A Parallel Finite-Element Framework for Large-Scale Gradient-Based Design Optimization of High-Performance Structures

Graeme J. Kennedy* and Joaquim R.R.A. Martins†

Abstract

Structural optimization using gradient-based methods is a powerful design technique that is well suited for the design of high-performance structures. However, the ever-increasing complexity of finite-element models and design formulations results in a bottleneck in the computation of the gradients required for the design optimization. Furthermore, in light of current high-performance computing trends, any methods intended to address this bottleneck must efficiently utilize parallel computing resources. Therefore, there is a need for solution and gradient evaluation methods that scale well with the number of design variables, constraints, and processors. We address this need by developing an integrated parallel finite-element analysis tool for gradient-based design optimization that is designed to use specialized parallel solution methods to solve large-scale high-fidelity structural optimization problems with thousands of design variables, millions of state variables, and hundreds of load cases. We describe the most relevant details of the parallel algorithms used within the tool. We present consistent constraint formulations and aggregation techniques for both material failure and buckling constraints. To demonstrate both the solution and functional accuracy, we compare our results to an exact solution of a pressure-loaded cylinder made with either isotropic or orthotropic material. To demonstrate the parallel solution and gradient evaluation performance, we perform a structural analysis and gradient evaluation for a large transport aircraft wing with over 5.44 million unknowns. The results show near-ideal scalability of the structural solution and gradient computation with the number of design variables, constraints, and processors, which makes this framework well suited for large-scale high-fidelity structural design optimization.

1 Introduction

Over the past few decades, increasingly powerful high-performance computational resources and the development of sophisticated numerical algorithms have enabled the solution of large-scale, high-fidelity structural design optimization problems [Venkataraman and Haftka, 2004]. We use the term large-scale to refer to design problems with a large number of design variables, structural

*Assistant Professor, Georgia Institute of Technology, School of Aerospace Engineering, Atlanta, Georgia, 30332-0150, graeme.kennedy@aerospace.gatech.edu
†Associate Professor, University of Michigan, Department of Aerospace Engineering, Ann Arbor, Michigan, 48109, jrram@umich.edu
state variables, load cases, or constraint functions, or some combination thereof, such that significant high-performance parallel computing resources are required to solve the problem within a reasonable time. This definition will change with advances in high-performance computing hardware. At present, this definition of large-scale translates to design problems with more than $O(10^5)$ design variables, $O(10^6)$ state variables, or $O(10^2)$ load cases. Several authors have presented solution methods for large-scale problems, including structural shape and sizing problems [Padula and Stone, 1998, Papadrakakis et al., 2003, 2001], large 3D topology problems with $O(10^6)$ design variables [Borrvall and Petersson, 2001, Wang et al., 2007], and high-fidelity multidisciplinary design optimization problems involving structural analysis as a discipline with $O(10^6)$ state variables [Kenway et al., 2014].

In this paper, we present an integrated approach to parallel analysis and gradient evaluation for large-scale structural design optimization problems. We have developed this framework for the analysis and design of the thin-shell structures that are used in many high-performance aerospace applications where strength, weight, and stiffness are critical design considerations. These aerospace structures are manufactured using high-performance materials, such as laminated composites or advanced metallic alloys, that achieve high stiffness-to-weight and strength-to-weight ratios. As a result, the design problem involves the simultaneous consideration of the geometry of the structure, the sizing of the members and, in the case of composites, manufacturing details such as the lamination stacking sequence [Abrate, 1994]. Therefore, the structural design problem may include stacking sequence design optimization schemes that can significantly increase the dimensionality of the design space [Stegmann and Lund, 2005, Hvejsel and Lund, 2011, Hvejsel et al., 2011, Kennedy and Martins, 2013a]. In addition, slender shell structures subjected to in-plane loading are susceptible to buckling and, as a result, the structural requirements frequently include both strength and buckling constraints. Within this framework, we impose the buckling constraints using a global-local analysis approach in which a global model determines the edge-loads for a local stiffened panel buckling problem. Other authors have reformulated these design problems using a bilevel approach where the global design problem determines the thicknesses, and the local design problem determines the lamination sequence [Liu and Haftka, 2004, Liu et al., 2000a, Herencia et al., 2008, Liu et al., 2008].

Although gradient-free optimization methods have been successfully applied to many important structural design problems, including lamination stacking sequence design [Le Riche and Haftka, 1993, Liu et al., 2000b, Adams et al., 2004] and sizing and shape optimization problems [Jansen and Perez, 2011], these applications involve at most $O(10^2)$ design variables. Gradient-free methods are easy to use since they require only function values, but they scale very poorly with the dimensionality of the design space. Since we focus on large-scale high-fidelity applications, we use gradient-based methods and tackle the challenge of efficiently evaluating the gradient of the objective and constraint functions in the design optimization problem.

When we evaluate the gradients required for optimization there are two main concerns: computational time and accuracy. Long gradient computational times might limit the number of optimization iterations that can be performed, while low accuracy might limit the ability of the optimizer to solve the optimization problem to a tight convergence tolerance.

For large-scale structural design problems with large numbers of design variables and constraints, the computational time required to compute the gradients exceeds the computational time required for the analysis. Therefore, gradient evaluation is frequently the computational bottleneck [Venkataraman and Haftka, 2004]. To minimize the gradient computational time, we use
the adjoint method, which requires additional computational time for each gradient. However, the overall gradient evaluation time scales very weakly with the number of design variables. We detail the costs of our adjoint implementation in Section 5.3.

To address gradient accuracy, we focus on minimizing the amount of computational time required to evaluate the gradient to the maximum accuracy possible. To achieve this goal, we do not use finite-difference methods to evaluate the derivatives. Instead, we exclusively use hand-coded derivative routines that achieve good computational performance while avoiding the subtractive cancellation issues suffered by finite-difference methods. We note that other authors have used automatic differentiation methods, rather than hand-coded routines, to obtain accurate derivatives [Mader et al., 2008]. To verify the accuracy of our derivative implementation, we use the complex-step derivative evaluation technique. The complex-step method uses a complex perturbation of the variables to determine the derivative and does not suffer from subtractive cancellation. As a result, a very small step size may be used, yielding derivatives with the same number of significant digits as the functional estimate [Squire and Trapp, 1998, Martins et al., 2003].

In this paper we present a fully verified, integrated framework for the parallel analysis and gradient-evaluation of shell structures. We verify that our methods achieve the optimal solution and functional accuracy. In our opinion, within the context of design optimization, functional accuracy and solution accuracy are of equal importance, yet functional accuracy is often overlooked in structural optimization applications. In addition, we verify our gradient evaluation methods using a complex-step derivative approximation technique that enables accurate verification of the derivatives without loss of accuracy due to subtractive cancellation. We have integrated the developments presented in this paper into a sophisticated parallel finite-element code that we call the Toolkit for the Analysis of Composite Structures (TACS). We have used TACS for large-scale structural analysis of composite beams [Kennedy and Martins, 2012a], structural topology optimization [Lee and Martins, 2012, Lee et al., 2012], lamination sequence design [Kennedy and Martins, 2013a], and both static and dynamic aeroelastic design optimization [Kennedy and Martins, 2010, Kenway et al., 2014, Kennedy and Martins, 2012b, Liem et al., 2012, Kennedy and Martins, 2013b].

1.1 The model optimization problem

Since structural weight reduction is critical in many aerospace applications, the most common structural design problem is to minimize the structural mass subject to stress and possibly buckling constraints. These structural constraints are imposed at a series of design load cases to ensure the safety of the aerospace vehicle within a prescribed operational envelope. With this standard structural design optimization problem in mind, we pose the following generic structural design optimization problem:

\[
\begin{align*}
\text{minimize } & \quad f(x, \mathbf{u}_1, \ldots, \mathbf{u}_{n_\ell}) \\
\text{with respect to } & \quad x, \mathbf{u}_1, \ldots, \mathbf{u}_{n_\ell} \\
\text{governed by } & \quad \mathbf{R}_i(x^N(\mathbf{x}_G), \mathbf{x}_M, \mathbf{u}_i) = 0 \quad \text{for } 1 \leq i \leq n_\ell \\
\text{such that } & \quad f_i(x, \mathbf{u}_i) \leq 1 \\
& \quad x_l \leq x \leq x_u
\end{align*}
\]
where \( f(x, u_1, \ldots, u_n) \) is the objective function and \( f_i(x, u_i) \in \mathbb{R}^n \) represents a vector of constraints for the \( i^{th} \) load case. Note that there are a total of \( n_l \) load cases. The design variables \( x = (x_G, x_M) \in \mathbb{R}^{n_x} \) are partitioned into either geometric or material design variables that we denote \( x_G \in \mathbb{R}^{n_{xG}} \) and \( x_M \in \mathbb{R}^{n_{xM}} \), respectively. The distinction between geometric and material design variables arises at the element level: geometric design variables modify the element nodes, and material design variables modify the element constitutive behavior. This distinction between design variables is critical for high-performance derivative evaluation methods. Treating the geometric and material variables uniformly would lead to computationally inefficient derivative evaluation methods. The finite-element residuals \( R_i \in \mathbb{R}^{6n} \) depend on the finite-element nodal locations \( X_N(x_G) \in \mathbb{R}^{3n} \), the material design variables \( x_M \), and the state variables \( u_i \in \mathbb{R}^{6n} \), for the \( i^{th} \) load case. Note that we write the nodal locations as functions of the geometric design variables to ensure that the partial derivative \( \partial X_N / \partial x_G \) is computed only once in each gradient evaluation.

