An Automated Approach for Parallel Adjoint-Based Error Estimation and Mesh Adaptation

Brian N. Granzow[a,*], Assad A. Oberai[b], Mark S. Shephard[a]

[a] *Scientific Computation Research Center*
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY 12180
[b] *Aerospace and Mechanical Engineering*
University of Southern California,
Los Angeles, CA 90089

Abstract

In finite element simulations, not all of the data is of equal importance. In fact, the primary purpose of a numerical study is often to accurately assess only one or two engineering ouptut quantities that can be expressed as functionals. Adjoint-based error estimation provides a means to approximate the discretization error in functional quantities and mesh adaptation provides the ability to control this discretization error by locally modifying the finite element mesh. In the past, adjoint-based error estimation has only been accessible to expert practioners in the field of solid mechanics. In this work, we present an approach to automate the process of adjoint-based error estimation and mesh adaptation on parallel machines. This process is intended to lower the barrier of entry to adjoint-based error estimation and mesh adaptation for solid mechanics practitioners. We demonstrate that this approach is effective for example problems in Poisson's equation, nonlinear elasticity, and thermomechanical elastoplasticity.

Keywords: automated, adjoint, a posteriori, error estimation, adaptation, finite element

1. Introduction

Numerical simulation has become ubiquitous in engineering and scientific practice due to the continuing increase in power and accessibility of computational resources. For scientific and engineering applications, ensuring the accuracy and reliability of the computed numerical solution is of primary importance. For example, it is often necessary to design structural components for which the von-Mises stress in the component's domain is less than a given material yield strength when subjected to a variety of loading conditions.

In the context of finite element methods, *a posteriori* error estimation and mesh adaptation provide useful tools for approximating and controlling the discretization error when solving partial differential equations (PDEs) (*cf.* [1, 25, 60]).

In an abstract finite element setting, one seeks to solve the variational problem: find $u \in \mathcal{S}$ such that

$$\mathcal{R}(w;u) = 0 \quad \forall w \in \mathcal{V}, \tag{1}$$

where $\mathcal{R} : \mathcal{S} \times \mathcal{V} \to \mathbb{R}$ is a semilinear form, linear in its first argument and potentially nonlinear in its second, and $\mathcal{S}$ and $\mathcal{V}$ are Hilbert spaces. Given the exact solution u to this problem and a corresponding finite element approximation u^H, the discretization error is defined as $e := u - u^H$. Traditional *a posteriori* error estimates attempt to bound or approximate the discretization error in a global norm $\|e\|$, such as the energy norm induced by the underlying partial differential operator. In the past twenty years, though, adjoint-based

*Corresponding author, brian.granzow@gmail.com

techniques have been developed and utilized to obtain approximations η or approximate bounds $\hat{\eta}$ for the error $|J(u) - J(u^H)|$ in some physically meaningful functional $J : \mathcal{S} \to \mathbb{R}$, such that

$$|J(u) - J(u^H)| \approx \eta < \hat{\eta}. \tag{2}$$

The use of adjoint (or duality) techniques for *a posteriori* error estimation can be traced back to Babuška and Miller [3, 4, 5] who explored the post-processing of functional quantities. These ideas were then expanded to the context of adaptive finite element methods by Johnson et al. [15]. Becker and Rannacher [8] developed an approach to functional *a posteriori* error estimation for Galerkin finite element methods called the *dual weighted residual* method. Giles and Pierce [20] developed an approach conceptually similar to the dual weighted residual method, but additionally concerned themselves with discretizations that lack Galerkin orthogonality, such as Godunov finite volume methods. Venditti and Darmofal [57, 58, 59] developed adjoint-based error estimates for arbitrary discretizations using discrete adjoint equations based on two-level discretization schemes. Prudhomme and Oden [41, 36] developed adjoint techniques to determine guaranteed upper and lower bounds for linear functionals in the context of linear variational problems.

Adjoint-based error estimation and mesh adaptation have been heavily adopted by the computational fluid dynamics (CFD) community [17]. This can in part be explained by the fact that the current increase in computing power alone may not be sufficient to accurately resolve quantities of interest (QoIs) in CFD applications, even when very finely generated *a priori* meshes are used [17]. However, despite its popularity in CFD, adjoint-based error estimation is not regularly applied to solid mechanics applications.

In the context of solid mechanics, adaptive adjoint-based error estimation has been used to study linear elasticity in two [44, 54, 21] and three [19] dimensions, two [45, 46] and three [18] dimensional elasto-plasticity, two dimensional thermoelasticity [42], two dimensional nonlinear elasticity [31], and two dimensional hyperelasticity [61]. We remark that in the majority of this literature, mesh adaptation is performed with structured adaptive mesh refinement using quadrilateral or hexahedral elements and without any discussion of parallelization. Additionally, none of these investigations apply the current state of the art capabilities of parallel three-dimensional unstructured mesh adaptation, as is currently the norm for CFD applications.

Perhaps one reason for this discrepancy is the fact that adjoint-based error estimation requires the development and implementation of a number of non-trivial steps. From a high-level, the following steps must be carried out to perform an adaptive adjoint-based error analysis:

1. Solve the original (primal) PDE of interest.
2. Using the primal solution, derive and solve an auxiliary adjoint PDE.
3. Enrich the solution to the adjoint PDE in some manner.
4. Compute error estimates using an adjoint-weighted residual method.
5. Localize the error to contributions at the mesh entity level.
6. Adapt the mesh based on the local error contributions.

Additionally, modern solid mechanics applications necessitate the use of parallel analysis, meaning each of these steps must execute effectively and efficiently on parallel machines.

Further, both the primal residual and functional QoI can be highly nonlinear in solid mechanics. Solving the primal and adjoint problems requires the linearization of the primal residual and the functional QoI with respect to the degrees of freedom of the problem. The derivation and numerical implementation of these linearizations can be time consuming and error-prone. As a result, adjoint-based error control has typically remained a tool for expert analysis with a high barrier of entry.

As a final observation, we note that unstructured mesh generation and adaptation is robust, reliable, and scalable for simplical elements. However, for solid mechanics applications with incompressibility constraints, such as isochoric plasticity or incompressible hyperelasticity, standard displacement-based Galerkin finite element discretizations are known to be unstable when using simplical elements. This fact may have further hindered the adoption of unstructured mesh adaptation for previous adaptive adjoint-based solid mechanics applications.

To make adjoint-based error estimation and mesh adaptation more accessible to solid mechanics practitioners, we seek to fully automate its steps for execution on parallel machines. Specifically, we seek to

develop software that automates steps 1-6 outlined above based solely on the inputs of a semilinear form $\mathcal{R}$ and a functional QoI J. We endeavor to develop this software to be applicable to both Galerkin and stabilized finite element methods.

Recently, Rognes and Logg [49] introduced a fully automated approach to goal-oriented error estimation and mesh adaptation for Galerkin finite element methods within the FEniCS [34] finite element framework. In this approach, the adjoint problem is derived in a discrete manner based on a user-implemented residual $\mathcal{R}$ and a functional J. The adjoint problem is then solved on the same finite element space as used for the primal problem and enriched to a higher order polynomial space by solving local patch-wise problems. Based on the given semilinear form $\mathcal{R}$, error contributions are then localized to the element-level by solving local element problems to recover the strong form of the residual operator over element interiors and element boundaries. The total error in the functional QoI is then computed as the sum of these error contributions. As a final step, the mesh is adapted using conforming unstructured mesh *refinement*.

In this work, we present an approach for automating goal-oriented analysis that is distinct in several ways. First, we consider adjoint-based error estimation in the context of both Galerkin and stabilized finite element methods. Additionally, we propose solving the adjoint problem in a richer finite element space, obtained via uniform refinement, than the space used for the primal problem. To localize the error to the mesh entity level, we utilize a partition of unity based approach proposed by Richter and Wick [48]. This allows us to directly re-use the implemented semilinear form $\mathcal{R}$ for error localization, eliminating the need to solve local element problems to recover the strong form residual. We also take advantage of fully unstructured conforming mesh adaptation, where the mesh can be *coarsened* as well as refined. As a final distinction, we highlight the ability of the proposed approach to execute on parallel machines.

In totality, our new approach can be described as follows. First, the primal problem is solved via Newton's method, where the Jacobian of the semilinear form $\mathcal{R}$ is obtained via automatic differentiation. The adjoint problem is derived in a discrete manner in a richer finite element space obtained by a uniform refinement of the initial mesh. The adjoint operator is derived by an application of automatic differentiation to the semilinear form $\mathcal{R}$, and the right-hand side of the adjoint problem is obtained by applying automatic differentiation to the functional J. An approximate error η is computed as a modified discrete adjoint weighted residual evaluated on the fine space. The error is then localized to mesh vertices using a variational localization approach by introducing a partition of unity into the weighting slot of the semilinear form $\mathcal{R}$. An approximate upper bound $\hat{\eta}$ is obtained by summing the absolute values of localized error contributions. Finally, the mesh is adapted by specifying a *mesh size field*, which defines the length of mesh edges over the mesh. We have implemented this approach in a `C++` finite element application which we have called Goal [22].

Underlying this approach is the concept of *template-based generic programming* (TBGP) [37, 38], which has previously been used to automate the solution of PDEs as well as embedded advanced analysis features, such as sensitivity analysis and uncertainty quantification. From a high-level the TBGP approach consists of a *gather* phase, a *compute* phase, and a *scatter* phase. The present work extends the TBGP approach to include the automation of error localization, as required to drive mesh adaptation.

The remainder of this paper is outlined as follows. First adjoint-based error estimation is reviewed for abstract Galerkin and stabilized variational problems. Next, a description of the software components utilized in this work is provided. In particular, each step of the automated adjoint-based analysis is discussed with respect to its utilized software components. A review of the concept of TBGP is then provided and its extension for the purposes of adjoint-based error estimation is discussed. A detailed description of each step in the adaptive adjoint-based process is then described. First the automated solution of the primal problem is discussed. Then the automated derivation and solution of the adjoint problem is described. Next, the automation of error localization to drive mesh adaptation is outlined and mesh adaptation procedures are discussed. The implementation of several QoIs in the Goal application is reviewed. Finally, the effectiveness of the proposed automated approach is demonstrated for several applications.

2. A Review of Adjoint-Based Error Representations

In this section, a brief review of the derivation of adjoint-based error representations is provided for Galerkin finite element methods as outlined by Becker and Rannacher [8], and for stabilized finite element methods as outlined by Cyr et al. [12]. This review is inteded to give context and serve as a road map for the remaining sections in this chapter.

Let $\mathcal{S}$ and $\mathcal{V}$ be Hilbert spaces, $\mathcal{R}_g : \mathcal{S} \to \mathcal{V}$ and $\mathcal{R}_\tau : \mathcal{S} \to \mathcal{V}$ be semilinear forms that are linear in their first argument and potentially nonlinear in their second argument. Let $\mathcal{S}^H \subset \mathcal{S}$ and $\mathcal{V}^H \subset \mathcal{V}$ be classical finite element function spaces, where H is a mesh-dependent parameter that denotes the fineness of the discretization. We introduce the following variational abstract model problem: find $u \in \mathcal{S}$ such that

$$\mathcal{R}_g(w; u) = 0 \quad \forall w \in \mathcal{V}. \tag{3}$$

Similarly, we introduce the following abstract *adjoint problem*: find $z \in \mathcal{V}$ such that

$$\mathcal{R}_g'[u^H](w, z) = J'[u^H](w) \quad \forall w \in \mathcal{V}, \tag{4}$$

where the prime indicates Fréchet linearization about the argument in square brackets, which can equivalently be expressed as the Gâteaux derivatives

$$J'[u^H](w) := \frac{d}{d\epsilon} J(u^H + \epsilon w)\big|_{\epsilon=0}, \tag{5}$$

and

$$\mathcal{R}_g'[u^H](w, z) := \frac{d}{d\epsilon} \mathcal{R}_g(z; u^H + \epsilon w)\big|_{\epsilon=0}. \tag{6}$$

Here, $u^H \in \mathcal{S}^H$ denotes some finite element approximation to the true solution u. The purpose of the adjoint problem is to relate the original problem of interest to the functional quantity J, and it is this relationship that allows us to derive adjoint-based error representations. Further, we note that the adjoint solution z can be interpreted as the sensitivity of the QoI to perturbations in the primal PDE residual [17].

2.1. Galerkin Finite Element Methods

The corresponding Galerkin finite element formulation of the abstract problem (3) can be stated as: find $u^H \in \mathcal{S}^H$ such that

$$\mathcal{R}_g(w^H; u^H) = 0 \quad \forall w^H \in \mathcal{V}^H. \tag{7}$$

Let $e := u - u^H$ denote the discretization error. We can then derive an error representation for the functional J in terms of the adjoint solution z as follows:

$$\begin{aligned} J(u) - J(u^H) &= J'[u^H](e) + \mathcal{O}(e^2) \\ &= \mathcal{R}_g'[u^H](e, z) + \mathcal{O}(e^2) \\ &= \mathcal{R}_g(z; u) - \mathcal{R}_g(z; u^H) + \mathcal{O}(e^2) \\ &= -\mathcal{R}_g(z; u^H) + \mathcal{O}(e^2) \\ &= -\mathcal{R}_g(z - z^H; u^H) + \mathcal{O}(e^2). \end{aligned} \tag{8}$$

Here, the first equality is due to the linearization [8] of the functional J, the second equality is due to the introduced adjoint problem (4), the third equality is due to the linearization [8] of the residual $\mathcal{R}_g$, the fourth equality holds due to the definition of the abstract primal problem (3), and the fifth equality is due to Galerkin orthogonality, where z^H denotes the interpolant of z onto the space $\mathcal{V}^H$. In reference to the notation introduced, the total residual semilinear form $\mathcal{R}$ is given as $\mathcal{R} = \mathcal{R}_g$.

