# PARALLEL LAGRANGE-NEWTON-KRYLOV-SCHUR METHODS FOR PDE-CONSTRAINED OPTIMIZATION. PART I: THE KRYLOV-SCHUR SOLVER *

GEORGE BIROS† AND OMAR GHATTAS‡

**Abstract.** Large scale optimization of systems governed by partial differential equations (PDEs) is a frontier problem in scientific computation. The state-of-the-art for such problems is reduced quasi-Newton sequential quadratic programming (SQP) methods. These methods take full advantage of existing PDE solver technology and parallelize well. However, their algorithmic scalability is questionable; for certain problem classes they can be very slow to converge. In this two-part article we propose a new method for steady-state PDE-constrained optimization, based on the idea of full space SQP with reduced space quasi-Newton SQP preconditioning. The basic components of the method are: Newton solution of the first-order optimality conditions that characterize stationarity of the Lagrangian function; Krylov solution of the Karush-Kuhn-Tucker (KKT) linear systems arising at each Newton iteration using a symmetric quasi-minimum residual method; preconditioning of the KKT system using an approximate state/decision variable decomposition that replaces the forward PDE Jacobians by their own preconditioners, and the decision space Schur complement (the reduced Hessian) by a BFGS approximation or by a two-step stationary method. Accordingly, we term the new method *Lagrange-Newton-Krylov Schur* (LNKS). It is fully parallelizable, exploits the structure of available parallel algorithms for the PDE forward problem, and is locally quadratically convergent. In the first part of the paper we investigate the effectiveness of the KKT linear system solver. We test the method on two optimal control problems in which the flow is described by the steady-state Stokes equations. The objective is to minimize dissipation or the deviation from a given velocity field; the control variables are the boundary velocities. Numerical experiments on up to 256 Cray T3E processors and on an SGI Origin 2000 include scalability and performance assessment of the LNKS algorithm and comparisons with the reduced SQP for up to 1,000,000 state and 50,000 decision variables. In the second part of the paper we present globalization and robustness algorithmic issues and we apply LNKS to the optimal control of the steady incompressible Navier-Stokes equations.

**Key words.** Sequential quadratic programming, Adjoint methods, PDE-constrained optimization, optimal control, Lagrange-Newton-Krylov-Schur methods, Navier-Stokes, finite elements, preconditioners, indefinite systems, nonlinear equations, parallel algorithms

**AMS subject classifications.** 49K20, 65F10, 65K05, 65K10, 65J22, 65N55, 65W10, 65Y05, 65Y20, 76D05, 76D07, 76D55, 90C52, 90C55, 90C90, 93C20

## 1. Introduction.

PDE-constrained optimization refers to the optimization of systems that are governed by partial differential equations (PDEs). The *forward* or *state problem* is to solve the PDEs for the *state variables*, given appropriate data, i.e. geometry, coefficients, boundary conditions, initial conditions, and source functions. The *optimization problem* seeks to determine some of these data—the *decision variables*—given an *objective function* (e.g. performance goals) and (possibly inequality) constraints on the behavior of the system. Since the behavior of the system is modeled by the PDEs, they appear as constraints in the optimization problem.

PDE-constrained optimization problems arise in several contexts, including optimal design, optimal control, and parameter estimation. They share the common difficulty that PDE solution is just a subproblem associated with optimization, and that the optimization problem can be ill-posed even when the forward problem is well-posed. Thus the optimization

*This work is a part of the Terascale Algorithms for Optimization of Simulations (TAOS) project at CMU, with support from NASA grant NAG-1-2090, NSF grant ECS-9732301 (under the NSF/Sandia Life Cycle Engineering Program), and the Pennsylvania Infrastructure Technology Alliance. Computing services on the Pittsburgh Supercomputing Center's Cray T3E were provided under PSC grant BCS-960001P.

†Courant Institute of Mathematical Sciences Departement of Computer Science, New York University, New York, NY 10012, USA (biros@cs.nyu.edu).

‡Mechanics, Algorithms, and Computing Laboratory, Departments of Biomedical Engineering and Civil & Environmental Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, 15213, USA (oghattas@cs.cmu.edu).

problem is often significantly more difficult to solve than the forward problem.

To illustrate the main issues, let us consider a model problem of optimal distributed control of a steady Navier-Stokes flow:

$$\min \mathcal{F}(\boldsymbol{u}, p, \boldsymbol{b}) = \frac{1}{2}\int_\Omega (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)\cdot(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) d\Omega + \frac{\rho}{2}\int_\Omega \boldsymbol{b}\cdot\boldsymbol{b}\, d\Omega,$$

subject to:

$$\begin{aligned} -\nu\nabla\cdot(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) + (\nabla \boldsymbol{u})\boldsymbol{u} + \nabla p + \boldsymbol{b} = \boldsymbol{0} \quad &\text{in} \quad \Omega, \\ \nabla\cdot\boldsymbol{u} = \boldsymbol{0} \quad &\text{in} \quad \Omega, \\ \boldsymbol{u} = \boldsymbol{g} \quad &\text{on} \quad \Gamma. \end{aligned}$$

Here, $\boldsymbol{u}$ is the fluid velocity field, $p$ the pressure field, $\boldsymbol{b}$ the body force control function, $\boldsymbol{g}$ the prescribed Dirichlet boundary conditions, $\rho$ a weighting parameter, and $\nu$ the inverse of the Reynolds number. The objective is to minimize the rate of dissipation of viscous energy without applying an excessively large body force. The constraints are the stationary incompressible Navier-Stokes equations with Dirichlet boundary conditions. The decision variable function is the body force $\boldsymbol{b}$.

We can form a Lagrangian functional, and require its stationarity with respect to the state ($\boldsymbol{u}$,$p$), decision variables ($\boldsymbol{b}$), and the Lagrange multipliers. Taking the respective variations and invoking the appropriate Green identities, we arrive at the (strong form of) the so-called *first order optimality*, or the *Karush-Kuhn-Tucker*, system:

**State Equations:**

$$\begin{aligned} -\nu\nabla\cdot(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) + (\nabla \boldsymbol{u})\boldsymbol{u} + \nabla p + \boldsymbol{b} = \boldsymbol{0} \quad &\text{in} \quad \Omega, \\ \nabla\cdot\boldsymbol{u} = \boldsymbol{0} \quad &\text{in} \quad \Omega, \\ \boldsymbol{u} = \boldsymbol{g} \quad &\text{on} \quad \Gamma. \end{aligned}$$

**Adjoint Equations:**

$$\begin{aligned} -\nu\nabla\cdot(\nabla \boldsymbol{\lambda} + \nabla \boldsymbol{\lambda}^T) + (\nabla \boldsymbol{u})^T\boldsymbol{\lambda} - (\nabla \boldsymbol{\lambda})\boldsymbol{u} + \nabla\mu - \nabla\cdot(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) = \boldsymbol{0} \quad &\text{in} \quad \Omega, \\ \nabla\cdot\boldsymbol{\lambda} = \boldsymbol{0} \quad &\text{in} \quad \Omega, \\ \boldsymbol{\lambda} = \boldsymbol{0} \quad &\text{on} \quad \Gamma. \end{aligned}$$

**Control Equations:**

$$\rho\,\boldsymbol{b} + \boldsymbol{\lambda} = \boldsymbol{0} \quad \text{in} \quad \Omega.$$

The first set of equations are just the original Navier-Stokes PDEs. The *adjoint* equations, which result from stationarity with respect to state variables, are themselves PDEs, and are linear in the Lagrange multipliers $\boldsymbol{\lambda}$ and $\mu$. Finally, the *control equations* are (in this case) algebraic.

The above optimality system is a large, nonlinear, coupled, unstructured system of PDEs (or at least larger, more coupled, and less structured than usually seen by the forward PDE solver). Solving this system presents significant challenges beyond the forward PDE: One way to mitigate some of the difficulties is to eliminate state variables and Lagrange multipliers, and correspondingly, the state equations and adjoint equations, to reduce the system to a manageable one in just the decision variables $\boldsymbol{b}$. This can be done as follows: given $\boldsymbol{b}$ at some iteration, we solve the state equations for the state variables $\boldsymbol{u}, p$. Knowing the state

variables then permits us to solve the adjoint equations for the Lagrange multipliers $\boldsymbol{\lambda}, \mu$. Finally, with the states and multipliers known, we can update $\boldsymbol{b}$ by taking a Newton-like iteration the control equation. The whole process is repeated until convergence. This elimination procedure is termed a *reduced space* method, in contrast to a *full space* method, in which one solves for the states, controls, and multipliers simultaneously. The variable reduction can be done either prior or after the Newton linearization. The first approach is a type of *nonlinear elimination* and is known as a *Generalized Reduced Gradient method*, and the second one a *linear elimination* or *reduced Sequential Quadratic Programming method* (SQP) .

Reduced space methods are attractive for several reasons. Solving the subsets of equations in sequence imparts some structure to the optimization problem. State equation solvers build on years of development of large-scale parallel PDE solvers. The strong similarities between the state and adjoint operators suggest that an existing PDE solver for the state equations, can often be modified readily to handle the adjoint system. Finally, the control equations are often reasonably straight forward, at least to evaluate. Another advantage of reduction is that the full space system is often very ill-conditioned, whereas the three subsystems are typically better conditioned individually.

Given these advantages, the overwhelming popularity of reduced space methods for PDE optimization is not surprising. We will not attempt to survey such methods but rather we give a sample of the literature on applications to various PDE problems: applications to compressible flow airfoil design [32], [39]; heat equation boundary control [27]; inverse parameter estimation [14], [26], Navier-Stokes flow control [18]; inverse acoustic scattering [15]; chemically reacting flows [38]; and structural optimization [33], [35], [36]. In addition, parallel implementations of RSQP methods exhibiting high parallel efficiency and good scalability have been developed [15], [19], [32],[34],[28].

On the other hand, the major disadvantage of reduced methods is the requirement to solve the (possibly linearized) state and adjoint equations *at each iteration* of the reduced system—which is a direct consequence of the reduction onto the decision variable space. Because of this difficulty it is natural to return to the full space, and ask if it is possible to solve simultaneously the entire optimality system, but retain the structure-inducing, condition-improving advantages of reduced space methods—while avoiding their disadvantages.

In this two-part article we present such a method. The key idea is to solve in the full space using a Newton method, but precondition with a quasi-Newton reduced space method. We refer to the (nonlinear) Newton iterations as *outer* iterations, and use the term *inner* to describe the (linear) Krylov iterations for the Karush-Kuhn-Tucker (KKT) system that arises at each Newton iteration. This linearized KKT system is solved using a Krylov iterative method (symmetric QMR), and it is this system to which the preconditioner is applied. The reduced space preconditioner is very effective in deflating the spectrum of the KKT matrix and in reducing the number of Krylov iterations—"inverting" it captures the favorable structure of reduced methods. On the other hand, since the reduction is used just as a preconditioner, we can approximate the state and adjoint solves, replacing them with approximations, which could be their own preconditioners. So we arrive at a method that combines rapid convergence in the outer Newton iteration (typically mesh-independent), with fast convergence of the inner Krylov iteration (which can be as good as mesh-independent).

The new method is termed *Lagrange-Newton-Krylov-Schur*. It is common in the PDE-solver literature to use the phrase *Newton-Krylov-X* to refer to Newton methods for solving PDEs that employ Krylov linear solvers, with *X* as the preconditioner for the Krylov method. Since *Lagrange-Newton* is used sometimes to describe a Newton method for solving the optimality system (i.e. an SQP method), and since a reduced space method can be viewed as a Schur complement method for the KKT system, we arrive at the concatenation *LNKS*.

The LNKS method is inspired by domain-decomposed Schur complement PDE solvers. In such techniques, reduction onto the interface space requires exact subdomain solves, so one often prefers to iterate within the full space while using a preconditioner based on approximate subdomain solution [25]. In our case, the decomposition is into states and decisions, as opposed to subdomain and interface spaces.

An early version of an LNKS method was presented in [9] and [10]. Here we extend the method in a number of directions. In Part I, we analyze and compare several different variants of the KKT preconditioner. Scalability to larger problems and processor counts is studied. In Part II, important globalization and inexactness issues, which were not considered in the earlier work, are in introduced and analyzed. We also apply the method to a set viscous flow control problems.

Haber and Asher [23] have recently proposed a KKT preconditioner similar to [10] and [9]. The main differences are in the reduced Hessian preconditioner and the application to inverse problems. Battermann and Heinkenschloss have presented a related KKT-system preconditioner that also makes use of state and decision space decompositions [7]. However, their preconditioner is based on congruence transformations of the original system and not on an exact factorization of it. The resulting preconditioned system has both positive and negative eigenvalues and its spectrum is less favorably distributed. Another important difference is that we precondition in the reduced space. We also consider parallelism and scalability issues.

