX with respect to systems of rigid bodies with various constraints and forces that induce a complex stiffness matrix, breaking the topology requirements of previous O(n) methods.

1.3 Overview

We first present the mapping between maximal coordinates and the REDMAX reduced coordinate space, and consider how this affects different forces. We then present our iterative solver within REDMAX, which is abbreviated PCG, as it utilizes the preconditioned conjugate gradient method for fast convergence. We then motivate our method as an alternative to direct articulated body solvers with arbitrary forces by presenting three carefully designed scenes to compare its performance to the off-the-shelf linear system Pardiso solver, which automatically combines iterative and direct solving method and has been optimized using parallel computing.

2 BACKGROUND AND THEORY*

2.1 Maximal Coordinates

Following the convention used by Cline and Pai [2003], maximal coordinates can be represented by the usual 4×4 transformation matrix consisting of rotational and translational components:

$$ {}^{0}_{i}\mathsf{E} = \begin{pmatrix} {}^{0}_{i}\mathsf{R} & {}^{0}p \\ 0 & 1 \end{pmatrix}. \tag{2.1} $$

The leading subscripts and superscripts indicate that the coordinates of rigid body (or frame) 'i' are defined with respect to the world frame, '0'. Given a local position ${}^{i}x$ on a rigid body, its world position is

$$ {}^{0}x = {}^{0}_{i}\mathsf{E}\,{}^{i}x, \tag{2.2} $$

omitting the homogeneous coordinates for brevity.

The spatial velocity, ${}^{i}\phi_i$, also called a "twist," is composed of the angular component, ${}^{i}\omega_i$, and the linear component, ${}^{i}v_i$, both expressed in body coordinates:

$$ {}^{i}\phi_i = \begin{pmatrix} {}^{i}\omega_i \\ {}^{i}v_i \end{pmatrix}. \tag{2.3} $$

This 6×1 vector can also be expressed as a 4×4 matrix similar to the transformation matrix in Eq. 2.1, with the rotational part in the 3×3 upper-left block and the translational part in the 3×1 upper-right block:

$$ \left[{}^{i}\phi_i\right] = \begin{pmatrix} \left[{}^{i}\omega_i\right] & {}^{i}v_i \\ 0 & 0 \end{pmatrix}, \tag{2.4} $$

where the 3×3 matrix, $[a]$, is the cross-product matrix.

*Reprinted with permission from "REDMAX: Efficient Flexible Approach for Articulated Dynamics" by Ying Wang, Nicholas J. Weidner, Margaret A. Baxter, Yura Hwang, Danny M. Kaufman, and Shinjiro Sueda (2019). *ACM Transactions on Graphics*, 38(4):104:1-104:10.

The spatial velocity transforms from one frame to another according to the adjoint of the coordinate transform, which is defined from the rigid transform ${}^0_i\mathsf{E}$:

$$ {}^0_i\mathsf{Ad} = \begin{pmatrix} {}^0_i\mathsf{R} & 0 \\ [{}^0_i p]\,{}^0_i\mathsf{R} & {}^0_i\mathsf{R} \end{pmatrix}. \tag{2.5} $$

The spatial velocity of the i^{th} rigid body in world coordinates is then

$$ {}^0\phi_i = {}^0_i\mathsf{Ad}\ {}^i\phi_i. \tag{2.6} $$

The time derivative of the adjoint, dropping the superscripts and subscripts for brevity, can be expressed as:

$$ \dot{\mathsf{Ad}} = \begin{pmatrix} \dot{\mathsf{R}} & 0 \\ [\dot{p}]\mathsf{R} + [p]\dot{\mathsf{R}} & \dot{\mathsf{R}} \end{pmatrix}. \tag{2.7} $$

This can be factored into a product of two matrices, $\mathsf{Ad}(\mathsf{E})$ and $\mathsf{ad}(\phi)$:

$$ \dot{\mathsf{Ad}}(\mathsf{E}, \phi) = \underbrace{\begin{pmatrix} \mathsf{R} & 0 \\ [p]\mathsf{R} & \mathsf{R} \end{pmatrix}}_{\mathsf{Ad}(\mathsf{E})} \underbrace{\begin{pmatrix} [\omega] & 0 \\ [v] & [\omega] \end{pmatrix}}_{\mathsf{ad}(\phi)}, \tag{2.8} $$

with the parameter list displayed to more be explicit. The second factor, $\mathsf{ad} = \mathsf{Ad}^{-1}\,\dot{\mathsf{Ad}}$, is the spatial cross product matrix, which is the adjoint action of the Lie algebra on itself [Selig 2004; Kim 2012].

Finally, the Newton-Euler equations of motion of a single rigid body can be written in a compact form as:

$$ \begin{aligned} \mathsf{M}_i\dot{\phi}_i &= [\text{Coriolis forces}] + [\text{body forces (e.g., gravity)}] \\ &= \mathsf{ad}(\phi_i)^\top \mathsf{M}_i\phi_i + f_{\text{body}}(\mathsf{E}_i), \end{aligned} \tag{2.9} $$

where, M_i is the spatial inertia of the rigid body, $\dot{\phi}_i$ is the *acceleration* of the rigid body, and $\mathrm{ad}(\phi_i)$ is the spatial cross product matrix from Eq. 2.8. The mass matrix is diagonal as all of the velocities are expressed in body coordinates—*i.e.,* we use a body-fixed frame aligned with the principal axis of the body and whose origin is coincident with the center of mass of the body. Stacking all of the maximal acceleration DOFs together, $\ddot{\mathsf{q}}_m = \begin{pmatrix}\dot{\phi}_1^\top & \cdots & \dot{\phi}_n^\top\end{pmatrix}^\top$, provides the following linear system for *maximal* DOFs:

$$\mathsf{M}_m \ddot{\mathsf{q}}_m = \mathsf{f}_m. \tag{2.10}$$

2.2 Reduced Coordinates

Recursive O(n) methods for the forward and inverse dynamics of articulated mechanisms with support for both reduced and maximal coordinates have existed for decades [Popov et al. 1978; Featherstone 1983; Baraff 1996; Negrut et al. 1997; Serban et al. 1997]. Unfortunately, when maximal springs apply implicit forces on the rigid bodies (*e.g.,* Figs.1b & 1c), these recursive $O(n)$ methods can no longer be used, because the stiffness matrix resulting from these springs breaks the tree topology of the system matrix. This section defines the RedMax approach (Eq. 2.13), and introduces a new preconditioner that gives linear to subquadratic time performance, depending on the scene, even in the presence of the maximal stiffness matrix (Eq. 3.1).

2.2.1 Define the Reduced Coordinate Space

Reduced coordinate methods have been used before; this re-imagining defines the state of the entire system through the joints. For the types of joints presented in the various scenes in this paper, as well as other simple and highly complex joints, there is a linear relationship between the maximal and reduced velocities.

Denoted by $\dot{q}_m$ the vector of all maximal velocities and by $\dot{q}_r$ the vector of all reduced velocities:

$$\dot{q}_m = J_{mr}\dot{q}_r, \quad \ddot{q}_m = \dot{J}_{mr}\dot{q}_r + J_{mr}\ddot{q}_r, \tag{2.11}$$

where J_{mr} is the Jacobian matrix that transforms velocities from reduced to maximal, which will be derived later in this section. Then, combining Eq. 2.10 and Eq. 2.11 yields

$$\left(J_{mr}^\top M_m J_{mr}\right) \ddot{q}_r = J_{mr}^\top \left(f_m - M_m \dot{J}_{mr}\, \dot{q}_r\right). \tag{2.12}$$

Where M_m is the diagonal maximal mass matrix, q_r is the reduced configuration (*e.g.,* joint angles), f_m is the maximal force, and J_{mr} is the velocity transforming Jacobian mentioned earlier. The Jacobian and its time derivative, $J_{mr}, \dot{J}_{mr}$, are of size $\#m \times \#r$, where $\#m$ is the number of maximal DOFs, and $\#r$ is the number of reduced DOFs.

The reduced inertia matrix, $M_r = J_{mr}^\top M_m J_{mr}$ is much smaller than its maximal counterpart. Furthermore, representing joints here does not require constraints, since the Jacobian automatically projects forces down to the reduced space. The last term, $-J_{mr}^\top M_m \dot{J}_{mr}\, \dot{q}_r$, is the extra *quadratic velocity vector* due to the change of coordinates, in analogy to the Coriolis force in Eq. 2.9 [Shabana 2013]. This formulation (Eq. 2.12) is an instance of the well-known "velocity transformations" for articulated dynamics [De Jalon and Bayo 2012]. This equation of motion can be used in conjunction with different choices of time integrators.

We now describe our RedMax formulation that exposes all the reduced and maximal quantities, and, following the common practice in graphics, discretizes Eq. 2.12 at the velocity level. Then, by combining the linearly implicit terms for both reduced and maximal coordinates with this discretization [Baraff and Witkin 1998], we arrive at our RedMax

formulation:

$$\begin{aligned}\left(\mathsf{J}_{mr}^{\top}\left(\mathsf{M}_m + h\mathsf{D}_m - h^2\mathsf{K}_m\right)\mathsf{J}_{mr} + h\mathsf{D}_r - h^2\mathsf{K}_r\right)\dot{\mathsf{q}}_r^{(k+1)} = \\ \left(\mathsf{J}_{mr}^{\top}\mathsf{M}_m\mathsf{J}_{mr}\right)\dot{\mathsf{q}}_r^{(k)} + h\left(\mathsf{f}_r + \mathsf{J}_{mr}^{\top}\left(\mathsf{f}_m - \mathsf{M}_m\dot{\mathsf{J}}_{mr}\,\dot{\mathsf{q}}_r^{(k)}\right)\right),\end{aligned} \tag{2.13}$$

where D_m and K_m are the maximal damping and stiffness matrices, D_r and K_r are the reduced damping and stiffness matrices, f_r is the reduced force vector, f_m is the maximal force vector, including the Coriolis force, and $\dot{\mathsf{q}}_r^{(k)}$ and $\dot{\mathsf{q}}_r^{(k+1)}$ are the reduced velocities at time steps k and $k+1$. The resulting LHS matrix, in reduced coordinates, is

$$\tilde{\mathsf{M}}_r = \left(\mathsf{J}_{mr}^{\top}\left(\mathsf{M}_m + h\mathsf{D}_m - h^2\mathsf{K}_m\right)\mathsf{J}_{mr} + h\mathsf{D}_r - h^2\mathsf{K}_r\right) \tag{2.14}$$

and the RHS vector is

$$\tilde{\mathsf{f}}_r = \left(\mathsf{J}_{mr}^{\top}\mathsf{M}_m\mathsf{J}_{mr}\right)\dot{\mathsf{q}}_r^{(k)} + h\left(\mathsf{f}_r + \mathsf{J}_{mr}^{\top}\left(\mathsf{f}_m - \mathsf{M}_m\dot{\mathsf{J}}_{mr}\,\dot{\mathsf{q}}_r^{(k)}\right)\right). \tag{2.15}$$

This equation, $\tilde{\mathsf{M}}_r\dot{\mathsf{q}}_r^{(k+1)} = \tilde{\mathsf{f}}_r$, gives us the flexibility to choose the types of forces we want to use, be they maximal or reduced.

