ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

The TAO Linearly-Constrained Augmented Lagrangian Method for PDE-Constrained Optimization[1]

Evan Gawlik, Todd Munson, Jason Sarich, and Stefan M. Wild

Mathematics and Computer Science Division

Preprint ANL/MCS-P2003-0112

January 2012

[1]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439. This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

The TAO Linearly-Constrained Augmented Lagrangian Method for PDE-Constrained Optimization*

Evan Gawlik* Todd Munson† Jason Sarich† Stefan M. Wild†

Abstract

This report describes a linearly-constrained augmented Lagrangian method for solving optimization problems with partial differential equation constraints. This method computes two types of directions: a Newton direction to reduce the constraint violation and reduced-space directions to improve the augmented Lagrangian merit function. The reduced-space directions are computed from limited-memory quasi-Newton approximations to the reduced Hessian matrix. This method requires a minimal amount of information from the user—only function, gradient, and Jacobian evaluations—yet can obtain good performance. Strong scaling results are presented for some model test problems on high-performance architectures, indicating that the code scales well provided the code for the PDE constraints scales well.

1 Introduction

Optimization problems with simulation constraints are fundamental to many scientific grand challenges, ranging from the design of nanophotonic devices [6], to controlling the coil currents in a fusion reactor to avoid instabilities [7], to optimizing the performance of both existing accelerators and future lepton collider accelerators [5]. When the underlying partial differential equation (PDE) constraints are discretized, such problems can be posed as finite-dimensional nonlinear optimization problems of the form

$$\begin{array}{ll} \min\limits_{u,v} & f(u,v) \\ \text{subject to} & g(u,v) = 0, \end{array} \tag{1}$$

where the state variable $u \in \mathbb{R}^{n_u}$ is the solution to a discretized PDE parameterized by the design variable $v \in \mathbb{R}^{n_v}$ defined by $g : \mathbb{R}^{n_u+n_v} \mapsto \mathbb{R}^{n_u}$ and $f : \mathbb{R}^{n_u+n_v} \mapsto \mathbb{R}$ is the objective function. Our goal is to develop methods for solving these problems that exploit

[1]Stanford University, Institute for Computational and Mathematical Engineering, 75 Via Ortega, Stanford, CA 94305-4042, `egawlik@stanford.edu`

[2]Argonne National Laboratory, Mathematics and Computer Science Division, 9700 S. Cass Ave., Argonne, IL 60439, `{tmunson,sarich,wild}@mcs.anl.gov`

*This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, and by a DOE Computational Science Graduate Fellowship to the lead author under grant number DE-FG02-97ER25308.

the structure of the underlying PDE constraint, require a minimal amount of derivative information, and can use the iterative methods and preconditioners developed for solving the PDE.

Naively, any PDE-constrained optimization problem of the form (1) can be reformulated as an unconstrained optimization problem in n_v variables by treating the state variables u as functions of the design variables v:

$$\min_v f(g^{-1}(v), v).$$

Such an approach, however, is impractical for large problems because it requires that the nonlinear PDE be solved for each evaluation of the objective function and its derivatives.

Alternatively, full-space methods can be derived by writing down the first-order optimality conditions—a $(2n_u + n_v)$-dimensional system of nonlinear equations in u, v, and the Lagrange multipliers—and solving them with an iterative method. Haber and Hanson [9], for example, apply a Gauss-Newton method for this purpose, using a flexible GMRES routine to solve the linear systems arising at each outer iteration. Biros and Ghattas [3, 4] employ Newton's method to solve the KKT system, where each linear solve is preconditioned with a quasi-Newton, reduced-space method. Their algorithm exhibits optimal scaling with respect to problem size in the sense that, under certain circumstances, the number of outer iterations taken to converge is independent of the mesh resolution. Being a pure Newton method, however, it requires the Hessian of the Lagrangian. Other full-space methods [13] take similar approaches, invoking Krylov solvers at each outer iteration of a Newton or quasi-Newton solver.

We present here a matrix-free, linearly-constrained augmented Lagrangian method that requires a minimal amount of information from the user: function evaluations and first derivatives and linearized forward and adjoint solves. Most of this information is readily available from simulations that employ Newton's method to solve their PDE. In contrast to other methods [1, 3, 4, 13], we do not require that the user provide second-order information, and we do not need an iterative method and preconditioner for the full system of optimality conditions. The method closely resembles the quasi-Newton, reduced SQP method described in [3, Algorithm 3]. The algorithms, however, differ in two key respects. First, our method searches along two types of directions—a Newton direction to reduce the constraint violations and reduced-space directions to improve the augmented Lagrangian merit function—and performs independent line searches along each. Second, we choose to minimize an augmented Lagrangian merit function rather than the pure Lagrangian. By separating feasibility and optimality steps, we can seamlessly enter a feasibility restoration phase in which we approximately solve the PDE constraint with a globalized Newton method for fixed design variables.

As will be shown, the primary expense of our algorithm is associated with solving linearized forward and adjoint systems of equations; a successful implementation will require a small number of these linear solves. We expect and recommend that a preconditioned iterative method be used to solve the systems of equations with a convergence tolerance specified by the optimization routine. The iterative method and preconditioner need not be the same for the linearized forward and adjoint systems. In particular, if one is using a left-preconditioned iterative method for the forward problem, then one can apply a right-preconditioned iterative method to the adjoint problem in which the adjoint preconditioner is the transpose of the forward preconditioner. Furthermore, the procedure supplied should take advantage of user knowledge regarding the partial differential equation being solved. For example, certain finite-element approximations produce symmetric systems of equations; the iterative method supplied should take advantage of this structure. All specialized knowledge for the application is encapsulated in the selected linear solver and preconditioners and is independent of the optimization algorithm.

Our linearly-constrained augmented Lagrangian method for solving PDE-constrained optimization problems is released as part of the open-source Toolkit for Advanced Optimization (TAO) [14]. TAO focuses on software for the solution of large-scale optimization problems on high-performance computers. The design philosophy strongly emphasizes the reuse of external tools where appropriate, enabling a bidirectional connection to lower-level linear algebra support. Our design decisions are motivated by the challenges inherent in the use of large-scale distributed-memory architectures. In particular, TAO is built on top of PETSc [12, 2], a package commonly used by the developers of PDE simulations that provides many parallel sparse-matrix formats, Krylov subspace methods, and preconditioners. PETSc is extensible so that new matrix formats, such as those suitable for GPUs, iterative methods, and preconditioners, can be readily supplied by the user.

After presenting notation, we describe our linearly-constrained augmented Lagrangian algorithm in Section 2. This method makes consecutive steps along first a Newton-like direction and then reduced-space directions with respect to the linearized constraints. Since the user is required to provide only first-order derivatives, all Hessian information is obtained from a limited-memory, quasi-Newton approximation. In its most basic form, the algorithm requires two forward solves and two adjoint solves per iteration. Section 3 briefly describes the implementation of the method in TAO. Section 4 then presents numerical results on a collection of test problems, including the parameter estimation problems in [9]. Our results confirm that the dominant computational cost is due to matrix-vector products associated with the linear solves. We also illustrate the effects of increasing the problem size, changing the linear solver tolerance, and varying the number of solves per iterations. We present both weak and strong scaling results indicating that the method scales well provided the code for the PDE constraint scales well.

