Nonlinear Programming Strategies on High-Performance Computers

Jia Kang, Naiyuan Chiang, Carl D. Laird, and Victor M. Zavala

***Abstract*—We discuss structured nonlinear programming problems arising in control applications, and we review software and hardware capabilities that enable the efficient exploitation of such structures. We focus on linear algebra parallelization strategies and discuss how these interact and influence high-level algorithmic design elements required to enforce global convergence and deal with negative curvature in a nonconvex setting.**

I. NLP Framework and Notation

We consider the general nonlinear programming problem (NLP) of the form

$$\begin{aligned} &\min f(x) && && \text{(I.1a)}\\ &\text{s.t. } c(x) = 0 && (\lambda) && \text{(I.1b)}\\ &x \geq 0 && (\nu) && \text{(I.1c)} \end{aligned}$$

Here, $x \in \Re^n$ are the primal variables, $\lambda \in \Re^m$ are the dual variables for the equality constraints, and $\nu \in \Re^n_+$ are the dual variables for the bounds. The objective function $f(x) : \Re^n \to \Re$ and constraints $c(x) : \Re^n \to \Re^m$ are assumed to be twice continuously differentiable and are allowed to be *nonconvex* (the linear algebra concepts presented also hold for convex problems). General inequality constraints can be handled by introducing slack variables.

To solve the NLP (I.1a), we use a primal-dual *interior-point* framework. This framework is particularly attractive because it enables modular implementations of linear algebra kernels. The barrier subproblem solved in the IP framework is given by

$$\begin{aligned} \min \quad & \varphi^\mu(x) := f(x) - \mu \sum_{j=1}^{n} \ln x_{(j)} \\ \text{s.t.} \quad & c(x) = 0, \quad (\lambda) \end{aligned} \tag{I.2}$$

where $\mu > 0$ is the barrier parameter and $x_{(j)}$ denotes the j_{th} component of vector x. To solve each barrier problem we apply Newton's method to the Karush-Kuhn-Tucker system:

$$\nabla_x \varphi^\mu(x) + \nabla_x c(x)\lambda = 0 \tag{I.3a}$$

$$c(x) = 0 \tag{I.3b}$$

with the implicit restriction $x \geq 0$. This restriction is typically enforced using a fraction-to-the-boundary rule [39]. We denote the variables at iterate k as (x_k, λ_k). In a line-search setting [16, 47], the primal search direction d_k and the dual updates λ_k^+ are typically obtained by solving the *augmented* system:

$$\begin{bmatrix} W_k(\delta_w) & J_k^T \\ J_k & -\delta_c \mathbb{I}_m \end{bmatrix} \begin{bmatrix} d_k \\ \lambda_k^+ \end{bmatrix} = - \begin{bmatrix} g_k \\ c_k \end{bmatrix}. \tag{I.4}$$

Here, $c_k := c(x_k)$, $J_k := \nabla_x c(x_k)^T$, $g_k := \nabla_x \varphi_k^\mu$, $W_k(\delta_w) := H_k + \Sigma_k + \delta_w \mathbb{I}_n$, $H_k := \nabla_{xx}\mathcal{L}(x_k, \lambda_k)$, $\mathcal{L}(x_k, \lambda_k) := \varphi^\mu(x_k) + \lambda_k^T c(x_k)$, and $\Sigma_k := X_k^{-1} V_k$ with $V_k := \text{diag}(\nu_k)$, and $X_k := \text{diag}(x_k)$. The term $\mathbb{I}_n$ denotes an identity matrix of dimension n. We also define the primal and dual *regularization parameters* $\delta_w, \delta_c \geq 0$ and, to enable compact notation, we define the augmented matrix

$$M_k(\delta_w, \delta_c) := \begin{bmatrix} W_k(\delta_w) & J_k^T \\ J_k & -\delta_c \mathbb{I}_m \end{bmatrix}. \tag{I.5}$$

In a trust-region setting [9], the primal step is typically decomposed as $d_k = n_k + t_k$, where n_k is the normal component satisfying $J_k n_k = -c_k$ and t_k is the tangential component computed from the augmented system,

$$\begin{bmatrix} W_k(0) & J_k^T \\ J_k & \end{bmatrix} \begin{bmatrix} t_k \\ \lambda_k^+ \end{bmatrix} = - \begin{bmatrix} g_k + W_k(0) n_k \\ 0 \end{bmatrix}. \tag{I.6}$$

We note that regularization is not added directly in the augmented matrix as in the line-search setting. The reason is that regularization is treated implicitly in trust-region procedures. The structure of the augmented systems in both settings, however, is the same. Consequently, unless stated otherwise, we will refer to the more general augmented system (I.4). Many methods can be considered for the parallel solution of (I.4) including iterative linear algebra or tailored decomposition of problem-specific structure; however, the possibility of nonconvexity may impact the algorithms selected.

A key difference between convex and nonconvex NLPs is the presence of negative curvature. The presence of negative curvature indicates that the Newton step d_k might not correspond to a minimum of the associated quadratic programming problem. In a line-search setting this is an important issue because the Newton step cannot be guaranteed to provide a descent direction for the objective function when the constraint violation is sufficiently small. In particular, we seek that $d_k^T g_k < 0$ whenever $c_k \approx 0$. In a trust-region setting, on the other hand, we need to guarantee that the

N. Chiang and V. M. Zavala are with the Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA. E-mail: {nychiang, vzavala}@mcs.anl.gov. J. Kang, is a Ph.D. student in the Department of Chemical Engineering, Texas A&M University, College Station, TX. Email: jkang@tamu.edu. C. D. Laird is an Associate Professor in the School of Chemical Engineering, Purdue University, West Lafayette, IN. Email: carllaird@purdue.edu. Preprint ANL/MCS-P5359-0615.

step improves the Cauchy step. Consequently, detecting and handling negative curvature is critical to ensure robustness of the algorithm.

The presence of negative curvature can be checked by using a curvature test of the form

$$p_k^T W_k(\delta_w) p_k > 0 \tag{I.7}$$

for some vector p_k. Alternatively, we can check for negative curvature by monitoring the inertia of the augmented matrix $M_k(\delta_w, \delta_c)$.

In a trust-region setting, the curvature condition is often checked by using an iterative preconditioned conjugate gradient (PCG) approach (while computing the step t_k from (I.6)). In this approach the PCG iterates are projected onto the null-space of the constraint Jacobian using a special preconditioner. This approach is used in KNITRO [9]. In a line-search setting, the curvature condition can also be checked after computing the step by setting $p_k := d_k$ from (I.4) using any linear algebra approach as in PIPS-NLP [14]. The motivation for the curvature test (I.7) in the line-search setting becomes evident when we multiply the first row of the augmented system (I.4) by d_k^T. We obtain

$$-d_k^T g_k = d_k^T W_k(\delta_w) d_k - c_k^T \lambda_k^+ + \frac{1}{\delta_c}(\lambda_k^+)^T \lambda_k^+, \tag{I.8}$$

where we use the second row of the augmented system to note that $J_k d_k - \frac{1}{\delta_c}\lambda_k^+ = -c_k$. We can thus see that if the curvature condition (I.7) holds with $p_k := d_k$, we have that $d_k^T g_k < 0$ when $c_k \approx 0$, as desired.