While there are numerous techniques available for solving the optimization problem (1), we employ a reduced-space approach where the governing equations for each load case, \( R_i(X_N(x_G), x_M, u_i) = 0 \), are solved at each optimization iteration, and the optimization problem is recast solely in terms of design variables. In the reduced-space approach, which is also referred to as the nested analysis and design (NAND) architecture [Haftka, 1985], the state variables are implicit functions of the design variables, and the adjoint or direct method must be used to determine the objective and constraint gradients. The reduced-space method is an alternative to full-space approaches that solve the design and analysis problems simultaneously [Haftka, 1985, Biros and Ghattas, 2005a,b]. We do not solve the optimization problem (1) directly using our framework, but we instead provide the objective and constraint values and gradients to a gradient-based optimizer. Typically, we solve the optimization problem with the Python-based optimization interface pyOpt [Perez et al., 2012], which provides access to several optimization packages. The optimization package we use here is SNOPT [Gill et al., 2005]. While there is ongoing work to parallelize optimization algorithms, especially in the context of PDE-constraint optimization [Biros and Ghattas, 2005a], for many applications the optimization algorithm requires two or three orders of magnitude less computational time than the finite-element analysis and gradient evaluation. For cases where this computational discrepancy is very large, it is still effective to use a serial gradient-based optimizer as long as the finite-element analysis and gradient evaluation are parallelized efficiently.

The remainder of this paper is structured as follows: In Section 2, we briefly describe the linear shell element used for the structural analysis. In Section 3, we describe the parallel finite-element methods we use to solve the linear systems arising from the finite-element discretization of shell structures and in the adjoint method. In Section 4, we describe the functions of interest that are used within the optimization problem (1). Finally, in Section 5, we describe the gradient evaluation method that we have implemented and we evaluate its computational performance.

## 2 Shell formulation

In this section, we briefly describe the general-purpose higher-order shell element formulation used for linear analysis within our integrated framework. We have also implemented higher-order two-dimensional quadrilateral plane stress and three-dimensional hexahedral elements for both geometrically linear and nonlinear analysis. We focus on the more challenging shell elements, but we note that the analysis and gradient evaluation techniques for the other elements are implemented
in an identical fashion to that for the shell elements.

The shell element employs a mixed-interpolation of tensorial components (MITC) formulation [Dvorkin and Bathe, 1984, Bucalem and Bathe, 1993, Bathe et al., 2000] to avoid shear and membrane locking. One of the key advantages of the MITC-based approach is that it easily extends to high-order shell element formulations. The formulation enables strain-free, linearized rigid body rotations. The drilling degrees of freedom are added based on a penalization approach to facilitate shell-shell intersections and shell-beam connections [Hughes and Brezzi, 1989, Fox and Simo, 1992]. In addition, the drilling degree of freedom formulation enables the use of block-based matrix algorithms since the number of degrees of freedom per node is fixed across the entire structural mesh [Simo, 1993]. Following Milford and Schnobrich [1986], the formulation uses an explicit integration of the strain energy through the thickness, enabling the direct use of the classical first-order deformation theory (FSDT) constitutive relationships. This explicit integration approach introduces a modeling error on the order of the ratio of the thickness to the minimum radius of curvature [Buechter and Ramm, 1992]. As a result, the shell element is not appropriate for very thick shells or shells with very high curvature.

2.1 The element implementation

In the finite-element implementation, we use bi-Lagrange shape functions of order $p$ to interpolate the mid-surface displacements, $U_0$, and the small rotation angles, $\theta$, for each element in the domain as follows:

$$\begin{bmatrix} U_0(\xi) \\ \theta(\xi) \end{bmatrix} = N(\xi) u^e_{ij} \quad (2)$$

where $N(\xi) \in \mathbb{R}^{6 \times n e}$ are the shape functions, $\xi$ are the isoparametric coordinates, and $u^e_{ij} \in \mathbb{R}^{6p^2}$ are the element state variables for the $i$th load case for the $j$th finite-element. The element residual is derived based on the method of virtual work and takes the form

$$\delta u^e_i^T R^e_{ij}(X^e_j(x_G), x_M, u^e_{ij}) = \delta W^e,$$

where $R^e_{ij}(X^e_j(x_G), x_M, u^e_{ij})$ are the element residuals for the $i$th load case for the $j$th element as a function of the element state variables, $u^e_{ij}$, the element nodal locations, $X^e_j(x_G)$, and the material design variables, $x_M$.

The element nodal locations and element displacements and rotations are obtained using the element injection operators $P_j \in \mathbb{R}^{n \times p^2}$, for $j = 1, \ldots, N_e$, which distribute the element residual components to the global residual as follows:

$$R_i(X^N(x_G), x_M, u_i) = \sum_{j=1}^{N_e} (P_j \otimes I_6) R^e_{ij}(X^e_j(x_G), x_M, u^e_{ij}),$$

where $\otimes$ denotes the Kronecker product and $I_6$ is the $6 \times 6$ identity matrix. Note that the element nodal displacements and rotations, $u^e_{ij}$, as well as the element nodal locations, $X^e_j(x_G)$, can also be obtained from the global solution vector and global nodal location vector using the injection operators, $P_j$, as follows:

$$u^e_{ij} = (P_j^T \otimes I_3) u_i,$$

$$X^e_j(x_G) = (P_j^T \otimes I_3) X^N(x_G),$$

where $I_3$ is the $3 \times 3$ identity matrix.
3 Parallel finite-element analysis

Parallel structural finite-element solvers used for gradient-based optimization must perform three central tasks efficiently in parallel: the assembly of the residual and stiffness matrix; the solution of linear systems arising from the finite-element discretization; and the parallel evaluation of functions and gradients required for design optimization. Of these three tasks, the most challenging to implement efficiently in parallel is the solution of the linear systems. In this framework, because of the poor conditioning of the stiffness matrices associated with the analysis of thin shells, we use a parallel direct matrix factorization. However, as noted previously, the bottleneck in gradient-based design optimization is usually the gradient evaluation. Therefore, we seek an approach in which we balance the performance of assembly tasks, required for the evaluation of constraint and objective gradients, and the cost of the direct matrix factorization. This requirement leads us to use a domain-decomposition-based approach to the direct matrix factorization.

Typically, the matrices arising from the finite-element discretization of structures are symmetric. Here, however, we are interested in aeroelastic applications where nonlinear follower forces are present [Kennedy and Martins, 2010]. Follower forces are nonconservative and introduce nonsymmetric terms in the Jacobian of the structural residuals [Elishakoff, 2005]. The simplest approach would be to neglect the nonsymmetric part of the Jacobian. However, to obtain accurate gradients it is critical to use the exact Jacobian of the structural residuals. Therefore, we concentrate on developing general nonsymmetric direct solvers that can be used even when follower forces are present.

There are many algorithms that solve nonsymmetric sparse linear systems using parallel direct solution methods. Amestoy et al. [2000] developed the code MUMPS (MUltifrontal Massively Parallel Solver), a multi-frontal matrix factorization framework for distributed and multi-core architectures that uses partial pivoting to achieve stability. To achieve parallelism, each front in the factorization is assigned to a different processor. As the factorization proceeds, the size of the frontal matrices grows. Once the frontal matrices reach a certain size, they are factored across groups of processors using a row-oriented storage format. Finally, the root frontal matrix is factored in parallel using a dense 2D block-cyclic factorization algorithm in ScaLAPACK [Blackford et al., 1996]. The direct solver SPOOLES (SParse Object Oriented Linear Equations Solver) developed by Ashcraft and Grimes [1999] also uses a multi-frontal factorization approach with either a partial or full pivoting strategy. In SPOOLES, parallelism in either distributed or multi-core environments is also achieved by assigning the fronts to different processes. Li and Demmel [2003] developed SuperLU_DIST, a parallel distributed direct solver for both symmetric and nonsymmetric linear systems. SuperLU_DIST uses a sparse block-cyclic data storage format where the size of the blocks is determined based on an analysis of the matrix. It also uses a static pivoting approach to factorization with preprocessing heuristics designed to maximize the entries on the diagonal of the matrix. Excellent parallel performance is achieved by utilizing a process queue to enable the execution of independent factorization tasks and by parallelizing the trailing-matrix update, which constitutes the single largest computational bottleneck in the factorization.

In the following section, we present our parallel direct matrix factorization approach. In a departure from the direct solvers discussed above, we utilize a domain-decomposition (or substructuring) approach that employs static pivoting. This approach achieves excellent parallel performance, and it conforms well with the repeated factorizations that are required at each optimization iteration.
3.1 Parallel direct matrix factorization

The matrix factorization proceeds in three main stages: the computation of the local Schur complement contribution; the global Schur complement assembly across all processors; and the global Schur complement factorization in parallel across all processors. Prior to the factorization, the matrix is computed and stored in memory in a distributed fashion across all processors. Each processor has a local contribution to the global matrix that is partitioned as follows:

$$A_i = \begin{bmatrix} B_i & E_i \\ F_i & C_i \end{bmatrix},$$

(3)

with \(i = 1, \ldots, N_p\), where \(N_p\) is the number of processors participating in the matrix factorization. The local matrix is split into blocks corresponding to unknowns in the interior of domain \(i\), and interface unknowns that lie on the border between two or more domains. The matrix \(B_i\) represents the contributions from the internal unknowns, the matrix \(C_i\) represents the contributions from the interface unknowns, and the matrices \(E_i\) and \(F_i\) couple the internal and interface unknowns for each domain. Note that the ordering of matrix \(A_i\) is constrained, since the interface unknowns must be ordered last.

