LLNL IM Release number: LLNL-JRNL-739001

A memory-distributed quasi-Newton solver for nonlinear programming problems with a small number of general constraints

Cosmin G. Petra[a,*]

[a]*Center for Applied Scientific Computing*
Lawrence Livermore National Laboratory
7000 East Avenue, Livermore, CA 94550, USA.

Abstract

We address the problem of parallelizing state-of-the-art nonlinear programming optimization algorithms. In particular, we focus on parallelizing quasi-Newton interior-point methods that use limited-memory secant Hessian approximations. Such interior-point methods are known to have better convergence properties and to be more effective on large-scale problems than gradient-based and derivative-free optimization algorithms. We target nonlinear and potentially nonconvex optimization problems with an arbitrary number of bound constraints and a small number of general equality and inequality constraints on the optimization variables. These problems occur for example in the form of optimal control, optimal design, and inverse problems governed by ordinary or partial differential equations, whenever they are expressed in a "reduced-space" optimization approach. We introduce and analyze the time and space complexity of a decomposition method for solving the quasi-Newton linear systems that leverages the fact that the quasi-Newton Hessian matrix has a small number of dense blocks that border a low-rank update of a diagonal matrix. This enables an efficient parallelization on memory-distributed computers of the iterations of the optimization algorithm, a state-of-the-art filter line-search interior-point algorithm by Wächter et. al. We illustrate the efficiency of the proposed method by solving structural topology optimization problems on up to $4,608$ cores on a parallel machine.

Keywords: parallel optimization, parallel interior-point, quasi-Newton.

[*]E-mail address: petra1@llnl.gov.

1. Introduction

The complexity and dimensionality of optimization problems occurring in various engineering areas, *e.g.* optimal control, optimal design, and inverse problems, operations research, data analysis, and climate research have undoubtedly increased enormously in the last decades. It is widely accepted that high-performance computing and parallel numerical solvers are needed to solve such complex and large-scale optimization problems. The present work joins the efforts in developing parallel optimization solvers and presents a parallelization methodology for nonlinear programming (NLP) algorithms.

The community of mathematical programming has a long history of developing parallel optimization algorithms. We mention structure-exploiting methods for stochastic optimization, such as parallel interior-point methods (IPMs) [18, 14, 32], parallel simplex methods [27], DantzigWolfe or Benders decomposition [26, 31, 10, 6], progressive hedging [34], many of which have resulted in massively parallel optimization solvers capable of achieving good parallel efficiencies on high-performance computing architectures [18, 28]. Central to these methods is to leverage the underlying structure of the problem, which is given by the presence of multiple optimization scenarios that are linked through only a subset of so-called first-stage optimization variables, to decompose the linear algebra computations inside the optimization iterations. The methodology presented in this paper is similar in this respect, however it addresses a different computational setup. The evaluations of *the functions and their gradients are assumed to be performed efficiently on parallel machines*, possibly by using black-box simulators. This is almost always the result of a data-type of parallelism that the simulators exploit. In order to exploit this parallelism opportunity, the present work uses a limited-memory secant quasi-Newton interior-point method and proposes a data-based parallelization approach for the linear algebra computations inside the optimization iterations.

Our work is primarily motivated by structural topology optimization problems that routinely occur in the optimal design of new materials and/or structures. This class of problems seeks to maximize the global stiffness of a structure while enforcing a maximum weight constraint; mathematically, they take the form of optimization problems constrained by partial differential equations (PDEs). The aforementioned simulator for evaluating the objective and constraints and their derivatives is in this case a PDE solver for the so-called forward or state linear elasticity problem and associated first-order adjoint sensitivity problem. To give an idea about the extreme size of topology optimization problems, which are described in Section 2.1, we mention that these problems have one optimization variable per discretization (finite) element; thus, optimization problems with billions of variables occur naturally for complex structures that require a large number of elements in the finite element analysis. This has been the case for example for wing plane structural design [1].

However, since our methodology is developed under the more general framework of mathematical programming, it is applicable to other PDE-constrained optimization problems, such as optimal control, optimal design, and inverse problems, as well as to general nonlinear optimization problems. In the context of PDE-constrained optimization, our optimization approach falls under umbrella of "reduced-space" methods, see for example [22], which essentially means that the optimization is performed only in the optimization variables and the system of equations governing the optimization is eliminated from the problem formulation (hence the PDE solver for the forward and adjoint problems can be used as a black-box in the optimization). The community of PDE-constrained optimization worked extensively to develop parallel algorithms, by using a variety of optimization approaches: trust-region methods [25, 21], augmented Lagrangian methods in TAO [16, 29], Newton-Krylov [9, 7, 8], and others. A detailed discussion of PDE-constrained optimization algorithms can be found, for example, in [4, 5, 15, 23]. The method of moving asymptotes (MMA) [35, 41, 36] is undoubtedly a very popular solution scheme for topology optimization. Essentially a sequential convex approximation method that uses first-order derivatives, MMA was consequently parallelized and used successfully to design structures with more than one billion degrees of freedom using HPC [1]. A recent benchmarking study [33] showed that nonlinear programming solvers Ipopt [39, 38] and SNOPT [17], including the Ipopt's quasi-Newton interior-point method used in this paper, perform better with respect to both computational cost and solution quality than MMA methods.

There is considerable evidence in mathematical programming and PDE-constrained optimization [30, 9, 18, 32, 7, 8, 28] supporting that that second-order, Newton-like algorithms have improved theoretical properties (*e.g.*, local quadratic convergence rates) and practical performance (*e.g.*, number of iterations) over methods that make only use of gradients or do not employ derivative/sensitivity information. Even though second-order derivatives or Hessians of the objective and constraints may exist mathematically, they may not be available computationally in some applications; for example, because of the high human cost required to develop second-order sensitivities within existing simulation engines. For this cases, quasi-Newton algorithms are a pragmatic choice since they can achieve local superlinear convergence and practical performance that is better than gradient-based or derivative-free methods, without requiring the evaluation or application of the Hessian [30]. Quasi-Newton methods with Hessian approximations based on limited-memory secant updates [13] have been emerged as a computationally feasible and robust approach. The present work differentiates from existing quasi-Newton methods for PDE-constrained optimization, *e.g.*, [20, 24], by solving problems with *general constraints* (bounds, equality, and inequality) on the optimization variables *in parallel*.

The filter linesearch quasi-Newton IPM used in this work follows closely the implementation present in the Ipopt solver [40]. This choice was motivated by the emergence of Ipopt as a reliable state-of-the-art algorithm in the mathematical programming community [40]. However the parallel linear algebra techniques proposed by this paper can be potentially used with other optimization methods as well, *e.g.* almost any flavor of IPMs, sequential quadratic programming methods, augmented Lagrangian methods, and possibly with trust-region methods. The contribution of this work consists of providing *a general paralleliza-*

tion methodology for the linear algebra of quasi-Newton nonlinear optimization algorithms. We believe that this is a key step in facilitating the use of state-of-the-art algorithms developed by the mathematical programming community for massively parallel optimization in various application areas.