Unless otherwise noted, we employ the Euclidean norm $\|\cdot\| = \|\cdot\|_2$ throughout. We also assume the linearized forward operator, $\nabla_u g(u, v)$, is invertible for all u and v and is uniformly bounded above and below in an appropriate norm. This assumption is satisfied for many real-world PDE constraints.

2 Linearly-constrained Augmented Lagrangian Method

Given a discretized problem (1) with n_u state variables u and n_v design variables v, we denote the Lagrange multipliers on the constraint $g : \mathbb{R}^{n_u+n_v} \mapsto \mathbb{R}^{n_u}$ by $y \in \mathbb{R}^{n_u}$.

Given a current iterate (u_k, v_k, y_k), the kth iteration of a linearly-constrained augmented Lagrangian method approximately solves the optimization problem

$$\begin{array}{ll} \min\limits_{u,v} & \tilde{f}_k(u,v) \\ \text{subject to} & A_k(u-u_k) + B_k(v-v_k) + g_k = 0, \end{array} \tag{2}$$

where $A_k = \nabla_u g(u_k, v_k)$, $B_k = \nabla_v g(u_k, v_k)$, $g_k = g(u_k, v_k)$, and

$$\tilde{f}_k(u,v) = f(u,v) - g(u,v)^T y_k + \frac{\rho_k}{2}\|g(u,v)\|^2$$

is the augmented Lagrangian merit function with penalty parameter $\rho_k \geq 0$. The current objective function gradient is given by $a_k = \nabla_u f(u_k, v_k)$ and $b_k = \nabla_v f(u_k, v_k)$.

We solve this optimization problem in two stages. In the first stage a Newton direction is computed, and a feasible point for the linear constraints is found. In the second stage reduced-space directions are computed that maintain feasibility with respect to the linearized constraints and improve the augmented Lagrangian merit function.

2.1 Phase I: Newton Step

The Newton direction is obtained by fixing the design variables at their current value v_k and solving the linearized constraint for the state variables. In particular, we approximately solve the (forward) system of equations

$$A_k d_u = -g_k$$

to obtain a direction d_u. Because the system is only approximately solved, the direction satisfies the equation

$$A_k(d_u + r_k) = -g_k, \tag{3}$$

where r_k is the residual. We need a direction that provides sufficient descent for the PDE constraint merit function

$$\frac{1}{2}\|g(u, v_k)\|^2, \tag{4}$$

and hence we require that

$$g_k^T A_k d_u \leq -\epsilon_1 \|d_u\|^{2+\epsilon_2}, \tag{5}$$

where $\epsilon_1 > 0$ and $\epsilon_2 > 0$ are parameters. If d_u does not provide descent for the merit function (4), we enter a truncated feasibility restoration phase to satisfy the PDE constraint for fixed design variables. In particular, we apply a globalized Newton method to solve

$$g(u, v_k) = 0$$

and stop this restoration phase when the Newton direction satisfies the descent criterion (5). If the PDE constraints are well behaved, then this descent criterion will be satisfied once we enter the domain of local fast convergence for the globalized Newton method.

Given that the Newton-like direction d_u is a descent direction for (4), we would like to choose parameters for the augmented Lagrangian merit function so that d_u is also a descent direction for this merit function. In particular, we want to satisfy the inequality

$$d_u^T \left(a_k - A_k^T y_k + \rho_k A_k^T g_k\right) \le -\epsilon_1 \|d_u\|^{2+\epsilon_2}. \tag{6}$$

Given multipliers y_k and the penalty parameter ($\rho_k = \rho_{k-1}$) from the previous iterate, we check this condition for d_u. If condition (6) is not satisfied, then $\|d_u\| > 0$ and $g_k^T A_k d_u < 0$, and we consider two cases. In the first case, we choose ρ_k so that

$$\rho_k = \min\left\{ \frac{d_u^T \left(A_k^T y_k - a_k\right) - \epsilon_1 \|d_u\|^{2+\epsilon_2}}{g_k^T A_k d_u}, \bar{\rho} \right\}, \tag{7}$$

where $\bar{\rho} > 1$ is a parameter bounding the magnitude of ρ_k. If condition (6) is still not satisfied with the updated value of ρ_k, we then calculate a new multipliers estimate by solving the system

$$A_k^T y_k = a_k$$

for y_k. This system need not be solved exactly, and we can stop the iterative method as soon as condition (6) is satisfied. In particular, we can stop the iterative method when

$$\|A_k^T y_k - a_k\| \le (\rho_k - 1)\, \epsilon_1 \|d_u\|^{1+\epsilon_2}.$$

In this case, we have

$$\begin{array}{rcl} d_u^T(a_k - A_k^T y_k + \rho_k A_k^T g_k) & \le & \|d_u\| \|A_k^T y_k - a_k\| \cos\theta - \rho_k \epsilon_1 \|d_u\|^{2+\epsilon_2} \\ & \le & \|d_u\| \|A_k^T y_k - a_k\| |\cos\theta| - \rho_k \epsilon_1 \|d_u\|^{2+\epsilon_2} \\ & \le & (\rho_k - 1)\epsilon_1 \|d_u\|^{2+\epsilon_2} - \rho_k \epsilon_1 \|d_u\|^{2+\epsilon_2} \\ & = & -\epsilon_1 \|d_u\|^{2+\epsilon_2}, \end{array}$$

where θ denotes the angle between d_u and $a_k - A_k^T y_k$. Both condition (6) and the possible update of ρ_k in (7) require only inner products, since the matrix-vector products $A_k^T y_k$ and $A_k^T g_k$ and evaluations g_k and a_k are already available. Calculation of new multipliers estimates y_k occurs rarely and requires an approximate solution to the linearized adjoint problem.

We then find α to approximately minimize the augmented Lagrangian function along the Newton-like direction,

$$\min_{\alpha \ge 0} \tilde{f}_k(u_k + \alpha d_u, v_k).$$

We can enforce either the sufficient decrease condition or the Wolfe conditions during the search procedure. The intermediate point

$$\begin{array}{rcl} u_{k,0} & = & u_k + \alpha_k d_u \\ v_{k,0} & = & v_k \end{array} \tag{8}$$

satisfies the linear constraint

$$A_k(u_{k,0} - u_k + \alpha r_k) + B_k(v_{k,0} - v_k) + \alpha_k g_k = 0.$$

2.2 Phase II: Modified Reduced-Space Steps

In the second phase, we compute reduced-space steps for the linearly-constrained optimization problem

$$\begin{array}{ll} \displaystyle\min_{d_u, d_v} & \tilde{f}_k(u_k + d_u,\, v_k + d_v) \\ \text{subject to} & A_k(d_u + \alpha r_k) + B_k d_v + \alpha_k g_k = 0, \end{array}$$

corresponding to (2) after a change of variables. Making the reduction

$$d_u = -A_k^{-1}(B_k d_v + \alpha_k g_k) - \alpha r_k,$$

we obtain the unconstrained problem

$$\min_{d_v} \tilde{f}_k\left(u_k - A_k^{-1}(B_k d_v + \alpha_k g_k) - \alpha_k r_k,\, v_k + d_v\right).$$

