SIAM J. SCI. COMPUT.
Vol. 36, No. 2, pp. C139–C162

AN AUGMENTED INCOMPLETE FACTORIZATION APPROACH FOR COMPUTING THE SCHUR COMPLEMENT IN STOCHASTIC OPTIMIZATION[*]

COSMIN G. PETRA[†], OLAF SCHENK[‡], MILES LUBIN[§], AND KLAUS GÄERTNER[¶]

Abstract. We present a scalable approach and implementation for solving stochastic optimization problems on high-performance computers. In this work we revisit the sparse linear algebra computations of the parallel solver PIPS with the goal of improving the shared-memory performance and decreasing the time to solution. These computations consist of solving sparse linear systems with multiple sparse right-hand sides and are needed in our Schur-complement decomposition approach to compute the contribution of each scenario to the Schur matrix. Our novel approach uses an incomplete augmented factorization implemented within the PARDISO linear solver and an outer BiCGStab iteration to efficiently absorb pivot perturbations occurring during factorization. This approach is capable of both efficiently using the cores inside a computational node and exploiting sparsity of the right-hand sides. We report on the performance of the approach on high-performance computers when solving stochastic unit commitment problems of unprecedented size (billions of variables and constraints) that arise in the optimization and control of electrical power grids. Our numerical experiments suggest that supercomputers can be efficiently used to solve power grid stochastic optimization problems with thousands of scenarios under the strict "real-time" requirements of power grid operators. To our knowledge, this has not been possible prior to the present work.

Key words. parallel linear algebra, stochastic programming, stochastic optimization, parallel-interior point, economic dispatch, unit commitment

AMS subject classifications. 65F05, 65F08, 90C51, 90C15, 90C05, 65Y05

DOI. 10.1137/130908737

1. Introduction. In this paper, we present novel linear algebra developments for the solution of linear systems arising in interior-point methods for the solution of

[*]Submitted to the journal's Software and High-Performance Computing section February 6, 2013; accepted for publication (in revised form) January 22, 2014; published electronically March 27, 2014. This work was supported by the U.S. Department of Energy under contract DE-AC02-06CH11357. This research used resources of the Laboratory Computing Resource Center and the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. Computing time on Intrepid was granted by a 2012 Department of Energy INCITE award "Optimization of Complex Energy Systems under Uncertainty," PI Mihai Anitescu, Co-PI Cosmin G. Petra. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.
http://www.siam.org/journals/sisc/36-2/90873.html

[†]Corresponding author. Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439 (petra@mcs.anl.gov).

[‡]Institute of Computational Science, Università della Svizzera italiana, Via Giuseppe Buffi 13, CH-6900 Lugano, Switzerland (olaf.schenk@usi.ch).

[§]MIT Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139 (mlubin@mit.edu).

[¶]Weierstrass Institute for Applied Analysis and Stochastics, Mohrenstrasse 39, 10117 Berlin, Germany (klaus.gaertner@wias-berlin.de).

structured optimization problems of the form

$$
\begin{aligned}
&\min_{x_i, i=0,\dots,N} \left(\frac{1}{2}x_0^T Q_0 x_0 + c_0^T x_0\right) + \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}x_i^T Q_i x_i + c_i^T x_i\right)\\
&\text{subject to (s.t.)} \quad
\begin{array}{lllll l}
T_0 x_0 & & & & & = b_0,\\
T_1 x_0 + & W_1 x_1 & & & & = b_1,\\
T_2 x_0 + & & W_2 x_2 & & & = b_2,\\
\vdots & & & \ddots & & \vdots\\
T_N x_0 + & & & & W_N x_N & = b_N,\\
x_0 \geq 0, & x_1 \geq 0, & x_2 \geq 0, & \dots & x_N \geq 0. &
\end{array}
\end{aligned}
\tag{1.1}
$$

Optimization problems of the form (1.1) are known as convex quadratic optimization problems with *dual block-angular* structure. Such problems arise as the *extensive form* in stochastic optimization, being either deterministic equivalents or sample average approximations (SAAs) of two-stage stochastic optimization problems with recourse [34]. Two-stage stochastic optimization problems are formulated as

$$
\begin{aligned}
&\min_{x} \left(\frac{1}{2}x_0^T Q_0 x + c^T x_0\right) + \mathbb{E}_\xi[G(x_0,\xi)]\\
&\text{s.t. } T_0 x_0 = b_0, x_0 \geq 0,
\end{aligned}
\tag{1.2}
$$

where the *recourse function* $G(x_0, \xi)$ is defined by

$$
\begin{aligned}
G(x_0,\xi) = \min_x \frac{1}{2}x^T Q_\xi x + c_\xi^T x\\
\text{s.t. } T_\xi x_0 + W_\xi x = b_\xi, x \geq 0.
\end{aligned}
\tag{1.3}
$$

The expected value $\mathbb{E}[\cdot]$, which is assumed to be well defined, is taken with respect to the random variable ξ, which contains the data $(Q_\xi, c_\xi, T_\xi, W_\xi, b_\xi)$. The SAA problem (1.1) is obtained by generating N samples $(Q_i, c_i, T_i, W_i, b_i)$ of ξ and replacing the expectation operator with the sample average. The matrix Q_ξ is assumed to be positive semidefinite for all possible ξ. W_ξ, the *recourse* matrix, is assumed to have full row rank. T_ξ, the *technology* matrix, need not have full rank. The deterministic matrices Q_0 and T_0 are assumed to be positive semidefinite and of full row rank, respectively. The variable x_0 is called the *first-stage* decision, which is a decision to be made now. The *second-stage* decision x is a recourse or corrective decision that one makes in the future after some random event occurs. The stochastic optimization problem finds the optimal decision to be made now that has the minimal expected cost in the future.

Stochastic optimization is one of the main sources of extremely large dual block-angular problems. SAA problems having billions of variables can be easily obtained in cases when a large number of samples is needed to accurately capture the uncertainties. Such instances necessitate the use of distributed-memory parallel computers. Dual block-angular optimization problems are also natural candidates for decomposition techniques that take advantage of the special structure. Existing parallel decomposition procedures for the solution of dual angular problems are reviewed by Vladimirou and Zenios [37]. Subsequent to their review, Linderoth and Wright [15] developed an asynchronous approach combining l_∞ trust regions with Benders decomposition on a large computational grid. Decomposition inside standard optimization techniques applied to the extensive form has been implemented in the state-of-the-art software package OOPS [10] as well as by some of the authors in PIPS-IPM [20, 19, 24] and PIPS-S [18]. OOPS and PIPS-IPM implement interior-point algorithms and PIPS-S implements the revised simplex method. The decomposition is obtained by specializing linear algebra to take advantage of the dual block-angular structure of the problems.

This paper revisits linear algebra techniques specific to interior-point methods applied to optimization problems with dual block-angular structure. Decomposition of linear algebra inside interior-point methods is obtained by applying a Schur complement technique (presented in section 3). The Schur complement approach requires at each iteration of the interior-point method solving a dense linear system (the Schur complement) and a substantial number of sparse linear systems for each scenario (needed to compute each scenario's contribution to the Schur complement). For a given scenario, the sparse systems of equations share the same matrix, a situation described in the linear algebra community as solving linear systems with *multiple right-hand sides*. The number of the right-hand sides is large relative to the number of the unknowns or equations for many stochastic optimization problems, which is unusual in traditional linear algebra practices. Additionally, the right-hand sides are extremely sparse, a feature that should be exploited to reduce the number of floating-point operations.

The classical Schur complement approach is very popular not only in the optimization community [10], but also in domain decomposition [8, 14, 16, 17, 27, 28, 29] and parallel threshold-based incomplete factorizations [26, 27, 28, 41] of sparse matrices. A complete list of applications is virtually impossible because of its popularity and extensive use. Previously, we followed the classical path in PIPS-IPM and computed the exact Schur complement by using off-the-shelf linear solvers, such as MA57 and WSMP, and solved for each right-hand side (in fact for small blocks of right-hand sides). However, this approach has two important drawbacks: (1) the sparsity of the right-hand sides is not exploited, sparse right-hand sides being a rare feature of linear solvers; and (2) the calculations limit the efficient use of multicore shared-memory environments because it is well known that the triangular solves do not scale well with the number of cores [13].

Here, we develop a fast and parallel reformulation of the sparse linear algebra calculations that circumvents the issues in the Schur complement presented in the previous paragraph. The idea is to leverage the good scalability of parallel factorizations as an alternative method of building the Schur complement. We employ an *incomplete augmented factorization* technique that solves the sparse linear systems with multiple right-hand sides at once using an incomplete sparse factorization of an auxiliary matrix. This auxiliary matrix is based on the original matrix augmented with the right-hand sides. In this paper, we also concentrate both on the node and internode level parallelism. Previous Schur/hybrid solvers all solve the global problem iteratively, while our approach instead complements existing direct Schur complement methods and implementations. In our application, the interior-point systems are indefinite and require advanced pivoting and sophisticated solution refinement based on preconditioned BiCGStab in order to maintain numerical stability. In addition, BiCGStab method is needed to cheaply absorb the pivot perturbations that occur from the factorization.

These new algorithmic developments and their implementation details in PARDISO and PIPS-IPM are presented in section 3. In section 4 we study the large-scale computational performance and parallel efficiency on different high-performance computing platforms when solving stochastic optimization problems of up to two billion variables and up to almost two billion constraints. Such problems arise in the optimization of the electrical power grid and are described in section 2. We also discuss in this section the impact of this work on the operational side of the power grid management.

