Computers and Structures 281 (2023) 107015

Contents lists available at ScienceDirect

Computers and Structures

journal homepage: www.elsevier.com/locate/compstruc

Scalable mesh partitioning for multibody-3D finite element based rotary-wing structures

Ravi Lumba [1,*], Anubhav Datta [2]

University of Maryland, College Park, MD 20742, USA

ARTICLE INFO

Article history:
Received 11 June 2022
Accepted 6 March 2023

Keywords:
High-performance computing
3D structures
Rotorcraft dynamics
Domain decomposition
Mesh partitioning

ABSTRACT

This paper presents a new and specialized mesh partitioner for large-scale three-dimensional multibody rotor dynamic models. This partitioner enables parallel solution with modern domain decomposition algorithms of complex, multibody, three-dimensional finite element problems. A parallel solver using a state-of-the-art iterative substructuring algorithm, FETI-DP, is developed in this paper to solve the partitioned data structures. The main feature of the partitioner is the ability to robustly partition any generic multibody structure, although it has several special features for rotary-wing structures. The NASA Tilt Rotor Aeroacoustic Model (TRAM), a 1/4 scale V-22 model, was specially released by NASA as a challenge test case. This model contains four flexible parts, six joints, 18 composite material decks, a blade fluid–structure interface, and control angle inputs. The parallel solver scalability is studied for progressively increasing complexity, from the isolated rotor blade to the blade and hub assembly. The use of a skyline solver for the coarse problem is shown to eliminate the coarse problem barrier. The special partitioner features are demonstrated to enable efficient parallel solution and significantly improve the performance and scalability. The principle barrier of computational time that prevented the use of high-fidelity three-dimensional structures in rotorcraft is thus resolved.

1. Introduction

This paper presents the development and application of a specialized mesh partitioner for a parallel and scalable three-dimensional (3D) finite element (FE) multibody solver built for complex rotary-wing structural dynamics problems. A 3D FE-multibody solver faces several barriers for routine use in the design of real rotors, chief among which is computational time. A specialized mesh partitioner, which partitions the full multibody system for parallel analysis, is the focus of this paper and promises to expand the application of such a solver from academic to realistic. The NASA Tilt Rotor Aeroacoustic Model (TRAM), a 1/4 scale V-22 proprotor, is used as a realistic test case to demonstrate the robustness of the partitioner and scalability of the solver. This test case was specially released for this project, with detailed drawings of its internal structure, to provide a modern challenge problem.

A robust, parallel, and scalable 3D FE-multibody solver for rotary-wing applications was envisioned for the design of modern rotorcraft by NASA in 2008 [1]. Conventional analyses uses 1D composite beam-based models for the blades, which imposes assumptions and limitations on the analysis. Moving to 3D removes these limitations, increasing both the scope and accuracy of analysis. However, the many multibody bearings and load paths associated with a modern rotor create complications for domain decomposition algorithms. The development of a new, specialized mesh partitioner was necessary due to a lack of suitable options. Current mesh partitioners, such as METIS, use graph-based partitioning, making them fast and efficient for regular FE problems. However, the addition of joints to model bearings and the constraints they impose on partitioning cannot easily be handled by these existing partitioners, justifying the need for a new, specialized mesh partitioner.

1.1. Background and motivation

Current rotorcraft structural dynamics analyses model the rotor blade as a slender one-dimensional (1D) composite beam, with two-dimensional (2D) finite element models used to calculate the structural and inertial cross-sectional properties of the blade [2]. This approach is well documented and provides adequate results for conventional rotorcraft problems due to the long slender shape of rotor blades. However, this method falls short in several key areas.

* Corresponding author.
E-mail addresses: rlumba@umd.edu (R. Lumba), datta@umd.edu (A. Datta).
[1] NSF Fellow, Alfred Gessow Rotorcraft Center, USA.
[2] Associate Professor, Alfred Gessow Rotorcraft Center, and AIAA Associate Fellow, USA.

https://doi.org/10.1016/j.compstruc.2023.107015
0045-7949/

Modern blades have material and geometric discontinuities due to design or ballistic damage that cannot be captured using beams. Modern hub components are 3D structures that provide kinematic couplings and absorb maximum stresses and hence are crucial for both stability and weight. Today modifications and enhancements are made to beam boundary conditions to accommodate these structures in a gross manner, but require fabrication and testing to measure their properties. Dynamic stresses cannot be calculated. The use of 3D models has the potential to overcome these barriers. The flexible parts of a rotor, the parts that produce strains, can be modeled from first principles with 3D finite elements (all bricks) and bearings that provide constraints modeled with multibody joints. Dynamic stresses and strains can be predicted in the blade and the hub components up front in the design process.

The use of higher dimensional finite elements in rotorcraft analysis is routine for the fuselage. Fuselage models are typically created in NASTRAN-like software with beam and shell elements. The U.S. Navy DAMVIBS program [3] summarized the state-of-the-art across U.S. industry with full-scale validation up to the 1990s. Since then some of these modes have been coupled with rotor analysis [4,5], but the rotor was still modeled using beams. Rotor dynamics require exact gyroscopic terms coupled with trim controls that are missing in traditional finite elements. It also requires aerodynamics. Over the past two decades, rotorcraft computational fluid dynamics has matured to the point that unsteady Reynolds-averaged Navier–Stokes computations can be carried out on practical problems with tens millions of grid points on thousands of cores. This has improved airload predictions, but the increased accuracy does not flow into structural stresses as they are tied to lower order beams [6]. The emergence of 3D structures for rotors will equalize the imbalance in fidelity. Recognizing this, the development of a parallel and scalable 3D FE-multibody solver was identified by NASA [1] as a central component for next-generation high-fidelity rotorcraft analysis, with work undertaken at the University of Maryland (UMD).

The new U.S. Army/UMD solver X3D [7] uses 3D brick finite elements unified with multibody dynamics. Three-dimensional structural models with meshes and joints are created directly from CAD models. This allows for the exact geometry, including any material discontinuities, to be captured with no assumptions. Staruk et al. [8,9] demonstrated Integrated 3D (I3D) aeromechanics modeling using X3D, defined as the coupling of 3D structures with 3D aerodynamics. A tiltrotor conversion from helicopter to airplane, considered one of the most complicated flight regimes where the critical design loads are encountered, was simulated. However, the demonstration was limited to coarse meshes because there was no partitioning tool. Even with the coarse mesh, the solution time was nearly 10 days. This remains the principal barrier to using 3D structures for design of new rotors. The objective of this paper is to break this barrier.

Modern iterative substructuring provides a path to breaking this barrier, if an appropriate partitioner can be designed. The presence of multibody joints connecting many flexible parts break the structure of finite element matrices, lead to excessive condition numbers ($\kappa > 1 \times 10^{12}$), and break the rules of iterative substructuring unless partitioned carefully to ensure non-null kernels. There are no mesh partitioners currently available that can breakup a 3D multibody problem while meeting the above challenges. Trying to mitigate these issues while retaining scalability requires a specialized mesh partitioner for helicopter rotors.

The finite element tearing and interconnecting dual primal (FETI-DP) method is a state-of-the-art iterative substructuring method introduced by Farhat [10,11]. It is a domain decomposition algorithm developed for structural mechanics, and hence uses non-overlapping subdomains. The subdomains are solved by direct factorization which works even for high condition numbers. The interface is divided into a primal and dual problem. The primal problem consists of a small subset of the interface, while the dual problem is the remaining subset. An iterative solver, typically equipped with a preconditioned conjugate gradient method, solves the dual problem while a direct solver solves the primal problem. For 3D elements, Farhat noted that the size of the primal problem grew quickly with increasing subdomains, reducing computational efficiency. Limiting the size of the primal interface produced better efficiency, but at the cost of speed of convergence. Instead, enriching the primal problem by adding a set of Lagrangian multipliers was suggested [10].

Datta and Johnson [12] introduced 3D modeling for rotary-wing dynamics and adapted the FETI-DP algorithm for parallel solution. They concluded that joints must be left out of the interface. But they were unable to partition practical geometries around joints, so they used idealized geometries for demonstration. They also encountered the same barrier observed by Farhat that the primal problem grew rapidly with increasing subdomains, a problem aggravated by the need to partition around joints. The authors addressed this problem by introducing special edge nodes with redundant dual variables but this increased the number of iterations and made partitioning even more complicated [13]. The authors noted that a specialized mesh partitioner would be needed to extend their work to realistic problems. The objective of this paper was to develop this partitioner.

Currently, there are several mesh partitioners used for structural mechanics problems such as Zoltan [14], developed by Sandia, and METIS [15], developed at the University of Minnesota. While both offer an array of partitioning options and load balancing schemes, neither is equiped to handle multibody structures. This paper proposes a partitioner that can handle multibody structures. While the partitioner can decompose any generic mesh it is demonstrated in the context of a complex helicopter geometry which motivated this work: the NASA TRAM Proprotor.

1.2. Organization of paper

The paper is organized as follows. First, the NASA TRAM Proprotor is presented. Then, the parallel solver is introduced, with the FETI-DP algorithm reviewed to establish the general requirements for the mesh partitioner. Next, the partitioner is described in detail, along with special features added for complex rotorcraft problems. Then the partitioner and solver are demonstrated on the NASA TRAM rotor and validated against test data. The final few sections study the performance and scalability of the solver, with emphasis on the special features needed for helicopter rotors. Lastly, the conclusions are summarized.

2. NASA TRAM proprotor

The geometry is that of the NASA Tiltrotor Aeroacoustic Model (TRAM). It is a 1/4 Mach-scale rotor designed to match the first flap, lag, and torsion frequencies of the V-22 Osprey tiltrotor aircraft. The TRAM proprotor was chosen as a test case for this paper as it is an accurate representation of a realistic modern rotor with multiple flexible hub components. The detailed internal drawings and materials of TRAM were made available by NASA for this task.

2.1. Geometry

The CAD was created from engineering drawings in CATIA and is shown in Fig. 1. An exploded view of the hub is shown in Fig. 2. The four parts highlighted in boxes are flexible parts and must be meshed. The two bearings, also boxed, are multibody

Fig. 1. The TRAM model.

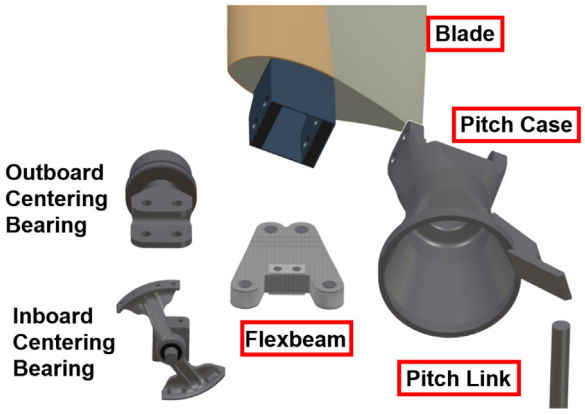


Fig. 2. The complex root end; four flexible parts (red) and six multibody joints. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

joints. In addition there is a gimbal at the hub and joints at the top and bottom of the pitch link through which control angles are input. The blade and flexbeam are composite structures containing 11 and 7 different materials respectively. The pitch case and pitch link are made of aluminum.

The four flexible parts are interconnected by 6 multibody joints producing two parallel load paths from the blade to the hub. From the blade, all loads are transferred through the grips to the pitch case. The spherical outboard bearing between the pitch case and flexbeam passes most of forces to the flexbeam while the moments continue through the pitch case. The radial inboard bearing also connects the flexbeam and pitch case, and passes only forces again to the flexbeam. The flexbeam transfers the forces to the hub via a bolt attachments at its root. The blade torsion loads are carried via the pitch case to the pitch link. The joints are placed strategically to govern natural frequencies, provide kinematic couplings for stability, admit control inputs, and distribute moments and shears across the flexible parts to absorb dynamic stresses at minimum weight. These details can be found in Staruk [8,9]. The partitioner must retain the proper physics of these load paths and allow for a varying number of partitions for each of the flexible parts, all while load balancing the full problem.

2.2. TRAM finite element model

The CAD model was meshed using the Sandia software Cubit. Each flexible part is meshed independently based on its resolution needs. The multibody joints that connect the flexible parts have six degrees of freedom (DOF): three translational and three rotational. These can be locked or have mass, stiffness, and damping assigned to match bearing properties. A joint motion can be commanded, which is how the collective and cyclic pitch inputs are introduced at the bottom of the pitch link. The 6 joints are summarized as follows:

1. blade to pitch case grips — representing the bolts,
2. pitch case to flexbeam — representing the outboard centering bearing,
3. pitch case to flexbeam — representing the inboard centering bearing,
4. flexbeam to hub center of rotation — representing the gimbal,
5. top of pitch link to pitch horn — representing a spherical bearing, and
6. bottom of pitch link to control pitch input – representing another spherical bearing.

The full TRAM mesh is presented in Fig. 3, with sizes for each component listed in Table 1. The number of nodes and elements connected to each joint are also shown. The nodes associated with each joint are eliminated by the solver and replaced by the joint motions, of which there can be up to 6. The total number of DOF is approximately 250,000.

3. Parallel solver

The parallel solver is reviewed in this section to help understand the requirements for the partitioner and help interpret the scalability results shown later. The partitioner is generic and not specific to FETI-DP, but FETI-DP forces it to provide higher-level data structures applicable to any non-overlapping substructuring algorithm. The core solver acts on a symmetric linearized system

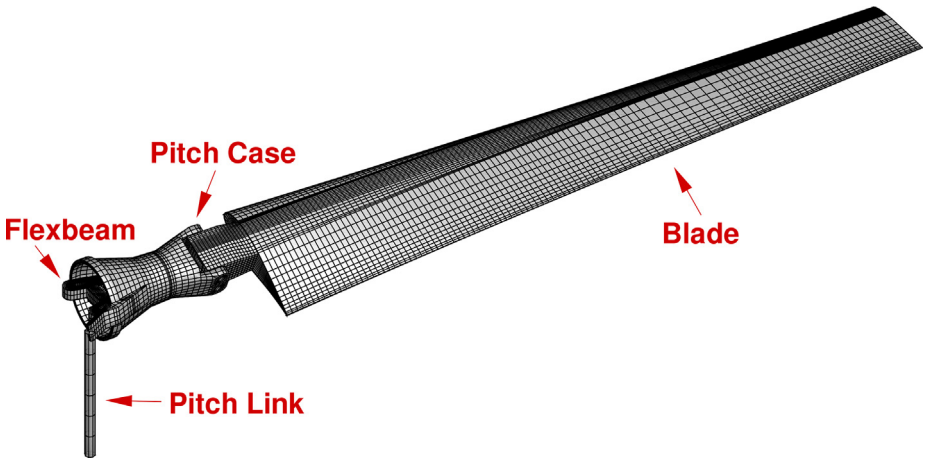


Fig. 3. The TRAM structural mesh, including 4 flexible parts and 6 multibody joints.