2.2. Stabilized Finite Element Methods

A corresponding stabilized finite element method of the abstract problem (3) can be expressed as: find $u^H \in \mathcal{S}^H$ such that

$$\mathcal{R}_g(w^H; u^H) + \mathcal{R}_\tau(w^H; u^H) = 0 \quad \forall w^H \in \mathcal{V}^H. \tag{9}$$

Here $\mathcal{R}_\tau$ denotes a consistent *stabilization residual* that adds stability to the numerical scheme. We say that the stabilization is *consistent* if $\mathcal{R}_\tau(w^H; u) \to 0$ as $H \to 0$.

Again, we let $e := u - u^H$ denote the discretization error, and derive an error representation for the functional J as follows

$$\begin{aligned} J(u) - J(u^H) &= J'[u^H](e) + \mathcal{O}(e^2) \\ &= \mathcal{R}_g'[u^H](e, z) + \mathcal{O}(e^2) \\ &= \mathcal{R}_g(z; u) - \mathcal{R}_g(z; u^H) + \mathcal{O}(e^2) \\ &= -\mathcal{R}_g(z; u^H) + \mathcal{O}(e^2) \\ &= -\mathcal{R}_g(z - z^H; u^H) + \mathcal{R}_\tau(z^H; u^H) + \mathcal{O}(e^2). \end{aligned} \tag{10}$$

Here, the first four equalities are obtained exactly as in the corresponding Galerkin finite element method. However, when we subtract the interpolant z^H of the adjoint solution z in the fifth equality, an additional term remains because the numerical scheme (9) lacks Galerkin orthogonality. In reference to the notation introduced, the total semilinear form $\mathcal{R}$ is given as $\mathcal{R} = \mathcal{R}_g + \mathcal{R}_\tau$.

3. Software Components

An adaptive adjoint-based simulation requires the implementation and coordination of a number of non trivial components. Namely, the solution of a primal problem, the construction and solution of an auxiliary adjoint problem, an enrichment of the adjoint solution, the estimation and localization of the error, and mesh adaptation are all required steps in the adjoint-based adaptive process.

To implement each of these components for effective execution on parallel machines, we make use of two state of the art software suites. The first is PUMI [30], which contains tools to support unstructured mesh services on massively parallel machines. In particular, PUMI provides all of the necessary machinery to store, query, adapt, and dynamically load balance parallel unstructured meshes via a collection of modern `C` and `C++` libraries. The second is Trilinos [27, 28], which provides a large variety of `C++` packages to support multiphysics simulations on parallel machines. In particular, Trilinos provides the ability to store and solve sparse parallel linear systems, as well as tools to perform automatic differentiation.

Using these two software suites as building blocks, we have written a new `C++` application for adjoint-based error estimation and mesh adaptation with an emphasis on nonlinear solid mechanics. We have called this application Goal [22]. Below, we describe how these software components are utilized for each portion of the adaptive adjoint-based process, where the analysis is automated based only on the inputs of a semilinear form $\mathcal{R}$ and a functional QoI J.

3.1. The Primal Problem

Based on an implemented weighted residual operator $\mathcal{R}$, the Goal application computes element-level residual vectors and element-level Jacobian matrices. The element-level Jacobian matrices are computed via automatic differentiation using the Trilinos library Sacado. Sacado provides efficient automatic differentiation using a `C++` meta-programming technique called expression templates [39].

After the computation of a single element's residual vector and Jacobian matrix, the Goal application performs a finite element assembly step to sum contributions to the global residual vector and global Jacobian matrix. To store and modify the global linear algebra objects, we utilize the Tpetra library provided by Trilinos. In particular, the Jacobian matrix is stored as a sparse compressed row storage matrix in parallel.

The primal problem is solved via Newton's method, which requires iterative evaluations of the global residual vector and Jacobian matrix. For each Newton iteration, a global linear system must be solved. We solve this linear system iteratively in parallel using either a CG or GMRES solver provided the Trilinos library Belos [7]. Additionally, we perform algebraic multigrid preconditioning using the Trilinos library MueLu [40].

Once the primal problem has been solved, we utilize the PUMI library APF to store the finite element solution information at nodes and if necessary secondary solution information at integration points. Additionally, the APF library is used to provide shape function information and to query stored solution information during residual and Jacobian evaluations. Throughout the entire solution process, the PUMI mesh data structure is utilized to query mesh specific information.

3.2. The Adjoint Problem

To solve the adjoint problem in a richer finite element space than the one used for the primal problem, we make use of the underlying PUMI mesh data structure [29] and the PUMI MeshAdapt software to create and store a uniformly refined nested mesh with parent-child relations back to the original mesh. This relational information is implemented in the Goal application, as it falls outside of the normal intended use case of the MeshAdapt software, which concerns itself with fully unstructured conforming mesh adaptation via edge splits, swaps, and collapses. However, the flexibility of the PUMI software allows us to additionally construct data structures similar to those used in traditional adaptive mesh refinement (AMR) with little implementation effort. Using the APF library and the parent-child relational information, we are able to interrogate stored solution fields on both the parent and nested meshes, which is required during the assembly of the adjoint problem.

On this finer mesh, element-level Jacobian matrices are computed using the Sacado library, based on the Goal implementation of the operator $\mathcal{R}$. Additionally, element-level derivatives of the functional quantity of interest J are computed with respect to degrees of freedom of the problem, resulting in an element-level functional derivative vector.

After the computation of a single element's Jacobian matrix and functional derivative vector, the Goal application performs a finite element assembly step to sum contributions to the global discrete adjoint matrix and the global functional derivative vector. Like the primal problem, these global parallel linear objects are stored using the Tpetra library. The global discrete adjoint operator and functional derivative vector fully define the linearized adjoint problem, which we again precondition with algebraic multigrid techniques using the MueLu library and solve using either CG or GMRES iterations using the Belos library. The fine-space adjoint solution is then attached to the mesh using the APF library.

3.3. Error Estimation and Localization

The error estimation and localization routines are implemented entirely in the Goal application. The error is localized by an evaluation of the stabilized weighted residual operator $\mathcal{R}$, where the weight is chosen to be the adjoint solution multiplied by a partition of unity. This error localization is discussed in further detail in Section 7. Based on these element-level residual vectors, Goal performs finite element assembly of the global residual vector, which is stored as a Tpetra vector. This vector represents an adjoint-weighted residual error estimate at each mesh vertex for each PDE equation in the fine mesh. The error is attached to the vertices of the fine mesh using the APF library.

3.4. Mesh Adaptation

Once the error is stored on the vertices of the fine mesh, we interpolate it to element centers of the coarse mesh. The fine mesh data structures are then destroyed and the Goal application computes a mesh size field that seeks to equidistribute the error for an output mesh with N elements. This mesh size field is given as the input to the PUMI MeshAdapt software, which adapts the mesh with a sequence of edge splits, swaps, and collapses [32, 2] to satisfy the given mesh size field. As a final step, we utilize the PUMI library ParMA [53, 13] to perform diffusive load balancing to ensure parallel partitioning quality.

3.5. In-Memory Integration of Components

The coupling of the software components described above is done *in-memory* [52]. That is, there is no file-based communication of data from one analysis component to the next in the automated process. This in-memory coupling is a key ingredient for parallel analysis, where filesystem bandwidth is a critical bottleneck.

4. Template-Based Generic Programming

In this section, we provide a review of the concept of *template-based generic programming* for the evaluation and solution of PDEs [37, 38] and how it has been extended in the Goal application to automate the process of adjoint-based error estimation and mesh adaptation. For PDE applications, TBGP is broken into three major components, a *seed* or *gather* phase, a *compute* phase, and a *scatter* phase. The seed and scatter operations must be programmed specifically for each evaluation purpose. In contrast, the compute phase, where the PDE and QoI expressions are implemented, are written in a totally generic manner. Figure 1 pictorially represents this design philosophy.

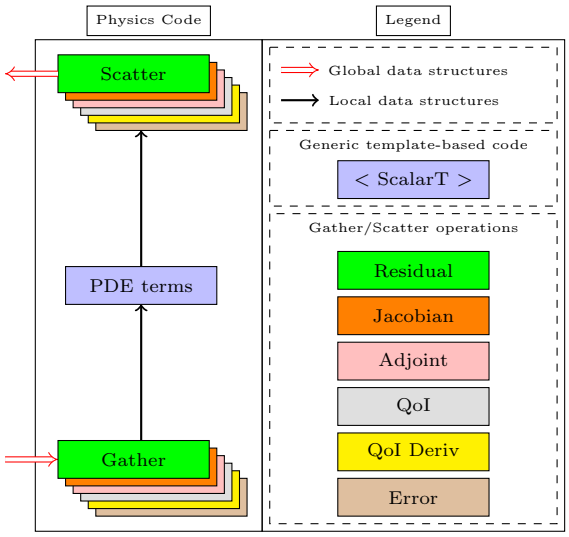

Figure 1: A schematic for the generic programming model of PDEs.

We invoke this approach at the element-level, meaning for each element we perform the process: Gather → Compute → Scatter. By doing so, we reduce memory overhead and eliminate complications introduced by parallel computation [38]. Underlying the TBGP approach is the use of forward automatic differentiation (FAD) [26]. The Goal application utilizes the Trilinos library Sacado [39] to perform automatic differentiation.

The purpose of the gather/seed operation is to collect information from global storage containers and 'gather' it to local element-level data structures. Further, any FAD derivative information is *seeded* during this operation, if necessary. For each evaluation type, the gather/seed operation initializes an array that physically represent the degrees of freedom associated with the current element, and initializes FAD variables' derivative arrays to physically represent derivatives with respect to the degrees of freedom associated with the current element, when necessary. This degree of freedom array is templated on a scalar type `ScalarT`. For the Residual, QoI, and Error evaluation types, this scalar type corresponds to a `C++` double. For the remaining evaluation types, this scalar type corresponds to a `Sacado::FAD` forward automatic differentiation variable type.

The compute phase computes local contributions to the equations or expressions of interest at the element level in terms of the degrees of freedom, as collected by the gather operation. The code for the compute

Evaluation Type	Scalar Type	Input	Output
Residual	`double`	$\boldsymbol{u}^H, \boldsymbol{s}^H$	$\boldsymbol{R}^H$
Jacobian	`Sacado::FAD`	$\boldsymbol{u}^H, \boldsymbol{s}^H$	$\frac{\partial \boldsymbol{R}^H}{\partial \boldsymbol{u}^H}$
Adjoint	`Sacado::FAD`	$\boldsymbol{u}^h_H, \boldsymbol{s}^h_H$	$\left[\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\Big\vert_{\boldsymbol{u}^h_H}\right]^T$
QoI	`double`	$\boldsymbol{u}^H, \boldsymbol{s}^H$	J^H
QoI Deriv	`Sacado::FAD`	$\boldsymbol{u}^h_H, \boldsymbol{s}^h_H$	$\left[\frac{\partial J^h}{\partial \boldsymbol{u}^h}\Big\vert_{\boldsymbol{u}^h_H}\right]^T$
Error	`double`	$\boldsymbol{u}^h_H, \boldsymbol{s}^h_H, \boldsymbol{z}^h$	$\boldsymbol{R}^h$

Table 1: A list of TBGP evaluation operations used in the Goal application. In this table $\boldsymbol{u}^H$ is the primal solution vector, $\boldsymbol{u}^h_H$ is the prolongation of the solution vector to a richer space, $\boldsymbol{s}^H$ is a (potentially empty) vector of history-dependent mechanics state variables, $\boldsymbol{s}^h_H$ is the prolongation of the state to a richer space, $\boldsymbol{z}^h$ is the adjoint solution vector, $\boldsymbol{R}^H$ is the residual vector evaluated on the coarse space, $\boldsymbol{R}^h$ is the residual vector evaluated on the fine space, and J^H is the scalar QoI.

Listing 1: The abstract Goal integrator class interface.

```
1  class Integrator {
2    public:
3      Integrator();
4      virtual ~Integrator();
5      std::string const& get_name() { return name; }
6      virtual void set_time(double, double) {}
7      virtual void pre_process(SolInfo*) {}
8      virtual void set_elem_set(int) {}
9      virtual void gather(apf::MeshElement*) {}
10     virtual void in_elem(apf::MeshElement*) {}
11     virtual void at_point(apf::Vector3 const&, double, double) {}
12     virtual void out_elem() {}
13     virtual void scatter(SolInfo*) {}
14     virtual void post_process(SolInfo*) {}
15   protected:
16     std::string name;
```

phase is written in an entirely generic fashion, and is templated on a scalar type `ScalarT`. Templating the code used for the compute phase, along with appropriately chosen gather and scatter operations, allows the same code to be re-used for the distinct evaluation purposes listed in Table 1.

The scatter phase takes the local element-level data evaluated in the compute phase and 'scatters' it to the appropriate global data structure, as determined by the current evaluation operation. For instance, for the residual evaluation operation, local element-level residuals are evaluated in the compute phase and then summed into appropriate locations in the global residual vector during the scatter phase. Similarly, for the Jacobian evaluation operations, local element-level Jacobians are evaluated in the compute phase and the scatter opeation sums these local contributions to appropriate locations in the global Jacobian matrix.

In the Goal application, we have considered six specific gather/scatter evaluation operations corresponding to the evaluation of the global residual vector, evaluation of the global Jacobian matrix, evaluation of the adjoint of the global Jacobian matrix, evaluation of the functional QoI, evaluation of the derivative of the functional QoI, and evaluation of localized adjoint-weighted residual error estimates. Table 1 lists the inputs and outputs for these specific evaluation operations.

To realize these specific gather/scatter operations, we have implemented an abstract degree of freedom class and an abstract quantity of interest class that are both templated on a scalar type `ScalarT`. This scalar type is explicitly instantiated to either be a `C++` double or a `Sacado::FAD` forward automatic differentiation variable type. Both the degree of freedom and QoI classes are equipped with `gather` and `scatter` methods, whose behavior changes based on an input parameter given to the class constructor. For the degree of

freedom class, this parameter selects gather/scatter operations for either the residual, Jacobian, adjoint Jacobian, or adjoint-weighted residual error evaluations. Similarly, for the QoI class, this input parameter selects gather/scatter operations for either the evaluation of the QoI or the derivative of the QoI with respect to the problem degrees of freedom.