The inertia of a matrix M is defined as the triplet Inertia$(M) := \{k_+, k_-, k_0\}$, where k_+, k_-, and k_0 are the number of positive, negative, and zero eigenvalues of matrix M. It is well known that the reduced Hessian $Z^T W_k(\delta_w) Z$ with Z satisfying $J_k Z = 0$ is positive definite if and only if the augmented matrix has an inertia of $\{n, m, 0\}$ [23]. In such a case we say that the inertia of $M_k(\delta_w, \delta_c)$ is *correct*. This result is of great practical significance because the inertia of the augmented matrix $M_k(\delta_w, \delta_c)$ can be recovered from a sparse LBL^T factorization (serial and parallel implementations for this are available in linear solver implementations such as MA57, Pardiso, MUMPS, and WSMP). When either the inertia is not correct or the curvature condition (I.7) does not hold, we must increase the primal regularization δ_w and compute an updated Newton step. This procedure continues until negative curvature is not present and we can thus guarantee the desired descent property. We highlight, however, that each regularization requires the solution of a new augmented system and it is thus expensive.

We may solve the augmented system and infer the inertia of the augmented matrix by permuting the matrix into the block partitioned matrix of the form

$$P M_k(\delta_w, \delta_c) P^T = \begin{bmatrix} K_0 & B \\ B^T & K_1 \end{bmatrix}, \tag{I.9}$$

where P is a nonsingular permutation matrix. Sylvester's law of inertia indicates that Inertia$(P M_k(\delta_w, \delta_c) P^T)$ = Inertia$(M_k(\delta_w, \delta_c))$. Moreover, Haynsworth's inertia additivity formula indicates that

$$\begin{aligned} &\text{Inertia}(M_k(\delta_w, \delta_c)) \\ &\quad = \text{Inertia}(K_0) + \text{Inertia}(K_1 - B^T K_0^{-1} B) && \text{(I.10a)} \\ &\quad = \text{Inertia}(K_1) + \text{Inertia}(K_0 - B K_1^{-1} B^T). && \text{(I.10b)} \end{aligned}$$

This formula is relevant because it can help us infer inertia using block permutations of the augmented system. We now provide some examples of how to do this in a general setting and in the next section we revisit this topic in a structured setting.

It is well known that the augmented system can be solved by using a normal or an augmented Lagrangian approach [19, 22]. The normal approach solves the following system:

$$\left(J_k W_k(\delta_w)^{-1} J_k^T + \delta_c \mathbb{I}_m\right) \lambda_k^+ = -c_k + J_k W_k(\delta_w)^{-1} g_k \tag{I.11a}$$

$$W_k(\delta_w) d_k = -g_k - J_k^T \lambda_k^+. \tag{I.11b}$$

We can interpret this as a permutation of the augmented system followed by a Schur decomposition. From (I.10) we thus have that

$$\begin{aligned} &\text{Inertia}(M_k(\delta_w, \delta_c)) \\ &\quad = \text{Inertia}(W_k(\delta_w)) + \text{Inertia}(-J_k W_k(\delta_w)^{-1} J_k^T - \delta_c \mathbb{I}_m). \end{aligned} \tag{I.12}$$

If $W_k(\delta_w)$ is positive definite, the inertia of $M_k(\delta_w, \delta_c)$ is correct if and only if $(J_k W_k(\delta_w)^{-1} J_k^T + \delta_c \mathbb{I}_m)$ is positive definite. Consequently, we can verify that we have the correct inertia for $M_k(\delta_w, \delta_c)$ by first choosing δ_w such that $W_k(\delta_w)$ is positive definite (this condition can be checked by using a PCG procedure or a Cholesky factorization). We then pick δ_c such that $(J_k W_k(\delta_w)^{-1} J_k^T + \delta_c \mathbb{I}_m)$ is positive definite (this can also be checked using PCG or Cholesky). This is beneficial because we can use existing parallel implementations for these approaches. The normal approach, however, is computationally beneficial only when the Hessian is diagonal or block diagonal and/or when the number of constraints is small.

The augmented Lagrangian approach solves the following system:

$$\left(W_k(\delta_w) + \frac{1}{\delta_c} J_k^T J_k\right) d_k = -g_k - \frac{1}{\delta_c} J_k^T c_k \tag{I.13a}$$

$$-\delta_c \lambda_k^+ = -c_k - J_k d_k \tag{I.13b}$$

Consequently, this approach does not require a factorization of the Hessian matrix $W_k(\delta_w)$ and thus provides a key advantage over the normal approach when the number of constraints is large. We can also interpret this approach as a permutation of the augmented system followed by a Schur decomposition. From (I.10) we have that

$$\begin{aligned} &\text{Inertia}(M_k(\delta_w, \delta_c)) \\ &\quad = \text{Inertia}(-\delta_c \mathbb{I}_m) + \text{Inertia}\left(W_k(\delta_w) + \frac{1}{\delta_c} J_k^T J_k\right) \\ &\quad = \{0, m, 0\} + \text{Inertia}\left(W_k(\delta_w) + \frac{1}{\delta_c} J_k^T J_k\right). \end{aligned} \tag{I.14}$$

We thus have that $M_k(\delta_w, \delta_c)$ has correct inertia if and only if the augmented Lagrangian Hessian $W_k(\delta_w) + \frac{1}{\delta_c} J_k^T J_k$ is positive definite. This condition can be checked by using a PCG or a Cholesky approach.

Despite the different avenues discussed to infer inertia, in some applications this information is simply not available. This situation can occur, for instance, when iterative linear algebra or general (e.g., LU) dense factorization approaches are used to solve the augmented system. This is important because many efficient linear algebra implementations for parallel architectures use these approaches (such as those implemented in the MAGMA, PETSc, and Trilinos libraries). When inertia information is not available, we can resort to the negative curvature condition (I.7).

II. Structured Optimization

We now present different structures arising in optimal control applications and discuss strategies for their parallel solution. Problem structures enable the implementation of decomposition techniques that can be used to accelerate solutions, avoid memory limitations, and/or facilitate model building. Such partitioning can be done externally at the problem formulation level or internally at the linear algebra level. The trade-offs between the two approaches are clear: the external approach is less intrusive and easier to implement but convergence rates and robustness are less favorable; while the internal approach is more intrusive and harder to implement but retains favorable convergence rates and robustness of existing algorithmic NLP frameworks. In particular, the interior-point framework previously discussed has superlinear convergence. We focus on the internal decomposition approach within an interior-point framework but provide pointers to existing work on external methods in Section V.