Table 1
TRAM mesh component information.

Component	Nodes	Elements
Fine Blade	77,217	8,532
Pitch Case	4,852	408
Flexbeam	8,575	844
Pitch Link	63	3
J1. Blade Root to Pitch Case	842	118
J2. Outboard Pitch Case to Flexbeam	918	192
J3. Inboard Pitch Case to Flexbeam	1560	242
J4. Flexbeam to Ref.	798	132
J5. Pitch Link to Pitch Horn	94	11
J6. Pitch Link to Ref.	9	1

$$Ku = f \tag{1}$$

where K is a structural stiffness, u is the solution at the nodes and joints, and f is the forcing.

3.1. The FETI-DP algorithm

A quick review of the basic FETI-DP algorithm is presented [10,11]. This algorithm partitions the structure into non-overlappping subdomains. The subdomain problem is solved with a direct solver, while the interface is solved iteratively. The algorithm constructs the interface and preconditioner subdomain-by-subdomain in a fully parallel manner. The interface is then iterated in Krylov subspace for convergence.

The subdomain nodes are divided into internal nodes and interface nodes, where interface nodes are shared by multiple subdomains. The interface is further divided into corner, edge, and face nodes as shown in Fig. 4. The governing equation for a single subdomain s is shown below with internal (I), face and edge (E), and corner nodes (C) formally regrouped one after the other.

$$\begin{bmatrix} K^s_{II} & K^s_{IE} & K^s_{IC} \\ K^s_{EI} & K^s_{EE} & K^s_{EC} \\ K^s_{CI} & K^s_{CE} & K^s_{CC} \end{bmatrix} \begin{bmatrix} u^s_I \\ u^s_E \\ u^s_C \end{bmatrix} = \begin{bmatrix} f^s_I \\ f^s_E \\ f^s_C \end{bmatrix} \tag{2}$$

All the non-corner nodes — internal, face, and edge can be lumped together into a single set of remaining nodes u^s_R.

$$\begin{bmatrix} K^s_{RR} & K^s_{RC} \\ K^s_{CR} & K^s_{CC} \end{bmatrix} \begin{bmatrix} u^s_R \\ u^s_C \end{bmatrix} = \begin{bmatrix} f^s_R \\ f^s_C \end{bmatrix} \tag{3}$$

The corner nodes are meant to create a coarse representation of the full problem. This is somewhat equivalent to treating each subdomain as a pseudo-element, whose nodes are the corner nodes. Hence the corner problem is also called the coarse problem. The corner variables, also called primal variables, remain the original DOF u. A Boolean matrix B^s_C is introduced to extract the subdomain corner nodes from the global problem.

$$B^s_C u_C = u^s_C \tag{4}$$

The continuity across the non-corner interface nodes is ensured by introducing an additional Boolean matrix B^s_R. It picks up the non-corner interface from u^s_R and applies the proper sign ($\pm$ 1) so that Eq. (6) is satisfied at convergence, where $s = 1, 2, 3, \ldots, N$ is a summation over all subdomains.

$$B^s_R u^s_R = \pm u^s_E \tag{5}$$

$$\sum_{s=1}^{N} B^s_R u^s_R = 0 \tag{6}$$

For an interface node that only touches two subdomains (face node) the above equation simplifies to $u^1_1 - u^2_1 = 0$, or simply equating the solution across subdomains. However, for edge nodes (nodes that touch more than 2 subdomains), one equation is not enough to ensure continuity. For X shared subdomains, X-1 equations are needed to ensure continuity. Normally, adding redundant dual variable improves convergence (Ref. [13]). In this paper the maximum number of redundant dual variables is used for all edge nodes. For an edge node common to two subdomains, this gives us the one equation presented previously. However, if an edge node is common to three subdomains, three equations are used (one more than necessary): $u^1_1 - u^2_1 = 0, u^1_1 - u^3_1 = 0$, and $u^2_1 - u^3_1 = 0$; and if an edge node is common to four subdomains, six equations are used (three more than necessary): $u^1_1 - u^2_1 = 0, u^1_1 - u^3_1 = 0, u^1_1 - u^4_1 = 0, u^2_1 - u^3_1 = 0, u^2_1 - u^4_1 = 0$, and $u^3_1 - u^4_1 = 0$.

Using Eq. (3) and (4), the subdomain equations of equilibrium are written as the following:

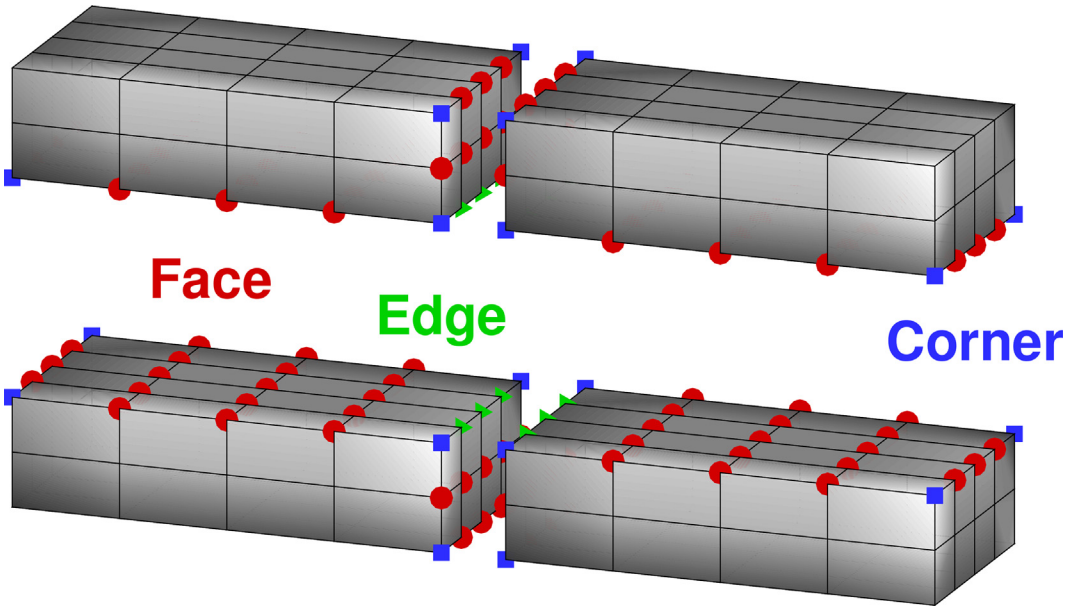


Fig. 4. Description of interface nodes; the red circles indicate face nodes, the green triangles edge nodes, and blue squares mark the corner nodes. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$$K_{RR}^s u_R^s + K_{RC}^s B_C^s u_C = f_R^s - B_R^{s\,T} \lambda^s \tag{7}$$

$$\sum_{s=1}^{N_s} B_C^{s\,T} K_{CR}^s u_R^s + \sum_{s=1}^{N_s} B_C^{s\,T} K_{CC}^s B_C^s u_C = \sum_{s=1}^{N_s} B_C^{s\,T} f_C^s = f_C, \tag{8}$$

where set of Lagrangian multipliers λ, also termed dual variables, is introduced to enforce continuity across the interface, and λ^s is the subdomain restriction. An explicit form for the subdomain nodes u_R^s is obtained by rearranging Eq. (7).

$$u_R^s = K_{RR}^{s\,-1}\left(f_R^s - B_R^{s\,T}\lambda^s - K_{RC}^s B_C^s u_C\right) \tag{9}$$

For this equation, K_{RR} must be invertible. This requires that the corner nodes be selected such that each subdomain has no rigid body modes – there must be at least three noncollinear corner nodes per subdomain.

Next, the following system of equations is created by: (1) substituting Eq. 9 into Eq. 6, and (2) substituting Eq. (9) into Eqs. (7) and (8).

$$\begin{bmatrix} F_{RR} & F_{RC} \\ F_{RC}^T & -K_{CC}^* \end{bmatrix} \begin{bmatrix} \lambda \\ u_C \end{bmatrix} = \begin{bmatrix} d_r \\ -f_C^* \end{bmatrix} \tag{10}$$

where,

$$F_{RR} = \sum_{s=1}^{N_s} B_R^s K_{RR}^{s\,-1} B_R^{s\,T} \tag{11}$$

$$F_{RC} = \sum_{s=1}^{N_s} B_R^s K_{RR}^{s\,-1} K_{RC}^s B_C^s \tag{12}$$

$$K_{CC}^* = \sum_{s=1}^{N_s} B_C^{s\,T} K_{CC}^s B_C^s - \sum_{s=1}^{N_s} B_C^{s\,T} K_{CR}^s K_{RR}^{s\,-1} K_{RC}^s B_C^s \tag{13}$$

$$d_R = \sum_{s=1}^{N_s} B_R^s K_{RR}^{s\,-1} f_R^s \tag{14}$$

$$f_C^* = \sum_{s=1}^{N_s} B_C^{s\,T} f_C^s - \sum_{s=1}^{N_s} B_C^{s\,T} K_{CR}^s K_{RR}^{s\,-1} f_R^s \tag{15}$$

This is a dual-primal problem, as it relates the primal displacements at the corner nodes u_C to the dual Lagrange multipliers λ. By eliminating u_C from Eq. (10), it can be turned into the symmetric positive definite dual interface problem:

$$\left(F_{RR} + F_{RC} K_{CC}^{*\,-1} F_{RC}^T\right)\lambda = d_R - F_{RC} K_{CC}^{*\,-1} f_C^* \tag{16}$$

which will be solved using a preconditioned conjugate gradient algorithm. The FETI-DP method can be broken into four steps.

1. **Interface Residual.** The interface residual is calculated from two sources: one that is local to the subdomain (Eq. (17)) and one that comes from the corner nodes (Eq. (18)). The initial value of λ^s is zero.

$$r_1 = \sum_{S=1}^{N} B_R^s K_{RR}^{s\,-1}\left(f_R^s - B_R^{s\,T}\lambda^s\right) \tag{17}$$

$$r_2 = -\sum_{S=1}^{N} B_R^s K_{RR}^{s\,-1} K_{RC}^s u_C^s \tag{18}$$

   To calculate r_2, the global coarse problem solution (u_C) is obtained with the following equation.

$$K_{CC}^* u_C = F_{RC}^T \lambda + f_C^* \tag{19}$$

   The solution of the coarse problem is a serial step, so minimizing the size will improve scalability. During the initial iteration process, the coarse problem is gathered on each processor and factorized. Therefore, during the ensuing iterations, the coarse problem is only a right-hand-side solve.

2. **Interface Preconditioning.** Each subdomain will calculate its own preconditioner for iterations. This paper uses the Dirichlet preconditioner

$$M^{-1\,s} = K_{EE}^s - K_{EI}^s K_{II}^{-1\,s} K_{IE}^s. \tag{20}$$

   The preconditioned residual is given by:

$$z = \sum_{i}^{N} M^{-1\,s} r. \tag{21}$$

   Both the preconditioner and preconditioned residual can be assembled in a parallel manner, as only subdomain variables are needed in addition to the residual r from the previous step.
3. **Matrix–vector Multiplication.** This step is nearly identical to the residual calculation, with Eqs. 17 and 18 becoming Eqs. 22 and 23.

$$Fp_1 = \sum_{S=1}^{N} B_R^s K_{RR}^{s\,-1}\left(B_R^{s\,T} v^s\right) \tag{22}$$

$$Fp_2 = -\sum_{S=1}^{N} B_R^s K_{RR}^{s\,-1} K_{RC}^s u_C^s \tag{23}$$

   where v^s is the subdomain restriction of any vector to be multiplied, and Fp is the product of the matrix vector multiplication.
4. **Krylov Iterations.** Here the Conjugate Gradient method is used. Steps 2 and 3 are repeated until convergence. After convergence, the subdomain nodes can be recovered using Eq. (24).

$$K_{RR}^s u_R^s = f_R^s - B_R^{s\,T}\lambda^s - K_{RC}^s u_C^s \tag{24}$$

To perform the above operations, special data structures must be created by the partitioner. These are of 4 types.

1. **Subdomain connectivity.** The partitioner must divide a mesh and reorder the element connectivity for each subdomain. The nodes must be grouped into internal and interface nodes. When partitioning the mesh all joints must be kept internal to a subdomain. A joint can have any number of DOF up to 6, and many are often free in rotors which can make K_{RR}^s non-invertible if the joint is on the interface. Having the joint internal to the subdomain avoids this problem, and includes it in the preconditioner M^{-1} which is beneficial for convergence.
2. **Interface Node Types.** The interface nodes must be grouped into distinct types: face, edge, and corner nodes. The corner nodes must be carefully selected to ensure that ensure K_{RR} is invertible and that they form a coarse approximation of the full mesh.
3. **Subdomain Boolean Matrices.** The Boolean matrix B_R must be created for each subdomain to pick the solution from a subdomain and assign it the proper sign.
4. **Global Boolean Matrices.** The Boolean matrix B_C must be created for each subdomain to pick the local corner solution from the global corner problem.

3.2. Skyline storage

The analysis of large-scale problems is limited by the size of the matrix K. Skyline storage [16] is a common solution. Fig. 5 shows the structure of the K matrix for the TRAM blade. Only the entries below the skyline are ever filled. The two largest subdomain matrices introduced in the previous section, K_{RR} and K_{II}, as well as the global coarse problem K_{CC}^*, are stored as skylines to reduce memory and computational requirements. Several reordering schemes are included in the partitioner to find and utilize the smallest subdomain skylines.

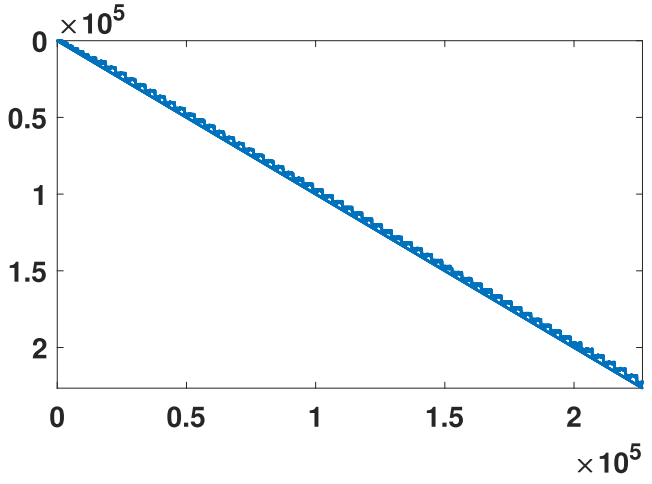


Fig. 5. The stiffness matrix for the TRAM blade. Only the terms below the skyline are stored.

4. Mesh partitioner

The main contribution of this work is the specialized mesh partitioner, which can breakup any generic multibody system of 3D meshes into subdomains for parallel analysis. It contains special features to improve the partition quality for helicopter rotors.