2. The optimization problem and underlying data parallelism

In this work we consider general nonlinear, possibly nonconvex optimization problems of the form

$$\min_{x\in\mathbb{R}^n} \quad f(x) \tag{1}$$

$$\text{s.t.} \quad c(x) = c_E, \tag{2}$$

$$d_l \le d(x) \le d_u, \tag{3}$$

$$x_l \le x \le x_u. \tag{4}$$

Here $f : \mathbb{R}^n \to \mathbb{R}$, $c : \mathbb{R}^n \to \mathbb{R}^{m_E}$, and $d : \mathbb{R}^n \to \mathbb{R}^{m_I}$. The bounds appearing in the inequality constraints (3) are assumed to be $d^l \in \mathbb{R}^{m_I} \cup \{-\infty\}$, $d^u \in \mathbb{R}^{m_I} \cup \{+\infty\}$, $d^l_i < d^u_i$, and at least of one of d^l_i and d^u_i are finite for each $i \in \{1, \dots, m_I\}$. The bounds in (4) are such that $x^l \in \mathbb{R}^n \cup \{-\infty\}$, $x^u \in \mathbb{R}^n \cup \{+\infty\}$, and $x^l_i < x^u_i$, $i \in \{1, \dots, n\}$. For the rest of the paper m will denote $m_E + m_I$, *i.e.*, the total number of constraints excepting the simple bounds constraints (4).

The computational method introduced in this paper addresses problems of the form (1)-(4) with large n and is tailored for a relatively small number of general constraints m. Note that one can specify simple bounds on all the optimization variables and doing so will not affect the efficiency of the parallelization. The interior-point method with quasi-Newton approximation of the Hessian used in this work requires first-order derivative to be specified for problems of the form (1)-(4). This is addition to objective and constraints functions evaluations. Computationally, in order to solve problems of the form (1)-(4) one needs to specify the following "input data:"

(D1) the objective and constraint functions $f(x)$, $c(x)$, $d(x)$;

(D2) the functions evaluating the first-order derivatives of the above: gradient $\nabla f(x) \in \mathbb{R}^n$ and Jacobians $J_c(x) \in \mathbb{R}^{m_E \times n}$ and $J_d(x) \in \mathbb{R}^{m_I \times n}$; and

(D3) the vectors specifying the simple bounds x_l and x_u, the inequality bounds d_l and d_u, and the right-hand size of the equality constraints c_E.

The salient idea of the data parallelism employed in this paper is to distribute the data structures that have storage requirements dependent on n, *i.e.*, x, x_l, x_u, $\nabla f(x)$, $J_c(x)$, $J_d(x)$ across MPI ranks. The remaining of the problem's data, which has leading space complexity depending on m is replicated on each rank. The replicated data include the function return scalar values $f(x)$, $c(x)$, and $d(x)$, the inequalities bounds d_l and d_u, and the right-hand c_E. As we will later show this data decomposition will translate in efficient parallelization of the computations required by the optimization algorithm. We remark that there are no matrices that have both the number of columns and rows depending on n. As illustrated in Figure 1, the Jacobians $J_c(x)$ and $J_d(x)$ are distributed column-wise. The function evaluations $f(x)$, $c(x)$, and $d(x)$ are assumed to be done in parallel and each rank has the return value. The evaluations of the gradient and Jacobians are assumed to be also performed parallel, however, each rank updates only the local of the vector or matrix return value, as illustrated in Figure 1.

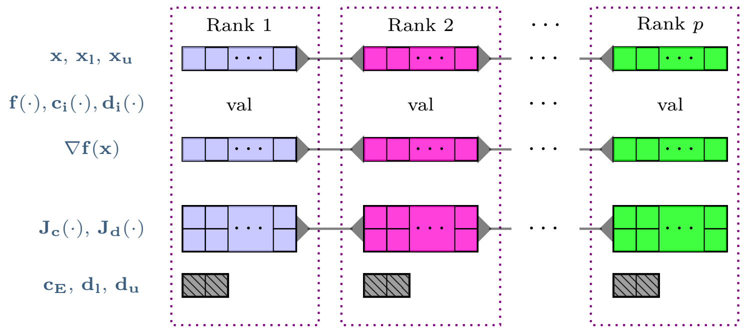
Rank 1
Rank 2
Rank p
$\mathbf{x}$, $\mathbf{x_l}$, $\mathbf{x_u}$
$\mathbf{f}(\cdot)$, $\mathbf{c_i}(\cdot)$, $\mathbf{d_i}(\cdot)$
val
$\nabla\mathbf{f}(\mathbf{x})$
$\mathbf{J_c}(\cdot)$, $\mathbf{J_d}(\cdot)$
$\mathbf{c_E}$, $\mathbf{d_l}$, $\mathbf{d_u}$

Figure 1: Depiction of the distribution of the data of the optimization problem (1)-(4) across MPI ranks. The vectors and matrices with storage dependent on the number of optimization variables are distributed. Other data, *i.e.*, scalar function values or vectors of small size (shown in dashed dark grey boxes), are replicated on each rank.

2.1. Motivating example: structural topology optimization

We succinctly present the minimum compliance topology optimization problems [33], which seek to maximize the global stiffness of a structure while enforcing a maximum structure weight constraint. The objective of the problem is to minimize the compliance, defined as integral of the product of the displacement field u and the applied external loads l. The displacement is obtained by solving the linear elastic equilibrium partial differential equations. After the discretization using a finite element method (FEM), for example, these equations take the form of $K(\rho)u = l$, where K is the finite element stiffness matrix, which is positive definite, and ρ are the optimization or design variables representing the relative density of the material in each finite element. These design variables are constrained within $[\rho_{\min}, 1]$, where $\rho_{\min}$ is a small positive lower bound that ensure the stiffness matrix K is invertible. Mathematically, minimum compliance problems can be compactly expressed as

$$\min_{\rho,u} \quad l^T u \tag{5}$$

$$\text{s.t.} \quad K(\rho)u = l, \tag{6}$$

$$\rho_{\min} \le \rho \le 1, \tag{7}$$

$$a^T\rho \le V_{\max}. \tag{8}$$

The last inequality constraints is a maximum (relative) volume constraint that imposes a maximum structure weight constraint. The vector a in this constraint represents the relative volume of the elements.

In this work we use a so-called reduced-space modeling and solution approach to solve (5)-(8). Since K is invertible, the (state) variable u can be expressed as an explicit function of the design variables, $u(\rho) = K^{-1}(\rho)l$; as a result, the above formulation can be expressed as a nonlinear optimization problem in ρ only:

$$\min_{\rho} \quad l^T K^{-1}(\rho) l \tag{9}$$

$$\rho_{\min} \leq \rho \leq 1, \tag{10}$$

$$a^T \rho \leq V_{\max}. \tag{11}$$

It should be apparent that this formulation fits under the umbrella of nonlinear programming problems (1)-(4).

A large body of work was done in the community of topology optimization to ensure good solid-void designs, *e.g.*, by using an SIMP material power law interpolation scheme [3], and to ensure the mathematical well-possessedness and mesh independent solutions of (9)-(11), *e.g.* by employing density or Helmholtz filters [12, 11]. We mention that these improvements do not alter the mathematical formulation (9)-(11) as a nonlinear programming problem [33].