Previously, TBGP has been utilized in the multiphysics code Albany [50, 56] with the capability to perform the Residual, Jacobian, Adjoint, QoI, and QoI derivative evaluation operations shown in Table 1. To extend the abilities of TBGP to include adjoint-based error estimation, the Goal application implements the ability to perform the Adjoint and QoI derivative evaluations in a richer finite element space, as discussed in Section 6, a feature not previously available in existing TBGP codes. Further, the Goal application implements a novel evaluation type for the localization of the error, referred to as the Error evaluation type in Table 1. For this purpose, we have implemented an abstract weighting function class whose behavior changes based on the chosen evaluation type. This class evaluates the appropriate finite element weighting function values and gradients based on linear Lagrange basis functions for the Residual, Jacobian and Adjoint evaluation types. However, for the Error evaluation type, the behavior of the weighting function class is modified such that it evaluates the value and gradient of the adjoint solution z^h multiplied by a partition of unity. This abstraction of the weighting function class allows us to re-use the PDE implementation of the semilinear form $\mathcal{R}$ to assemble a residual vector $\boldsymbol{R}^h$ that represents an adjoint-weighted residual error estimate at each mesh vertex for each PDE equation in the richer finite element space, which is then used to drive mesh adaptation.

Listing 1 demonstrates the abstract integrator interface that has been implemented in the Goal application. The abstract degree of freedom, QoI, and weighting function classes inherit from this base class. For each of these classes, the `gather` and `scatter` methods are implemented specifically for each appropriate evaluation type. The PDE equations in the Goal application are written as a combination of `Goal::Integrator`s. Given an ordered array of integrators, the Goal application performs finite element assembly for every evaluation type in a generic manner, as outlined by Algorithm 1.

Algorithm 1 Assembly algorithm used in the Goal application

Given a mesh M and an ordered array of integrators I:
Call `pre_process` for each integrator i in I.
for each element set es in mesh M **do**
  Call `set_elem_set` for each integrator i in I.
  for each element e in element set es **do**
    Call `gather` for each integrator i in I.
    Call `in_elem` for each integrator i in I.
    for each integration point ip in element e **do**
      Call `at_point` for each integrator i in I.
    end for
    Call `out_elem` for each integrator i in I.
    Call `scatter` for each integrator i in I.
  end for
end for
Call `post_process` for each integrator i in I.

5. The Primal Problem

5.1. Galerkin Finite Element Methods

We recall the definition of the abstract Galerkin finite element model problem, given by equation (7). In this context, the weighted residual form $\mathcal{R}_g$ is implemented in the Goal application. As an example, Listing 2 demonstrates the implementation of the Poisson residual $\mathcal{R}_g(w;u) := (\nabla w, \nabla u) - (w, f)$ in the Goal application.

Listing 2: Poisson residual.

```
template <typename ScalarT>
void Residual<ScalarT>::at_point(
    apf::Vector3 const& p, double ipw, double dv) {
  apf::Vector3 x(0,0,0);
  apf::mapLocalToGlobal(elem, p, x);
  double fval = eval(f, x[0], x[1], x[2], 0.0);
  for (int n = 0; n < u->get_num_nodes(); ++n)
  for (int i = 0; i < num_dims; ++i)
    u->resid(n) += u->grad(i) * w->grad(n, i) * ipw * dv;
  for (int n = 0; n < u->get_num_nodes(); ++n)
    u->resid(n) -= fval * w->val(n) * ipw * dv;
}
```

Listing 3: Pressure stabilization residual for mechanics.

```
template <typename ScalarT>
void Stabilization<ScalarT>::at_point(
    apf::Vector3 const&, double ipw, double dv) {
  double h = get_size(mesh, elem);
  double tau = 0.5*c0*h*h/mu;
  auto J = k->get_det_def_grad();
  auto F = k->get_def_grad();
  auto Cinv = inverse(transpose(F)*F);
  for (int n = 0; n < p->get_num_nodes(); ++n)
  for (int i = 0; i < num_dims; ++i)
  for (int j = 0; j < num_dims; ++j)
    p->resid(n) += tau * J * Cinv(i, j) *
      p->grad(i) * w->grad(n, j) * ipw * dv;
}
```

5.2. Stabilized Finite Element Methods

We recall the definition of the abstract stabilized finite element model problem, given by equation (9). In this context, both the weighted residual statement $\mathcal{R}_g$ and the stabilized weighted residual form $\mathcal{R}_\tau$ are implemented in the Goal application. As an example, Listing 3 demonstrates the implementation of the pressure stabilization [43] residual $\mathcal{R}_\tau(w;u)$ term used in the Goal application for finite deformation solid mechanics. This stabilization term is discussed in greater detail in Section 10.2.

5.3. Automated Solution Based on Residual Implementation

For each element, we compute element level Jacobian matrices by applying automatic differentiation [26] to element-level contributions to the residual vector. For example, Listing 2 demonstrates how contributions to the element-level Poisson's equation residual $\mathcal{R}(w;u) = (\nabla w, \nabla u) - (w, f)$ are implemented. The element level Jacobian matrices are then assembled into the global system Jacobian operator $\boldsymbol{\mathcal{J}}^H \in \mathbb{R}^{N\times N}$, given by

$$\boldsymbol{\mathcal{J}}^H = \frac{\partial \boldsymbol{R}^H(\boldsymbol{u}^H)}{\partial \boldsymbol{u}^H} \tag{11}$$

Listings 2 and 3 both demonstrate how element-level contributions to the semilinear forms $\mathcal{R}_g$ and $\mathcal{R}_\tau$, respectively, are computed in the Goal application. Notice that this code is templated on a scalar type `ScalarT`. When the scalar type is chosen as a `C++ double`, element-level contributions to the residual vector $\boldsymbol{R}^H$ are computed. When the scalar type is chosen as a Sacado forward automatic differentiation variable, element-level contributions to the Jacobian matrix $\boldsymbol{\mathcal{J}}^H$ are computed. This illustrates a key concept of template-based generic programming, in that the governing equations need only be implemented once to compute a variety of additional information.

With the ability to fully assemble the Jacobian matrix $\boldsymbol{\mathcal{J}}^H$ and the residual vector $\boldsymbol{R}^H$, we solve the

governing equations with Newton's method, where we iterate over the steps

$$\begin{aligned} \boldsymbol{\mathcal{J}}^H(\boldsymbol{u}_k^H)\,\delta\boldsymbol{u}_k^H &= -\boldsymbol{R}^H(\boldsymbol{u}_k^H) \\ \boldsymbol{u}_{k+1}^H &= \boldsymbol{u}_k^H + \delta\boldsymbol{u}_k^H, \end{aligned} \tag{12}$$

unitl the convergence criterion $\|\boldsymbol{R}^H(\boldsymbol{u}^H)\|_2 < \epsilon$ is met for a user-specified tolerance ϵ. Here $\boldsymbol{u}_k^H$ denotes the solution vector at the k^{th} iteration obtained by solving the Newton linear system. For linear variational problems, we simply restrict ourselves to a single Newton linear solve, which reduces exactly to classical FEM assembly for linear problems.

6. The Adjoint Problem

6.1. A Richer Space via Uniform Refinement

The adjoint solution must be represented in a richer space than the one used for the primal problem to obtain meaningful error estimates. There are several strategies that are commonly used to obtain such a representation. First, the adjoint problem can be solved in the same finite element space as the primal problem and then be enriched to a higher order polynomial space [8] or a nested mesh [35] by some local patch-wise operation, or variational multiscale enrichment [24] can be used in the context of stabilized finite elements. Alternatively, the adjoint problem can be solved in a higher order polynomial space [16], which we will refer to as p-enrichment. As a final option, the adjoint problem can be solved on a uniformly refined mesh [11], which we will refer to as h-enrichment.

In this work, we choose the h-enrichment approach for several reasons. First, we would like the adjoint solution to be as accurate as possible for error estimation purposes, so we choose to solve the adjoint problem in a globally richer finite element space. Additionally, for stabilized finite element methods, the use of p-enrichment would in general necessitate the use of higher order stabilization terms that vanish for lower-order finite element methods with simplical elements. These higher order terms are typically more difficult to implement than their lower order counterparts. Further, we remark that higher-order stabilized finite element methods are rarely used in practice, as stable higher-order mixed methods can usually be derived with fewer overall degrees of freedom [55]. Finally, we note that the unstructured mesh adaptation capabilities of the PUMI software make the h-enrichment approach readily available.

In the present context, we consider the term *uniform refinement* for triangles and tetrahedra to mean splitting each edge of the parent element at its midpoint. Or, in other words, creating new edges by by connecting the midpoints of the parent element's existing edges. The uniform refinement of a triangle results in 4 nested triangles and the uniform refinement of a tetrahedron results in 8 nested tetrahedra.

We have denoted the trial and test spaces used for the primal problem as $\mathcal{S}^H$ and $\mathcal{V}^H$, respectively. We denote the trial and test spaces on the uniformly nested mesh as $\mathcal{S}^h$ and $\mathcal{V}^h$, respectively, where $h < H$ is representative of a finer mesh size. Figure 2 illustrates the discretization for the coarse and fine trial and test spaces defined for a three dimensional geometry with a complex void inclusion.

6.2. Discrete Adjoint Approximation

Let $\boldsymbol{R}^h : \mathbb{R}^n \to \mathbb{R}^n$ denote the residual form of the system of nonlinear algebraic equations arising either from the Galerkin (7) or stabilized (9) model problem posed on the uniformly nested mesh. Let $\boldsymbol{u}_H^h := I_H^h \boldsymbol{u}^H$ denote the prolongation of the primal finite element solution onto the richer space $\mathcal{S}^h$ via interpolation, Let $J^h : \mathbb{R}^n \to \mathbb{R}$ denote the discretization of the functional QoI on the uniformly nested fine space. We approximate the adjoint problem (4) in a discrete manner [17, 57, 58, 59], by solving

$$\left[\left.\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\right|_{u_H^h}\right]^T \boldsymbol{z}^h = \left[\left.\frac{\partial J^h}{\partial \boldsymbol{u}^h}\right|_{u_H^h}\right]^T . \tag{13}$$

This allows us to automate the process of solving the adjoint problem, as discussed below. Here $\boldsymbol{z}^h \in \mathbb{R}^n$ denotes the adjoint solution vector on the nested discretization.

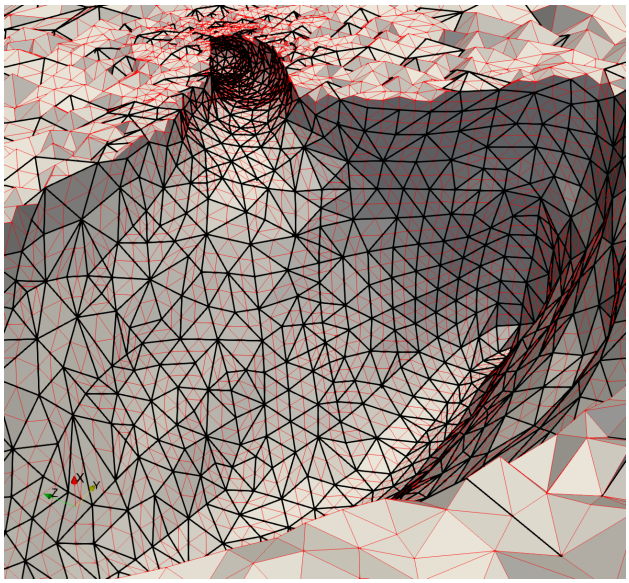

Figure 2: Example of a nested mesh (red edges) obtained via a uniform refinement of a base mesh (black edges) in three dimensions.

6.3. Automated Solution Based on Residual Formulation

The construction of the Jacobian transpose matrix $\left[\partial \boldsymbol{R}^h / \partial \boldsymbol{u}^h\right]^T$ is performed in the same automated manner as the Jacobian for the primal problem. That is, for each element, we compute consistent element tangent stiffness matrices via automatic differentiation of element-level contributions to the residual vector. However, during the *scatter* phase of the template-based generic programming process, we transpose the element-level tangent matrices and sum them into global Jacobian adjoint matrix. The computation of the Jacobian adjoint is done using the same templated code that is used to compute the primal residual vector and the Jacobian matrix, as illustrated by listings 2 and 3.

Similarly, the construction of the functional derivative vector $\left[\partial J^h / \partial \boldsymbol{u}^h\right]^T$ is done by evaluating derivatives of element-level contributions to the functional via automatic differentiation. This results in element-level derivative vectors that are then assembled into the global functional derivative vector. Listings 4 and 5 illustrate the implementation of two quantities of interest in the Goal application. Once the Jacobian transpose matrix and functional derivative vector have been assembled, we solve the adjoint problem (13) using a sparse iterative solver in parallel.

7. Error Estimation

7.1. Two-Level Error Estimates

Following Venditti and Darmofal [57, 58, 59], we review adjoint-based error estimation using two discretization levels. The discrete residual form of the governing equations for a Galerkin (7) finite element method or a stabilized finite element method (9) posed on the fine space can be expressed as

$$\boldsymbol{R}^h(\boldsymbol{u}^h) = \boldsymbol{0}. \tag{14}$$

Taking Taylor expansions of the discrete residual $\boldsymbol{R}^h$ evaluated on the fine space and the discrete functional J^h evaluated on the fine space about the point $\boldsymbol{u}_H^h$ yields

$$\boldsymbol{R}^h(\boldsymbol{u}^h) = \boldsymbol{R}^h(\boldsymbol{u}_H^h) + \left[\left.\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\right|_{\boldsymbol{u}_H^h}\right](\boldsymbol{u}^h - \boldsymbol{u}_H^h) + \dots \tag{15}$$

and

$$J^h(\boldsymbol{u}^h) = J^h(\boldsymbol{u}_H^h) + \left[\frac{\partial J^h}{\partial \boldsymbol{u}^h}\bigg|_{\boldsymbol{u}_H^h}\right](\boldsymbol{u}^h - \boldsymbol{u}_H^h) + \dots \tag{16}$$

respectively.