Before the partitioner is presented, the partitioner requirements are listed to guide the remainder of the section. Next, the partitioning process is broken into four steps and each step is described in detail. The special features for helicopter rotors are highlighted, and the partitioner is demonstrated on the NASA TRAM proprotor. Despite all of the features to ensure a quality partition, the total computational time for the partitioner is approximately 5 min for the TRAM blade and hub model.

4.1. Mesh partitioner requirements

The requirements for the mesh partitioner can be split into two categories: general and algorithmic. The general requirements are to:

1. partition unstructured, 27-node brick meshes into non-overlapping subdomains,
2. sort subdomain nodes into internal and interface nodes, and
3. maintain load balance.

The algorithmic requirements are to:

1. keep multibody joints internal to a subdomain,
2. separate interface nodes into face, edge, and corner nodes, while choosing corner nodes to ensure subdomain invertibility and provide a coarse approximation to the full mesh, and
3. create subdomain Boolean structures (B_R and B_C) for each subdomain

The requirement that all joints must be internal to a subdomain can conflict with maintaining load balance. In this case, the joint requirement overrides the need for load balance.

4.2. Physical partitioning

The first step in the partitioning process is to physically partition the structure. Two different strategies are used to physically partition the structure: nominal and cross-sectional. The former is generic to any structure, while the latter is designed to take advantage of the geometry of a rotor blade.

4.2.1. Nominal strategy

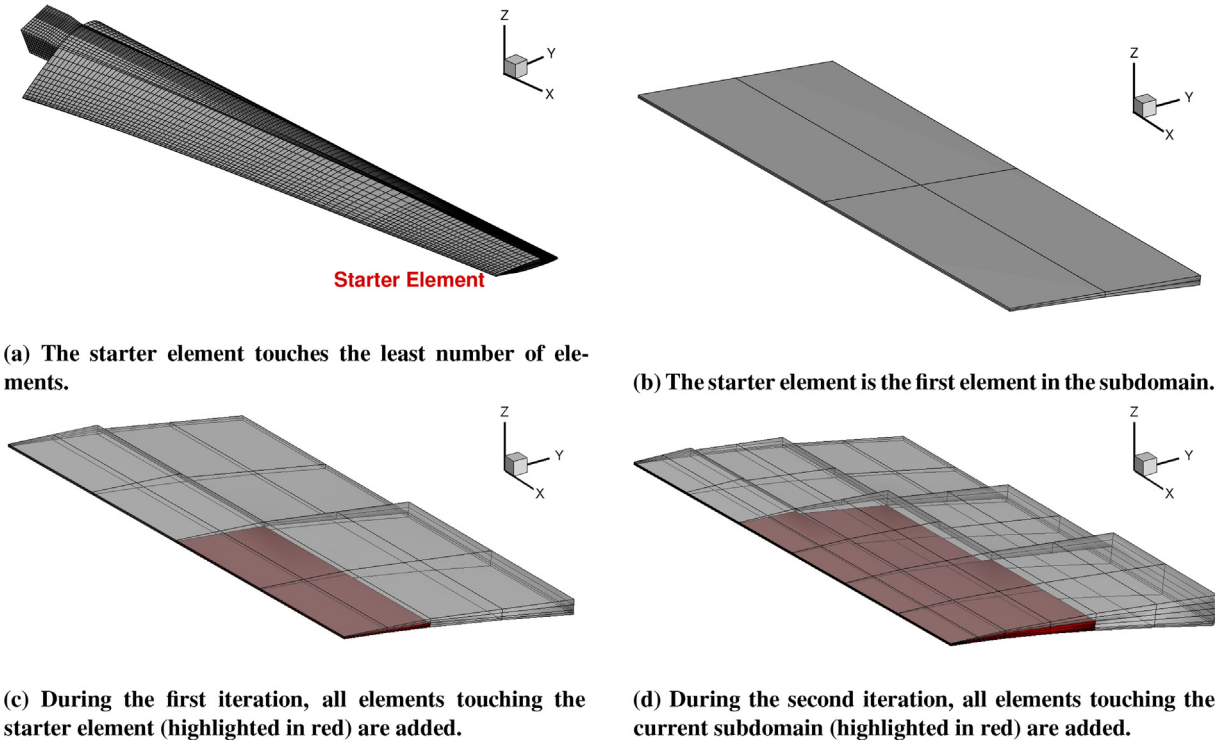
The nominal partitioning strategy only uses element connectivity to physically divide the structure without any geometric bias. The procedure is presented in Algorithm 1 and is explained below. For simplicity, the procedure will first be explained without joints, before adding them back at the end. An illustration is given in Figs. 6d for the blade mesh of the TRAM.

Algorithm 1. Nominal Subdomain Construction

```
1: Create elem_conn
2: elem_sort(all elements) = 0
3: Find start_elem (element with least numbers of
   connections) using elem_conn
4: for {Each subdomain S} do
5:    sub_elem = start_elem
6:    look_list = elem_conn(start_elem)
7:    while Have less elements than desired do
8:       for {Go through each element in look_list} do
9:          if {elem_sort == 0 (element unsorted)} then
10:            Add element to sub_elem
11:            Set elem_sort = S for this element
12:            if {New element connected to joint} then
13:               Add all elements connected to that joint to
   sub_elem
14:               elem_sort = S for these elements
15:            end if
16:            if {Subdomain reached desired number of
   elements} then 17:              Break out of for loop
18:            end if
19:         end if
20:      end for
21:      Rebuild look_list using newly added elements
22:   end while
23:   Save sub_elem to global data structure
24:   if {Not last subdomain} then
25:      Find start_elem for next subdomain.
26:      Find an unsorted element touching last subdomain
27:   end if
28: end for
```

Before the partitioning process begins, two global data structures are created: *elem_conn* and *elem_sort*. *elem_conn* contains the element connectivity and has one entry per element, where entry N contains a vector of all elements that touch element N. The vector *elem_sort* is used to record if and where each element is sorted and has one entry per element. For example, if *elem_sort*(N) = 0, then the element N is unsorted, but if *elem_sort*(N) = C, than the element N belongs to subdomain C. Next, the partitioner identifies the element that touches the least number of other elements, known as the starter element. The starter element will be the first element of the first subdomain. For rotor problems, the starter element is usually located at the blade tip, Fig. 6a, with a close up view shown separately in Fig. 6b.

For a single subdomain, two data structures are created on lines 5 and 6: *sub_elem* and *look_list*. The vector *sub_elem* contains all elements belonging to that single subdomain and initially only contains the starter element. The vector *look_list* contains all elements that touch any element within that subdomain and will be used to find new elements to add. Initially, *look_list* only contains the elements that touch the starter element. Next, the subdo-



(a) The starter element touches the least number of elements.

(b) The starter element is the first element in the subdomain.

(c) During the first iteration, all elements touching the starter element (highlighted in red) are added.

(d) During the second iteration, all elements touching the current subdomain (highlighted in red) are added.

Fig. 6. Subdomain construction using the nominal partitioning strategy.

main construction begins. The subdomain is constructed iteratively, with each iteration being one cycle through the while loop from line 7. During each iteration, all unsorted elements in *look_list* are added to the subdomain on line 10. For the first iteration of the first subdomain, no other elements are sorted, so all elements from *look_list* will be added. This is shown in Fig. 6c, where the starter element is highlighted in red, and the gray translucent elements are those added during the first iteration.

As each element is added, two conditions are checked. The first has to do with joints so is ignored for now. The second check is on lines 16–18, which determines if the subdomain has reached the desired number of elements. If so, the subdomain stops looking for new elements and breaks out of the while loop from line 7. However, if all elements from *look_list* have been checked and more elements are still needed, *look_list* will be rebuilt on line 21 using the newly added elements. This entails going through each element added in the last iteration and adding all elements that touch that element to *look_list*. The partitioner will then repeat the process, going through *look_list* and adding only the unsorted elements. Fig. 6d shows the subdomain after this second iteration of adding elements. In this Figure, the subdomain after the first iteration is highlighted in red, and the translucent elements are those added during the second iteration. This process is repeated until the current subdomain has enough elements, at which point the while loop at line 7 is broken. Next, *sub_elem* is saved to a global data structure. If there is another subdomain to be constructed a new starter element must be identified. This is done by selecting an unsorted element that touches the recently constructed subdomain (line 24–27). Next, a new *sub_elem* and *look_list* are created based on this new starter element and the process is repeated.

When considering joints, there is only one change in the procedure illustrated above. When an element is added to *sub_elem* in line 10, the partitioner checks if it is a joint element. If so, all elements connected to that joint are added during the same iteration to ensure that a joint remains internal to a subdomain.

During the partitioning process, the subdomains are divided into equal sizes based on the number of elements as an initial approximation to load balance. Due to the presence of joints, equating the number of elements across subdomains may not be possible. However, this is of little consequence, as it will be shown that equating the number of elements is not true load balance, and a refined load balancing scheme will be used later in the partitioning process.

Fig. 7 shows the TRAM model partitioned into six subdomains using the nominal partitioning strategy. The subdomains grow in every direction each iteration, resulting in irregular shapes. There is nothing fundamentally wrong with this; solving this partition will still yield the correct answer. However, the irregular partitioning will increase the complexity of the interface problem, resulting in a high computational time.

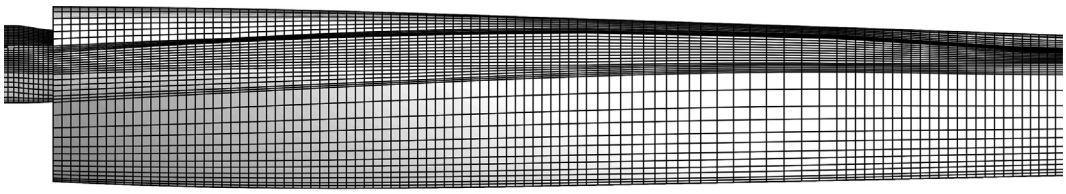
4.2.2. Cross-sectional strategy for rotor blades

Due to the nature of their construction, most rotor meshes can be structured in the radial direction, while having unstructured cross sections, as shown in Fig. 8. The cross-sectional partitioning strategy utilizes this structure.

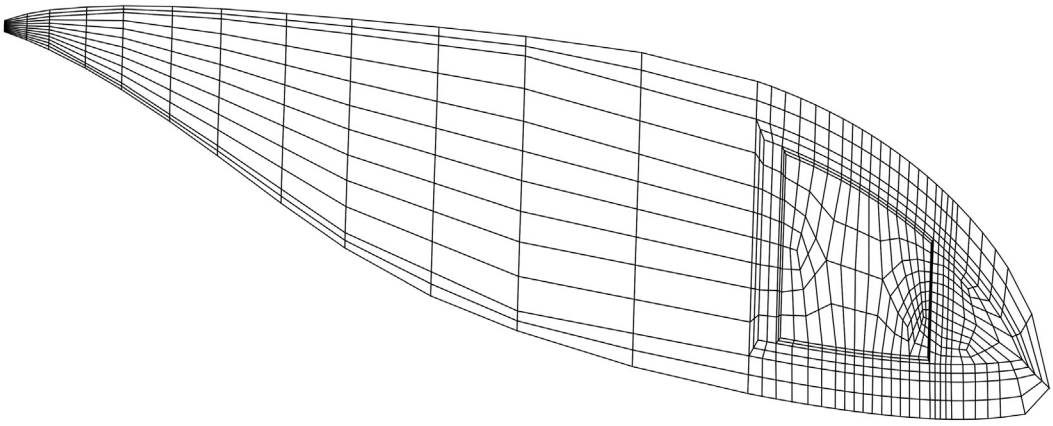
Instead of building a subdomain from a starter element with no preferred direction, the subdomain is constructed section-by-section. Before partitioning begins, the partitioner sorts the elements into cross-sections based on the element coordinates, with the only user input being the radial direction. The only change made to the nominal subdomain construction (Algorithm 1) is that the variable *look_list* is assembled with a preference for elements in the current cross-section. Only once the cross-section is full does the partitioner move onto the next one. Figs. 9d show the first two subdomain construction iterations for the same example as the previous section.

While this approach is ideal for axial components like blades, it is not suitable for non-axial components, such as hub parts. For multibody structures, the partitioner uses a hybrid approach, with

Fig. 7. TRAM Model partitioned into 6 subdomains using the nominal partitioning approach.



(a) Top view of TRAM blade.



(b) TRAM cross section (distorted for publication clearance).

Fig. 8. TRAM blade and a cross section. Cross sections are unstructured, and change from section to section with internal geometry.

the cross-sectional strategy adopted for the blade and the nominal strategy used for the hub components. Fig. 10 shows the TRAM model partitioned into six subdomains using the hybrid strategy. Note that the partition is more regular.

4.3. Subdomain reordering

The second step in the partitioning process is reorder the subdomain nodes to produce a lean skyline. The computational time for each subdomain is heavily influenced by the band of the skylines, with a narrow band significantly reducing the computational time. There are two skylines per subdomain – one each for K_{RR} and K_{II}. Each subdomain is reordered independently of the others.

The partitioner uses two reordering schemes: connectivity-based and geometry-based. For each subdomain, both schemes are applied, creating multiple copies of the matrices with different local ordering. A performance metric is calculated for each copy, based on the number of floating-point operations (see next section), and the copy with the lowest metric is used for that subdomain. Using both schemes for each subdomain allows for each subdomain to be optimized individually.

In the connectivity-based scheme the subdomain will be rebuilt N times, using N different starter elements, where N is a user input. The use of various starter elements allows for the true minimum metric to be found despite the subdomain geometry. These N performance metrics are compared for the single subdomain and the copy associated with the smallest metric is chosen. Note that the composition of an individual subdomain does not change, rather just the local subdomain numbering. For the geometric reordering scheme, the nodes and elements are reordered along the X, Y, and Z axes. The performance metric is checked for all six possible geometric ordering cases. The effect of subdomain reordering on solver performance was previously studied by the authors, and it was observed that the computational load decreases for all subdomains, providing a reduction in computational time [17].