2.2.2 The Mapping Jacobian

Assuming there are no loops (as loops are handled with constraints), the topology of the system can be described by a tree system. Furthermore, there is a one-to-one relationship between a body and a joint—every body has a joint between itself and its parent body. The body frame is aligned to the body's inertial frame (as described below Eq. 2.9), and the joint frame is aligned according to the joint type (*e.g.,* Z axis along the axis of rotation). Here i will be used to denote a body, j to denote its corresponding joint, and p to denote the parent *body* of i (or to the parent *joint* of j depending on the context).

The twists of bodies p and i at joint j are

$$ {}^{j}\phi_p = {}^{j}_{p}\mathrm{Ad}\,{}^{p}\phi_p, \quad {}^{j}\phi_i = {}^{j}_{i}\mathrm{Ad}\,{}^{i}\phi_i, \tag{2.16} $$

and their *relative* twist is

$$ \begin{aligned} {}^{j}\phi_j &= {}^{j}\phi_i - {}^{j}\phi_p \\ &= {}^{j}_{i}\mathrm{Ad}\,{}^{i}\phi_i - {}^{j}_{p}\mathrm{Ad}\,{}^{p}\phi_p \\ &= {}^{j}_{i}\mathrm{Ad}\,{}^{i}\phi_i - {}^{j}_{i}\mathrm{Ad}\,{}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad}\,{}^{p}\phi_p, \end{aligned} \tag{2.17} $$

where 0 indicates the world frame. Since i owns the joint, ${}^{j}_{i}\mathrm{Ad}$ is constant. (It is constructed from ${}^{j}_{i}\mathsf{E}$, which represents where the i's body frame is with respect to the joint frame, which is set at initialization.) Note that in maximal coordinates, positions are stored with respect to the world and velocities with respect to the body itself. In other words, for each body, store ${}^{0}_{i}\mathsf{E}$ and ${}^{i}\phi_i$. So in the above expression, the adjoint matrices of the form ${}^{0}_{i}\mathrm{Ad}$ and ${}^{i}_{0}\mathrm{Ad}$ can be computed easily from ${}^{0}_{i}\mathsf{E}$. Then rearrange Eq. 2.17 to solve for body i's spatial velocity:

$$ \begin{aligned} {}^{j}_{i}\mathrm{Ad}\,{}^{i}\phi_i &= {}^{j}_{i}\mathrm{Ad}\,{}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad}\,{}^{p}\phi_p + {}^{j}\phi_j \\ {}^{i}\phi_i &= {}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad}\,{}^{p}\phi_p + {}^{i}_{j}\mathrm{Ad}\,{}^{j}\phi_j. \end{aligned} \tag{2.18} $$

What this expression implies is that if parent body's velocity, ${}^{p}\phi_p$, is known, and the joint's velocity, ${}^{j}\phi_j$,is known, then we can compute the child body's velocity, ${}^{i}\phi_i$. In reduced coordinates, ${}^{j}\phi_j$ is parameterized not with the full 6 degrees of freedom but with some subset $\dot{\mathrm{q}}_j \subseteq \mathbb{R}^6$. For example, a revolute joint about the Z axis can be expressed as

$$ {}^{j}\phi_j = S_j\dot{\mathrm{q}}_j, \quad S_j = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}^{\top}. \tag{2.19} $$

Here S is used to follow the notation of Park et al. [1995] and Kim [2012]. S takes on this simple constant form for revolute joints, but in general is a nonlinear function of the joint's DOFs, $\dot{\mathrm{q}}_j$. Combining Eq. 2.18 and Eq. 2.19, provides the recursive expression for the velocity of body i:

$$ {}^{i}\phi_i = {}^{i}_{p}\mathrm{Ad}\,{}^{p}\phi_p + {}^{i}_{j}\mathrm{Ad}\,S_j\dot{\mathrm{q}}_j, \tag{2.20} $$

where ${}^{i}_{p}\mathrm{Ad} = {}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad}$.

The recursive relationship between the velocity of a body and its parents' bodies can be recursively applied to form the system Jacobian. For example, for a serial chain with three links, the Jacobian is the following lower triangular matrix, where each body quantity is labeled with i and each joint quantity with j:

$$ \underbrace{\begin{pmatrix} {}^{i1}\phi_{i1} \\ {}^{i2}\phi_{i2} \\ {}^{i3}\phi_{i3} \end{pmatrix}}_{\dot{\mathrm{q}}_m} = \underbrace{\begin{pmatrix} {}^{i1}_{j1}\mathrm{Ad}\,S_{j1} & 0 & 0 \\ {}^{i2}_{i1}\mathrm{Ad}\,{}^{i1}_{j1}\mathrm{Ad}\,S_{j1} & {}^{i2}_{j2}\mathrm{Ad}\,S_{j2} & 0 \\ {}^{i3}_{i2}\mathrm{Ad}\,{}^{i2}_{i1}\mathrm{Ad}\,{}^{i1}_{j1}\mathrm{Ad}\,S_{j1} & {}^{i3}_{i2}\mathrm{Ad}\,{}^{i2}_{j2}\mathrm{Ad}\,S_{j2} & {}^{i3}_{j3}\mathrm{Ad}\,S_{j3} \end{pmatrix}}_{\mathrm{J}_{mr}} \underbrace{\begin{pmatrix} \dot{\mathrm{q}}_{j1} \\ \dot{\mathrm{q}}_{j2} \\ \dot{\mathrm{q}}_{j3} \end{pmatrix}}_{\dot{\mathrm{q}}_r}. \tag{2.21} $$

The pseudocode to fill J_{mr} is given in Alg. 1. This function must be called on the joints in a tree traversal order starting from the root, as it takes advantage of the recursive structure of the tree hierarchy. As the ancestor's hierarchy is traversed, we reuse the products already computed by the ancestor instead of recomputing those values. Since this matrix has $O(n^2)$ elements, it takes $O(n^2)$ time to fill, even with its recursive structure. However, since we only need the product of this matrix with a vector, the algorithm only takes $O(n)$ time, as shown in Alg. 2 and Alg. 3, a strategy implicitly exploited by the recursive dynamics algorithm [Featherstone 1983; Kim 2012].

To compute $\dot{\mathrm{J}}$, take the derivative of each term in Eq. 2.21. For the diagonal terms, the derivative is

$$\dot{\mathrm{J}}(i,j) = {}^{i}_{j}\mathrm{Ad}\,\dot{S}_j, \tag{2.22}$$

which is 0 for revolute joints, since S is constant. For off-diagonal terms, traverse the hierarchy through the joint's ancestors back to the root, and the recursive expression for the derivative is

$$\dot{\mathrm{J}}(i,a) = {}^{i}_{p}\dot{\mathrm{Ad}}\,\mathrm{J}(p,a) + {}^{i}_{p}\mathrm{Ad}\,\dot{\mathrm{J}}(p,a), \tag{2.23}$$

where $a = a(j)$ is an ancestor *joint* of j, and $p = p(i)$ is the parent *body* of i. To compute ${}^{i}_{p}\dot{\mathrm{Ad}}$, use Eq. 2.8 and the identity for taking the derivative of the matrix inverse: $\dot{A}^{-1} = -A^{-1}\dot{A}A^{-1}$:

$${}^{i}_{p}\dot{\mathrm{Ad}} = -{}^{i}_{0}\mathrm{Ad}\,{}^{0}_{i}\dot{\mathrm{Ad}}\,{}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad} + {}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\dot{\mathrm{Ad}}. \tag{2.24}$$

2.2.3 Mapping Various Joints

This section describes all joint types used in the various scenes presented. For all joint types, the joint transform is a 4×4 matrix that defines where the joint, j, is with respect to its

Algorithm 1 Fills the Jacobian matrix and its time derivative

while forward traversal **do**
  $\mathrm{J}(i,j) = {}^{i}_{j}\mathrm{Ad}\,S_j$
  $\dot{\mathrm{J}}(i,j) = {}^{i}_{j}\mathrm{Ad}\,\dot{S}_j$
  ancestor a = parent *joint* of j
  while $a \neq$ null **do**
    $\mathrm{J}(i,a) = {}^{i}_{p}\mathrm{Ad}\,\mathrm{J}(p,a)$ ▷ $p = p(i)$ is the parent *body* of i
    $\dot{\mathrm{J}}(i,a) = {}^{i}_{p}\dot{\mathrm{Ad}}\,\mathrm{J}(p,a) + {}^{i}_{p}\mathrm{Ad}\,\dot{\mathrm{J}}(p,a)$
    $a = a$'s parent joint
  end while
end while

Algorithm 2 Computes products $y = \mathrm{J}x$ and $z = \dot{\mathrm{J}}x$

while forward traversal **do**
  $y(i) = {}^{i}_{j}\mathrm{Ad}\, S_j\, x(j)$
  $z(i) = {}^{i}_{j}\mathrm{Ad}\, \dot{S}_j\, x(j)$
  if $p \neq$ null **then** ▷ $p = p(j)$ is the parent joint of j
    $y(i)$ += ${}^{i}_{p}\mathrm{Ad}\, y(p)$
    $z(i)$ += ${}^{i}_{p}\mathrm{Ad}\, z(p) + {}^{i}_{p}\dot{\mathrm{Ad}}\, y(p)$
  end if
end while

Algorithm 3 Computes product $x = \mathrm{J}^{\top} y$

while backward traversal **do**
  $y_i = y(i)$
  for all children c **do** ▷ $c = c(j)$ is a child joint of j
    y_i += α_c ▷ α is a temp variable stored by each joint
  end for
  $\alpha_i = {}^{i}_{p}\mathrm{Ad}^{\top} y_i$ ▷ to be used by j's parent later
  $x(j) = S_j^{\top}\, {}^{i}_{j}\mathrm{Ad}^{\top}\, y_i$
end while

parent joint, $p(j)$:

$$ {}^{p}_{j}\mathsf{E} = {}^{p}_{j}\mathsf{E}_0\ \mathrm{Q}_j(\mathrm{q}_j), \tag{2.25} $$

where ${}^{p}_{j}\mathsf{E}_0$ is the initial transform (often a translation) that specifies where the joint is with respect to its parent joint, and $\mathrm{Q}_j(\mathrm{q}_j)$ is the transform that actually applies the degrees of freedom of that joint. Additionally, for each joint type, S and $\dot{S}$ are required for the computation of J and $\dot{\mathrm{J}}$.

Fixed Joint

A fixed joint is used for rigidly attaching two bodies together. For a fixed joint, $\mathrm{q}_j = \varnothing$, and $\mathrm{Q}_j(\mathrm{q}_j)$ is simply the 4×4 identity matrix. The joint Jacobian, S, is an empty 6×0 matrix.