A reduced-space PDE-constrained optimization solution approach for minimum compliance problems (9)-(11) require the evaluation of the the objective for a given vector of optimization variables ρ. This is performed by solving the linear elasticity problem $u(\rho) = K^{-1}(\rho)l$, for example by using parallel implementations of the finite element method and multigrid linear solvers [1] and computing the inner product $l^T u(\rho)$. The gradient is computed by using adjoint sensitivity analysis (for example see [37]), which approximately has the cost of one objective function evaluation. Additional operations are performed for both the objective and its gradient to apply the material interpolation scheme and the filters. The constraint (11) is computed similarly. We note that this constraint may become nonlinear when filters are used.

3. The interior-point algorithm

The general NLP (1)-(4) is transformed internally by the optimization solver to an equivalent form that is more amenable to the use of interior-point methods.

This form is using slacks and can be mathematically expressed as

$$\begin{aligned}
\min_{x,\, d,\, s_l^x, s_u^x,\, s_l^d, s_u^d} \quad & f(x) && && (12)\\
\text{s.t.} \quad & c(x) = c_E, && [y_c] && (13)\\
& d(x) - d = 0, && [y_d] && (14)\\
& d - s_l^d = d_l, && [v_l] && (15)\\
& d + s_u^d = d_u, && [v_u] && (16)\\
& x - s_l^x = x_l, && [z_l] && (17)\\
& x + s_u^x = x_u, && [z_u] && (18)\\
& s_l^x, s_u^x,\, s_l^d, s_u^d \geq 0. && && (19)
\end{aligned}$$

The symbols in brackets are Lagrange multipliers or the dual variables. They are required because the interior-point method we use is a primal-dual method. The inequality constraints that have infinite right-hand sides should be understood as not being part of the problem and their multipliers are zero.

Specific to interior-point methods is the use of log-barrier functions for the inequality constraints. The log barrier function $\mu \ln(\cdot)$ is applied to each (entry of the) signed slacks in (19)). IPMs solve a sequence of log-barrier problems, each corresponding to a log-barrier parameter $\mu_k > 0$; to achieve optimality, μ_ks are decreased such that $\mu_k \to 0$. The log-barrier subproblem is

$$\begin{aligned}
\min_{x,\, d,\, s_l^x, s_u^x,\, s_l^d, s_u^d} \quad & f(x) - \mu \ln(s_l^x) - \mu \ln(s_u^x) - \mu \ln(s_l^d) - \mu \ln(s_u^d) && && (20)\\
\text{s.t.} \quad & c(x) = c_E, && [y_c] && (21)\\
& d(x) - d = 0, && [y_d] && (22)\\
& d - s_l^d = d_l, && [v_l] && (23)\\
& d + s_u^d = d_u, && [v_u] && (24)\\
& x - s_l^x = x_l, && [z_l] && (25)\\
& x + s_u^x = x_u. && [z_u] && (26)
\end{aligned}$$

The optimal solution of the log-barrier problem is found by approximately solving the first-order optimality conditions. To derive these, we first introduce the Lagrangian function

$$\begin{aligned}
L_\mu(x,\, d,\, & s_l^x, s_u^x, s_l^d, s_u^d; y_c, y_d, v_l, v_u, z_l, z_u) = \\
& = f(x) - \mu \ln(s_l^x) - \mu \ln(s_u^x) - \mu \ln(s_l^d) - \mu \ln(s_u^d) \\
& + y_c^T(c(x) - c_E) + y_d^T(d(x) - d) + \\
& + z_l^T(-x + s_l^x + x_l) + z_u^T(x + s_u^x - x_u) \\
& + v_l^T(-d + s_l^d + d_l) + v_u^T(d + s_u^d - d_u),
\end{aligned}$$

which allows the optimality conditions for the log-barrier problem (20)-(26) to be written as

$$\nabla L_\mu(x,\, d,\, s_l^x, s_u^x,\, s_l^d, s_u^d; y_c, y_d, v_l, v_u, z_l, z_u) = 0.$$

Here the gradient is taken with respect to all arguments of L_μ. The stationarity condition above can be written as

$$\begin{aligned}
\nabla f(x) + J_c^T(x) y_c + J_d^T(x) y_d - z_l + z_u &= 0, \\
-y_d - v_l + v_u &= 0, \\
c(x) &= c_E, \\
d(x) - d &= 0, \\
-x + s_l^x + x_l = 0, \quad x + s_u^x - x_u &= 0, \\
-d + s_l^d + d_l = 0, \quad d + s_u^d - d_u &= 0, \\
s_l^x z_l = \mu e, \quad s_u^x z_u = \mu e, \quad s_l^d v_l = \mu e, \quad s_u^d v_u &= \mu e.
\end{aligned} \tag{27}$$

At each iteration, the linesearch IPM computes a search direction by solving a linearization of the above system of equations and then updates the iteration 165 using this direction. The barrier parameter μ is decreased whenever the norm of the residual of (27), which we refer to as "log-barrier error", is small. The IPM reaches the optimality when μ is small, and the norm of the residual of (27) for this value of μ, which we refer to as the "NLP error", is also close to zero.

The search direction is computed by performing a (damped) Newton iteration for the above nonlinear systems of equations using the incumbent optimization iterate as starting point. The Newton direction $[\Delta x,\ \Delta d,\ \Delta s_l^x,\ \Delta s_u^x,\ \Delta s_l^d,\ \Delta s_u^d,\ \Delta y_c,\ \Delta y_d,\ \Delta v_l,\ \Delta v_u,\ \Delta z_l,\ \Delta z_u]$ is obtained by solving the linear system

$$\begin{aligned}
B\Delta x + J_c^T(x)\Delta y_c + J_d^T(x)\Delta y_d - \Delta z_l - \Delta z_u = r_x &:= -\nabla f(x) - J_c^T(x) y_c, \\
&\quad - J_d^T(x) y_d + z_l - z_u, \\
-\Delta y_d - \Delta v_l + \Delta v_u = r_d &:= y_d + v_l - v_u, \\
J_c^T(x)\Delta x = r_{y_c} &:= -c(x) + c_E, \\
J_d^T(x)\Delta x - \Delta d = r_{y_d} &:= d - d(x), \\
-\Delta x + \Delta s_l^x + x_l = r_l^x &:= x - s_l^x - x_l, \\
\Delta x + \Delta s_u^x - x_u = r_u^x &:= x - s_u^x + x_u, \\
-\Delta d + \Delta s_l^d + d_l = r_l^d &:= d - s_l^d - d_l, \\
\Delta d + \Delta s_u^d - d_u = r_u^d &:= d - s_u^d + d_u, \\
Z_l \Delta s_l^x + S_l^x \Delta z_l = r_l^{sz} &:= \mu e - s_l^x z_l, \\
Z_u \Delta s_u^x + S_u^x \Delta z_u = r_u^{sz} &:= \mu e - s_u^x z_u, \\
V_l \Delta s_l^d + S_l^d \Delta v_l = r_l^{sv} &:= \mu e - s_l^d v_l, \\
V_u \Delta s_u^d + S_u^d \Delta v_u = r_u^{sv} &:= \mu e - s_u^d v_u.
\end{aligned} \tag{28}$$

Here the uppercase symbols denote diagonal matrices having the diagonal given 170 by lowercases symbols, *e.g.*, $X_l = \text{diag}(x_l)$. The matrix B is an approximation of the Hessian of the Lagrangian namely

$$B \approx \nabla_{xx}^2 L(x, d,\ s_l^x, s_u^x,\ s_l^d, s_u^d; y_c, y_d, v_l, v_u, z_l, z_u).$$

This is dictated by the fact that the problems targeted in this work do not have the Hessian available (not even in matrix-times-vector form). Hence our numerical optimization approach falls under the umbrella of quasi-Newton linesearch IPMs. Constructing B using secant approximations, as well as solving the above linear system are discussed in detail in Section 4.1. We list a sketch of the "outer" optimization loop in Algorithm 1. For the remaining of this section, we use the following notations: $\mathbf{x} = \left[x, d, s_l^x, s_u^x, s_l^d, s_u^d\right]$, $\mathbf{y} = [y_c, y_d]$, and $\mathbf{z} = [v_l, v_u, z_l, z_u]$.