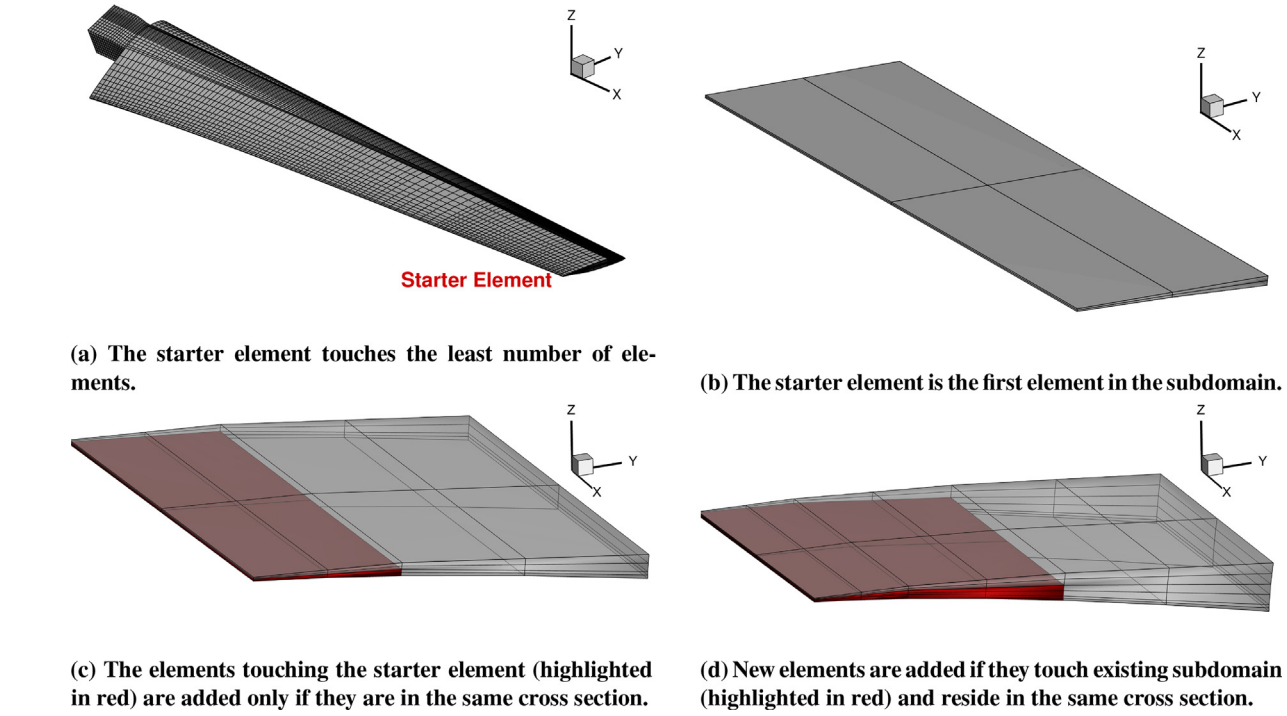
4.4. Load balancing

To achieve practical solver scalability, equal computational workload for all processors, or load balance, is required. A load balancing scheme was implemented based on the number of floating-point operations. The first step to setup this scheme was to determine the most computationally expensive parts of the algorithm. It was found that the following two steps consistently took most of the computational time.

1. Calculating the preconditioner.

$$M^{-1^S} = K^s_{EE} - K^s_{EI} K^{-1^S}_{II} K^s_{IE} \tag{25}$$

In this process, factorizing the matrix K_{II}, the repeated right-hand solves of K_{II} with columns of K_{IE}, and the matrix multipli-



(a) The starter element touches the least number of elements.

(b) The starter element is the first element in the subdomain.

(c) The elements touching the starter element (highlighted in red) are added only if they are in the same cross section.

(d) New elements are added if they touch existing subdomain (highlighted in red) and reside in the same cross section.

Fig. 9. Subdomain construction using the cross-sectional partitioning strategy.

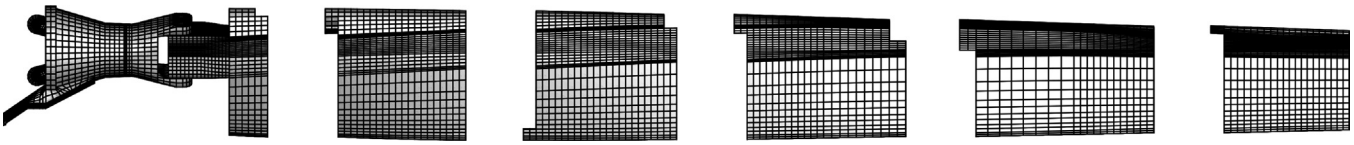


Fig. 10. TRAM Model Partitioned into 6 subdomains using the hybrid partitioning strategy.

cation $K_{EI}K_{II}^{-1}$ are the major operations. Recall that skyline storage is used for the matrices, complicating these operations.

2. Calculating the residual and the coarse problem K_{CC}^{*}.

$$K_{CC}^{*} = \sum_{S=1}^{N} B_C^{s\,T}\left(K_{CC}^{s} - K_{CR}^{s}K_{RR}^{s\,-1}K_{RC}^{s}\right)B_C^{s} \tag{26}$$

In this step, factorizing K_{RR}, repeated right-hand solves of K_{RR}, and the matrix multiplication $K_{CR}K_{RR}^{-1}$ are the major operations.

In addition to these steps, the only other notable amount of time was the time required for the CG iterations, which varied based on the number of subdomains and other partitioner features, namely corner node selection and partitioning strategy. However, it was observed that load balancing the above steps also resulted in adequate load balancing of the iterations, as each iteration involves many of the same operations as the steps above, including the repeated right-hand solves of K_{RR} (Eqs. 18 and 22).

The load balancing scheme is used after the structure has been partitioned. This process involves moving elements between subdomains, changing their composition. However, the computational workload is dependent on the local node numbering (shape of skyline), and therefore subdomain reordering after load balancing might change the computational workload for each subdomain, destroying the load balance created. Rather, each time an element is moved during the load balancing process, the two affected subdomains are reordered, to ensure that load balance is achieved with the minimal subdomain workload. The affected subdomains must be reordered each time an element is moved due to the highly nonlinear relationship between a subdomain's computational workload and its number of elements, to ensure that the load balancing process uses an accurate workload for each subdomain.

The psuedo-code for the refined load balancing scheme is presented in Algorithm 2. For a single load balancing iteration, the first step is to calculate the performance metric (*P_metric*) for each subdomain, equal to the number of floating-point operations for the steps presented in the previous section. During this calculation, the two reordering schemes introduced previously are used to ensure that the smallest performance metric is used for each subdomain. Then each subdomain is examined, beginning with the smallest subdomain (based on *P_metric*). For subdomain *s*, two conditions are checked; first, if there any subdomains with a larger metric, and second, do any of those subdomains touch subdomain *s*. Subdomains that satisfy both conditions are classified as candidate subdomains and added to *sub_cand*. If no subdomains satisfy both conditions (*sub_cand* is empty), the focus moves on to the next smallest subdomain.

Algorithm 2. Refined load balancing

```
1: for {Load balancing iterations} do
2:    P_metric = f(sub) ▷Calculate performance metric with current subdomains
3:    for {Each subdomain s based on P_metric} do
4:       sub_cand = [] ▷Initialize list of subdomains that can help
5:       move = 1
6:       while move = 1 do
7:          for {All subdomains except subdomain i} do
8:             Find subdomains that touch subdomain s AND have smaller P_metric – add to sub_cand
9:          end for
10:         if sub_cand not empty then
11:            for {Go through sub_cand} do
12:               Find element that touches both subdomains
13:               if {Element not a joint element} then
14:                  Move element to subdomain s
15:                  Break out of current for loop, go to line 21
16:               end if
17:            end for
18:         else
19:            Set move = 0
20:         end if
21:         if {Successfully moved an element} then
22:            P_metric = f(sub) ▷ Recalculate metric with updated subdomains
23:         else
24:            Set move = 0
25:         end if
26:      end while
27:   end for
28: end for
```

If there are candidate subdomains, then the partitioner steps through each one on line 11, checking the elements that connect the candidate subdomain to subdomain s. If the connecting element is not a joint element, it is moved to subdomain s, and the partitioner stops looking through the candidate subdomains and moves on to line 21. After passing through all subdomain candidates, if no element was moved, the partitioner exits the while loop on line 6 and moves on to the next smallest subdomain. However, if an element was moved, then the performance metric is recalculated for the two subdomains that were altered, and the process repeats for subdomain s until it has an equal performance metric to surrounding subdomains. This process is repeated for every subdomain for a single load balancing iteration. Typically, using $2N_S$ iterations, where N_S is the number of subdomains, will produce the optimal load balance for the system.

The floating-point operation-based load balance was observed to be essential to obtain meaningful scalability [17]. However, the one thing that can disrupt load balance is the presence of multibody joints. A joint cannot be partitioned – therefore a joint subdomain can only be as small as the joint itself. As the number of subdomains is increased, eventually the joint subdomain(s) will become significantly larger than the others, causing a load imbalance that can limit the solver scalability. However, as demonstrated by the authors [18], the partitioner can predict the number of subdomains when the load imbalance will occur before the analysis, allowing the user to avoid over-partitioning and run close to optimal parallel efficiency.

4.5. Creation of interface data structures

After the mesh has been partitioned and nodes reordered, the interface must be organized. The first step is to identify the corner nodes for each subdomain, which can have a large effect on convergence behavior. The number of corner nodes per subdomain is a user input. The complex geometry associated with realistic problems makes corner node selection difficult, and two approaches are introduced to address this issue. After the corner nodes are finalized, the Boolean and communication maps are created for each subdomain.

4.5.1. Strict corner node problem

A strict definition of corner nodes – nodes shared by at least 3 subdomains or located at the geometric corners, works well for an academic structured mesh, such as that shown in Fig. 12b. However, for an unstructured mesh, the strict definition can lead to the corner nodes being a poor approximation of the full problem. Fig. 11a shows the TRAM mesh split into 4 subdomains with 8 corner nodes per subdomain using the strict definition. Due the complexity of the partitioned geometry, the corner nodes for the first two subdomains are concentrated in a small region. The requirement that the coarse problem should represent a coarse finite element description of the original problem cannot be consistently met with strict corner nodes.

4.5.2. Minimal corner node problem

To improve the coarse problem, a robust selection strategy is implemented. It works within the confines of the user specified number of corner nodes per subdomain, but improves the quality of the coarse problem. The strategy starts by identifying many corner candidates from existing interface nodes. To be a corner candidate, a node must satisfy the following conditions:

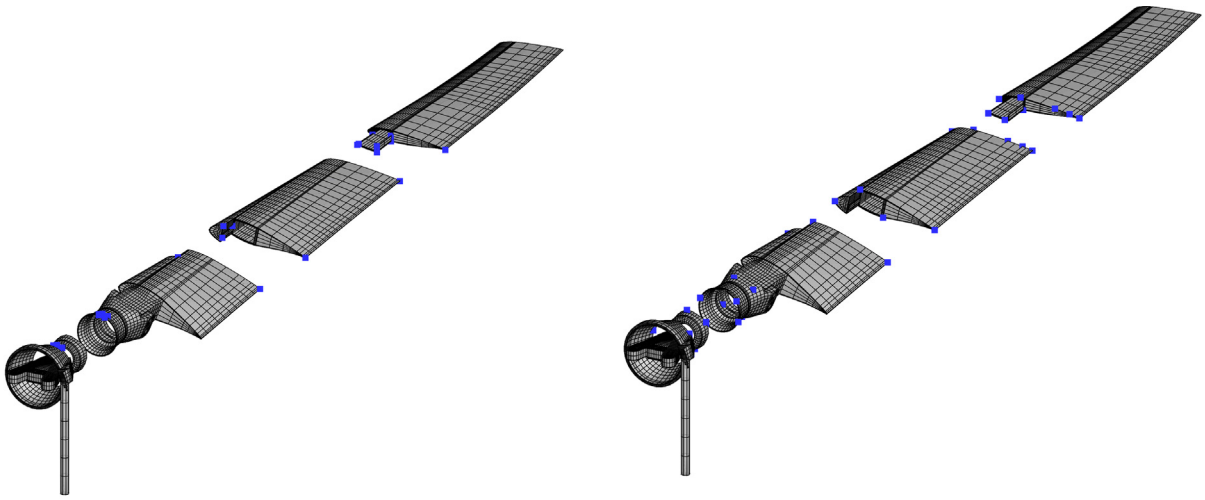
1. it must either touch at least three subdomains,
2. or it must touch at least two subdomains and be shared by two or less elements within each individual subdomain.

This initial selection adds in more than just geometric corners, providing plenty of candidates. Next, the partitioner down selects the candidates by removing nodes, so that the final corner nodes provide a proper coarse representation of the full problem. The down selection is done subdomain-by-subdomain. The first step is to identify all possible pairs of corner candidates. For example, if there are 4 corner candidates then there are 6 possible pairs (1–2, 1–3, 1–4, 2–3, 2–4, and 3–4). For each pair, the distance between the nodes is calculated, and one node is removed from the closest pair. Then the pairs and distances are recalculated, and the process is repeated until the subdomain has the user defined number of corner nodes. Fig. 11b shows 4 subdomains with the same number of corner nodes per subdomain as Fig. 11a; however the new corner nodes, if connected, create a coarse mesh that well represents the geometry.

The minimal corner problem uses the robust selection strategy to choose corner nodes. It gets its name from the fact that it provides the best coarse approximation of the full problem with the minimum (no more than the user defined) number of corner nodes. For a simple problem, the minimal corner problem will collapse to selecting the geometric corners as seen in Fig. 12a. The size of the coarse problem is kept small by allowing edge nodes that touch three or more subdomains to be treated as interface nodes with extra dual variables, similar to the procedure used in Ref. [13].

4.5.3. Enriched corner node problem

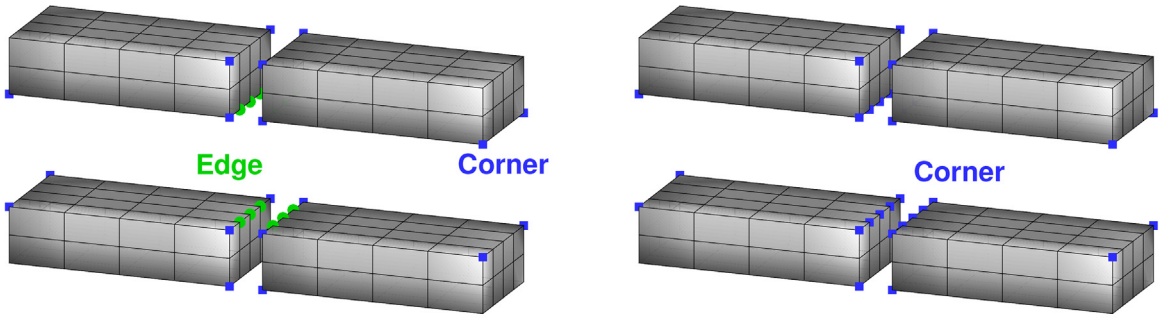
The enriched corner node problem was developed to prevent the increase in the number of interface iterations associated with



(a) The strict corner node definition can result in a poor approximation of the full problem.

(b) The robust corner selection strategy spreads out the corner nodes (blue dots) providing a better coarse approximation to the full problem.

Fig. 11. The TRAM blade partitioned into 4 subdomains – corner nodes chosen using two different approaches.