Notation and terminology. Lower-case Latin characters are used for vectors, and upper-case Latin characters are used for matrices. Lower-case Greek characters denote scalars. Borrowing the terminology from stochastic optimization, we refer to the zero-indexed variables or matrices of problem (1.1) as "first stage," the rest as "second stage." By a scenario i ($i = 1, \dots, N$) we mean the first-stage data (Q_0, c_0 and T_0) and second-stage data Q_i, c_i, T_i, and W_i. Unless stated otherwise, lower indexing of a vector indicates the first-stage or second-stage part of the vector (not its components). Matrices Q and A are a compact representation of the quadratic term of the objective and of the constraints:

$$Q = \begin{bmatrix} Q_0 & & & \\ & Q_1 & & \\ & & \ddots & \\ & & & Q_N \end{bmatrix} \text{ and } A = \begin{bmatrix} T_0 & & & \\ T_1 & W_1 & & \\ \vdots & & \ddots & \\ T_N & & & W_N \end{bmatrix}.$$

2. Motivating application: Stochastic unit commitment for power grid systems.

Today's power grid will require significant changes in order to handle increased levels of renewable sources of energy; these sources, such as wind and solar, are fundamentally different from traditional generation methods because they cannot be switched on at will. Instead, their output depends on the weather and may be highly variable within short time periods. Figure 1 illustrates the magnitude and frequency of wind supply fluctuations under hypothetical adoption levels compared

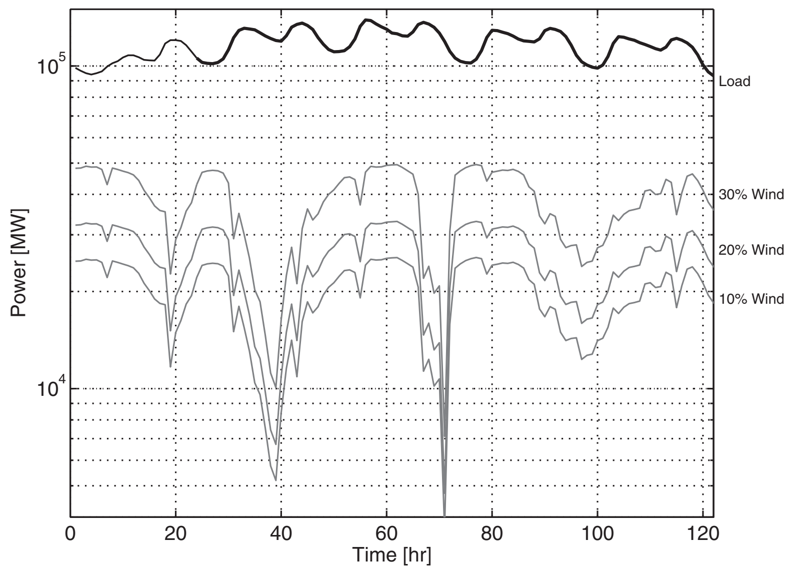

FIG. 1. *Snapshot of total load and wind supply variability at different adoption levels.*

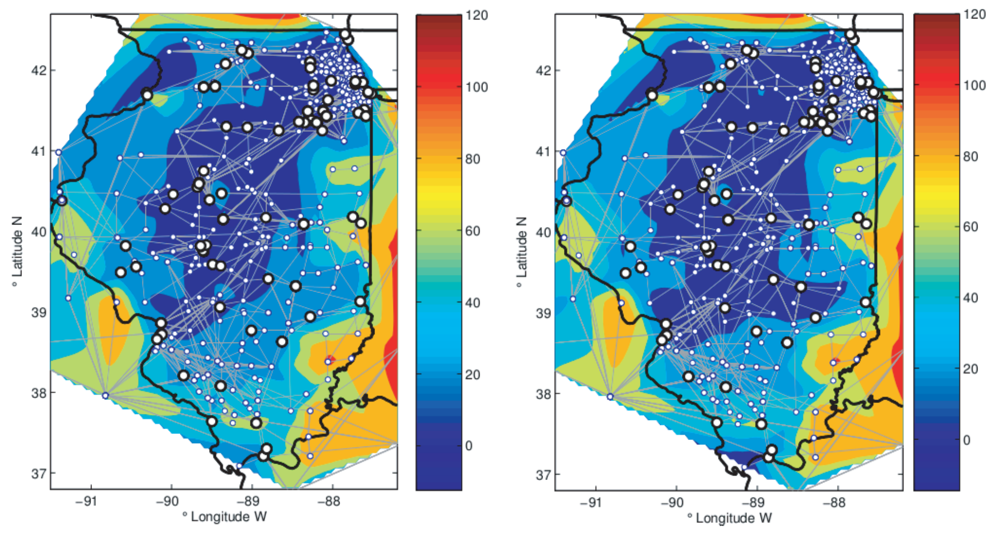

FIG. 2. *Snapshot of the price distribution across the state of Illinois under deterministic (left) and stochastic (right) formulations. We note more uniform prices in the interior of the state under stochastic optimization, as this approach enables the anticipation of uncertainties. Transmission grid is also shown. Figures credit: Victor Zavala.*

with a typical total load profile in Illinois. Uncertainty in weather forecasts and other risks such as generator and transmission line failure are currently mitigated by using conservative reserve levels, which typically require extra physical generators operating so that their generation levels may be increased on short notice. Such reserves can be both economically and environmentally costly. Additionally, inevitable deviations from output levels estimated by weather forecasts can lead to inefficiencies in electricity markets manifested as wide spatiotemporal variations of prices (see Figure 2).

Stochastic optimization has been identified in a number of studies as a promising mathematical approach for treating such uncertainties in order to reduce reserve requirements and stabilize electricity markets in the next-generation power grid [3, 5, 25]. Constantinescu et al. [5] made the important empirical observation that while reality may deviate significantly from any single weather forecast, a suitably generated family of forecasts using numerical weather prediction and statistical models can capture spatiotemporal variations of weather over wide geographical regions. Such a family of forecasts naturally fits within the paradigm of stochastic optimization when considered as samples (scenarios) from a distribution on weather outcomes.

Computational challenges, however, remain a significant bottleneck and barrier to real-world implementation of stochastic optimization of energy systems. This paper is the latest in a line of work [19, 20, 24] intended to address these challenges by judicious use of linear algebra and parallel computing within interior-point methods.

Power-grid operators (ISOs) solve two important classes of optimization problems as part of everyday operations; these are unit commitment (UC) and economic dispatch (ED) [33]. UC decides the optimal on/off schedule of *thermal* (coal, nuclear) generators over a horizon of 24 hours or longer. (California currently uses 72-hour horizons.) This is a nonconvex problem that is typically formulated as a mixed-integer linear optimization problem and solved by using commercial software [35]. In practice, a near-optimal solution to the UC problem must be computed within an hour.

In our analysis, we consider a two-stage stochastic optimization formulation for UC. The problem has the following structure (c.f. [19]; see [2, 3, 33] for more details):

$$\min \left(\sum_{k=0}^{T}\sum_{j\in\mathcal{G}} f_j \cdot x_{k,j}\right) + \frac{1}{N}\sum_{s\in\mathcal{N}}\left(\sum_{k=0}^{T}\sum_{j\in\mathcal{G}} c_j \cdot G_{s,k,j}\right) \tag{2.1a}$$

$$\text{s.t. } G_{s,k+1,j} = G_{s,k,j} + \Delta G_{s,k,j},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{G}, \tag{2.1b}$$

$$\begin{aligned}\sum_{(i,j)\in\mathcal{L}_j} P_{s,k,i,j} + \sum_{i\in\mathcal{G}_j} G_{s,k,i} &= \sum_{i\in\mathcal{D}_j} D_{s,k,i}\\ &\quad - \sum_{i\in\mathcal{W}_j} W_{s,k,i},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{B} \qquad (p_{s,k,j}),\end{aligned} \tag{2.1c}$$

$$P_{s,k,i,j} = b_{i,j}(\theta_{s,k,i} - \theta_{s,k,j}),\ s\in\mathcal{N}, k\in\mathcal{T}, (i,j)\in\mathcal{L}, \tag{2.1d}$$

$$0 \le G_{s,k,j} \le x_{k,j} G_j^{\max},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{G}, \tag{2.1e}$$

$$|\Delta G_{s,k,j}| \le \Delta G_j^{\max},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{G}, \tag{2.1f}$$

$$|P_{s,k,i,j}| \le P_{i,j}^{\max},\ s\in\mathcal{N}, k\in\mathcal{T}, (i,j)\in\mathcal{L}, \tag{2.1g}$$

$$|\theta_{s,k,j}| \le \theta_j^{\max},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{B}, \tag{2.1h}$$

$$x_{k,j}\in\{0,1\},\ k\in\mathcal{T}, j\in\mathcal{G}. \tag{2.1i}$$

Here, $\mathcal{G}, \mathcal{L}$, and $\mathcal{B}$ are the sets of generators, lines, and transmission nodes (intersections of lines, known as buses) in the network in the geographical region, respectively. $\mathcal{D}_j$ and $\mathcal{W}_j$ are the sets of demand and wind-supply nodes connected to bus j, respectively. The symbol $\mathcal{N}$ denotes the set of scenarios for wind level and demand over the time horizon $\mathcal{T} := \{0, \ldots, T\}$. The first stage decision variables are the generator on/off states $x_{k,j}$ over the complete time horizon. The decision variables in each second-stage scenario s are the generator supply levels $G_{s,k,j}$ for time instant k and bus j, the transmission line power flows $P_{s,k,j}$, and the bus angles $\theta_{s,k,j}$ (which are related to the *phase* of the current). The random data in each scenario are the wind supply flows $W_{s,k,i}$ and the demand levels $D_{s,k,i}$ across the network. The values of $G_{s,0,j}$ and $x_{0,j}$ are fixed by initial conditions.

Constraints (2.1c) and (2.1d) balance the power flow across the network according to Kirchoff's current law and the grid's physical network and are known as direct current (DC) power flow equations. DC power flow equations are linear approximations of the highly nonlinear power flow equations for alternative currents and are used to keep the simulation and optimization of large power systems computationally tractable [4, 23]. Constraints (2.1f) are the so-called ramp constraints that restrict how quickly generation levels can change. The objective function contains the fixed costs f_j for operating a generator and the generation costs c_j. Generation costs are convex and are more accurately modeled as quadratic, although in practice they are treated as linear and piecewise linear functions for simplicity. Our solver is equally capable of handling both cases, but in our test problems we used the linear form.