Prismatic Joint

A prismatic joint allows one degree of translational freedom. Let a represent the axis along which the joint is able to translate. Then

$$\mathrm{Q}_j(\mathrm{q}_j) = \begin{pmatrix} I & a\mathrm{q}_j \\ 0 & 1 \end{pmatrix}, \tag{2.26}$$

which is a 4×4 translation matrix. The corresponding joint Jacobian is

$$S = \begin{pmatrix} 0 \\ a \end{pmatrix} \in \mathbb{R}^{6\times 1}. \tag{2.27}$$

Revolute Joint

A revolute joint allows rotation about an axis, a. The rotation matrix is constructed from the (axis, angle) pair: (a, q_j).

$$\mathrm{Q}_j(\mathrm{q}_j) = \begin{pmatrix} \mathrm{R}(a, \mathrm{q}_j) & 0 \\ 0 & 1 \end{pmatrix}, \tag{2.28}$$

and the corresponding joint Jacobian is

$$S = \begin{pmatrix} a \\ 0 \end{pmatrix} \in \mathbb{R}^{6\times 3}. \tag{2.29}$$

Universal Joint

A universal joint allows bending in X and Y but no twisting along Z. We start with the rotation matrix corresponding to the XYZ Euler angles:

$$\mathrm{R} = \begin{pmatrix} c_2c_3 & -c_2s_3 & s_2 \\ c_1s_3 + s_1s_2c_3 & c_1c_3 - s_1s_2s_3 & -s_1c_2 \\ s_1s_3 - c_1s_2c_3 & s_1c_3 + c_1s_2s_3 & c_1c_2 \end{pmatrix}, \tag{2.30}$$

where $c_1 = \cos(q_1), c_2 = \cos(q_2)$, etc. Then fix the third angle at 0, so that $c_3 = 1$ and $s_3 = 0$. This results in

$$\mathrm{R} = \begin{pmatrix} c_2 & 0 & s_2 \\ s_1s_2 & c_1 & -s_1c_2 \\ -c_1s_2 & s_1 & c_1c_2 \end{pmatrix}. \tag{2.31}$$

Q is then

$$\mathrm{Q} = \begin{pmatrix} \mathrm{R} & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.32}$$

The joint Jacobian, S, is going to be a 6×2 matrix. To get the 1st column of S, take the derivative of R with respect to q_1 and premultiply by $\mathrm{R}^\top$. After some cancellations, the result is a skew symmetric matrix, from which the angular elements are extracted into the first

column of S. Repeat this for the second column, and the resulting matrix is

$$S = \begin{pmatrix} c_2 & 0 \\ 0 & 1 \\ s_2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{2.33}$$

The time derivative of the joint Jacobian is

$$\dot{S} = \begin{pmatrix} -s_2\dot{q}_2 & 0 \\ 0 & 0 \\ c_2\dot{q}_2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{2.34}$$

2.2.4 Forces

Our REDMAX formulation Eq. 2.13 exposes terms that easily allow us to account for constraints and forces in both maximal and reduced coordinates. These terms may come from a variety of sources, including geometric stiffness [Tournier et al. 2015]. For example, we can easily combine body damping (D_m), maximal springs acting on the bodies (K_m and f_m), joint damping (D_r), and joint stiffness (K_r and f_r). Recall that we can use a O(n) solve when the maximal forces present only connect adjacent bodies. In this case both D_m and K_m would have block entries only along the diagonal. When forces reach across bodies that do not share a connecting joint, these matrices will have entries beyond the diagonal, breaking the topology

limitation of previous O(n) methods. Following the approach of Baraff and Witkin [1998], we can then add implicit damping and elastic forces into the various exposed matrices.

Damping Force

With simple viscous damping, $\mathsf{D} = dI$ is a diagonal matrix, where d is the damping coefficient. There is no contribution to the right-hand-side force vector, f.

Directional Point Force

Let us say that we want to pull on a point ${}^{i}x$ on a rigid body in a particular direction ${}^{0}a$, where ${}^{i}x$ is in local coordinates and ${}^{0}a$ is in world coordinates. Then the linear wrench to be applied to the rigid body can be computed as follows:

$$\mathsf{f} = k\Gamma^{\top}\mathsf{R}^{\top}\,{}^{0}a, \tag{2.35}$$

where k is the stiffness constant, $\Gamma = ([{}^{i}x]^{\top}\ I)$ transforms twists to local point velocities, and R is the rotation matrix of the rigid body. The corresponding potential energy is

$$V = -k\,{}^{0}x^{\top}\,{}^{0}a, \tag{2.36}$$

where ${}^{0}x$ is the position of the force application point in world coordinates. The force in Eq. 2.35 is the negative gradient of this potential energy with respect to the 6 rigid degrees of freedom. We obtain the stiffness matrix if we differentiate again:

$$\mathsf{K} = k\begin{pmatrix} [{}^{i}x][\mathsf{R}^{\top}\,{}^{0}a] & 0 \\ [\mathsf{R}^{\top}\,{}^{0}a] & 0 \end{pmatrix}, \tag{2.37}$$

where we use the following identity for the derivatives with respect to the 6 rigid DOFs:

$$\frac{\partial \mathsf{R}^{\top}a}{\partial \omega} = [\mathsf{R}^{\top}a], \quad \frac{\partial \mathsf{R}a}{\partial \omega} = -\mathsf{R}[a], \quad \frac{\partial p}{\partial v} = \mathsf{R}. \tag{2.38}$$

Point-to-Point Force

For a linear force between two points on two different bodies, the wrenches acting on these two bodies are

$$\mathsf{f} = k \begin{pmatrix} \Gamma_1^\top \mathsf{R}_1^\top \, \Delta x \\ -\Gamma_2^\top \mathsf{R}_2^\top \, \Delta x \end{pmatrix}, \tag{2.39}$$

where $\Delta x = {}^0x_2 - {}^0x_1$, and ${}^0x_1$ and ${}^0x_2$ are the world coordinate positions of the two points, which are obtained by transforming the corresponding local coordinate positions. This force is the negative gradient of the potential energy:

$$V = \frac{1}{2} k \Delta x^\top \Delta x. \tag{2.40}$$

As before, we obtain the stiffness matrix by differentiating the force with respect to the DOFs:

$$\mathsf{K} = k \begin{pmatrix} [{}^1x_1][\mathsf{R}_1^\top (p_1 - {}^0x_2)] & [{}^1x_1] & [{}^1x_1]\mathsf{R}_1^\top \mathsf{R}_2[{}^2x_2] & -[{}^1x_1]\mathsf{R}_1^\top \mathsf{R}_2 \\ [\mathsf{R}_1^\top (p_1 - {}^0x_2)] & I & \mathsf{R}_1^\top \mathsf{R}_2[{}^2x_2] & -\mathsf{R}_1^\top \mathsf{R}_2 \\ [{}^2x_2]\mathsf{R}_2^\top \mathsf{R}_1[{}^1x_1] & -[{}^2x_2]\mathsf{R}_2^\top \mathsf{R}_1 & [{}^2x_2][\mathsf{R}_2^\top (p_2 - {}^0x_1)] & [{}^2x_2] \\ \mathsf{R}_2^\top \mathsf{R}_1[{}^1x_1] & -\mathsf{R}_2^\top \mathsf{R}_1 & [\mathsf{R}_2^\top (p_2 - {}^0x_1)] & I \end{pmatrix}. \tag{2.41}$$

2.2.5 Constraints

Of the constraints that may be present in a system, loop closing constraints are the most common. They do not fit in reduced coordinates, which define the relationship of bodies in terms of their parent body through a connecting joint, as directly adding a loop constraint in this manner would over-constrain the linear system. For example, in the BRIDGE scene shown in Fig. 1b, we apply bilateral loop-closing constraints, $\mathsf{G}\dot{\mathsf{q}}_r = 0$, to ground both ends of the bridge deck. We solve the following dual problem:

$$\mathsf{G}\tilde{\mathsf{M}}_r^{-1}\mathsf{G}^\top \lambda = \mathsf{G}\tilde{\mathsf{M}}_r^{-1}\tilde{\mathsf{f}}_r, \quad \tilde{\mathsf{M}}_r\dot{\mathsf{q}}_r = \tilde{\mathsf{f}}_r - \mathsf{G}^\top \lambda. \tag{2.42}$$

3 OUR NEAR LINEAR TIME METHOD

3.1 Solving RedMax

When there are no maximal springs imposing off-diagonal terms in D_m and K_m , we can apply the standard $O(n)$ approach to solve the system, more concisely written $\tilde{\mathsf{M}}_r\dot{\mathsf{q}}_r = \tilde{\mathsf{f}}_r$. By allowing for arbitrary maximal forces we can no longer directly solve $\tilde{\mathsf{M}}_r^{-1}\tilde{\mathsf{f}}_r$. Instead we introduce our preconditioner, P, an approximation of $\tilde{\mathsf{M}}_r$, and of a form that allows to compute its product with a vector in linear time. We can then apply a preconditioned conjugate method (PCG) to iteratively solve $\mathsf{P}^{-1}\tilde{\mathsf{M}}_r\dot{\mathsf{q}}_r = \mathsf{P}^{-1}\tilde{\mathsf{f}}_r$, which converges to the solution of Eq. 2.13.

3.2 Projected Block Jacobi Preconditioner

This section will introduce a preconditioner that gives linear to subquadratic performance in the presence of the maximal stiffness matrix. This preconditioner is effective when the rigid DOFs make up a large portion of the system DOFs, and when these rigid DOFs are tied together by *maximal* forces, such as damped springs between various bodies. These cover some important simulation scenarios, including architectural design with cables [Whiting et al. 2012; Deuss et al. 2014] and biomechanical simulations with line-based forces [Delp et al. 2007; Wang et al. 2012].

The preconditioner, P, can be expressed as follows:

$$\mathsf{P} = \mathsf{J}_{mr}^{\top}(\mathsf{M}_m + \text{blkdiag}(h\mathsf{D}_m - h^2\mathsf{K}_m))\,\mathsf{J}_{mr} + h\mathsf{D}_r - h^2\mathsf{K}_r, \tag{3.1}$$

where 'blkdiag' is a filter that keeps only the 6×6 diagonal blocks of D_m and K_m. We call this the "projected block Jacobi" preconditioner because we take the block diagonals of the maximal terms and project them into the reduced space. When there are no maximal

Algorithm 4 Computes $y = (\mathsf{M}_r + \mathsf{J}_{mr}^\top \text{blkdiag}(h\mathsf{D}_m - h^2\mathsf{K}_m)\mathsf{J}_{mr} + h\mathsf{D}_r - h^2\mathsf{K}_r)^{-1}x$ in linear time for preconditioning a linearly implicit solver. Script j refers to the current joint, i to the associated body, c to the joint's child joint, and p to the joint's parent joint.

// Run this loop once as a preprocessing step
while backward traversal **do** ▷ c = child joint of j
  $\mathsf{A}_j^m = {}_j^i\mathsf{Ad}^\top \text{blkdiag}(h\mathsf{D}_i^m - h^2\,\mathsf{K}_i^m)\,{}_j^i\mathsf{Ad}$ ▷ Maximal term
  $\mathsf{A}_j^r = h\mathsf{D}_j^r - h^2\mathsf{K}_j^r$ ▷ Reduced term
  $\hat{\mathsf{M}}_j = (\mathsf{M}_j + \mathsf{A}_j^m) + \sum_c {}_j^c\mathsf{Ad}^\top\, \Pi_c\, {}_j^c\mathsf{Ad}$
  $\Psi_j = (S_j^\top \hat{\mathsf{M}}_j S_j + \mathsf{A}_j^r)^{-1}$
  $\Pi_j = \hat{\mathsf{M}}_j - \hat{\mathsf{M}}_j S_j \Psi_j S_j^\top \hat{\mathsf{M}}_j$
end while

// Run these two loops for each RHS vector x
while backward traversal **do** ▷ c = child joint of j
  $\hat{\mathcal{B}}_j = \sum_c {}_j^c\mathsf{Ad}^\top\, \beta_c$
  $\beta_j = \hat{\mathcal{B}}_j + \hat{\mathsf{M}}_j(S_j\Psi_j(x_j - S_j^\top \hat{\mathcal{B}}_j))$
end while
while forward traversal **do** ▷ p = parent joint of j
  $y_j = \Psi_j(x_j - S_j^\top \hat{\mathsf{M}}_j\, {}_p^j\mathsf{Ad}\,\dot{\mathsf{V}}_p - S_j^\top \hat{\mathcal{B}}_j)$
  $\dot{\mathsf{V}}_j = {}_p^j\mathsf{Ad}\,\dot{\mathsf{V}}_p + S_j y_j$
end while

springs, this preconditioner still gives the same performance as the $O(n)$ recursive approach—it gracefully reverts back to the standard $O(n)$ approach. This preconditioner is motivated by Featherstone's algorithm [Featherstone 1983], and is shown in Alg. 4.