The first step in the factorization process is the calculation of the local contribution to the global Schur complement. We obtain the local Schur complement contribution by first computing the LU-decomposition of the diagonal block matrix, \(B_i\), such that \(B_i = L_{B_i} U_{B_i}\). Next, we compute the local Schur complement, \(S_i\), of the local matrix (3),

$$S_i = C_i - F_i U_{B_i}^{-1} L_{B_i}^{-1} E_i.$$

(4)

Note that both the LU-factorization of the block matrix \(B_i\) and the computation of the local Schur complement (4) are independent of the calculations on any other processors. For good parallel performance it is essential to achieve a good balance of the workload so that each processor completes the local calculation in roughly the same amount of time. Unfortunately, this is often difficult to achieve, since equally sized domains may not have the same matrix structure and therefore require different computational times.

Once each processor has completed the local Schur complement computation, the global Schur complement can be assembled on all processors as follows:

$$S = \sum_{i=1}^{N_p} T_i^T S_i T_i.$$

(5)

The permutation matrices, \(T_i\), are selected to ensure a low fill-in for the LU factorization of the global Schur complement. Once the global Schur complement has been formed, it is factored such that

$$S = L_S U_S.$$

(6)

Note that both the assembly and factorization of the global Schur complement matrix require communication among all the processors.

This outline summarizes the steps involved in the direct matrix factorization. However, the performance of the implementation depends in large part on the data structures and algorithms used for each step of the factorization. We now describe in greater detail the data structures and operations that are used for the local Schur complement matrix computation (4) and the factorization of the global Schur complement (6).
3.1.1 Local Schur complement factorization

On each processor, we store the local matrix (3) in a block-compressed sparse row format (BCSR) [Saad, 2003]. In this format, the matrix elements corresponding to a single finite-element node are grouped together into a block stored contiguously in memory. For instance, the block matrix size is 6 for the finite-element discretization of shells with 3 displacements and 3 rotations per node. Our BCSR implementation uses optimized routines for specific block sizes that are designed to increase the number of arithmetic operations for each memory access, while also minimizing the number of logical control operations.

To reduce the number of fill-ins experienced in the factorization, we experimented with different orderings of the local matrix (3). A good ordering reduces the time required to factor the matrix $B_i$ and to compute the local Schur complement $S_i$ using Equation (4). We use the approximate minimum degree (AMD) algorithm of Amestoy et al. [1996] and the nested-disection (ND) algorithm from the METIS package [Karypis and Kumar, 1998] to order $B_i$ and $C_i$ independently. We also use our own implementation of AMD that enforces the requirement that the interface unknowns be ordered last and takes into consideration off-diagonal fill-ins. We call this modified version AMD off-diagonal (AMD-OD).

3.1.2 Sparse block-cyclic factorization

To achieve good parallelism during the global Schur complement factorization and to distribute the memory required for storage, we employ a sparse 2D block-cyclic matrix storage format. In this format, the global Schur complement matrix (5) is split into a series of rectangular block matrices denoted $S_{ij}$. The blocks are assigned to each processor based on a logical 2D process grid that is overlaid on the matrix. Only nonzero blocks are stored. Figure 1 illustrates the storage format with a sparse matrix and a $2 \times 3$ process grid for a 6-processor case. In our implementation, we store each block as a dense matrix in column-major ordering and use BLAS level 3 operations for all block-level operations required for the matrix factorization. Memory savings could be achieved by exploiting the sparsity within each block matrix. We have found, however, that the global Schur complement has many more entries than the local matrices (3), due to the large number of fill-ins produced during the local Schur complement computations. As a result, the global Schur complement is much more dense than the original matrix. The size of the block matrices is determined based on the matrix structure. However, there is a performance tradeoff between communication latency and memory-cache size when computing the matrix factorization [Li and Demmel, 2003].

Algorithm 1 gives an outline of the algorithm used to compute the LU-factorization for the block-cyclic matrix. Here, the function is_block_owner($i$, $j$) returns true if the block matrix $S_{ij}$ is nonzero and is assigned to the current processor; otherwise it returns false. The main loop of the algorithm consists of three main computational steps. First, the block diagonal is factored in place, $L_{ii}U_{ii} = S_{ii}$. Second, row $i$ and column $i$ are updated by applying $L_{ii}$ and $U_{ii}$ to the blocks in row $i$ and column $i$, respectively. Third, the trailing-matrix update must be applied to the remaining unfactored portion of the matrix. Note that the trailing-matrix update requires factored portions of the matrix that are not stored locally. We use a buffered approach where all required matrix components are buffered locally before being sent to the required processors.

The parallel performance of Algorithm 1 depends on the degree to which the column and row
updates, as well as the trailing matrix update, can be parallelized. The row and column updates involve only a subset of the processors in column \( i \) and row \( i \). As a result, these operations can be distributed only across these subsets of processors, resulting in suboptimal parallel performance. However, these steps constitute a small portion of the computational time in the factorization. The trailing-matrix update, on the other hand, constitutes the main cost of the algorithm and utilizes all the processors. This step is implemented efficiently in parallel.

### 3.2 Parallel direct factorization performance

We use a large finite-element model of a transport aircraft wing to demonstrate the performance of the domain-decomposition-based parallel matrix factorization. This type of structure is currently the principal application area of TACS. The transonic aircraft wing is based on the geometry of a Boeing 777-200ER wing. The wing has a semi-span of 30.45 m, a root chord of 13.6 m, and a taper ratio of 0.2. The crank in the wing occurs at a station at 30% of the semi-span. The wing structure consists of 44 chord-wise ribs spaced evenly over the wing and two span-wise spars. The wing is modeled using either an isotropic or composite material, shown in Table 1. For simplicity,
the skin thickness of the wing is set to 5 mm uniformly over the entire structure. The wing is discretized using either second-, third-, or fourth-order MITC shell elements with 907388 nodes resulting in just over 5.44 million degrees of freedom. The second-order discretization contains 912384 elements, the third-order discretization contains 228096 elements, and the fourth-order discretization contains 101376 elements. The wing is loaded with a set of aerodynamic forces computed at a 2.5 g maneuver flight condition. No follower forces are used in this example, but this omission does not impact the matrix factorization times presented here since we use static pivoting methods. Figure 2 shows the domain decomposition corresponding to a 64-processor case where each contiguously colored segment corresponds to a domain belonging to a different processor.

The calculations in this section are performed on the General Purpose Cluster (GPC) at SciNet [Loken et al., 2010]. Each node of the GPC consists of dual Intel Xeon E5540 processors with a clock speed of 2.53 GHz, with 16 GB of dedicated RAM and 8 processor cores. In these comparisons, we use only nodes connected with nonblocking 4x-DDR InfiniBand.

Figure 3 shows the factorization times for the finite-element wing model using AMD, ND, and AMD-OD ordering for 24, 32, 48, 64, 96, and 128 processors for the second-, third-, and fourth-order problems. Because of memory constraints, the problem must be run on at least 32 processors for most orderings. Note that the ordering has the largest impact on the factorization times, and that there is significantly less variation between the factor times for the second-, third-, and fourth-order problems. For the AMD-OD ordering, the computational time actually decreases slightly with increasing order in some cases. Both the AMD and AMD-OD ordering methods are effective at taking advantage of the matrix structure that exists in the higher-order problems through supernode identification [Amestoy et al., 1996]. These super nodes help to produce reordered matrices that are faster to factorize.

From Figure 3 it is clear that the AMD-OD ordering scheme results in the fastest factorization times for 24, 32, and 48 processors. However, the AMD-OD factorization times do not scale as well as those for ND and AMD. The AMD-OD ordering is effective in reducing the off-diagonal fill-ins, but these fill-ins have the greatest impact when there are fewer processors and the off-

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>ND</th>
<th>AMD</th>
<th>AMD-OD</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>40</td>
<td>60</td>
<td>80</td>
</tr>
<tr>
<td>64</td>
<td>80</td>
<td>120</td>
<td>160</td>
</tr>
<tr>
<td>96</td>
<td>120</td>
<td>240</td>
<td>320</td>
</tr>
<tr>
<td>128</td>
<td>160</td>
<td>320</td>
<td>480</td>
</tr>
</tbody>
</table>

Figure 3: Total factorization time spent on 24, 32, 48, 64, 96, and 128 processors.
Figure 4: Fraction of time spent in each step of the matrix factorization.
diagonal matrices are large. As a result, AMD-OD tends to perform better for fewer processors, and for more processors it tends to perform as well as or slightly better than AMD. In all cases, the ND ordering scales the most consistently.

Figure 4 presents a detailed breakdown of the fraction of time spent in different factorization operations normalized to the ideal line presented in Figure 3. It shows the fractions of time for the processors with the least idle time, the average idle time, and the most idle time. These correspond to the processors that require the maximum local time, average local time, and minimum local time, respectively. The local factorization time corresponds to the time to compute the block factorization of $B_i$ and the local contribution to the Schur complement $S_i$ from Equation (4). The communication time is the time required to communicate the local Schur complement contributions to the required processors for the global Schur complement. Finally, the global Schur complement time is the time required to factor the global Schur complement in the block-cyclic data format. Note that the proportion of time spent in each stage of the factorization changes as the number of processors increases. In particular, the fraction of time to factor the global Schur complement increases. This behavior makes it difficult to obtain an ideal speed-up consistently.