Note that the scenarios are coupled only by the constraint (2.1e), which enforces that the generation level in each scenario be zero if the generator is off at a particular time. Under this model, the on/off states are chosen such that (1) under each scenario, there is a feasible generation schedule, and (2) the (approximate) expected value of the generation costs is minimized.

In this work, we consider solving a convex relaxation of (2.1) obtained by replacing the binary restrictions (2.1i) with the constraints $0 \le x_{k,j} \le 1$. This relaxation is a linear optimization problem that would be solved at the "root node" of branch-and-bound approaches for solving (2.1). Empirically, deterministic UC problems have been observed to have a small gap between the optimal value of the convex relaxation and the true optimal solution when combined with cutting-plane techniques and feasibility heuristics, which are standard practice in mixed-integer optimization. This often implies that a sufficiently small gap is obtainable with little or no enumeration of the branch-and-bound tree [35]. Our future work will be devoted to implementing and developing such techniques and heuristics for stochastic UC formulations.

Even without any additional work, one can estimate an upper bound on the optimality gap between the relaxation and the nonconvex mixed-integer solution. Note that in the given formulation, the generator states $x_{k,j}$ in a feasible solution to the convex relaxation may be "rounded up" to 1 to obtain a feasible solution to the nonconvex problem. By rounding, we increase the objective value by at most $T\sum_{j\in\mathcal{G}} f_j$, which, because of the technological characteristics of the thermal units, is empirically smaller than the contribution to the objective from the generation costs c_j. We therefore conservatively expect an increase in the objective by at most 50%, providing us with a optimality gap. Intuitively, the convex part of the problem (generation costs) contributes a more significant proportion of the objective value than the nonconvex part (fixed costs).

We believe it is reasonable, therefore, to first focus our efforts on the computational tractability of the convex relaxation. Our developments apply directly to stochastic ED formations as well.

The convex relaxation of (2.1) itself is an extremely large-scale linear optimization problem. Our model incorporates the transmission network of the state of Illinois, which contains approximately 2,000 transmission nodes, 2,500 transmission lines, 900 demand nodes, and 300 generation nodes (illustrated in Figure 2). A deterministic formulation over this geographical region can have as many as 100,000 variables and constraints since the network constraints are imposed separately for each time period. In our stochastic formulation, this number of variables and constraints is effectively multiplied by the number of scenarios. As scenarios effectively correspond to samples in a high-dimensional Monte Carlo integration, it is reasonable to desire the capability to use thousands if not tens of thousands; hence total problem sizes of tens to hundreds of millions of variables, which are presently far beyond the capabilities of commercial solvers, are easily obtainable. The number of variables in the first-stage block x_0 is the number of generators times the number of time steps, leading to sizes of 10,000 or more, which makes parallel decomposition nontrivial.

3. Computational approach and implementation.

In this section we first provide a compact presentation of the Mehrotra's primal-dual path-following algorithm and the Schur complement-based decomposition of the linear algebra from PIPS-IPM. After that we present the novel approach for computing the Schur complement.

3.1. Interior-point method.

Let us consider a general form of a quadratic programming (QP) problem:

$$\min \frac{1}{2}x^TQx + c^Tx \text{ subject to } Ax = b,\ x \geq 0. \tag{3.1}$$

We consider only convex QPs (Q needs to be positive semidefinite) and linear programming problems ($Q = 0$). Additionally, the matrix A is assumed to have full row rank. Observe that stochastic programming problem (1.1) is a convex QP.

Path-following interior-point methods for the solution of problem (3.1) make use of the "central path," which is a continuous curve $(x(\mu), y(\mu), z(\mu))$, $\mu > 0$, satisfying

$$\begin{aligned} Qx - A^T y - z &= -c, \\ Ax &= b, \\ xz &= \mu e, \\ x, z &> 0. \end{aligned} \tag{3.2}$$

Here $y \in \mathbb{R}^m$ and $z \in \mathbb{R}^n$ correspond to the Lagrange multipliers, $e = [\ 1 \quad 1 \quad \dots \quad 1\ ]^T \in \mathbb{R}^n$, and xz denotes the componentwise product.

In the case of a feasible problem (3.1), the above system has a unique solution $(x(\mu), y(\mu),\ z(\mu))$ for any $\mu > 0$, and, as μ approaches zero, $(x(\mu), y(\mu), z(\mu))$ approaches a solution of (3.1); see Chapter 2 in [40]. A path-following method is an iterative numerical process that follows the central path in the direction of decreasing μ toward the solution set of the problem. The iterates generated by the method generally do not stay on the central path. Rather, they are located in a controlled neighborhood of the central path that is a subset of the positive orthant.

In the past two decades, predictor-corrector methods have emerged as practical path-following IPMs in solving linear and quadratic programming problems. Among the predictor-corrector methods, the most successful is Mehrotra's predictor-corrector algorithm. Although Mehrotra [22] presented his algorithm in the context of linear programming, it has been successfully applied also to convex quadratic programming [9] and standard monotone linear complementarity problems [42]. It also has been widely used in the implementation of several IPM-based optimization packages, including OB1 [21], HOPDM [11], PCx [6], LIPSOL [43], and OOQP [9].

Two linear systems of the form (3.4) are solved at each IPM iteration, one to compute predictor search direction and one to compute corrector search directions. For a detailed description of Mehrotra's method used in this paper, we refer the reader to [9] and [22, 40]. Let us denote the kth IPM iteration by (x_k, y_k, z_k). Also let X_k and Z_k denote the diagonal matrices with the (positive) entries given by x_k and z_k. The linear system solved during both the predictor and corrector phase to obtain the search direction $(\Delta x_k, \Delta y_k, \Delta z_k)$ is

$$Q\Delta x_k - A^T \Delta y_k \quad - \Delta z_k = r_k^1, \tag{3.3}$$

$$A\Delta x_k \qquad\qquad\qquad = r_k^2, \tag{3.4}$$

$$Z_k \Delta x_k \qquad\quad + X_k \Delta z_k = r_k^3. \tag{3.5}$$

While the right-hand sides r_k^1, r_k^2, and r_k^3 are different for the predictor and the corrector, the matrix remains the same. (This feature of the Mehrotra's algorithm gives important computational savings, since only one factorization, not two, per IPM iteration is required.)

By performing block elimination for Δz_k, the linear systems (3.4) can be reduced to the following symmetric indefinite linear system:

$$\begin{bmatrix} Q + D_k^2 & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ -\Delta y_k \end{bmatrix} = \begin{bmatrix} r_k^1 + X_k^{-1} r_k^3 \\ r_k^2 \end{bmatrix}, \tag{3.6}$$

where $D_k = X_k^{-\frac{1}{2}} Z_k^{\frac{1}{2}}$.

Factorizing the matrix of system (3.6) and then solving twice with the factors accounts for most of the computational effort of each interior-point iteration. Section 3.2

presents the algorithmic and implementation details of performing these operations in parallel for stochastic optimization problems.

The rest of the calculations consists of computing the residuals of (3.6) (mat-vec operations Qx, Ax, and A^Ty), linesearch and iteration updates (vec-vec operations $x = x + \alpha\Delta x$), as well as stopping criteria (vector norms). In PIPS we exploit the special structure of the problem data to distribute both the data and all computations across computational nodes. Data distribution is done by partitioning the scenarios and assigning a partition to each computational node. (Partitions have an equal or close to equal number of scenarios in order to avoid load imbalance.) The first-stage data (Q_0, T_0) and variables (x_0, y_0) are replicated across nodes in order to avoid extra communication. Mat-vecs Qx, Ax, and A^Ty can be efficiently parallelized for stochastic optimization. For example, $r = Qx$ can be done without communication since Q is block diagonal, each node computing $r_0 = Q_0r_0$ and $r_i = Q_ix_i$ ($i \geq 1$) for scenarios i that were assigned to it. The mat-vec $r = Ax$ also requires no communication; each node computes $r_0 = T_0x_0$ and $r_i = T_ix_0 + W_ix_i$. For the mat-vec

$$r = A^Ty = \begin{bmatrix} T_0^Ty_0 + \sum_{i=1}^N T_i^Ty_i \\ W_1^Ty_1 \\ \dots \\ W_N^Ty_N \end{bmatrix},$$

each node n computes $r_i = W_i^Ty_i$ and $\tilde{r}_0^n = \sum_i T_i^Ty_i$ for each of its scenarios i and performs an "all-reduce" communication to sum $\tilde{r}_0^n$ across nodes and calculate $r_0 = T_0^Ty_0 + \sum_{i=1}^N T_i^Ty_i$. The vec-vec operations are parallelized in a similar way. The same approach is used in OOPS; we refer the reader to [10] for a detailed description.

3.2. Linear algebra overview. In PIPS-IPM we exploit the arrow-shaped structure of the optimization problem (1.1) to produce highly parallelizable linear algebra. The linear system that is solved at each IPM iteration can be reordered in the primal-dual angular form

$$K := \begin{bmatrix} K_1 & & & B_1 \\ & \ddots & & \vdots \\ & & K_N & B_N \\ B_1^T & \dots & B_N^T & K_0 \end{bmatrix}, \tag{3.7}$$

where $K_0 = \left[\begin{smallmatrix} Q_0+D_0 & T_0^T \\ T_0 & 0 \end{smallmatrix}\right]$, $K_i = \left[\begin{smallmatrix} Q_i+D_i & W_i^T \\ W_i & 0 \end{smallmatrix}\right]$, and $B_i = \left[\begin{smallmatrix} 0 & 0 \\ T_i & 0 \end{smallmatrix}\right]$, $i = 1, \dots, N$. D_0, D_1, $\dots, D_N$ are diagonal matrices with positive entries arising from the use of interior-point methods and change at each IPM iteration.