A. Primal Couping

NLPs arising in control applications are often large-scale but are also highly structured. Consider the following structure

$$\begin{aligned}
\min \quad & f_0(x_0) + \sum_{p \in \mathcal{P}} f_p(x_p, x_0) && && \text{(II.15a)}\\
\text{s.t.} \quad & c_0(x_0) = 0 && (\lambda_0) && \text{(II.15b)}\\
& c_p(x_p, x_0) = 0,\ p \in \mathcal{P} && (\lambda_p) && \text{(II.15c)}\\
& x_0 \geq 0 && (\nu_0) && \text{(II.15d)}\\
& x_p \geq 0,\ p \in \mathcal{P} && (\nu_p) && \text{(II.15e)}
\end{aligned}$$

We denote this NLP as a structured problem with *primal coupling*. Here, $\mathcal{P} := \{1...P\}$ is a set of partitions with variables $x_p \in \Re^{n_p}$, and the interface or coupling variables are $x_0 \in \Re^{n_0}$. The dual variables are $\lambda_0 \in \Re^{m_0}$ and $\lambda_p \in \Re^{m_p}$. This structure arises in stochastic model predictive control in which the coupling variables are controls and the partitions correspond to scenarios as discussed in [10, 29] and in parameter estimation problems as discussed in [48, 50]. This structure also arises in multi-stage optimal control formulations where the time horizon is partitioned into *stages* and the coupling variables correspond to states at neighboring nodes. Optimal control problems *over networks* can also be reformulated into this form.

The NLP representation (II.15) is convenient from an analysis standpoint. In an actual implementation, however, it might be convenient to *lift* the problem by introducing coupling variables. This can be done by duplicating the coupling variables x_0 in each partition (we denote this as $x_{0,p}$) and by introducing the dummy primal variables $y \in \Re^{n_0}$. This gives the following form,

$$\begin{aligned}
\min \quad & \sum_{p \in \mathcal{P}} \left(f_0(x_{0,p}) + f_p(x_p, x_0)\right) && && \text{(II.16a)}\\
\text{s.t.} \quad & c_0(x_{0,p}) = 0 && (\lambda_{0,p}) && \text{(II.16b)}\\
& c_p(x_p, x_{0,p}) = 0,\ p \in \mathcal{P} && (\lambda_p) && \text{(II.16c)}\\
& x_{0,p} \geq 0,\ p \in \mathcal{P} && (\nu_{0,p}) && \text{(II.16d)}\\
& x_p \geq 0,\ p \in \mathcal{P} && (\nu_p) && \text{(II.16e)}\\
& x_{0,p} = y,\ p \in \mathcal{P} && (\bar{\lambda}_p) && \text{(II.16f)}
\end{aligned}$$

In this formulation we have that the coupling between partitions and the coupling variables y is linear as opposed to nonlinear as in (II.15). Consequently, we do not need to compute derivatives for the coupling part (i.e., we only need derivatives for the partitions). This is beneficial when expressing structured problems by using off-the-shelf algebraic modeling languages such as AMPL, Pyomo, and JuMP because each partition can be expressed as an independent optimization problem [34]. The lifting approach retains the primal structure of the problem but alters the dual structure. Consequently, the dual variables of the original problem (II.15) cannot be easily recovered from those of (II.16). This relevant in certain applications.

The augmented system of the primal-coupled problem (II.15) can be permuted to the following block-bordered-diagonal (BBD) form,

$$\left[\begin{array}{c|cccc}
K_0 & B_1^T & B_2^T & \dots & B_P^T \\
\hline
B_1 & K_1 & & & \\
B_2 & & K_2 & & \\
\vdots & & & \ddots & \\
B_P & & & & K_P
\end{array}\right]
\left[\begin{array}{c}
\Delta w_0 \\ \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_P
\end{array}\right]
= -
\left[\begin{array}{c}
r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_P
\end{array}\right], \tag{II.17}$$

where $\Delta w_0 = (\Delta x_0, \Delta \lambda_0)$, $\Delta w_p = (\Delta x_p, \Delta \lambda_p)$,

$$K_0 = \left[\begin{array}{cc} W_0(\delta_w) & J_0^T \\ J_0 & -\delta_c \mathbb{I}_{m_0} \end{array}\right], \tag{II.18a}$$

$$K_p = \left[\begin{array}{cc} W_p(\delta_w) & J_p^T \\ J_p & -\delta_c \mathbb{I}_{m_p} \end{array}\right], \tag{II.18b}$$

$$B_p^T = \left[Q_p^T \; T_p^T\right], \tag{II.18c}$$

$J_0 = \nabla_{x_0} c_0(x_0)$, $J_p = \nabla_{x_p} c_p(x_0, x_p)$, $T_p = \nabla_{x_0} c_p(x_0, x_p)$, $W_0(\delta_w) = \nabla_{x_0,x_0} \mathcal{L} + X_0^{-1} V_0 + \delta_w \mathbb{I}_{n_0}$, $W_p(\delta_w) = \nabla_{x_p,x_p} \mathcal{L} + X_p^{-1} V_p + \delta_w \mathbb{I}_{n_p}$, and $Q_p = \nabla_{x_0,x_p} \mathcal{L}$. The augmented system of the lifted problem (II.16) can also be permuted into the BBD form (II.17) under an appropriate definition of the blocks.

The BBD system (II.17) can be solved in parallel by using a Schur complement decomposition approach. This requires the solution of the following systems:

$$\left(K_0 - \sum_{p \in \mathcal{P}} B_p K_p^{-1} B_p^T\right) \Delta w_0 = -r_0 + \sum_{p \in \mathcal{P}} K_p^{-1} B_p r_p \tag{II.19a}$$

$$K_p \Delta w_p = -r_p - B_p^T \Delta w_0, \; p \in \mathcal{P}. \tag{II.19b}$$

Here, $S{:=}K_0 - \sum_{p\in\mathcal{P}} B_p K_p^{-1} B_p^T$ is the *Schur complement matrix*. This is a symmetric and indefinite matrix of dimension $n_0{+}m_0$. From Haynsworth's formula (I.10) we have

$$\begin{aligned}&\text{Inertia}(M_k(\delta_w,\delta_c))\\ &\quad = \sum_{p\in\mathcal{P}} \text{Inertia}(K_p) + \text{Inertia}\left(K_0 - \sum_{p\in\mathcal{P}} B_p K_p^{-1} B_p^T\right).\end{aligned} \tag{II.20}$$

We recall that $n = n_0 + \sum_p n_p$ and $m = m_0 + \sum_p m_p$. Consequently, if we have that Inertia$(K_p){=}\{n_p, m_p, 0\}$ for all $p \in \mathcal{P}$ then the inertia of $M_k(\delta_w, \delta_c)$ is correct if and only if Inertia$(S) = \{n_0, m_0, 0\}$. One can obtain the inertia of the blocks K_p using LBL^T factorizations, but obtaining the inertia of the Schur matrix S using an LBL^T factorization can be inefficient because S is often a dense matrix or contains dense blocks. When this is the case, one can resort to the inertia-free curvature condition test (I.7) performed using the entire primal step d_k.

In the special case when $m_0 = 0$ (i.e., no constraints of the form $c_0(x_0) = 0$ are present in the problem) the inertia of $M_k(\delta_w, \delta_c)$ is correct if and only if S is positive definite [31]. Consequently, we can check for negative curvature of the entire augmented system by first picking δ_w such that Inertia$(K_p) = \{n_p, m_p, 0\}$ for all $p \in \mathcal{P}$ and then solving the Schur system (II.19a) using a Cholesky factorization or PCG approach and checking whether the matrix is positive definite; if it is not, then we increase δ_w and try again.

The Schur decomposition approach (II.19) *is the basic paradigm behind any parallel implementation.* In the most basic setting the idea is to factorize each block K_p independently in a given computing node and form the contributions $B_p K_p^{-1} B_p^T$. The contributions are then communicated to a coordinator node to form the Schur complement S, and we solve the Schur system (II.19a) to obtain the step for the coupling variables Δw_0. We then communicate Δw_0 to the partition nodes and use (II.19b) to obtain the steps for the partitions Δw_p in parallel.