Part I of the article is organized as follows: in Section 2 we discuss sequential quadratic programming methods and in particular reduced space variants; in Section 3 we introduce the LNKS method and present several approaches for preconditioning the KKT matrix; in Section 4 we examine the formulation of a Stokes flow problem; and in Section 5 we conclude with numerical results, and study parallel and algorithmic scalability of the method.

A note on notation: we use boldface characters to denote vector-valued functions and vector-valued function spaces. We use roman characters to denote discretized quantities and italics for their continuous counterparts. For example $\boldsymbol{u}$ will be the continuous velocity field and $\mathbf{u}$ will be its discretization. Greek letters are overloaded and whether we refer to the discretization or the continuous fields should be clear from context. We also use $(+)$ as a subscript or superscript to denote variable updates within an iterative algorithm.

**2. Reduced Space Methods.** We begin with a constrained optimization problem formulated as follows:

$$\min_{\mathbf{x}\in\mathbb{R}^N} f(\mathbf{x}), \quad \text{subject to} \quad \mathbf{c}(\mathbf{x}) = \mathbf{0}, \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^N$ are the optimization variables, $f : \mathbb{R}^N \to \mathbb{R}$ is the objective function, and $\mathbf{c} : \mathbb{R}^N \to \mathbb{R}^n$ are the constraints. We assume that the constraints consist of only (discretized) state equations.[1] In order to exploit the structure of the problem we partition $\mathbf{x}$ into the state variables $\mathbf{x}_s \in \mathbb{R}^n$ and the decision variables $\mathbf{x}_d \in \mathbb{R}^m$,

$$\mathbf{x} = \left\{ \begin{array}{c} \mathbf{x}_s \\ \mathbf{x}_d \end{array} \right\}, \tag{2.2}$$

so that $N = m+n$. By introducing the Lagrangian function $\mathcal{L}$ one can derive first and higher order optimality conditions. The Lagrangian is defined by

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) := f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{c}(\mathbf{x}), \tag{2.3}$$

[1] However, the methodology can be extended to problems that include additional equality or inequality constraints.

where $\boldsymbol{\lambda} \in \mathbb{R}^n$ are the Lagrange multipliers (or adjoint variables). The first order optimality conditions state that at a local minimum the Lagrangian gradient must vanish:

$$\left\{ \begin{array}{c} \partial_x \mathcal{L} \\ \partial_\lambda \mathcal{L} \end{array} \right\} (\mathbf{x}, \boldsymbol{\lambda}) = \left\{ \begin{array}{c} \partial_x f(\mathbf{x}) + (\partial_x \mathbf{c}(\mathbf{x}))^T \boldsymbol{\lambda} \\ \mathbf{c}(\mathbf{x}) \end{array} \right\} = \mathbf{0}. \tag{2.4}$$

Points at which the gradient of the Lagrangian vanishes are often called *KKT points*.[2] Equations (2.4) are typically known as the Karush-Kuhn-Tucker (KKT) optimality conditions. To simplify the notation further, let us define:

| | | |
|---|---|---|
| $\mathbf{A}(\mathbf{x}) := \partial_x \mathbf{c}(\mathbf{x})$ | $\in \mathbb{R}^{n \times N}$ | Jacobian matrix of the constraints, |
| $\mathbf{W}(\mathbf{x}, \boldsymbol{\lambda}) := \partial_{xx} f(\mathbf{x}) + \sum_i \lambda_i \partial_{xx} \mathbf{c}_i(\mathbf{x})$ | $\in \mathbb{R}^{N \times N}$ | Hessian matrix of the Lagrangian, |
| $\mathbf{g}(\mathbf{x}) := \partial_x f(\mathbf{x})$ | $\in \mathbb{R}^N$ | gradient vector of the objective. |

Consistent with the partitioning of the optimization variables into states and decisions, we logically partition $\mathbf{g},\ \mathbf{A},\ \mathbf{W}$ as follows:[3]

$$\mathbf{g} = \left\{ \begin{array}{c} \mathbf{x}_s \\ \mathbf{x}_d \end{array} \right\},\ \mathbf{A} = \left[ \begin{array}{cc} \mathbf{A}_s & \mathbf{A}_d \end{array} \right],\ \text{and}\ \mathbf{W} = \left[ \begin{array}{cc} \mathbf{W}_{ss} & \mathbf{W}_{sd} \\ \mathbf{W}_{ds} & \mathbf{W}_{dd} \end{array} \right].$$

At each iteration of the SQP method a quadratic programming problem (QP) is solved to generate a new search direction $\mathbf{p}_x$. When SQP is derived from a Newton method for solving the optimality conditions (2.4), the resulting QP is of the form

$$\min_{\mathbf{p}_x \in \mathbb{R}} \frac{1}{2} \mathbf{p}_x^T \mathbf{W} \mathbf{p}_x + \mathbf{g}^T \mathbf{p}_x \quad \text{subject to} \quad \mathbf{A}\mathbf{p}_x + \mathbf{c} = \mathbf{0}. \tag{2.5}$$

Reduced space methods for solving (2.5) eliminate the linearized constraint by using the null and range space decomposition of the search direction,

$$\mathbf{p}_x = \mathbf{Z}\mathbf{p}_z + \mathbf{Y}\mathbf{p}_y, \tag{2.6}$$

where the columns of $\mathbf{Z} \in \mathbb{R}^{n \times m}$ form a basis for the null space of $\mathbf{A}$ (so that $\mathbf{AZ} = \mathbf{0}$). Note that $\mathbf{Y}$ is often not orthogonal to the null space of the constraints (and hence not a true range space basis for $\mathbf{A}^T$). However, $\mathbf{AY}$ should have full rank to ensure solvability for the (not strictly) range-space component $\mathbf{p}_y$. The decomposition (2.6) permits the $\mathbf{p}_z$ and $\mathbf{p}_y$ components to be computed separately, as shown bellow. Let us define the reduced gradient of the objective function $\mathbf{g}_z$, the reduced Hessian $\mathbf{W}_z$, and an auxiliary matrix $\mathbf{W}_y$:

$$\begin{array}{rcl} \mathbf{g}_z & := & \mathbf{Z}^T \mathbf{g}, \\ \mathbf{W}_z & := & \mathbf{Z}^T \mathbf{W} \mathbf{Z}, \\ \mathbf{W}_y & := & \mathbf{Z}^T \mathbf{W} \mathbf{Y}. \end{array} \tag{2.7}$$

Then the reduced space SQP (RSQP) algorithm (without line search) is given by Algorithm 1. The "+" subscript signifies an updated variable.

Choices on how to approximate $\mathbf{W}_z$, and on how to construct the bases $\mathbf{Z}$ and $\mathbf{Y}$, determine some of the different RSQP variants. An important step of this algorithm is "inverting" $\mathbf{W}_z$. The condition number of the reduced Hessian is affected by the choice of $\mathbf{Z}$, and ideally $\mathbf{Z}$ would come from an orthogonal factorization of $\mathbf{A}$. This approach is not possible for discretized PDE constraints, particularly in three dimensions. There is, however, a convenient

[2] Saddle points and local maxima are also KKT points.

[3] All vectors and matrices depend on the optimization variables $\mathbf{x}$; in addition the Hessian $\mathbf{W}$ depends also on the Lagrange multipliers $\boldsymbol{\lambda}$. For clarity, we suppress identification of these dependencies.

**Algorithm 1** Reduced space Sequential Quadratic Programming (RSQP)

1: Choose $\mathbf{x}$
2: **loop**
3: Evaluate $\mathbf{c},\ \mathbf{g},\ \mathbf{A},\ \mathbf{W}$
4: $\mathbf{g}_z = \mathbf{Z}^T\mathbf{g}$
5: **if** $\|\mathbf{g}_z\| \leq tol$ and $\|\mathbf{c}\| \leq tol$ **then**
6: Converged
7: **end if**
8: $(\mathbf{AY})\mathbf{p}_y = -\mathbf{c}$ Solve for $\mathbf{p}_y$
9: $\mathbf{W}_z\mathbf{p}_z = -(\mathbf{g}_z + \mathbf{W}_y\mathbf{p}_y)$ Solve for $\mathbf{p}_z$
10: $\mathbf{p}_x = \mathbf{Z}\mathbf{p}_z + \mathbf{Y}\mathbf{p}_y$
11: $\mathbf{x}_+ = \mathbf{x} + \mathbf{p}_x$
12: $(\mathbf{AY})^T\boldsymbol{\lambda}_+ = -\mathbf{Y}^T(\mathbf{g} + \mathbf{W}\mathbf{p}_x)$ Solve for $\boldsymbol{\lambda}_+$
13: **end loop**

form of $\mathbf{Z}$ that is easy to compute and takes advantage of the structure of the constraints. (It is motivated by the fact that $\mathbf{A}_s$ is the (linearized) forward PDE operator and expressing the null basis in terms of the (formal) inverse of $\mathbf{A}_s$ builds on a large body of PDE solver technology.) It is given by

$$\mathbf{Z} := \begin{bmatrix} -\mathbf{A}_s^{-1}\mathbf{A}_d \\ \mathbf{I} \end{bmatrix} \tag{2.8}$$

and then $\mathbf{Y}$ can be defined by

$$\mathbf{Y} := \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix}. \tag{2.9}$$

The expression for the null space basis $\mathbf{Z}$ implies that the reduced gradient is given by

$$\mathbf{g}_z = \mathbf{g}_d - \mathbf{A}_d^T\mathbf{A}_s^{-T}\mathbf{g}_s, \tag{2.10}$$

and the reduced Hessian is given by

$$\mathbf{W}_z = \mathbf{A}_d^T\mathbf{A}_s^{-T}\mathbf{W}_{ss}\mathbf{A}_s^{-1}\mathbf{A}_d - \mathbf{A}_d^T\mathbf{A}_s^{-T}\mathbf{W}_{sd} - \mathbf{W}_{ds}\mathbf{A}_s^{-1}\mathbf{A}_d + \mathbf{W}_{dd}. \tag{2.11}$$

Algorithm 2 states a particularized (to the special form of $\mathbf{Z}$ and $\mathbf{Y}$ given above) description of RSQP, which we refer to as Newton-RSQP. The algorithm can be decomposed into three main steps: inverting the reduced Hessian for the decision search direction (step 8), inverting the (linearized) forward operator for the state search direction (step 9), and inverting the adjoint of the (linearized) forward operator for the Lagrange multipliers (step 10).

There are two ways to solve the decision equation in step 8. We can either compute and store the reduced Hessian $\mathbf{W}_z$ in order to use it with a direct solver, or we can use an iterative method, which only requires just matrix–vector products with $\mathbf{W}_z$. Direct solvers are very effective, but computing the reduced Hessian requires $m$ linearized forward solves. If we count the solves for the right-hand side of the decision equation and the adjoint and forward steps, then we have a total of $m + 4$ solves *at each* optimization iteration.

If an iterative method is used to solve the decision equation in step 8, then a matrix-vector multiplication with $\mathbf{W}_z$ requires one solve with each $\mathbf{A}_s^{-1}$ and $\mathbf{A}_s^{-T}$. A Krylov method like Conjugate Gradients (CG) will converge in $m$ steps (in exact arithmetic) and thus the number

**Algorithm 2** Newton RSQP

Choose $\mathbf{x}_s, \mathbf{x}_d, \boldsymbol{\lambda}$
**loop**
  Evaluate $\mathbf{c},\ \mathbf{g},\ \mathbf{A},\ \mathbf{W}$
  $\mathbf{g}_z = \mathbf{g}_d + \mathbf{A}_d^T \boldsymbol{\lambda}$
  **if** $\|\mathbf{g}_z\| \leq tol$ and $\|\mathbf{c}\| \leq tol$ **then**
    Converged
  **end if**
  $\mathbf{W}_z \mathbf{p}_d = -\mathbf{g}_z + (\mathbf{W}_{ds} - \mathbf{A}_d^T \mathbf{A}_s^{-T} \mathbf{W}_{ss}) \mathbf{A}_s^{-1} \mathbf{c}$ solve for $\mathbf{p}_d$ (**Decision step**)
  $\mathbf{A}_s \mathbf{p}_s = -\mathbf{A}_d \mathbf{p}_d - \mathbf{c}$ solve for $\mathbf{p}_s$ (**State step**)
  $\mathbf{A}_s^T \boldsymbol{\lambda}_+ = -(\mathbf{g}_s + \mathbf{W}_{ss} \mathbf{p}_s + \mathbf{W}_{sd} \mathbf{p}_d)$ solve for $\boldsymbol{\lambda}_+$ (**Adjoint Step**)
  $\mathbf{x}_+ = \mathbf{x} + \mathbf{p}_x$
**end loop**

of forward/adjoint solves will not exceed $2m$. In practice the number of iterations depends on the spectrum of the reduced Hessian. With an optimal preconditioner the iteration count can be independent of $m$. However, it is not obvious how to devise optimal preconditioners for the reduced Hessian. Furthermore, we still require four additional forward (and adjoint) solves for each SQP iteration, and two solves per CG iteration.