Since the Newton-like direction exactly satisfies (3), the intermediate point defined by (8) gives rise to the equivalent problem

$$\min_{d_v} \tilde{f}_k\left(u_{k,0} - A_k^{-1} B_k d_v,\, v_{k,0} + d_v\right). \tag{9}$$

We approximately solve the reduced-space problem (9) by applying one or more steps of a limited-memory quasi-Newton method. We obtain a direction d_v by solving the quadratic problem

$$\min_{d_v} \frac{1}{2} d_v^T \tilde{H}_{k,i} d_v + \tilde{g}_{k,i}^T d_v,$$

where $\tilde{H}_{k,i}$ is a (positive-definite) limited-memory quasi-Newton approximation to the reduced Hessian matrix and $\tilde{g}_{k,i}$ is the reduced gradient

$$\tilde{g}_{k,i} \;=\; \nabla_v \tilde{f}_k\left(u_{k,i},\, v_{k,i}\right) - B_k^T A_k^{-T} \nabla_u \tilde{f}_k\left(u_{k,i},\, v_{k,i}\right). \tag{10}$$

The reduced gradient is thus obtained from one linearized adjoint solve

$$A_k^T y_{k,i} = \nabla_u \tilde{f}_k\left(u_{k,i},\, v_{k,i}\right) \tag{11}$$

and some linear algebra

$$\tilde{g}_{k,i} = \nabla_v \tilde{f}_k\left(u_{k,i},\, v_{k,i}\right) - B_k^T y_{k,i}.$$

Because the limited-memory quasi-Newton Hessian approximation we use is positive definite and we can easily apply its inverse to vectors, we obtain the direction

$$d_v = -\tilde{H}_{k,i}^{-1} \tilde{g}_{k,i}.$$

We then want to perform a line search along the direction d_v to obtain sufficient reduction in the augmented Lagrangian merit function. A reduced-space line search could require calculating the reduced gradient at each trial point, requiring a solve with the linearized adjoint. Therefore, we instead recover the full-space direction from one linearized forward solve

$$A_k d_u = -B_k d_v \tag{12}$$

and approximately minimize the augmented Lagrangian merit function along this direction:

$$\min_{\beta \geq 0} \quad \tilde{f}_k(u_{k,i} + \beta d_u, v_{k,i} + \beta d_v).$$

The solves in (11) and (12) can be done inexactly. However, we require that the full-space direction be a descent direction for the augmented Lagrangian merit function,

$$d_u^T \nabla_u \tilde{f}_k\,(u_{k,i},\, v_{k,i}) + d_v^T \nabla_v \tilde{f}_k\,(u_{k,i},\, v_{k,i}) \leq -\epsilon_1 \|(d_u, d_v)\|^{2+\epsilon_2}.$$

If the direction computed is not a sufficient descent direction, we revert, at no additional computational expense, to the steepest descent direction.

We enforce the Wolfe conditions (see, e.g., [11]) during the search procedure and obtain the new point

$$\begin{aligned} u_{k,i+1} &= u_{k,i} + \beta_{k,i} d_u \\ v_{k,i+1} &= v_{k,i} + \beta_{k,i} d_v. \end{aligned}$$

The reduced gradient at the new point is computed from

$$\begin{aligned} A_k^T y_{k,i+1} &= \nabla_u \tilde{f}_k(u_{k,i+1}, v_{k,i+1}) \\ \tilde{g}_{k,i+1} &= \nabla_v \tilde{f}_k(u_{k,i+1}, v_{k,i+1}) - B_k^T y_{k,i+1}. \end{aligned} \tag{13}$$

The vectors $v_{k,i}$, $v_{k,i+1}$, $\tilde{g}_{k,i}$, and $\tilde{g}_{k,i+1}$ are used to update $\tilde{H}_{k,i}$ to obtain the limited-memory quasi-Newton approximation to the reduced Hessian matrix used in the next iteration. The update is skipped if it cannot be performed.

We keep iterating for a fixed number of steps i or until the norm of the reduced gradient is sufficiently small and we have solved the subproblem. Our default strategy is to compute only one reduced-space step, but we consider applying multiple steps in the numerical results. At the end of the iterations, we set $u_{k+1} = u_{k,i_k+1}$, $v_{k+1} = v_{k,i_k+1}$ and $y_{k+1} = y_{k,i_k+1}$ in preparation for the next major iteration, where i_k is the number of reduced-space steps performed during major iteration k. The Hessian approximation is also reused from one major iteration to the next, $\tilde{H}_{k+1,0} = \tilde{H}_{k,i_k+1}$.

2.3 Summary and Computational Cost

In summary, the algorithm is written as follows:

1. Given initial points u_0, v_0, and y_0, an initial Hessian approximation $\tilde{H}_{0,0}$, and parameters $\bar{\rho} > \rho_0 > 1$, $\epsilon_1 > 0$, and $\epsilon_2 > 0$.

2. For $k = 0, \ldots$

    (a) Evaluate functions, gradients, and Jacobians at u_k and v_k.

    (b) If the first-order optimality conditions are satisfied or an iteration limit is reached, then stop.

    (c) Compute the Newton direction by approximately solving

$$A_k d_u = -g_k.$$

(d) If $g_k^T A_k d_u > -\epsilon_1 \|d_u\|^{2+\epsilon_2}$, then enter feasibility restoration.

(e) Otherwise choose ρ_k and/or y_k so that condition (6) is satisfied and d_u is a descent direction for the augmented Lagrangian merit function.

(f) Perform a line search to determine steplength α_k by solving the one-dimensional optimization problem

$$\min_{\alpha \geq 0} \tilde{f}_k(u_k + \alpha d_u, v_k).$$

(g) Initialize $u_{k,0} = u_k + \alpha_k d_u$ and $v_{k,0} = v_k$.

(h) Compute the reduced gradient $\tilde{g}_{k,0}$ using equation (10).

(i) For $i = 0, \ldots$

  i. If the norm of the reduced gradient is within tolerances or the inner iteration limit is reached, then break; otherwise proceed to ii.

  ii. Compute the reduced-space direction,

$$d_v = -\tilde{H}_{k,i} \tilde{g}_{k,i}.$$

  iii. Recover the full-space direction by solving the equation

$$A_k d_u = -B_k d_v.$$

  iv. Perform a line search to determine steplength $\beta_{k,i}$ by solving the one-dimensional optimization problem

$$\min_{\beta \geq 0} \tilde{f}_k(u_{k,i} + \beta d_u, v_{k,i} + \beta d_v).$$

  v. Update $u_{k,i+1} = u_{k,i} + \beta_{k,i} d_u$ and $v_{k,i+1} = v_{k,i} + \beta_{k,i} d_v$.

  vi. Compute reduced gradient $\tilde{g}_{k,i+1}$ using equation (10).

  vii. Calculate a new reduced Hessian approximation $\tilde{H}_{k,i+1}$ using the L-BFGS update formula.