Using equation (14), the discretization error between the two spaces can be approximated to first order as

$$(\boldsymbol{u}^h - \boldsymbol{u}_H^h) \approx -\left[\frac{\partial \boldsymbol{R}^h}{\partial \boldsymbol{u}^h}\bigg|_{\boldsymbol{u}_H^h}\right]^{-1} \boldsymbol{R}^h(\boldsymbol{u}_H^h), \tag{17}$$

which can then be substituted into the functional Taylor expansion (16) to obtain the so-called *adjoint weighted residual*

$$J^h(\boldsymbol{u}^h) - J^h(\boldsymbol{u}_H^h) \approx -\boldsymbol{z}^h \cdot \boldsymbol{R}^h(\boldsymbol{u}_H^h) \tag{18}$$

where $\boldsymbol{z}^h$ is the solution to the adjoint problem (13).

7.2. Modified Functional Error Estimate

Assume that the QoI converges at the rate k, such that $J - J^h(\boldsymbol{u}_H^h) = cH^k$ and $J - J^h(\boldsymbol{u}^h) = ch^k$, where J denotes the exact value of the QoI. If the fine space is obtained via uniform mesh refinement, then the ratio of the fine mesh size to the coarse mesh size is given as $\frac{h}{H} = \frac{1}{2}$. Consider the ratio

$$\begin{aligned}
\frac{J^h(\boldsymbol{u}^h) - J^h(\boldsymbol{u}_H^h)}{J - J^h(\boldsymbol{u}_H^h)} &= \frac{\left[J - J^h(\boldsymbol{u}_H^h)\right] - \left[J - J^h(\boldsymbol{u}^h)\right]}{J - J^h(\boldsymbol{u}_H^h)} \\
&= \frac{cH^k - ch^k}{cH^k} \\
&= 1 - \left(\frac{h}{H}\right)^k \\
&= 1 - \left(\frac{1}{2}\right)^k
\end{aligned} \tag{19}$$

in the limit as $H \to 0$ [17]. We call this ratio $\alpha := 1 - (1/2)^k$. Let η denote an approximation to the functional error $J - J^h(\boldsymbol{u}_H^h)$. Let $\mathcal{I}$ denote the effectivity index given by

$$\mathcal{I} = \frac{\eta}{J - J^h(\boldsymbol{u}_H^h)}. \tag{20}$$

We would like to obtain error estimates η that lead to effectivity indices of $\mathcal{I} = 1$ as $H \to 0$. To this end, we recall $J^h(\boldsymbol{u}^h) - J^h(\boldsymbol{u}_H^h) \approx -\boldsymbol{z}^h \cdot \boldsymbol{R}^h(\boldsymbol{u}_H^h)$ from equation (18) to obtain the scaled adjoint weighted residual error estimate

$$\eta = -\frac{1}{\alpha}\boldsymbol{z}^h \cdot \boldsymbol{R}^h(\boldsymbol{u}_H^h). \tag{21}$$

7.3. Error Localization for Galerkin Methods

Following the approach of Richter and Wick [48], we introduce a partition of unity ϕ_i, such that $\sum_i \phi_i = 1$, into the weighting function slot for the error estimate to localize the error. In this work, the partition of unity is realized as linear Lagrange basis functions. This yields local error contributions η_i at the n_{vtx} mesh vertices in the mesh. Let $z^h \in \mathcal{V}^h$ be the finite element solution obtained by solving the discrete adjoint problem (13). We assume that this solution well approximates the continuous adjoint problem (4), such that

$z \approx z^h$. Let z^H denote the interpolant of z^h onto the coarse space $\mathcal{V}^H$. Recalling the error representation (8) for Galerkin finite elements, we obtain partition of unity-based correction indicators η_i in the following manner

$$J(u) - J(u^H) \approx \sum_{i=1}^{n_{vtx}} \underbrace{-\mathcal{R}_g((z^h - z^H)\phi_i \; ; \; u^H)}_{\eta_i}. \tag{22}$$

7.4. Error Localization for Stabilized Methods

Error localization for the stabilized finite element formulation (9) proceeds in the same manner as the previous section. Let $z^h \in \mathcal{V}^h$ denote the finite element solution obtained by solving the discrete adjoint problem (13) and let z^H denote the interpolant of z^h onto the coarse space $\mathcal{V}^H$. Introducing a partition of unity into the error representation (10) for stabilized finite element methods with the approximation $z \approx z^h$ yields the vertex-based correction indicators η_i:

$$J(u) - J(u^H) \approx \sum_{i=1}^{n_{vtx}} \underbrace{-\mathcal{R}_g((z^h - z^H)\phi_i \; ; \; u^H) + \mathcal{R}_\tau(z^H\phi_i \; ; \; u^H)}_{\eta_i}. \tag{23}$$

Once correction indicators η_i have been evaluated, an approximate upper bound $\hat{\eta}$ for the error is computed by summing the absolute value of the error contributions over all mesh vetrices

$$\hat{\eta} = \sum_{i=1}^{n_{vtx}} |\eta_i|. \tag{24}$$

7.5. Automated Error Localization Based on Residual Implementation

During the assembly of the adjoint problem (13), the evaluation of element-level contributions to the residual vector evaluated on the fine space $\boldsymbol{R}^h(\boldsymbol{u}_H^h)$ are necessarily computed by the machinery of forward automatic differentiation. Thus, during the `scatter` phase for the adjoint problem computation, we additionally sum element-level contributions to the fine residual to assemble the global vector $\boldsymbol{R}^h(\boldsymbol{u}_H^h)$. This, along with the solution $\boldsymbol{z}^h$ to the adjoint problem (13) provides enough information to compute the adjoint-weighted residual estimate (21) in an automated fashion.

Again, we let $z^h \in \mathcal{V}^h$ denote the finite element solution to the discrete adjoint problem (13) and let z^H denote the interpolant of z^h onto the coarse space $\mathcal{V}^H$. We refer again to Listings 2 and 3, which illustrate implementations of Galerkin and stabilized semilinear forms $\mathcal{R}_g$ and $\mathcal{R}_\tau$, respectively, in the Goal application. Specifically, we remark that these residual evaluations contain the evaluation of weighting functions and their derivatives, given with calls to the methods `w->val(node)` and `w->grad(node, dim)`, respectively. To localize the error in an automated fashion, we override the calls to these methods such that they return values of the adjoint solution multiplied by a partition of unity. For instance, at a given reference location $\boldsymbol{\xi}$ in a given element, the partition of unity-based weight for the Galerin residual $\mathcal{R}_g$ is computed as

$$\texttt{w->val(n)} = \left[(z^h - z^H) \cdot \phi_n\right]\Big|_{\boldsymbol{\xi}}, \tag{25}$$

and its corresponding gradient is computed as

$$\texttt{w->grad(n)} = \nabla\left[(z^h - z^H) \cdot \phi_n\right]\Big|_{\boldsymbol{\xi}}. \tag{26}$$

Similarly, for the stabilized residual $\mathcal{R}_\tau$, the partition of unity-based adjoint weight is computed as

$$\texttt{w->val(n)} = \left[z^H \cdot \phi_n\right]\Big|_{\boldsymbol{\xi}}, \tag{27}$$

and its corresponding gradient is computed as

$$\texttt{w->grad(n)} = \nabla \left[z^H \cdot \phi_n \right] \big|_{\boldsymbol{\xi}}. \tag{28}$$

In this manner, we have introduced partition of unity-based adjoint weights that have the same data type as the weights used for the computation of the primal and adjoint problems.

Using the adjoint weights in the error localization evaluation results in element-level residual vectors that correspond to contributions to the localized correction indicators η_i. During the `scatter` phase of the error localization evaluation, we sum these element level contributions to the appropriate mesh vertices to compute the localized correction indicators η_i.

8. Mesh Adaptation

Given localized correction indicators η_i at mesh vertices, we compute element-level correction indicators η_e for $e = 1, 2, \ldots, n_{el}$, where n_{el} is the number of elements in the coarse discretization, by interpolating the vertex-based indicators to element centers and taking the result's absolute value.

We then specify a *mesh size field* that defines the desired value of edge lengths over the mesh. From a high-level, we would like to specify this size field such that areas of the mesh that contribute strongly to the error in the QoI are refined, and areas of the mesh that are insensitive to the error are coarsened. Following Boussetta et al. [10], we specify a size field that attempts to equidistribute the error in an output adapted mesh with N target elements. Let p be the polynomial interpolant order for the chosen finite element method. In the present setting, $p = 1$. We first define the global quantity G as

$$G = \sum_{e=1}^{n_{el}} (\eta_e)^{\frac{2d}{2p+d}}. \tag{29}$$

This global quantity arises by considering *a priori* convergence rates for the input mesh and attempting to find an optimal mesh size for an output mesh with N elements. [10]. With this global quanity, new element sizes H_e^{new} are computed by scaling the previous element size H_e

$$H_e^{\text{new}} = \left(\frac{G}{N} \right)^{\frac{1}{d}} (\eta_e)^{\frac{-2}{2p+d}} H_e. \tag{30}$$

Finally, to prevent excessive refinement or coarsening in a single adaptive step, we clamp the element size such that it is no smaller than one quarter and no greater than twice the previous element size. This clamping is performed to ensure that mesh adaptation is being driven by accurate error indicators.

$$\frac{1}{4} \leq \frac{H_e^{\text{new}}}{H_e} \leq 2. \tag{31}$$

9. Quantities of Interest

In this section, we review three quantities of interest that we have implemented in the Goal application. One benefit of the current automated approach is that additional quantities of interest can be rapidly prototyped and investigated with relative ease. Here, we refer to the domain discretized by the finite element mesh as Ω.

9.1. Point-Wise Solution Component

First, we consider the evaluation of a component u_i of the solution u at a given spatial localtion $\boldsymbol{x}$. This functional can be expressed as

$$J(u) = \int_{\Omega} \delta(\boldsymbol{x} - \boldsymbol{x}_0)\, u_i \, \mathrm{d}\Omega, \tag{32}$$

Listing 4: Evaluation of the integrated displacement over a sub-domain.

```
template <typename ScalarT>
void AvgDisp<ScalarT>::at_point(
    apf::Vector3 const&, double w, double dv) {
  for (int i = 0; i < num_dims; ++i)
    this->elem_value += u->val(i) * w * dv;
  this->elem_value /= num_dims;
}
```

where δ is the Dirac delta function. We implement this quantity of interest as a discrete delta function, such that the right-hand side for the adjoint problem takes the form

$$\frac{\partial J^h}{\partial \boldsymbol{u}^h} = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 \end{bmatrix}. \tag{33}$$

For this implementation, a mesh vertex is always placed at the spatial location $\boldsymbol{x}_0$, the QoI derivative vector is zeroed out, and we place a one in the row of the QoI derivative vector that corresponds to the i^{th} component of the solution at the vertex.

9.2. Integrated Solution Over a Sub-Domain

Next, we consider the integrated solution over a sub-domain $\Omega_0 \subset \Omega$, which can be expressed as

$$J(u) = \int_{\Omega_0} \frac{1}{n_c} \sum_{i=1}^{n_c} u_i \, \mathrm{d}\Omega. \tag{34}$$

Here, n_c denotes the number of components for the solution vector. As an example, Listing 4 demonstrates the Goal implementation for the QoI corresponding to the integrated displacement over a sub-domain.

9.3. Integrated von-Mises Stress Over a Sub-Domain

Finally, specifically for mechanics problems, we consider the evaluation of the von-Mises stress integrated over a sub-domain $\Omega_0 \subset \Omega$, given as

$$J(u) = \int_{\Omega_0} \sigma_{vm} \, \mathrm{d}\Omega, \tag{35}$$

where the von-Mises stress σ_{vm} is defined as

$$\sigma_{vm} := \sqrt{\frac{3}{2} \boldsymbol{\sigma}'_{ij} \boldsymbol{\sigma}'_{ij}}. \tag{36}$$

Here summation over repeated indices is implied and $\boldsymbol{\sigma}' = \boldsymbol{\sigma} - \frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma})\boldsymbol{I}$ denotes the deviatoric part of the Cauchy stress tensor $\boldsymbol{\sigma}$. The von-Mises stress is often used in yield criterion for elastoplastic constitutive models, and is hence of particular interest for solid mechanics design applications.

We note that this funciton $J(u)$ has sources of nonlinearities from the deviatoric stress tensor $\boldsymbol{\sigma}'$ and further nonlinearites introduced by the definition of the von-Mises stress, which includes the square of deviatoric stress components and a square root operation. The linearization and implementation of this QoI, as required for adjoint-based error estimation, would be cumbersome at best without some kind of automated approach. In contrast, Listing 5 illustrates the simplicity of the relevant `C++` code that implements integration point contributions to this specific QoI in the Goal application.

Listing 5: Evaluation of the integrated von-Mises stress over a sub-domain.

```
template <typename ScalarT>
void AvgVM<ScalarT>::at_point(
    apf::Vector3 const&, double w, double dv) {
  auto sigma = model->get_cauchy();
  ScalarT vm = compute_von_mises<ScalarT>(sigma);
  this->elem_value += vm * w * dv;
}
```

10. Results

10.1. Poisson's Equation

As a first example, we investigate error estimation and mesh adaptation in Poisson's equation for the model problem

$$\begin{cases} -\nabla^2 u = f & \boldsymbol{x} \in \Omega, \\ u = 0 & \boldsymbol{x} \in \partial\Omega. \end{cases} \tag{37}$$

This model problem leads to the Galerkin finite element method: find $u^H \in \mathcal{V}^H$ such that $\mathcal{R}_g(w^H; u^H) = 0$ for all $w^H \in \mathcal{V}^H$. Here the residual $\mathcal{R}_g$ is defined as

$$\mathcal{R}_g(w^H; u^H) := (\nabla w^H, \nabla u^H) - (w^H, f), \tag{38}$$

and the space $\mathcal{V}^H$ is given by

$$\mathcal{V}^H := \{u^h \in H^1(\Omega) : u^H = 0 \text{ on } \partial\Omega\,,\, u^H|_{\Omega_e} \in \mathbb{P}^1\}. \tag{39}$$

Here Ω_e denotes an element in a decomposition of the domain Ω into n_{el} non-overlapping elements such that $\cup_{e=1}^{n_{el}} \Omega_e = \Omega$ and $\Omega_i \cap \Omega_j = \varnothing$ if $i \neq j$. Additionally, $\mathbb{P}^1$ denotes the space of piecewise linear polynomials.