It is well known in the linear algebra community that primal-dual angular linear systems of the form (3.7) can be parallelized by using a Schur complement technique. We follow the same approach in PIPS-IPM and present it here for completeness. By performing a block Gaussian elimination of the bordering blocks, one can solve the linear system (3.7) by

1. computing the Schur complement

$$C = K_0 - \sum_{i=1}^N B_i^T K_i^{-1} B_i; \tag{3.8}$$

2. solving Schur linear system

$$C\Delta z_0 = r_0 - \sum_{i=1}^{N} B_i^T K_i^{-1} r_i; \tag{3.9}$$

3. solving second-stage linear systems ($i = 1, \dots, N$)

$$K_i \Delta z_i = B_i \Delta z_0 - r_i. \tag{3.10}$$

Most of the computations—in particular, obtaining the scenarios contributions $B_i^T K_i^{-1} B_i$ to the Schur complement, computing the residual in step II, and solving for Δz_i—can be performed independently, yielding an efficient parallelization scheme. However, solving with the dense Schur complement C in step II may be a parallelization bottleneck for problems having a large number of first-stage variables. The bottleneck can be overcome by distributing C and solving the Schur complement linear systems in parallel [20]. By default, PIPS-IPM uses LAPACK and multithreaded BLAS to factorize and solve with C on each node.

Algorithm 1 lists the parallel procedure we use in PIPS-IPM to solve $K\Delta z = r$ using the Schur complement-based decomposition scheme (3.8)–(3.10). The verb "reduce" refers to the communication operation that combines data held by different processes through an associative operator, in our case summation, and accumulates the result on a single process (*reduce*) or on all processes (*all-reduce*). Message passing interface (MPI) routines *MPI_Reduce* and *MPI_Allreduce* correspond to these operations. A different communication strategy is also available in PIPS-IPM. Instead of all-reducing C in step 3 and replicating the factorization in step 4 on all nodes, one could only reduce C to process 1 and perform the factorization on process 1 only. Consequently, in step 6, v_0 is reduced only to process 1, which then performs step 7 and broadcasts (*MPI_Bcast*) v_0 to the rest of the processes. In theory this communication pattern should be faster than the one of Algorithm 1, especially since the most expensive communication, all-reducing C, is avoided. However, it is slower on "Intrepid" BG/P because all-reduce is about two times faster than reduce. (This anomaly is likely to be caused by an implementation problem of BG/P's *MPI_Reduce* or lack of optimization.)

The computation of the Schur complement (step 2 of Algorithm 1) was by far the most expensive operation, and it was traditionally done in PIPS-IPM [19] by solving with the factors of K_i for each nonzero column of B_i and multiplying the result from the left with B_i^T. A slightly different approach is to apply a primal-dual regularization [1] to the optimization problem to obtain quasidefinite matrices K_i that are strongly factorizable to a form of Cholesky-like factorization $K_i = L_i L_i^T$ and to compute $B_i^T K_i^{-1} B_i$ as sparse outer products of $L_i^{-1} B_i^T$. This approach is implemented in OOPS [10]. In both cases, the computational burden is on solving with sparse factors (i.e., triangular solves) of K_i against multiple sparse right-hand sides.

This triangular solve approach is very popular; see [8, 14, 16, 26, 41] as the most recent papers on this parallelization based on incomplete Schur complements. Some of the papers recognize that the triangular solve approach is the limitation in the computation of the Schur complement and focus on overcoming it; for example, in [41] the authors try to exploit sparsity during the triangular solves (with no reference to multithreading), while the authors of [26] are concerned in [39] with the multithreaded performance of the sparse triangular solves.

ALGORITHM 1. SOLVING $K\Delta z = r$ IN PARALLEL BASED ON THE SCHUR COMPLEMENT DECOMPOSITION (3.8)–(3.10).

```
   Given the set $\mathcal{P} = \{1, 2, \dots, P\}$ processes, distribute $N$ scenarios evenly
   across $\mathcal{P}$ and let $\mathcal{N}_p$ be the set of scenarios assigned to process $p \in \mathcal{P}$.
   Each process $p \in \mathcal{P}$ executes the following procedures:
   (factorization phase)
   Function $(L_1, D_1, \dots, L_N, D_N, L_C, D_C)$=Factorize($K$)
1.   Factorize $L_i D_i L_i^T = K_i$ for each $i \in \mathcal{N}_p$.
2.   Compute SC contribution $S_i = B_i^T K_i^{-1} B_i$ for each $i \in \mathcal{N}_p$ and accumulate
     $C_p = -\sum_{i \in \mathcal{N}_p} S_i$. On process 1, let $C_1 = C_1 + K_0$.
3.   All-reduce SC matrix $C = \sum_{r \in \mathcal{P}} C_r$.
4.   Factorize SC matrix $L_c D_c L_c^T = C$;
   (solve phase)
   Function $\Delta z$=Solve($L_1, D_1, \dots, L_N, D_N, L_C, D_C, r$)
5.   Solve $w_i = K_i^{-1} r_i = L_i^{-T} D_i^{-1} L_i^{-1} r_i$ for each $i \in \mathcal{N}_p$ and compute
     $v_p = \sum_{i \in \mathcal{N}_p} B_i^T w_i$.
     On process 1, let $v_1 = v_1 + r_0$.
6.   All-reduce $v_0 = \sum_{i \in \mathcal{N}_p} v_i$.
7.   Solve $\Delta z_0 = C^{-1} v_0 = L_c^{-T} D_c^{-1} L_c^{-1} v_0$.
8.   Solve $\Delta z_i = K_i^{-1}(B_i \Delta z_0 - r_i) = L_i^{-T} D_i^{-1} L_i^{-1}(B_i \Delta z_0 - r_i)$ for each $i \in \mathcal{N}_p$.
```

3.3. Computing $B_i^T K_i^{-1} B_i$ using the augmented approach. In many computational science applications, the numerical factorization phase $K_i = L_i D_i L_i$ of forming the partial Schur-complement contributions $S_i = B_i^T K_i^{-1} B_i$ has generally received the most attention, because it is typically the largest component of execution time; most of the algorithmic improvements [7, 30, 32] in the factorization are related to the exploitation of the sparsity structure in K_i. In PIPS-IPM, however, the solve step $K_i^{-1} B_i$ dominates and is responsible for a much higher proportion of the memory traffic. This makes it a bottleneck in PIPS-IPM on multicore architectures that have a higher ratio of computational power to memory bandwidth.

The multicore architectures that emerged in recent years brought substantial increases in the number of processor cores and their clock rates but only limited increases in the speed and bandwidth of the main memory. For this reason, in many application codes, processor cores spend considerable time in accessing memory, causing a performance bottleneck known as the "memory bandwidth wall." This adverse behavior noindent is likely to be aggravated by the advent of many-core architectures (having hundreds of cores per chip), because it is expected that the speed and bandwidth of the memory will virtually remain unchanged.

In our computational approach, the memory bandwidth wall occurs when solving with the factors of K_i. This is because triangular solves are known to parallelize poorly on multicore machines [13]. In PIPS-IPM, the number of the right-hand sides can be considerably large (for some problems B_i has more than 10,000 columns), and most of the execution time is spent in solving with the factors of K_i, causing an inefficient utilization of the cores for most of the time. Even more important, the right-hand sides B_i are very sparse, a feature that can considerably reduce the number of arithmetic operations. However, exploiting the sparsity of the right-hand sides during the triangular solve increases the memory traffic and exacerbates the memory bandwidth wall.

On the other hand, indefinite sparse factorizations can achieve good speed-ups on multicores machines [31, 13]. We propose an augmented factorization-based technique for computing $B_i^T K_i^{-1} B_i$. Our approach consists of performing a partial factorization of the augmented matrix

$$M_i = \begin{bmatrix} K_i & B_i^T \\ B_i & 0 \end{bmatrix}. \tag{3.11}$$

More specifically, the factorization of the augmented matrix is stopped after the first k_i pivots (k_i being the dimension of K_i). At this point in the factorization process, the lower right block of the factorization contains the exact Schur complement matrix $-B_i^T K_i^{-1} B_i$.

The Schur complement matrix is defined with the help of a block LU factorization

$$A = \begin{pmatrix} K_i & B_i^T \\ B_i & A_{22} \end{pmatrix} := \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}, \tag{3.12}$$

L_{11} lower triangular, U_{11} upper triangular, and

$$L_{22}U_{22} = A_{22} - B_i^T U_{11}^{-1} L_{11}^{-1} B_i = A_{22} - B_i^T K_i^{-1} B_i = S(K_i) := S. \tag{3.13}$$

Hence, halting the factorization before factorizing the bottom-right block $L_{22}U_{22}$ produces the Schur complement matrix S. Equation (3.13) is well known but is hardly exploited in state-of-the art direct solvers packages, whereas it has been used in [17] to construct an incomplete LU approximation of S. Most modern sparse direct linear solvers introduce the solution of blocked sparse right-hand sides to compute explicit Schur complements for applications of interest or replace it by approximations (see, e.g., [26, 27, 28, 41]). The use of these alternative, less efficient approaches, may be explained by the focus on solving linear systems $Ax = b$ in a black-box fashion, whereas computing the Schur complement by (3.13) requires modifying the factorization procedure.

The implicit assumptions used in (3.13) are as follows: A and K_i are nonsingular matrices, and it is sufficient to restrict pivoting to K_i. Furthermore, one can assume that K_i is irreducible, and hence S is dense, because K_i^{-1} is a dense matrix and A is nonsingular; otherwise one should solve each irreducible block of K_i. Within these assumptions a sparse direct factorization can perform the Schur complement computation easily by skipping the pivoting and elimination involving elements of S.