Even when then number of decision variables is small, it is advantageous to choose an RSQP variant that avoids computing $\mathbf{W}_z$. The main reason is that second derivatives are often difficult to compute. Moreover, Newton's method is not globally convergent and far from the solution the quality of the decision step $\mathbf{p}_z$ is questionable.

In a quasi-Newton RSQP method, $\mathbf{W}_z$ is replaced by a quasi-Newton approximation which drops the convergence rate from quadratic to linear. In addition, the second derivative terms are dropped from the right hand sides of the decision and adjoint steps, at the expense of a reduction from one-step to two-step superlinear convergence [8]. An important

**Algorithm 3** Quasi-Newton RSQP

Choose $\mathbf{x}_s, \mathbf{x}_d, \boldsymbol{\lambda}, \mathbf{B}_z$
**loop**
  Evaluate $\mathbf{c},\ \mathbf{g},\ \mathbf{A}$
  $\mathbf{g}_z = \mathbf{g}_d + \mathbf{A}_d^T \boldsymbol{\lambda}$
  **if** $\|\mathbf{g}_z\| \leq tol$ and $\|\mathbf{c}\| \leq tol$ **then**
    Converged
  **end if**
  $\mathbf{B}_z \mathbf{p}_d = -\mathbf{g}_z$ solve for $\mathbf{p}_d$ (**Decision step**)
  $\mathbf{A}_s \mathbf{p}_s = -\mathbf{A}_d \mathbf{p}_d - \mathbf{c}$ solve for $\mathbf{p}_s$ (**State step**)
  $\mathbf{A}_s^T \boldsymbol{\lambda}_+ = -\mathbf{g}_s$ solve for $\boldsymbol{\lambda}_+$ (**Adjoint Step**)
  $\mathbf{x}_+ = \mathbf{x} + \mathbf{p}_x$
  Update $\mathbf{B}_z$
**end loop**

advantage of this quasi-Newton method is that only two linearized forward/adjoint problems need to be solved at each iteration, as opposed to the $m$ forward solves needed by N-RSQP for constructing $\mathbf{A}_s^{-1} \mathbf{A}_d$ in $\mathbf{W}_z$ [18]. Furthermore, no second derivatives are required. The combination of a sufficiently accurate line search and an appropriate quasi-Newton update guarantees a descent search direction and thus a globally convergent algorithm [11].

QN-RSQP has been efficiently parallelized for moderate numbers of decision variables [28]; the parallelism is fine-grained across the state and adjoint fields. Unfortunately, the number of iterations taken by quasi-Newton methods often increases as the number of decision variables grows,[4] rendering large-scale problems intractable. Additional processors will not help since the bottleneck is in the iteration dimension.

On the other hand, convergence of the nonlinear iterations of N-RSQP method can be independent of the number of decision variables $m$, particularly when these variables are mesh related. But unless an optimal $\mathbf{W}_z$ preconditioner is used, the large number of forward/adjoint solves per iteration preclude its use, particularly on a parallel machine, where iterative methods for the forward problem must be used. However, there is a way to exploit the fast convergence of the Newton method and avoid solving the PDEs exactly, and this method is proposed in the next section. The basic idea is to solve the full-space system (2.4) simultaneously for states, adjoints, and decision variables, but invoke QN-RSQP as a preconditioner. Since it is just a preconditioner we can avoid exact solution of the linearized state and adjoint equations, or even the solves with the state and adjoint equations preconditioners.

**3. Lagrange-Newton-Krylov-Schur method.** The KKT optimality conditions (2.4) define a system of nonlinear equations. The Jacobian $\mathbf{K}$ of this system is termed the *KKT matrix*. A Newton step on the optimality conditions is given by

$$\begin{bmatrix} \mathbf{W} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{p}_x \\ \mathbf{p}_\lambda \end{Bmatrix} = - \begin{Bmatrix} \mathbf{g} + \mathbf{A}^T \boldsymbol{\lambda} \\ \mathbf{c} \end{Bmatrix} \quad (\text{ or } \mathbf{K}\mathbf{v} = -\mathbf{h}), \tag{3.1}$$

where $\mathbf{p}_x$ and $\mathbf{p}_\lambda$ are the updates of $\mathbf{x}$ and $\boldsymbol{\lambda}$ from current to next iterations. Assuming sufficient smoothness, and that the initial guess is sufficiently close to a solution, Newton steps obtained by the above system will converge quadratically to a solution of the KKT system[16]. Thus, the forward solves required for reduced methods can be avoided by remaining in the full space of state and decision variables, since it is the reduction onto the decision space that necessitates the solves with $\mathbf{A}_s^{-1}$. Nevertheless, the full space approach presents difficulties too: a descent direction is not guaranteed, second derivatives are required, and the KKT system itself can be difficult to solve. The size of the KKT matrix is more than twice that of the forward problem, and it is expected to be very ill-conditioned. Ill-conditioning results not only from the ill-conditioning of the forward problem, but also from the different scales between first and second derivatives submatrices. Moreover, the system is indefinite; mixing negative and positive eigenvalues is known to slow down Krylov solvers. Therefore, a good preconditioner is essential to make the method efficient.

In this article we present an algorithm that uses a proper Newton method to solve the KKT optimality conditions. To compute the Newton step we solve the KKT system using an appropriate Krylov method. At the core of the algorithm lies the preconditioner $\mathbf{P}$ for the Krylov method, which in our case is an *inexact* version of the QN-RSQP algorithm. An outline of the LNKS algorithm is given by Algorithm 4.

This algorithm will produce the same steps as solving the optimization problem with N-RSQP (in exact arithmetic). It is easier to see the connection between RSQP and LNKS if we rewrite (3.1) in a block-partitioned form:

$$\begin{bmatrix} \mathbf{W}_{ss} & \mathbf{W}_{sd} & \mathbf{A}_s^T \\ \mathbf{W}_{ds} & \mathbf{W}_{dd} & \mathbf{A}_d^T \\ \mathbf{A}_s & \mathbf{A}_d & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{p}_s \\ \mathbf{p}_d \\ \mathbf{p}_\lambda \end{Bmatrix} = - \begin{Bmatrix} \mathbf{g}_s + \mathbf{A}_s^T \boldsymbol{\lambda} \\ \mathbf{g}_d + \mathbf{A}_d^T \boldsymbol{\lambda} \\ \mathbf{c} \end{Bmatrix}. \tag{3.2}$$

[4] E.g. for the limiting quadratic programming case, the popular BFGS quasi-Newton method is equivalent to conjugate gradients, which scales with the square root of the condition number of the reduced Hessian.

**Algorithm 4** Lagrange-Newton-Krylov-Schur

1: Choose $\mathbf{x}, \boldsymbol{\lambda}$
2: **loop**
3: Check for convergence
4: Compute $\mathbf{c}, \mathbf{g}, \mathbf{A}, \mathbf{W}$
5: Solve $\mathbf{P}^{-1}\mathbf{K}\mathbf{v} = -\mathbf{P}^{-1}\mathbf{h}$
6: Update $\mathbf{x}_+ = \mathbf{x} + \mathbf{p}_x$
7: Update $\boldsymbol{\lambda}_+ = \boldsymbol{\lambda} + \mathbf{p}_\lambda$
8: **end loop**

RSQP is equivalent to a block-row elimination; given $\mathbf{p}_d$, solve the last block of equations for $\mathbf{p}_s$, then solve the first to find $\mathbf{p}_\lambda$, and finally solve the middle one for $\mathbf{p}_d$, the search direction for the decision variables. Therefore RSQP can be written as a particular factorization of the KKT matrix:

$$\mathbf{K} = \begin{bmatrix} \mathbf{W}_{ss}\mathbf{A}_s^{-1} & \mathbf{0} & \mathbf{I} \\ \mathbf{W}_{ds}\mathbf{A}_s^{-1} & \mathbf{I} & \mathbf{A}_d^T\mathbf{A}_s^{-T} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{A}_s & \mathbf{A}_d & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_z & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_{sd} - \mathbf{W}_{ss}\mathbf{A}_s^{-1}\mathbf{A}_d & \mathbf{A}_s^T \end{bmatrix}. \tag{3.3}$$

Note that these factors are permutable to block triangular form, ( and therefore this is equivalent to a block-LU factorization) and that $\mathbf{W}_z$ is the Schur complement for $\mathbf{p}_d$.

This block factorization suggests a preconditioner created by replacing the reduced Hessian $\mathbf{W}_z$ with approximation $\mathbf{B}_z$, for example a quasi-Newton approximation. However, we still require four forward solves per Krylov iteration. One way to restore the two solves per iteration of QN-RSQP is to, in addition, drop second order information from the preconditioner, just as one does when going from N-RSQP to QN-RSQP. A further simplification of the preconditioner is to replace the exact forward operator $\mathbf{A}_s$ by an approximation $\tilde{\mathbf{A}}_s$, which could be any appropriate forward problem preconditioner. With these changes, *no* forward solves need to be performed at each inner iteration. Thus, the work per inner iteration becomes linear in the state and adjoint variable dimension (e.g. when $\tilde{\mathbf{A}}_s$ is a constant-fill domain decomposition approximation). Furthermore, when $\mathbf{B}_z$ is based on a limited-memory quasi-Newton update, the work per inner iteration is also linear in the decision variable dimension. Since all of the steps involved in an inner iteration not only require linear work but are also readily parallelized at the fine-grained level, we conclude that each inner (KKT) iteration will have high parallel efficiency and scalability.

Scalability of the entire method additionally requires mesh-independence of both inner and outer iterations. Newton methods (unlike quasi-Newton) are often characterized by a number of iterations that is independent of problem size [1]. With an "optimal" forward preconditioner and a good $\mathbf{B}_z$ approximation, we can hope that the number of inner iterations is also insensitive to the problem size. Scalability with respect to both state and decision variables would then result.

**3.1. Preconditioning variants for the KKT system.** Below we present several preconditioners for the KKT matrix. They are based on the block-factorization of the KKT matrix and therefore are indefinite matrices. To examine separately the effects of discarding the Hessian terms and approximating the forward solver, we define four different variations. The subscript denotes the number of the forward PDE operator per Krylov iteration, and a tilde mark on (top of) the preconditioner means that a forward/adjoint solve is replaced by a single application of its preconditioner.

- Preconditioner $\mathbf{P}_4$ replaces $\mathbf{W}_z$ by $\mathbf{B}_z$ and retains the four linearized solves per iteration. The preconditioner is

$$\mathbf{P}_4 = \begin{bmatrix} \mathbf{W}_{ss}\mathbf{A}_s^{-1} & \mathbf{0} & \mathbf{I} \\ \mathbf{W}_{ds}\mathbf{A}_s^{-1} & \mathbf{I} & \mathbf{A}_d^T\mathbf{A}_s^{-T} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{A}_s & \mathbf{A}_d & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_z & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_y & \mathbf{A}_s^T \end{bmatrix}, \tag{3.4}$$

  and the preconditioned KKT matrix is

$$\mathbf{P}_4^{-1}\mathbf{K} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_z\mathbf{B}_z^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{3.5}$$

- Preconditioner $\mathbf{P}_2$ replaces $\mathbf{W}_z$ by $\mathbf{B}_z$ and discards all other Hessian terms, resulting in two linearized solves per iteration. The preconditioner is

$$\mathbf{P}_2 = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{A}_d^T\mathbf{A}_s^{-T} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{A}_s & \mathbf{A}_d & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_z & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_s^T \end{bmatrix}, \tag{3.6}$$

  and the preconditioned KKT matrix is

$$\mathbf{P}_2^{-1}\mathbf{K} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}_y^T\mathbf{A}_s^{-1} & \mathbf{W}_z\mathbf{B}_z^{-1} & \mathbf{0} \\ \mathbf{W}_{ss}\mathbf{A}_s^{-1} & \mathbf{W}_y\mathbf{B}_z^{-1} & \mathbf{I} \end{bmatrix}. \tag{3.7}$$