Algorithm 1 Pseudocode of the linesearch IPM used in this paper

```
Input: User-supplied initial point $x_0$, algorithm parameters $\mu_0$ and $\sigma_0$, and
    stopping tolerance $\epsilon_{tol}$
 1: Let $\mu = \mu_0$
 2: Adjust $x_0$ for feasibility by projecting in the interior of the bounds box
 3: Set the quasi-Newton approximation to $B_0 = \sigma_0 I$
 4: for $k = 0, 1, \dots$ do
 5:     If NLP error less than $\epsilon_{tol}$ then break and return optimal solution
 6:     If log-barrier error less then $10 \cdot \epsilon_{tol}$ reduce $\mu$ and continue
 7:     Compute search direction $[\Delta\mathbf{x}, \Delta\mathbf{y}, \Delta\mathbf{z}]$ from (28) at incumbent iteration
        $[\mathbf{x}_k, \mathbf{y}_k, \mathbf{z}_k]$
 8:     Backtracking linesearch to find primal and dual steplengths $\alpha_p$ and $\alpha_d$
 9:     Update the iteration: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_p\Delta\mathbf{x}$, $\mathbf{y}_{k+1} = \mathbf{y}_k + \alpha_p\Delta\mathbf{y}$, and
        $\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha_d\Delta\mathbf{z}$
10:     Update quasi-Newton Hessian approximation $B_{k+1}$.
11: end for
```

Our implementation of the interior-point method follows closely the Ipopt quasi-Newton IPM implementation and we refer the reader to [40] for the details of Algorithm 1.

4. Parallelization of the interior-point linear systems

The majority of the computational cost of Algorithm 1 is given by the solution of the search direction linear system (28) in line 7 of Algorithm 1. Before presenting the parallelization scheme of these linear systems we introduce the quasi-Newton Hessian approximation formula for B and a variant of it, in the form of a compact inverse formula, that we derived for quasi-Newton IPMs.

The Ipopt solver, which serves as the benchmark solver in this paper, solves the IPM linear systems in serial. Its solution technique for solving the quasi-Newton linear system (28) uses a secant low-rank Hessian approximation similar to the one described in the next section; however, it relies on sparse linear solves with multiple right-hand sides in order to deal efficiently with the low-rank structure of the Hessian approximation (see [30, Chapter 19]). The approach presented in this section is different and is based on dense linear algebra. We build an explicit compact inverse formula for the low-rank Hessian approximation and apply it (mostly as dense matrix-matrix multiplications) to compute a "reduced" linear system (*e.g.*, (41)) that is only of size m and, therefore, cheap to solve. We mention that the underlying data parallelism of the optimization problem transfers to the data structures and efficient computations decomposition, as we show later in this section.

4.1. The secant quasi-Newton approximation of the Hessian

The BFGS formula is a popular and effective approach for Hessian approximations. The salient idea of the BFGS method is to use the change in the gradients,

$$w_k := \nabla_{\mathbf{x}} L(\mathbf{x}_{k+1}, \mathbf{y}_k, \mathbf{z}_k) - \nabla_{\mathbf{x}} L(\mathbf{x}_k, \mathbf{y}_k, \mathbf{z}_k) \tag{29}$$

and in the optimization variables,

$$p_k := \mathbf{x}_{k+1} - \mathbf{x}_k$$

during the previous iteration to build an approximation of the Hessian. Namely, a secant equation, $B_{k+1} p_k = w_k$ is enforced upon B_{k+1}. Among the many symmetric matrices that satisfy the secant equation, the one closest to the previous approximation B_k (in the sense of a weighted Frobenius norm, see [19, 30]) is chosen as a way to maintain Hessian information gathered during the previous optimization iterations. The matrix is given recursively by the formula

$$B_{k+1} = B_k - \frac{B_k p_k p_k^T B_k}{p_k^T B_k p_k} + \frac{w_k w_k^T}{w_k^T p_k}. \tag{30}$$

The initial approximation B_0 is a scaled identity matrix. We remark that the above formula is a rank-two update.

Storing a large number of pairs (p_k, w_k) quickly becomes a storage bottleneck. Limited-memory variants of the secant BFGS formula (30) are used in practice. The limited-memory secant BFGS approximations use only the most recent l pairs (p_i, w_i), $i = \{k-l, \ldots, k-1\}$; after computing a new iterate, the oldest pair is deleted and replaced by the newest one. It is generally accepted that $l \in \{6, 12\}$ offers a good trade-off between the cost per iteration and number of iterations [13].

An explicit matrix representation for the limited-memory secant BFGS approximation can be derived [13] in the form of

$$B_k = B_0 - [B_0 P_k \ W_k] \begin{bmatrix} P_k^T B_0 P_k & L_k \\ L_k^T & -D_k \end{bmatrix} \begin{bmatrix} P_k^T B_0 \\ W_k^T \end{bmatrix}, \tag{31}$$

where $P_k = [p_{k-l}, \ldots, p_{k-1}]$, $W_k = [w_{k-l}, \ldots, w_{k-1}]$, $L_k \in \mathbb{R}^{l \times l}$ is given by

$$[L_k]_{ij} = \begin{cases} p_{i-1}^T w_{j-1}, & \text{if } i > j \\ 0, & \text{otherwise,} \end{cases} \tag{32}$$

and $D_k = \operatorname{diag}\left[p_{k-l}^T w_{k-l}, \ldots, p_{k-1}^T w_{k-1}\right]$. Such compact representation is especially useful in our interior-point method-based optimization approach since it is a low-rank update of the diagonal B_0, and can be leveraged to solve the interior-point linear system (28) efficiently in parallel, as we show in Section 4.3.