(j) Update $u_{k+1} = u_{k,i_k+1}$, $v_{k+1} = v_{k,i_k+1}$ and $\tilde{H}_{k+1,0} = \tilde{H}_{k,i_k+1}$. Moreover, let the multiplier estimates $y_{k+1} = y_{k,i_k+1}$ from the computation of the reduced gradient.

The dominant computational cost of this method is associated with the linearized solves with the Jacobian A_k: one forward solve (3) to obtain the Newton direction per major iteration, one adjoint solve (11) to obtain the reduced gradient, one forward solve (12) to recover the full-space direction for each minor iteration, and one adjoint solve (13) per major iteration to complete the update of the Hessian approximation and obtain multiplier estimates.

Other substantial operations are associated with evaluating the Jacobian of g. Each iteration involves at least two Jacobian evaluations: one to obtain A_k and B_k to linearize the constraints and one to compute the gradient of the augmented Lagrangian merit function at the intermediate (and possibly additional trial) point(s). In our experience, this computational cost is negligible; the computation of the gradient of the augmented Lagrangian merit function requires only a single Jacobian-vector product.

3 Implementation

The linearly-constrained augmented Lagrangian algorithm described here is available in version 2.0 of the Toolkit for Advanced Optimization [14]. For these problems, the user needs to set routines for computing the objective function and its gradient, the constraints, and the Jacobian of the constraints with respect to the state and design variables. TAO also needs to know which variables in the solution vector correspond to state variables and which correspond to design variables.

The objective and gradient routines are set as for other TAO applications, with `TaoSet-ObjectiveRoutine()` and `TaoSetGradientRoutine()`. The user can also provide a fused objective function and gradient evaluation with `TaoSetObjectiveAndGradientRoutine()`. The input and output vectors include the combined state and design variables. Index sets for the state and design variables must be passed to TAO by using the function

```
TaoSetStateDesignIS(TaoSolver, IS, IS);
```

where the first IS is a PETSc IndexSet containing the indices of the state variables and the second IS corresponds to the design variables.

The routine that evaluates the constraint equations must have the form

```
PetscErrorCode EvaluateConstraints(TaoSolver,Vec,Vec,void*);
```

The first argument of this routine is a TAO solver object. The second argument is the variable vector at which the constraint function should be evaluated. The third argument is the vector of function values $g(x)$, and the fourth argument is a pointer to a user-defined context. This routine and the user-defined context should be set in the TAO solver with the

```
TaoSetConstraintsRoutine(TaoSolver,Vec,
                    PetscErrorCode (*)(TaoSolver,Vec,Vec,void*),
                    void*);
```

command. In this function, the first argument is the TAO solver object, the second argument a vector in which to store the constraints, the third argument is a function pointer to the routine for evaluating the constraints, and the fourth argument is a pointer to a user-defined context.

The Jacobian of $g(x)$ is a matrix such that each column contains the partial derivatives of $g(x)$ with respect to one variable. The evaluation of the Jacobian of c is performed by calling the

```
PetscErrorCode JacobianState(TaoSolver,Vec,Mat*,Mat*,Mat*,
                    MatStructure*, void*);
PetscErrorCode JacobianDesign(TaoSolver,Vec,Mat*,void*);
```

routines. In these functions, The first argument is the TAO solver object. The second argument is the variable vector at which to evaluate the Jacobian matrix, the third argument is the Jacobian matrix, and the last argument is a pointer to a user-defined context. The fourth and fifth arguments of the Jacobian evaluation with respect to the state variables are for providing PETSc matrix objects for the preconditioner and for applying the inverse of the state Jacobian, respectively. This inverse matrix may be `PETSC_NULL`, in which case TAO will use a PETSc Krylov subspace solver to solve the state system. These evaluation routines should be registered with TAO by using the

```
TaoSetJacobianStateRoutine(TaoSolver,Mat,Mat,Mat,
                    PetscErrorCode (*)(TaoSolver,Vec,Mat*,Mat*,
                    MatStructure*,void*), void*);
TaoSetJacobianDesignRoutine(TaoSolver,Mat,
                    PetscErrorCode (*)(TaoSolver,Vec,Mat*,void*),
                    void*);
```

routines. The first argument is the TAO solver object, and the second argument is the matrix in which the Jacobian information can be stored. For the state Jacobian, the third argument is the matrix that will be used for preconditioning, and the fourth argument is an optional matrix for the inverse of the state Jacobian. One can use `PETSC_NULL` for this inverse argument and let PETSc apply the inverse using a KSP method, but faster results may be obtained by manipulating the structure of the Jacobian and providing an inverse. The fifth argument is the function pointer, and the sixth argument is an optional user-defined context. Since no solve is performed with the design Jacobian, no preconditioner or inverse matrices are needed. For symmetric matrices, we exploit the symmetry in the forward and adjoint solves. Note that matrix-free versions are supported by changing the PETSc matrix type and implementing the necessary functions for applying the Jacobian and Jacobian transpose matrices.

To approximate the reduced Hessian $\tilde{H}_{k,i}$ of the augmented Lagrangian merit function, we use an L-BFGS scheme [11] that is rescaled at each iteration. The scalings are described in the TAO users manual [10].

4 Numerical Results

In this section we describe the performance results of the implementation. We focus on two aspects of the algorithm: its dependence on solver parameters (Section 4.2), and its scalability with respect to problem size and number of cores (Section 4.3).

For the former studies, we have chosen a set of default solver parameters and studied the effects of varying individual members of the set while holding the remaining parameters fixed. As our defaults, we used a relative residual tolerance of $\tau = 10^{-4}$ in the iterative solves, we stored a maximum of five history vectors for the limited-memory quasi-Newton approximation to the reduced Hessian of the augmented Lagrangian, and we employed a single reduced-space step in each outer iteration. At the end of Section 4.2, we propose modifications to these default parameters and use the modified parameters throughout the scaling studies of Section 4.3.

All experiments in Section 4.2 were performed at Argonne National Laboratory on dedicated nodes of the Fusion cluster, which comprises 320 nodes, each with 2.6 GHz Pentium Xeon 8-core chips and 36 GB of RAM. All experiments in Section 4.3 were performed at Lawrence Berkeley National Laboratory on dedicated nodes of the Franklin cluster, which comprises 9,572 nodes, each with a 2.3 GHz AMD Budapest 4-core processor and 8 GB of RAM. In all cases, the code was compiled against version 3.2 of PETSc [12].

4.1 Model Problems

We tested the algorithm on the collection of model problems for PDE-constrained optimization proposed by Haber and Hanson [9]. The collection consists of three parameter estimation problems that are constrained by elliptic, parabolic, and hyperbolic PDEs.