3.3 PCG

To use P in the preconditioned conjugate gradient (PCG) method to solve Eq. 2.13 in near linear time, we have the following requirements:

(1) Form the RHS vector of Eq. 2.13 in $O(n)$ time.

(2) Multiply a vector by the LHS matrix of Eq. 2.13 in $O(n)$ time.

(3) Apply the preconditioner, P, in $O(n)$ time.

(4) Converge in a sublinear number of iterations.

Steps (1) and (2), require multiplying a vector by J, $\mathsf{J}^\top$, and $\dot{\mathsf{J}}$, as required by the RHS of Eq. 2.13, in $O(n)$ time. Although filling these matrices takes $O(n^2)$ time, computing the product can be done in $O(n)$ time by taking advantage of the recursive nature of the topology. To multiply by J and $\dot{\mathsf{J}}$, we traverse forward starting from the root, whereas to multiply by $\mathsf{J}^\top$, we traverse backward starting from the leaf. The recursive dynamics method takes this approach while computing the reduced velocities and forces. Alg. 2 shows the procedures for computing $y = \mathsf{J}x$ and $z = \dot{\mathsf{J}}x$, and Alg. 3 shows the procedure for computing $x = \mathsf{J}^\top y$. Adding the spring contributions to the RHS force and LHS stiffness matrix can be done trivially in $O(m)$ time using standard techniques, where m is the number of springs.

To enable (3), we must be able to *solve* by P in linear time. We draw inspiration from the fact that the recursive *forward* dynamics algorithm solves the reduced system $\mathsf{M}_r\ddot{\mathsf{q}}_r = \mathsf{f}_r$ in linear time, allowing it be utilized efficiently to construct, or to multiply by, the inverse inertia matrix by setting all forces and velocities to zero [Kim 2012; Drumwright 2012]. In the same way, this preconditioner can be used to solve the block diagonal approximation of the LHS matrix of Eq. 2.13 in linear time. The standard recursive forward dynamics algorithm can achieve this with two important modifications corresponding to the maximal and reduced implicit terms, as shown lines 3-6 in Alg. 4. These two types of implicit terms must be handled differently, since they operate in different spaces. In each joint j, we store the reduced stiffness and damping matrices (scalars for revolute joints), and in the corresponding body i, we store the 6×6 block diagonal components of the maximal stiffness and damping matrices. The maximal terms are first transformed to be in j's coordinate space and then are added to the j's

inertia matrix in line 5. Then, the reduced terms are added prior to taking the inverse, in line 6 of Alg. 4. These terms are then processed recursively together with the inertia.

For (4), we will offer empirical evidence comparing the performance of REDMAX to an optimized sparse direct solver when simulating a TREE (Fig. 1a), BRIDGE (Fig. 1b) and the UMBRELLA (Fig. 1c) scenes. When using the direct solver, the system indices are ordered backward from leaf to minimize fill-ins [Negrut et al. 1997]. The TREE scene provides a worst-case evaluation of our preconditioner with respect to the Pardiso solver. In this scene the mass matrix is very sparse (Fig. 1d), and behaves well for optimized sparse solvers (e.g. Pardiso). In the BRIDGE scene wherein the towers are infinitely stiff, PCG converges in 1 iteration because all of the cables are attached to a stationary body, and so the stiffness matrix becomes block diagonal. We also test the bridge scene with variably stiff body chains in place of the stationary towers; this imposes entries in the mass matrix off the diagonal and provides an interesting evaluation of the performance of REDMAX. Finally, the umbrella scene features a dense mass matrix with off-diagonal terms and exemplifies a case when REDMAX is expected to perform well.

3.4 Loop Closure and PCG

Recall that in the presence of loop-closing constraints we solve the dual problem:

$$\mathrm{G}\tilde{\mathrm{M}}_r^{-1}\mathrm{G}^\top\lambda = \mathrm{G}\tilde{\mathrm{M}}_r^{-1}\tilde{\mathrm{f}}_r, \quad \tilde{\mathrm{M}}_r\dot{\mathrm{q}}_r = \tilde{\mathrm{f}}_r - \mathrm{G}^\top\lambda. \tag{3.2}$$

Let l be the number of rows in the constraint matrix G. Then we need to run PCG l times to form the dense LHS matrix $\mathrm{G}\tilde{\mathrm{M}}_r^{-1}\mathrm{G}^\top$ by backsolving with the columns of $\mathrm{G}^\top$. We also run PCG once to form the RHS vector, and then solve these small, dense LHS and RHS matrices for the Lagrange multipliers in $O(l^3)$ time. They are then fed into a final PCG to compute the new velocities. The first l calls to PCG are independent of each other and compute the

reduced column vectors of $\tilde{M}_r^{-1}G^\top$. Using OpenMP to run these PCG operations in parallel will minimize the overhead associated with including maximal constraints in a scene. The overall run time is then $O(n^\alpha l + l^3)$, where α depends on the scene and should typically be near linear.

3.5 Code Optimization

The Intel Math Kernel Library includes optimizations for Eigen as well as our chosen direct system solver, Pardiso. We use Eigen to perform all mathematical operations for both PCG and the direct solver. The Pardiso solver also utilizes parallelization, automatic combination of direct and iterative solvers, and can rely on a high rate of cache hits.

To ensure REDMAX is competitive with the direct solver, we considered ways of decreasing the overall runtime rather than only focusing on the asymptotic behavior of the method. For simplicity, we initially implemented REDMAX using c++ structures for joints and bodies and stored the configuration as arrays of these structures. To reduce the overhead associated with using many different object potentially stored inconveniently in memory, we instead stored all associated member variables directly in many arrays. Using the Structure of Arrays framework increased the efficiency of REDMAX by about 3 times.

All Eigen containers are initialized during the setup of the scene, as doing so repeatedly would incur a lot of overhead. Additionally, there are several instances where the algorithms presented in the previous section perform redundant computations. To address the redundancies, we took care to save any such products and thereby prevent performing the same matrix operations multiple times.

The last significant improvement to performance improves scenes with maximal constraints (wherein each constraint requires a separate call to PCG) by performing the calls in parallel. We feel utilizing parallel computing is justified because the Pardiso solver does so

as well. In our method, parallelization is only used when solving systems with many loop closing constraints, whereas Pardiso utilizes parallelization for all solves. For this reason, future efforts might well be focused on adding parallel computations throughout the method.

4 RESULTS

We implemented our system in C++ and ran the simulations on a consumer desktop with an Intel Core i7-7700 CPU @ 3.6 Ghz and 16 GB of RAM. We use Eigen for linear algebra computations, both sparse and dense, and Pardiso as the comparison for sparse linear solves. We ran each simulation for both REDMAX and the direct solver (Pardiso) at different resolutions and display the recorded wall-clock timing for each case on a $\log_2 - \log_2$ plot. The slope of fitted lines of the $\log_2 - \log_2$ plot of time vs. DOFs shows the empirical order of each approach.

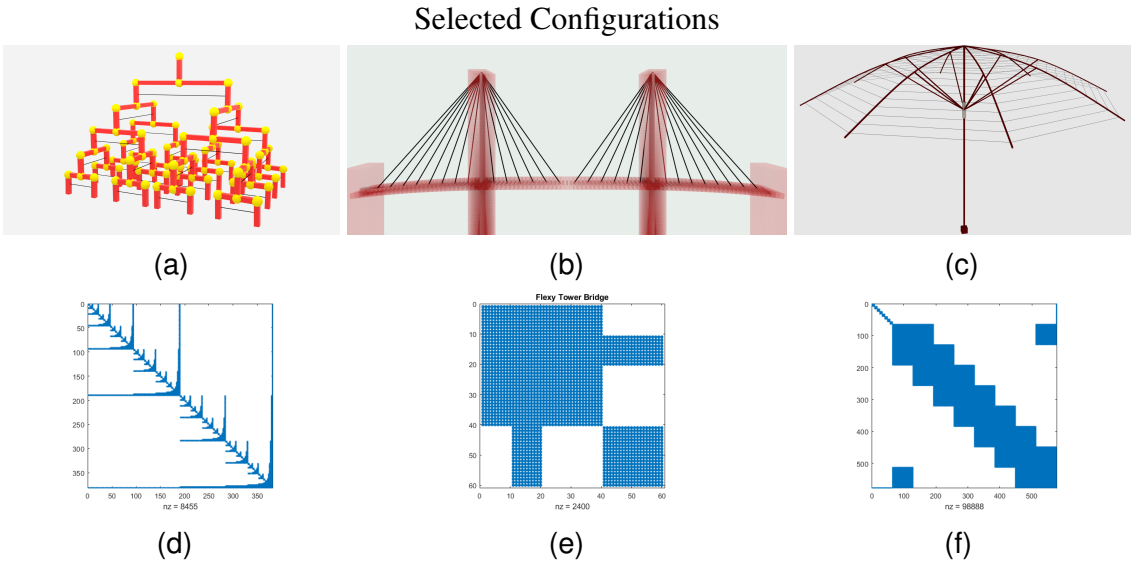

Fig. 1. (a) TREE and, (d) shows sparsity of mass matrix for the tree scene. (b) A cable-stayed BRIDGE and, (e) sparsity. (c) A deployable UMBRELLA and, (f) shows sparsity.

We will discuss the results of simulating 5 seconds of scene time for these three scenes under different circumstances using both our REDMAX solver and the Pardiso solver.

4.1 Bridge

This scene is modeled after a cable-stayed bridge with a fan design, (Figs. 1b & 1e). The towers and the deck are composed of a sequence of bodies connected by revolute joints, with 30 cables attaching evenly spaced deck pieces to the towers. In this scene, when the towers are infinitely stiff, we instead model the towers as a single fixed body. Then PCG converges in 1 iteration, because all of the cables are attached to a stationary body, and so the stiffness matrix becomes block diagonal. Our block diagonal preconditioner then becomes exactly the inverse of the system matrix. When we model the towers as variably stiff body chains, in place of stationary towers, this fills entries in the mass matrix off the diagonal and provides an interesting evaluation of performance.

Bridge Skeleton

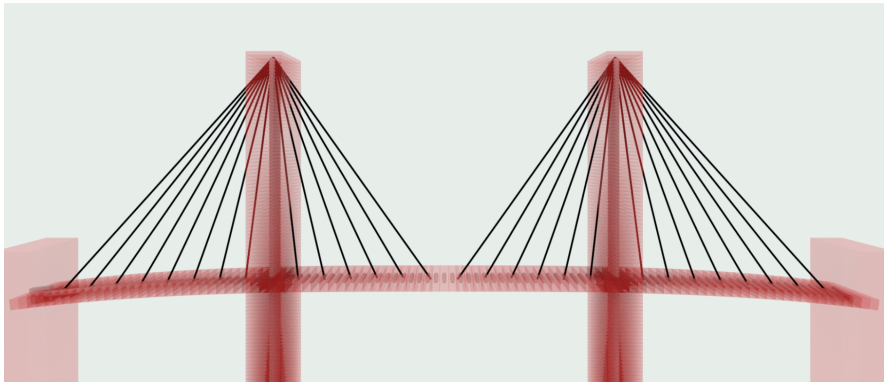

Fig. 2. Flexible tower bridge scene with exposed joints and links

The bridge scene in (Fig. 2) shows towers and deck comprised of a body-chain with variable stiffness. For the rigid tower case we define only the number of bodies in the deck, and when the towers are made to be flexible, and composed of a link chain, we keep the number of bodies in each of the towers and the bridge equivalent. A scene with flexible towers can be related to a rigid tower bridge where the bridge decking has three times as many DOFs as the deck in the flexible tower scene but the same number of DOFs overall.