  Note that the spectrum of the preconditioned KKT matrix is unaffected by dropping the second derivative terms.
- Preconditioner $\tilde{\mathbf{P}}_4$ replaces $\mathbf{W}_z$ by $\mathbf{B}_z$ and $\mathbf{A}_s$ by $\tilde{\mathbf{A}}_s$, and retains all other Hessian terms, resulting in four preconditioner applications but no forward/adjoint solves. The preconditioner is

$$\tilde{\mathbf{P}}_4 \begin{bmatrix} \mathbf{W}_{ss}\tilde{\mathbf{A}}_s^{-1} & \mathbf{0} & \mathbf{I} \\ \mathbf{W}_{ds}\tilde{\mathbf{A}}_s^{-1} & \mathbf{I} & \mathbf{A}_d^T\tilde{\mathbf{A}}_s^{-T} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{A}}_s & \mathbf{A}_d & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_z & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{W}}_y & \tilde{\mathbf{A}}_s^T \end{bmatrix}, \tag{3.8}$$

  and the preconditioned KKT matrix is

$$\tilde{\mathbf{P}}_4^{-1}\mathbf{K} = \begin{bmatrix} \mathbf{I}_s & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_z\mathbf{B}_z^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_s^T \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathcal{O}(\mathbf{E}_s) & \mathbf{0} \\ \mathcal{O}(\mathbf{E}_s) & \mathcal{O}(\mathbf{E}_s) & \mathcal{O}(\mathbf{E}_s) \\ \mathcal{O}(\mathbf{E}_s) & \mathcal{O}(\mathbf{E}_s) & \mathbf{0} \end{bmatrix}, \tag{3.9}$$

  where $\mathbf{E}_s := \mathbf{A}_s^{-1} - \tilde{\mathbf{A}}_s^{-1}$ and $\mathbf{I}_s := \mathbf{A}_s\tilde{\mathbf{A}}_s^{-1}$. $\mathbf{W}_y$ is given by (2.7) with the difference that in $\mathbf{Z}$ , $\mathbf{A}_s^{-1}$ is being replaced by $\tilde{\mathbf{A}}_s^{-1}$. Clearly, $\mathbf{E}_s = \mathbf{0}$ and $\mathbf{I}_s = \mathbf{I}$ for exact forward solves.
- Preconditioner $\tilde{\mathbf{P}}_2$ replaces $\mathbf{W}_z$ by $\mathbf{B}_z$ and $\mathbf{A}_s$ by $\tilde{\mathbf{A}}_s$, and drops all other Hessian terms, resulting in two preconditioner applications and no forward/adjoint solves. The preconditioner is

$$\tilde{\mathbf{P}}_2 = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{A}_d^T\tilde{\mathbf{A}}_s^{-T} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{A}}_s & \mathbf{A}_d & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_z & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \tilde{\mathbf{A}}_s^T \end{bmatrix}, \tag{3.10}$$

and the preconditioned KKT matrix is

$$\tilde{\mathbf{P}}_2^{-1}\mathbf{K} = \begin{bmatrix} \mathbf{I}_s & \mathbf{0} & \mathbf{0} \\ \tilde{\mathbf{W}}_y^T & \mathbf{W}_z\mathbf{B}_z^{-1} & \mathbf{0} \\ \mathbf{W}_{ss}\mathbf{A}_s^{-1} & \tilde{\mathbf{W}}_y^T & \mathbf{I}_s^T \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathcal{O}(\mathbf{E}_s) & \mathbf{0} \\ \mathbf{0} & \mathcal{O}(\mathbf{E}_s) & \mathcal{O}(\mathbf{E}_s) \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}. \quad (3.11)$$

We review some basic properties of the Krylov methods that will allow us to understand the effectiveness of the preconditioners. The performance of a Krylov method depends highly on the preconditioned operator spectrum [21, 37]. In most Krylov methods the number of iterations required to obtain the solution is at most equal to the number of distinct eigenvalues (in exact arithmetic). When solving $\mathbf{Kv} = -\mathbf{h}$ such methods satisfy the following relation for the residual at the $i_{th}$ iteration:

$$\mathbf{r}_i = \text{span}\left\{\mathbf{r}_0 + \mathbf{K}\mathbf{r}_0 + \mathbf{K}^2\mathbf{r}_0 + \cdots + \mathbf{K}^i\mathbf{r}_0\right\} = \psi(\mathbf{K})\mathbf{r}_0,\ \psi \in \mathcal{P}_i.$$

Here $\mathbf{v}_0$ is the initial guess, $\mathbf{r}$ is defined as $\mathbf{r} = \mathbf{h} + \mathbf{K}\mathbf{v}$, $\mathbf{r}_0 = \mathbf{h} + \mathbf{K}\mathbf{v}_0$, and $\mathcal{P}_i$ is the space of monic polynomials of degree at most $i$. Minimum residual methods like MINRES and GMRES determine a polynomial $\psi$ so that[5]

$$\|\mathbf{r}_i\| = \min_{\psi \in \mathcal{P}_i} \|\psi(\mathbf{K})\mathbf{r}_0\|.$$

If $\mathbf{K}$ is diagonalizable (with spectrum $\mathcal{S}(\mathbf{K})$ and $\mathbf{X}$ the matrix or its eigenvectors), then[6]

$$\frac{\|\mathbf{r}_i\|}{\|\mathbf{r}_0\|} \leq \text{cond}(\mathbf{X}) \min_{\lambda \in \mathcal{S}(\mathbf{K})} |\psi(\lambda)|. \quad (3.12)$$

Additionally, if the spectrum lies on the positive real axis, it can be shown that:

$$\frac{\|\mathbf{r}_i\|}{\|\mathbf{r}_0\|} \leq \text{cond}(\mathbf{X})\frac{\lambda_{max}}{\lambda_{min}}. \quad (3.13)$$

From (3.12) we can see why clustering the eigenvalues is important. Indeed, if the eigenvalues are clustered, then the solution polynomial in the right hand side of (3.12) can be approximated by the zeros of a low degree polynomial (resulting in fewer Krylov iterations). When $\mathbf{K}$ is normal (unitarily diagonalizable) then $\text{cond}(\mathbf{X}) = 1$; in this case the estimates (3.12) and (3.13) are known to be sharp [21].

The preconditioned KKT matrix $\mathbf{P}_4^{-1}\mathbf{K}$ is a block diagonal matrix with two unit eigenvalues and the $m$ eigenvalues of the preconditioned reduced Hessian. It is immediate that GMRES takes $\mathcal{O}(\text{cond}(\mathbf{B}_z^{-1}\mathbf{W}_z))$ or at most $m + 2$ steps to converge. This is similar to the complexity estimates for a Krylov-based solution of the $\mathbf{B}_z$-preconditioned reduced Hessian in N-RSQP. Preconditioner (3.6) has the same effect on the spectrum of the KKT matrix, but the preconditioned KKT system is no longer a normal operator. Yet, if $\mathbf{P}_2^{-1}\mathbf{K}$ is diagonalizable and its eigenvector matrix well conditioned, relation (3.13) is still a good indicator of the effectiveness of the preconditioner.

A more efficient approach, of course, is to replace the exact forward operators $\mathbf{A}_s$ by their forward preconditioners $\tilde{\mathbf{A}}_s$. In this case, the preconditioned KKT matrix assumes a more complicated structure. We write $\tilde{\mathbf{P}}_4^{-1}\mathbf{K}$ as the sum of two matrices; the second matrix in this sum includes terms that approach $\mathbf{0}$ as the forward preconditioner improves. The spectrum of

[5] Throughout this paper we use the vector 2-norm for norms and condition numbers.

[6] Here $\lambda$ is temporary overloaded to denote an eigenvalue of $\mathbf{K}$.

the first matrix in (3.9) is given by $\mathcal{S}_n = \mathcal{S}(\mathbf{B}_z^{-1}\mathbf{W}_z) \cup \mathcal{S}(\tilde{\mathbf{A}}_s^{-1}\mathbf{A}_s)$. If $\tilde{\mathbf{A}_s}^{-1}\mathbf{A}_s$ is normal, then by the Bauer-Fike theorem [20] the eigenvalues are not sensitive to small KKT matrix perturbations and $\mathcal{S}(\tilde{\mathbf{P}}_4^{-1}\mathbf{K}) \cong \mathcal{S}_n$. Hence, a good preconditioner for the forward problem would bring the spectrum of $\tilde{\mathbf{P}}_4^{-1}\mathbf{K}$ close to the spectrum of $\mathbf{P}_4^{-1}\mathbf{K}$. Thus, we would expect a similar iteration count to N-RSQP, but at a much reduced cost per iteration—since the PDE solves would be replaced by preconditioner applications. For preconditioner $\tilde{\mathbf{P}}_2$, the left matrix in For preconditioner (3.11) is not normal and may have ill-conditioned eigenvalues. In our numerical experiments, however, the number of Krylov iterations was unaffected by switching from $\tilde{\mathbf{P}}_4$ to $\tilde{\mathbf{P}}_2$.

Finally, we comment on our choice of a Krylov method for solving the symmetric indefinite KKT system (3.3) that has been preconditioned with any of the four preconditioners described above. The most popular method for large symmetric indefinite systems is the MINRES method. However, MINRES requires positive-definite preconditioners, and the KKT preconditioners defined above are indefinite. Indefinite preconditioners are allowed in the quasi-minimum residual (QMR) method [17]. The version commonly used is the transpose-free QMR algorithm, which is designed for general nonsymmetric problems and requires two matrix-vector multiplications per Krylov iteration. In our implementation we use another variant, described in in [17], that exploits symmetry and uses only one matrix-vector multiplication per iteration.

**3.2. Preconditioners for the reduced Hessian.** A very important component of the LNKS preconditioner is an approximation to the inverse of the reduced Hessian. In the preceding sections we suggested a quasi-Newton approximation for the reduced Hessian—in the spirit of using QN-RSQP as a preconditioner. Below, we elaborate upon this choice, and describe a two-step stationary iterative preconditioner that we use either alone, or in conjunction with the quasi-Newton preconditioner.

**Limited memory Quasi-Newton updates.** Quasi-Newton methods have been used to approximate the reduced Hessian matrix in constrained optimization, not as preconditioners but as drivers. They are thus natural candidates to precondition the exact reduced Hessian. In particular, we use the BFGS update [31, §8.1]. At each outer iteration we compute

$$\mathbf{s} = \mathbf{p}_d^{(+)} - \mathbf{p}_d,$$
$$\mathbf{y} = \mathbf{g}_z^{(+)} - \mathbf{g}_z,$$

and update the approximation to the inverse of the reduced Hessian by

$$\mathbf{B}_+^{-1} = (I - \omega \mathbf{y}\mathbf{s}^T)^T \mathbf{B}^{-1} (I - \omega \mathbf{y}\mathbf{s}^T) + \omega \mathbf{s}\mathbf{s}^T, \;\; \omega = \frac{1}{\mathbf{y}^T\mathbf{s}}. \tag{3.14}$$

If we encounter an iterate that has $\mathbf{y}^T\mathbf{s} \leq \kappa\|\mathbf{y}\|$ we skip the update. $\mathbf{B}^{-1}$ is dense; thus for large decision-space problems, storing this matrix (or a factorization of it) can be expensive or even impossible. Furthermore, applying the (dense) $\mathbf{B}^{-1}$ to a vector in parallel involves all-to-all communication and may be too expensive on a parallel computer. Thus, we have chosen a limited memory variant of BFGS (L-BFGS), which uses a limited number of vectors to approximate the inverse of the reduced Hessian. When updating we do not compute (3.14) but instead we simply store $\mathbf{s}$ and $\mathbf{y}$. Then the action of $\mathbf{B}^{-1}$ to a vector $\mathbf{x}$ is given in Algorithm 5. Analysis and details for the convergence properties of this algorithm can be found in [31, §9.1–9.2]. The initial inverse reduced Hessian approximation, $\mathbf{B}_{z,0}$, is generated by the stationary preconditioner presented below.

**Two-step stationary method**. Both as an initialization for the L-BFGS approximation above, as well as a stand-alone preconditioner, we turned to stationary iterative methods.