In these problems, the design variable v is a parameter distribution in a domain Ω, and the state variable u is the solution to a PDE on Ω that involves the parameter v. The continuous optimization problem asks for the design variable distribution v^* for which the corresponding solution u^* to the PDE most closely matches a set of observed data d. The objective function takes the form

$$f(u,v) = \frac{1}{2}||Qu - d||^2 + \alpha R(v - v_r),$$

where Q denotes a projection operator onto the locations of the measurement data, v_r is a reference parameter distribution, α is a positive scalar, and R is a regularization functional. Details of the PDE constraints and the associated discretizations, each of which are low-order finite-difference schemes on regular grids, can be found in [9]. An illustrative example of each type of problem is provided in TAO.

The regularization functionals (and hence the objective functions) appearing in the model problems constrained by elliptic and parabolic PDEs are convex. The hyperbolic model problem's objective function is nonconvex. The PDE constraints appearing in all three model problems are of the form

$$g(u,v) = \mathcal{A}(v)u - q$$

with $\mathcal{A}$ an operator depending nonlinearly on v.

To solve the linearized forward and adjoint problems arising in the elliptic and parabolic model problems, we used the conjugate gradient method with a successive over-relaxation preconditioner for serial computations and a Jacobi preconditioner for parallel computations. For the hyperbolic model problem, GMRES was used with preconditioners of the same type. We chose to use a Jacobi preconditioner for parallel scalability studies to eliminate variability in the linear solver's efficiency under increased parallelization. (PETSc supports only local SOR sweeps in parallel computations.)

When comparing problem sizes in the subsequent sections, we will sometimes refer to spatial and temporal resolutions of the PDE discretizations rather than to the total number of unknowns in the model problem. We use m_x to denote the unidimensional spatial resolution (i.e., the inverse of the grid spacing) of the discretization, m_t to denote

Table 1: Relationship between spatial resolution (m_x), temporal resolution (m_t), and problem size for each of the Haber-Hanson model problems.

Problem	Spatial Dimension	Time-dependent	# State (n_u)	# Design (n_v)	Total Size (n)
Elliptic	3	No	$m_x^3 m_e$	m_x^3	$m_x^3(m_e+1)$
Parabolic	3	Yes	$m_x^3 m_t$	m_x^3	$m_x^3(m_t+1)$
Hyperbolic	2	Yes	$m_x^2 m_t$	$2m_x^2 m_t$	$3m_x^2 m_t$

Table 2: Performance of the LCL algorithm on the Haber-Hanson model problems with default linear solver tolerances $\tau_i = 10^{-4}$, $i = 1, 2, 3, 4$.

Problem	m_x	m_t	n_u	n_d	Time (sec)	Outer Iters.	Itsolver Iters.	Mat-vecs
Elliptic	32	-	32768	32768	79.2	51	13028	18319
Parabolic	16	8	32768	4096	57.6	71	57339	68955
Hyperbolic	32	32	32768	65536	5.5	31	10980	31276

the number of time steps employed if the problem is time-dependent, and m_e to denote the number of experiments used for the elliptic problem. For all the computational results, $m_e = 1$ was used. Table 1 summarizes the relationship between these parameters and the problem sizes for each of the three model problems.

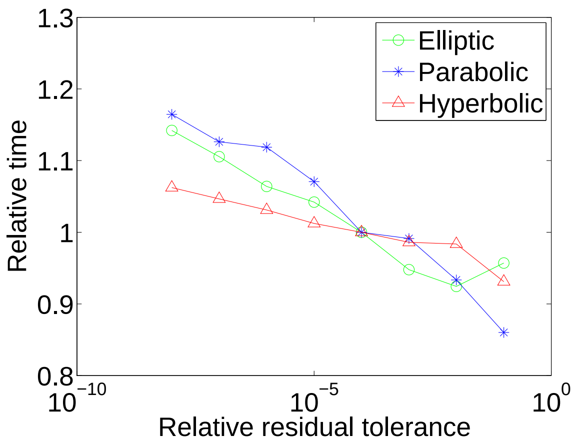
4.2 Accuracy of Solves

We first study the influence of inexact linear solves, approximate subproblem solves, and approximate Hessians on the performance of the algorithm.

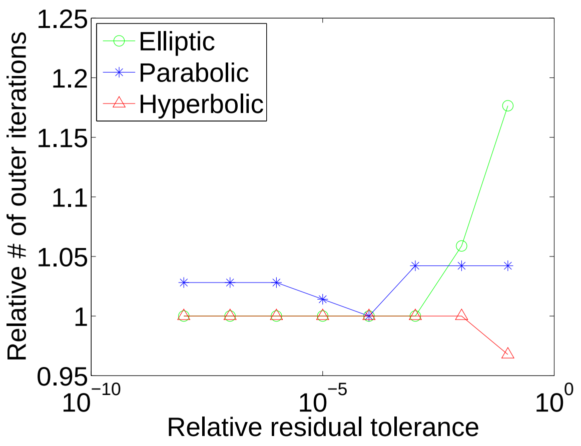
Accuracy of iterative linear solves. We varied the relative residual tolerances τ_i, $i = 1, 2, 3, 4$ for each of the linearized forward and adjoint solves within an outer iteration of the LCL algorithm. The subscript i enumerates the four linear solves in the order that they appear in the algorithm: $i = 1$ corresponds to the first forward solve, $i = 2$ the first adjoint solve, $i = 3$ the second forward solve, and $i = 4$ the second adjoint solve.

Figures 1 and 2 show the performance of the LCL algorithm under different choices of the two illustrative parameters τ_2 and τ_3. The data for each model problem are reported relative to their values at the level $\tau_1 = \tau_2 = \tau_3 = \tau_4 = 10^{-4}$, which are presented in Table 2. The number of iterative solver iterations and matrix-vector products are denoted by "Itsolver Iters." and "Mat-vecs," respectively.

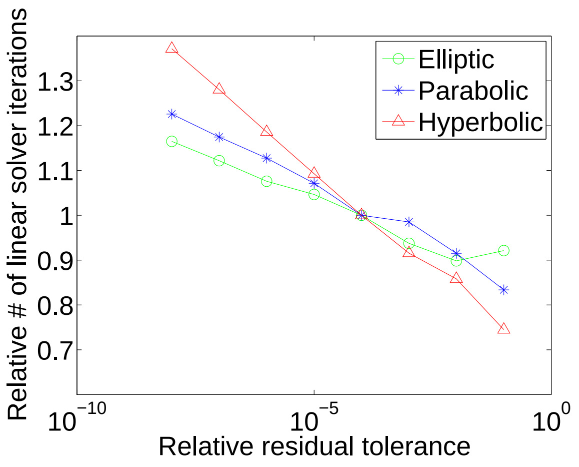
The tolerances τ_1 and τ_2 appear more amenable to loosening. As Figure 1 shows, reductions in solve time accompany decreases in τ_2 beyond 10^{-4} for all three model problems, despite occasional increases in outer iteration counts.