The basic Schur setting has two key bottlenecks. First, forming the contributions $B_p K_p^{-1} B_p^T$ can be expensive when we have many columns in B_p^T (number of columns is $n_0 + m_0$). Second, factorizing the Schur matrix S can be challenging when $n_0 + m_0$ is large because this matrix is dense or is composed of dense sub-blocks.

In the special case when $m_0 = 0$, we can circumvent these issues by using a PCG procedure to solve the Schur system (II.19a). This approach requires only a product of the form $S \cdot v$ for a given vector v at each PCG iteration, and this can be obtained from products of the form $B_p K_p^{-1} B_p^T \cdot v$. Consequently, the contribution matrices $B_p K_p^{-1} B_p^T$ do not need to be formed explicitly. To precondition the PCG procedure, one can use an incomplete Cholesky factorization or an automatic BFGS preconditioner. The BFGS approach is particularly attractive because no factorizations are needed and it has been shown to dramatically reduce computing times over the basic Schur setting [31].

When $m_0 > 0$, we can still use a general iterative solver such as GMRES or QMR to solve the Schur system (II.19a). Obtaining a fast preconditioner, however, is nontrivial in this case because the Schur complement has no obvious structure and is indefinite. We can circumvent the Schur bottlenecks using direct factorization routines. We can form the products $B_p K_p^{-1} B_p^T$ indirectly by computing a symmetric indefinite factorization of the sparse indefinite matrix:

$$\left[\begin{array}{cc} K_p & B_p^T \\ B_p & \end{array}\right]. \tag{II.21}$$

This approach has been shown to decrease times by an order of magnitude [41]. One can then use a parallel dense factorization (such as those implemented in ELEMENTAL or MAGMA) to factorize the Schur complement S [37].

Another approach has been recently proposed in the context of stochastic programming to circumvent the bottlenecks of the basic Schur setting. The idea is to cluster scenarios to form a sparse compressed representation of the BBD system (II.17). This compression is used to compute a step for the coupling variables Δw_0, and then the partition steps are recovered from (II.19b) in parallel. This approach has shown significant reductions in computational time for problems in which $n_0 + m_0$ is large [11].

The *primal coupling* structure can be induced recursively. Consider that each partition $p \in \mathcal{P}$ is partitioned into P_p partitions with associated sets $\mathcal{P}_p$, $p \in \mathcal{P}$. This approach creates the variable partitions $x_p = (x_{p,0}, x_{p,1}, ..., x_{p,P_p})$, $p \in \mathcal{P}$ and splits the objective and constraints as follows:

$$f_p(x_0, x_p){=}f_{p,0}(x_0, x_{p,0}){+} \sum_{j\in\mathcal{P}_p} f_{p,j}(x_0, x_{p,0}, x_{p,j}), \; p \in \mathcal{P} \tag{II.22a}$$

$$c_p(x_0, x_p){=}\left\{\begin{array}{ll} c_{p,0}(x_0, x_{p,0}) & p \in \mathcal{P} \\ c_{p,j}(x_0, x_{p,0}, x_{p,j}), & j \in \mathcal{P}_p, \; p \in \mathcal{P}. \end{array}\right. \tag{II.22b}$$

We thus obtain an NLP of the following form:

$$\min \quad f_0(x_0) + \sum_{p\in\mathcal{P}} \left(f_{p,0}(x_0, x_{p,0}) + \sum_{j\in\mathcal{P}_p} f_{p,j}(x_0, x_{p,0}, x_{p,j})\right) \tag{II.23a}$$

$$
\begin{aligned}
\text{s.t.} \quad & c_0(x_0) = 0 && (\lambda_0) && \text{(II.23b)}\\
& c_{p,0}(x_0, x_{p,0}) = 0,\ p \in \mathcal{P} && (\lambda_{p,0}) && \text{(II.23c)}\\
& c_{p,j}(x_0, x_{p,0}, x_{p,j}) = 0,\ j \in \mathcal{P}_p,\ p \in \mathcal{P} && (\lambda_{p,j}) && \text{(II.23d)}\\
& x_0 \geq 0 && (\nu_0) && \text{(II.23e)}\\
& x_{p,0} \geq 0,\ p \in \mathcal{P} && (\nu_{p,0}) && \text{(II.23f)}\\
& x_{p,j} \geq 0,\ j \in \mathcal{P}_p,\ p \in \mathcal{P} && (\nu_{p,j}) && \text{(II.23g)}
\end{aligned}
$$

One can show that the nested problem yields an augmented system of the form (II.17) in which each diagonal block K_p has a BBD structure of the form

$$
K_p = \left[\begin{array}{c|cccc}
K_{p,0} & B_{p,1}^T & B_{p,2}^T & \dots & B_{p,P_p}^T \\
\hline
B_{p,1} & K_{p,1} & & & \\
B_{p,2} & & K_{p,2} & & \\
\vdots & & & \ddots & \\
B_{p,P_p} & & & & K_{p,P_p}
\end{array}\right], \quad p \in \mathcal{P}. \tag{II.24}
$$

This low-level system can also be solved by using a Schur decomposition approach. It is not difficult to see that we can infer the inertia of the augmented system of (II.23) recursively.

The recursive structure can naturally arise in applications, but it can also be induced as a way to trade off dimensionality of block partitions and of the Schur complement. For instance, consider an optimal control problem with n_x state variables and N time steps, and assume that each time step gives a block of dimension n_p. If we introduce many partitions (say $P = N$) each partition will be small (of dimension n_p) but the size of the Schur matrix will be large ($n_x \cdot P$) and perhaps unmanageable. Now assume that we partition the horizon into $P = N/2$ partitions in the first level, and then we now have that the Schur complement in the first level has a smaller dimension of $n_x \cdot N/2$ but each block partition has a larger dimension of $n_p \cdot N/2$. Each first-level block partition, however, can be split in further (say into $N/2$) partitions. Consequently, in the second level we have that each block is of dimension n_p and the second-level Schur complement is of dimension $n_x \cdot N/2$. We have thus split the complexity of the Schur complement. We note, however, that this two-level approach is not perfectly parallelizable. This is because the computations of the first-level cannot proceed until those of the second-level are completed.

B. Dual Coupling

Another important structure arising in applications is:

$$
\begin{aligned}
\min \quad & \sum_{p \in \mathcal{P}} f_p(x_p) && && \text{(II.25a)}\\
\text{s.t.} \quad & c_p(x_p) = 0,\ p \in \mathcal{P} && (\lambda_p) && \text{(II.25b)}\\
& x_p \geq 0,\ p \in \mathcal{P} && (\nu_p) && \text{(II.25c)}\\
& \sum_{p \in \mathcal{P}} \Pi_p\, x_p = 0 && (\lambda_0) && \text{(II.25d)}
\end{aligned}
$$

We refer to this NLP as a structured problem with *dual coupling*. Here, $\lambda_0 \in \Re^{m_0}$ are the multipliers of the coupling constraints, and $\Pi_p \in \Re^{m_0 \times n_p}$ are the coupling matrices. This structure also arises in multi-stage optimal control formulations; but, as opposed to the primal coupling case, the state transition constraints define the coupling. A similar observation applies to decomposition over networks [38]. One can also reformulate stochastic programming problems in this form by using the so-called nonanticipativity constraints [7].