4.2. Revisiting the secant quasi-Newton approximation to include the log-barrier terms

As we later show in (4.3), the BFGS compact representation (31) needs to be revisited to allow the incorporation of the log-barrier terms. In particular, multiple linear systems sharing the same matrix $B_k + D_x$, where D_x is a diagonal matrix with positive diagonal entries, needs to be solved at each optimization iteration by the elimination scheme of Section (4.3). For this reason we derive an explicit formula for the inverse of $B_k + D_x$ based on the compact representation (31) of B_k and Sherman-Morison-Woodbury formula. For compactness, we write the compact representation (31) as $B_k = B_0 + UVU^T$. Our explicit compact inverse representation is derived as follows:

$$\begin{aligned}
(B_k + D_x)^{-1} &= \left(B_0 + D_x - UVU^T\right)^{-1} \\
&= (B_0 + D_x)^{-1} \\
&\quad - (B_0 + D_x)^{-1} U \left(-V + U^T(D_x + B_0)^{-1}U\right)^{-1} U^T(D_x + B_0)^{-1} \\
&= (B_0 + D_x)^{-1} \\
&\quad - (B_0 + D_x)^{-1} U \cdot \mathcal{L}^{-T}\mathcal{D}^{-1}\mathcal{L}^{-1} \cdot U^T(D_x + B_0)^{-1}, \qquad (33)
\end{aligned}$$

where $\mathcal{L}$ and $\mathcal{D}$ are the matrix factors obtained from a LDL^T factorization of the matrix $\mathcal{A} = -V + U^T(D_x + B_0)^{-1}U \in \mathbb{R}^{2l\times 2l}$. One can verify that $\mathcal{A}$ is exactly

$$\begin{bmatrix} P_k^T(B_0(D_x + B_0)^{-1}B_0 - B_0)P_k & P_k^T B_0(D_x + B_0)^{-1}W_k - L_k \\ W_k^T(D_x + B_0)^{-1}B_0P_k - L_k^T & W_k^T(D_x + B_0)^{-1}W_k + D_k \end{bmatrix}.$$

We also note that $(D_x + B_0)$ is positive definite diagonal matrix.

In what follows we discuss time and space complexities of

(I.) computing the inverse compact representation (33) and

(II.) multiplying $(B_k + D_x)^{-1}$ (as a way to solve with $B_k + D_x$) with one vector.

We assume that $l = O(1)$ and n is much larger than l, l^2, and l^3.

The symmetric LDL^T direct factorization of the dense matrix $\mathcal{A}$ in (33) contributes with a time cost of $(2l)^3/3$ and a $O(l^2)$ space cost to (I.). The computation of $\mathcal{A}$ exploits the symmetry and only the $(1,1)$, $(2,1)$, and $(2,2)$ blocks are computed, at $(l^2/2 + l + 3)n + (l^2 + l + 1)n + (l^2/2 + l)n = (2l^2 + 3l + 4)n$ leading time complexity. The remaining cost is for computing the compact representation (31); at each optimization iteration this requires only the computation of one element of D_k and a column of L_k and therefore it has leading cost of only $O(ln)$. In conclusion, the *leading time complexity for (I.) is* $O(l^2n)$.

Furthermore, (II.) is performed by multiplying in left-to-right order matrices (33) with one vector of size n. During this step, the multiplications with the inverse factors are performed as triangular and diagonal solves with $\mathcal{L}$, $\mathcal{D}$, and

$\mathcal{L}^{\mathcal{T}}$ for a right-hand side of size l, thus, the cost is $O(l^2)$; *the dominant cost in (II.) is* $O(ln)$ and is given by the multiplications with the diagonals $B_0 + D_x$ and B_0 and by the mat-vec multiplications involving P_k, W_k, P_k^T, and W_k^T.

In addition, the leading space complexity of the compact inverse representation is $O(2ln)$, given by the storage requirements of P_k and W_k. The rest of the data from (33) is at most linear in n.

4.3. Solving the linear systems of the quasi-Newton IPM

For compactness we drop the subscripts k, but we remind the reader that the solution methodology of this section is used at each optimization iteration. Since many of block matrices in (28) are diagonals or identities, a series of computationally efficient variable eliminations can be also performed. First, observe that one can write

$$\Delta s_l^x = r_l^x + \Delta x,\ \Delta s_u^x = r_u^x - \Delta x,\ \Delta s_l^d = r_l^d + \Delta x,\ \text{and}\ \Delta s_u^d = r_u^d - \Delta x. \quad (34)$$

Furthermore, from the last four equations of (28) and the equations above, one can also write

$$\Delta z_l = -(S_x^l)^{-1}\left(Z_l \Delta s_l^x + r_l^{sz}\right) = -(S_x^l)^{-1}\Delta x + (S_x^l)^{-1}(r_l^{sz} - Z_l r_l^x), \quad (35)$$
$$\Delta z_u = -(S_x^u)^{-1}\left(Z_u \Delta s_u^x + r_u^{sz}\right) = -(S_x^u)^{-1}\Delta x + (S_x^u)^{-1}(r_u^{sz} - Z_u r_u^x), \quad (36)$$
$$\Delta v_l = -(S_d^l)^{-1}\left(V_l \Delta s_l^d + r_l^{sv}\right) = -(S_d^l)^{-1}\Delta x + (S_d^l)^{-1}(r_l^{sv} - V_l r_l^x), \quad (37)$$
$$\Delta v_u = -(S_d^u)^{-1}\left(V_u \Delta s_u^d + r_u^{sv}\right) = -(S_d^u)^{-1}\Delta x + (S_d^u)^{-1}(r_u^{sv} - V_u r_u^x). \quad (38)$$

By substituting the expressions (34)-(38) into (28) and using the notation $D_x := (S_l^x)^{-1}Z_l + (S_u^x)^{-1}Z_u$ and $D_d := (S_l^d)^{-1}V_l + (S_u^d)^{-1}V_u$, one can obtain

$$\begin{aligned}
(B + D_x)\Delta x + J_c^T \Delta y_c + J_d^T \Delta y_d = \widetilde{r}_x &:= r_x + (S_l^x)^{-1}\left(r_l^{sz} - Z_l r_l^x\right) \\
&\quad - (S_u^x)^{-1}\left(r_u^{sz} - Z_u r_u^x\right) \\
D_d \Delta d - \Delta y_d = \widetilde{r}_d &:= r_d + (S_l^d)^{-1}\left(r_l^{sv} - V_l r_l^d\right) \\
&\quad - (S_u^x)^{-1}\left(r_u^{sv} - V_u r_u^d\right) \\
J_c \Delta x &= r_{y_c} \\
J_d \Delta x - \Delta d &= r_{y_d}.
\end{aligned}$$

This above linear system can be further reduced since D_d is a diagonal matrix and $\Delta d = D_d^{-1}(\Delta y_d + \widetilde{r}_d)$; then the rest of directions can be computed from

$$\begin{bmatrix} B + D_x & J_c^T & J_d \\ J_c & 0 & 0 \\ J_d & 0 & D_d^{-1} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y_c \\ \Delta y_d \end{bmatrix} = \begin{bmatrix} \widetilde{r}_x \\ r_{y_c} \\ \widetilde{r}_{y_d} \end{bmatrix}, \quad (39)$$

where $\widetilde{r}_{y_d} := r_{y_d} + \left[(S_l^d)^{-1}V_l + (S_u^d)^{-1}V_u\right]^{-1}\widetilde{r}_d$. We note that the symmetric linear system (39) is commonly known in mathematical programming as the augmented system.