**Algorithm 5** Limited memory BFGS

1: $\mathbf{q} = \mathbf{x}$, and $l =$ number of stored vectors
2: **for** $i = k-1, k-2, \ldots, k-l$ **do**
3: $\quad \alpha_i = \omega_i \, \mathbf{s}_i^T \mathbf{q}$
4: $\quad \mathbf{q} = \mathbf{q} - \alpha_i \mathbf{y}_i$
5: **end for**
6: $\mathbf{z} = \mathbf{B}_{z,0}^{-1} \mathbf{q}$
7: **for** $i = k-l, k-l+1, \ldots, k-1$ **do**
8: $\quad \beta = \omega_i \, \mathbf{y}_i^T \mathbf{z}$
9: $\quad \mathbf{z} = \mathbf{z} + \mathbf{s}_i(\alpha_i - \beta)$
10: **end for**

Such methods retain a constant preconditioner while providing an approximate solution of the reduced Hessian system. To guarantee convergence, the most popular such methods (such as Jacobi and SOR) require the iteration matrix to have spectral radius less than one, which is difficult to guarantee in the general case. Furthermore, most such methods require some kind of splitting, which is not convenient with unassembled operators like the reduced Hessian. A method that does not have the above restrictions is a two-step stationary iterative method, first presented by Frankel in 1950 [3, §5.2]. The method is suitable for positive-definite matrices, but requires an accurate estimate of the minimum and maximum eigenvalues. The preconditioner based on this method can be arbitrarily effective, depending on the number of iterations $L$ it is carried to. If $\lambda_1 \leq \lambda_m$ are the estimates for the extreme eigenvalues of $\tilde{\mathbf{W}}_z$, then the application of the preconditioner to a vector $\mathbf{d}_{in}$ is given by Algorithm 6. Step 4

**Algorithm 6** Two-step stationary iterative reduced space preconditioner

1: $\rho = \frac{1-\lambda_1/\lambda_n}{1+\lambda_1/\lambda_n}$, $\alpha = \frac{2}{1+(1-\rho^2)^{1/2}}$, $\beta_0 = \frac{1}{\lambda_1+\lambda_n}$, $\beta = \frac{2\alpha}{\lambda_1+\lambda_n}$,
2: $\mathbf{r} = -\mathbf{d}_{in}$, $\mathbf{d}_0 = \mathbf{0}$, $\mathbf{d}_1 = \beta_0 \mathbf{r}$
3: **for** $i = 1 \ldots L$ **do**
4: $\quad \mathbf{r} = \tilde{\mathbf{W}}_z \mathbf{d}_1 - \mathbf{d}_{in}$
5: $\quad \mathbf{d} = \alpha \mathbf{d}_1 + (1-\alpha)\mathbf{d}_0 - \beta \mathbf{r}$
6: $\quad \mathbf{d}_0 = \mathbf{d}_1, \ \mathbf{d}_1 = \mathbf{d}$
7: **end for**

requires the action of the reduced Hessian on a vector. To avoid exact forward solves, we use the approximate reduced Hessian $\tilde{\mathbf{W}}_z$ instead of $\mathbf{W}_z$. $\tilde{\mathbf{W}}_z$ is obtained by (2.11) if we replace the linearized forward PDE operators $\mathbf{A}_s^{-1}$ and $\mathbf{A}_s^{-T}$ with their preconditioners $\tilde{\mathbf{A}}_s^{-1}$ and $\tilde{\mathbf{A}}_s^{-T}$. Like CG, the method's convergence rate depends on the square root of the condition number of the (approximate) reduced Hessian. It parallelizes well because it does not require any inner products, and because in our context the matrix-vector products require just two forward preconditioner applications. To obtain estimates of the extremal eigenvalues, we use a few iterations of the Lanczos method (once per Newton iteration).

**Other possibilities for reduced Hessian preconditioning.** In addition to two aforementioned preconditioners we use, there are other possibilities, which we summarize below.

- **Incomplete factorizations**. Incomplete factorizations are popular and robust preconditioners, but an assembled matrix (modulo some exceptions for structured grids) is required. Not only is the reduced Hessian expensive to assemble, but it is in general dense and thus impossible to store for a large number of decision variables. An incomplete factorization would be feasible only if the exact solves are replaced

by the forward preconditioner and if some kind of sparsity is enforced (perhaps via element-by-element computations and threshold-ILU methods).

- **Sparse approximate inverse techniques**. SPAI techniques are attractive since they require just matrix-vector multiplications and a given sparsity pattern. In [13] an analysis for CFD-related Schur-complements shows that this method is very promising.
- **Range space preconditioners**. In [30] two different reduced Hessian preconditioners are presented, one based on a state-Schur complement factorization and the second on power series expansions. It is assumed that a (non-singular approximation) $\tilde{\mathbf{W}}^{-1}$ is available. The basic component of these preconditioners is the application of $\mathbf{Z}^T\tilde{\mathbf{W}}^{-1}\mathbf{Z}$ on a vector. We did not test this method because in our problems $\mathbf{W}$ has thousands of zero eigenvalues and it is not clear how to construct an approximation to $\mathbf{W}^{-1}$. If $\mathbf{W}_{dd}$ is non-singular then a block-Jacobi-ILU technique could be used to approximate $\mathbf{W}_z^{-1}$ with $\mathbf{W}_{dd}^{-1}$ (or an approximation $\tilde{\mathbf{W}}_{dd}^{-1}$).
- **Krylov self-preconditioning**. Another option is to take a few CG iterations in the reduced space at each preconditioner application. Since we want to avoid solving exactly the forward problem we replace $\mathbf{A}_s^{-1}$ with $\tilde{\mathbf{A}}_s^{-1}$ in (2.11). We experimented with this approach but found that full convergence of CG was needed to avoid loss of orthogonality for the Krylov vectors. This slowed down the method significantly. We have experimented also with flexible GMRES for the KKT solver, but the nice properties of a symmetric Krylov solver are lost and the algorithm was slow to converge.
- **Infinite dimension-based preconditioners**. For PDE-constrained optimization problems one may be able to derive an expression for the infinite-dimensional reduced Hessian. A discretization of this expression can be then be used to precondition the reduced Hessian, e.g. [2].

In the next section we discuss results from numerical experiments that we conducted to assess the effectiveness of the preconditioners. To isolate the effects of the Krylov-Schur solver on LNKS, we will study problems with linear constraints and quadratic objective functions. For this class of problems, the KKT system is linear, and thus Newton's method requires just one step to converge.

**4. Numerical results.** The LNKS method has been tested on two different quadratic optimization problems. Both cases have the 3D Stokes equations as the PDE constraints. The decision variables are Dirichlet boundary conditions on some portion of the boundary. The first test case is a matching velocity problem for a Poiseuille flow and the second is an energy dissipation minimization problem for a flow around a cylinder. In this section, we define the two test problems, discuss a forward preconditioner (which as we have seen plays a key role in the effectivess of the KKT preconditioner), and provide numerical results for the two test problems.

**4.1. Optimal control of Stokes flows.** An optimal flow control problem can be stated as follows:

$$\min_{\boldsymbol{u},\boldsymbol{u}_d,p} \mathcal{J}(\boldsymbol{u},\boldsymbol{u}_d,p) \tag{4.1}$$

$$\begin{aligned} \text{subject to} \quad & -\nu\nabla\cdot(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^T)+\nabla p=\boldsymbol{b} \text{ in } \Omega,\\ & \nabla\cdot\boldsymbol{u}=0 \quad \text{in} \quad \Omega,\\ & \boldsymbol{u}=\boldsymbol{u}^* \quad \text{on} \quad \Gamma_u,\\ & \boldsymbol{u}=\boldsymbol{u}_d \quad \text{on} \quad \Gamma_d,\\ & -p\,\boldsymbol{n}+\nu(\nabla\boldsymbol{u}+\nabla\boldsymbol{u}^T)\boldsymbol{n}=\boldsymbol{0} \quad \text{on} \quad \Gamma_N. \end{aligned} \tag{4.2}$$

Here $\boldsymbol{u}$ is the fluid velocity, $p$ is the pressure, $\nu$ is a non-dimensional viscosity, $\boldsymbol{b}$ is a body force, and $\boldsymbol{n}$ is the outward unit normal vector on the boundary. The first problem we studied is a velocity matching optimal control problem. A velocity profile is prescribed on the inflow

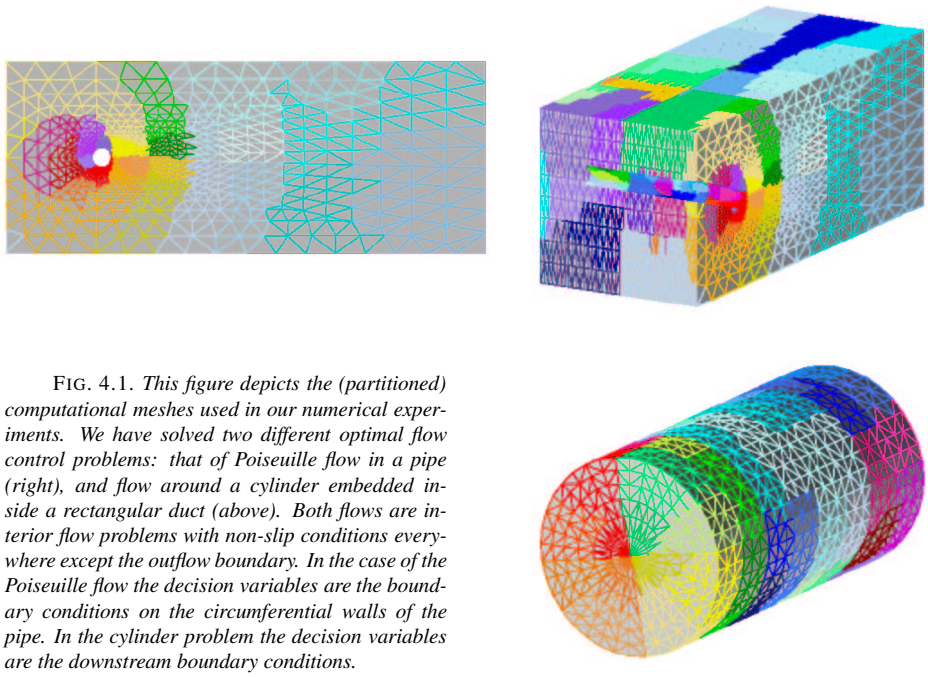

FIG. 4.1. *This figure depicts the (partitioned) computational meshes used in our numerical experiments. We have solved two different optimal flow control problems: that of Poiseuille flow in a pipe (right), and flow around a cylinder embedded inside a rectangular duct (above). Both flows are interior flow problems with non-slip conditions everywhere except the outflow boundary. In the case of the Poiseuille flow the decision variables are the boundary conditions on the circumferential walls of the pipe. In the cylinder problem the decision variables are the downstream boundary conditions.*

boundary $\Gamma_u$, and we specify a traction-free outflow boundary $\Gamma_N$. The velocities $\boldsymbol{u}_d$, defined on $\Gamma_d$, are the decision variables. In this example, $\boldsymbol{u}^*$ is taken as a Poiseuille flow inside a pipe, and the decision variables correspond to boundary velocities on the circumferential surface of the pipe (Fig 4.1). The objective functional is given by

$$\mathcal{J}(\boldsymbol{u}, \boldsymbol{u}_d, p) = \frac{1}{2} \int_\Omega (\boldsymbol{u}^* - \boldsymbol{u})^2 \, d\Omega + \frac{\rho}{2} \int_{\Gamma_d} \boldsymbol{u}_d^2 \, d\Omega,$$

where $\rho$ penalizes the "cost" of the controls. For this problem the exact solution is given by $\boldsymbol{u}_d = \mathbf{0}$.

In the second problem we examine the flow around a cylinder (which is anchored inside a rectangular duct). A quadratic velocity profile is used as an inflow Dirichlet condition and we prescribe a traction-free outflow. The decision variables are defined to be the velocities on the downstream portion of the cylinder surface. In this problem the objective functional is given by

$$\mathcal{J}(\boldsymbol{u}, \boldsymbol{u}_d, p) = \frac{1}{2} \int_\Omega |\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T|^2 \, d\Omega + \frac{\rho}{2} \int_{\Gamma_d} \boldsymbol{u}_d^2 \, d\Omega.$$

The Stokes equations are discretized by the Galerkin finite element method, using tetrahedral Taylor-Hood elements [22]. We implemented the reduced-space algorithm as well as the LNKS method with the four different preconditioners. Our code is based on the PETSc library [4, 5, 6], and makes use of PETSc parallel domain-decomposition preconditioners for the approximate forward solves. We use the symmetric QMR for both the KKT and Stokes linear systems. The two flow control problems have quadratic objective functions and linear constraints and thus Newton's method takes only one iteration to converge. Hence, it is

not possible to build a quasi-Newton approximation (as described above) to use within the KKT preconditioner (but see [29] for other options). Instead, we use the two-step stationary algorithm to precondition the reduced Hessian.