The augmented system of this problem can also be permuted into the BBD form (II.17) by defining $\Delta w_0 = \Delta\lambda_0$, $\Delta w_p = (\Delta x_p, \Delta\lambda_p)$,

$$
\begin{aligned}
& K_0 = -\delta_c \mathbb{I}_{m_0}, \quad K_p = \left[\begin{array}{cc} W_p(\delta_w) & J_p^T \\ J_p & -\delta_c \mathbb{I} \end{array}\right]\\
& B_p^T = [\Pi_p\ 0],
\end{aligned} \tag{II.26}
$$

$J_p = \nabla_{x_p} c_p(x_p)$, and $W_p(\delta_w) = \nabla_{x_p, x_p} \mathcal{L} + X_p^{-1} V_p + \delta_w \mathbb{I}$. If we solve the BBD system using a Schur decomposition, we obtain

$$
-\left(\delta_c \mathbb{I}_{m_0} + \sum_{p \in \mathcal{P}} B_p K_p^{-1} B_p^T\right) \Delta w_0 = -r_0 + \sum_{p \in \mathcal{P}} K_p^{-1} B_p r_p. \tag{II.27}
$$

The coefficient matrix of this system is a regularized version of the dual Hessian used in dual-Newton decomposition methods [32].

C. Primal-Dual Coupling

Similar to the case of primal coupling, the dual coupling structure can be inherited recursively. Consider, for instance, the following problem:

$$
\begin{aligned}
\min \quad & \sum_{p \in \mathcal{P}} \sum_{j \in \mathcal{P}_p} f_{p,j}(x_{p,j}) && && \text{(II.28a)}\\
\text{s.t.} \quad & c_{p,j}(x_{p,j}) = 0,\ p \in \mathcal{P},\ j \in \mathcal{P}_p && (\lambda_{p,j}) && \text{(II.28b)}\\
& x_{p,j} \geq 0,\ p \in \mathcal{P},\ j \in \mathcal{P}_p && (\nu_{p,j}) && \text{(II.28c)}\\
& \sum_{j \in \mathcal{P}_p} \Pi_{p,j}\, x_{p,j} = 0 && (\lambda_{0,j}) && \text{(II.28d)}\\
& \sum_{p \in \mathcal{P}} \Pi_p\, x_p = 0 && (\lambda_0) && \text{(II.28e)}
\end{aligned}
$$

Here, $x_p := (x_{p,1}, ..., x_{p,P_p})$ is the connection of variables in partition $p \in \mathcal{P}$. One immediately realizes that primal and dual structures can also be superimposed to create a rich set of structures. For instance, consider the problem with primal-dual coupling:

$$
\begin{aligned}
\min \quad & f_0(x_0) + \sum_{p \in \mathcal{P}} \sum_{j \in \mathcal{P}_p} f_{p,j}(x_{p,j}, x_0) && && \text{(II.29a)}\\
\text{s.t.} \quad & c_0(x_0) = 0 && (\lambda_0) && \text{(II.29b)}\\
& c_{p,j}(x_{p,j}, x_0) = 0,\ p \in \mathcal{P},\ j \in \mathcal{P}_p && (\lambda_{p,j}) && \text{(II.29c)}\\
& x_0 \geq 0 && (\nu_0) && \text{(II.29d)}\\
& x_{p,j} \geq 0,\ p \in \mathcal{P},\ j \in \mathcal{P}_p && (\lambda_{p,j}) && \text{(II.29e)}\\
& \sum_{j \in \mathcal{P}_p} \Pi_{p,j}\, x_{p,j} = 0,\ p \in \mathcal{P}. && (\lambda_{0,p}) && \text{(II.29f)}
\end{aligned}
$$

These recursive structures can be exploited by solvers such as PIPS-NLP. One can use these constructs to consider, for insatnce, space-time partitioning of optimal control problems over networks [17].

III. High-Performance Computing Platforms

Successful implementation of the ideas discussed in Section II requires consideration of the strengths and limitations of the particular parallel computing architecture targeted. Parallel architectures are typically classified according to Flynn's taxonomy, where a key differentiator is whether the architecture can perform different instructions simultaneously.

At one extreme, single-instruction-multiple-data (SIMD) architectures can perform parallel computations; however, each core must be executing the same fundamental instruction simultaneously (albeit on different data). These SIMD architectures are highly appropriate for iterative linear algebra [12] (e.g., PCG, GMRES), but their limitations make it difficult to exploit these architectures for general structural decomposition. Furthermore, while these architectures provide massive parallelism at a relatively low price (e.g. the Tesla K80 provides almost 5,000 cores for a few thousand dollars), they are most effective when the algorithm can be implemented by using a large number of threads to keep the cores loaded (e.g., while waiting for memory operations to complete). Doing so may be difficult in structural decomposition of many large-scale problems. Graphics processing units, commonly used for parallel scientific computing, are a hybrid of the pure SIMD architecture. They comprise a number of multiprocessors, each containing a number of streaming processors or cores. The cores within a single multiprocessor share instructions (i.e. they are true SIMD); and although each multiprocessor can support execution of different kernels, these architectures still do not support parallel execution of different instructions at the same granularity as the number of processing cores. Thus, efficient utilization of these hybrid architectures demands the same considerations as do pure SIMD architectures.

Multiple-instruction-multiple-data (MIMD) architectures are more typically utilized for problems like those described in this paper. These architectures have the disadvantage of fewer cores than currently available SIMD architectures (at least for the equivalent cost) but have the advantage that each core is more capable. Most notably, MIMD architectures can execute different instructions simultaneously. Within this class, we consider *shared-memory* and *distributed-memory* architectures. With shared-memory architectures, all cores have access to the same physical memory. With this architecture, communicating or sharing data between processes can be very fast. However, one bottleneck that can arise is the total bandwidth available for accessing memory. Shared-memory MIMD architectures include common consumer-grade multi-core computers, and a typical shared-memory MIMD architecture has access to far fewer cores than is possible with current distributed-memory machines.

Distributed-memory MIMD architectures can be scaled to significantly larger numbers of cores. In distributed-memory architectures, individual machines are connected with one another through standard or specialized networking interfaces, and communication between processes occurs across this network. For many algorithms, intercommunication becomes the bottleneck that can deteriorate parallel efficiency as the number of cores for a particular problem increases. Each machine has its own dedicated access to local memory, and these architectures are highly efficient for problems with a high percentage of independent computation and less intensive communication needs. *Beowulf clusters* are one implementation for distributed-memory parallel computing, and access to computing resources like these is common for industrial and academic researchers. Modern clusters are hybrid architectures, typically composed of a large number of shared-memory, multicore machines (nodes) with fast network access for communication between nodes.