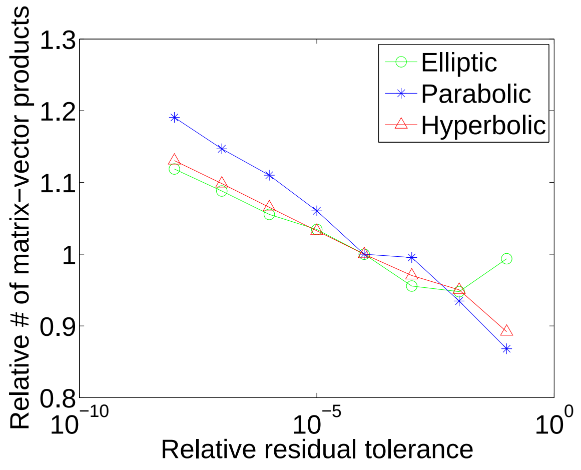
(a)



(b)

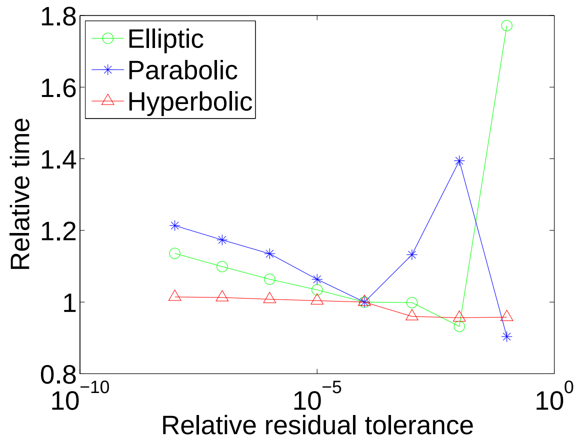


(c)

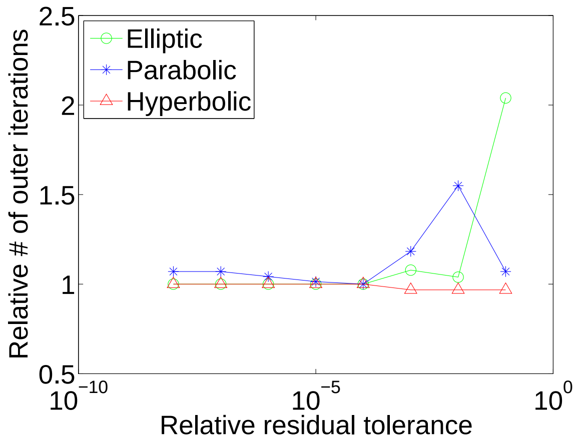


(d)

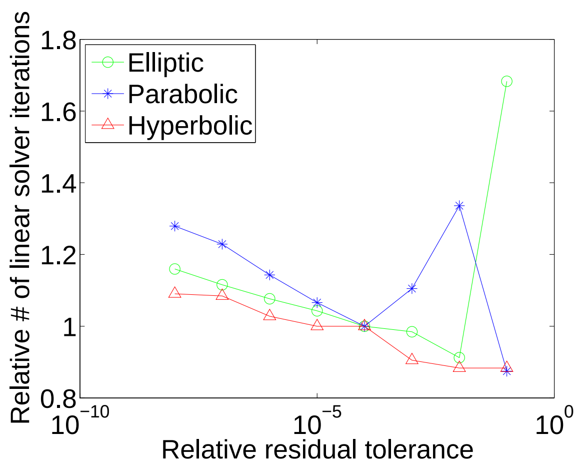
Figure 1: Performance of the LCL algorithm on the Haber-Hanson model problems as a function of the relative residual tolerance τ_2 of the linear solver during the first adjoint solve. Data for each model problem is reported relative to the values at the level $\tau_2 = 10^{-4}$ (see Table 2).



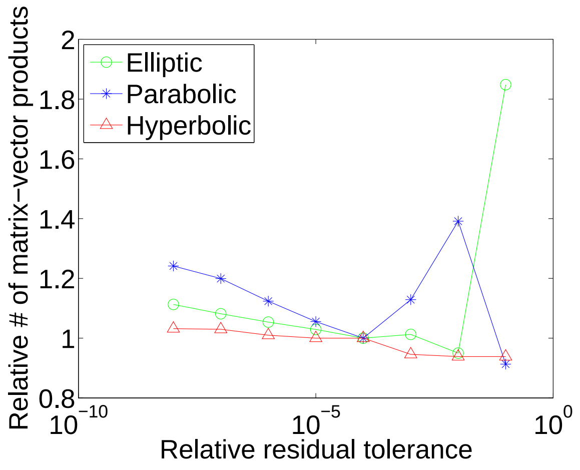
(a)



(b)



(c)



(d)

Figure 2: Performance of the LCL algorithm on the Haber-Hanson model problems as a function of the relative residual tolerance τ_3 of the linear solver during the second forward solve. Data for each model problem is reported relative to the values at the level $\tau_3 = 10^{-4}$ (see Table 2).

Table 3: Performance on the Haber-Hanson model problems as a function of the number l of reduced-space steps taken during each outer iteration.

Problem	l	Time (sec)	Outer Iters.	Itsolver Iters.	Mat-vecs
Elliptic $m_x = 32, m_e = 1$	1	60.3	51	13028	18319
	2	46.3	28	10644	14420
	4	45.1	16	9938	13081
	8	49.2	10	10844	14196
Parabolic $m_x = 16, m_t = 8$	1	33.0	71	57514	69160
	2	29.6	43	52517	61577
	4	43.1	38	78301	89275
	8	53.8	27	99616	111514
Hyperbolic $m_x = 32, m_t = 32$	1	4.6	31	10980	56759
	2	3.9	19	10188	45140
	4	8.9	28	26038	92907

The results indicate that the tolerances τ_3 and τ_4, corresponding to the second forward and adjoint solves, respectively, are less amenable to loosening. For example, Figure 2 shows that setting $\tau_3 < 10^{-4}$ for the parabolic problem results in increases in all four performance measures (solve time, outer iterations, linear solver iterations, and matrix-vector products). Decreasing τ_4 beyond 10^{-4} prevented convergence of the LCL algorithm for the parabolic problem, and decreasing τ_4 beyond 10^{-3} prevented convergence for the elliptic problem. We suspect that the impeded convergence stems from the poor quality of the Lagrange multiplier estimates obtained from the second adjoint solve when a loose tolerance is used.

Based on these observations, we advocate the use of tolerances $\tau_1 = \tau_2 = 10^{-3}$ and $\tau_3 = \tau_4 = 10^{-4}$ in the LCL algorithm.

Accuracy of linearly-constrained subproblem solves. The accuracy to which the linearly constrained subproblem (2) is solved can be adjusted by performing more than one reduced-space step in Phase II of the algorithm. In so doing, one reduces the augmented Lagrangian residual and accumulates more Hessian information during each outer iteration, at the expense of extra computational effort within that iteration.

Table 3 studies the effect of varying the number l of reduced-space steps taken during each outer iteration over the range $1 \leq l \leq 8$. Improvements in computation time accompany the use of two reduced-space steps for each of the model problems, as well as the use of four reduced-space steps for the elliptic problem. We advocate the use of two reduced-space steps per outer iteration on the basis of these tests, although it may be worthwhile to study heuristics for choosing l adaptively as the optimization routine proceeds.