The domain is chosen to be $\Omega := [-1, 1] \times [-1, 1] \setminus [-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$ as shown in Figure 3. The data is chosen to be $f = 1$ and we consider a point-wise QoI of the form $J(u) = \int_\Omega \delta(\boldsymbol{x} - \boldsymbol{x}_0) u \, \mathrm{d}\Omega$, where the point of interest $\boldsymbol{x}_0$ is chosen to be $\boldsymbol{x}_0 = (0.75, 0.75)$. This problem was initially studied in the reference [6], where the QoI was determined to have a reference value of $J(u) = 0.0334474 \pm 1e\text{-}7$. Presently, we demonstrate that our automated approach can reproduce the results for traditional adjoint-based error estimation found in [6].

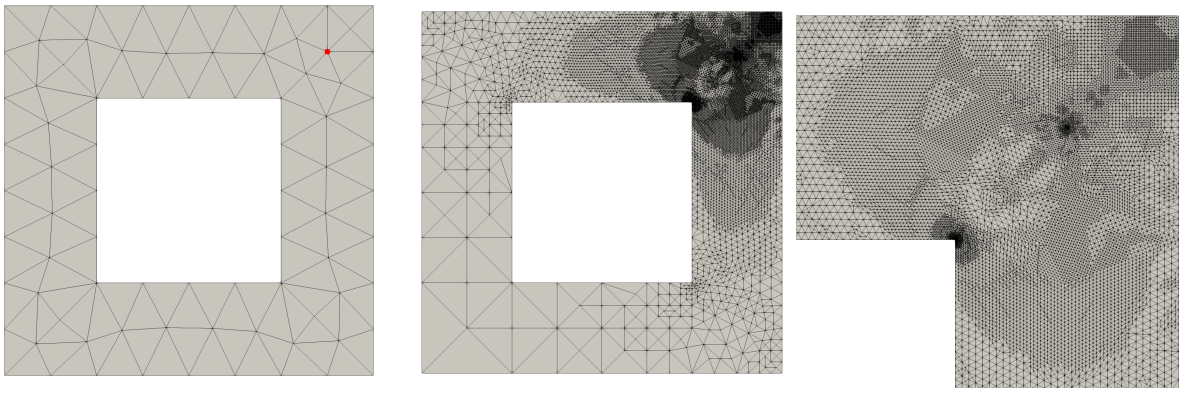

Figure 3: Domain and initial mesh (left) for the Poisson's equation example with the QoI point indicated in red, final adapted mesh (middle), and a close up of the upper-right hand corner of the final adapted mesh (right).

Starting from the initial mesh shown in Figure 3, the steps:

Solve Primal → Solve Adjoint → Estimate Error → Adapt Mesh

were performed seven times. The adaptive simulation was run using 4 MPI ranks. The mesh size field was set according to equation (30) so that the target number of elements is twice that of the current mesh. Figure 3 also shows the final adapted mesh resulting from this procedure. We remark that the distribution of degrees of freedom in this mesh closely resembles the results obtained in reference [6].

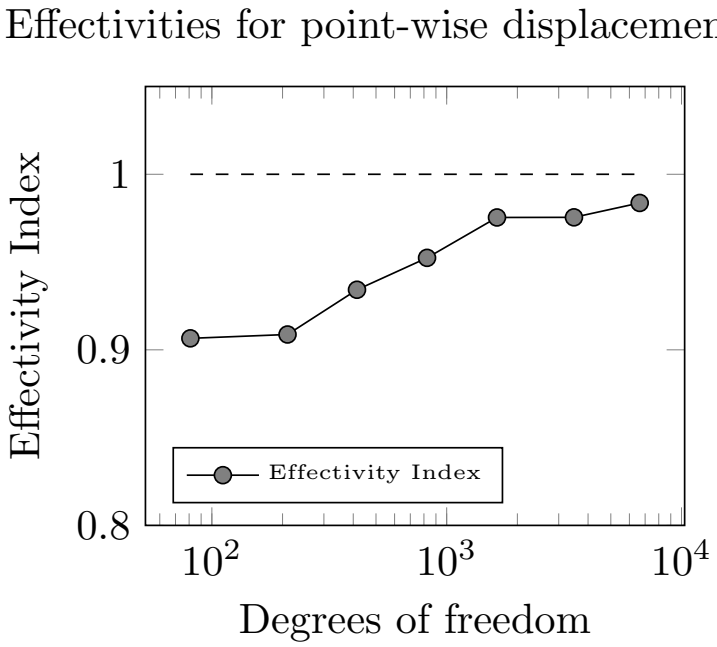

Figure 4: Effectivity indices for the adaptive Poisson's equation example.

We expect this functional to converge at the rate $k = 2$, such that the scaling factor α used in the estimate (21) is given as $\alpha = \frac{3}{4}$. We consider the "exact error" $\mathcal{E} = J(u) - J(u^H)$ and the effectivity index $\mathcal{I} = \frac{\eta}{\mathcal{E}}$, where η is the estimate given by equation (21). Here we have placed quotations around the term exact error because we have only approximated the exact value of the QoI $J(u)$, and not truly recovered its exact value. Figure 4 plots the effectivity index $\mathcal{I}$ versus the number of degrees of freedom in the adaptive process. This plot demonstrates the ability of the error estimate to recover the "exact error" as $H \to 0$.

Figure 5 displays the evolution of various errors during the adaptive process. The "exact error" $\mathcal{E}$ and the estimated error η are very close, as previously demonstrated by the effectivity index $\mathcal{I}$. Additionally, the approximated upper bound on the error $\hat{\eta}$ overestimates the error, but not to a large degree. This provides some indication that the correction indicators are effective in that they do not drastically overestimate error. Finally, we remark that an improved *corrected* QoI functional value can be computed as $J^*(u^H) = J(u^H) + \eta$. Figure 5 demonstrates that this corrected value is nearly an order of magnitude more accurate than the computed functional value $J(u^H)$ during the adaptive process.

Finally, Figure 6 demonstrates the evolution of the "exact error" for two adaptive strategies. The first strategy uniformly refines the mesh at each adaptive step and the second strategy performs the adjoint-based adaptive scheme developed in this work. We note that the error for the adjoint-based adaptive scheme converges faster than the uniform refinement scheme. Further, this convergence plot is consistent with the reference [6].

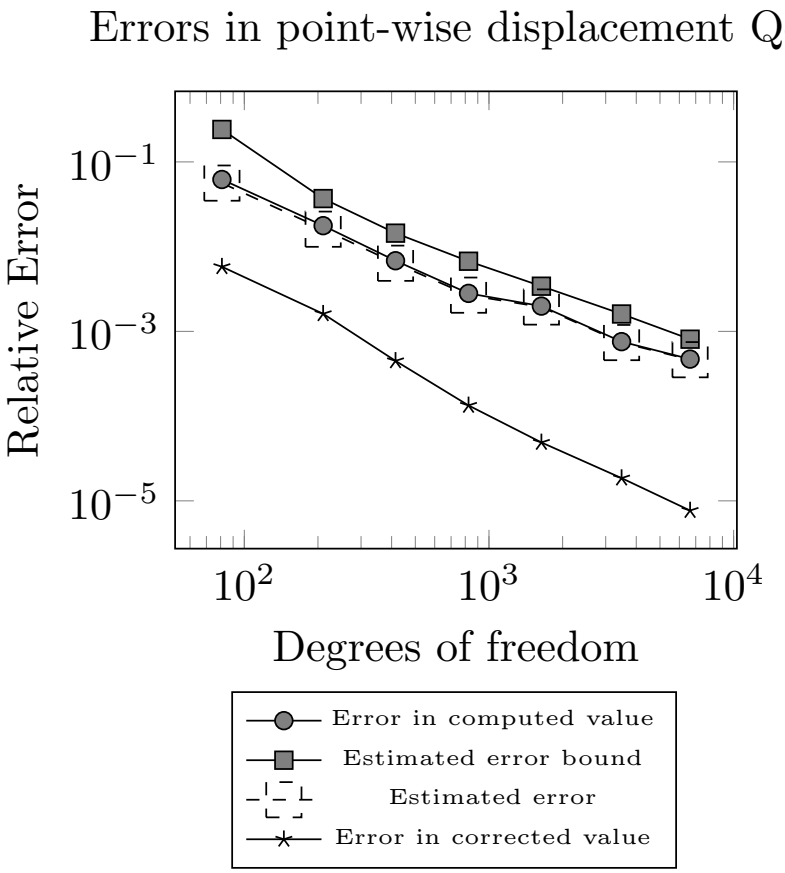

Figure 5: Errors for the point-wise QoI for the adaptive Poisson's equation example.

10.2. A Cell Embedded in a Matrix

Recently, the automated approach developed in this paper was applied to a stabilized mixed pressure-displacement finite element formulation [43] for the governing equations of finite deformation elasticity in a total Lagrangian setting [23]. For two and three dimensional problems in nonlinear elasticity, the automated approach was shown to effectively estimate the error and provide improved error convergence rates via adjoint-based mesh adaptation over uniform refinement.

In this section, the parallelization of a biomechanical application presented in the reference [23] is discussed. First, the governing equations are briefly reviewed. For mixed pressure-displacement formulations in the Goal application, the Galerkin residual is defined as:

$$\begin{aligned}&\mathcal{R}_g(\boldsymbol{W}^H;\boldsymbol{U}^H) := \\ &\int_\Omega \boldsymbol{P} : \nabla \boldsymbol{w}^H \,\mathrm{d}\Omega + \int_\Omega \left[\frac{p^H}{\kappa} - \frac{1}{2j}(j^2-1)\right] q^H \,\mathrm{d}\Omega - \int_{\partial\Omega_h} \boldsymbol{h}\cdot\boldsymbol{w} \,\mathrm{d}\Gamma,\end{aligned} \tag{40}$$

and the stabilization residual is defined as:

$$\mathcal{R}_\tau(\boldsymbol{W}^H;\boldsymbol{U}^H) := \sum_{e=1}^{n_{el}} \int_{\Omega_e} \tau_e (j\boldsymbol{F}^{-1}\boldsymbol{F}^{-T}) : (\nabla p^H \otimes \nabla q^H) \,\mathrm{d}\Omega. \tag{41}$$

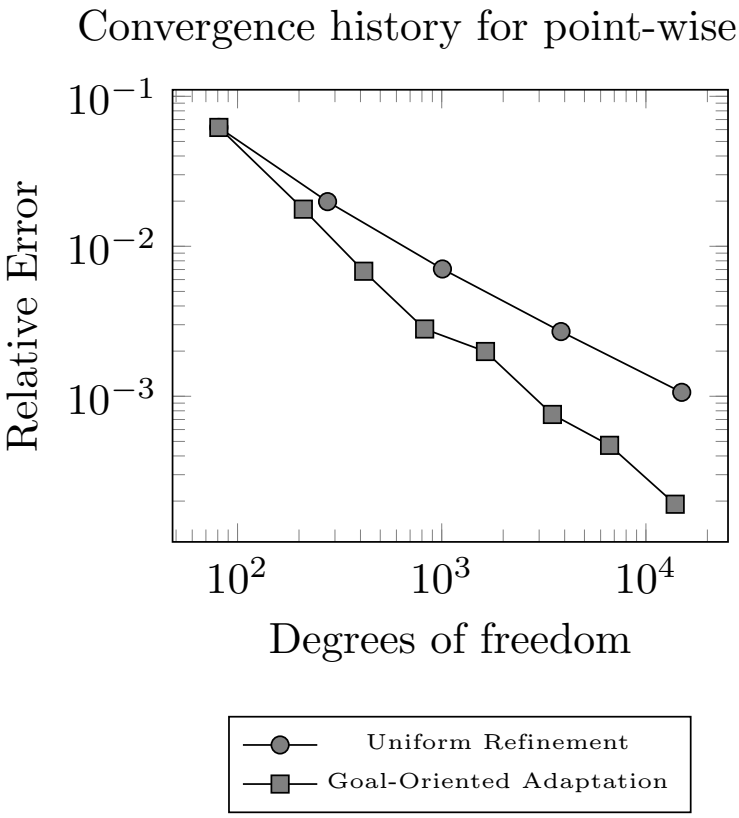

Figure 6: Error convergence using uniform mesh refinement and adjoint-based error estimation for the adaptive Poisson's equation example.

Here, $\boldsymbol{F}$ is the deformation gradient, $j := \det(\boldsymbol{F})$, $\boldsymbol{h}$ is an applied traction over the boundary $\partial\Omega_h$, $\boldsymbol{P} := j\boldsymbol{\sigma}\boldsymbol{F}^{-T}$ is the first Piola-Kirchhoff stress tensor, $\boldsymbol{\sigma}$ is the Cauchy stress tensor, n_{el} is the total number of elements in the mesh, and $\tau_e := \frac{c_0 H_e^2}{2\mu}$ is a mesh-dependent stabilization parameter, where c_0 is a non-negative stability constant, H_e denotes an element mesh size and μ denotes the bulk modulus. The Cauchy stress tensor is defined via a neo-Hookean constitutive relationship. The total solution vector is defined as $\boldsymbol{U}^H := [\boldsymbol{u}^H, p^H]$, where $\boldsymbol{u}^H$ corresponds to displacements and p^H corresponds to pressures. Similarly, the total weighting vector is defined as $\boldsymbol{W}^H := [\boldsymbol{w}^H, q^H]$, where $\boldsymbol{w}^H$ denotes a weighting function corresponding to displacements and q^H is a weighting function corresponding to pressures.