The efficiency of the ordering techniques can be inferred from the discrepancy between the processors with the maximum local time and the average local time; see Figure 4. Large gaps arise when one processor takes significantly longer than any of the others. Note that this gap is smallest for the AMD-OD ordering for 24, 32, and 48 processors, and it increases between 48 and 64 processors. As the number of processors increases, the gap grows more slowly for ND than for AMD or AMD-OD, while the discrepancy between the maximum and minimum local times for ND becomes smaller. The gap between the maximum and average local times is largest for the AMD ordering but does not increase significantly.

Figure 5 shows the memory required to store the full direct matrix factorization. This figure illustrates that the full matrix factorization requires between 31 and 49 gigabytes of main memory. While the total memory requirements grow with the number of processors, the memory requirement per processor decreases. Note that the AMD-OD ordering requires the least memory, while the AMD ordering requires the most for all element orders on all processors. The factorization con-
stitutes the primary memory requirement within the code, but other objects, including the stiffness matrix, require significant memory as well. We also note that the full factorization may fail when the total memory available from all processors is sufficient, but the local memory requirements are exceeded on a single processor.

3.3 Assembly and LU-solution performance

The matrix and residual assembly and LU-solution operations require an order of magnitude less computational time than the matrix factorization and are typically less performance-critical tasks. Nevertheless, it is important that these operations be as scalable as possible to enable the solution of large finite-element problems. Furthermore, we demonstrate in Section 5 that the LU solution times are critical for certain gradient-evaluation operations. Figure 6 shows the matrix and residual assembly times as well as the LU solution time for 32, 48, 64, 96, and 128 processors with ND ordering. Note that we restrict the results to a single ordering scheme, since the matrix and residual assembly are insensitive to the ordering of the unknowns, while the LU solution times for the other ordering schemes exhibit similar trends. The matrix and residual assembly of all the orders scale ideally to plotting precision, with the third- and fourth-order elements taking roughly 1.8 and 3.4 times as long as the second-order elements, respectively. The LU solution performance does not scale ideally, and the relative performance varies with element order. While the solution scales moderately well between 32 and 64 processors, the efficiency decreases significantly for 96 and 128 processors. In addition, the performance between element orders is not consistent. This behavior is due to the difficulty of implementing upper- and lower-triangular solution methods efficiently in parallel, where the ratio of communication to computation operations is much higher than for the matrix factorization. While the code does not exhibit the desired ideal speed-up behavior, a single LU solution for a system with 5.44 million degrees of freedom requires roughly 0.2 to 0.3 seconds on anywhere between 64 and 128 processors.
4  Functions of interest

Once the solution of the structural problem has been obtained, it is necessary to evaluate the functions of interest required for the optimization problem (1). In this section, we present the formulation of the functions of interest including the structural mass, material failure constraints, and buckling constraints. We then present a verification of both the solution and the $p$-norm functional for a free-cylinder test case. These verification cases ensure that both the finite-element implementation and the implementation of the functional is correct.

4.1  Structural mass

The structural mass is just the summation of the density per unit area integrated with Gauss quadrature over each element. The total mass is obtained by summing the mass contributions from each element in the entire finite-element mesh.

4.2  Failure constraints

Stress or failure constraints are imposed to limit the stress or strain within the structure to a permissible envelope defined by material allowables. We prefer the term failure constraint, since either the stress or the strain may be restricted depending on the criterion selected by the designer. The permissible envelope typically varies over the domain and depends on the structural design variables, such as the thickness or lamination stacking sequence.

In this work, we present an approach to limit the point-wise failure function written as follows:

$$F(x_M, \epsilon) \leq 1, \quad (7)$$

where $F(x_M, \epsilon) \in \mathbb{R}$ is a scalar-valued function that depends on material characterization data, the material design variables, $x_M$, and the local strain values, $\epsilon(X^j(x_G), u^j_i)$. Note that the failure function depends indirectly on the geometric design variables through the strain that is computed as a function of the element displacements and element nodal locations. Many classical failure criteria can be written in the form of Equation (7). For instance, the von Mises failure criterion can be written as follows:

$$F_{vM}(x_M, \epsilon) = \frac{\sigma_{vM}}{\sigma_{ys}} = \frac{1}{\sigma_{ys}} \sqrt{\sigma_x^2 + \sigma_y^2 - \sigma_x \sigma_y + 3 \sigma_{xy}^2}, \quad (8)$$

where $\sigma_{ys}$ is some specified yield stress. The Tsai–Wu failure criterion for composite materials can be written as follows:

$$F_{TW}(x_M, \epsilon) = F_1 \sigma_1 + F_2 \sigma_2 + F_{11} \sigma_1^2 + 2F_{12} \sigma_1 \sigma_2 + F_{22} \sigma_2^2 + F_{66} \sigma_{12}^2, \quad (9)$$

where the coefficients $F_1$, $F_2$, $F_{11}$, $F_{12}$, $F_{22}$, and $F_{66}$ are a function of the failure stress in the orthotropic material axis [Jones, 1996]. Other failure criteria, such as the maximum stress or maximum strain criterion, can be written as nonsmooth functions of the stress or strain.
4.3 Buckling constraints

We impose buckling constraints in a manner that is completely analogous to the point-wise failure constraint (7). For each component of the structure susceptible to buckling, we construct a buckling envelope that takes the following form:

\[ B(x_M, \varepsilon) \leq 1, \quad (10) \]

where \( B(x_M, \varepsilon) \in \mathbb{R} \) is a scalar valued function that depends on the material design variables, \( x_M \), and the local strain values, \( \varepsilon(x_G, u_{ij}) \). Using this approach, the buckling constraints can be handled in the same manner as the point-wise failure constraint (7).

4.4 Failure and buckling constraint aggregation

Gradient-based optimization requires the derivatives of all the constraints with respect to all the design variables. In structural optimization based on finite-element models, both the number of constraints and the number of design variables can scale with the number of finite elements. Thus, the problems may have large numbers of constraints and design variables, and the matrix of derivatives will be large and dense. As will be discussed in Section 5, there is no efficient way of computing such a matrix [Martins and Hwang, 2013]. For this reason, researchers have resorted to aggregation of the constraints.

In many structural optimization formulations, the failure and buckling constraints are imposed at only a set of discrete points within the domain, such as the quadrature points, or other parametric trial points within each element [Akgun et al., 2001, Poon and Martins, 2007]. When trial points are selected for each element, the number of constraints increases in direct proportion to the number of elements in the finite-element model.

To reduce the number of constraints for gradient-based design optimization, a number of authors have developed aggregation techniques that combine sets of the failure constraints into a single equivalent constraint in a conservative manner [Akgun et al., 2001]. In this section, we examine two failure-constraint aggregation techniques: the Kreisselmeier–Steinhauser (KS) aggregation function and the \( p \)-norm aggregation technique.

The KS function was first employed in an optimization setting by Wrenn [1989], who used it to convert a constrained optimization problem into an unconstrained problem. Other authors have used the KS function to impose structural constraints in various applications including structural sizing problems [Akgun et al., 2001, Poon and Martins, 2007], topology optimization [Pereira et al., 2004, Pars et al., 2009, Le et al., 2010], and aerostructural design optimization [Martins et al., 2004, Kenway et al., 2014]. The discrete KS function can be written as follows:

\[ KS_d(x, u) = \frac{1}{\rho} \ln \left[ \sum_{i=1}^{M} e^{\rho F(x_M, \varepsilon_i)} \right] \quad (11) \]

where \( \rho \) is a fixed parameter value and \( \varepsilon_i \) is the strain at the \( i^{th} \) trial point. As \( \rho \to \infty \), the KS approximation becomes more accurate such that in the limit, \( KS_d \to \max_i F(x_M, \varepsilon_i) \). However, large values of \( \rho \) lead to poorly conditioned optimization problems, so there is a tradeoff between accuracy and conditioning. As a result, the optimal value of the aggregation parameter is difficult
Figure 7: Comparison of the convergence behavior of the discrete KS function and the KS functional for the cylinder test problem. The discrete KS function exhibits mesh dependence, while the KS functional does not.

to determine a priori, so Poon and Martins [2007] developed an approach to adaptively select the parameter to achieve a tighter bound on the original constraints.

Another constraint aggregation technique, related to the KS function, is the discrete $p$-norm. This approach is mostly employed in topology optimization problems [Le et al., 2010]. The discrete $p$-norm function can be written as follows:

$$ PN_d(x, u) = \left[ \sum_{i=1}^{M} |F(x_M, \varepsilon_i)|^p \right]^{1/p} \tag{12} $$

where $p > 1$ is a parameter such that as $p \to \infty$, the $p$-norm approximation becomes more accurate such that in the limit, $PN_d \to \max_i |F(x_M, \varepsilon_i)|$. Like the KS function, large values of $p$ lead to poorly conditioned optimization problems.