To find an accurate solution during the Gaussian block elimination process we need to find a permutation matrix P that minimizes fill-in and avoids small pivots during the factorization [32]. Two options are available. The first possible permutation P_a can be constructed from A with the constraint that no element of $A_{22} = 0$ is permuted out of the $(2,2)$ block. (But B_i and B_i^T can influence the ordering of K_i.) The alternative option is to find a permutation P_b based on K_i and to extend the permutation with an identity in the second block. Hence, computing a predefined Schur complement introduces a constraint on P anyway. Because we are not aware of good algorithms for computing P_a, we use the second option $P(K_i) = P_b$ as a straightforward solution. The fill-in disadvantage with respect to $P(A)$ can be verified by executing a symbolic factorization phase of A and comparing the size of L and U with that of the Schur complement computation. For the given permutation the sparse factorization does exactly what it should do: it is taking care of the necessary operations with the nonzero elements in an efficient, dense matrix blocked approach.

In our application, the diagonal blocks K_i have saddle-point structure, and hence small perturbed pivots cannot be excluded with the pivoting extensions available in PARDISO. These pivoting strategies are mainly based on graph-matching Bunch–Kaufman pivoting that trades fill-in versus stability to make efficient parallelization possible [32]. While the solution refinement is discussed in the next section in detail, the new Schur complement option can be used to restrict the permutation gradually (permuting selected equations from the zero block to the end) up to preserving the saddle-point structure by specifying A_{22} to be the complete zero-bock.

3.4. Solution refinement. The factorization of M_i is based on a static Bunch–Kaufman pivoting developed in [31] and implemented in the solver PARDISO [32]. A similar approach is available as an option in WSMP [12]. In this case, the coefficient matrix is perturbed whenever numerically acceptable 1×1 and 2×2 pivots cannot be found within a diagonal supernode block. (Checks on potential pivots are made only within the supernode block.) The perturbations are not large in magnitude, having order of the machine precision; however, the number of perturbed pivots can range from $\mathcal{O}(1)$ at the beginning of the optimization to $\mathcal{O}(10^4)$ when the solution is reached. This method has been shown to perform well in many applications. A downside of the pivot perturbation that is not negligible is that solving with K_i might require an increased number of refinement steps to achieve the requested accuracy in the interior-point optimization algorithm [32].

In our case, the pivot perturbations during the factorization of K_i also propagate in $S_i = B_i^T K_i^{-1} B_i$. Absorbing these perturbations through iterative refinement [38] requires solving with K_i for each column of B_i, an operation that we wanted to avoid in the first place. Instead, we let the perturbations propagate through the factorization phase in the Schur complement C. At the end of the factorization phase of Algorithm 1, we have an implicit factorization of a perturbed matrix

$$\tilde{K} := \begin{bmatrix} \tilde{K}_1 & & & B_1 \\ & \ddots & & \vdots \\ & & \tilde{K}_N & B_N \\ B_1^T & \dots & B_N^T & K_0 \end{bmatrix}, \tag{3.14}$$

where $\tilde{K}_i$, $i = 1, \dots, N$, denote the perturbed matrices that were factorized during the (incomplete) factorization of M_i.

It is essential that the solve phase accounts for and absorbs the perturbations; otherwise, the interior-point directions are inaccurate, and the optimization stalls. Our first approach was to perform iterative refinement for $K\Delta z = r$; however, this technique showed a large variability in the number of steps. In comparison, the biconjugate gradient stabilized (BiCGStab) method [36] proved to be more robust, seemingly able to handle a larger number of perturbations more efficiently than iterative refinement. BiCGStab is a Krylov subspace method that solves unsymmetric systems of equations and was designed as a numerically stable variant of the biconjugate gradient method. For symmetric indefinite linear systems such as ours, BiCGStab also outperforms classical conjugate gradient method in terms of numerical stability.

As any other Krylov subspace method, the performance of BiCGStab strongly depends on the use of a preconditioner and on the quality of the preconditioner. Since $\tilde{K}$ is a perturbation of K, the obvious choice for the preconditioner in our case is $\tilde{K}^{-1}$.

Alternatively, one could absorb the errors by performing iterative refinement independently when solving the Schur linear system $C\Delta z_0 = v_0$ (step 7 in Algorithm 1) and the second-stage linear systems (steps 5 and 8 in Algorithm 1). Additionally, the multiplication with the error-free matrix C needed at each refinement iteration when the residual is computed needs to be done based on (3.8) and requires additional second-stage linear solves. We have tested this technique and discarded it because of considerable load imbalance caused by different numbers of iterative refinement iterations in the second-stage solves.

Algorithm 2 lists the preconditioned BiCGStab algorithm. All vector operations (dot-products, addition, and scalar multiplications) are performed in parallel. (No communication occurs since the first-stage part of the vectors is replicated across all processes.) The multiplication $v = Kz$ is also performed in parallel; the only communication required is in forming the vector $v_0 = \sum_{i=1}^{N} B_i^T z_i + K_0 z_0$. The applica-

ALGORITHM 2. SOLVING A LINEAR SYSTEM $Kz = b$ USING PRECONDITIONED BICGSTAB, WITH THE PRECONDITIONER $\tilde{K}$ BEING APPLIED FROM THE RIGHT. IMPLICIT FACTORIZATION OF $\tilde{K}$ IS PROVIDED AS INPUT. HERE z_i DENOTES THE iTH ITERATE OF THE ALGORITHM.

```
Function $z$ = BiCGStabSolve($K, \tilde{L}_1, \tilde{D}_1, \ldots, \tilde{L}_N, \tilde{D}_N, \tilde{L}_C, \tilde{D}_C, b$)
    Compute initial guess $z_0$ =Solve($\tilde{L}_1, \tilde{D}_1, \ldots, \tilde{L}_N, \tilde{D}_N, \tilde{L}_C, \tilde{D}_C, b$).
    Compute initial residual $r_0 = b - Kz_0$.
    Let $\tilde{r}_0$ be an arbitrary vector such that $\tilde{r}_0^T r_0 \neq 0$ (for example $\tilde{r}_0 = r_0$).
    $\rho_0 = \alpha = \omega_0 = 1$; $v_0 = p_0 = 0$.
    For $i = 1, 2, \ldots$
        $\rho_i = \tilde{r}_0^T r_{i-1}$; $\beta = (\rho_i/\rho_{i-1})(\alpha/\omega_{i-1})$.
        $p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1} v_{i-1})$.
        (preconditioner application $y = \tilde{K}^{-1} p_i$)
        $y$ =Solve($\tilde{L}_1, \tilde{D}_1, \ldots, \tilde{L}_N, \tilde{D}_N, \tilde{L}_C, \tilde{D}_C, p_i$).
        (matrix application)
        $v = Ky$.
        $\alpha = \rho_i/(\tilde{r}_0, v_i)$.
        $s = r_{i-1} - \alpha v_i$.
        $z_i = z_{i-1} + \alpha y$.
        If $\|s\|$ is small enough then return $z_i$ if $\|Kz_i - b\|$ is small enough.
        (preconditioner application $w = \tilde{K}^{-1} s$)
        $w$ =Solve($\tilde{L}_1, \tilde{D}_1, \ldots, \tilde{L}_N, \tilde{D}_N, \tilde{L}_C, \tilde{D}_C, s$).
        (matrix application)
        $t = Kw$.
        $\omega_i = (t^T s)/(t^T t)$.
        $z_i = z_i + \omega_i w$.
        If $\|w\|$ is small enough then return $z_i$ if $\|Kz_i - b\|$ is small enough.
        $r_i = s - \omega_i t$.
    end
```

tion of the preconditioner, that is, computing $y = \tilde{K}^{-1}v$, is the most computationally expensive part and is done by calling the *Solve* function listed in Algorithm 1. Factors $\tilde{L}_1$,$\tilde{D}_1, \ldots, \tilde{L}_N$,$\tilde{D}_N$,$\tilde{L}_C$,$\tilde{D}_C$ of $\tilde{K}$ are obtained by calling the *Factorize* function of Algorithm 1 before the BiCGStab iteration starts. The complete computational procedure of solving the linear system with BiCGStab is summarized in Algorithm 3.

ALGORITHM 3. SOLVING $K\Delta z = r$ IN PARALLEL WITH BICGSTAB USING $\tilde{K}$ AS PRECONDITIONER.

```
Given the set $\mathcal{P} = \{1, 2, \ldots, P\}$ processes, distribute $N$ scenarios evenly
across $\mathcal{P}$ and let $\mathcal{N}_p$ be the set of scenarios assigned to process $p \in \mathcal{P}$.
Each process $p \in \mathcal{P}$ executes:
(preconditioner computation)
    $(\tilde{L}_1, \tilde{D}_1, \ldots, \tilde{L}_N, \tilde{D}_N, \tilde{L}_C, \tilde{D}_C)$=Factorize($\tilde{K}$)
(BiCGStab solve phase)
    $\Delta z$ = BiCGStabSolve($K, \tilde{L}_1, \tilde{D}_1, \ldots, \tilde{L}_N, \tilde{D}_N, \tilde{L}_C, \tilde{D}_C, r$).
```

4. Numerical experiments.

In this section we report the parallel performance and efficiency of PIPS-IPM equipped with the PARDISO implementation of the augmented approach.

4.1. Experimental testbed.

Before moving on to the parallel scalability benchmarks of the stochastic optimization application, we briefly describe the target hardware, namely, an IBM BG/P at the Argonne Leadership Computing Facility (ALCF) and a Cray XE6 system installed at the Swiss National Supercomputing Center CSCS.

Large-scale runs were performed on the Intrepid BG/P supercomputer that has 40 racks with a total 40,960 nodes and a high-performance interconnect. Small-scale experiments were performed on Challenger BG/P which consists of only one rack (1024 nodes) and is intended for small test runs. Each BG/P node has a 850 MHz quad-core PowerPC processor and 2 GB of RAM.