Table 4: Performance of the LCL algorithm on the Haber-Hanson model problems as a function of the number m of history vectors stored in the quasi-Newton approximation of the reduced Hessian.

Problem	m	Time (sec)	Outer Iters.	Itsolver Iters.	Mat-vecs
	3	3.8	29	3767	6761
Elliptic $m_x =$	5	3.6	28	3621	6527
$16, m_e = 1$	10	3.8	29	3755	6749
	20	3.8	29	3766	6747
	40	3.9	29	3768	6749
	3	94.0	60	15479	21744
Elliptic $m_x =$	5	79.4	51	13028	18319
$32, m_e = 1$	10	79.4	51	12985	18263
	20	81.9	52	13345	18737
	40	85.2	54	13798	19392
	3	510.6	75	28714	36546
Elliptic $m_x =$	5	466.5	69	26280	33402
$48, m_e = 1$	10	507.3	75	28477	36231
	20	488.8	72	27351	34802
	40	498.1	73	27765	35304
	3	61.0	75	60825	73075
Parabolic	5	57.4	71	57339	68955
$m_x = 16, m_t = 8$	10	54.3	67	54052	64954
	20	55.2	68	54755	65783
	40	54.5	67	53882	64714
	3	550.3	147	236173	265487
Parabolic	5	500.1	133	215135	241415
$m_x = 24, m_t = 12$	10	413.7	109	176342	198758
	20	482.0	126	206633	231771
	40	401.0	105	171762	192810
	3	2192.1	139	378060	411880
Parabolic	5	2171.0	138	375393	408267
$m_x = 32, m_t = 16$	10	2173.4	137	374608	407804
	20	2083.4	131	359407	390703
	40	2139.8	134	367425	399545
	3	0.4	16	3064	9020
Hyperbolic	5	0.4	16	3062	9018
$m_x = 16, m_t = 16$	10	0.5	16	3060	9016
	20	0.5	16	3060	9016
	40	0.5	16	3060	9016
	3	5.4	31	10948	31317
Hyperbolic	5	5.5	31	10980	31276
$m_x = 32, m_t = 32$	10	5.2	28	9983	28631
	20	5.8	31	10980	31349

Table 5: Modifications to default parameters.

Parameter	Meaning	Initial Default	New Default
τ_1	Residual tolerance, first forward solve	10^{-4}	10^{-3}
τ_2	Residual tolerance, first adjoint solve	10^{-4}	10^{-3}
τ_3	Residual tolerance, second forward solve	10^{-4}	10^{-4}
τ_4	Residual tolerance, second adjoint solve	10^{-4}	10^{-4}
l	Number of reduced-space steps	1	2
m	Number of quasi-Newton history vectors	5	10

Accuracy of quasi-Newton approximation. We considered the role played by the quality of the limited-memory quasi-Newton approximation $\tilde{H}_{k,i}$ to the reduced Hessian of the augmented Lagrangian. Table 4 shows the influence of the number m of quasi-Newton history vectors stored on the performance of the algorithm.

The results indicate that a good choice of m lies somewhere near $m = 10$ for this test suite. An optimal choice of m for a given application will depend on the nature of the objective function, the constraint equations, the size of the problem, and any constraints on computer memory that may be present.

Modifications to default parameters. Based on the results of the preceding studies, we have listed in Table 5 a revised set of default residual tolerances, number of reduced space steps, and number of quasi-Newton history vectors to be used in the LCL algorithm. These defaults will be used throughout the scaling studies in the following section.

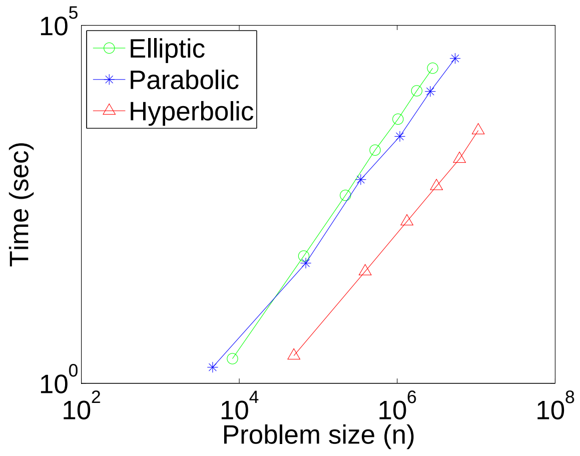
4.3 Scaling Studies

We now study the performance of the algorithm as a function of the problem size and the number of cores used.

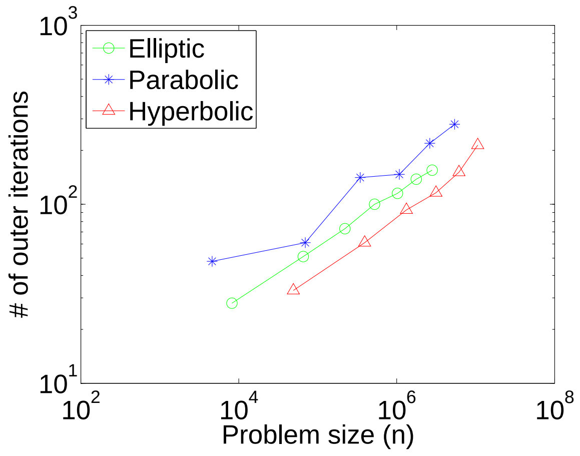
Scaling with respect to problem size. We begin by examining the performance of the algorithm under an increase in problem size with the number of cores held fixed. Figure 3 plots the computational expenses associated with solving each of the model problems on a single core for a range of problem sizes. Specifically, we studied the elliptic problem with $m_x = 16, 32, 48, 64, 80, 96, 112$ and $m_e = 1$, the parabolic problem with $m_x = m_t = 8, 16, 24, 32, 40, 48$, and the hyperbolic problem with $m_x = 2m_t = 32, 64, 96, 128, 160, 192$.

In all three model problems, the solution time increases like n_u^γ, where n_u is the number of state variables and $\gamma \approx 1.5$ (elliptic), $\gamma \approx 1.4$ (parabolic), and $\gamma \approx 1.3$ (hyperbolic).