Specific to our approach is that the augmented system (39) is further eliminated, which is possible due to the compact inverse representation of $B + D_x$ that was derived in Section 4.2. Namely, since

$$\Delta x = (B + D_x)^{-1} \left(\widetilde{r}_x - J_c^T \Delta y_c - J_d^T \Delta y_d\right), \tag{40}$$

one can reduce the augmented system to a system in $(\Delta y_c, \Delta y_d)$ only:

$$\begin{bmatrix} J_c(B + D_x)^{-1} J_c^T & J_c(B + D_x)^{-1} J_d^T \\ J_d(B + D_x)^{-1} J_c^T & J_d(B + D_x)^{-1} J_d^T + D_d^{-1} \end{bmatrix} \begin{bmatrix} \Delta y_c \\ \Delta y_d \end{bmatrix} = \begin{bmatrix} J_c(B + D_x)^{-1} \widetilde{r}_x - r_{y_c} \\ J_d(B + D_x)^{-1} \widetilde{r}_x - \widetilde{r}_{y_d} \end{bmatrix}. \tag{41}$$

This linear system is dense but of small size (*i.e.*, $m = m_E + m_I$, which is assumed to be $O(1)$); therefore the space and solution time complexities, $O(m^2)$ and $O(m^3)$, respectively, are negligible. The dominant cost is given by the computation of the matrix and right-hand side in (41). For this, $(B + D_x)^{-1}$ is applied to J_c^T, J_d^T, $\widetilde{R}_x$. Only $m + 1$ such multiplication are needed because the symmetry is exploited. Since each of the multiplications has a cost of $O(ln)$, as we have showed in Section 4.2, the resulting time complexity is $O(mln)$. This is in fact the dominant time complexity term for the operations presented in this section. This should be apparent by remarking that the eliminations that lead to the reduced systems (40) and (41) involve only element-wise vector-vector operations and diagonal matrix-vector multiplications and, therefore, are of at most $O(n)$ complexity; furthermore, one should remark that the time complexity for evaluating the right-hand sides in (28) is only $O(mn)$.

Evidently, the space complexity for the IPM linear systems and for our elimination scheme is given by the $O(mn)$ storage for the Jacobians J_c and J_d (excluding the compact inverse representation of $B+D_x$, which is analyzed in the previous section). The rest of the linear algebra objects are diagonal matrices and vectors and have space cost of at most $O(n)$. In addition, $(m+1)n$ doubles are used to store the intermediary terms $(B + D_x)^{-1} J_c^T$, $(B + D_x)^{-1} J_d^T$, and $(B + D_x)^{-1} \widetilde{r}_x$ in (41). Hence, the total space complexity for the solution to the augmented system (41) is $O(2mn)$.

writing the augmented systema series sparse linear solves with multiple right-hand sides that is required to avoidsolveto solve the quasi-Newton linear system (28). This is required to handle the low-rank

4.4. Summary of the time and space complexities of linear algebra

It should be apparent from the complexity discussions presented in Section 4.2 and Section 4.3 that the solution of the IPM search direction linear systems (28) using the methodology proposed by this paper has

- $O((ml + l^2)n)$ time complexity and
- $O(2(l + m)n)$ space complexity.

4.5. Parallel computational approach and theoretical parallel speedup analysis

The data parallelism discussed in Section 2 applies equally to the data structures used by the IPM Algorithm 1 and the linear algebra technique we proposed in Section 4.2 and Section 4.3. The scalars and the data (*i.e.*, vectors and matrices) with storage that does not depend on the number of variables n are replicated on all MPI ranks. For example, the matrices L_k and D_k in the compact representation (31), the matrix $\mathcal{A}$ in the compact inverse representation (33), as well as the reduced IPM system matrix (41) are stored on all ranks. Vectors and matrices with storage depending on n are distributed across ranks; we mention that the P_k and W_k in the compact representation (31) are distributed row-wise.

Similarly, the parallel computations required by our decomposition fall under two categories: replicated computations corresponding to replicated data and "embarrassingly parallel" computations performed concurrently by the ranks on their local slices of data. For a given operation, the ideal execution time on P ranks is given by $t_0 + t_P + c_P$, where t_0 is the time spent in the replicated computations, t_P is the time spent by the ranks in the embarrassingly parallel computations, and c_P is the communication time. By Amdahl's law [2], the serial bottleneck t_0, communication cost c_P, and load imbalance are the limiting factors in achieving parallel efficiency. We note that that the ranks have identical workload throughout the proposed computational technique. Therefore, for simplicity, we assume t_P is the same across processes, that is, we assume perfect load balancing; note that in this case one necessarily obtains that $t_1 = Pt_p$.

We are interested in understanding the potential for speedup of our parallel linear algebra computations. For this we look at the relationship between the relative speedup with P ranks over 1 rank and the ideal speedup P, namely at their ratio $e_P = \frac{t_1+t_0}{t_p+c_P+t_0}/P$ as a measure of speedup efficiency: $e_P = 1$ corresponds to perfect speed-up and small e_P corresponds to poor speedup. We write $e_P = \frac{t_1+t_0}{t_1+P(c_p+t_0)} = 1 - \frac{\frac{P-1}{P}t_0+c_P}{\frac{t_1}{P}+t_0+c_P} \approx 1 - \frac{t_0+c_P}{\frac{t_1}{P}+t_0+c_P}$ for large P, which gives the following a relationship for the ideal speedup:

$$e_P \approx 1 - \frac{1}{\frac{t_1}{P(t_0+c_P)} + 1}. \tag{42}$$

This indicates that the speedup is good (e_P close to one), whenever $f_P := \frac{t_1/P}{t_0+c_P}$ is large. We chose to use f_P as an indicator of speed-up since it measures the ratio of "parallelizable" computations and total "serial" time, *i.e.*, serial bottleneck t_0 plus communication time c_P, and, thus, it is unit-less. For example, if f_P is 1 for some number of ranks P, then $e_P = 0.500$ (or 50% speed-up efficiency); if f_P is 10, then $e_P = 0.909$; and, if f_P is 100, then $e_P = 0.990$.

We now discuss the f_P term for the computation of the inverse compact representation (33) and the solution of the reduced IPM system (41). These two have the largest serial bottlenecks t_0 (as well as and nontrivial c_P). For the inverse compact representation (33), the dominant term in t_0 is coming from the factorization of $\mathcal{A}$ and is approximately $O(8l^3/3)$. The computation of $\mathcal{A}$

has a dominant cost $t_1 = O(2l^2n)$ as we have shown in Section 4.2. Therefore $f_P \approx \frac{3n}{4Pl}$, which shows that speed-up is good as long as n/P, which is the size of the local data, is large compared to l, the memory of the quasi-Newton method. A similar analysis reveals that $f_P \approx \frac{3nl}{8m^2P}$ for solving the reduced IPM system (41); this points out that the solution technique for system (41) also speeds up well as long as n/P is larger relatively to m^2/l.

We have neglected the communication time c_P in the analysis of the previous paragraph since it is not apparent to us what would be the exact theoretical cost c_P of `MPI_AllReduce`-based interprocess communication. We observe that the operations analyzed in the previous paragraph involve parallel matrix multiplications that require reducing $O(m^2)$ and $O(l^2)$ doubles on all processors; it is reasonable to assume that the cost c_P of each reduce operations is at least $O(\log(P) \cdot \max(l^2, m^2))$. Consequently, n/P needs to be larger than $\log(P)\max(l^2, m^2)$ in order to have a large f_P and obtain good speed-up. Finally, we observe that the serial bottleneck correspondent is only $\max(l, m^2/l)$, as we have shown in the previous paragraph. This alludes to the possibility that communication overhead can adversely impact the speed-up before (for a smaller P) the serial bottleneck does.