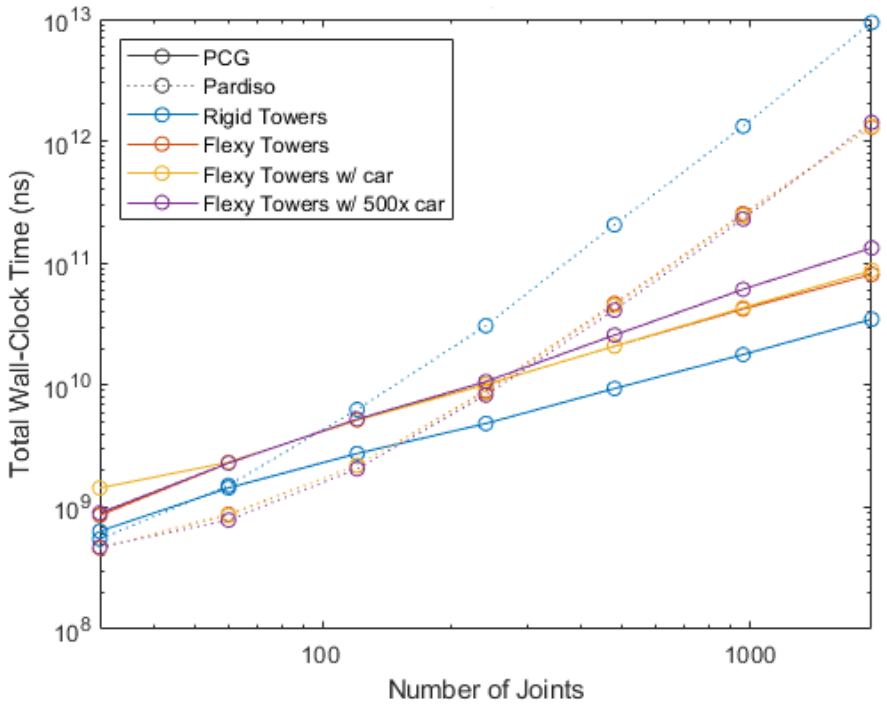

Fig. 3. Physical time cost of 5 seconds of simulation for the various bridge scenes

Fig. 3 displays the relative performance of different bridge scenes that we investigated. REDMAX features the most improvement over the direct approach for the bridge with rigid towers. This is not surprising considering that, when the towers are stationary, our block diagonal preconditioner becomes exactly the inverse of the system matrix, and so only the block diagonal portions of the local stiffness matrices enter the system matrix and PCG converges in *a single iteration*. That is to say, using REDMAX on the rigid towers scene (solid blue line) is a direct solve of the system. The next two scenes with flexible towers comprised of rigid body chains are slower than REDMAX for the rigid scene; this is expected since the towers are no longer stationary and the elastic cables between them and the bridge now impose off-diagonal terms and PCG requires more than a single iteration to converge.

Furthermore, scene one of these scenes introduces a weighted object, modeled after a car traversing the bridge in the 5 second simulation. For REDMAX this scene (yellow solid line) performs slightly worse than the flexible tower scene without the car. Again, this is expected as the car traversing the bridge is determined, but the small effect it has on the bridge deck and towers makes the warm start of PCG less accurate. In the same manner, the scene with a car weighing 500 times that of the first car (purple line) performs notably worse than all the previous scenes, as the extreme weight of this car causes dramatic bending in the towers and bridge (Fig. 2).

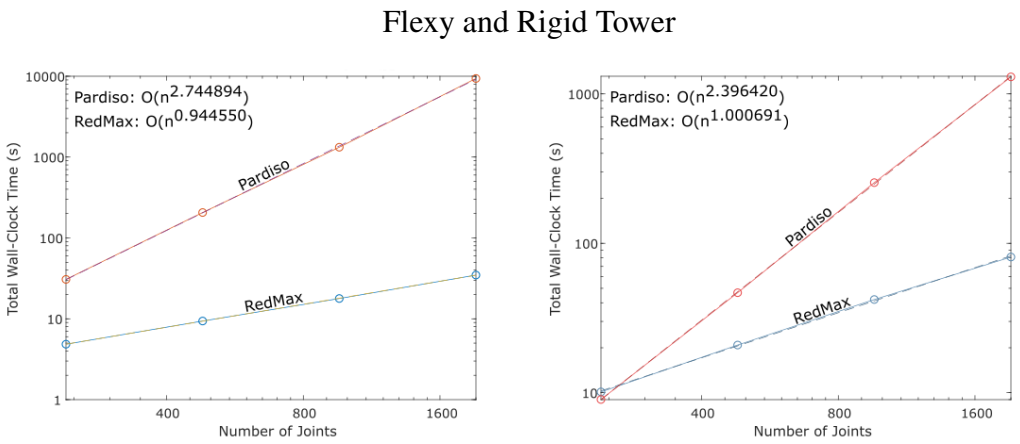

Fig. 4. Left: The bridge with rigid, infinitely stiff towers. Right: The bridge scene with chain-link towers. Dotted lines in the graphs represent the linear fit displayed on the graph.

Fig. 4 shows, on the left, the fitted timing result of the rigid tower bridge with a varied number of bodies comprising the decking of the bridge. Note that the first few points from Fig. 3 are not included in the fit since their behavior does not match the trend of larger scenes; we attribute these discrepancies to caching effects and Eigen container overhead. We follow this method for all asymptotic runtime evaluations. REDMAX still tends to have more overhead than the direct method for smaller scenes, but demonstrates better performance for larger scenes. With respect to the resolution of the deck, these results show nearly linear

growth for PCG, $O(n^{0.94})$, and worse than quadratic growth for the direct solver, $O(n^{2.74})$. REDMAX also performs better overall for scenes with 60 or more DOFs (seen in Fig. 3).

When the tower joints are not infinitely stiff, the system matrix has structure shown in Fig. 5. In this example the bridge and towers are each composed of 20 rigid bodies. The three dense 20 by 20 blocks centered on the diagonal represent the body chain

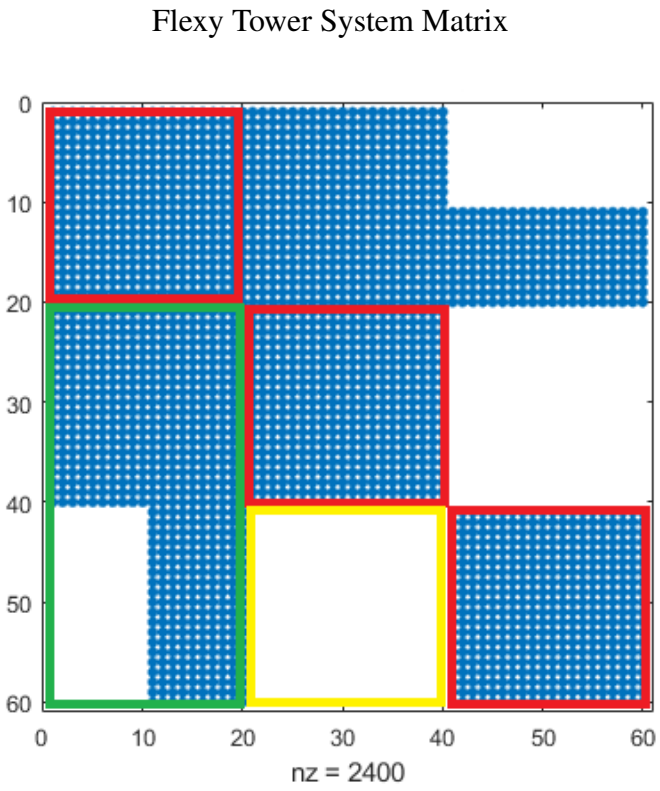

Fig. 5. Sparsity pattern of the system matrix for the bridge with flexible towers

of bridge and two towers (outlined in red), the off-diagonal block (highlighted in green) represents the tower's dependencies on the deck, and the 20 by 20 overlap of the towers (highlighted in yellow) shows that there are no direct dependencies between the two towers.

Fig. 4 shows, on the right, the timing result of the bridge with flexible towers. In this case, however, the bridge and each of the two towers are comprised of $\frac{1}{3}$ of the total number of links apiece. The same number of DOFs are present but, due to the off-diagonal terms in the system matrix, PCG no longer converges in just one iteration. The tower chains are made very stiff, as under normal conditions we would not expect a bridge to bend under the weight of the cars travelling across it. The results of this simulation show $O(n^{1.00})$ time for PCG, and $O(n^{2.39})$ for the direct solver.

REDMAX is expected to perform well when there is little movement, as the previous result will provide a closer warm start for PCG. However, given that there are plenty of potential scenes involving a lot of movement, we investigate the performance of PCG when the bridge scene with flexible towers has a car of varying weight traveling across it.

Flexy Tower with Car

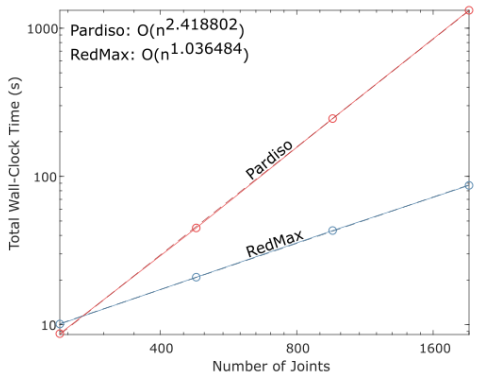

Fig. 6. Bridge with flexible towers and a car

Flexy Tower with Heavy Car

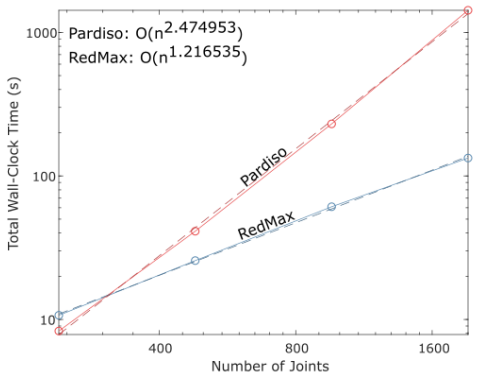

Fig. 7. Bridge with flexible towers and 500x car

Given the size of the modeled bridge in this scene, it would only physically fit about 600 average sized cars in stop-and-go traffic. If we model the bridge to be made from steel and concrete, then Fig. 6 represents the bridge scene with flexible towers and a medium sized car of average weight traversing the deck of the bridge. Fig. 7 features the same scene with