We focus on a microglial cell with dimensions of about $20\mu m \times 20\mu m \times 20\mu m$ embedded in an extracellular matrix of dimension $100\mu \times 100\mu m \times 100\mu m$. The QoI is chosen to be a local integrated displacement $J(\boldsymbol{U}) = \int_{\Omega_0} \frac{1}{3}(u_x + u_y + u_z)\,\mathrm{d}\Omega$, defined over a box Ω_0 with dimensions $30\mu m \times 30\mu m \times 30\mu m$ that bounds the microglial cell. Figure 7 shows the geometry defining the microglial cell, the bounding box Ω_0, and the extracellular matrix. The shear modulus is defined as $\mu = 600$ Pa and Poisson's ratio is set to be $\nu = 0.4999$.

To drive the problem, traction boundary conditions are imposed along the surface of the microglial cell. The magnitude of the applied traction $\boldsymbol{h}$ is defined to be 10 times the distance to the center of the cell and its direction points inward towards the cell center. This traction is consistent with observed physical behavior [14]. Displacements $u_x = 0$, $u_y = 0$, and $u_z = 0$ are applied to the faces with constant minimum x-coordinate value, constant minimum y-coordinate value, and constant minimum z-coordinate value, respectively, to constrain rigid body rotations and translations.

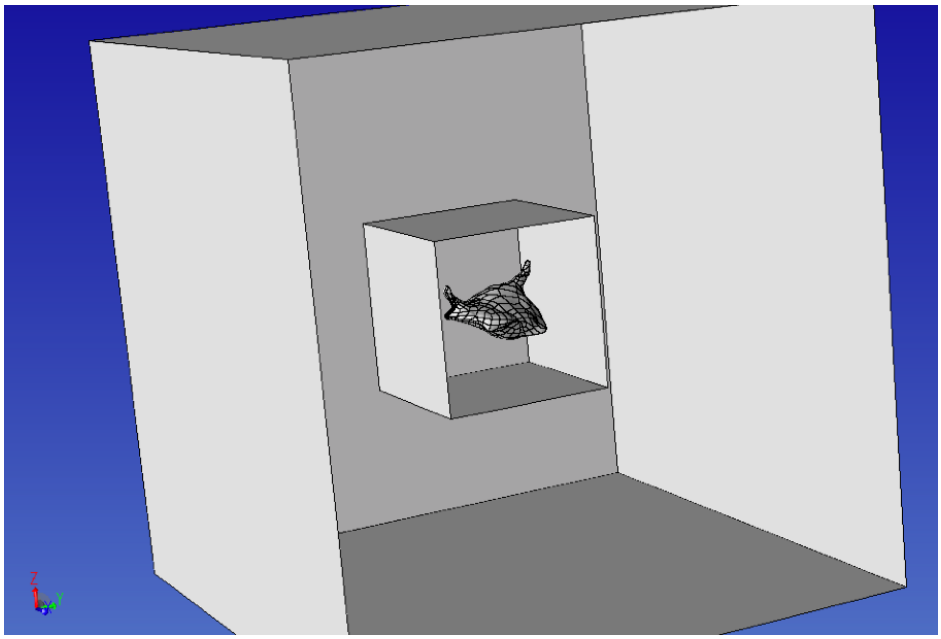

Figure 7: Domains for the microglial cell example.

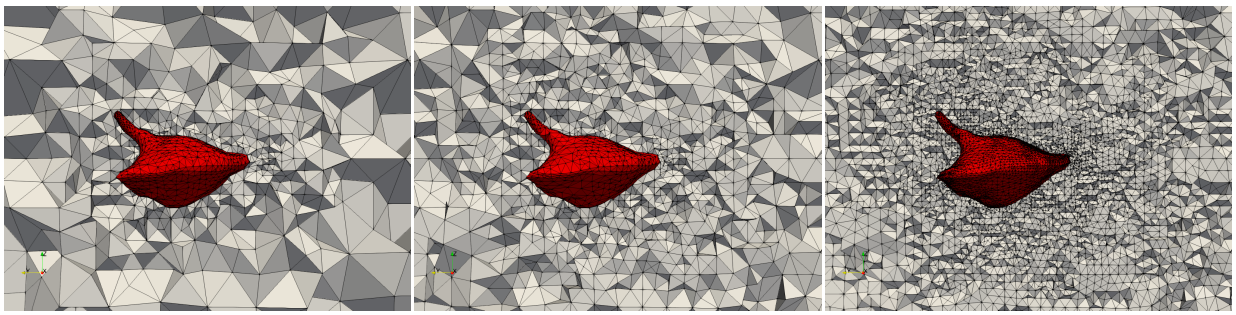

Figure 8: A close-up of the initial mesh (left) the mesh after 5 adaptive iterations (center) and the final adapted mesh (right) for the microglial cell example.

Figure 8 demonstrates an initial mesh, which contains around $30,000$ degrees of freedom. From this initial mesh, the steps

$$\text{Solve primal PDE} \rightarrow \text{Solve adjoint PDE} \rightarrow \text{Localize error} \rightarrow \text{Adapt mesh}$$

were successively performed 10 times. During the adapt stage, the mesh size field was set such that desired number of elements N in the output mesh is 1.5 times the number of elements in the previous mesh, according to equation (30). Figure 8 additionally demonstrates the adapted meshes obtained at the fifth and final adaptive iteration. In particular, both *coarsening* and *refinement* is performed during the adaptive iterations.

The problem was run using 16 MPI ranks. Figure 9 demonstrates the parallel partitioning for the initial mesh and for the final adapted mesh obtained after 10 adjoint-based adaptive iterations. To ensure partitioning quality, ParMA was utilized to guarantee the imbalance of vertices and elements across parallel partitions is no greater than 5%.

Figure 10 presents a breakdown of the total percentange of CPU time spent on each step in the adaptive analysis. For every adaptative iteration, the error localization (23) takes only a small percentage of the

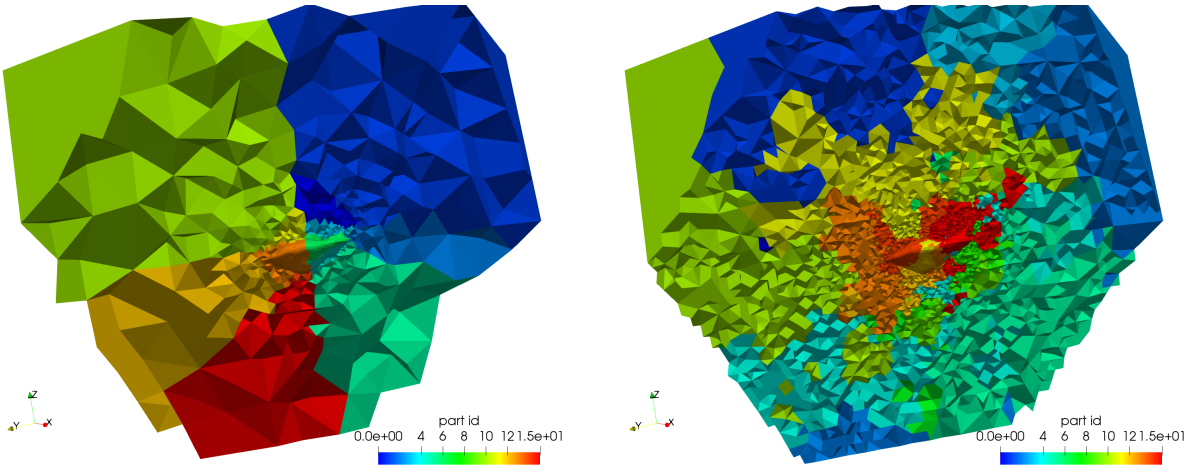

Figure 9: The parallel mesh partitioning for the initial mesh (left) and the final adapted mesh (right) for the microglial cell example.

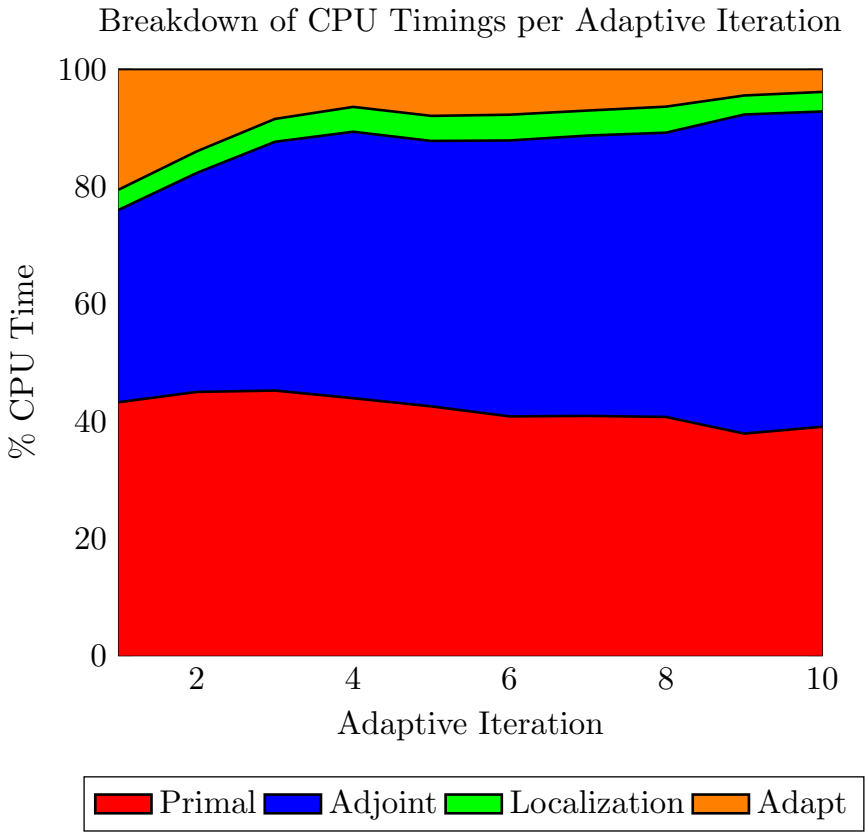

Figure 10: Breakdown of the CPU time spent for each portion of the adaptive process for the microglial cell example.

total CPU time, as it essentially amounts to an evaluation of the residual vector on the fine space. More interestingly, mesh adaptation initially accounts for about 20 percent of the total CPU time but decreases as

the adaptive simulation progresses. This is explained by the fact that the initial adaptive iteration requires more work to optimally distribute the degrees of freedom for the functional QoI as compared to subsequent adaptive iterations. In addition to refinement and coarsening operations, the mesh adaptation step also performs *shape correction* to ensure elements are not too heavily skewed [32]. Finally, we note that the adjoint problem accounts for roughly 40 to 50 percent of the CPU time over the course of the adaptive simulation. While process of adjoint-based error estimation is not cheap for this example, we provide two justifying remarks. First, this problem required only 3 to 4 Newton iterations for each primal solve. For constitutive models with higher degrees of nonlinearity or for problems loaded to higher strains, it is not uncommon for Newton's method to converge in 7 to 10 iterations. In these scenarios, the relative cost of adjoint-based error estimation is not as extreme. Second, for this computational price, we have achieved very accurate error estimates as shown in the reference [23].

10.3. Elastoplasticity in an Array of Solder Joints

In this section, we investigate the utility of adjoint-based mesh adaptation for a thermomechanical analysis of an array of solder joints used in microelectronics fabrication. We consider a 6×6 array of solder joints sandwiched between two materials with distinct thermomechanical properties to model a portion of the process of 'flip-chip' manufacturing [9]. The full geometry is shown in Figure 11. We consider an elastoplastic constitutive model with a von-Mises yield surface and linear isotropic hardening, as given by Simo and Hughes [51] with a temperature correction for the stress tensor [33]. The top slab, solder joints, and bottom slab are modeled with the distinct material properties given in reference [9].

To drive the problem, the entirety of the domain is cooled from a reference temperature $T_{ref} = 393K$ to a resting temperature of $T_f = 318K$ in a single load step. The faces with minimum x, y, and z coordinate values were constrained to have zero displacements in the x, y, and z directions, respectively. As a QoI, we consider the integrated von-Mises stress given by equation (35) over three solder joints shown in yellow in Figure 11.

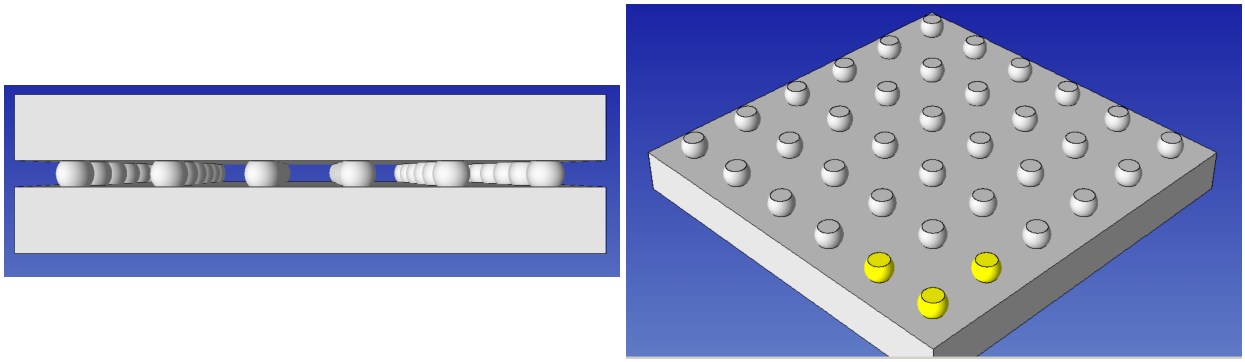

Figure 11: The solder joint array geometry (left) and the geometric specification of the integrated von-Mises QoI (right).