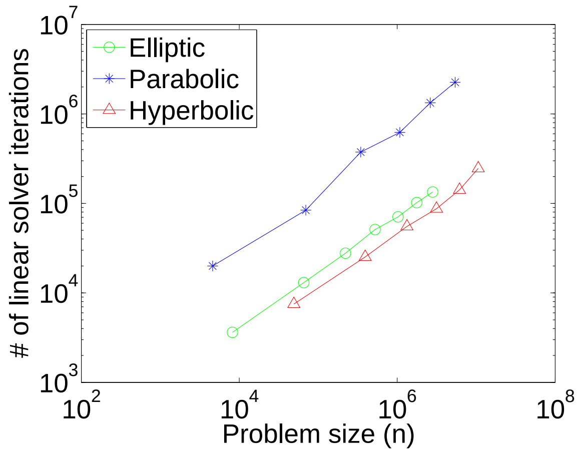
The $O(n_u^{3/2})$ scaling relationship for the elliptic problem is consistent with well-known properties of Krylov subspace methods. Indeed, consider the application of an iterative Krylov subspace method to an n_u-dimensional linear system $Ax = b$ with a preconditioner P. Each iteration of the solver requires $O(n_u)$ flops if A is sparse. Moreover, the number of iterations needed to meet a fixed relative residual grows like the square root of the condition



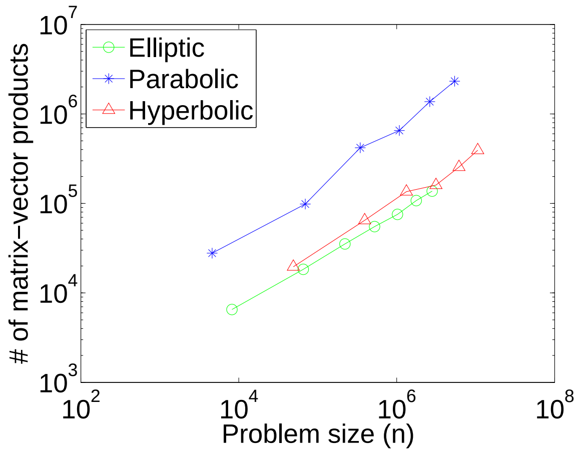
(a)



(b)



(c)



(d)

Figure 3: Performance of the LCL algorithm on the Haber-Hanson model problems as a function of the problem size n.

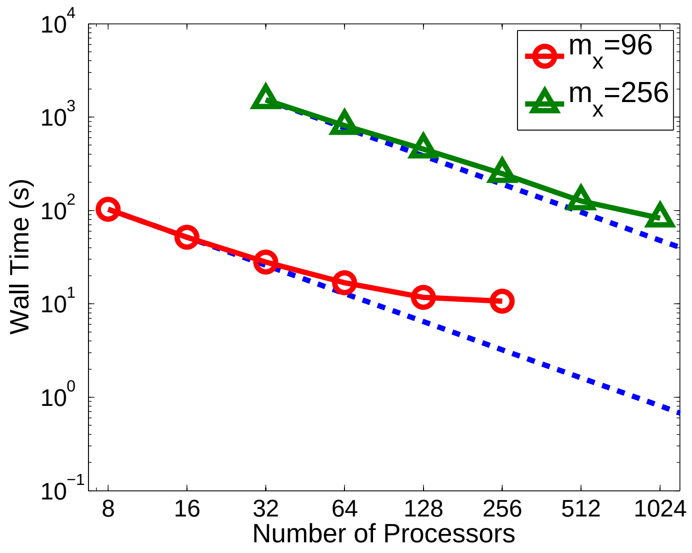


Figure 4: Strong scaling results for the Haber-Hanson elliptic model problem.

number κ of $P^{-1}A$ [8]. For the constraint Jacobian appearing in the elliptic model problem, $\kappa(P^{-1}A) = O(n_u)$ [8], so the $O(n_u^{3/2})$ run time is fully explicable.

Clearly, superior scaling with respect to problem size is achievable in many circumstances, for example, through the use of multigrid methods. We have not explored such enhancements in this study.

Strong scaling. We now examine the performance on a fixed problem size as the number of cores increases. Figure 4 plots the computational expense associated with running five outer iterations on the elliptic problem. We kept the number of iterations constant to obtain a more accurate view of the overhead associated with increasing the cores. For $m_x = 96$ and $m_e = 1$ (1.7M variables), we see that the results follow the ideal scaling trajectory for between 8 and 64 cores, but that for more than 128 cores, the communication and set-up overhead start to dominate. For larger problem sizes, strong scaling is evident for a larger number of cores, as illustrated by the $m_x = 256$ and $m_e = 1$ results (33M variables), which scale well up to 1,024 cores.

5 Conclusion

We have developed a linearly-constrained augmented Lagrangian method for solving optimization problems with partial differential equation constraints. The computational cost of the algorithm is dominated by the cost of inexactly solving linearizations of the forward and adjoint PDEs. Numerical tests on a suite of model problems indicate that the algorithm exhibits good parallel scalability and that as the problem size increases, the solution time grows almost as slowly as the cost of the inexact linear solves of dimension equal to the number of state variables. Further speedups can be realized through judicious choices of linear solver tolerances. The algorithm and model problems are available in version 2.0 of the Toolkit for Advanced Optimization.

Acknowledgments

We are grateful to Lauren Taralli (née Hanson) for providing code for the model problems in [9]. We gratefully acknowledge the computing resources provided by the Laboratory Computing Resource Center at Argonne National Laboratory and by the National Energy Research Scientific Computing Center at Lawrence Berkeley National Laboratory.

References

[1] V. Akçelik, G. Biros, O. Ghattas, J. Hill, and B. van Bloemen Waanders. Parallel algorithms for PDE-constrained optimization. In M. Heroux, P. Raghaven, and H. Simon, editors, *Frontiers of Parallel Computing*, pages 291–322. SIAM, 2006.

[2] Satish Balay, Jed Brown, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Memorandum ANL-95/11 - Revision 3.2, Argonne National Laboratory, 2011.

[3] G. Biros and O. Ghattas. Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization, part I: The Krylov-Schur solver. *SIAM Journal on Scientific Computing*, 27:687–713, 2005.

[4] G. Biros and O. Ghattas. Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization, part II: The Lagrange-Newton solver and its application to optimal control of steady viscous flows. *SIAM Journal on Scientific Computing*, 27:714–739, 2005.

[5] Scientific grand challenges: Challenges for understanding the quantum universe and the role of computing at the extreme scale, December 2008. `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Hep_report.pdf`.

[6] Scientific grand challenges: Discovery in basic energy sciences: The role of computing at the extreme scale, August 2009. `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Bes_exascale_report.pdf`.

[7] Scientific grand challenges: Fusion energy sciences and the role of computing at the extreme scale, March 2009. `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Fusion_report.pdf`.

[8] Ivar Gustafsson. A class of first order factorization methods. *BIT Numerical Mathematics*, 18:142–156, 1978.

[9] Eldad Haber and Lauren Hanson. Model problems in PDE-constrained optimization. Technical Report TR-2007-009, Emory, Atlanta, Georgia, 2007.

[10] T. Munson, J. Sarich, Stefan M. Wild, S. Benson, and L. Curfman McInnes. TAO 2.0 users manual. Technical Memorandum ANL/MCS-TM-322, Argonne National Laboratory, Argonne, Illinois, 2012.

[11] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, 2nd edition, 2006.

[12] PETSc. Portable Extensible Toolkit for Scientific Computation. See `www.mcs.anl.gov/petsc`.

[13] E. Prudencio, R. Byrd, and X.-C. Cai. Parallel full space SQP Lagrange-Newton-Krylov-Schwarz algorithms for PDE-constrained optimization problems. *SIAM Journal on Scientific Computing*, 27:1305–1328, 2006.

[14] TAO. Toolkit for Advanced Optimization. See `http://www.mcs.anl.gov/tao`.