One issue with the discrete KS function and the discrete $p$-norm is that they do not converge asymptotically as the mesh is refined for fixed values of $\rho$ and $p$. This is due to the discrete nature of the functions: they use quadrature points or other discrete sets of trial points whose number and location vary depending on the discretization. Therefore, if the discrete aggregation approaches are used as constraints within an optimization problem, the optimal solution also exhibits mesh dependence. As a result, we prefer either the continuous $p$-norm or the KS functional. The KS functional was first introduced by Akgun et al. [2001]. The KS and $p$-norm functionals are closely related to their discrete counterparts, but they exhibit asymptotic convergence behavior, since they employ integrals over the structural domain rather than discrete sums. For comparison, the convergence behavior of the discrete KS function and the KS functional are shown in Figure 7 with fixed $\rho$ for a series of meshes for the isotropic cylinder problem described in Section 4.5. The discrete KS function does not display asymptotic convergence behavior.
The $p$-norm aggregation of the failure criterion can be written as follows:

$$\text{PN}(\mathbf{x}, \mathbf{u}; p) = ||F(\mathbf{x}_M, \varepsilon)||_p = \left( \int_{\Omega} |F(\mathbf{x}_M, \varepsilon)|^p \, d\Omega \right)^{\frac{1}{p}},$$

where $p > 1$. When evaluating the $p$-norm, we use the latter expression, which is mathematically equivalent to the former but less susceptible to numerical overflow. Here $\varepsilon_m$ is the strain that produces the maximum value of the failure criterion among all quadrature points within the domain of integration. This ensures that the integrand is evaluated only at points where it is bounded by unity. We evaluate the integral in Equation (13) approximately using the same order quadrature scheme used by the finite-element implementation. The $p$-norm functional has the property that $\text{PN}(\mathbf{x}, \mathbf{u}; p) \to \max |F|$ as $p \to \infty$.

The KS functional is closely related to the $p$-norm and can be written as follows:

$$\text{KS}(\mathbf{x}, \mathbf{u}; \rho) = \frac{1}{\rho} \ln \left( \int_{\Omega} e^\rho F(\mathbf{x}_M, \varepsilon_m) \, d\Omega \right)$$

$$= F(\mathbf{x}_M, \varepsilon_m) + \frac{1}{\rho} \ln \left( \int_{\Omega} e^{\rho (F(\mathbf{x}_M, \varepsilon_m) - F(\mathbf{x}_M, \varepsilon_m))} \, d\Omega \right),$$

where $\varepsilon_m$ is the strain that produces the maximum failure criterion among all quadrature points within the domain. Again, in practice we approximate the integral in Equation (14) using the same quadrature scheme employed by each element in the domain. In a similar manner to the $p$-norm, the KS functional approaches an upper bound on the failure function: $\text{KS}(\mathbf{x}, \mathbf{u}; \rho) \to \max F$ as $\rho \to \infty$.

In theory, a single aggregation functional could be used to enforce the failure or buckling constraint over the entire structure. However, this often produces highly nonlinear constraints, which lead to poor optimization performance [Poon and Martins, 2007]. Instead, we use a series of $n_d$ independent aggregation domains, $\Omega_i$, with $i = 1, \ldots, n_d$, and aggregate the failure or buckling criterion separately over each domain. These subdomains satisfy the following properties:

$$\bigcup_{i=1}^{n_d} \Omega_i = \Omega,$$

$$\Omega_i \cap \Omega_j = \emptyset \quad \forall i \neq j.$$  

(15)

Instead of using a random domain assignment or domains based on the parallel domain decomposition, we use aggregation domains that share a common structural purpose or component. For instance, in the case of a wing-box optimization, we often form aggregation domains from all or part of the top skin, bottom skin, ribs, spars, and stiffeners, independently. This ensures that the design variables that directly modify the structure have similar scaling, and that the constraints have a physical significance that is easy to understand (e.g., the failure constraint for the top skin).

4.5 Solution and functional accuracy

In this section, we present a verification of both the solution and the functional accuracy for our implementation. These tests demonstrate the accuracy of the second-, third-, and fourth-order shell
elements for the analysis of both composite and isotropic materials. In addition, we demonstrate
the accuracy of the functional estimates of the $p$-norm by comparison with the $p$-norm of the exact
solution for both the von Mises and Tsai–Wu failure criteria. In addition to these tests, we have also
verified that the elements pass the classical MacNeal–Harder tests [MacNeal and Harder, 1985] for
shell elements; we omit these results.

In this study, we use an exact solution of a specially orthotropic circular cylindrical shell that is
subject to a distributed pressure load. A presents an outline of the derivation of the exact solution
as well as the exact value of the $p$-norm functional for both an isotropic and a specially orthotropic
cylinder. The length of the cylinder is $L = 100$, the radius is $a = 100/\pi$, and the thickness is $t = 1$.
The cylinder is subject to a distributed pressure load as follows:

$$p(\theta, z) = p_0 \sin \left( \frac{4\theta}{R} \right) \sin \left( \frac{3\pi z}{L} \right),$$  \hspace{1cm} (16)

where $z$ is the axial direction and the magnitude of the load, $p_0$, will be defined below.

Instead of using a finite-element mesh aligned with the directions of principal curvature, we
analyze the full cylinder and use a distorted mesh parametrized as follows:

$$z = L \xi_1 + \frac{L}{10} \sin(2\pi \xi_1) \cos(2\pi \xi_2),$$

$$\theta = 2\pi \xi_2 + \frac{\pi}{4} \xi_1 + \frac{1}{10} \cos(2\pi \xi_1) \sin(2\pi \xi_2),$$  \hspace{1cm} (17)

where $(\xi_1, \xi_2) \in [0, 1]^2$. The mesh for the cylinder under consideration is shown in Figure 8; it
consists of $m$ elements in the axial direction and $2m$ elements in the circumferential direction.
These dimensions are selected to ensure that the aspect ratio of the elements is roughly unity.

In the context of a structural optimization problem, the failure function should be bounded
by 1 everywhere, such that the structure can sustain the applied load without a material failure.
To mimic these conditions, we select \( p_0 \) such that the maximum value of the failure criterion is unity, i.e., \( \max F = 1 \). To obtain the value of \( p_0 \), we first determine the solution for a unit load and then calculate the value of \( p_0 \) required to obtain \( \max F = 1 \). In the cases presented here, the maximum von Mises failure stress is \( \sigma_{\text{vM}} = \sqrt{a_{mn}^2 + b_{mn}^2 - a_{mn}b_{mn}} \), while the maximum Tsai–Wu criterion is \( \max F_{TW} = f_{mn} + g_{mn} \), where the coefficients \( a_{mn} \), \( b_{mn} \), \( f_{mn} \), and \( g_{mn} \) are defined in \( A \). As a result, for the case of the isotropic cylinder, the required value of \( p_0 \) is

\[
p_0 = \frac{1}{\sqrt{a_{mn}^2 + b_{mn}^2 - a_{mn}b_{mn}}} ,
\]

while for the case of the specially orthotropic cylinder, \( p_0 \) is obtained by obtaining the solution to the following quadratic equation:

\[
p_0^2 g_{mn} + p_0 f_{mn} - 1 = 0.
\]

Figure 9 shows the solution accuracy for the isotropic and specially orthotropic cylinders for the second-, third-, and fourth-order elements. All elements achieve their expected accuracy. Figure 10 shows the absolute error between the \( p \)-norm functional and the functional estimate obtained from the finite-element discretization for both the isotropic cylinder with the von Mises stress and the specially orthotropic cylinder for the Tsai–Wu failure criterion. For the second-order meshes we use \( m = 32 \) to \( m = 128 \) elements in increments of 4, for the third-order meshes we use \( m = 4 \) to \( m = 66 \) in increments of 2, and for the fourth-order meshes we use \( m = 4 \) to \( m = 44 \) in increments of 2. The finite-element mesh is distorted, so we plot the solution error and functional estimate error against the average mesh spacing \( \Delta x = L/(m+1) \), \( \Delta x = L/(2m+1) \), and \( \Delta x = L/(3m+1) \). The solution error is estimated using the \( L_2 \) norm of the displacement normal to the surface as
Figure 10: Functional $p$-norm accuracy for the isotropic and specially orthotropic cylinders. All elements exhibit the expected accuracy asymptotically.

follows:

$$||w - w_h||_2 = \left( \int_{\Omega} (w - w_h)^2 d\Omega \right)^{\frac{1}{2}},$$

where $w$ is the normal displacement. This error estimate is obtained by integrating the square of the error over the finite-element mesh using a Gauss quadrature scheme one order higher than the order of the finite-element. From Figure 9, it is clear that all elements achieve the expected accuracy for both the isotropic and specially orthotropic cylinders. The relative offset between the errors is due to the normalization of the applied load, resulting in a higher load and larger displacement for the specially orthotropic cylinder. The convergence behavior of the $p$-norm functional estimates, shown in Figure 10, is not as smooth as the solution error. For the isotropic cylinder, the $p$-norm estimate error does not behave smoothly until the average discretization length, $\Delta x/L$, decreases below 2% for the third- and fourth-order meshes. For the specially orthotropic cylinder, the second-order results do not enter a region of asymptotic convergence within the range of $\Delta x/L$ shown here, while the third- and fourth-order solutions behave smoothly below an average discretization length of $\Delta x/L < 1.75\%$.