The primal problem was solved on a sequence of uniformly refined meshes, starting with an initial mesh with about 1 million elements distributed over 16 MPI ranks, and finalizing with a mesh with over half a billion elements distributed over 8192 MPI ranks. For each solve, the work load for each mesh part (MPI rank) was held constant at approximately $70,000$ elements. Figure 12 demonstrates weak scaling timing results for various aspects of the primal solve. In particular, we remark that the assembly of the residual vector and Jacobian matrix scale well as the number of MPI ranks increases. The preconditioning routine shows a slight increase in time as the number of MPI ranks increases, but this increase is not drastic. The time to solve the linear system, however, does not scale optimally. Improvements to parallel performance could likely be made by more finely tuning the preconditioning and linear solver routines for the specific problem, but this is outside the scope of the present work.

The approximate QoI, $J^H(\boldsymbol{u}^H)$, was computed at each primal solve. Using the QoI evaluations from the finest three meshes, we performed Richardson extrapolation [47] to obtain a more accurate representation

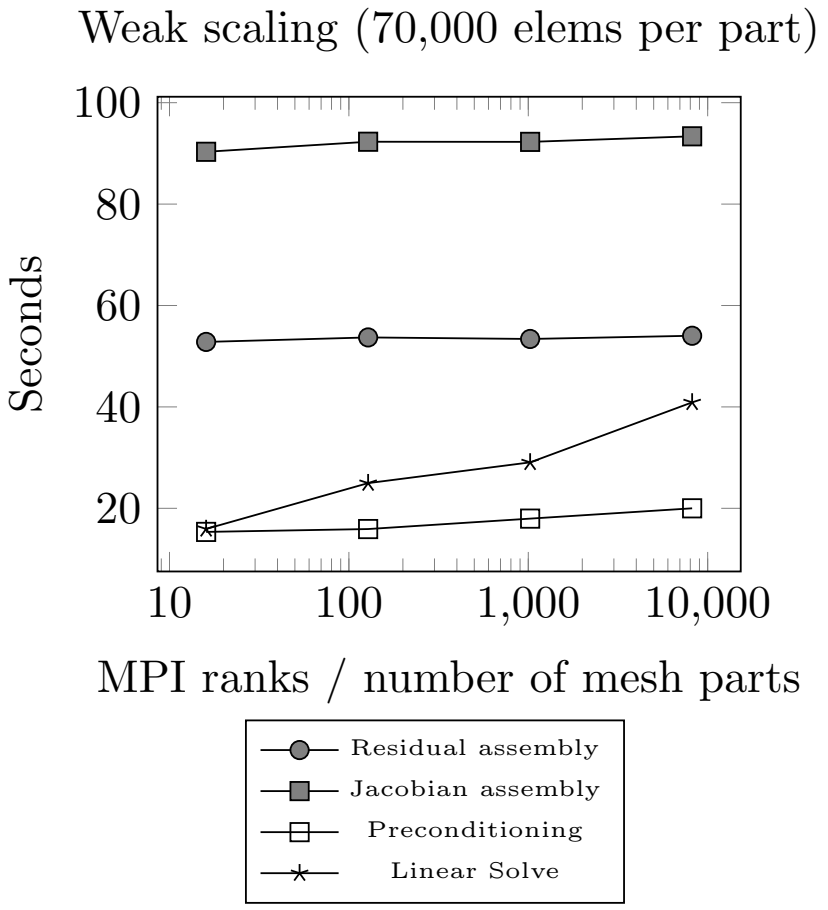

Figure 12: Weak scaling for the Goal application.

of the QoI. This value was given as $J(u) = 328.9$. We consider the extrapolated value to be the "true" QoI value and measure errors with respect to it. The expected convergence rate of the QoI is $k = 1$, which is confirmed by the Richardson extrapolation procedure.

From the same initial mesh used in the weak scaling study, we iteratively performed the steps:

$$\text{Solve Primal} \rightarrow \text{Solve Adjoint} \rightarrow \text{Estimate Error} \rightarrow \text{Adapt Mesh}$$

with a restart after each mesh adaptation using 128, 256, and 512 MPI ranks, such that the output number of elements N in the adapted mesh was targeted to be 1 million, 2 million, and 4 million elements, respectively. Figure 13 shows different components of the adjoint solution obtained during the adjoint-based adaptive process. Figure 14 shows the spatial distribution of the error for the given QoI as computed by the adjoint-based error estimation process. Unsurprisingly, the majority of the error is localized to the area which geometrically defines the QoI. However, there are also contributions to the error from nearby solder joints that decrease as the distance from the 3 QoI solder joints increases. These additional contributions to the error are mostly gathered at the interface between solder joints and the underlying material slab, where von-Mises stress concentrations exist.

Figures 15 and 16 demonstrate the intial mesh used for the solder joint problem and the final adapted mesh obtained via adjoint-based adaptation. These figures clearly demonstrate that the 3 solder joints that

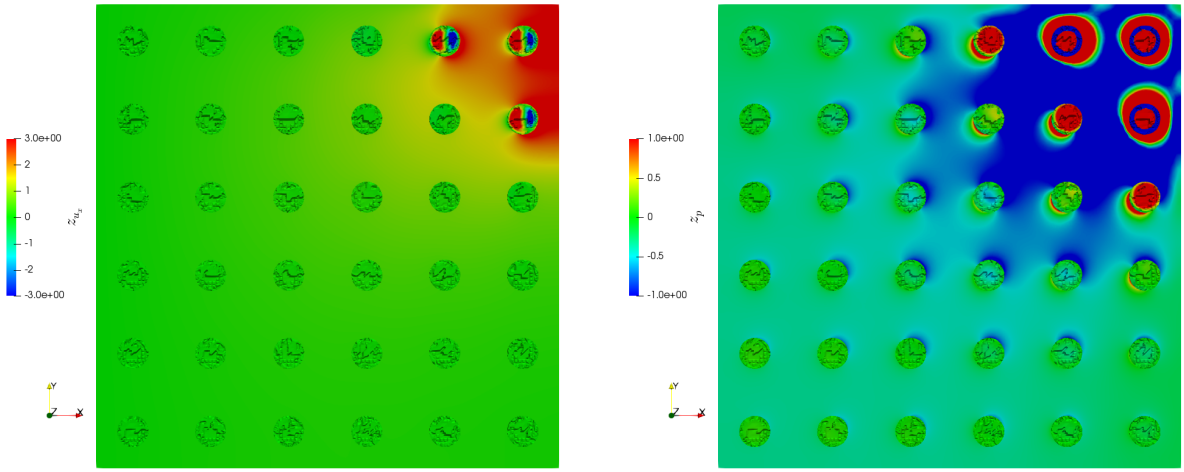

Figure 13: The x-component of the adjoint displacement solution (left), and the pressure component of the adjoint solution (right).

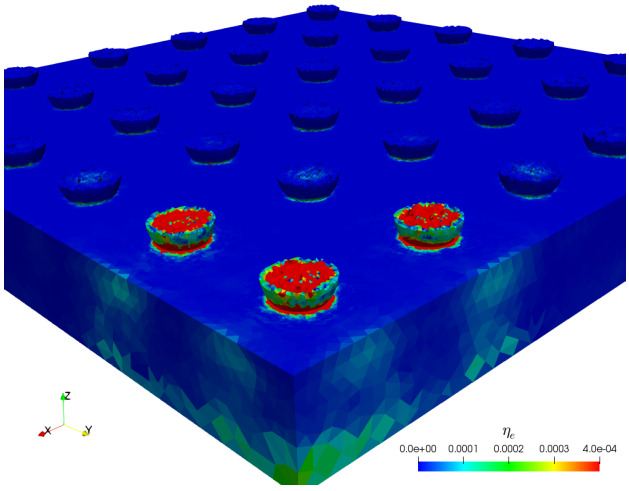

Figure 14: The spatial distribution of errors as computed by adjoint-based error estimation for the solder joint array.

define the QoI sub-domain are heavily refined, as expected. Additionally, notice that Figure 15 demonstrates that there is refinement at the left-most solder joint, which is not included in the geometric definition of the QoI. The adjoint-based error estimation procedure indicates that mesh must be refined in additional areas to accurately assess the QoI.

Figure 17 demonstrates the convergence history of the error in the functional QoI, as defined by the difference of the QoI obtained via Richardson extrapolation and the QoI approximated by the finite element solution. We compare the convergence for two adaptive schemes, one achieved by successive uniform refinements of the mesh and the other achieved by adjoint-based error estimation. After 4 adaptive iterations, the adjoint-based adaptive procedure achieves nearly the same degree of accuracy as the uniform refinement procedure with two orders of magnitude fewer degrees of freedom.

Finally, we remark that automated parallel adaptive workflows have been developed in reference [9]. As an avenue for future investigation, adjoint-based error estimation could be folded into these automated workflows. In particular, an automated primal analysis could be used to inform the actual selection of the QoI itself, which could then be accurately assessed using adjoint-based error estimation.

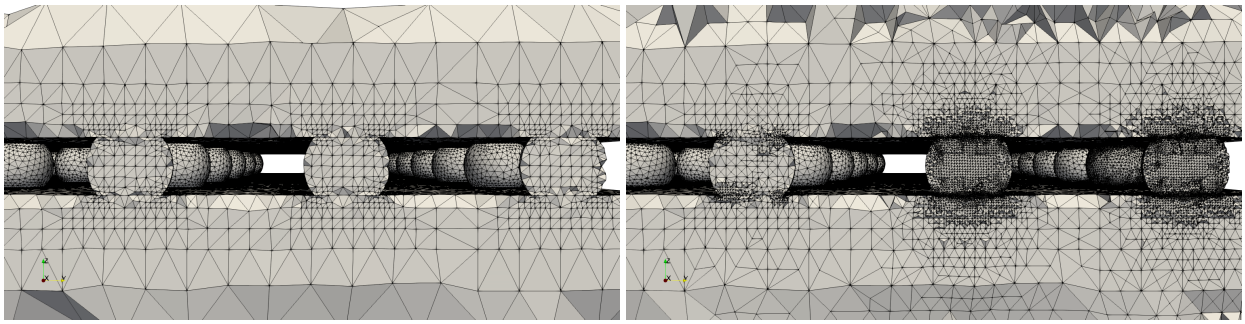

Figure 15: Cross-sectional view of the initial mesh for the solder joint geometry (left) and the final adapted mesh (right).

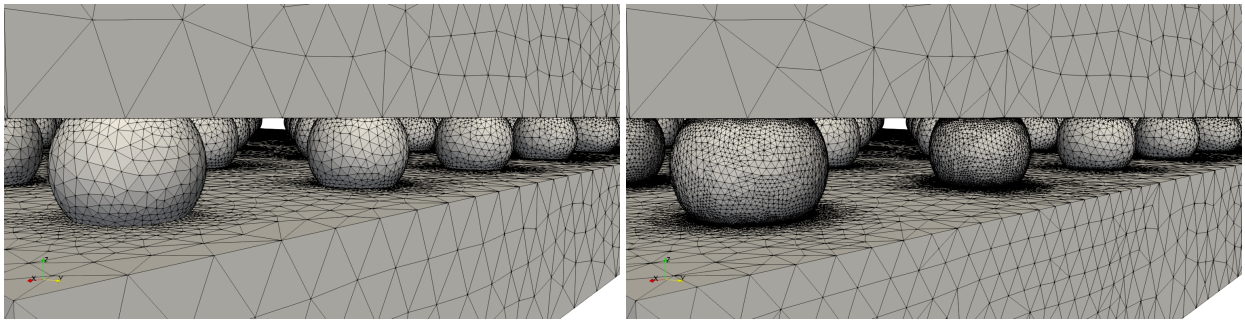

Figure 16: The initial mesh for the solder joint geometry (left) and the final adapted mesh (right).

11. Conclusions

In this work, we have developed an automated approach for adjoint-based error estimation and mesh adaptation for execution on parallel machines. We have developed this approach to be applicable to both Galerkin and stabilized finite element methods. To realize this approach, we have extended the concept of *template-based generic programming* for PDE models to include the automatic localization of error contributions using a partition of unity-based localization approach. We have demonstrated that this approach is effective for a variety of example applications, including nonlinear elasticity and elastoplasticity.

12. Acknowledgements

The authors acknowledge the support of IBM Corporation in the performance of this research. The computing resources of the Center for Computational Innovations at Rensselaer Polytechnic Institute are also acknowledged. The development of tools used in this work was partly supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under award DE-SC00066117 (FASTMath SciDAC Institute). The authors would like to thank Li Dong for providing the discrete geometric model used for the microglial cell example and Max Bloomfield for the geometric model used for the solder ball example.

References

[1] Mark Ainsworth and J. Tinsley Oden. *A Posteriori Error Estimation in Finite Element Analysis.* John Wiley & Sons, Ltd, Hoboken, NJ, USA, 2011.

[2] Frédéric Alauzet, Xiangrong Li, E Seegyoung Seol, and Mark S Shephard. Parallel anisotropic 3D mesh adaptation by mesh modification. *Eng. with Comp.*, 21(3):247–258, Jan. 2006.

[3] Ivo Babuška and Anthony Miller. The post-processing approach in the finite element method, Part 1: Calculation of displacements, stresses and other higher derivatives of the displacements. *Int. J. for Numerical Methods. in Eng.*, 20(6):1085–1109, Jun. 1984.