(a) For an academic structured mesh, the minimal coarse problem will only consist of geometric corners.

(b) For an academic structured mesh, the strict and enriched corner problems will be identical.

Fig. 12. The minimal and enriched coarse problems are presented.

edge nodes [13]. The enriched corner problem keeps the corner nodes used in the minimal coarse problem, and adds all edge nodes that touch three or more subdomains as corner nodes, as shown in Fig. 12b. This will increase the size of the corner problem, but is made viable by the use of skyline storage for the coarse problem, significantly reducing the communication and solution times.

4.5.4. Corner node summary

To summarize, there are three corner node problems that will be used in this paper. The effect of the three corner node problems on parallel solver performance will be studied later in the paper.

1. **Strict** - Only geometric vertices and nodes shared by three or more subdomains.
2. **Minimal** - Some subset of interface nodes selected using the robust corner node selection strategy to create best approximation of coarse problem.
3. **Enriched** - Exact set of corner nodes from the minimal corner problem plus all nodes shared by three or more subdomains.

4.5.5. Boolean and communication maps

The Boolean maps, B_C^s and B_R^s from Eqs. (4) and (6), must be created after the corner nodes are finalized. Processor-to-processor communication requires maps of what is communicated and their destination, because no subdomain has access to the full problem. Therefore, both are created as the last step of the mesh partitioner.

The coarse problem requires a global communication step — each processor must send its contribution and receive the global solution. The Boolean map B_C from Eq. (4) is created for each subdomain to extract the local solution from the global corner problem. The Boolean map B_R^s is used to enforce continuity across neighboring subdomains. For a face node that only touches 2 subdomains, the requirement is that the sum of the Boolean operators must equal 0 — the subdomain operators must alternate between + 1 and −1. For edge nodes, multiple redundant constraints are used to improve performance, so each edge node from a single subdomain needs N Boolean operators, corresponding to the N subdomains it touches. To enforce continuity across the interface, only local communication is required – each subdomain need only exchange data with its neighbors. These maps are present in the placement of the Boolean operators within the map B_R^s.

5. Implementation

The X3D solver [7] was re-designed to implement the FETI-DP algorithm. It covered all solution procedures: static, dynamic,

eigenvalues, hover, and forward flight. Each subdomain was solved on a separate processor, with Message Passing Interface (MPI) used for communication. The results shown in later sections were obtained with this new version of X3D.

The platform used was Deepthought2, the University of Maryland High-Performance Computing (HPC) cluster. This cluster contains 480 nodes, each with 20 Intel Ivy Bridge E5-2680v2 processors running at 2.8 GHz. Each processor has a separate L1 (64 KB) and L2 (256 KB) cache, a shared L3 cache (25 MB), and a total shared memory of 128 GB. For the work presented in this paper, a maximum of 2048 processors (103 nodes) were used. The input and output portion of the code (file I/O) are left out of timings.

6. Validation

Before we delve into the performance of the partitioner and parallel solver, we take a brief detour to validate them with experimental test data from Ref. [19]. During the test, the blade collectives were set and power P and thrust T measured.

The parallel solver was executed on 11 different partitions ranging from 2–76 subdomains and identical results confirmed. The aerodynamic model used a state-of-the-art lifting line model with C81 airfoil decks. Airfoil tables developed for the NASA large civil tiltrotor project are used [20]. Details of the aerodynamic model can be found in Ref. [18]. The nondimensional power C_P/σ, nondimensional thrust C_T/σ, and efficiency metric Figure of Merit FM are used for validation.

$$C_P/\sigma = \frac{P}{\rho A_b V_{TIP}^3} \tag{27}$$

$$C_T/\sigma = \frac{T}{\rho A_b V_{TIP}^2} \tag{28}$$

$$FM = \frac{C_T^{3/2}}{\sqrt{2}C_P} \tag{29}$$

where P is power (W), ρ is density (kg/m^3), A is disk area (πR^2, where R is rotor radius), V_{TIP} is the blade tip velocity, σ is the ratio of blade area to disk area (A_b/A, where A_b is the total blade area), and T is thrust (N).

Fig. 13 and 14 present predicted nondimensional power and FM as a function of nondimensional thrust. The airfoils required a Reynolds number correction and a stall delay model [19]. Overall, the predictions, though not satisfactory, represent the state-of-the-art.

The principal benefit of using 3D structures is the ability to predict the resulting stresses and strains for both the blade and hub components. These stresses and strains are very hard to measure in the rotating frame. The 3D solution provides a direct insight into their nature that is not available otherwise. Fig. 15 shows the axial stress (σ_{11}) for the blade and hub for a high thrust case C_T/σ = 0.125 near maximum figure of merit. The localized stress concentrations are clearly seen. Fig. 16a shows the stresses at the root section. In addition to the axial stress σ_{11}, the other five stress components are also generated by the solver. These stresses, especially in the hub components, will provide unique information not available using current beam-based analysis.

7. Idealized beam

The performance and scalability of the solver is first studied for an idealized test case: a cantilevered beam of 6.6 million DOF with a structured mesh, uniform properties, a single joint only as a boundary condition, and no control inputs. This is done to determine the performance and limitations of the algorithm on an elementary problem. For this structure, it is trivial to load balance

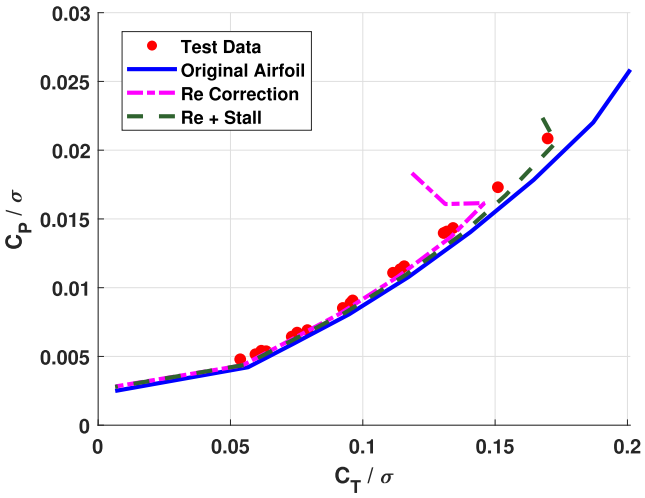


Fig. 13. Power vs. nondimensional thrust.

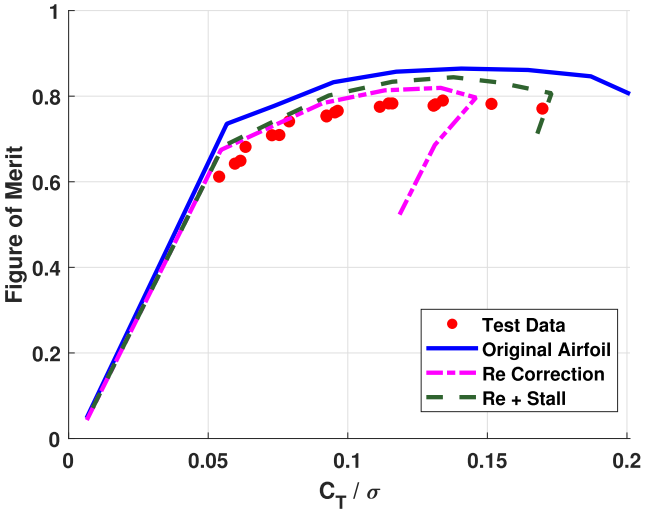


Fig. 14. Figure of merit vs. nondimensional thrust.

and select corner nodes – none of the special features are needed. Two different coarse problems were studied, the minimal (Fig. 17a) and the enriched (Fig. 17b). For this simple problem, the enriched coarse problem matches the strict coarse problem, while the minimal coarse problem uses only geometric vertices. The solver performance is studied for only a single linear matrix solve. The full solution involving aerodynamics consists of repeated iterations of the same solve, so one suffices for algorithmic purposes.

Fig. 18 shows the solution time on a single processor versus the number of subdomains. For this work, a single processor solution still uses a partitioned structure, however the subdomains are solved sequentially on the single processor. It is well known that simply partitioning a structural problem, even without parallel implementation, will have a large reduction in solution time. However, what is noteworthy is that the use of the enriched coarse problem reduces the total computational time compared to the minimal coarse problem, opposite of what was seen by Datta and Johnson [13]. To further investigate this, Tables 2 and 3 provide a detailed break-up of timings. In the Tables, No. Corner refers to the number of corner nodes and Corner Sky refers to the percent of the total coarse matrix that is filled and stored. Sub. Decomp refers to the subdomain factorization. Lastly, Coarse is the time taken to solve the coarse problem and Iterations is the time required for CG iterations.

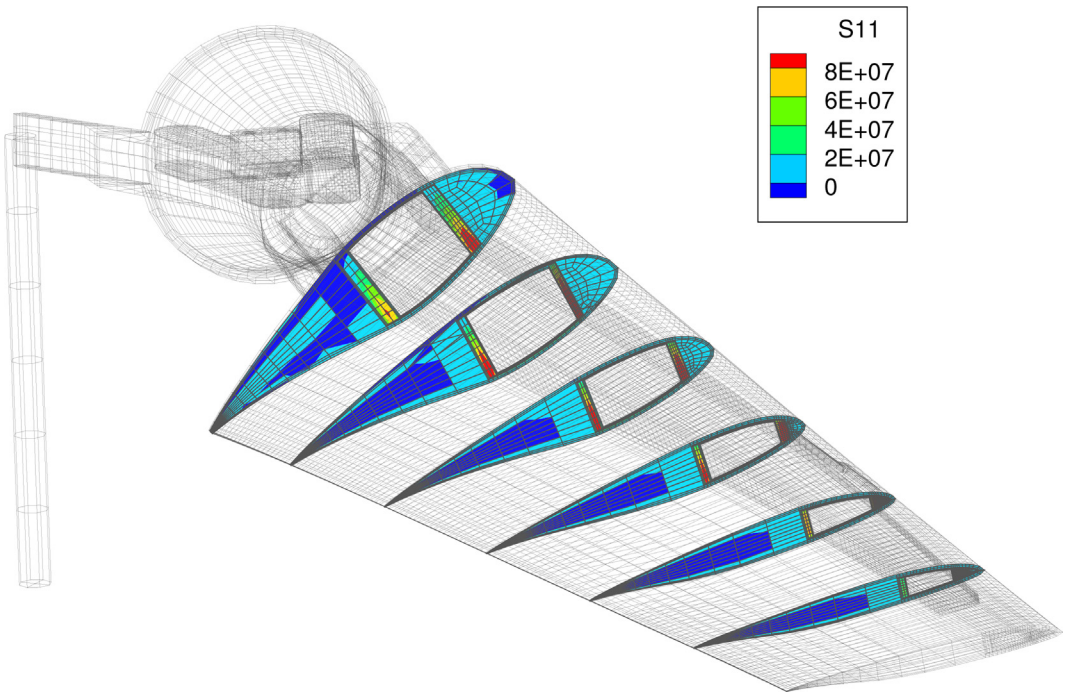
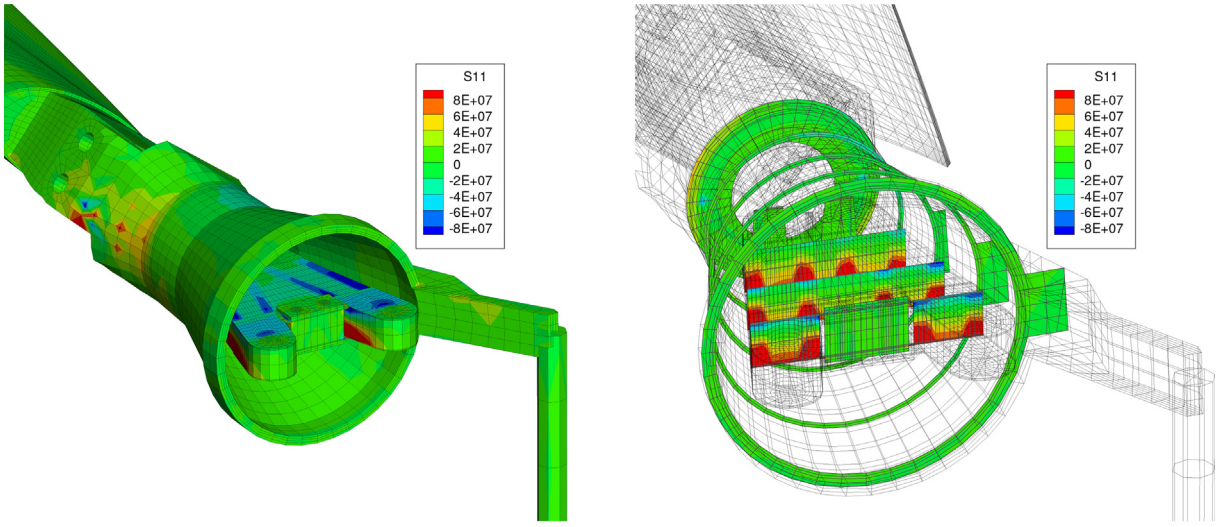


Fig. 15. Axial Stress σ_{11} throughout the blade for high thrust (C_T/σ = 0.125).

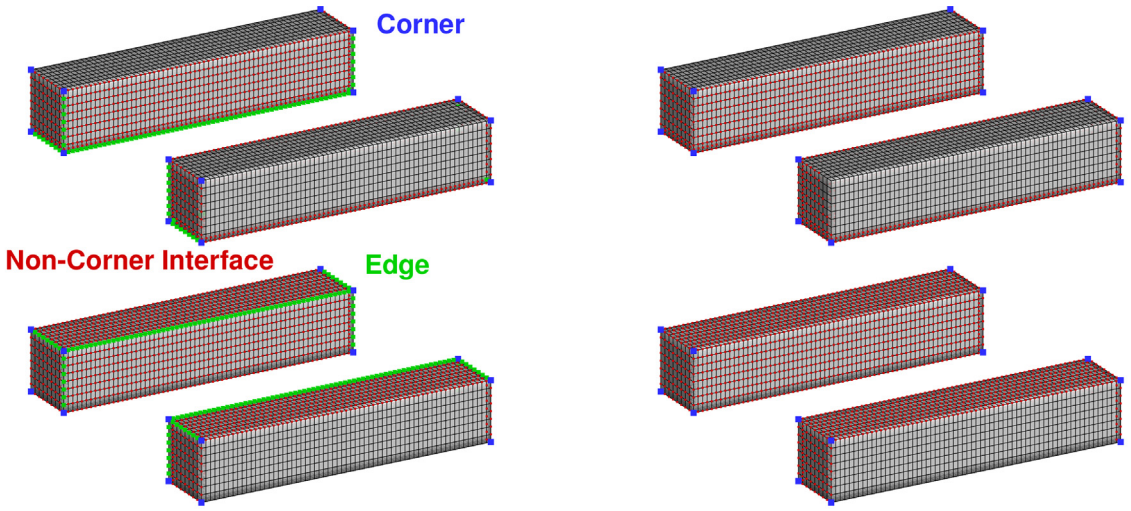


(a) Axial Stress σ_{11} at the root. **(b) Axial Stress σ_{11} at the root cross-sections.**

Fig. 16. Axial stress is shown at the root for high thrust (C_T/σ = 0.125).