**4.2. The Stokes forward preconditioner.** It is evident that one of the two major components of the LNKS method is the forward solver preconditioner (the choice of which is, of course, problem-dependent). The Stokes problem, in algebraic form, is given by

$$\begin{bmatrix} \mathbf{V} & \mathbf{P}^T \\ \mathbf{P} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{u} \\ \mathbf{p} \end{Bmatrix} = \begin{Bmatrix} \mathbf{b}_u \\ \mathbf{b}_p \end{Bmatrix}. \tag{4.3}$$

In our first round of numerical experiments (results are reported in [9, 10]) we used the following preconditioner:

$$\begin{bmatrix} \mathbf{V}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^{-1} \end{bmatrix}. \tag{4.4}$$

PETSc's block-Jacobi preconditioners with local ILU(0) were used to provide a domain decomposition approximation of $\mathbf{V}^{-1}$ and $\mathbf{M}^{-1}$, i.e. the discrete Laplacian and discrete pressure mass matrices, respectively. We found the performance of this preconditioner unsatisfactory, especially when it was part of the KKT preconditioner. For this reason we have switched to a preconditioner that is based on the following exact factorization of the inverse of (4.3)

$$\begin{bmatrix} \mathbf{I} & -\mathbf{V}^{-1}\mathbf{P}^T \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{V}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{P}\mathbf{V}^{-1} & \mathbf{I} \end{bmatrix}, \tag{4.5}$$

where $\mathbf{S} := -\mathbf{P}\mathbf{V}^{-1}\mathbf{P}^T$ is the Schur complement for the pressure. Based on this factorization, the preconditioner is defined by replacing the exact solves $\mathbf{V}^{-1}$ with domain decomposition approximations $\tilde{\mathbf{V}}^{-1}$. For the pressure Schur complement block, we use the two-step stationary method described in Algorithm 6. Performance statistics are presented in Table 4.1. We can see that the factorization-based preconditioner reduces significantly solution times compared to the earlier block-Jacobi variant. Nevertheless, the efficiency of the new preconditioner is still mesh-dependent. It is known that block-Jacobi-ILU preconditioners require work that is superlinear in problem size. This can be overcome with an additive Schwarz domain-decomposition preconditioner, with generous overlap and a coarse grid component [12].

**4.3. Numerical results for Poiseuille flow problem.** Results for the Poiseuille flow problem are presented in Table 4.2. We have solved for five different problem sizes on up to 256 processors in order to assess the performance and scalability of the LNKS algorithm. Our first and most important finding is that the fastest variant is LNKS-II (which uses $\tilde{\mathbf{P}}_2$ as a preconditioner, i.e. it performs two approximate forward solves per KKT iteration), which is approximately *30 times* faster than QN-RSQP. Another observation is that LNKS-I (which uses $\mathbf{P}_2$, i.e. two exact forward solves), despite its discarding second order terms and approximating the reduced Hessian, is very effective in reducing the number of iterations—note the difference in KKT iterations between LNK (unpreconditioned) and LNKS-I for each problem instance. Indeed, the mesh-independence of LNKS-I iterations suggests that the two-step stationary preconditioner is very effective for the reduced Hessian,[7] and that if a forward preconditioner that is spectrally equivalent to the forward operator is available, one can expect

[7] Also, the weak increase in quasi-Newton iterations suggests that the reduced Hessian for this problem is well-conditioned.

TABLE 4.1

*Effectiveness of various preconditioners for Stokes forward solver with increasing number of processors. The first two columns list the number of processors (***PEs***) and the problem size (***n***). The remaining columns give the number of Krylov iterations required to satisfy* $\|\mathbf{r}\|/\|\mathbf{r}_0\| \leq 1 \times 10^{-8}$*;* **none** *signifies no preconditioner;* **bj** *is a block-Jacobi preconditioner, with each diagonal block mapped to a processor and approximated by ILU(0);* **psc3** *is the factorization-based preconditioner, with three (two-step) stationary iterations for the Schur complement block* $\mathbf{S}$*;* **psc7** *employs seven stationary iterations. In parentheses is wall-clock time in seconds on the Pittsburgh Supercomputing Center's CRAY T3E-900.*

POISEUILLE FLOW

| PEs | n | none | bj | psc3 | psc7 |
|---|---|---|---|---|---|
| 16 | 64,491 | 18,324 (250) | 1,668 (86) | 140 (30) | 128 (39) |
| 32 | 114,487 | 24,334 (620) | 2,358 (221) | 197 (39) | 164 (45) |
| 64 | 280,161 | 27,763 (1410) | 3,847 (515) | 294 (71) | 249 (84) |
| 128 | 557,693 | 32,764 (1297) | 5,010 (620) | 446 (109) | 357 (123) |
| 256 | 960,512 | 49,178 (2272) | 6,531 (780) | 548 (122) | 389 (128) |

FLOW AROUND A CYLINDER

| PEs | n | none | bj | psc3 | psc7 |
|---|---|---|---|---|---|
| 16 | 50,020 | 24,190 (720) | 2,820 (206) | 251 (72) | 234 (103) |
| 32 | 117,048 | 35,689 (1284) | 4,512 (405) | 363 (120) | 327 (176) |
| 64 | 389,440 | 41,293 (1720) | 7,219 (1143) | 581 (332) | 497 (456) |
| 128 | 615,981 | 52,764 (2345) | 10,678 (1421) | 882 (421) | 632 (512) |
| 256 | 941,685 | 71,128 (3578) | 13,986 (1578) | 1,289 (501) | 702 (547) |

mesh-independence for the entire KKT preconditioner. Even if a suboptimal forward preconditioner is used (witnessed by the growth in KKT iterations with increasing problem size for LNKS-II), one can trade off increasing KKT iterations for reduced work per iteration to realize a significant reduction in overall work. For example, in the largest problem size case, we give up a factor of 24 in KKT iterations, but we gain a factor of 4.5 in execution time.

The number of unpreconditioned KKT iterations illustrates the severe ill-conditioning of the KKT matrix (even for a problem as simple as controlling a Poiseuille flow). The comparison between LNKS-I and QN-RSQP simply illustrates the better convergence properties of a true Newton, as opposed to quasi-Newton, method. The comparison between LNKS-II and LNKS-III reveals the significance of a good preconditioner for the reduced space. When the two-step preconditioner is used, LNKS runs approximately twice as quickly as with no preconditioning (and requires four times fewer iterations).

The scalability of LNKS is studied by tracking the execution time for LNKS as the problem size and number of processors increase proportionately. The problem size per processor is held (relatively) constant, and execution time increases from about 8 minutes for 16 processors (65K states, 7K decisions) to about 30 minutes for 256 processors (960K states, 40K decisions). One may conclude that the method is not scalable. However, a glance at the KKT iterations column reveals that the number of optimization iterations when using the LNKS-I variant (so that the effect of inexact forward problem solves is factored out) is largely independent of problem size, and thus the work efficiency of the LNKS should be very high. Furthermore, the Mflop rate drops (probably due to a poorer-quality mesh partition) only slowly as the problem size increases, suggesting good implementation efficiency. What, then, accounts for the increase in execution time?

Table 4.3 provides the answer. Following [24], the overall parallel efficiency $\eta$ (based on execution time) has been decomposed into an implementation efficiency $\eta_i$ (based on Mflop/s

TABLE 4.2

*Performance of LNKS and comparisons with QN-RSQP for the Poiseuille flow problem as a function of increasing number of state and decision variables and number of processors. Here,* **QN-RSQP** *is quasi-Newton reduced-space SQP;* **LNK** *is the full-space Lagrange-Newton-Krylov method with no preconditioning for the KKT system;* **LNKS-I** *is the* $\mathbf{P}_2$ *preconditioner—which requires two Stokes solves—combined with the two-step stationary preconditioner for the reduced Hessian (3 iterations); in* **LNKS-II** *the exact solves have been replaced by approximate solves; in* **LNKS-III** *the exact solves have been replaced by approximate ones and there is no preconditioning in the reduced space, i.e.* $\mathbf{B}_z = \mathbf{I}$*;* **QN iter** *reports the number of quasi-Newton iterations (the number of outer iterations for the LNKS variants is just one, since this is a quadratic optimization problem;* **KKT iter** *is the number of Krylov (symmetric QMR) iterations for the KKT system to converge; finally* **time** *is wall-clock time in seconds on a Cray T3E-900. Both QN-RSQP and LNKS methods are converged so that* $\|\mathbf{c}\|/\|\mathbf{c}_0\| \leq 1 \times 10^{-6}$ *and* $\|\mathbf{g}\|/\|\mathbf{g}_0\| \leq 1 \times 10^{-6}$*, in all cases but the unpreconditioned KKT case, in which the Krylov method was terminated when the number of iterations exceeded 500,000.*

| states<br>decisions | method | QN iter | KKT iter | time |
|---|---|---|---|---|
| 64,491 | QN-RSQP | 221 | — | 15,365 |
| 7,020 | LNK | — | 274,101 | 19,228 |
| (16 PEs) | LNKS-I | — | 27 | 1,765 |
| | LNKS-II | — | 170 | 482 |
| | LNKS-III | — | 1,102 | 969 |
| 114,487 | QN-RSQP | 224 | — | 19,493 |
| 10,152 | LNK | — | 499,971 | 39,077 |
| (32 PEs) | LNKS-I | — | 26 | 2,089 |
| | LNKS-II | — | 258 | 519 |
| | LNKS-III | — | 1,525 | 1225 |
| 280,161 | QN-RSQP | 228 | — | 33,167 |
| 18,144 | LNK | — | >500,000 | — |
| (64 PEs) | LNKS-I | — | 29 | 4,327 |
| | LNKS-II | — | 364 | 934 |
| | LNKS-III | — | 1,913 | 1,938 |
| 557,693 | QN-RSQP | 232 | — | 53,592 |
| 28,440 | LNK | — | >500,000 | — |
| (128 PEs) | LNKS-I | — | 29 | 6,815 |
| | LNKS-II | — | 603 | 1,623 |
| | LNKS-III | — | 2,501 | 3,334 |
| 960,512 | QN-RSQP | 241 | — | 63,865 |
| 40,252 | LNK | — | >500,000 | — |
| (256 PEs) | LNKS-I | — | 32 | 8,857 |
| | LNKS-II | — | 763 | 1,987 |
| | LNKS-III | — | 2,830 | 3,735 |

and which represents costs like latency, load imbalance and interprocessor communication), and a work efficiency $\eta_a$ (based on the number of optimization iterations), and the forward solver work efficiency $\eta_f$ (based on Table 4.1). Whereas the optimization algorithm and the implementation are reasonably scalable (84% and 87% efficiency over a 16-fold increase in number of processors), the forward solver's parallel efficiency drops to near 26% for the largest problem size. The last column, $\eta'$, gives an estimate of what the overall efficiency would be, had we not used the time measurement but instead had factored in the forward solver efficiency. The agreement between the last two columns makes apparent that the over-

TABLE 4.3

*Isogranular scalability results for the LNKS-I variant. (***PEs***) is the number of processors; (***max Mflop/s/PE***) is the per processor maximum (across PEs) sustained Mflop/s; (***Mflop/s/PE***) is the average sustained Mflop/s per processor. A comparison between the second and third columns is an indication of load imbalance. Implementation efficiency ($\eta_i$) is based on Mflop rate; optimization work efficiency ($\eta_a$) is based on number of optimization iterations; forward solver work efficiency ($\eta_f$) is based on the number of forward solver Kyrlov iterations* **psc3** *Table 4.1; overall efficiency ($\eta$) is based on execution time; ($\eta'$) is an estimate of the overall efficiency given by $\eta' = \eta_f \times \eta_i \times \eta_a$.*

| PEs | max Mflop/s/PE | Mflop/s/PE | $\eta_i$ | $\eta_a$ | $\eta_f$ | $\eta$ | $\eta'$ |
|---|---|---|---|---|---|---|---|
| 16 | 76.5 | 52.0 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 32 | 74.8 | 51.1 | 0.98 | 1.04 | 0.71 | 0.84 | 0.72 |
| 64 | 74.9 | 49.2 | 0.96 | 0.93 | 0.48 | 0.41 | 0.43 |
| 128 | 71.2 | 47.8 | 0.91 | 0.93 | 0.31 | 0.26 | 0.26 |
| 256 | 69.8 | 45.1 | 0.87 | 0.84 | 0.26 | 0.18 | 0.19 |

all efficiency of the algorithm depends greatly upon the forward solver, which is essentially a question of preconditioning. The parallel inefficiency of the forward preconditioner can be addressed by switching to a more scalable approximation than the one we are currently using (non-overlapping local ILU(0) block-Jacobi). With a better forward preconditioner, we anticipate good overall scalability.

Timings and flop measurements were performed by using PETSc logging routines, which were validated with native performance analyzers on the T3E and Origin platforms. At first glance CPU performance appears to be mediocre—less than 10% of the peak machine performance. Recall however, that unstructured grid computations are not cache coherent and therefore the bottleneck is in memory bandwidth and not in the CPU flop rate.

**Remark 1**. We do not advocate the use of exact forward solves within the LNKS context. If one can afford exact solves, then one should iterate in the reduced space. If both exact forward solves and Hessian terms are retained then $4$ solves per KKT-Krylov iteration are required; iterating with N-RSQP requires only 2 solves per $\mathbf{W}_z$-Krylov iteration. Even when the Hessian terms are dropped (which is equivalent to retaining only $\mathbf{g}_z$ on the right-hand side of the decision equation (in Algorithm 2) it seems natural to use CG in the reduced space (with a good preconditioner).