The Cray XE6 has 1,496 compute nodes, each of the compute nodes consisting of two 16-core AMD Opteron 6,272 2.1 GHz Interlagos processors, giving 32 cores in total per node with 32 GBytes of memory. In total there are 47,872 compute cores and over 46 Terabytes of memory available on the compute nodes. The Interlagos CPUs implement AMD's recent "Bulldozer" microarchitecture; each Interlagos socket contains two dies, each of which contains four so-called modules.

4.2. Intranode performance.

The first set of experiments investigates the speed-up of the incomplete augmented factorization over the triangular solves approach described in section 3.2 and previously used in PIPS-IPM when computing the scenario contribution $B_i^T K_i^{-1} B$ to the Schur complement. The augmented Schur complement technique described in the previous section was implemented in the PARDISO solver package;[1] it will be refereed as PARDISO-SC, whereas we will use the acronym PARDISO for the approach based on triangular solves.

4.2.1. Artificial test scenarios.

In this section we extend a PDE-constrained quadratic program that has been used in [32] to compare the triangular solves approach with the augmented approach. The artificial scenarios provide a test framework for the solution of stochastic elliptic partial differential equation constrained

[1]See http://www.pardiso-project.org.

Table 1

*Average times in seconds (and Gflop/s in brackets) for different numbers of Cray XE*6 *and BG/P cores inside PARDISO and PARDISO-SC when computing the Schur complement contribution* $B_i^T K_i^{-1} B$ *with* $n = 97{,}336$ *columns in* K_i *and* 2,921 *sparse columns in* B_i*. The artificial test scenarios in* K_i *are similar to the discretization of a Laplacian operator on a* 3*D cube. The ratio column shows the performance acceleration of the triangular solves with multiple right-hand sides in PARDISO versus the incomplete augmented factorization method in PARDISO-SC.*

Number	AMD Interlagos			BG/P		
of cores	PARDISO	PARDISO-SC	Ratio	PARDISO	PARDISO-SC	Ratio
1	454.64	14.43 (5.28)	31.50	3,011.78	101.09 (0.72)	29.79
2	301.03	11.53 (7.19)	26.23	1,506.97	51.23 (1.42)	29.52
4	202.54	6.35 (13.62)	31.89	905.83	26.30 (2.72)	9.66
8	153.14	4.35 (20.38)	35.20	–	–	–
16	92.06	2.84 (31.68)	32.41	–	–	–
32	83.96	1.79 (41.01)	46.90	–	–	–

Table 2

Average times in seconds inside PARDISO and PARDISO-SC when computing the Schur complement contribution $B_i^T K_i^{-1} B$ *with* n_c *columns in* K_i *and* $0.03 \cdot n_c$ *sparse columns in* B_i*. The artificial test scenarios in* K_i *are similar to the discretization of a Laplacian operator on a* 3*D cube. The ratio column shows the performance acceleration of the triangular solves with multiple right-hand sides in PARDISO versus the incomplete augmented factorization method in PARDISO-SC.*

n_c	AMD Interlagos			BG/P		
	PARDISO	PARDISO-SC	Ratio	PARDISO	PARDISO-SC	Ratio
9,261	0.61	0.02	30.51	2.95	0.13	22.63
27,000	8.32	0.28	29.71	35.85	1.48	24.22
97,336	202.54	6.25	32.40	905.83	26.30	34.84
341,061	11,009.46	508.14	21.70	19,615.17	2,431.31	8.05

optimization problems. The matrix K_i in all test scenarios are obtained after a seven-point finite-difference discretization of a three-dimensional (3D) Laplacian operator.

Table 1 compares runtime in seconds for different numbers of Cray XE6 and BG/P cores inside PARDISO and PARDISO-SC when computing the Schur complement contribution $B_i^T K_i^{-1} B$. The matrix K_i used in this experiment has $n = 97{,}336$ columns and rows and is augmented with a matrix B_i with 2,921 columns. The nonzeros in the matrix B_i are randomly generated, with one nonzero element in each column. Table 1 shows strong scaling results for 1 to 32 threads. In fact, the table also shows that the exploitation of sparsity in the augmented approach is highly beneficial and can already accelerate the overall construction of the Schur complement matrix C up to a ratio of 31.5 on one core. It is also demonstrated that the augmented approach results in better scalability on multicores on both the AMD Interlagos and the BG/P cores.

Table 2 demonstrates the impact of the augmented approach for various matrices K_i. The size of the matrices increases from $n_c = 9{,}261$ columns up to $n_c = 341{,}061$ columns. The matrix B_i always has $0.03 \cdot n_c$ nonzero columns. We always used four Interlagos and BG/P cores for these experiments. The timing numbers in the table show that the execution speed always is between a factor of 21.70 and 30.51 depending on the size of K_i on AMD Interlagos.

Table 3 compares the influence of the number of columns in B_i on the performance of both the augmented approach and triangular solves approach. In this case, the number of columns and rows in the matrix K_i is always constant with $n = 97{,}336$, and, again, we always used four Interlagos and BG/P cores. We varied the number of

TABLE 3
Average times in seconds for $k \cdot n$ numbers of sparse columns in B_i inside PARDISO and PARDISO-SC when computing the Schur complement contribution $B_i^T K_i^{-1} B$. The artificial test problem in K_i is similar to the discretization of a Laplacian operator on a 3D cube and always has $n = 97{,}336$ columns. The ratio column shows the performance acceleration of the triangular solves with multiple right-hand sides in PARDISO versus the incomplete augmented factorization method in PARDISO-SC.

k	AMD Interlagos			BG/P		
	PARDISO	PARDISO-SC	Ratio	PARDISO	PARDISO-SC	Ratio
0.03	202.54	6.25	32.40	905.83	26.30	34.80
0.06	392.32	21.85	17.95	1,869.01	100.96	18.51
0.10	667.42	70.22	9.50	2,987.81	280.31	10.66
0.13	952.40	159.90	5.95	3,351.10	556.98	6.02
0.16	1,108.20	233.65	4.75	4,886.56	1,046.43	4.67
0.20	1,334.68	359.32	3.71	5,977.95	1,608.01	3.71

TABLE 4
Computation times of the Schur complement contribution $B_i^T K_i^{-1} B_i$ on BG/P nodes for stochastic problems with 6-, 12-, and 24-hour horizon. The dimensions of K_i and B_i are shown in Table 5. The ratio column shows the performance acceleration of the triangular solves with multiple right-hand sides in PARDISO versus the incomplete augmented factorization method in PARDISO-SC.

Test problem	Cores	$B_i^T K_i^{-1} B$ time (sec) PARDISO	PARDISO-SC	Ratio
UC6	1	109.21	9.31	11.73
	2	58.85	5.58	10.54
	4	32.01	4.09	7.82
UC12	1	481.89	58.41	8.25
	2	250.77	34.36	7.29
	4	136.97	22.50	6.08
UC24	1	1,986.35	273.28	7.26
	2	1,090.01	167.47	6.52
	4	568.78	94.72	5.99

columns in B_i from 2,920 columns ($k = 0.03$) to $19,467$ columns ($k = 0.20$). Although both approaches are mathematically equivalent, the efficiency of the PARDISO-SC implementation is considerable and compelling for problems with large scenarios and smaller Schur complement matrices with up to 20% of columns in B_i.

4.2.2. Stochastic optimization problems. We solved 4-hour, 12-hour, and 24-hour horizon instances (denoted by UC4, UC12, and UC24) of the stochastic UC with 32 scenarios on 32 nodes of Challenger (1 MPI process per node). For PARDISO-SC we ran with 1, 2, and 4 threads and report the intranode scaling efficiency. When using PARDISO, PIPS uses the multiple right-hand side feature of the solver in computing $K_i^{-1}B$. The execution times represent the average (computed over the scenarios and first 10 IPM iterations) time needed for one scenario. We do not list the standard deviation since it is very small (less than 1%). Table 4 summarizes the results.

The shared-memory parallel efficiency of the PARDISO-SC implementation is considerable and compelling for problems with large scenarios (such as UC12 and UC24) that requires one dedicated node per scenario. In addition, the speed-ups over the triangular solves approach show great potential in achieving our goal of considerably reducing the time to solution for problems with a large number of scenarios.

Table 5
Sizes of K_i and B_i for UC instances with 6-, 12-, *and* 24-*hour horizon.*

UC instance	Size K_i	# Nonzero cols. of B_i
UC6	59,478	1,566
UC12	118,956	3,132
UC24	237,912	6,264

Table 6
Solve times in seconds, number of iterations, and average time per iteration needed by PIPS to solve $UC12$ instances with increasingly large numbers of scenarios. One scenario per node was used in all runs.

No. of nodes and scenarios	Solve time in minutes	IPM iterations	Average time per iteration in seconds	Size of K in billions
4,096	59.14	103	33.57	0.487
8,192	64.72	112	34.67	0.974
16,384	70.14	123	34.80	1.949
32,768	79.69	133	35.95	3.898

4.3. Large-scale runs. Following the great intranode speed-up showed by the PARDISO-SC approach, in this section we investigate and report on the large-scale performance of PIPS equipped with PARDISO-SC in solving the UC instances. We look at several performance metrics that are relevant both to the application (such as time-to-solution) and to the parallel efficiency of the implementation (such as scaling efficiencies and sustained FLOPS performance of PIPS).

The BG/P Intrepid experiments solve the UC12 instances with up to 32,768 scenarios on up to 32,768 nodes (131,072 cores). UC24 problems are too big to fit even 1 scenario per node for a large number of scenarios/nodes. All runs of this section were run in "SMP" mode, which means one MPI process per node using four cores. All problems solved terminated with $\mu < 10^{-8}$ and $\|r\| < 10^{-8}$, standard termination criteria for interior-point methods [40].