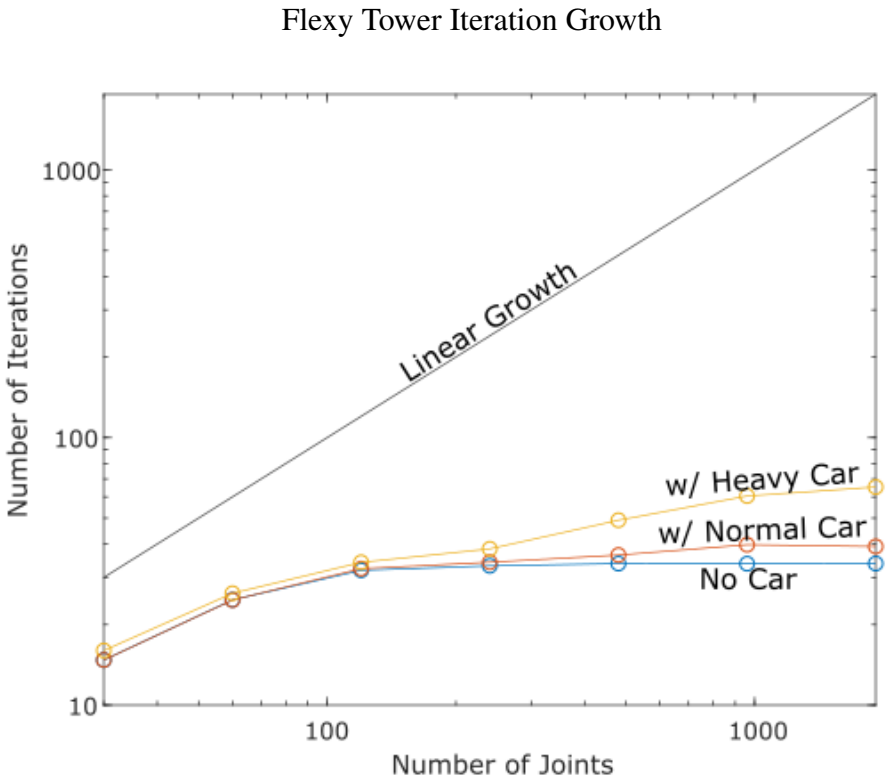

Fig. 8. The number of PCG iterations for the various bridge scenes

a single car weighing relatively 500 times that of an average, medium sized car. While we only roughly estimated the masses, sizes, and material composition of the elements in this scene, the desired outcome is a bridge simulation that seems to model intense bending in both the decking and the towers of the bridge as a result of an inordinately heavy load. We see that REDMAX performs better than the direct method for the scene with the regularly weighted car where there are more than 270 DOFs present, and, in the case of the scene with the inordinately heavy car, with more than 306 DOFs present. For both scenes, the asymptotic growth is much better for REDMAX then the direct solve. Even for the scene with lots of movement, with growth $O(n^{1.21})$ for REDMAX and $O(n^{2.47})$ for the direct method.

As expected, the number of PCG iterations required is greater when the rigid bodies move around more in the scene. Fig. 8 exemplifies this, showing that the scene without a car

(blue line), which we expect to have the least movement has the fewest iterations, while the scene with the heavy car (yellow line) and the most movement also takes the most iterations. Furthermore, we clearly see that the asymptotic growth of the number of iterations is sublinear for all scenes, as their slope is always less than the grey line exemplifying linear growth.

Fig. 9 shows how much bending occurs in the towers and bridge deck as a result of the car weighing 500 times the average weight of a car its size.

Buckling Bridge

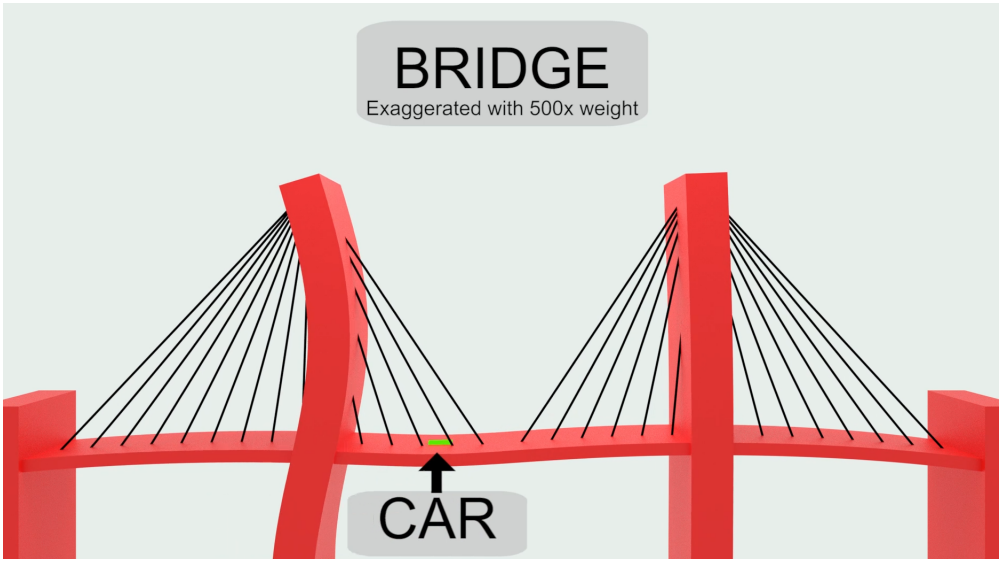

Fig. 9. Showing the bending of the flexible towers when an extremely heavy car is present

Fig. 10 shows the effect of many differently weighted cars as they traverse the bridge. This is interesting because the weight of the vehicles does not affect the performance of the direct solver, whereas for PCG the amount of clock time changes as the weight of the car increases. In Fig. 11 we see that the scene with the normally weighted car (dark blue) takes the least amount of time, and that, as the weight of the car increases, so does the amount of clock time required to simulate 5 seconds of scene time.

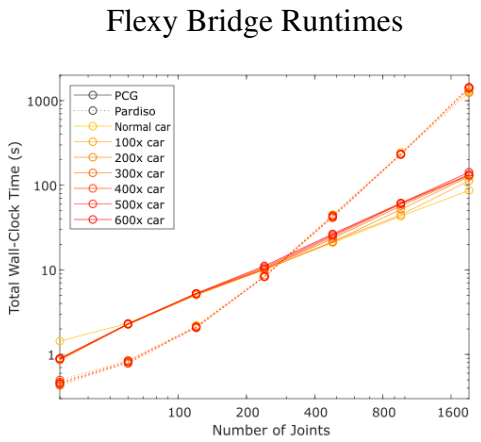

Fig. 10. Bridge with flexible towers and many differently weighted cars

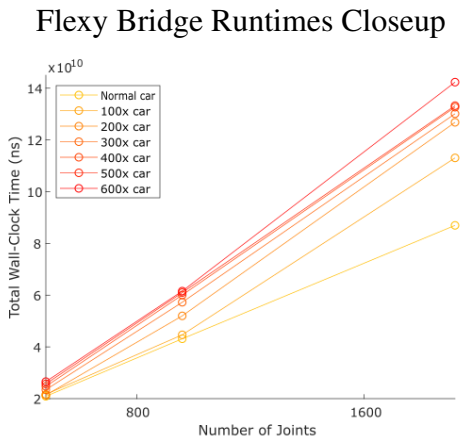

Fig. 11. Closeup on the runtime of PCG in the presence of cars with different weights

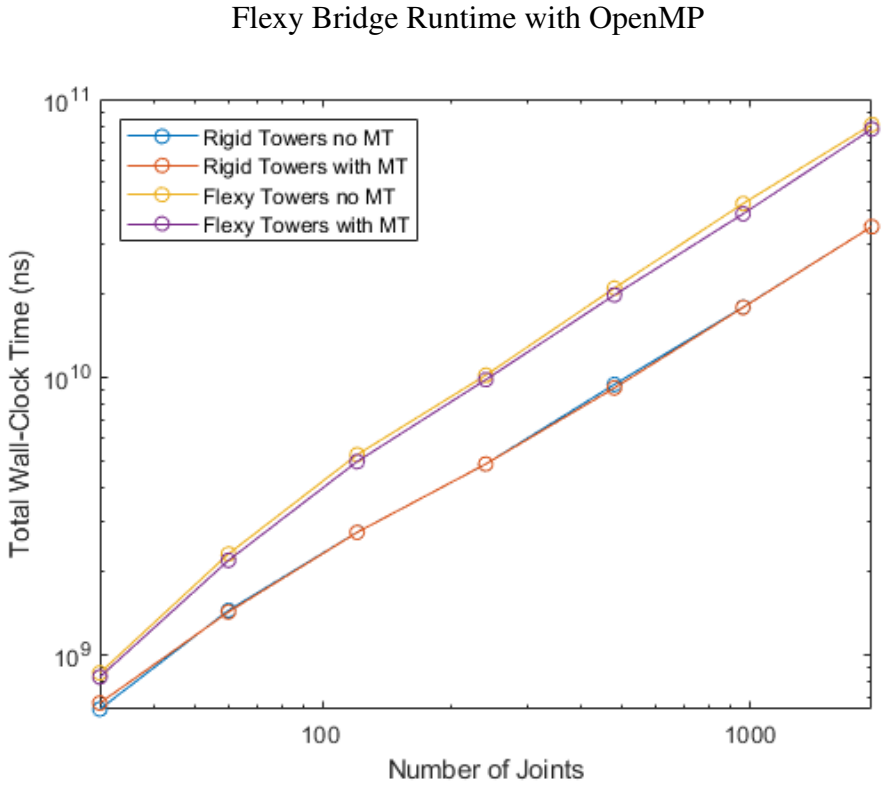

Fig. 12. The rigid and flexible tower scenes with and without parallelization

Pardiso, the direct solver, utilizes multi-threading and optimized sparse matrix multiplications. For the bridge we did not employ multi-threading, as we found that doing so did not

reliably or significantly improve the performance of either the rigid tower or felxible tower scene. Fig. 12 shows that for the scene with flexible towers (*i.e.,* more than one PCG iteration) parallelization speeds up the simulation only by about 5%. While, for the rigid tower scene, multi-threading had little to no effect on the runtime of the simulation.