An order-of-magnitude argument can be used (assuming $\mathbf{A}_s$ is symmetric) to illustrate why it should be advantageous to stay in the full space when approximate solves are used. If the condition number of the preconditioned reduced Hessian is given by $\mu_m/\mu_1$ and the condition number of the preconditioned forward operator is given by $\lambda_n/\lambda_1$, the complexity for N-RSQP (Algorithm 2) is $\mathcal{O}(\sqrt{\mu_m/\mu_1} \times \sqrt{\lambda_n/\lambda_1} \times N)$. Assuming effective reduced Hessian and forward problem preconditioners, the expected complexity for the solution of the KKT system (3.9) is $\mathcal{O}(\sqrt{\max(\mu_m, \lambda_n)/\min(\mu_1, \lambda_1)} \times N)$. If, in addition, the spectra of the preconditioned forward and reduced Hessian operators overlap, then a reduced method has a condition number which is approximately the square of the KKT system's condition number.

**Remark 2**. We have tested the four different preconditioners for the LNKS algorithm. Here we report results only for preconditioners $\mathbf{P}_2$ and $\tilde{\mathbf{P}}_2$. In our numerical experiments preconditioners $\mathbf{P}_4$ and $\mathbf{P}_2$ took approximately the same number of KKT iterations to converge. Since $\mathbf{P}_2$ requires two solves fewer, it is twice as fast as $\mathbf{P}_4$ and for this reason we report results only for $\mathbf{P}_2$. The same is true when comparing the preconditioners $\tilde{\mathbf{P}}_4$ and $\tilde{\mathbf{P}}_2$ although the difference is less pronounced.

**Flow around a cylinder**. In order to further test the KKT solver we performed additional numerical experiments for a flow around a cylinder. Figure 4.2 compares the controlled and the uncontrolled flow. The optimizer drives the downstream surface of the cylinder to become

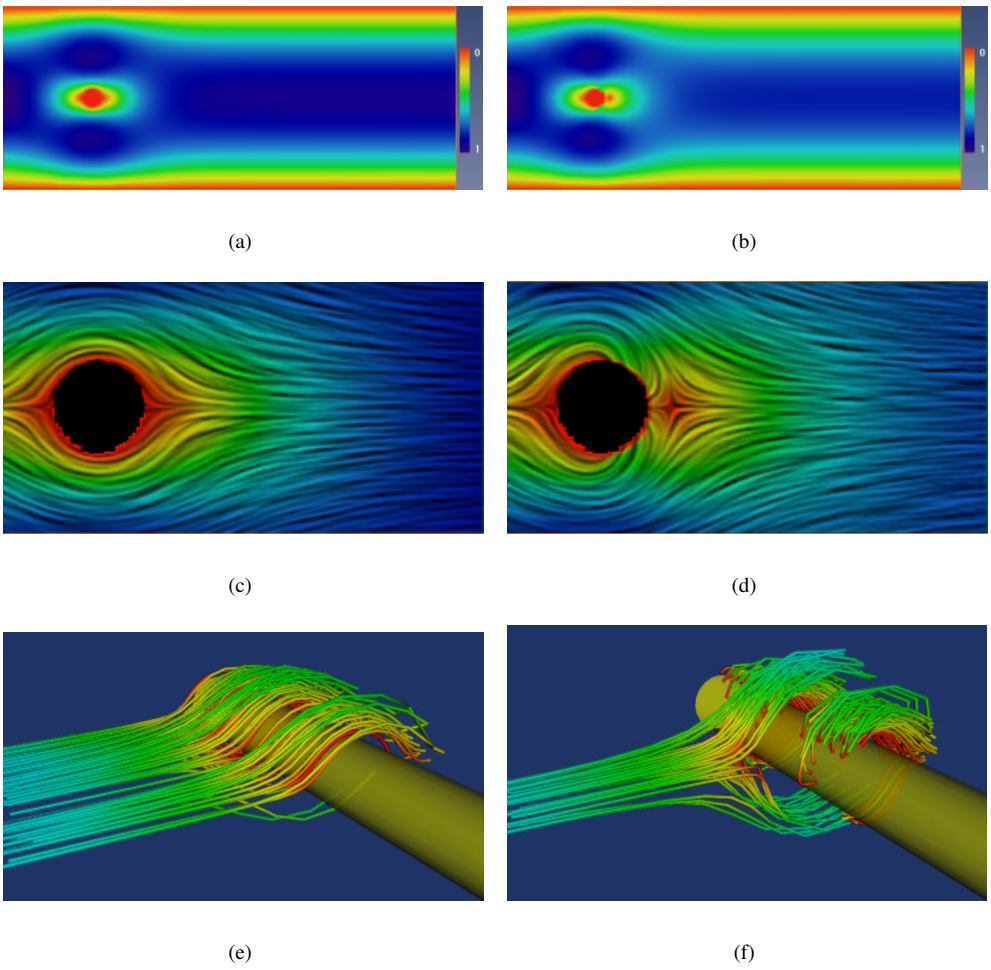


FIG. 4.2. *Optimal boundary control problem of a steady incompressible Stokes flow around a cylinder. The objective is to minimize the energy dissipation. The controls are injection points (velocity Dirichlet boundary conditions) on the downstream portion of the cylinder surface. The left column is the uncontrolled flow and the right column is the controlled one. Images (a) and (b) depict a color map of the velocity magnitude on a cross section at the middle of the duct. Comparing image (a) with (b) notice the lighter blue (or grey in b/w) throughout the downstream region. This indicates smaller velocities and thus reduced dissipation. Images (c) and (d) show a close up snapshot of the flow around the cylinder. The suction Dirichlet conditions (set by the optimizer) are clearly visible. Finally, the last two images show stream tubes of the flow around the cylinder. Although the flow appears to be more irregular (locally) for the controlled flow, overall the dissipation is reduced.*

a sink for the flow. As we can see in Fig 4.2 the locally controlled flow (around the cylinder) appears to be more irregular than the uncontrolled one. Yet, the overall dissipation is reduced. The results can be explained if we consider the dissipation function as the sum along both the upstream and the downstream part of the duct. Flow suction results in reduced mass flow (i.e. velocity) on the downstream portion of the duct and thus minimizes dissipation losses.

Table 4.4 presents the scalability results for this problem. We observe similar performance to the Poiseuille flow. The use of a Newton method accelerates the algorithm by

TABLE 4.4

*Performance and scalability of the LNKS algorithm for the control of a 3D flow around a cylinder. Here the objective function is the energy dissipation and the constraints are the Stokes equations.* **QN-RSQP** *is quasi-Newton reduced-space SQP;* **LNK** *is the the full-space Lagrange-Newton-Krylov method with no preconditioning;* **LNKS-I** *requires two exact solves per Krylov step combined with the two-step stationary preconditioner for the reduced Hessian; in* **LNKS-II** *the exact solves have been replaced by approximate solves;* **LNKS-III** *is the same as LNKS-II but the reduced Hessian is not preconditioned;* **time** *is wall-clock time in seconds on a T3E-900. The reduced gradient and the constraints were converged to a relative norm of* $1 \times 10^{-6}$.

| **states controls** | **method** | **N or QN iter** | **KKT iter** | **time** |
|---|---|---|---|---|
| 50,020 | QN-RSQP | 116 | — | 9,320 |
| 1,653 | LNK | 1 | 390,456 | 37,670 |
| (16 PEs) | LNKS-I | 1 | 41 | 6.833 |
| | LNKS-II | 1 | 1,101 | 3,100 |
| | LNKS-III | 1 | 1,312 | 1,812 |
| 117,048 | QN-RSQP | 120 | — | 27,669 |
| 2,925 | LNK | 1 | >500,000 | — |
| (32 PEs) | LNKS-I | 1 | 39 | 10,780 |
| | LNKS-II | 1 | 1,522 | 5,180 |
| | LNKS-III | 1 | 1,731 | 2,364 |
| 389,440 | QN-RSQP | 128 | — | 92,874 |
| 6,549 | LNK | 1 | $>$ 500,000 | — |
| (64 PEs) | LNKS-I | 1 | 50 | 34,281 |
| | LNKS-II | 1 | 2,987 | 18,451 |
| | LNKS-III | 1 | 3,637 | 15,132 |
| 615,981 | QN-RSQP | 132 | — | 113,676 |
| 8,901 | LNK | 1 | $>$ 500,000 | — |
| (128 PEs) | LNKS-I | 1 | 53 | 54,678 |
| | LNKS-II | 1 | 3,150 | 17,144 |
| | LNKS-III | 1 | 4,235 | 9,325 |
| 941,685 | QN-RSQP | 138 | — | 140,085 |
| 11,817 | LNK | 1 | $>$ 500,000 | — |
| (256 PEs) | LNKS-I | 1 | 52 | 59,912 |
| | LNKS-II | 1 | 4,585 | 20,384 |
| | LNKS-III | 1 | 5,687 | 11,028 |

a factor of two. Switching to the inexact preconditioners makes the method another 6 (or more) times faster. For the 256 processor problem (941,685 states and 11,817 controls) quasi-Newton RSQP required 38 hours, whereas the LNKS algorithm with the $\tilde{\mathbf{P}}_2$ preconditioner required only 3 hours, almost 13 times faster. Notice that the fastest LNKS variant for this example (LNKS-III) does not precondition in the reduced space. Although LNKS-II takes fewer iterations to converge, the cost per KKT Krylov iteration is increased due to load imbalance—the Poiseuille flow problem has more uniform distribution of decision variables across processors.

The CPU performance has dropped compared to the Poiseuille flow results, but not significantly, as we can see in Table 4.5. The overall efficiency appears to be governed by the

forward problem preconditioner; however there is a slight disagreement between the last two columns. An explanation can be found in $\eta_a$ which seems to drop faster than in the Poiseuille flow case. The reason is that the faster LNKS variant for the cylinder example does not precondition in the reduced space.

TABLE 4.5

*Isogranular scalability results for the LNKS-III variant (Preconditioner given by (3.10)) with no preconditioning for the reduced Hessian. The Mflop rates given are sustained Mflop per processor; the first column is maximum across processors and the second column lists the average rate. The difference is an indication of imbalance. Efficiency ($\eta_i$) is based on Mflop rate; work efficiency ($\eta_a$) is based on number of optimization iterations; forward solver work efficiency ($\eta_f$) is based on Table 4.1; overall efficiency ($\eta$) is based on execution time; ($\eta'$) is an estimate of the overall efficiency given by $\eta' = \eta_f \times \eta_i \times \eta_a$.*

| **PEs** | **max Mflop/s/PE** | **Mflop/s/PE** | $\eta_i$ | $\eta_a$ | $\eta_f$ | $\eta$ | $\eta'$ |
|---|---|---|---|---|---|---|---|
| 16 | 69.1 | 50.9 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 32 | 69.2 | 48.5 | 0.94 | 1.05 | 0.69 | 0.64 | 0.67 |
| 64 | 73.9 | 45.7 | 0.89 | 0.82 | 0.62 | 0.20 | 0.45 |
| 128 | 65.6 | 42.9 | 0.84 | 0.77 | 0.29 | 0.13 | 0.18 |
| 256 | 66.4 | 39.3 | 0.77 | 0.78 | 0.19 | 0.12 | 0.11 |

Our performance analyses are based on isogranular scaling. More common are fixed-problem size scalability analyses. Although these kinds of tests are very good indicators of the performance of an algorithm, they do not capture the important issue of mesh dependence of iterations. For completeness, we present a standard fixed-problem-size scalability analysis for the (117,048 states, 2,925 controls) cylinder problem for 4 different partitions and across two different platforms: a T3E-900 and an Origin 2000. Our experiments showed Origin to have superior performance, which is surprising since the T3E has a much faster interconnect. The fact that the Origin has bigger cache size is a possible explanation. Another observation

TABLE 4.6

*Fixed-size scalability analysis for the 117,028 state variables problem. The experiments were performed on a SGI Origin 2000 and on a T3E-900. The 450 MHz Compaq/Alpha 21164 processor on the T3E is equipped with 8 KB L1 cache and 96 KB L2 cache. The Origin uses the 250 MHz MIPS R10000 processor with 32 KB L1 and 4MB L2 cache. Bisection bandwidth is 128GB/s for the T3E and 20.5 GB/s for the Origin (128 PEs).*

CRAY T3E-900

| **procs** | **agr Gflop/s** | **its** | **time** | **speedup** | $\eta_i$ | $\eta_a$ | $\eta$ |
|---|---|---|---|---|---|---|---|
| 16 | 0.81 | 38 | 18,713 | 1.00 | 1.00 | 1.00 | 1.00 |
| 32 | 1.55 | 39 | 10,170 | 1.84 | 0.95 | 0.97 | 0.92 |
| 64 | 3 14 | 40 | 5,985 | 3.13 | 0.97 | 0.95 | 0.92 |
| 128 | 4.86 | 40 | 3,294 | 5.68 | 0.75 | 0.95 | 0.71 |

SGI ORIGIN 2000

| **procs** | **agr Gflop/s** | **its** | **time** | **speedup** | $\eta_i$ | $\eta_a$ | $\eta$ |
|---|---|---|---|---|---|---|---|
| 16 | 1.09 | 37 | 13,512 | 1.00 | 1.00 | 1.00 | 1.00 |
| 32 | 2.13 | 40 | 6,188 | 1.84 | 0.96 | 0.93 | 0.89 |
| 64 | 6.08 | 38 | 3,141 | 3.13 | 0.96 | 0.97 | 0.93 |
| 128 | 7.90 | 39 | 1,402 | 5.68 | 0.87 | 0.95 | 0.83 |

is that the effectiveness of the LNKS algorithm degrades with the number of processors. The

overall efficiency drops to 71% for the T3E and to 83% for the Origin. The work efficiency remains constant but part of it is hidden in $\eta$ because the time per preconditioner application increases. Recall that the condition number of a linear system preconditioned with block-Jacobi is approximately $\mathcal{O}((pn)^{1/6})$ and therefore the flop count increases with the number of processors $p$. Our forward solver implementation uses such a preconditioner and this is why we observe the decrease in overall efficiency. We have not conducted any measurements on latency and bandwidth dependencies on the number of processors. The inferior performance of the T3E is somewhat surprising but this would probably change had we increased the number of processors further[8].