The software tools available for developing parallel algorithms depend on the architecture targeted. Distributed-memory and shared-memory MIMD architectures benefit from the availability of a wide range of compiler tools. For shared-memory machines, parallelism can be handled any number of ways, including the direct use of threads or APIs such as OpenMP. For distributed-memory machines, the most widely used paradigm for algorithms discussed here is the *Message Passing Interface* (MPI), and several implementations exist for different architectures. MPI can also be used in shared-memory environments, but care must be taken to ensure competitive performance with dedicated shared-memory tools. For SIMD architectures, the software tool used for development of parallel algorithms is often hardware specific. For example, NVidia has released the Tesla series of graphics processing units for scientific computing along with the platform-specific CUDA API and compiler extensions. While work has been done on general parallel tools for use on different architectures (e.g. OpenCL), these cannot compete with the performance of dedicated tools.

IV. Software and Implementation

Parallel strategies for NLP problems based on problem partitioning or decomposition can be used at the problem level or the linear algebra level. We focus on those strategies that achieve parallel speedup by exploiting problem structure and decomposing the internal linear algebra operations performed by the NLP algorithm. While significant work has been done on parallel algorithms for simulation and optimization of partial differential equations with notable codes PETSc [4–6] and Trillinos [27], we restrict our discussion to the problem structures described in Section II.

The dominant computational cost for the interior-point methods described in this paper is the solution of an augmented system like that in (I.4). Two broad strategies can be used for parallel solution of the augmented system: interface the NLP code with an existing, off-the-shelf parallel linear solver, or write a parallel decomposition approach specifically tailored to the structure of the problem. Several general parallel linear solvers exist, including shared-memory parallel solvers such as PARDISO [33, 44–46] and MA86/MA97 [28] and shared/distributed-memory solvers such as MUMPS [2, 3], WSMP [25], and Elemental [42]. Many of these solvers have been used with nonlinear interior-point methods, and IPOPT has existing interfaces to MA86, MA97, MUMPS, PARDISO, and WSMP.

While ease of implementation is a major benefit of using an existing parallel linear solver in one's NLP code, truly scalable performance to hundreds of processors typically requires using a decomposition specifically tailored to the structure of the problem. Amdahl's law provides an estimate of the maximum achievable speedup as the inverse of the fraction of the algorithm that must be executed serially ($S^{\infty}{=}1/\phi_s$) [1]. Therefore, in order to achieve significant speedup on large computing clusters, scale-dependent operations must be serialized. These include model evaluations (which can be parallelized at a block level), and all *vector*, *vector-vector*, and *matrix-vector* operations. For the block structures described in this paper, parallel evaluation of the scale-dependent operations is relatively straightforward for all but the solution of the linear system used to compute the step. Utilizing the techniques outlined in Section II, parallel decomposition algorithms can be implemented to exploit the specialized block structure in the linear system. At the core of this decomposition approach, the implementation makes parallel calls to separate instances of a serial linear solver for individual blocks (which themselves have the same structure as (I.4)). MA27 and MA57 from the Harwell Subroutine Library [28] have been widely used in serial nonlinear interior-point algorithms and for block decomposition in parallel interior-point methods [14, 31, 49, 50]. Of course, many of the parallel linear solvers discussed above perform well in serial, and can be used in this context as well.

Several nonlinear interior-point algorithms have been developed based on structural decomposition of the linear algebra, including OOPS[20, 21], PIPS-NLP[14, 36], PRBLOCK_IP[13] and Schur-IPOPT [31, 50]. In Castro [13] structured, convex QP problems with constraint coupling are solved through a method that performs Cholesky factorizations on the diagonal blocks and a PCG method for the linking constraints. An explicit Schur-complement approach based on the IPOPT algorithm is implemented in [50]. This algorithm is appropriate for problems with mild primal coupling; however, the performance deteriorates significantly as the number of coupling variables increases. This is due to the explicit formation of the Schur complement through repeated backsolves and the direct factorization of the dense Schur complement. This work is extended by Kang et al. [31], using a PCG approach on the implicit equation for the Schur complement. This approach avoids the need to form and factorize the Schur complement, however, it is not appropriate for all the structures explored in this paper. The PIPS and PIPS-NLP codes implement a number of improvements over standard algorithms, including the use of factorizations of (II.21) in place of repeated backsolves with columns from B_p^T [41], support for recursive block structures, parallel dense factorization of the Schur complement [37], and iterative methods on the Schur complement [40]. Another recent code, IPCLUSTER [11], implements an interior-point method for stochastic programming problems that improves the computational time by constructing a sparse, compressed representation of the structured KKT system to compute the step in the coupling variables.

While parallel computing architectures are becoming ubiquitous, a major barrier to the widespread adoption of parallel NLP codes has been the lack of appropriate modeling languages. While many modeling languages exist, the parallel implementations described above have unique requirements. For efficient scale-up to many processors, the model must be evaluated in parallel, and few languages support this directly.

Furthermore, for many large-scale problems, construction of the entire model on a serial machine is not possible because of memory and time limitations. Therefore, these languages must support parallel instantiation of partial models along with appropriate metadata to describe this structure to the solver. For many problems, the modeler is aware of the structure and can provide guidance on the construction and labeling. Several languages support suffixes as a mechanism for assigning metadata to variables and constraints, including AMPL [18]. This mechanism was used in [50] and [31] to describe coupling where each block in the problem was coded as a separate AMPL model and several instances of the AMPL Solver Library (ASL) were used to support parallel evaluation of the NLP residuals and gradients. Recent work has sought to simplify this effort through the development of modeling languages that provide native support for interfacing with parallel solvers.

Pyomo [26] is a python-based open-source algebraic modeling language that supports the definition and solution of optimization applications using the Python scripting language. It is portable and can be used on most platforms. Pyomo supports the general concept of model blocks and allows for custom modeling extensions. The PySP framework (based on Pyomo) provides a high-level interface for parallel instantiation and evaluation of block-structured stochastic programming problems, including interfaces to parallel decomposition algorithms. The structure-conveying modeling language (SML) proposed by [15] provides an extension to the AMPL modeling language to support the concept of blocks. A model generation package has been developed for SML that supports parallel instantiation of models described by the block structure in SML [24]. Based on the Julia programming language, JuMP [35] provides a mathematical programming modeling language that has compilation and execution speeds similar to those of AMPL, while retaining much of the flexibility of traditional scripting languages. StochJuMP [30] provides an extension to JuMP to support parallel model construction and evaluation for stochastic programming problems. These new developments in modeling languages open the door for mainstream use of specialized parallel solvers.

V. EXTERNAL DECOMPOSITION

In this section we discuss strategies for external problem decomposition. Most of these strategies do not have convergence guarantees in nonconvex settings but can be often used as a heuristic. Moreover, the convergence of external decomposition strategies can be slow, but these strategies are easier to implement. In this section we consider the NLP with dual coupling (II.25), and we assume that the bounds are handled by using a logarithmic barrier function.

A. Dual Decomposition

In this setting we seek to maximize the Lagrangian dual function

$$D(\lambda_0) = \min_{w_p} \sum_{p \in \mathcal{P}} \left(\varphi(x_p) + \lambda_p^T c_p(x_p) + \lambda_0^T \Pi_p x_p \right), \tag{V.30}$$

where $w_p := (x_p, \lambda_p)$ and we denote the solution at fixed λ_0 as $w_p(\lambda_0)$. Because the dual function is separable, each partition $p \in \mathcal{P}$ can be solved in parallel when the duals λ_0 are fixed. In a first-order method, we update the duals λ_0 as

$$\lambda_0^+ = \lambda_0 + \nabla_{\lambda_0} D(\lambda_0), \tag{V.31}$$

where

$$\nabla_{\lambda_0} D(\lambda_0) = \sum_{p \in \mathcal{P}} \Pi_p x_p \tag{V.32}$$

is the gradient of $D(\lambda_0)$. The dual update takes place in a coupling node. The first-order dual decomposition method is easy to implement and requires minimal communication between the partitions and the coupling node (only $\Pi_p x_p$ and λ_0 need to be communicated). The convergence rate, however, is only linear (this method is essentially a steepest descent method).