4.2 Umbrella

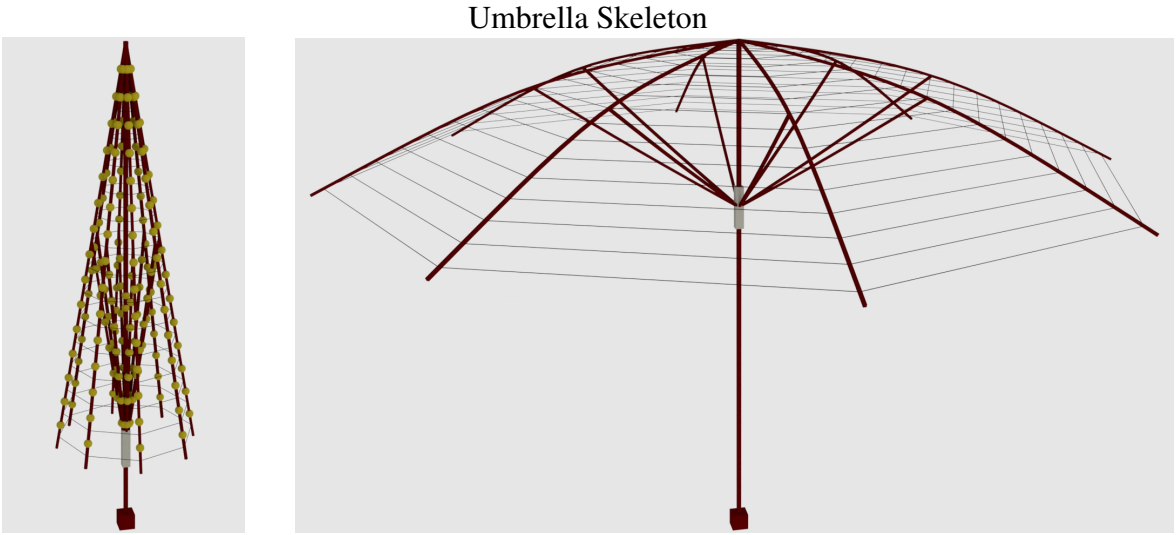

Fig. 13. Left: Umbrella scene with joints exposed. Right: Umbrella scene showing the open canopy

In this scene we model a deployable umbrella, Fig. 13. The root body is the *tube*, with 8 *ribs* attached to its tip. The 8 ribs are pushed open by the 8 *stretchers*, as the *runner*, connected to other end of all 8 stretchers, pushes along the tube. The runner has a prismatic joint with respect to the tube. Both the ribs and the stretchers are modeled using a sequence of universal joints, and are connected to each other by bilateral constraints. Springs are placed between the 8 ribs to model the canopy. For this scene, the number of bodies in the ribs and the stretchers

are varied. The system stiffness matrix dense dense diagonal regions that extend well beyond the block diagonal terms. This exemplifies a case when REDMAX is expected to perform well.

The umbrella as 8 revolute loop closing constraints (imposing 16 maximal constraints), one for each stretcher-rib sequence, where the bridge only had one. This means that PCG is called 18 times to solve the umbrella and 16 of these calls to PCG are independent from each other with respect to the DOF that they are limiting. This is where parallelization makes a difference, as the 2 PCG calls for each of the 8 loop closing constraints can all be executed separately in parallel. Provided the presence of enough threads to distribute the load, the solve for the umbrella can be even better than the bridge, which did not utilize parallelization as we discussed in the previous section. The remaining two PCG calls must be executed in order and only after PCG is called for each of the constraints.

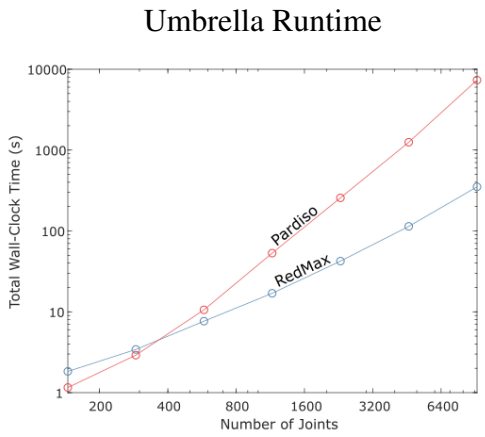

Fig. 14. The runtime of the umbrella scene

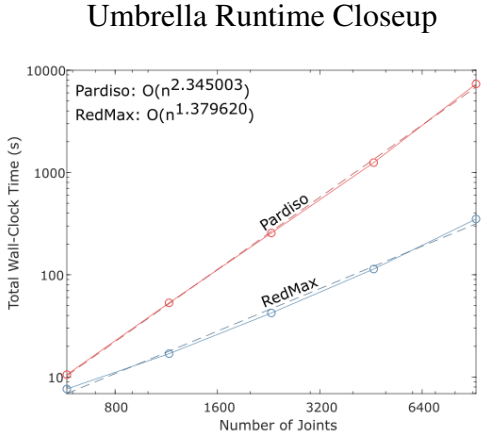

Fig. 15. Asymptotic evaluation of the umbrella

The i7-7700 CPU used in our tests has 8 threads, therefore the umbrella scene should scale similarly to the bridge scene. Fig. 14 shows the relative runtime of the two scenes, and

Fig. 15 shows the empirical orders of the two methods are $O(n^{1.38})$ and $O(n^{2.35})$ for PCG and direct, respectively. REDMAX is initially slower than the direct solver, but when there are more than 350 DOFs, PCG becomes faster.

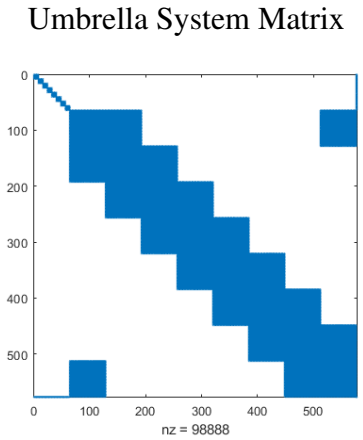

Fig. 16. Sparsity pattern of the system matrix

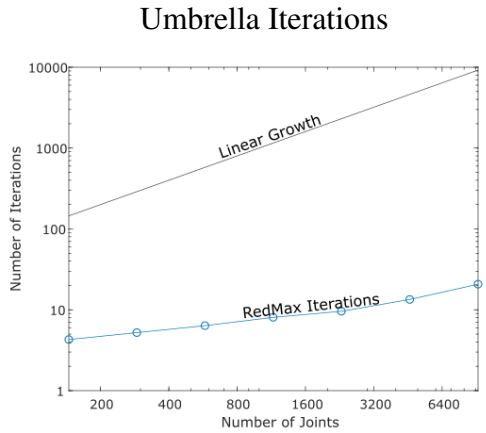

Fig. 17. PCG iterations of the umbrella scene

The umbrella scene is designed to test REDMAX for a setup expected to perform better than the direct solver due to the sparsity of the system matrix, while also depicting a scene that undergoes a lot of movement. The first block diagonal in the system matrix stores the stretcher chains, and the larger blocks along the diagonal show the dependencies between the ribs and the elastic forces, Fig. 16. Because the system matrix has large dense regions, we would expect REDMAX to perform better than Pardiso. We can see from Fig. 17 that, despite this scene featuring a lot of movement, the growth of the number of iterations in PCG is still clearly sublinear. Therefore, in this scene REDMAX has a much better asymptotic growth than the direct solver, but only outperforms Pardiso when there are many DOFs in the scene.

4.3 Tree

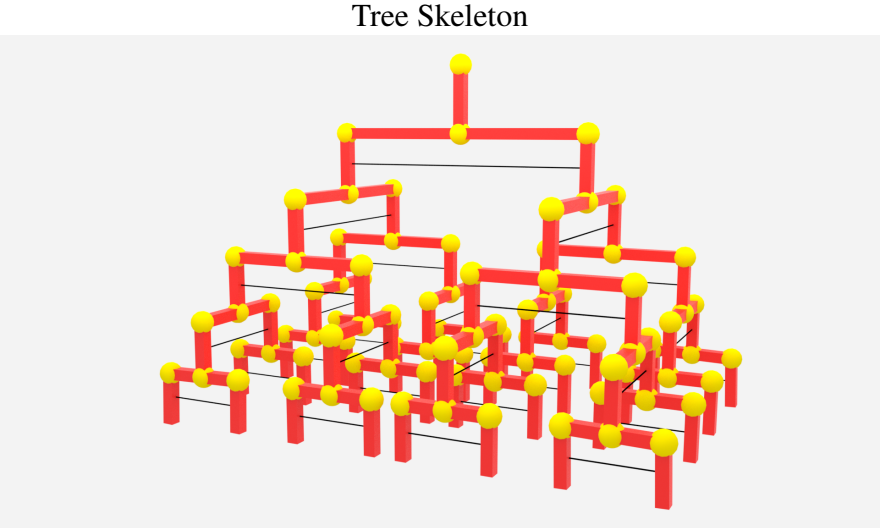

Fig. 18. Tree scene with exposed joints

This scene (Fig. 18) models a 3D binary tree and provides a *worst-case* evaluation of our preconditioner, as the mass matrix for this setup with many dependency branches is very sparse and enables optimized sparse direct system solvers (*e.g.,* Pardiso) to become sub-quadratic. Additionally, the sparse direct system solver has little overhead compared to our PCG method, which utilizes many small dense matrix computations in place of directly computing the sparse system matrix. Consequently, we expect that the method utilizing the sparse direct solve will be much faster than PCG for small systems, and for this case, with a sparse dependency tree, we also expect the direct method to scale well compared to PCG.

From the root body, each subsequent level adds a horizontal spacing body and two vertical bodies on either side. Only revolute joints are used, but the hinge axis of the joints

on the horizontal bodies rotate 90 degrees locally with each level. At each level springs are placed between the sibling nodes. Each corner leaf body has a spring force attached, pulling the corner leaves down and away from the tree. To ensure that the tree is always balanced, and that all scenes we are considering have the same movement pattern, we vary only the number of levels in the tree.

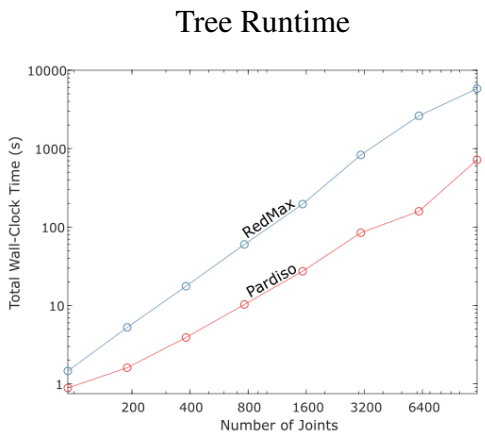

Fig. 19. The runtime of the tree scene

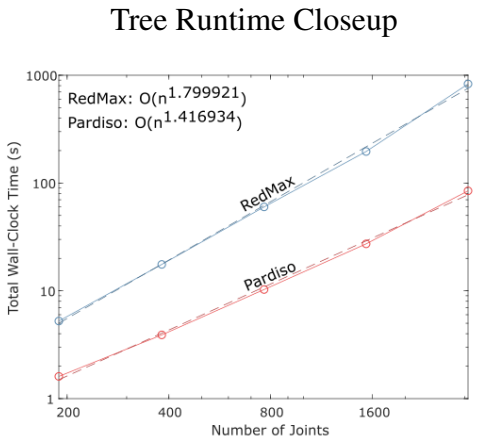

Fig. 20. Asymptotic evaluation of the tree

Fig. 19 shows the results of these simulations with a sparse system matrix. It is important to note that REDMAX does not perform better than the direct method for this scene, as the sparsity of the system matrix enables Pardiso to work more efficiently. For this scene even the asymptotic behavior of REDMAX is worse than the direct solver, Fig. 20.

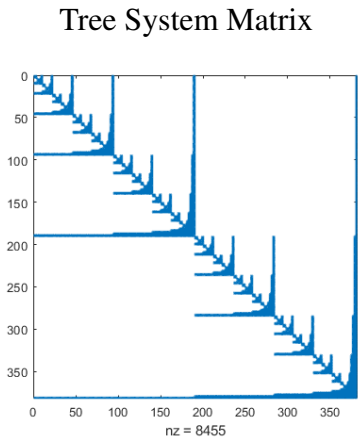

Fig. 21. The sparsity of the system matrix

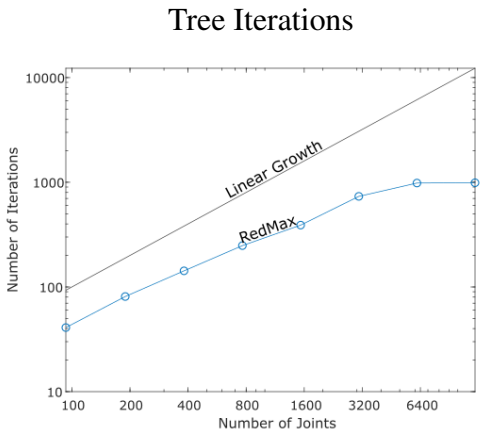

Fig. 22. Growth of iterations in PCG

Fig. 22 shows nearly linear growth of the number of PCG iterations with respect to the number of bodies in the scene, because the system matrix is so sparse each subsequent iteration of PCG learns very little about the scene. However, if we change the configuration of the scene slightly and move the elastic forces from between siblings on each level to a square pattern between indirect siblings on the lowest level, Fig. 23, then the resulting system matrix is shown in Fig. 24.