4.3.1. Time to solution. As we mentioned in section 2, industry practice is to solve UC procedures under strict "real-time" requirements, which in the case of the UC models mean solving the problems under 1 hour (size of the time horizon step). To test the real-time capabilities of PIPS with PARDISO-SC, we solved UC12 instances with an increasingly larger number of scenarios and nodes (one scenario per node). The total execution times shown in Table 6 are within the requirements. Moreover, the speed-up over the triangular solves approach is substantial. With the triangular solves approach and using the MA57 as linear solver, PIPS needed 4 hours 3 minutes to solve a $UC4$ instance with 8,192 scenarios and 8,192 nodes on the same architecture. With the augmented factorization implementation from PARDISO-SC, a $UC12$ instance with the same number of scenarios can be solved in a little more than 1 hour using the same number of nodes.

Additional reduction in the total execution time can be obtained by reusing solution information from the UC instance solved previously, a process known as warm-starting. Reduction of 30–40% in the iteration count has been reported for interior-point methods applied to stochastic programming [10]. On the other hand, finding a binary integer solution requires additional effort, as described in section 2. While this is future research, we expect the cost of this phase to be low and not to offset the gains obtained by warm-starting.

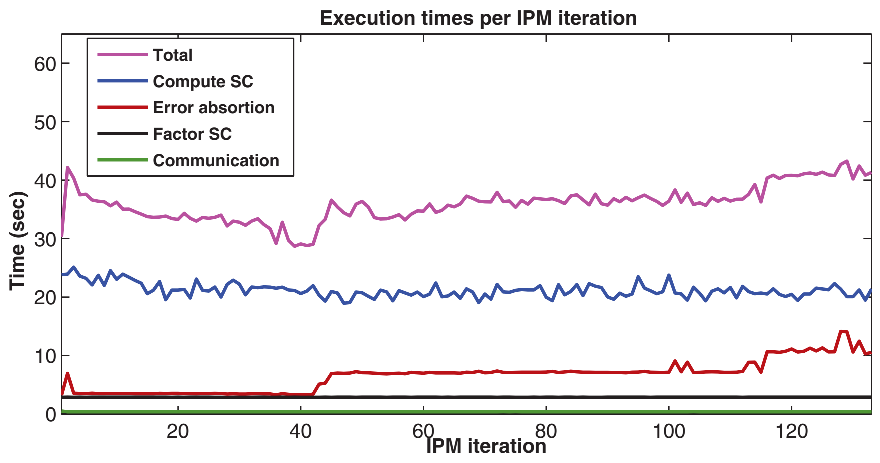

FIG. 3. *Breakdown of the execution time per interior-point iteration for UC12 with 32K scenarios on 32K nodes. The total execution time and its three most expensive components are shown. "Compute SC" represents average times of computing $B_i^T K_i^{-1} B_i$, "Error absorbtion" shows the cost of BiCGStab procedure, and "Communication" indicates the communication time.*

The solve time increases with the number of scenarios because the optimization problems are larger and require a larger iteration count. However, we observe that even though the size of the problem increases by 8 times from 4K to 32K scenarios, the iteration count shown in Table 6 increases by less than 30% and shows very good performance of the Mehrotra's predictor-corrector despite the extreme size of the problems.

The average cost of an interior-point iteration increases by only 7% from 4K scenarios to 32K scenarios, mainly because of a small increase in the number of BiCGStab iterations (more pivot perturbation errors are present and need to be absorbed as the scenario count increases), and not to communication or load imbalance overhead.

4.3.2. Breakdown of the execution time. In Figure 3 we show the execution time and the three most expensive components of the execution time for each interior-point iteration for our largest simulation (UC12 with 32,768 scenarios on 32,768 nodes).

The Compute SC displays the time needed by PARDISO-SC to compute the Schur complement contribution $B_i^T K_i^{-1} B_i$, averaged across all scenarios. Error absorbtion indicates the time spent in the BiCGStab procedure that absorbs the errors in K that occurred because of the pivot perturbations in PARDISO-SC. Factor SC represents the time spent in the dense factorization of $\tilde{C}$. Communication depicts the total time spent in internode communication in both the Compute-SC and Error absorbtion computational steps.

As can be seen in Figure 3, the overhead of the error absorbtion steps is fairly low (ranges from 10% to 40% of the Compute SC step) and plays a crucial role in the speed-ups in the time to solution of the PARDISO-SC approach over the triangular solves approach previously implemented in PIPS [19].

The cost increase of the error absorbtion phase with the interior-point iterations is due to an increase in the number of BiCGStab iterations. This behavior is most likely caused by an increasing ill-conditioning of the linear systems K_i (a well-known behavior of interior-point methods) that amplifies the pivot perturbation errors and, consequently, decreases the quality of the preconditioner $\tilde{C}$. The average number of

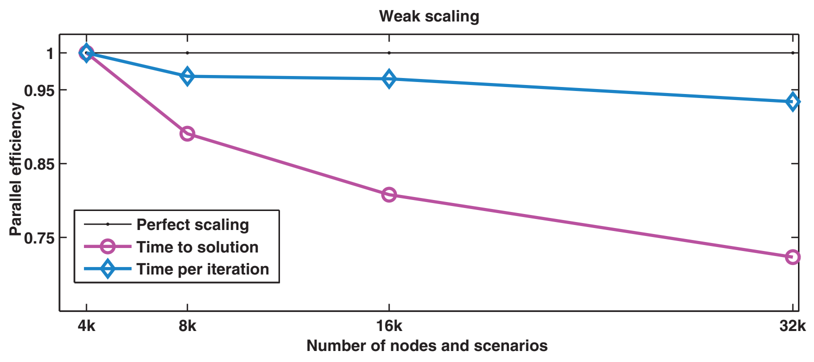

FIG. 4. *"Weak scaling" efficiency plot showing the parallel efficiency of PIPS in solving UC*12 *instances on Intrepid. The closer to the Perfect scaling horizonal line, the better, meaning that there is less parallel efficiency loss.*

BiCGStab iterations ranges from 0 in the incipient phases of the optimization to 1.5 near the solution.

The very low cost of the communication ($<$ 0.4 seconds) is the effect of the very fast BG/P communication collectives routines. Other parallel computing systems may show different behaviors, depending on the speed on the network. However, the next-generation supercomputers (such as BG/Q) will operate even faster networks, and so it is very unlikely that the communication will become a bottleneck on this architecture.

4.3.3. Parallel efficiency.

Scalability studies of this section are aimed at providing a lightweight measurement of the efficiency of PIPS-IPM and an extrapolation basis for the performance of PIPS-IPM for larger problems.

We first present the so-called weak-scaling efficiency study. It consists of solving increasingly large problems with additional computational resources. In our study we used UC12 instances, and we increased the number of scenarios and nodes at the same rate (their ratio is 1). Linear scaling or perfect efficiency is achieved if the run time stays constant. The efficiency of PIPS-IPM is shown in Figure 4. We have used the same UC12 instances and run times of Table 6. The efficiency of the entire optimization process (excluding problem loading) is about 74%, and the efficiency of the parallel linear algebra is more than 94% percent. In our opinion these are very good parallel efficiencies, in part due to the fast BG/P collectives (relative to the speed of the cores) and in part to a judicious use of MPI and numerical linear algebra libraries in our implementation. The reduced efficiency of the overall optimization process is caused mostly by the increase in the number of optimization iterations, as we pointed out in section 4.3.1.

The second efficiency study fixes the size of the problem and increases the number of computational nodes. This is known as a strong-scaling study. An application scales linearly if the run time decreases at the same rate that the number of nodes increases. In our experiments we solved the UC12 instance with 32K scenarios on 8K, 16K, and 32K nodes. (The run on $4K$ nodes ran out of memory.) The strong scaling efficiency displayed in Figure 5 is over 75% from 8K to 32K nodes. The slight deterioration in the efficiency is caused mainly by the dense factorization of the Schur complement and solves with its factors, which is replicated across all computational nodes. Reducing the cost of these dense linear algebra calculations (for example, by

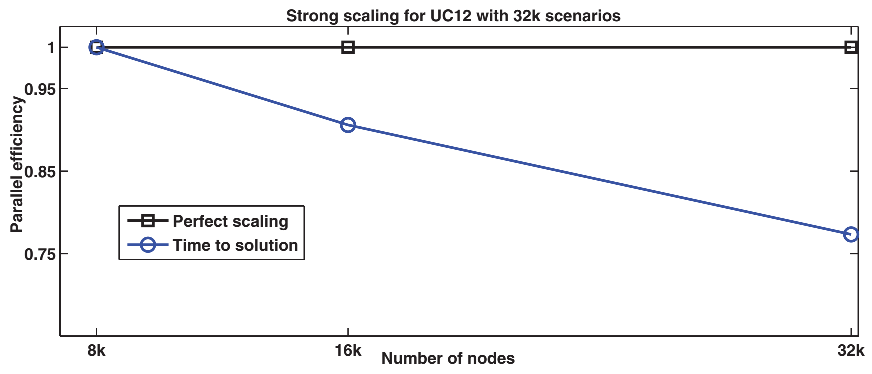

FIG. 5. *"Strong scaling" efficiency plot showing the parallel efficiency of PIPS in solving UC*12 *instance with* 32*k scenarios on* 8*k,* 16*k, and* 32*k nodes of Intrepid BG/P.*

using GPU acceleration) would greatly improve the strong-scaling efficiency of our code.

We also report the performance of PIPS-IPM in terms of sustained floating-point operations per second (Flops). For this experiment we solved UC12 with 4K scenarios on 4K nodes (1 BG/P rack) and used the Hardware Performance Monitor (HPM) library to count the number of floating-point operations. The sustained Flops rate of PIPS-IPM on 1 rack was 3.37 TFlops, which accounts for 6.05% of theoretical FLOPS peak rate. Inside our code, the two largest Flops rates were attained by the PARDISO augmented factorization, 4.21 TFlops (7.55% of peak), and LAPACK symmetric indefinite factorization routine DSYTRF, 14.75 TFlops (26.48% of peak). With LAPACK we used ESSL Blue Gene SMP BLAS provided by the computer manufacturer (IBM). We caution that sustained Flops performance may not be the best measure of efficiency in our case because a large proportion of computations is dedicated to sparse linear algebra, which by definition is difficult to vectorize and requires additional integer arithmetic (fixed-point operations) that are not counted by HPM. However, this work improved the Flops rate of PIPS-IPM by a factor of 6 over the previously used triangular solves technique; the improvement comes from the use of the incomplete augmented factorization, which is considerably less memory bound than triangular solves on multicores chips.