These results show that the $p$-norm estimate obtained from the finite-element solution converges to the $p$-norm of the exact solution. However, it is also important to assess how well the $p$-norm and KS functional estimates approximate the upper bound of the failure criterion. To address this issue, Figure 11 shows the difference between the $p$-norm and KS functional estimates for the third-order meshes with $m = 66$ for increasing values of the $p$ and $\rho$ parameters for the isotropic and specially orthotropic cylinders. The two cases exhibit almost identical behavior: the KS functional estimate provides a tighter bound for the maximum value of the failure criterion, especially for low values of the parameters $p$ and $\rho$. As discussed previously, large penalty parameters are often undesirable in optimization applications since these parameters tend to increase the
Figure 11: Error between the upper bound and the $p$-norm and KS functional estimate of the upper bound for the isotropic and specially orthotropic cylinders. The KS functional provides a tighter bound.

number of optimization iterations required to obtain a solution [Poon and Martins, 2007]. Based on the example presented above and on our experience with these constraint aggregation methods in numerous optimization problems, we prefer the KS functional to the $p$-norm functional. However, to our knowledge there is no general proof of the relative accuracy of the two approaches, so the constraint aggregation technique should be chosen on a case-by-case basis.

5 Gradient evaluation

Gradient-based design optimization methods can solve large-scale design problems in a reasonable computational time only if both the analysis and gradient evaluation are performed efficiently. Furthermore, for a fixed computational budget, improving the performance of the gradient evaluation gives the user additional flexibility to modify the design problem by adding more design variables and constraints. Therefore, we seek to evaluate the gradient in such a way that the computational time scales as weakly as possible with both the number of functions of interest and the number of design variables, and parallelizes well with increasing numbers of processors. As we will show here, we must compromise by accepting a higher scaling factor for either increasing numbers of functions or increasing numbers of design variables.

Within the reduced-space paradigm, we must evaluate the gradient of any function of interest while taking into account the effect of satisfying the governing equations. This means that we require the total derivative of a vector of functions $\nabla_x f_i \in \mathbb{R}^{n_f \times n_x}$, which can be evaluated as follows:

$$\nabla_x f_i = \frac{\partial f_i}{\partial x} - \frac{\partial f_i}{\partial u} \left[ \frac{\partial R_i}{\partial u} \right]^{-1} \frac{\partial R_i}{\partial x}.$$  (18)
There are two methods that may be used to evaluate the gradient: the adjoint method and the direct method [Adelman and Haftka, 1986, Haftka and Adelman, 1989]. The ratio of the number of functions of interest to the number of design variables is the key factor that controls the relative computational cost of the adjoint and direct methods. If there are more design variables than functions \((n_x > n_f)\), then the adjoint method requires the solution of fewer linear systems. As a result, it requires less computational time. If there are more functions than design variables \((n_f > n_x)\), then the direct method requires less computational time. However, if the numbers of functions and design variables are both large, then both methods require significant computational time. By using the constraint aggregation techniques presented in Section 4.4, we ensure that the number of functions per load case can be reduced to a manageable number. Therefore, we focus on the implementation of the adjoint method.

We utilize the constraint structure of the optimization problem (1) to achieve better computational efficiency when computing the total derivative. In particular, we leverage the fact that multiple constraint gradients are required for each load case. Under these conditions the adjoint method can be obtained from Equation (18) by introducing the matrix of adjoint variables \(\psi_i\), such that

\[
\begin{bmatrix}
\frac{\partial R_i}{\partial u}
\end{bmatrix}^T \psi_i = \frac{\partial f_i}{\partial u}^T.
\]

(19)

We note that when the system of equations is linear, the matrix factorization may be reused from the analysis to solve the transposed adjoint system. Once the adjoint variables \(\psi_i\) have been determined, the total derivative (18) reduces to

\[
\nabla_x f_i = \frac{\partial f_i}{\partial x} - \psi_i \frac{\partial R_i}{\partial x}.
\]

(20)

In the following section, we describe how we implemented the adjoint in a way that seeks to minimize the computational costs when there are large numbers of geometric and material design variables. We then evaluate the computational performance and scalability of our implementation.

### 5.1 Adjoint implementation

We treat the design variables as a global vector, rather than a distributed vector, so that any finite-element on any processor may access any design variable. Therefore, the design variable vector is duplicated on all the processors. For the gradient implementation, we likewise store the full gradient, \(\nabla_x f_i\), on all the processors. Since the number of design variables is typically three orders of magnitude smaller than the finite-element model degrees of freedom, this is not an issue. Both the adjoint and direct methods require the partial derivatives of the functions of interest with respect to the design variables, \(\frac{\partial f_i}{\partial x}\). We compute this term by adding the contributions from the material and geometric design variables that are evaluated using two separate techniques. For the material design variables, we compute the derivative of the function with respect to all material design variables over each element and sum the results on all the processors. The implementation for the geometric design variables is more complex. First, we compute the derivative of the functions with respect to all nodes. Next, we multiply this term by the derivative of the nodes with respect to the
geometric design variables. This calculation can be written as follows:

$$\frac{\partial f_i}{\partial x_G} = \left[ \sum_{j=1}^{N_e} \frac{\partial f_i}{\partial X^e_j} (P^T_j \otimes I_3) \right] \frac{\partial X^N}{\partial x_G}. \tag{21}$$

As with all gradient operations, we sum the contributions locally on each processor and, after each processor has finished, sum the results across all the processors.

The first step in the adjoint method is to obtain the adjoint variables $\psi_i$ by evaluating the gradients $\frac{\partial f_i}{\partial u}$ for $j = 1, \ldots, n_f$ and solving Equation (19). We compute the derivative, $\frac{\partial f_i}{\partial u}$, by evaluating the contribution from each element, and assembling the global vector as follows:

$$\frac{\partial f_i}{\partial u} = \sum_{j=1}^{N_e} \frac{\partial f_i}{\partial u_{ij}} (P^T_j \otimes I_6). \tag{22}$$

The load balancing during the computation of the terms in Equations (21) and (22) depends on the size and distribution of the aggregation domains for the functions $f_i$. Since the union of all the aggregation domains is the entire structure, this calculation tends to scale well, as long as the aggregation domains are approximately of equal size.

To complete the gradient calculation, we must multiply the adjoint variables $\psi_i$ by the derivative of the residuals with respect to the design variables. One approach to this calculation would be to compute the derivative $\frac{\partial R_i}{\partial x_i}$, for $i = 1, \ldots, n_x$, one column vector at a time, and then to compute the inner product with each adjoint vector. However, this approach ignores the sparsity pattern of $\frac{\partial R_i}{\partial x}$ and would incur the computational cost of a dense matrix-vector product. Instead, we implement a matrix-free method that computes the matrix-vector product $\psi_i^T [\frac{\partial R_i}{\partial x}]$ directly and exploits sparsity. As before, we use separate calculations for the geometric and material design variable contributions. We compute the contribution from the geometric design variables as follows:

$$\psi_i^T \frac{\partial R_i}{\partial x_G} = \psi_i^T \frac{\partial R_i}{\partial X^N} \frac{\partial X^N}{\partial x_G} = \left[ \sum_{j=1}^{N_e} (\psi_i^T (P^T_j \otimes I_6)) \frac{\partial R^e_{ij}}{\partial X^e_j} (P^T_j \otimes I_3) \right] \frac{\partial X^N}{\partial x_G}. \tag{23}$$

This term is calculated in three stages. First, we compute the term between the square brackets independently on each processor by evaluating the derivative, $\frac{\partial R^e_{ij}}{\partial X^e_j}$ for each element, and then taking a local matrix-matrix product with all the adjoint variables corresponding to the local element. The result from each element-wise calculation is accumulated locally without communication in a temporary array. Next, after all the element contributions have been added locally, we evaluate the product of the term within the square brackets with the derivative of the nodal locations with respect to the geometric design variables, $\frac{\partial X^N}{\partial x_G}$. Finally, we perform a global reduction by adding the local contribution from each processor to a global array. The calculation of the term $\psi_i^T [\frac{\partial R_i}{\partial x}]$ is much simpler for the material design variables. This contribution can be written as follows:

$$\psi_i^T \frac{\partial R_i}{\partial x_M} = \sum_{j=1}^{N_e} (\psi_i^T (P^T_j \otimes I_6)) \frac{\partial R^e_{ij}}{\partial x_M}, \tag{24}$$

where we sum the contribution from each element on each processor, and then perform a global reduction by adding the local contribution from each processor to a global array.
5.2 Gradient accuracy verification

In this section, we verify the accuracy of the gradient calculations using the complex-step method [Squire and Trapp, 1998, Martins et al., 2003]. The complex-step method can be used to obtain high-accuracy derivative approximations, since this technique is not susceptible to subtractive cancellation. To evaluate the projected derivative with the complex-step method, we evaluate the imaginary part of the function that is perturbed by a small complex step:

$$p^T \nabla_x f = \frac{\text{Im}(f(x + ihp))}{h} + \mathcal{O}(h^2),$$

where each component of the vector $p$ is either 1 or $-1$ such that $p = \text{sign} (\nabla_x f)$. This selection ensures that all the derivative components add a positive contribution to the inner product $p^T \nabla_x f$.

To verify the derivatives, we use a small third-order mesh of the wing described above with 576 elements and 2130 nodes, with the isotropic design parametrization described above. Figure 12 shows the relative error between the projected gradient calculated using the complex step and the projected gradient obtained using the adjoint method implementation for both the $p$-norm and the KS functional. The relative error between the complex step approximation and the adjoint implementation is less than $10^{-11}$ for complex step sizes less than $10^{-8}$.

5.3 Gradient computational cost

In this section, we study the computational cost of the adjoint gradient evaluation method. We examine the scalability of the gradient with the number of processors, functions, and design vari-
ables. Our objective is to ensure that problems with large numbers of constraints and design variables scale well.