5. Parallel performance evaluation

The parallel linear algebra methodology of this paper has been implemented in the `Hiop` optimization solver. `Hiop` implements Algorithm 1 following closely the filter lineasearch IPM of Ipopt [40]. `Hiop` has similar iterations and iteration count to Ipopt using the monotone strategy for decreasing the log-barrier parameter μ. For this reason, we do not focus on evaluating the performance of the outer IPM algorithm since it has been already the subject of many studies and it proved to be robust and efficient; instead, we focus on benchmarking the implementation of the parallel linear algebra proposed in this paper.

`Hiop` is a compact C++ solver that relies on internal parallel data structures and parallel implementation for basic linear algebra (vectors and matrices) and optimization-related (iterates, directions, linear systems) data structures. These are implemented using BLAS/LAPACK for intranode computations and MPI for interprocess communication. In fact, these two packages are the only external dependencies of `Hiop`. As of September 2017, `Hiop` is in process of being released for the general public as a open-source under a BSD license.

5.1. Strong scaling study

To study `Hiop`'s speed-up potential and limitations, we perform a so-called strong scaling study, in which the problem size is kept constant and the number of nodes/cores is increased. For this we have interfaced `Hiop` with TOPOPT_In_PETSc [1], which is scalable code that uses finite element method and multigrid solvers for solving topology optimization problems, including structural problems such as (10). We solve the default problem test in TOPOPT_In_PETSc, a 3D cantilever beam discretized by $512 \times 256 \times 256$ finite elements, approximately 101.6

million degrees of freedom. For this problem the number of optimization variables n is slightly more than 33.5 million. In addition, Ipopt was interfaced with TOPOPT_In_PETSc as validation method for `Hiop`'s implementation. We mention that the results of both `Hiop` and Ipopt matched the MMA implementation present in TOPOPT_In_PETSc. Both `Hiop` and Ipopt are stopped when the NLP error is less than 10^{-5}. Ipopt was used with the MA27 as the linear solver. We mention that Ipopt runs in serial, using only one process.

The simulations were ran on the Quartz cluster at Lawrence Livermore National Laboratory. Quartz has 2 634 nodes, each with a Intel Xeon E5-2695 with 36 cores operating at 2.1 Ghz and 128 Gb RAM memory. The interconnect is Omni-Path and the nodes run a TOSS3 operating system. We used MVAPICH 2.2 and Intel 16.0.3 compilers with `-O2` code optimization flag.

P ranks	Time per iteration (s)		Speed-up efficiency e_P	
and cores	`Hiop`	Overall	`Hiop`	Overall
288	0.248	124.012	100.00%	100.00%
576	0.135	56.522	91.90%	109.70%
1 152	0.067	34.294	92.12%	90.40%
2 304	0.044	18.232	71.00%	85.03%
4,608	0.033	10.623	46.38%	72.96%

Table 1: Shown are `Hiop`'s and overall topology optimization average iteration times and speed-up efficiencies during a strong scaling study.

We started with a simulation on 288 ranks (288 cores/8 nodes). This run corresponds to the minimum number of nodes for which the test problem does not run out of memory. We then doubled the number of ranks repeatedly to up to 4 608 (4 608 cores/128 nodes). Since we focus on study of the speed-up of our linear algebra parallelization approach, that is, time-per-optimization iteration, we recorded and report the performance only for the first 20 iterations in order to avoid excessive consumption of core-hours. To give an idea about the cost of the simulation, we mention that `Hiop` requires about 60 iterations and more than 2 hours on 288 cores to solve the problem.

In Table 1 we show the execution times of `Hiop` and overall optimization simulation, as well as their speed-up efficiencies e_P. We first observe that the `Hiop` speeds up well up to 2 304 ranks/cores; for the 4 608 ranks simulation, the speed-up efficiency is only 46.38%. For this simulation, the iteration time is on average merely 0.0318 seconds, which makes it likely that the interprocess communication overhead on 128 nodes is the culprit for the deterioration of the speed-up efficiency, as we anticipated in Section 4.5.

We also observe that `Hiop`'s time is only a small fraction of the total simulation time ($< 0.3\%$). As a consequence, the speed-up efficiency is mainly given by the multigrid linear solver and is quite good. We mention that neither TOPOPT_In_PETSc nor `Hiop` were profiled and optimized for this architecture, which could drastically improve performance. Such efforts will be considered in future work. We also observed a significant variability in the execution times on the Quartz cluster. For example, we repeated the experiment for $P = 1,152$ and we obtained a difference of 65% in `Hiop`'s execution time and about 20% in the overall optimization time. Such variability could explain the superlinear speed-up for $P = 576$ in the overall optimization time and the superlinear speed-up in `Hiop` from $P = 576$ to $P = 1,152$.

Finally, in Figure 2, we show the execution times of Ipopt and overall optimization using Ipopt together with the corresponding times obtained with `Hiop` from columns two and three in Table 1). Ipopt, a serial solver, quickly becomes the serial bottleneck of the simulation, to the extent that it drastically affects the speed-up of the overall optimization process. As a side note, we mentioned that `Hiop` on only one process is still faster than Ipopt. The goal of this figure is mostly instructive, to point out that scalable solvers for the linear elasticity forward problem and adjoint sensitivities are not enough for scalable topology optimization and they need to be complemented with scalable optimization algorithms and solvers.

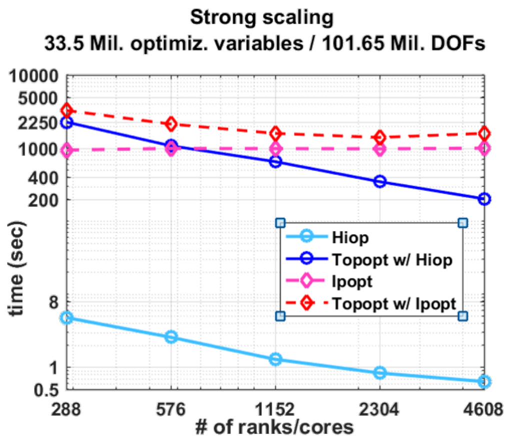

Figure 2: Execution (wall-clock) times of the optimization solvers `Hiop` and Ipopt and overall optimization with `Hiop` and Ipopt (first 20 iterations). The plots show the benefits of using parallel optimization (`Hiop` solver) in addition to parallel finite element analysis in topology optimization.

6. Conclusions

We have presented linear algebra techniques necessary to parallelize quasi-Newton interior-point methods using limited-memory secant updates. For this we leveraged the fact that the secant Hessian approximation has a small number of dense blocks that border a low-rank update of a diagonal matrix and exploited the data parallelism present in the simulation (in our case linear elasticity solve based on finite-element method and multigrid solvers) that defines the optimization problem. We also have shown linear time and space complexity of our linear algebra method and discussed the theoretical potential for speed-up. The latter was demonstrated computationally when solving a structural topology optimization problem in parallel on up to 4, 608 processes and cores.

Future work will be dedicated to a thorough profiling and code optimization of our parallel implementation. Given the reliance of methodology presented here on dense linear algebra, we also plan to revisit the linear algebra technique and possibly the IPM for GPU computations.

Acknowledgments

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. The author also acknowledges the support from the LDRD Program of Lawrence Livermore National Laboratory under the projects 16-ERD-025 and 17-SI-005.