An important conclusion drawn is that the use of skyline storage for the coarse problem is crucial for performance. Even for the largest coarse problem (enriched 2048 subdomains) its solution time remains a fraction of the total time. The coarse problem for this case is 110 K DOF, yet the matrix is very sparse, with just over 0.25 percent of the full matrix being stored by the skyline (Fig. 19). This reverses the conclusions made in previous studies [12,13]. The largest difference between the coarse problems is the time spent in iterating the dual variables. The minimal coarse problem requires more iterations to converge, a fact that agrees with previous studies. However, now the increase in time to solve the coarse problem is insignificant when compared to the reduction of time for iterations, so there is a net gain in moving to the larger coarse problem.

Next, the performance in parallel is examined, where every subdomain is solved on a separate processor. The speedup is defined as single processor time divided by parallel time for the same number of subdomains and same coarse problem. This ensures the speedup is the true parallel speedup un-contaminated by the inherent benefits of partitioning. Fig. 20a compares the speedup for the two cases up to 2048 processors. For both, linear scalability is achieved up to 1024 subdomains, but the speedup falls off by 2048 subdomains. A closer look reveals that the minimal coarse problem has a marginally better speedup compared to the enriched coarse problem. This is because the coarse problem is a serial step, so the larger coarse problem is naturally less scalable. But speedup is defined relative to the single processor time; the enriched coarse problem still beats the minimal in wall-clock time (Fig. 20b).



(a) The minimal corner problem uses edge nodes to keep the coarse problem small. Only nodes at the geometric vertices are considered corner nodes.

(b) The enriched corner problem considers all nodes that touch 3 or more subdomains or are located at geometric corners as corner nodes.

Fig. 17. The two coarse problems are presented for the idealized problem.

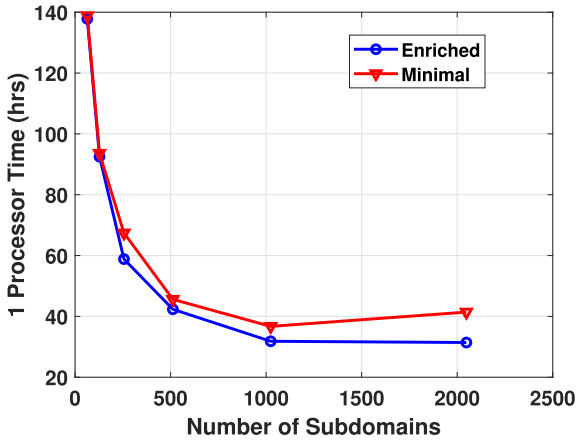


Fig. 18. The computational time on a single processor is shown for the cantilevered beam.

Table 2
Detailed timings comparison on single processor for enriched coarse problem (sec).

No. Sub	No. Corner	Corner Sky	Sub. Factor	Coarse	Iterations	Solver Total
512	10693	0.99	144931	2.0	3531	152404
1024	19397	0.53	99518	3.3	11112	114555
2048	36805	0.27	55273	5.9	53988	113086

Table 3
Detailed timings comparison on single processor for minimal coarse problem (sec).

No. Sub	No. Corner	Corner Sky	Sub. Factor	Coarse	Iterations	Solver Total
512	1153	1.06	147378	5.0e-3	12817	164170
1024	2305	0.53	103122	1.0e-2	25146	132185
2048	4609	0.27	54915	2.0e-2	90412	149095

To gain insights into the eventual barriers to speedup, a detailed breakdown of the 1024 and 2048 processor cases was created and is presented in Fig. 21a and 21b. Both the minimal and enriched cases had similar trends, so only the enriched case is shown. The algorithm was timed separately for each of the four major steps: building the structural matrices, calculating the preconditioner, calculating the residual, and iterating the dual variables until convergence. The speedup for each step was computed, as well as the relative time spent in each step. First, let us note the similar trends between the two cases. The first two steps are achieving linear – and in some cases superlinear – speedup. Superlinear speedup is when the speedup is greater than the number of processors and is due to cache benefits that come with running in parallel. The performance of these two steps is expected, as they involve no communication and should produce linear speedup for a perfectly load-balanced case.

The main difference between the 1024 and 2048 subdomain cases is the large decrease in efficiency during the CG iterations and the increase in relative time spent in this step. The decrease in efficiency is due almost entirely to the communication overhead. The remaining inefficiencies are due to the small sections that cannot be parallelized – only noticeable for high numbers of subdomains, when each iteration takes only a few tenths of a second. For the 1024 processor case, approximately 20 percent of every iteration is communication overhead, with less than 5 percent for serial steps. While this may seem high, the cache savings are higher, as this step has superlinear speedup. However, when moving to the 2048 processor case, nearly 50 percent of each iteration is communication overhead, with an additional 20 percent spent on the serial steps. The cache savings are unable to compensate, resulting in poor parallel efficiency. For the 2048 subdomain case, most of the solver time is spent performing iterations, contributing to the poor overall speedup. This ratio of communication time to computational time is common for high numbers of subdomains and shows that nothing more can be squeezed out at this problem size.

This main conclusion from studying the solver performance on the idealized beam is that using skyline storage practically eliminates the algorithmic barrier of the coarse problem. Next it will be shown that for a realistic problem joints are the true barriers to scalability.

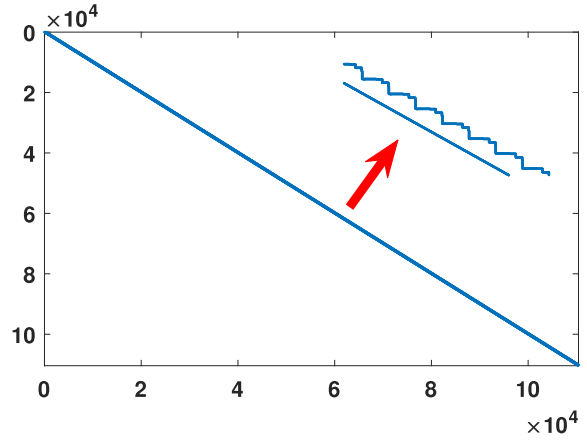


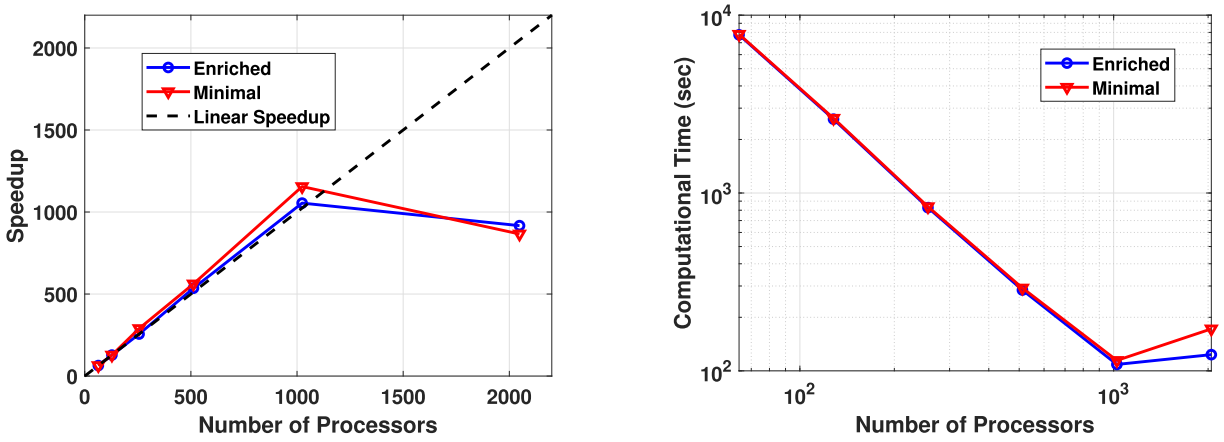
Fig. 19. The coarse problem skyline is shown for the enriched problem with 2048 subdomains.

8. TRAM Blade

This section demonstrates the performance and scalability of the parallel solver on the NASA TRAM blade. Now the mesh is unstructured, and although there is only one finite element part, there is still a joint at the root used to apply the proper boundary condition. The effect of partitioning strategy will be examined, followed by the effect of corner node selection. Finally, the solver scalability limits will again be explored.

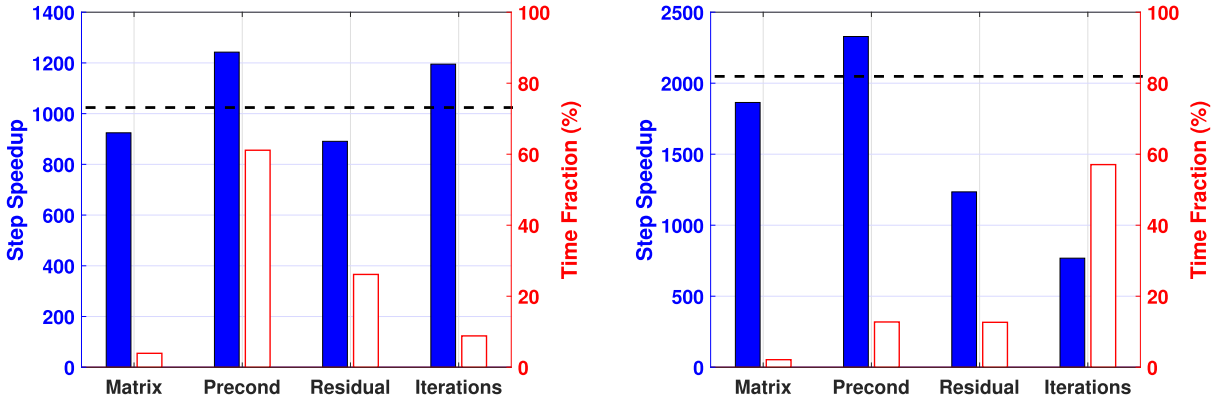
8.1. Effect of partitioning strategy

This section compares the two partitioning strategies – nominal versus cross-sectional. The enriched corner problem is used for all cases with 30 corner nodes per subdomain. Fig. 22a shows cross sectional partitioning can produce more dual variables than nominal, particularly for a large number of subdomains. However, cross-sectional partitioning will produce a much leaner skyline of the interface, as presented in Fig. 22b.



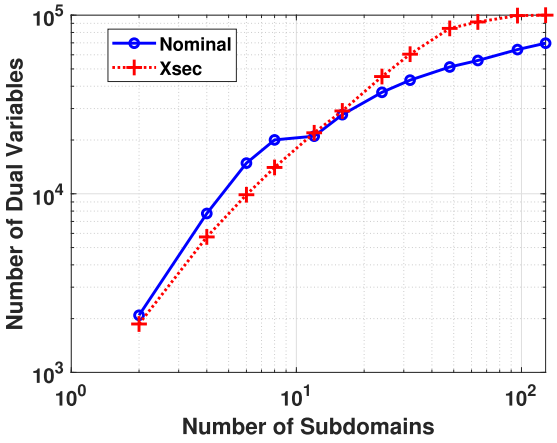
(a) Parallel speedup **(b) Parallel computational time**

Fig. 20. The parallel performance is presented for the idealized cantilevered beam problem.

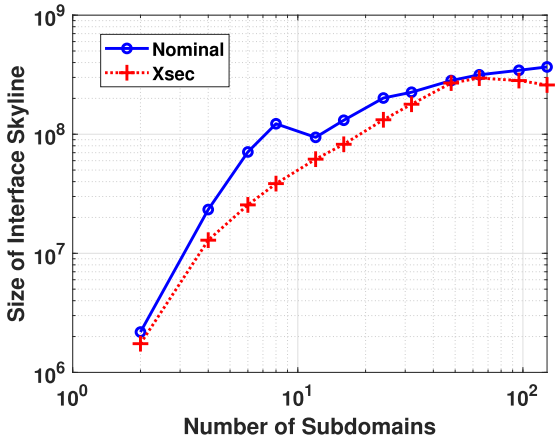


(a) Performance for 1024 processors. **(b) Performance for 2048 processors.**

Fig. 21. The efficiency of each step and the percentage of the total time taken for that step are compared for 1024 and 2048 processors. The horizontal back line represents linear speedup.

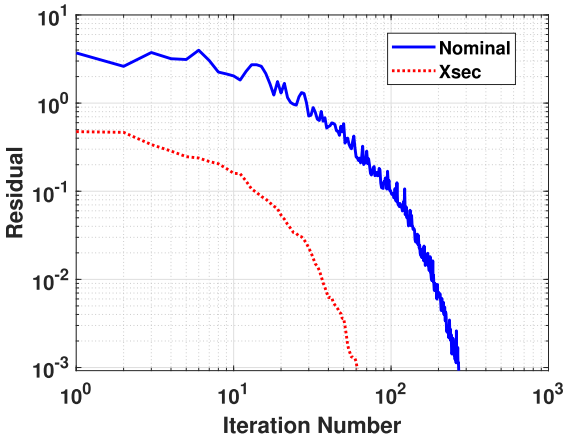


(a) The number of dual variables.

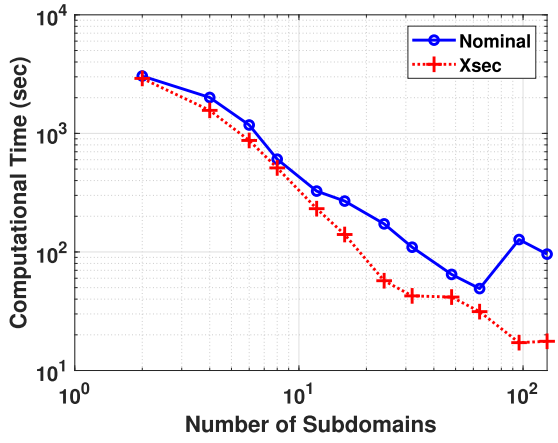


(b) Size of the dual skyline.

Fig. 22. The interface characteristics are compared between the two partitioning strategies.



(a) Convergence behavior



(b) Parallel solver time

Fig. 23. The solver performance is compared between the two partitioning strategies.

Fig. 23a compares the solver convergence for the two partitioning strategies when the blade is split into 12 subdomains. Due to the simpler interface, the cross-sectional case converges much faster than the nominal case. Fig. 23b compares the parallel computational time for the two partitioning strategies. There are two important conclusions. First, cross-sectional partitioning provides a significant reduction in computational time compared to nominal partitioning. Second, the use of nominal partitioning can lead to jumps in computational time such as the one from 64 to 96 subdomains. These jumps are due to an increase in the number of iterations and are directly related to the complexity of the interface problem. Using cross-sectional partitioning will not only improve performance but reduce the inconsistencies in solver performance.