We examine the four computational steps required to evaluate the total derivative: the evaluation of the derivative \( \frac{\partial f_i}{\partial u} \), labeled “SV time,” the solution of the adjoint equations (19), the evaluation of the derivative \( \frac{\partial f_i}{\partial x} \), which we label “DV time,” and the evaluation of the inner product \( \psi_i^T \frac{\partial R_i}{\partial u} \). For these comparisons, we use the finite-element wing models from Section 3.2 and in all cases we use an ND ordering of the unknowns. Note that the performance results in this section are primarily dependent on the number of finite-elements within the model and should hold for all models of similar size. However, the adjoint solution time depends on the connectivity of the underlying mesh and thus will vary for different finite-element models with equal numbers of elements.

For this study, we examine the computational costs of both geometric and material design variables. The geometric variables are the span and chord-length of the wing, as well as the twist at a series of sections spaced evenly over the span of the wing using the geometric parametrization presented by Kennedy and Martins [2012b]. We denote the number of geometric design variables \( n_{xg} \), where there are either 11 or 201 such variables. The material design variables are either 220 thicknesses for the metallic wing, or 1320 design variables for the composite wing with 220 thickness variables, 660 ply fraction variables, and 440 lamination parameters using the parametrization of Liu et al. [2004]. We also examine the effect of varying the number of constraint aggregation domains. We study cases with \( n_d = 1, 5, 15, \) and 30, where we enforce the properties of the domain outlined in Equation (15).

Figure 13 shows the parallel speed-up of the gradient computation for second-, third-, and fourth-order elements on 32, 48, 64, 96, and 128 processors. For this study, we use 15 KS functionals with 221 geometric variables and the composite material parametrization with 1320 material design variables. Figure 13a shows the parallel scalability of the computation, while Figure 13b shows the fraction of time spent in each operation as a fraction of the ideal lines shown in Figure 13a. The gradient evaluation for the higher-order elements requires more computational time, with the fourth-order elements requiring more than 3 times longer than the second-order elements. Unfortunately, the gradient computational time does not scale ideally, with the second-order elements exhibiting the poorest parallel performance between 96 and 128 processors. This behavior is primarily due to the adjoint solution time, which increases as a fraction of the ideal time as shown in Figure 13b. The fraction of time spent in the remaining operations increases by roughly 12%, 6%, and 3% for the second-, third-, and fourth-order elements, respectively.

Figure 14 shows the computational cost of evaluating the gradient for 1, 5, 15, and 30 KS functionals with domains that satisfy condition (15). For this study, we use 64 processors, with a design parametrization that includes 221 geometric variables and the composite material parametrization with 1320 material design variables. Figure 14a shows the computational time required for the second-, third-, and fourth-order elements and the closest linear approximation of the gradient cost with number of functions. The cost of the gradient evaluation increases at a rate of approximately 0.3 seconds per function for elements of all orders. Figure 14b shows the proportion of the computational time spent in each operation normalized to the time required for the evaluation of a single gradient. The increase in the computational time is primarily a result of the additional adjoint solutions required, while the computational time for all other terms increases modestly for increasing function count. For all but the second-order case with 30 functions, the most computational time is spent in the evaluation of the term \( \psi_i^T \frac{\partial R_i}{\partial x} \). However, the computational cost of this term
Figure 13: Computational time to compute 15 KS functional gradients for increasing numbers of processors. Suboptimal scaling is due to poor LU-solution performance.
Figure 14: Gradient evaluation time for increasing numbers of KS functionals with 1521 material and geometric design variables for the 64-processor case with ND ordering.
scales only weakly with the number of functions.

Figure 15 shows the computational time required to evaluate the derivative of 15 KS functionals for increasing numbers of geometric and material design variables. Figure 15a shows the time required to evaluate the gradient for \( n_{xm} = 220 \) and 1320 material design variables, as well as for cases with \( n_{xm} = 220 \) material design variables and either \( n_{xg} = 11 \) or \( n_{xg} = 201 \) geometric design variables. In addition, Figure 15a shows the linear fit between these design variable groups. Note that the cost of evaluating the derivative with additional geometric design variables is approximately equal for all element orders: 0.017 seconds per geometric design variable. The cost of additional material design variables scales at a rate that is an order of magnitude lower than that for the geometric design variables but depends on the element order. Figure 15b shows the proportion of computational time spent in each operation normalized to the \( n_{xm} = 220, n_{xg} = 11 \) case for each element order. Note that the “adjoint-residual product” and “DV time” vary significantly with the number of design variables, while the remaining operations take roughly the same fraction of the overall time. Figure 15b demonstrates that the computational times for the term \( \psi^T \partial R_i/\partial u \) are much more significant for the geometric design variables, especially for the higher-order elements. For the second-order and third-order elements, a larger proportion of the time is spent in the solution of the adjoint vectors.

6 Conclusions

In this paper we presented a fully verified, integrated framework for parallel analysis and gradient-evaluation of shell structures. We described the implementation of a parallel direct matrix factorization method using a domain-decomposition approach that is well suited for both factorization and gradient evaluation tasks. We demonstrated that this direct factorization technique achieves excellent parallel scalability for a large-scale finite-element problem with 5.44 million degrees of freedom on between 24 and 128 processors. We examined the effect of ordering on the performance of the method and found that the AMD, ND, and AMD-OD methods all scale well over this range of processors and can be used to solve the aircraft wing case in less than 40 seconds on 128 processors for second-, third-, and fourth-order shell elements.

To demonstrate the correctness of our implementation, we also presented a series of verification studies to show that our methods achieve the optimal solution and functional accuracy for each corresponding element. We also presented a gradient evaluation technique that scales weakly with the number of design variables, scales moderately with the number of functions, and exhibits good parallel scalability for all element orders. In addition, we have verified our gradient evaluation methods using a complex-step derivative approximation, demonstrating an accuracy of \( 10^{-11} \) in all the gradient components.

We have integrated the developments presented in this paper into a sophisticated parallel finite-element code that we call the Toolkit for the Analysis of Composite Structures (TACS). These developments make our framework well suited for large-scale high-fidelity structural design optimization problems.
Figure 15: Gradient evaluation time for increasing numbers of material and geometric design variables for the 64-processor ND ordering case with 15 functions.
References


A Cylindrical shell solution

The strain expressions for a moderately deep cylinder are:

\[
\begin{align*}
\varepsilon_x &= u_x \\
\varepsilon_y &= v_y + \frac{w}{a} \\
\gamma_{xy} &= u_y + v_x \\
\kappa_x &= \psi_{xx} \\
\kappa_y &= \psi_{yy} - \frac{v_y}{a} - \frac{w}{a^2} \\
\kappa_{xy} &= \psi_{xy} + \psi_{yx} - \frac{u_y}{a} \\
\gamma_{yz} &= w_y + \psi_y - \frac{v}{a} \\
\gamma_{xz} &= w_x + \psi_x
\end{align*}
\]  \hspace{1cm} (26)

where \( u \) is the axial displacement, \( v \) is the displacement along the circumferential direction, and \( w \) is the normal displacement. In addition, \( x \in [0, L] \) is the axial position, while \( y \in [0, 2\pi a] \) is the circumferential position.

If the cylinder is subjected to a sinusoidally varying pressure distribution \( p(x, y) = \sum_{n,m} \rho_{mn} \sin(\alpha_m y) \sin(\beta_n x) \), a displacement solution can be obtained in the following form:

\[
\begin{align*}
u(x, y) &= \sum_{n,m} U_{mn} \sin(\alpha_m y) \cos(\beta_n x) \\
\phi(x, y) &= \sum_{n,m} \theta_{mn} \sin(\alpha_m y) \cos(\beta_n x) \\
v_x(x, y) &= \sum_{n,m} \phi_{mn} \cos(\alpha_m y) \sin(\beta_n x) \\
\psi_y(x, y) &= \sum_{n,m} \psi_{xy} \cos(\alpha_m y) \sin(\beta_n x) \\
w(x, y) &= \sum_{n,m} W_{mn} \sin(\alpha_m y) \sin(\beta_n x)
\end{align*}
\]