**4.4. LNKS parallel scalability.** How scalable is the method, with respect to increasing problem size and number of processors? For scalability, we require that the work increases near-linearly with problem size (work scalability) and that it parallelizes well. Let us examine the major components:

**Formation of the KKT matrix–vector product.** For PDE-constrained optimization, the Hessian of the Lagrangian function and the Jacobian of the constraints are usually sparse with structure dictated by the mesh (particularly when the decision variables are mesh-related). Thus, formation of the matrix-vector product at each QMR iteration is linear in both state and decision variables, and parallelizes well due to a high computation-to-communication ratio and minimal sequential bottlenecks.

**Application of the QN-RSQP preconditioner.** The main work involved is application of the state Jacobian preconditioner $\tilde{\mathbf{A}}_s$ and its transpose, and "inversion" of an approximation to the reduced Hessian, $\mathbf{B}_z$. We can often make use of scalable, parallel state Jacobian preconditioners that requires $\mathcal{O}(n)$ work to apply (as in various domain decomposition preconditioners for elliptic problems). The stationary preconditioner for the reduced Hessian is also scalable since it only involves matrix-vector multiplications. Furthermore, when $\mathbf{B}_z$ is based on a limited-memory quasi-Newton update or the two-step stationary preconditioner (as in our implementation) the work is also linear in the decision variables. Despite the need for inner work, quasi-Newton updates and applications to a vector are easily parallelized. The same is true for the stationary preconditioner. Therefore, we conclude that application of the QN-RSQP preconditioner requires linear work and parallelizes well.

**The Krylov (inner) iteration.** As argued above, with an "optimal" state preconditioner and a good $\mathbf{B}_z$ approximation, we can anticipate that the number of inner (Krylov) iterations will be relatively insensitive to the problem size.

**The Lagrange-Newton (outer) iteration.** The number of outer (Newton) iterations is often independent of problem size for PDE-type problems, and the PDE-constrained optimization problems we have solved exhibit this type of behavior as well.

This combination of linear work per Krylov iteration, weak dependence of Krylov iterations on problem size, and independence of Lagrange-Newton iterations on problem size suggest a method that scales well with increasing problem size and number of processors.

**5. Conclusions.** The basic new component LNKS brings to PDE-constrained optimization is the use of QN-RSQP not as a driver but rather as a preconditioner for the KKT system. We advocate that in order to achieve algorithmic scalability, a proper Newton method is necessary. The method requires second derivatives (only matrix-vector multiplications) and the adjoint operator of the forward problem. The most important result is that we have presented a methodology by which the problem of devising a good preconditioner for the KKT system is reduced to that of finding a good preconditioner for the PDE operator.

[8] At the time these experiments were performed, the 256 partition on the Origin was not available.

The problems we have chosen to investigate are relatively simple, yet provide a reasonable testbed for algorithmic tuning and experimentation. The results obtained thus far are very encouraging: the full space Newton-Krylov optimization method with reduced-space preconditioning is by a factor of 10–30 faster than current reduced space methods. We have no reason to believe that other problems should behave very differently. Moreover, the method can be parallelized efficiently, and work efficiency can be achieved provided a good forward preconditioner is available. Scalability then results from the combination of these two.

In the companion paper we extend our discussion to nonlinear constraints and we examine issues as robustness and globalization of the LNKS method.

**Acknowledgments.** We thank the PETSc development group at Argonne National Lab for making this work possible. We also thank Jonathan Shewchuk of UC Berkeley for providing the meshing and partitioning routines Pyramid and Slice. Finally, we thank David Keyes of ICASE/Old Dominion University, David Young of Boeing, and the other members of the TAOS project—Roscoe Bartlett, Larry Biegler, Greg Itle, Ivan Malčević, and Andreas Wächter—for their useful comments.

## REFERENCES

[1] E. L. ALLGOWER, K. BÖHMER, F. A. POTRA, AND W. C. RHEINBOLDT, *A mesh-independence principle for operator equations and their discretizations*, SIAM Journal on Numerical Analysis, 23 (1986), pp. 160–169.

[2] E. ARIAN AND S. TA'ASAN, *Analysis of the Hessian for aerodynamic optimization*, Tech. Rep. 96-28, Institute for Computer Applications in Science and Engineering, 1996.

[3] O. AXELSSON, *Iterative Solution Methods*, Cambridge University Press, 1994.

[4] S. BALAY, K. BUSCHELMAN, W. D. GROPP, D. KAUSHIK, L. C. MCINNES, AND B. F. SMITH, *PETSc home page*. http://www.mcs.anl.gov/petsc, 2001.

[5] S. BALAY, W. D. GROPP, L. C. MCINNES, AND B. F. SMITH, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhauser Press, 1997, pp. 163–202.

[6] ———, *PETSc users manual*, Tech. Rep. ANL-95/11 - Revision 2.1.1, Argonne National Laboratory, 2001.

[7] A. BATTERMANN AND M. HEINKENSCHLOSS, *Preconditioners for Karush-Kuhn-Tucker matrices arising in the optimal control of distributed systems*, in Optimal control of partial differential equations, W. Desch, F. Kappel, and K. Kunisch, eds., vol. 126 of International Series of Numerical Mathematics, Birkhäuser Verlag, 1998, pp. 15–32.

[8] L. T. BIEGLER, J. NOCEDAL, AND C. SCHMID, *A reduced Hessian method for large-scale constrained optimization*, SIAM Journal on Optimization, 5 (1995), pp. 314–347.

[9] G. BIROS AND O. GHATTAS, *Parallel Newton-Krylov algorithms for PDE-constrained optimization*, in Proceedings of SC99, The SCxy Conference series, Portland, Oregon, November 1999, ACM/IEEE.

[10] ———, *Parallel preconditioners for KKT systems arising in optimal control of viscous incompressible flows*, in Parallel Computational Fluid Dynamics 1999, D. E. Keyes, A. Ecer, J. Periaux, and N. Satofuka, eds., North-Holland, 1999.

[11] R. BYRD AND J. NOCEDAL, *An analysis of reduced Hessian methods for constrained optimization*, Mathematical Programming, 49 (1991), pp. 285–323.

[12] X.-C. CHAI AND O. B. WIDLUND, *Domain decomposition algorithms for indefinite elliptic problems*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 243–258.

[13] E. CHOW AND Y. SAAD, *Approximate inverse techniques for block-partitioned matrices*, SIAM Journal on Scientific Computing, 18 (1997), pp. 1657–1675.

[14] J. E. DENNIS AND R. M. LEWIS, *A comparison of nonlinear programming approaches to a an elliptic inverse problem and a new domain decomposition approach*, Tech. Rep. TR-9433, Department of Computational and Applied Mathematics, Rice University, 1994.

[15] C. FARHAT, R. TEZAUR, AND R. DJELLOULI, *On the solution of three dimensional inverse obstacle scattering problems by a regularized Newton method*, Tech. Rep. CU-CAS-01-11, Center for Aerospace Structures, University of Colorado, Boulder, December 2001.

[16] R. FLETCHER, *Practical Methods of Optimization*, John Wiley and Sons, second ed., 1987.

[17] R. W. FREUND AND N. M. NACHTIGAL, *An implementation of the QMR method based on coupled two-term recurrences*, SIAM Journal of Scientific Computing, 15 (1994), pp. 313–337.

[18] O. GHATTAS AND J.-H. BARK, *Optimal control of two- and three-dimensional incompressible Navier-Stokes flows*, Journal of Computational Physics, 136 (1997), pp. 231–244.
[19] O. GHATTAS AND C. E. OROZCO, *A parallel reduced Hessian SQP method for shape optimization*, in Multidisciplinary Design Optimization: State-of-the-Art, N. Alexandrov and M. Hussaini, eds., SIAM, 1997, pp. 133–152.
[20] G. H. GOLUB AND C. H. V. LOAN, *Matrix Computations*, Johns Hopkins, third ed., 1996.
[21] A. GREENBAUM, *Iterative methods for solving linear systems*, SIAM, 1997.
[22] M. D. GUNZBURGER, *Finite Element for Viscous Incompressible Flows*, Academic Press, 1989.
[23] E. HABER AND U. C. ASCHER, *Preconditioned all-at-once methods for large, sparse parameter estimation problems*, in 2001 International Conference On Preconditioning Techniques For Large Sparse Matrix Problems In Industrial Applications, Granlibakken Conference Center, Tahoe City, CA, April 2001.
[24] D. E. KEYES, *How scalable is domain decomposition in practice?*, in Proceedings of the 11th International conference on Domain Decomposition Methods, C.-H.Lai, M. Cross, and O. Widlund, eds., 1998.
[25] D. E. KEYES AND W. D. GROPP, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. S166–S202.
[26] K. KUNISCH AND E. W. SACHS, *Reduced SQP methods for parameter identification problems*, SIAM Journal on Numerical Analysis, 29 (1992), pp. 1793–1820.
[27] F.-S. KUPFER AND E. W. SACHS, *Numerical solution of a nonlinear parabolic control problem by a reduced SQP method*, Computational Optimization and Applications, 1 (1992), pp. 113–135.
[28] I. MALČEVIĆ, *Large-scale unstructured mesh shape optimization on parallel computers*, Master's thesis, Carnegie Mellon University, 1997.
[29] J. L. MORALES AND J. NOCEDAL, *Automatic preconditioning by limited memory quasi-Newton updating*, SIAM Journal on Optimization, 10 (2000), pp. 1079–1096.
[30] S. G. NASH AND A. SOFER, *Preconditioning reduced matrices*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 47–68.
[31] J. NOCEDAL AND S. J. WRIGHT, *Numerical Optimization*, Springer, 1999.
[32] C. E. OROZCO AND O. GHATTAS, *Massively parallel aerodynamic shape optimization*, Computing Systems in Engineering, 1–4 (1992), pp. 311–320.
[33] ———, *A reduced SAND method for optimal design of nonlinear structures*, International Journal for Numerical Methods in Engineering, 40 (1997), pp. 2759–2774.
[34] J. REUTHER, J. ALONSO, M. J. RIMLINGER, AND A. J. JAMESON, *Aerodynamic shape optimization of supersonic aircraft configurations via an adjoint formulation on parallel computers*, Computers and Fluids, 29 (1999), pp. 675–700.
[35] U. RINGERTZ, *Optimal design of nonlinear shell structures*, Tech. Rep. FFA TN 91-18, The Aeronautical Research Institute of Sweden, 1991.
[36] ———, *An algorithm for optimization of nonlinear shell structures*, International Journal for Numerical Methods in Engineering, 38 (1995), pp. 299–314.
[37] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, 1996.
[38] A. SALINGER, R. PAWLOWSKI, J. SHADID, B. VAN BLOEMEN WAANDERS, R. BARLETT, G. ITLE, AND L. BIEGLER, *rSQP optimization of large-scale reacting flow applications with MPSalsa*, in Proceedings of First Sandia Workshop on PDE-constrained Optimization, Santa-Fe, April 2001.
[39] V. H. SCHULZ AND H. G. BOCK, *Partially reduced SQP methods for large-scale nonlinear optimization problems*, in Proceedings of the Second World Congress of Nonlinear Analysis, Elsevier, 1997.