In a second-order method (dual-Newton) we update the duals in the coordinator node as,

$$\lambda_0^+ = \lambda_0 - \nabla^2_{\lambda_0} D(\lambda_0)^{-1} \nabla_{\lambda_0} D(\lambda_0), \tag{V.33}$$

where $\nabla^2_{\lambda_0} D(\lambda_0)$ is the Hessian of the dual function. One can show that this dual Hessian matrix has the following structure:

$$\nabla^2_{\lambda_0} D(\lambda_0) = -\sum_{p \in \mathcal{P}} B_p K_p^{-1} B_p^T, \tag{V.34}$$

where K_p is defined in (II.26) and is evaluated at $w_p(\lambda_0)$ with $\delta_w = 0$. The dual-Newton method has superlinear convergence and the dual update (V.33) can be implemented by using a CG method as is done in [32]. With this, it is possible to limit communication between the partition nodes and the coupling node (only vectors need to be communicated). One can also use automatic BFGS preconditioning to accelerate the CG method as proposed in [31]. Alternatively, one can approximate the dual Hessian matrix using a BFGS approach.

B. ADMM

The alternating direction method of multipliers (ADMM) uses the augmented Lagrangian function:

$$\begin{aligned} &\mathcal{L}_A(w_1, w_2, ..., w_P, \lambda_0) \\ &:= \sum_{p \in \mathcal{P}} \left(\varphi(x_p) + \lambda_p^T c_p(x_p) + \lambda_0^T \Pi_p x_p \right) + \frac{\rho}{2} \left\| \sum_{p \in \mathcal{P}} \Pi_p x_p \right\|^2. \end{aligned} \tag{V.35}$$

We then apply a Gauss-Seidel scheme of the form

$$\begin{aligned} w_1^{k+1} &= \text{argmin}_{w_1} \mathcal{L}_A(w_1, w_2^k, ..., w_P^k, \lambda_0^k) \\ w_2^{k+1} &= \text{argmin}_{w_2} \mathcal{L}_A(w_1^{k+1}, w_2, ..., w_P^k, \lambda_0^k) \\ &\vdots \\ w_P^{k+1} &= \text{argmin}_{w_P} \mathcal{L}_A(w_1^{k+1}, w_2^{k+1}, ..., w_P, \lambda_0^k) \\ \lambda_0^{k+1} &= \lambda_0^k + \rho \sum_{p \in \mathcal{P}} \Pi_p x_p^{k+1}. \end{aligned}$$

The key observation is that each subproblem in this scheme is separable, and consequently the partitions can be computed independently. The last step is performed in a coupling node. The communication between partition nodes, however, is not necessarily trivial. In particular, each partition p needs to obtain the components $\Pi_{p'} x_{p'}$ from all partitions $p' \neq p$. Note also that the ADMM approach is tightly connected to the dual decomposition approach with first-order dual updates (V.31). The key difference between the approaches is that the augmented Lagrangian regularization term couples the subproblems and thus the augmented Lagrangian is no longer separable for fixed multipliers λ_0. This forces the use of a Gauss-Seidel scheme to enable decomposition. This makes the convergence rate of ADMM linear. The ADMM approach has been used for stochastic programs in [43] and in a more general setting in [8].

VI. Summary and Challenges

Interior-point methods have emerged as an efficient platform for parallel solution of structured nonlinear programming problems, oweing largely to the fact that the linear system solved to find the step direction retains consistent structure over all iterations. Algorithms have been developed that provide parallel decomposition of the linear algebra for block-bordered problems with primal coupling and dual coupling, as well as combined and recursive structures. These methods have also been shown to provide effective parallel solution of spatially and temporally discretized problems. At this point, the most successful implementations are based on parallel MIMD architectures (clusters), and high-level modeling languages are being developed and interfaced with these algorithms, providing support for parallel model construction and evaluation in a more user friendly environment.

Nevertheless, several research opportunities and challenges remain in this area.

- These methods have seen little use outside the expert research community. Improvements are still necessary in modeling languages and implementation details such as ease of installation on high-performance computing software.
- Emerging architectures like the GPU provide potential for massively parallel computions at relatively low cost. However, the SIMD nature of these architectures makes it significantly more challenging to implement effective parallel algorithms. More research is necessary to determine effective ways of utilizing these architectures for

parallel solution of general and block-structured NLP problems.

- Iterative linear solvers provide a natural framework for parallel algorithms, and they are highly appropriate for SIMD architectures. While there has been significant work on the use of iterative methods for the augmented system, for problems of general structure (or general structure within blocks), effective preconditioners are difficult to find. These methods have not been as successful as direct factorization methods based on block decomposition, and more research in this area is required.
- The majority of research in this area has focused on block-bordered diagonal structures like those arising with primal and dual coupling. More research is necessary for other common structures, including those arising from time-discretized systems and network-structured problems.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of science, under Contract No. DE-AC02-06CH11357. Thanks is also extended for partial financial support provided to Carl Laird by the National Science Foundation (CAREER Grant CBET# 0955205).

References

[1] Gene 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*, pages 483–485. ACM, 1967.

[2] Patrick R Amestoy, Iain S Duff, and J-Y L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer methods in applied mechanics and engineering*, 184(2):501–520, 2000.

[3] Patrick R Amestoy, Iain S Duff, Jean-Yves LExcellent, and Jacko Koster. Mumps: a general purpose distributed memory sparse solver. In *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, pages 121–130. Springer, 2001.

[4] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[5] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, and Hong Zhang. PETSc Web page. http://www.mcs.anl.gov/petsc, 2014. URL http://www.mcs.anl.gov/petsc.

[6] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015. URL http://www.mcs.anl.gov/petsc.

[7] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[8] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[9] Richard H Byrd, Jorge Nocedal, and Richard A Waltz. Knitro: An integrated package for nonlinear optimization. In *Large-scale nonlinear optimization*, pages 35–59. Springer, 2006.

[10] Giuseppe C Calafiore and Lorenzo Fagiano. Stochastic model predictive control of lpv systems via scenario optimization. *Automatica*, 49(6):1861–1866, 2013.

[11] Yankai Cao, Carl D Laird, and Victor M Zavala. Clustering-based preconditioning for stochastic programs. *submitted to Computational Optimization and Applications*, 2015.

[12] Yankai Cao, Arpan Seth, and Carl D Laird. A parallel augmented lagrangian interior-point approach for large-scale nlp problems on graphics processing units. *submitted to Computers and Chemical Engineering*, 2015.

[13] Jordi Castro. An interior-point approach for primal block-angular problems. *Computational Optimization and Applications*, 36(2-3):195219, 2007. doi: 10.1007/s10589-006-9000-1.