The principle of minimum potential energy, in combination with the strain expressions (26) and the assumed displacement distributions, yields the following coupled algebraic equations for \( U_{mn} \),
where the coefficients $p_{mn}, \theta_{mn}$, and $\phi_{mn}$:

$$- (A_{11} \beta_n^2 + A_{33} \alpha_m^2) U_{mn} - (A_{33} + A_{12}) \alpha_m \beta_n V_{mn} + A_1 \frac{\beta_n}{a} W_{mn} + \frac{D_{33}}{a} \left( \alpha_m^2 \theta_{mn} + \alpha_m \beta_n \phi_{mn} - \frac{\alpha_m^2}{a} U_{mn} \right) = 0$$

$$- (A_{12} + A_{33}) \alpha_m \beta_n U_{mn} + (A_{33} \beta_n^2 + A_{22} \alpha_m^2) V_{mn} + A_2 \frac{\alpha_m}{a} W_{mn} + \frac{D_{12}}{a} \alpha_m \beta_n \theta_{mn} +$$

$$\frac{D_{22}}{a} \left( \alpha_m^2 \left( \phi_{mn} - \frac{V_{mn}}{a} \right) - \alpha_m \frac{W_{mn}}{a} \right) + \frac{A_1}{a} \left( \alpha_m W_{mn} + \phi_{mn} - \frac{V_{mn}}{a} \right) = 0$$

$$- (A_{11} \alpha_{m}^2 + A_{22} \beta_{n}^2) W_{mn} - A_{22} \beta_n \theta_{mn} - \bar{A}_{11} \alpha_m \left( \phi_{mn} - \frac{V_{mn}}{a} \right) +$$

$$\frac{A_1}{a} \beta_n U_{mn} - \frac{A_2}{a} \left( \frac{W_{mn}}{a} - \alpha_m V_{mn} \right) + \beta_n \frac{D_{12}}{a^2} \theta_{mn} - \frac{D_{22}}{a^2} \left( \frac{W_{mn}}{a^2} - \alpha_m \phi_{mn} \right) + p_{mn} = 0$$

$$- (D_{11} \beta_{n}^2 + D_{33} \alpha_{m}^2) \theta_{mn} - (D_{12} + D_{33}) \alpha_m \beta_n \phi_{mn} - D_{12} \left( \frac{\beta_n}{a^2} W_{mn} - \alpha_m \beta_n \frac{V_{mn}}{a} \right) +$$

$$\frac{D_{33}}{a} \alpha_m^2 U_{mn} - \bar{A}_{22} \left( \beta_n W_{mn} + \theta_{mn} \right) = 0$$

$$- (D_{33} + D_{12}) \alpha_m \beta_n \theta_{mn} - (D_{33} \beta_n^2 + D_{22} \alpha_m^2) \phi_{mn} + \alpha_m \frac{D_{22}}{a^2} W_{mn} - D_{22} \left( \frac{W_{mn}}{a^2} - \alpha_m \frac{V_{mn}}{a} \right) +$$

$$\frac{D_{33}}{a} \alpha_m \beta_n U_{mn} - \bar{A}_{11} \left( \alpha_m W_{mn} + \phi_{mn} - \frac{V_{mn}}{a} \right) = 0.$$ (27)

Once the coefficients have been determined from Equation (27), the stresses in the cylinder can be obtained as follows:

$$\sigma_1 = \sum_{n,m} a_{mn} \sin(\alpha_{m} y) \sin(\beta_n x)$$

$$\sigma_2 = \sum_{n,m} b_{mn} \sin(\alpha_{m} y) \sin(\beta_n x)$$

$$\sigma_{12} = \sum_{n,m} c_{mn} \cos(\alpha_{m} y) \cos(\beta_n x)$$

where the coefficients $a_{mn}, b_{mn}$, and $c_{mn}$ are as follows:

$$a_{mn} = - \beta_n Q_{11} \left( U_{mn} + z \theta_{mn} \right) - Q_{12} \left( \alpha_m \left( V_{mn} \left( 1 - \frac{z}{a} \right) + z \phi_{mn} \right) - W_{mn} \frac{a - z}{a^2} \right)$$

$$b_{mn} = - \beta_n Q_{12} \left( U_{mn} + z \theta_{mn} \right) - Q_{22} \left( \alpha_m \left( V_{mn} \left( 1 - \frac{z}{a} \right) + z \phi_{mn} \right) - W_{mn} \frac{a - z}{a^2} \right)$$

$$c_{mn} = Q_{33} \left( \alpha_m \left( U_{mn} \left( 1 - \frac{z}{a} \right) + z \theta_{mn} \right) + \beta_n \left( V_{mn} + z \phi_{mn} \right) \right).$$ (28)

### A.1 p-norm evaluation

To evaluate the $p$-norm, we assume that the cylinder is subject to a load such that only one term $p_{mn} \neq 0$ for one value of the pair $(m,n)$. Under this type of loading condition, with an isotropic cylinder, the von Mises stress distribution in the cylinder becomes

$$\sigma_{vm}^2 = (a_{mn}^2 + b_{mn}^2 - a_{mn} b_{mn}) \sin^2(\alpha_{m} y) \sin^2(\beta_n x) + 3 c_{mn}^2 \cos^2(\alpha_{m} y) \cos^2(\beta_n x).$$ (29)
Note that the maximum value of the von Mises stress in the cylinder is

\[
\max \sigma_{vm} = \max \left\{ \sqrt{3} c_{mn}, \sqrt{a_{mn}^2 + b_{mn}^2 - a_{mn} b_{mn}}, \sqrt{\frac{3c_{mn}^2(a_{mn}^2 + b_{mn}^2 - a_{mn} b_{mn})}{a_{mn}^2 + b_{mn}^2 - a_{mn} b_{mn} + 3c_{mn}^2}} \right\}.
\]  

(30)

The \( p \)-norm can be evaluated if \( p = 2k \) is an even integer. The \( p \)-norm of the von Mises stress can be expressed in the following binomial expansion:

\[
||\sigma_{vm}||_p = \left( \sum_{i=0}^{k} \binom{k}{i} \text{sc}(2i, 2(k-i))(a_{mn}^2 + b_{mn}^2 - a_{mn} b_{mn})^i (3c_{mn}^2)^{k-i} \right)^{\frac{1}{p}},
\]  

(31)

where \( \text{sc}(p,q) \) is the value of the following integral:

\[
\text{sc}(p,q) = \int_{\Omega} \sin^p(\alpha_m y) \sin^p(\beta_n x) \cos^q(\alpha_m y) \cos^q(\beta_n x) \, d\Omega
\]

\[
= \int_{0}^{L} \sin^p(\beta_n x) \cos^q(\beta_n x) \, dx \int_{0}^{2\pi a} \sin^p(\alpha_m y) \cos^q(\alpha_m y) \, dy.
\]  

(32)

The value of the integral \( \text{sc}(p,q) \) is

\[
\text{sc}(p,q) = \begin{cases} 
\frac{A}{\prod_{k=1}^{p/2} \left( \frac{2k-1}{q+2k} \right)^{2} \prod_{k=1}^{q/2} \left( \frac{2k-1}{2k} \right)^{2}} & \text{p and q even} \\
\frac{2}{\alpha_m \beta_n(q+1)} \prod_{k=1}^{(p-1)/2} \left( \frac{p-2k-1}{p+q-2k} \right)^{2} & \text{p odd q even}
\end{cases}
\]  

(33)

where \( A = 2\pi a L \).

The Tsai–Wu failure criterion is more appropriate than the von Mises stress for specially orthotropic laminates. The Tsai–Wu failure criterion is

\[
F_{TW}(\sigma) = F_1 \sigma_1 + F_2 \sigma_2 + F_{11} \sigma_1^2 + 2F_{12} \sigma_1 \sigma_2 + F_{22} \sigma_2^2 + F_{66} \sigma_{12}^2 \leq 1,
\]  

(34)

where the coefficients \( F_\ast \) are selected such that the boundary is a closed convex surface.

In this case, the \( p \)-norm can be expressed as follows:

\[
||F_{TW}||_p = \left( \sum_{i+j+k=p} \binom{p}{i,j,k} (f_{mn})^i (g_{mn})^j (h_{mn})^k \text{sc}(i+2j,2k) \right)^{\frac{1}{p}},
\]  

(35)

where the coefficients \( f_{mn}, g_{mn}, \) and \( h_{mn} \) are defined as follows:

\[
f_{mn} = F_1 a_{mn} + F_2 b_{mn}
\]

\[
g_{mn} = F_{11} a_{mn}^2 + 2F_{12} a_{mn} b_{mn} + F_{22} b_{mn}^2
\]

\[
h_{mn} = F_{66} c_{mn}^2.
\]
Algorithm 1: Factorization of a matrix stored in a block-cyclic data format

Given $S$, compute $L$ and $U$ such that $S = LU$;

for $i = 1$ to $n$ do

Diagonal update

if is_block_owner($i, i$) then
  compute $L_{ii}U_{ii} = S_{ii}$;
  send $U_{ii}$ to each processor in column $i$;
  send $L_{ii}$ to each processor in row $i$;

Column update

if processor in column $i$ then
  receive $U_{ii}$;
  for $j = i + 1$ to $n$ do
    if is_block_owner($j, i$) then
      compute $L_{ji} = S_{ji}U_{ii}^{-1}$ and copy $L_{ji}$ to row_buffer;
      send row_buffer to the row processors;
    else
      receive row_buffer;

Row update

if processor in row $i$ then
  receive $L_{ii}$;
  for $j = i + 1$ to $n$ do
    if is_block_owner($i, j$) then
      compute $U_{ij} = L_{ii}^{-1}S_{ij}$ and copy $U_{ij}$ to column_buffer;
      send column_buffer to the column processors;
    else
      receive column_buffer;

GEMM update

for $j = i + 1$ to $n$ do
  for $k = i + 1$ to $n$ do
    if is_block_owner($j, k$) then
      extract $L_{ji}$ from column_buffer and $U_{ik}$ from row_buffer;
      compute $S_{jk} \leftarrow S_{jk} - L_{ji}U_{ik}$;

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_{11}$</td>
<td>164 GPa</td>
</tr>
<tr>
<td>$E_{22}$</td>
<td>8.3 GPa</td>
</tr>
<tr>
<td>$G_{12}, G_{13}$</td>
<td>21.0 GPa</td>
</tr>
<tr>
<td>$G_{23}$</td>
<td>12 GPa</td>
</tr>
<tr>
<td>$\nu_{12}$</td>
<td>0.34</td>
</tr>
<tr>
<td>$t_{\text{ply}}$</td>
<td>0.125 mm</td>
</tr>
</tbody>
</table>

Table 1: Representative IM7/3501-6 stiffness and strength properties.