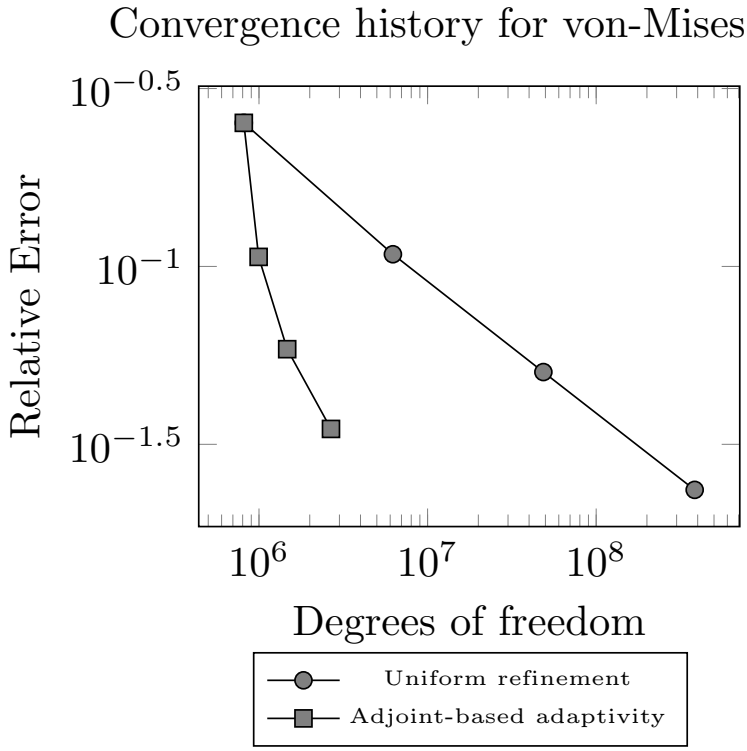

Figure 17: Error convergence histories for the solder joint example problem with the integrated von-Mises stress QoI.

[4] Ivo Babuška and Anthony Miller. The post-processing approach in the finite element method, Part 2: The calculation of stress intensity factors. *Int. J. for Numerical Methods. in Eng.*, 20(6):1111–1129, Jun. 1984.

[5] Ivo Babuška and Anthony Miller. The post-processing approach in the finite element method, Part 3: A posteriori error estimates and adaptive mesh selection. *Int. J. for Numerical Methods. in Eng.*, 20(12):2311–2324, Dec. 1984.

[6] Wolfgang Bangerth. Deal ii step 14, 2017.

[7] Eric Bavier, Mark Hoemmen, Sivasankaran Rajamanickam, and Heidi Thornquist. Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems. *Scientific Programming*, 20(3):241–255, Jan. 2012.

[8] Roland Becker and Rolf Rannacher. An optimal control approach to a posteriori error estimation in finite element methods. *Acta Numerica*, 10:1–102, May. 2001.

[9] Max O Bloomfield, Zhen Li, Brian Granzow, Daniel A Ibanez, Assad A Oberai, Glen A Hansen, Xiao Hu Liu, and Mark S Shephard. Component-based workflows for parallel thermomechanical analysis of arrayed geometries. *Eng. with Comput.*, 33(3):509–517, Jul. 2017.

[10] Ramzy Boussetta, Thierry Coupez, and Lionel Fourment. Adaptive remeshing based on a posteriori error estimation for forging simulation. *Comput. Methods in Appl. Mechanics and Eng.*, 195(48):6626–6645, Oct. 2006.

[11] Carsten Burstedde, Omar Ghattas, Georg Stadler, Tiankai Tu, and Lucas C Wilcox. Parallel scalable adjoint-based adaptive solution of variable-viscosity stokes flow problems. *Comput. Methods in Appl. Mechanics and Eng.*, 198(21):1691–1700, May. 2009.

[12] Eric C Cyr, John Shadid, and Tim Wildey. Approaches for adjoint-based a posteriori analysis of stabilized finite element methods. *SIAM J. on Scientific Comput.*, 36(2):A766–A791, Apr. 2014.

[13] Gerrett Diamond, Cameron W. Smith, and Mark S. Shephard. Dynamic load balancing of massively parallel unstructured meshes. In *Proc. of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, Denver, CO, USA*. Denver, CO, USA, Nov. 2017.

[14] Li Dong and Assad A Oberai. Recovery of cellular traction in three-dimensional nonlinear hyperelastic matrices. *Comput. Methods in Appl. Mechanics and Eng.*, 314:296–313, Feb. 2017.

[15] K Eriksson, D Estep, P Hansbo, and C Johnshon. *Computational Differential Equations*. Cambridge Univ. Press, New York, NY, USA, 2 edition, 1996.

[16] Krzysztof J Fidkowski. Output error estimation strategies for discontinuous galerkin discretizations of unsteady convection-dominated flows. *Int. J. for Numerical Methods in Eng.*, 88(12):1297–1322, May. 2011.

[17] Krzysztof J Fidkowski and David L Darmofal. Review of output-based error estimation and mesh adaptation in computational fluid dynamics. *AIAA J.*, 49(4):673–694, Apr. 2011.
[18] S. S. Ghorashi and T. Rabczuk. Goal-oriented error estimation and mesh adaptivity in 3D elastoplasticity problems. *Int. J. of Fracture*, 203:3–19, Jan. 2017.
[19] S Sh Ghorashi, J Amani, AS Bagherzadeh, and T Rabczuk. Goal-oriented error estimation and mesh adaptivity in three-dimensional elasticity problems. In *WCCM XI-ECCM V-ECFD VI, Barcelona, Spain*, Barcelona, Spain, Jul. 2014.
[20] Michael B. Giles and Niles A. Pierce. *Chapter 2 - Adjoint Error Correction for Integral Outputs*, pages 47–95. Springer, Berlin, Germany, 2016.
[21] Octavio Andres González-Estrada, E Nadal, JJ Ródenas, Pierre Kerfriden, Stéphane Pierre-Alain Bordas, and FJ Fuenmayor. Mesh adaptivity driven by goal-oriented locally equilibrated superconvergent patch recovery. *Comput. Mechanics*, 53(5):957–976, May. 2014.
[22] Brian N. Granzow. Goal GitHub Repository, 2017.
[23] Brian N Granzow, Assad A Oberai, and Mark S Shephard. Adjoint-based error estimation and mesh adaptation for stabilized finite deformation elasticity. submitted for publication.
[24] Brian N Granzow, Mark S Shephard, and Assad A Oberai. Output-based error estimation and mesh adaptation for variational multiscale methods. *Comput. Methods in Appl. Mechanics and Eng.*, 322:441–459, Aug. 2017.
[25] Thomas Grätsch and Klaus-Jürgen Bathe. A posteriori error estimation techniques in practical finite element analysis. *Comput. & Structures*, 83(4-5):235–265, Dec. 2005.
[26] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Soc. for Ind. & Appl. Math., Philadelphia, PA, USA, 2 edition, 2008.
[27] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, and et al. An overview of the Trilinos project. *ACM Trans. on Math. Software*, 31(3):397–423, Sep. 2005.
[28] Michael A Heroux and James M Willenbring. A new overview of the Trilinos project. *Scientific Programming*, 20(2):83–88, Mar. 2012.
[29] Dan Ibanez and Mark S Shephard. Modifiable array data structures for mesh topology. *SIAM J. on Scientific Comput.*, 39(2):C144–C161, Apr. 2017.
[30] Daniel A Ibanez, E Seegyoung Seol, Cameron W Smith, and Mark S Shephard. PUMI: Parallel unstructured mesh infrastructure. *ACM Trans. on Math. Software*, 42(3):17–45, Jun. 2016.
[31] Fredrik Larsson, Peter Hansbo, and Kenneth Runesson. Strategies for computing goal-oriented a posteriori error measures in non-linear elasticity. *Int. J. for Numerical Methods in Eng.*, 55(8):879–894, Aug. 2002.
[32] Xiangrong Li, Mark S Shephard, and Mark W Beall. 3D anisotropic mesh adaptation by mesh modification. *Comput. Methods in Appl. Mechanics and Eng.*, 194(48):4915–4950, Nov. 2005.
[33] Zhen Li, Max O Bloomfield, and Assad A Oberai. Simulation of finite-strain inelastic phenomena governed by creep and plasticity. *Comput. Mechanics*. to be published.
[34] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Springer, Heidelberg, Germany, 2012.
[35] Marian Nemec and Michael J Aftosmis. Adjoint error estimation and adaptive refinement for embedded-boundary Cartesian meshes. In *18th AIAA Computational Fluid Dynamics Conf., Miami, FL, USA*, Miami, FL, USA, Jun. 2007.
[36] John Tinsley Oden and Serge Prudhomme. Goal-oriented error estimation and adaptivity for the finite element method. *Comput. & Math. with Applications*, 41(5-6):735–756, Mar. 2001.
[37] Roger P Pawlowski, Eric T Phipps, and Andrew G Salinger. Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, Part I: Template-based generic programming. *Scientific Programming*, 20(2):197–219, Apr. 2012.
[38] Roger P Pawlowski, Eric T Phipps, Andrew G Salinger, Steven J Owen, Christopher M Siefert, and Matthew L Staten. Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, Part II: Application to partial differential equations. *Scientific Programming*, 20(3):327–345, Jul. 2012.
[39] Eric Phipps and Roger Pawlowski. Efficient expression templates for operator overloading-based automatic differentiation. In *Recent Advances in Algorithmic Differentiation*, pages 309–319. Springer, Berlin, Germany, 2012.
[40] Andrey Prokopenko, Jonathan J. Hu, Tobias A. Wiesner, Christopher M. Siefert, and Raymond S. Tuminaro. MueLu users guide 1.0. Technical Report SAND2014-18874, Sandia Nat. Lab., Albuquerque, NM, USA, Oct. 2014.
[41] Serge Prudhomme and J Tinsley Oden. On goal-oriented error estimation for elliptic problems: Application to the control of pointwise errors. *Comput. Methods in Appl. Mechanics and Eng.*, 176(1-4):313–331, Jul. 1999.
[42] E Rabizadeh, A Saboor Bagherzadeh, and T Rabczuk. Adaptive thermo-mechanical finite element formulation based on goal-oriented error estimation. *Comput. Materials Science*, 102:27–44, May. 2015.
[43] Binoj Ramesh and Antoinette M Maniatty. Stabilized finite element formulation for elastic–plastic finite deformations. *Comput. Methods in Appl. Mechanics and Eng.*, 194(6):775–800, Feb. 2005.
[44] Rolf Rannacher and F-T Suttmeier. A feed-back approach to error control in finite element methods: Application to linear elasticity. *Comput. Mechanics*, 19(5):434–446, Apr. 1997.
[45] Rolf Rannacher and F-T Suttmeier. A posteriori error control in finite element methods via duality techniques: Application to perfect plasticity. *Comput. Mechanics*, 21(2):123–133, Mar. 1998.
[46] Rolf Rannacher and Franz-Theo Suttmeier. A posteriori error estimation and mesh adaptation for finite element models in elasto-plasticity. *Comput. Methods in Appl. Mechanics and Eng.*, 176(1-4):333–361, Jul. 1999.
[47] Lewis Fry Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Trans. of the Royal Society of London*, 210:307–357, Jan. 1911.

[48] Thomas Richter and Thomas Wick. Variational localizations of the dual weighted residual estimator. *J. of Comput. and Appl. Math.*, 279:192–208, May. 2015.

[49] Marie E Rognes and Anders Logg. Automated goal-oriented error control I: Stationary variational problems. *SIAM J. on Scientific Comput.*, 35(3):C173–C193, May. 2013.

[50] Andrew G Salinger, Roscoe A Bartett, Quishi Chen, Xujiao Gao, Glen Hansen, Irina Kalashnikova, Alejandro Mota, Richard P Muller, Erik Nielsen, Jakob Ostien, et al. Albany: A component-based partial differential equation code built on trilinos. Technical Report SAND2013-8430J, Sandia Nat. Lab., Albuquerque, NM, USA, Nov. 2013.

[51] Juan C Simo and Thomas JR Hughes. *Computational Inelasticity*. Springer, New York, NY, USA, 2006.

[52] Cameron W Smith, Brian Granzow, Dan Ibanez, Onkar Sahni, Kenneth E Jansen, and Mark S Shephard. In-memory integration of existing software components for parallel adaptive unstructured mesh workflows. In *Proc. of the XSEDE16 Conf. on Diversity, Big Data, and Science at Scale, Miami, FL, USA*. Miami, FL, USA, Jul. 2016.

[53] Cameron W. Smith, Michel Rasquin, Dan Ibanez, Kenneth E. Jansen, and Mark S. Shephard. Improving unstructured mesh partitions for multiple criteria using mesh adjacencies. *SIAM J. Scientific Comput.* to be published.

[54] Erwin Stein, Marcus Rüter, and Stephan Ohnimus. Error-controlled adaptive goal-oriented modeling and finite element approximations in elasticity. *Comput. Methods in Appl. Mechanics and Eng.*, 196(37):3598–3613, Aug. 2007.

[55] Cedric Taylor and Paul Hood. A numerical solution of the Navier-Stokes equations using the finite element technique. *Comput. & Fluids*, 1(1):73–100, Jan. 1973.

[56] Irina K Tezaur, Mauro Perego, Andrew G Salinger, Raymond S Tuminaro, and Stephen F Price. Albany/FELIX: A parallel, scalable and robust, finite element, first-order Stokes approximation ice sheet solver built for advanced analysis. *Geoscientific Model Development*, 8(4):1197–1220, Apr. 2015.

[57] David A Venditti and David L Darmofal. Adjoint error estimation and grid adaptation for functional outputs: Application to quasi-one-dimensional flow. *J. of Comput. Physics*, 164(1):204–227, Oct. 2000.

[58] David A. Venditti and David L. Darmofal. Grid adaptation for functional outputs: Application to two-dimensional inviscid flows. *J. of Comput. Physics*, 176(1):40–69, Feb. 2002.

[59] David A. Venditti and David L. Darmofal. Anisotropic grid adaptation for functional outputs: Application to two-dimensional viscous flows. *J. Comput. Phys.*, 187(1):22–46, May. 2003.

[60] Rüdiger Verfürth. A posteriori error estimation and adaptive mesh-refinement techniques. *J. of Comput. and Appl. Math.*, 50(1-3):67–83, May. 1994.

[61] JP Whiteley and SJ Tavener. Error estimation and adaptivity for incompressible hyperelasticity. *Int. J. for Numerical Methods. in Eng.*, 99(5):313–332, Apr. 2014.