5. Conclusions.

This paper presents a novel technique for the computation of Schur complement matrices that occur in decomposition schemes for the solution of optimization problems with dual-block angular structure. This class of structured optimization problems includes stochastic optimizations problems with recourse, an important class of problems for addressing the difficult issue of integrating renewable sources of energy into the power grid. We present and discuss a stochastic model for the real-time UC problem.

Based on an incomplete sparse factorization of an augmented system that is implemented in PARDISO, the proposed approach is capable of using multicore nodes efficiently, as well as fully exploiting sparsity. The pivot perturbation errors are not managed for each scenario independently; instead, we use preconditioned BiCGStab to absorb the errors at once for all scenarios. This approach maintains good load balancing even when tens of thousands of nodes are simultaneously involved in computations. The implementation of the augmented approach in the PIPS optimization solver showed substantial speed-up over the previous approach. PIPS solved realistically sized instances (12-hour horizon and state of Illinois power grid) of the stochastic UC with thousands of scenarios in about 1 hour on the Intrepid BG/P supercomputer.

Our future work will attempt to solve larger horizon problems on modern architectures such as IBM BG/Q, Cray XK7, and Cray XC30. For example, preliminary runs on a Cray XC30 architecture indicate that 24-hour stochastic UC relaxations can be solved in less than 40 minutes. This time can be further reduced by 30–40% by warm-starting the relaxation with the solution of stochastic UC instance solved previously. Finding *integer* solutions to this class of problems—better than those obtained by rounding—requires additional algorithmic and implementation developments as outlined in section 2. While this remains future research, by the tightness of the relaxation, we expect to be able to find high-quality integer solutions by solving a small number of warm-started relaxations; hence, the linear-algebra improvements in this work have reduced solution times for the 24-hour stochastic UC problem to near-practical levels.

Acknowledgments. We acknowledge Jeff Hammond for his guidance in BG/P implementation and informative discussions. We also acknowledge Edward Rothberg of Gurobi for an early discussion on the use of the augmented approach. K. Gärtner also acknowledges King Abdullah University of Science and Technology (KAUST) for providing access to a BG/P platform.

REFERENCES

[1] A. Altman and J. Gondzio, *Regularized symmetric indefinite systems in interior-point methods for linear and quadratic optimization*, Optim. Methods Softw., 11 (1999), pp. 275–302.

[2] M. Carrión and J. Arroyo, *A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem*, Power Systems, IEEE Trans. 21 (2006), pp. 1371–1378.

[3] M. Carrión, A. Philpott, A. Conejo, and J. Arroyo, *A stochastic programming approach to electric energy procurement for large consumers*, IEEE Trans. Power Systems, 22 (2007), pp. 744–754.

[4] C. Coffrin, P. Van Hentenryck, and R. Bent, *Accurate load and generation scheduling for linearized DC models with contingencies*, in Proceedings of the Power and Energy Society General Meeting, IEEE, 2012, pp. 1–8.

[5] E. Constantinescu, V. Zavala, M. Rocklin, S. Lee, and M. Anitescu, *A computational framework for uncertainty quantification and stochastic optimization in unit commitment with wind power generation*, IEEE Trans. Power Systems, 26 (2011), pp. 431–441.

[6] J. Czyzyk, S. Mehrotra, and S. J. Wright, *PCx User Guide*, Technical report OTC 96/01, Optimization Technology Center, Argonne National Laboratory and Northwestern University, 1996.

[7] T. Davis, *Direct Methods for Sparse Linear Systems*, Vol. 2, SIAM, Philadelphia, 2006.

[8] J. Gaidamour and P. Henon, *A parallel direct/iterative solver based on a Schur complement approach*, in Proceedings of the 11th IEEE International Conference on Computational Science and Engineering, 2008, pp. 98–105.

[9] E. M. Gertz and S. J. Wright, *Object-oriented software for quadratic programming*, ACM Trans. Math. Software, 29 (2003), pp. 58–81.

[10] J. Gondzio and A. Grothey, *Exploiting structure in parallel implementation of interior point methods for optimization*, Comput. Manag. Sci., 6 (2009), pp. 135–160.

[11] J. Gondzio, *HOPDM (version* 2.12*)—A fast LP solver based on a primal-dual interior point method*, European J. Oper. Res., 85 (1995), pp. 221–225.

[12] A. Gupta, *WSMP: Watson Sparse Matrix Package*, Technical report, IBM Research Report, 2000.

[13] J. Hogg and J. Scott, *A Note on the Solve Phase of a Multicore Solver*, Technical report RAL-TR-2010-007, SFTC Rutherford Appleton Laboratory, 2010.

[14] G. Larrazabal and J. M. Cela, *A parallel algebraic preconditioner for the Schur complement system*, in Proceedings of the 15th International Symposium on Parallel and Distributed Processing, 2001, pp. 1205–1210.

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

[16] X. S. Li, M. Shao, I. Yamazaki, and E. G. Ng, *Factorization-based sparse solvers and preconditioners*, J. Phys. Conf. Ser., 180 (2009), 012015.

[17] Z. Li and Y. Saad, *SchurRAS: A restricted version of the overlapping Schur complement preconditioner*, SIAM J. Sci. Comput., 27 (2006), pp. 1787–1801.

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

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

[20] M. Lubin, C. G. Petra, and M. Anitescu, *The parallel solution of dense saddle-point linear systems arising in stochastic programming*, Optim. Methods Softw., 27 (2012), pp. 845–864.

[21] I. J. Lustig, R. E. Marsten, and D. F. Shanno, *On implementing Mehrotra's predictor–corrector interior-point method for linear programming*, SIAM J. Optim., 2 (1992), pp. 435–449.

[22] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM J. Optim., 2 (1992), pp. 575–601.

[23] T. Overbye, X. Cheng, and Y. Sun, *A comparison of the AC and DC power flow models for LMP calculations*, in Proceedings of the 37th Annual Hawaii International Conference on System Sciences, 2004, pp. 9.

[24] C. Petra and M. Anitescu, *A preconditioning technique for Schur complement systems arising in stochastic optimization*, Comput. Optim. Appl., 52 (2012), pp. 315–344.

[25] G. Pritchard, G. Zakeri, and A. Philpott, *A single-settlement, energy-only electric power market for unpredictable and intermittent participants*, Oper. Res., 58 (2010), pp. 1210–1219.

[26] S. Rajamanickam, E. Boman, and M. Heroux, *ShyLU: A hybrid-hybrid solver for multicore platforms*, in Proceedings of the 26th IEEE International Parallel Distributed Processing Symposium (IPDPS), 2012, pp. 631–643.

[27] Y. Saad and M. Sosonkina, *Distributed Schur complement techniques for general sparse linear systems*, SIAM J. Sci. Comput., 21 (1999), pp. 1337–1356.

[28] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numer. Linear Algebra Appl., 9 (2002), pp. 359–378.

[29] Y. Saad and J. Zhang, *BILUTM: A domain-based multilevel block ILUT preconditioner for general sparse matrices*, SIAM J. Matrix Anal. Appl., 21 (1999), pp. 279–299.

[30] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Future Generation Computer Systems, 20 (2004), pp. 475–487.

[31] O. Schenk and K. Gartner, *On fast factorization pivoting methods for symmetric indefinite systems*, Electron. Trans. Numer. Anal., 23 (2006), pp. 158–179.

[32] O. Schenk, A. Wächter, and M. Hagemann, *Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization*, Comput. Optim. Appl., 36 (2007), pp. 321–341.

[33] M. Shahidehpour, H. Yamin, and Z. Li, *Market Operations in Electric Power Systems: Forecasting, Scheduling, and Risk Management*, Wiley, New York, 2002.

[34] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory*, MPS/SIAM Ser. Optim. 9, SIAM, Philadelphia, 2009.

[35] D. Streiffert, R. Philbrick, and A. Ott, *A mixed integer programming solution for market clearing and reliability analysis*, in Proceedings of the Power Engineering Society General Meeting, IEEE, 2005, pp. 2724–2731.

[36] H. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.

[37] H. Vladimirou and S. Zenios, *Scalable parallel computations for large-scale stochastic programming*, Ann. Oper. Res., 90 (1999), pp. 87–129.

[38] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Notes on Applied Science 32, HMSO, London, UK, 1963.

[39] M. M. Wolf, M. A. Heroux, and E. G. Boman, *Factors impacting performance of multithreaded sparse triangular solve*, in Proceedings of the 9th International Conference on High Performance Computing for Computational Science—VECPAR 2010, Berlin, Springer-Verlag, 2011, pp. 32–44.

[40] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, 1997.

[41] I. Yamazaki and X. S. Li, *On techniques to improve robustness and scalability of a parallel hybrid linear solver*, in Proceedings of the Conference on High Performance Computing for Computational Science—VECPAR 2010, Lecture Notes in Comput. Sci. 6449, Springer-Verlag, Berlin, 2011, pp. 421–434.

[42] D. Zhang and Y. Zhang, *A Mehrotra-type predictor-corrector algorithm with polynomiality and Q-subquadratic convergence*, Ann. Oper. Res., 62 (1996), pp. 131–150.

[43] Y. Zhang, *Solving large-scale linear programs by interior-point methods under the MATLAB environment*, Technical report TR96-01, University of Maryland Baltimore County, Baltimore, MD, 1996.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.