[14] Naiyuan Chiang, Cosmin G Petra, and Victor M Zavala. Structured nonconvex optimization of large-scale energy systems using pips-nlp. In *Proc. of the 18th Power Systems Computation Conference (PSCC), Wroclaw, Poland*, 2014.

[15] Marco Colombo, Andreas Grothey, Jonathan Hogg, Kristian Woodsend, and Jacek Gondzio. A structure-conveying modelling language for mathematical and stochastic programming. *Mathematical Programming Computation*, 1(4):223–247, 2009.

[16] Frank E Curtis, Olaf Schenk, and Andreas Wächter. An interior-point algorithm for large-scale nonlinear optimization with inexact step computations. *SIAM Journal on Scientific Computing*, 32(6):3447–3475, 2010.

[17] Klaus Ehrhardt and Marc C Steinbach. *Nonlinear optimization in gas networks*. Springer, 2005.

[18] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press (now an imprint of Boyd & Fraser Publishing Co.), Danvers, MA, USA, 1993. ISBN 0-89426-232-7.

[19] Michael P Friedlander and Dominique Orban. A primal–dual regularized interior-point method for convex quadratic programs. *Mathematical Programming Computation*, 4(1):71–107, 2012.

[20] Jacek Gondzio and Andreas Grothey. Parallel interior-

point solver for structured quadratic programs: Application to financial planning problems. *Annals of Operations Research*, 152(1):319–339, 2007.

[21] Jacek Gondzio and Andreas Grothey. Exploiting structure in parallel implementation of interior point methods for optimization. *Computational Management Science*, 6(2):135–160, 2009.

[22] Jacek Gondzio and Robert Sarkissian. Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96(3):561–584, 2003.

[23] Nicholas IM Gould. On practical conditions for the existence and uniqueness of solutions to the general equality quadratic programming problem. *Mathematical Programming*, 32(1):90–99, 1985.

[24] Andreas Grothey and Feng Qiang. Psmg: A parallel problem generator for structure conveying modelling language for mathematical programming. *presentation at ICCOPT*, 2009, 2013.

[25] Anshul Gupta. Wsmp: Watson sparse matrix package (part-i: direct solution of symmetric sparse systems). *IBM TJ Watson Research Center, Yorktown Heights, NY, Tech. Rep. RC*, 21886, 2000.

[26] William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

[27] Michael A Heroux, Roscoe A Bartlett, Vicki E Howle, Robert J Hoekstra, Jonathan J Hu, Tamara G Kolda, Richard B Lehoucq, Kevin R Long, Roger P Pawlowski, Eric T Phipps, et al. An overview of the trilinos project. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):397–423, 2005.

[28] HSL. A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk, 2011.

[29] Rui Huang and Lorenz T Biegler. Robust nonlinear model predictive controller design based on multi-scenario formulation. In *Proc. of the American Control Conference*, pages 2341–2342, 2009.

[30] Joey Huchette, Miles Lubin, and Cosmin Petra. Parallel algebraic modeling for stochastic optimization. In *Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 29–35. IEEE Press, 2014.

[31] Jia Kang, Yankai Cao, Daniel P Word, and CD Laird. An interior-point method for efficient solution of block-structured nlp problems using an implicit schur-complement decomposition. *Computers & Chemical Engineering*, 71:563–573, 2014.

[32] Attila Kozma, Emil Klintberg, Sebastien Gros, and Moritz Diehl. An improved distributed dual Newton-CG method for convex quadratic programming problems. In *American Control Conference (ACC), 2014*, pages 2324–2329. IEEE, 2014.

[33] A. Kuzmin, M. Luisier, and O. Schenk. Fast methods for computing selected elements of the greens function in massively parallel nanoelectronic device simulations. In F. Wolf, B. Mohr, and D. Mey, editors, *Euro-Par 2013 Parallel Processing*, volume 8097 of *Lecture Notes in Computer Science*, pages 533–544. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40046-9. doi: 10.1007/978-3-642-40047-6_54. URL http://dx.doi.org/10.1007/978-3-642-40047-6_54.

[34] Carl D Laird and Lorenz T Biegler. Large-scale nonlinear programming for multi-scenario optimization. In *Modeling, simulation and optimization of complex processes*, pages 323–336. Springer, 2008.

[35] Miles Lubin and Iain Dunning. Computing in operations research using julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

[36] Miles Lubin, Cosmin G Petra, Mihai Anitescu, and Victor Zavala. Scalable stochastic optimization of complex energy systems. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–10. IEEE, 2011.

[37] Miles Lubin, Cosmin G Petra, and Mihai Anitescu. The parallel solution of dense saddle-point linear systems arising in stochastic programming. *Optimization Methods and Software*, 27(4-5):845–864, 2012.

[38] Ion Necoara, Carlo Savorgnan, Dinh Quoc Tran, Johan Suykens, and Moritz Diehl. Distributed nonlinear optimal control using sequential convex programming and smoothing techniques. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 543–548. IEEE, 2009.

[39] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[40] C. G. Petra and M. Anitescu. A preconditioning technique for schur complement systems arising in stochastic optimization. *Computational Optimization and Applications*, 52:315–344, June 2012.

[41] Cosmin G Petra, Olaf Schenk, Miles Lubin, and Klaus GŁertner. An augmented incomplete factorization approach for computing the schur complement in stochastic optimization. *SIAM Journal on Scientific Computing*, 36(2):C139–C162, 2014.

[42] Jack Poulson, Bryan Marker, Robert A Van de Geijn, Jeff R Hammond, and Nichols A Romero. Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software (TOMS)*, 39(2):13, 2013.

[43] Charles H Rosa and Andrzej Ruszczyński. On augmented lagrangian decomposition methods for multistage stochastic programs. *Annals of Operations Research*, 64(1):289–309, 1996.

[44] Olaf Schenk, Andreas Wchter, and Michael Hagemann. Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization. *Computational Optimization and Applications*, 36(2-3):321–341, 2007. ISSN 0926-6003. doi: 10.1007/s10589-006-9003-y. URL http://dx.doi.org/10.1007/s10589-006-9003-y.

[45] Olaf Schenk, Matthias Bollhöfer, and Rudolf A. Römer. On large-scale diagonalization techniques for the an-

derson model of localization. *SIAM Rev.*, 50(1):91–112, February 2008. ISSN 0036-1445. doi: 10.1137/070707002. URL http://dx.doi.org/10.1137/070707002.

[46] Olaf Schenk, Andreas Wächter, and Martin Weiser. Inertia-revealing preconditioning for large-scale nonconvex constrained optimization. *SIAM Journal on Scientific Computing*, 31(2):939–960, 2008.

[47] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[48] Daniel P Word, Derek AT Cummings, Donald S Burke, Sopon Iamsirithaworn, and Carl D Laird. A nonlinear programming approach for estimation of transmission parameters in childhood infectious disease using a continuous time model. *Journal of The Royal Society Interface*, 9(73):1983–1997, 2012.

[49] Daniel P Word, Jia Kang, Johan Akesson, and Carl D Laird. Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Computational Optimization and Applications*, 59(3):667–688, 2014.

[50] Victor M Zavala, Carl D Laird, and Lorenz T Biegler. Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. *Chemical Engineering Science*, 63(19): 4834–4845, 2008.