8.2. Effect of corner node selection strategy

This section studies the effect of different corner problems on solver performance. For this section, all partitions are created using cross-sectional partitioning. The number of corner nodes per subdomain is set to 30.

Fig. 25 shows the number of corner nodes for each of the three corner problems: strict, minimal, and enriched. For low numbers of subdomains, the minimal and enriched coarse problems are nearly identical. Recall that both use the robust corner node selection strategy, with the only difference occuring if there are nodes that are shared by three or more subdomains. The strict corner node problem has less corner nodes than the other two for lower numbers of subdomains, as the number of geometric vertices is small for these cases. For higher numbers of subdomains these trends flip. The strict and enriched corner problems increase rapidly as many nodes are shared by three or more subdomains. The minimal problem still increases, but at a much slower rate.

Fig. 26 shows the number of solver iterations. The number of iterations drastically increases for the minimal coarse problem in agreement with previous studies [12,13]. An important observation is that the use of the enriched coarse problem does not provide any significant benefit over the strict definition. This is because the partitioned geometry results in the geometric vertices providing a good approximation to the full problem. This is specific to this problem, and will change when the full blade and hub model is studied later. Finally, note that the use of corner node enrichment provides the lowest number of iterations.

Fig. 27a compares the parallel computational time. The same trends seen for the number of iterations persist here. Fig. 27b shows the parallel speedup for the same cases. The trends are now flipped, with the minimal corner problem producing the max-

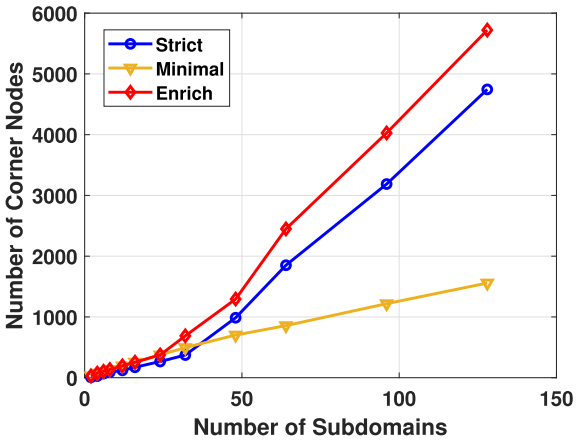


Fig. 24. Convergence behavior for two corner problems with different number of corner nodes.

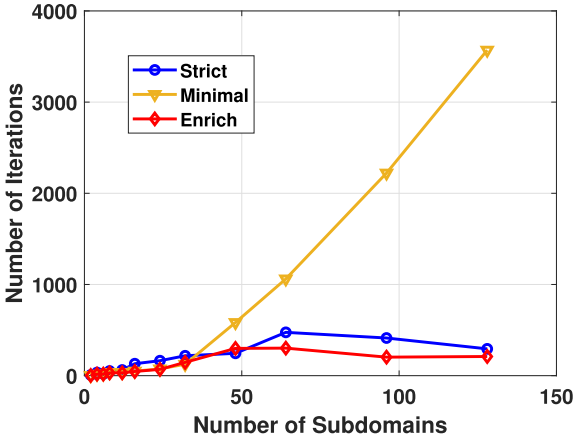


Fig. 25. Number of corner nodes.

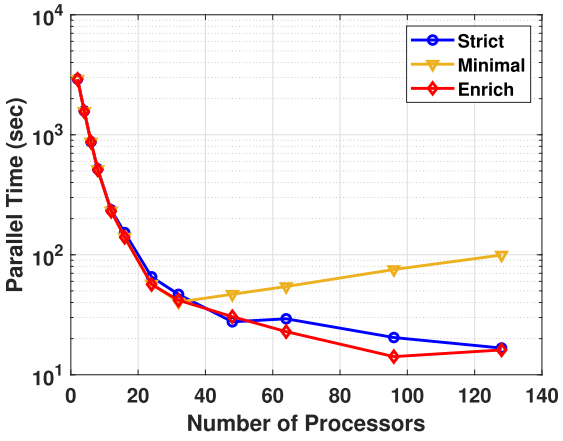
imum speedup. Like the ideal beam up to 1024 subdomains, there is a drop off in speedup that comes with increasing the size of the coarse problem. This comes despite the decrease in actual computational time.

From this section, two major conclusions can be drawn. The first is that admitting edge nodes to minimize the size of the corner problem does not improve throughput, but rather leads to a significant increase in computational time. The second is that the use of the enriched corner problem does not provide any substantial improvements compared to the strict selection.

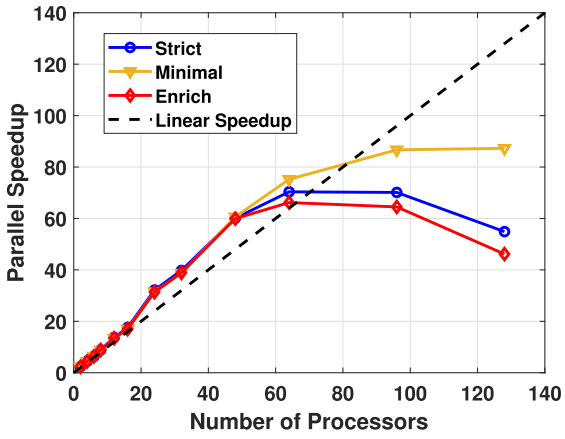
8.3. Limit of scalability

This section explores the limits to solver scalability for the TRAM blade. From Fig. 27b, linear speedup is obtained up to 64 subdomains for all corner node selection strategies. Partitioning the problem further will only decrease the efficiency of the solver. This can be traced back to two underlying causes. To better understand these causes, Fig. 28a shows the detailed speedup for the 128-subdomain case with the enriched corner problem. Like the ideal beam, most of the time is spent performing iterations, and this step has very low speedup. A small part of this is again due to the communication overhead. In this particular case, communication makes up about 8 percent of the time required for the iterations. However, while this causes some decrease in solver efficiency, it is not the main barrier to scalability.

From Fig. 28a, the construction of the subdomain matrices and the calculation of the residual also have poor speedup. This illustrates that these inefficiencies are not just related to communication overhead but are in fact due to poor load balance caused by the root joint. As the structure is partitioned into smaller and smaller subdomains, the root joint eventually makes up an entire subdomain. Since it cannot be split, further partitioning only affects the other subdomains, and causes a load imbalance. Fig. 28b shows the predicted load balance versus the number of subdomains. The predicted load balance is obtained using the floating-point operations-based metric introduced in refined load balancing section. The predicted load balance is the largest metric divided by the smallest metric and should be 1 for a perfectly balanced partition. Note that the predicted load balance is smaller than 1.2 until 96 subdomains, after which it quickly rises. Up until this point, any small load imbalances are overcome by cache benefits, resulting in linear speedup. From 96 subdomains onward, the load imbalance caused by the root joint acts as a barrier to further scalability. The partitioner calculates the predicted load balance before the solver is run, so the user is able to select the highest number of subdomains that provides good load balance. For realistic problems, the presence of joints will almost always become a barrier to scalability long before communication overhead. The conclusion that joints will act as a barrier to scalability for realistic problems will be reinforced in the next section for the full blade and hub model.

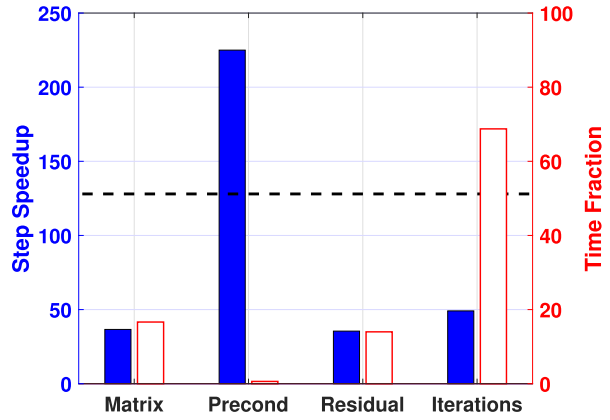


(a) Parallel solver time.

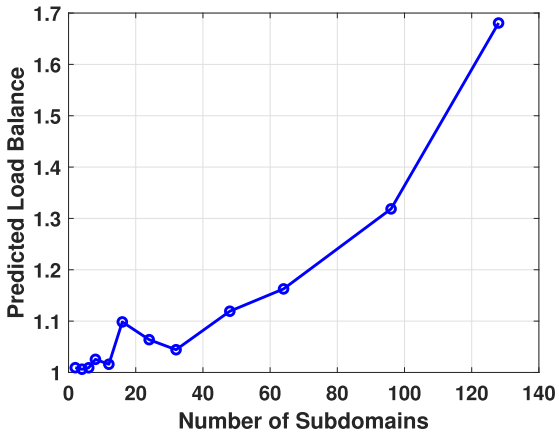


(b) Parallel speedup.

Fig. 26. Number of Iterations required.

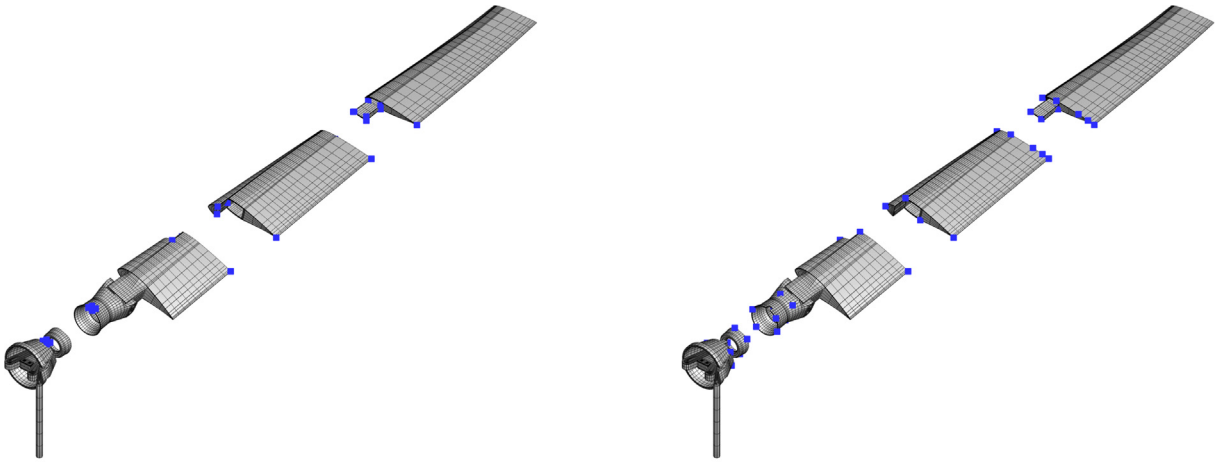


(a) Performance for 128 processors.



(b) Predicted Load Balance from Mesh Partitioner.

Fig. 27. The solver performance is compared between the different corner node selection strategies.



(a) The strict corner selection.

(b) The robust corner selection.

Fig. 28. The limits to scalability are explored for the NASA TRAM Blade.

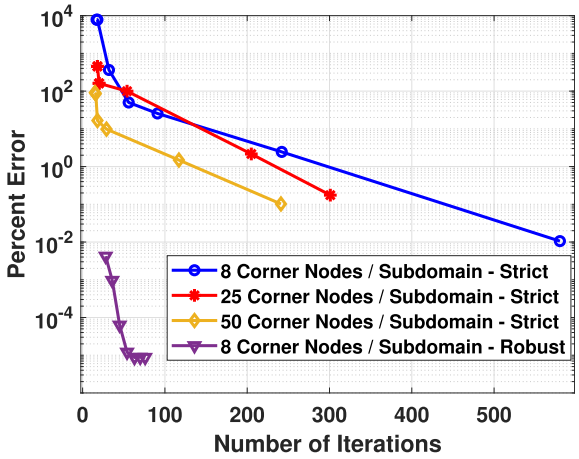


Fig. 29. TRAM partitioned into 4 subdomains with corner nodes chosen using two different strategies.

Table 4
Coarse problem comparison.

Metric	Strict	Robust
No. of Corner Nodes	20	20
Size of Coarse Skyline	1830	1830
Coarse Problem Condition No.	3×10^{12}	2.5×10^{5}
Initial Residual	8×10^{6}	0.25

9. TRAM

This section introduces the full model with all the complexities of an advanced rotor blade and hub – multiple unstructured meshes connected via multibody joints. The previous section already showed cross-sectional partitioning consistently produces the best performance, so that will not be studied. The effect of corner node selection will be re-examined in detail, since that is the feature of greatest concern for a multibody structure.

9.1. Effect of corner node selection strategy

For the TRAM blade, there was little benefit in using the enriched corner node problem over the strict. However, the impact

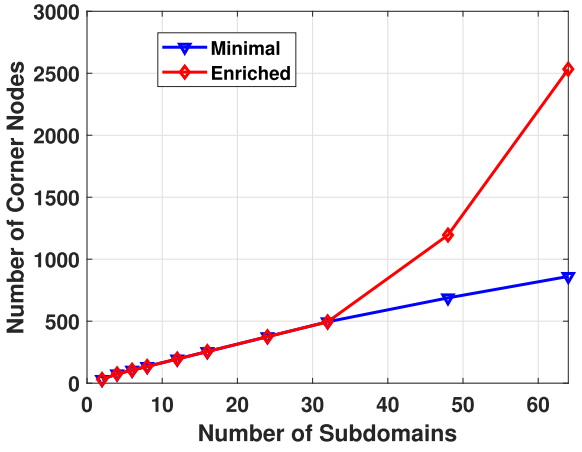


Fig. 30. Number of corner nodes.

can be dramatic for a multibody structure. Fig. 29a and 29b show the TRAM model partitioned into four subdomains with eight corner nodes per subdomain chosen using the strict definition and robust selection. For this low number of subdomains, the minimal and enriched coarse problems will be the same, with the corner nodes obtained using the robust selection strategy. For this section, this case will be known as the robust corner selection. Just four subdomains suffice to illustrate the need for robust corner node selection.

Recall that the strict selection is dependent on the partitioned geometry, and in this case results in a poor approximation to the full problem. The robust corner node selection strategy spreads out the corner nodes, resulting in a good approximation. Table 4 compares the coarse problem characteristics. The number of corner nodes and the size of the coarse skyline are the same, indicating that the coarse problem solution time will be nearly identical. However, the condition number for the strict selection is orders of magnitude higher, leading to a very large initial residual.

Fig. 24 shows the robust selection strategy produces a lower residual and a faster convergence compared to the strict selection case, regardless of how many corner nodes are used. As the number of subdomains is increased, this makes it impractical to fully converge cases using the strict selection. The use of strict corner selection, while acceptable for the blade only, is highly dependent on the partitioned geometry and therefore not sufficient for a generic problem.