References

[1] N. Aage, E. Andreassen, and B. S. Lazarov, *Topology optimization using petsc: An easy-to-use, fully parallel, open source topology optimization framework*, Structural and Multidisciplinary Optimization, 51 (2015), pp. 565–572.

[2] G. M. Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*, in Proceedings of the April 18-20, 1967, spring joint computer conference, ACM, 1967, pp. 483–485.

[3] M. P. Bendsøe and O. Sigmund, *Topology Optimization*, Springer Berlin Heidelberg, 2004.

[4] L. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. van Bloemen Waanders, *Real-Time PDE-Constrained Optimization*, Society for Industrial and Applied Mathematics, 2007.

[5] L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. van Bloemen Waanders, eds., *Large-Scale PDE-Constrained Optimization*, Lecture Notes in Computational Science and Engineering, Vol. 30, Springer-Verlag, Heidelberg, 2003.

[6] J. R. Birge and F. Louveaux, *Introduction to stochastic programming*, Springer-Verlag, New York,, 1997.

[7] G. Biros and O. Ghattas, *Parallel domain decomposition methods for optimal control of viscous incompressible flows*, in Proceedings of Parallel CFD '99, Williamsburg, VA, May 1999.

[8] ——, *Parallel Newton-Krylov algorithms for PDE–constrained optimization*, in Proceedings of SC99, Portland, Oregon, 1999.

[9] ——, *Parallel Lagrange–Newton–Krylov–Schur methods for PDE-constrained optimization. Part I: The Krylov–Schur Solver*, SIAM Journal on Scientific Computing, 27 (2005), pp. 687–713.

[10] J. F. Bonnans, J. C. Gilbert, C. Lemarechal, and C. A. Sagastizábal, *Numerical Optimization*, Springer-Verlag, New York, 2003.

[11] B. Bourdin, *Filters in topology optimization*, International Journal for Numerical Methods in Engineering, 50 (2001), pp. 2143–2158.

[12] T. E. Bruns and D. A. Tortorelli, *Topology optimization of non-linear elastic structures and compliant mechanisms*, Computer Methods in Applied Mechanics and Engineering, 190 (2001), pp. 3443 – 3459.

[13] R. H. Byrd, J. Nocedal, and R. B. Schnabel, *Representations of quasi-newton matrices and their use in limited memory methods*, Mathematical Programming, 63 (1994), pp. 129–156.

[14] N. Chiang, C. G. Petra, and V. Zavala, *Structured nonconvex optimization of large-scale energy systems using PIPS-NLP*, in Power Systems Computation Conference (PSCC), 2014, Aug 2014, pp. 1–7.

[15] J. C. De los Reyes, *Numerical PDE-constrained optimization*, Springer, 2015.

[16] E. Gawlik, T. Munson, J. Sarich, and S. M. Wild, *The TAO linearly-constrained augmented Lagrangian method for PDE-constrained optimization*, Preprint ANL/MCS-P2003-0112, Mathematics and Computer Science Division, (2012).

[17] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization*, SIAM Review, 47 (2005), pp. 99–131.

[18] J. Gondzio and A. Grothey, *Exploiting structure in parallel implementation of interior point methods for optimization*, Computational Management Science, 6 (2009), pp. 135–160.

[19] O. Guler, F. Gurtuna, and O. Shevchenko, *Duality in quasi-Newton methods and new variational characterizations of the DFP and BFGS updates*, Optimization Methods Software, 24 (2009), pp. 45–62.

[20] E. Haber, *Quasi-Newton methods for large-scale electromagnetic inverse problems*, Inverse Problems, 21 (2005), pp. 305–329.

[21] M. Heinkenschloss and D. Ridzal, *A matrix-free trust-region SQP method for equality constrained optimization*, SIAM Journal on Optimization, 24 (2014), pp. 1507–1541.

[22] R. Herzog and K. Kunisch, *Algorithms for PDE-constrained optimization*, GAMM-Mitteilungen, 33 (2010), pp. 163–176.

[23] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich, *Optimization with PDE Constraints*, Springer, 2009.

[24] G. J. Kennedy, *Large-scale multimaterial topology optimization for additive manufacturing*, in 56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Kissimmee, FL, 2015.

[25] D. P. Kouri, M. Heinkenschloss, D. Ridzal, and B. G. van Bloemen Waanders, *Inexact objective function evaluations in a trust-region algorithm for PDE-constrained optimization under uncertainty*, SIAM Journal on Scientific Computing, 36 (2014), pp. A3011–A3029.

[26] J. Linderoth and S. Wright, *Decomposition algorithms for stochastic programming on a computational grid*, Comput. Optim. Appl., 24 (2003), pp. 207–250.

[27] M. Lubin, J. A. J. Hall, C. G. Petra, and M. Anitescu, *Parallel distributed-memory simplex for large-scale stochastic LP problems*, Computational Optimization and Applications, 55 (2013), pp. 571–596.

[28] M. Lubin, C. G. Petra, M. Anitescu, and V. Zavala, *Scalable stochastic optimization of complex energy systems*, in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, New York, USA, 2011, ACM, pp. 64:1–64:64.

[29] T. Munson, J. Sarich, S. Wild, S. J. Benson, and L. C. McInnes, *TAO 3.7 user manual*, Tech. Rep. ANL/MCS-TM-322, Mathematics and Computer Science Division, Argonne National Laboratory, 2017.

[30] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, 2nd ed., 2006.

[31] W. Oliveira, C. Sagastizbal, and S. Scheimberg, *Inexact bundle methods for two-stage stochastic programming*, SIAM Journal on Optimization, 21 (2011), pp. 517–544.

[32] C. G. Petra, O. Schenk, M. Lubin, and K. Gärtner, *An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization*, SIAM Journal on Scientific Computing, 36 (2014), pp. C139–C162.

[33] S. Rojas-Labanda and M. Stolpe, *Benchmarking optimization solvers for structural topology optimization*, Structural and Multidisciplinary Optimization, 52 (2015), pp. 527–547.

[34] S. M. Ryan, R. J. B. Wets, D. L. Woodruff, C. Silva-Monroy, and J. P. Watson, *Toward scalable, parallel progressive hedging for stochastic unit commitment*, in 2013 IEEE Power Energy Society General Meeting, July 2013, pp. 1–5.

[35] K. Svanberg, *The method of moving asymptotes - a new method for structural optimization*, International journal for numerical methods in engineering, 24 (1987), pp. 359–373.

[36] ——, *A class of globally convergent optimization methods based on conservative convex separable approximations*, SIAM journal on optimization, 12 (2002), pp. 555–573.

[37] D. A. Tortorelli and P. Michaleris, *Design sensitivity analysis: Overview and review*, Inverse Problems in Engineering, 1 (1994), pp. 71–105.

[38] A. Wächter and L. T. Biegler, *Line search filter methods for nonlinear programming: Local convergence*, SIAM Journal on Optimization, 16 (2005), pp. 32–48.

[39] ——, *Line search filter methods for nonlinear programming: Motivation and global convergence*, SIAM Journal on Optimization, 16 (2005), pp. 1–31.

[40] A. Wächter and L. T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Mathematical Programming, 106 (2006), pp. 25–57.

[41] C. Zillober, *A globally convergent version of the method of moving asymptotes*, Structural and Multidisciplinary Optimization, 6 (1993), pp. 166–174.