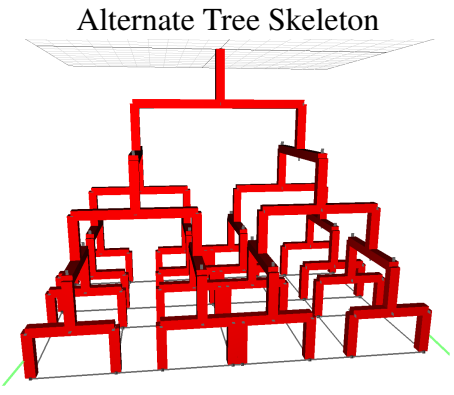

Fig. 23. The new tree scene

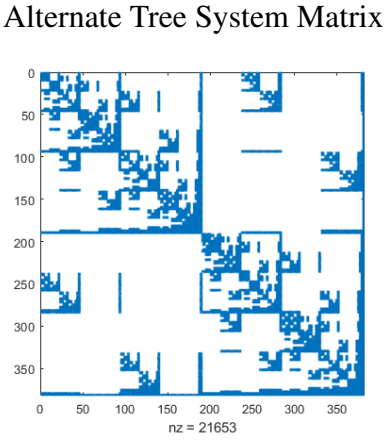

Fig. 24. The sparsity of the system matrix

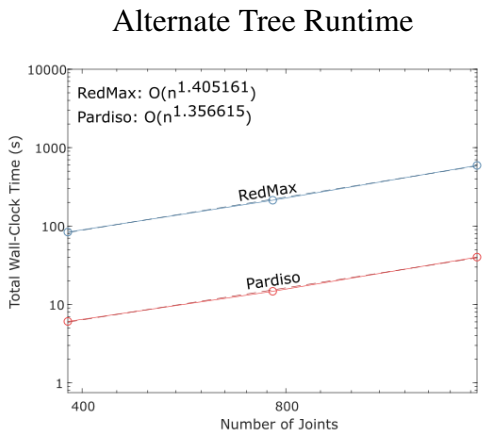

Fig. 25. Asymptotic evaluation of the new tree

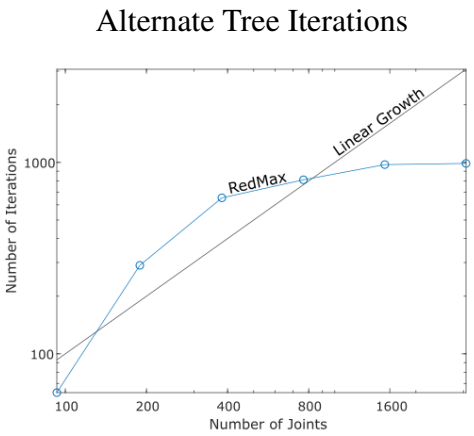

Fig. 26. Growth of the number of iterations in PCG

REDMAX still does not outperform the direct method, but, in this scene, we see that its asymptotic behavior is close to that of the direct solver. While REDMAX is not the best choice for highly sparse solves, it remains a robust and competitive method in many contexts.

5 SUMMARY AND CONCLUSIONS

We introduced an efficient and flexible approach for computing the dynamics of articulated rigid bodies. Unlike prior approaches that require $O(n^3)$ time, our approach maintains near linear performance, even in the presence of maximal stiffness matrix used by a linearly implicit integrator. In some simulation scenarios, our preconditioned solver converges in a single iteration. Our approach also provides flexibility, allowing us to mix and match implicit and explicit forces in both reduced and maximal coordinates, as well as bilateral and unilateral constraints in either coordinates. We showed this flexibility with several results including those that use hybrid dynamics in both coordinates and fully two-way coupled dynamics of articulated and deformable bodies. The C++ implementation of REDMAX is available open-source at github.

5.1 Further Study

Although the theoretical runtime of factorization methods are $O(n^3)$ [De Jalon and Bayo 2012], in practice, they exhibit better asymptotic behavior depending on the sparsity pattern of the system matrix. When the scene has many branches, the system often becomes very sparse, and these methods become subquadratic, with very small overhead compared to our PCG method. Automatically detecting when to switch between the two methods would be of practical interest. This is especially true since we have only shown *empirically* that PCG takes a sublinear number of iterations using our preconditioner, without a formal proof.

We have not taken into account all avenues of parallelization or any GPU implementations. Some parts of this approach could be easily parallelizable (*e.g.,* each branch in the tree topology can be processed in parallel). Such improvement might help REDMAX to be competitive in cases where the sparsity of the system matrix better suits existing sparse linear system solvers. Existing $O(n)$ and $O(n^3)$ methods have shown good parallelizability

(*e.g.,* [Avello et al. 1993; Negrut et al. 1997]), and so we believe it is worthwhile to explore similar techniques that work even with the inclusion of the stiffness matrix.

REFERENCES

Avello, A., Jiménez, J. M., Bayo, E., and de Jalón, J. G. (1993). A simple and highly parallelizable method for real-time dynamic simulation based on velocity transformations. *Computer Methods in Applied Mechanics and Engineering*, 107(3):313–339.

Bae, D.-S. and Haug, E. J. (1987a). A recursive formulation for constrained mechanical system dynamics: Part I. open loop systems. *Journal of Structural Mechanics*, 15(3):359–382.

Bae, D.-S. and Haug, E. J. (1987b). A recursive formulation for constrained mechanical system dynamics: Part II. closed loop systems. *Journal of Structural Mechanics*, 15(4):481–506.

Baraff, D. (1996). Linear-time dynamics using lagrange multipliers. In *Annual Conference Series (Proc. SIGGRAPH)*, pages 137–146.

Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation. In *Annual Conference Series (Proc. SIGGRAPH)*, pages 43–54.

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

Bertails, F. (2009). Linear time super-helices. *Computer Graphics Forum (Proc. Eurographics)*, 28(2):417–426.

Cline, M. B. and Pai, D. K. (2003). Post-stabilization for rigid body simulation with contact and constraints. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 3744–3751.

De Jalon, J. G. and Bayo, E. (2012). *Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time Challenge*. Springer Science & Business Media.

Delp, S. L., Anderson, F. C., Arnold, A. S., Loan, P., Habib, A., John, C. T., Guendelman, E., and Thelen, D. G. (2007). OpenSim: Open-source software to create and analyze dynamic

simulations of movement. *IEEE Transactions on Biomedical Engineering*, 54(11):1940–1950.

Demaine, E. D. and O'Rourke, J. (2008). *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, reprint edition.

Deuss, M., Panozzo, D., Whiting, E., Liu, Y., Block, P., Sorkine-Hornung, O., and Pauly, M. (2014). Assembling self-supporting structures. *ACM Transactions on Graphics*, 33(6):214:1–214:10.

Drumwright, E. (2012). Fast dynamic simulation of highly articulated robots with contact via $\theta(n^2)$ time dense generalized inertia matrix inversion. In *Int. Conf. on Sim., Model., & Prog. for Auton. Robots*, pages 65–76. Springer.

Featherstone, R. (1983). The calculation of robot dynamics using articulated-body inertias. *The International Journal of Robotics Research*, 2(1):13–30.

Hadap, S. (2006). Oriented strands: Dynamics of stiff multi-body system. In *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '06, pages 91–100.

Hernandez, F., Garre, C., Casillas, R., and Otaduy, M. A. (2011). Linear-time dynamics of characters with stiff joints. In *V Ibero-American Symposium on Computer Graphics (SIACG 2011)*. The Eurographics Association and Blackwell Publishing Ltd.

Jain, S. and Liu, C. K. (2011). Controlling physics-based characters using soft contacts. *ACM Transactions on Graphics*, 30(6):163:1–163:10.

Kim, J. (2012). Lie group formulation of articulated rigid body dynamics. Technical report, Carnegie Mellon University.

Kim, J. and Pollard, N. S. (2011). Fast simulation of skeleton-driven deformable body characters. *ACM Transactions on Graphics*, 30(5):121:1–121:19.

Liu, L., Yin, K., Wang, B., and Guo, B. (2013). Simulation and control of skeleton-driven soft body characters. *ACM Transactions on Graphics*, 32(6):215:1–215:8.

Negrut, D., Serban, R., and Potra, F. A. (1997). A topology-based approach to exploiting sparsity in multibody dynamics: Joint formulation. *Journal of Structural Mechanics*, 25(2):221–241.

Park, F. C., Bobrow, J. E., and Ploen, S. R. (1995). A lie group formulation of robot dynamics. *The International Journal of Robotics Research*, 14(6):609–618.

Popov, E. P., Vereshchagin, A. F., and Zenkevich, S. L. (1978). Robot manipulators: Dynamics and algorithms.

Quigley, E., Yu, Y., Huang, J., Lin, W., and Fedkiw, R. (2018). Real-time interactive tree animation. *IEEE Transactions on Visualization and Computer Graphics*, 24(5):1717–1727.

Roberson, R. E. (1966). A dynamical formalism for an arbitrary number of interconnected rigid bodies with reference to the problem of satellite attitude control. *Proc. 3rd Congr. of Int. Fed. Automatic Control*, 1:46D1–46D8.

Selig, J. M. (2004). Lie groups and Lie algebras in robotics. In *Computational Noncommutative Algebra and Applications*, pages 101–125. Springer.

Serban, R., Negrut, D., Haug, E. J., and Potra, F. A. (1997). A topology-based approach for exploiting sparsity in multibody dynamics in cartesian formulation. *Journal of Structural Mechanics*, 25(3):379–396.

Shabana, A. A. (2013). *Dynamics of Multibody Systems*. Cambridge University press.

Shinar, T., Schroeder, C., and Fedkiw, R. (2008). Two-way coupling of rigid and deformable bodies. In *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 95–103.

Tournier, M., Nesme, M., Gilles, B., and Faure, F. (2015). Stable constrained dynamics. *ACM Trans. Graph.*, 34(4):132:1–132:10.

Walker, M. W. and Orin, D. E. (1982). Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, 104(3):205–211.

Wang, J. M., Hamner, S. R., Delp, S. L., and Koltun, V. (2012). Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Transactions on Graphics*, 31(4):25:1–25:11.

Wasfy, T. M. and Noor, A. K. (2003). Computational strategies for flexible multibody systems. *Applied Mechanics Reviews*, 56(6):553–613.

Whiting, E., Shin, H., Wang, R., Ochsendorf, J., and Durand, F. (2012). Structural optimization of 3D masonry buildings. *ACM Transactions on Graphics*, 31(6):159:1–159:11.

Zhou, Y., Sueda, S., Matusik, W., and Shamir, A. (2014). Boxelization: Folding 3D objects into boxes. *ACM Transactions on Graphics*, 33(4):71:1–71:8.