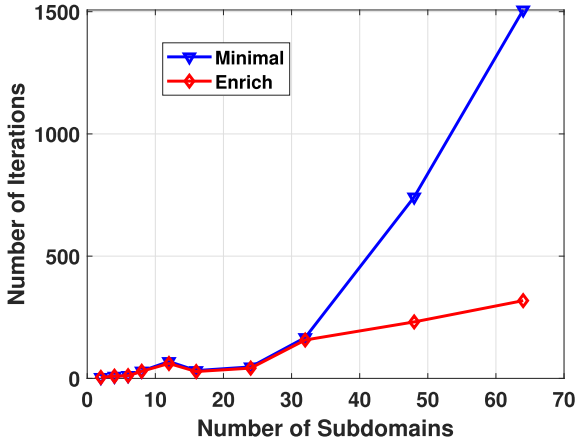
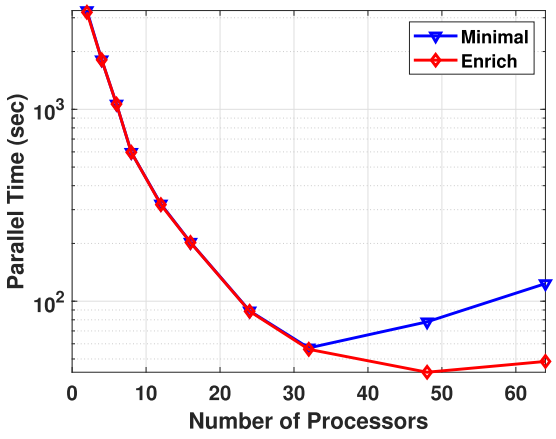


Fig. 31. Number of Iterations required.

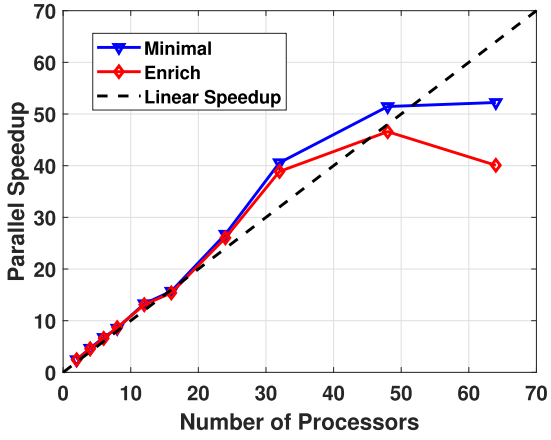
9.2. Effect of corner node enrichment

The previous section illustrated the need for the robust selection strategy for complex multibody problems. This section compares the minimal and enriched corner problems, focusing on higher number of subdomains. Fig. 30 shows the number of corner nodes for both corner problems. Note that the number of corner nodes remains the same until 48 subdomains – this marks the when nodes begin to be shared by more than three subdomains. Fig. 31 compares the number of solver iterations for the two corner problems. Two conclusions are drawn. First, any difference between the enriched and minimal coarse problems is only seen for large number of subdomains, in this case greater than 32. Second, the minimal corner problem causes a large increase in iterations with the number of subdomains.

Fig. 32a shows the parallel computational time for the two corner problems. Again, differences are only seen for more than 32 subdomains. For these cases, the enriched coarse problem provides nearly a 50 percent reduction in computational time. Looking at the parallel speedup in Fig. 32b similar trends to previous problems are observed – higher efficiency does not necessarily mean the best throughput, due to the definition of speedup used in this paper.

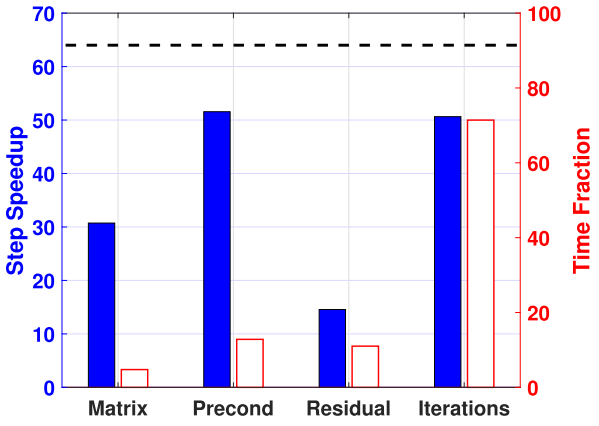


(a) Parallel solver time.

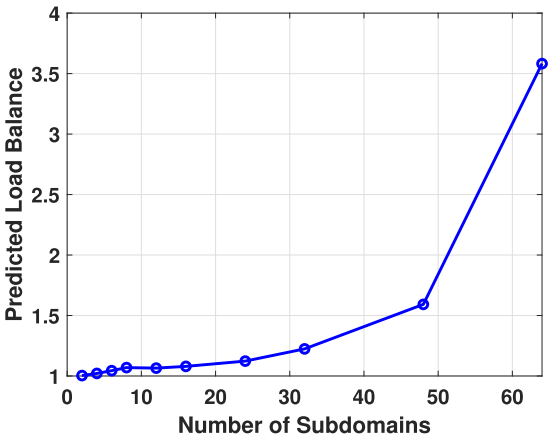


(b) Parallel speedup.

Fig. 32. The solver performance is compared between the two corner selection strategies.



(a) Performance for 64 processors.



(b) Predicted Load Balance from Mesh Partitioner.

Fig. 33. The limits to scalability are explored for the NASA TRAM Blade.

From the last two sections, we conclude that the strict definition for corners cannot be used for a general multibody problem. For moderate numbers of subdomains, both the minimal and enriched coarse problem will produce identical results. But ultimately, the enriched corner problem should be used for the best performance. The use of the enriched corner problem will not hurt performance, even for a low number of subdomains, so it is always the recommended strategy.

9.3. Limit of scalability

This section explores the limits to solver scalability for the TRAM model. Fig. 32b shows that linear speedup is achieved up to 48 subdomains before decreasing. Fig. 33a shows the detailed speedup for the 64-subdomain case with the enriched corner problem. The trends match those seen for the TRAM blade at 128-subdomains. There are still some inefficiencies due to communication, but these only make up about 3% of the time required for iterations. The main barrier to scalability is again the presence of joints. Fig. 33b shows the predicted load balance versus the number of subdomains. The problem is relatively well balanced until 48 subdomains but jumps drastically for 64 subdomains. As seen before, cache benefits can overcome small load imbalances and maintain linear scalability up to 48 subdomains, but are not able to compensate any further. As in the previous section, the roll off in efficiency can be predicted by the partitioner before execution, allowing the user to choose a load balanced partition.

10. Conclusions

A specialized mesh partitioner was developed for large-scale rotor structures which use 3D finite element parts connected with multibody joints. Several special features are included in the partitioner specifically for rotary-wing structures. A state-of-the-art iterative substructuring algorithm, FETI-DP, was implemented to verify the partitioner, and the integrated solver was validated with hover test data. The solver performance and scalability were measured for test problems of increasing complexity, with the final being the NASA 1/4 scale V-22 model rotor. For each case, the sensitivity of the solver to the general and unique partitioner features was documented to understand its performance and scalability. Based on this work, the following conclusions are drawn:

1. The parallel and scalable solution of a large-scale, multibody, rotary-wing blade and hub assembly is achievable. However, a specialized mesh partitioner is needed to accommodate the unique requirements of the multibody system.
2. For complex, multibody problems, joints are the main barrier to scalability by causing poor load balance. However, it is possible to predict the load imbalance up front, allowing the user to execute near peak efficiency. Despite the presence of joints, a TRAM blade and hub model of 250,000 DOF showed linear speedup up to 48 subdomains, with solution time reduced from over 30 min to about 40 s.
3. The use of skyline storage for the coarse problem is crucial for extending the scalability and performance of the solver. For a finite element part with no joint, the coarse problem barrier is eliminated, and linear speedup is achieved up to 1000 processors for a problem of size 6.6 million DOF. For practical rotary-wing structures, additional corner nodes are added to improve solver performance without significantly increasing the coarse problem solution time.
4. For realistic rotary-wing structures, corner nodes must be carefully selected and enriched to maximize performance. Although the speedup decreases due to additional corner nodes, the computational time generally decreases by nearly 50 percent across the board when using corner node enrichment. The robust corner node selection strategy used is automatic and is described in the paper.
5. The cross-sectional partitioning strategy is essential for maximizing performance specially for a rotary-wing structure with an elongated aspect ratio.
6. The partitioner integrated with a solver was validated on the NASA TRAM rotor test case in hover for rotor thrust and power. Unlike conventional beam-based analysis, aeroelastic stresses were directly predicted in the blades and hub. These stresses are impossible to measure in flight, and demonstrate the effectiveness of the new solver.

The new solver opens up brand new areas of physical investigation and design optimization. The 3D stresses and strains in hover and forward flight will now be available for future engineers. These remain the tasks for the future.

Data availability

The authors do not have permission to share data.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work presented in this paper was carried out at the Alfred Gessow Rotorcraft Center, University of Maryland at College Park, under the Army/Navy/NASA Vertical Lift Research Center of Excellence (VLRCOE) grant (number W911W61120012), with Dr. Mahendra Bhagwat as technical monitor. The authors would like to thank Dr. Roger Strawn (U.S. Army) and Dr. Andrew Wissink (U.S. Army) for their guidance on this work, as well as Dr. William Warmbrodt and Dr. Wayne Johnson from NASA Ames Research Center for access to the TRAM geometry. This work was also supported by the National Science Foundation Graduate Research Fellowship Program under No. DGE 1840340.

References

[1] Johnson W, Datta A. Requirements for next generation comprehensive analysis of rotorcraft. In: American helicopter society specialist's conference on aeromechanics, San Francisco, CA, January 23–25; 2008. https://ntrs.nasa.gov/citations/20080047714.
[2] Johnson W. A history of rotorcraft comprehensive analyses. Tech Rep NASA/TP-2012-216012; April 2012. https://ntrs.nasa.gov/citations/20130014338.
[3] Kvaternik RG. A government/industry summary of the design analysis methods for vibrations (DAMVIBS) Program. Tech Rep NASA-CP-10114, NASA; January 1993. https://ntrs.nasa.gov/citations/19930012121.
[4] Yeo H, Chopra I. Coupled rotor/fuselage vibration analysis using detailed 3-D airframe models. Math Comput Modell 2001;33(10):1035–54. May–June. https://doi.org/https://doi.org/10.1016/S0895-7177(00)00299-5.
[5] Yang M, Chopra I, Haas DJ. Vibration prediction for rotor system with faults using coupled rotor-fuselage model. J Aircraft 2004;41(2):348–58. March–April. https://doi.org/https://doi.org/10.2514/1.9330.
[6] Datta A, Nixon M, Chopra I. Review of rotor loads prediction with the emergence of rotorcraft CFD. J Am Helicopter Soc 2007;52(4):287–317. Octobr, https://doi.org/https://doi.org/10.4050/JAHS.52.287.
[7] Datta A. X3D - A 3D solid finite element multibody dynamic analysis for rotorcraft. In: American helicopter society technical meeting on aeromechanics design for vertical lift, San Francisco, CA, January 20–22; 2016. https://vtol.org/store/product/x3d-a-3d-solid-finite-element-multibody-dynamic-analysis-for-rotorcraft-11008.cfm.
[8] Staruk W, Datta A, Chopra I, Jayaraman B. An integrated three-dimensional aeromechanics analysis of the NASA tilt rotor aeroacoustic model. J Am Helicopter Soc 2018;63(3):1–12. https://doi.org/https://doi.org/10.4050/JAHS.63.032002.
[9] Staruk W, Datta A. Gimbaled tiltrotor conversion flight loads prediction using three-dimensional structural analysis. AIAA J Aircraft 2019;56(2):758–70. March-April. https://doi.org/https://doi.org/10.2514/1.C035075.
[10] Farhat C, Lesoinne M, Pierson K. A scalable dual-primal domain decomposition method. Numer Linear Algebra with Appl 2000;7(7–8):687–714. https://doi.org/https://doi.org/10.1002/1099-1506(200010/12)7:7/8%3C687::AID-NLA219%3E3.0.CO;2-S.
[11] Farhat C, Lesoinne M, LeTallec P, Pierson K, Rixen D. FETI-DP: a dual-primal unified FETI method - part i: a faster alternative to the two-level FETI Method. Int J Numer Methods Eng 2001;50(7):1523–44. https://doi.org/https://doi.org/10.1002/nme.76.
[12] Datta A, Johnson W. Three-dimensional finite element formulation and scalable domain decomposition for high fidelity rotor dynamic analysis. J Am Helicopter Soc 2011;56(2):1–14. https://doi.org/https://doi.org/10.4050/JAHS.56.022003.
[13] Datta A, Johnson W. Large-scale domain decomposition for a scalable, three-dimensional brick finite element based rotor dynamic analysis. In: American helicopter society specialists' conference on aeromechanics, San Francisco, CA, Jan 20–22; 2010. https://apps.dtic.mil/sti/citations/ADA532063.
[14] Devine KD, Boman EG, Riesen LA, Catalyurek UV, Chevalier C. Getting started with zoltan: a short tutorial," combinatorial scientific computing, edited by U. Naumann, O. Schenk, H.D. Simon, and S. Toledo, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Dagstuhl, Germany; 2009. http://drops.dagstuhl.de/opus/volltexte/2009/2088.
[15] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. Siam J Scient Comput 1999;20(1):359–92. https://doi.org/https://doi.org/10.1137/S1064827595287997.
[16] Duff IS, Erisman AM, Reid JK. Direct methods for sparse matrices. 1st ed.,. Oxford University Press; 1986.
[17] Lumba R, Datta A. Scalable mesh partitioning for large-scale 3D finite element-multibody structures. In: American helicopter society 76th annual forum; 2020. https://vtol.org/store/product/scalable-mesh-partitioning-for-largescale-3d-finite-elementmultibody-structures-16442.cfm.
[18] Lumba R, Datta A. Development of a parallel 3D FEA multibody solver and partitioner for rotorcraft. In: Proceedings of the AIAA Scitech Forum; 2022. https://doi.org/https://doi.org/10.2514/6.2022-2410.
[19] Johnson W. Calculation of tilt rotor aeroacoustic model (TRAM DNW) performance, airloads, and structural loads. in: American helicopter society aeromechanics specialists' meeting; 2000. https://apps.dtic.mil/sti/citations/ADA480704.
[20] Acree CW, Martin P, Romander E. Impact of airfoils on aerodynamic optimization of heavy lift rotorcraft. in: American helicopter society vertical lift aircraft design conference; 2006. https://apps.dtic.mil/sti/citations/ADA517168.