Preprint ANL/MCS-P1858-0311

Scalable Stochastic Optimization of Complex Energy Systems

Miles Lubin, Cosmin G. Petra, Mihai Anitescu, and Victor Zavala
Argonne National Laboratory
9700 S. Cass Avenue
Argonne, IL 60439, USA
{mlubin, petra, anitescu, vzavala}@mcs.anl.gov

ABSTRACT

We present a scalable approach and implementation for solving stochastic programming problems, with application to the optimization of complex energy systems under uncertainty. Stochastic programming is used to make decisions in the present while incorporating a model of uncertainty about future events (scenarios). These problems present serious computational difficulties as the number of scenarios becomes large and the complexity of the system and planning horizons increase, necessitating the use of parallel computing. Our novel hybrid parallel implementation PIPS is based on interior-point methods and uses a Schur complement technique to obtain a scenario-based decomposition of the linear algebra. PIPS is applied to a stochastic economic dispatch problem that uses hourly wind forecasts and a detailed physical power flow model. Solving this problem is necessary for efficient integration of wind power with the Illinois power grid and real-time energy market. Strong scaling efficiency of 96% is obtained on 32 racks (131,072 cores) of the "Intrepid" Blue Gene/P system at Argonne National Laboratory.

1. INTRODUCTION

Stochastic programming is one of the leading paradigms for decision-making under uncertainty [3]. In this work, we consider two-stage, convex, quadratic stochastic sample average approximation problems of the form

$$\begin{array}{lllllll}
\displaystyle\min_{x_i, i=0,\ldots,N} & \multicolumn{6}{l}{\left(\frac{1}{2}x_0^TQ_0x_0 + c_0^Tx_0\right) + \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}x_i^TQ_ix_i + c_i^Tx_i\right)} \\
\text{s.t.} & T_0x_0 & & & & & = b_0, \\
 & T_1x_0 + & W_1x_1 & & & & = b_1, \\
 & T_2x_0 + & & W_2x_2 & & & = b_2, \\
 & \vdots & & & \ddots & & \vdots \\
 & T_Nx_0 + & & & & W_Nx_N & = b_N, \\
 & x_0 \geq 0, & x_1 \geq 0 & , x_2 \geq 0, & \ldots & x_N \geq 0. &
\end{array} \qquad (1)$$

Here $Q_0 \in \mathbb{R}^{n_0 \times n_0}$ and $Q_i \in \mathbb{R}^{n_1 \times n_1}$ are symmetric positive semidefinite matrices; $T_0 \in \mathbb{R}^{m_0 \times n_0}$, $T_i \in \mathbb{R}^{m_1 \times n_0}$ are full row rank matrices known as technology matrices; $W_i \in \mathbb{R}^{m_1 \times n_1}$ are full row rank matrices known as recourse matrices; and $c_0 \in \mathbb{R}^{n_0}$, $c_i \in \mathbb{R}^{n_1}$, $b_0 \in \mathbb{R}^{m_0}$, $c_i \in \mathbb{R}^{m_1}$ $(i = \{1, \ldots, N\})$. The data indexed by zero are related to the so-called first-stage (sub)problem. Each $(Q_i, c_i, T_i, W_i, b_i)$ is called a sample or a scenario and is one of the N possible realizations of the uncertainty associated with the second stage (sub)problems.

Problems of form (1) are deterministic equivalents of two-stage convex quadratic stochastic problems [17] with equiprobable and finitely many realizations (or scenarios) of the underlying randomness. When the randomness is described by an infinite or prohibitively large number of realizations, the sample average approximation (SAA) method [22] is used to obtain an approximate solution to the original problem by solving a deterministic approximation problem of form (1).

Interior-point methods (IPMs) have been used as early as 1988 to decompose and solve SAA problems [4]. The SAA problems are usually extremely large; and even though the data are usually sparse, such problems can be solved only with distributed or parallel computing. The decomposition of the problem in the context of IPMs is achieved at the linear algebra level by taking advantage of the block-separable form of the objective function and the half-arrow shape of the Jacobian. The main computational effort of a interior-point method consists of solving a symmetric indefinite linear system, known as a saddle-point linear system, at each iteration. The special structure of SAA problems (1) allows the matrix of the IPM saddle-point linear system to be permuted to

$$\begin{bmatrix}
\bar{Q}_1 & W_1^T & & & & 0 & 0 \\
W_1 & 0 & & & & T_1 & 0 \\
 & & \ddots & & & \vdots & \vdots \\
 & & & \bar{Q}_N & W_N^T & 0 & 0 \\
 & & & W_N & 0 & T_N & 0 \\
0 & T_1^T & \ldots & 0 & T_N^T & \bar{Q}_0 & T_0^T \\
0 & 0 & \ldots & 0 & 0 & T_0 & 0
\end{bmatrix}. \qquad (2)$$

Each matrix $\bar{Q}_i$ is the sum of $\frac{1}{N}Q_i$ and a diagonal matrix with positive diagonal entries coming from the use of interior-point methods. Linear systems having matrices of the above form are known as symmetric bordered block-diagonal or arrow-shaped linear systems. Solving such linear systems by using the Schur complement of the lower right 2-by-2 first-stage saddle-point block allows most of the computations to be performed independently for each scenario. The only work that cannot be done independently for each scenario consists of linear solves with the dense or almost dense Schur complement matrix. The Schur complement decomposition to parallelize IPMs has been implemented in state-of-the-art software packages such as OOPS [10, 11, 12, 13] and IPOPT [26]. In particular, in [10] a six-stage problem having a total of approximately 1 billion variables and 16 million scenarios has been solved. We also note the approach in [15], which uses asynchronous versions of L-shaped methods and a trust-region method to solve two-stage problems with up to 125 million variables and up to 250,000 scenarios on a heterogeneous computational grid.

The present work includes several novel techniques motivated by the specific characteristics of the stochastic optimization problems arising in the optimization of complex energy systems. Such problems have large dimensions in both the first-stage and second-stage subproblems, because they incorporate complex physics such as network constraints and ramp constraints.

The large number of variables in the first-stage causes a memory bottleneck when the dense Schur complement matrix cannot fit entirely in the local memory of a computational node. Distributing the Schur complement matrix (and linear solves with it) across nodes is the only feasible alternative; however, assembling the distributed matrix is nontrivial because all nodes contribute to all elements of the matrix, necessitating a large amount of interprocess communication. Our streamlined assembly procedure is presented in detail in Section 4.3.

When the dimension of the second-stage data increases, sometimes only one or two scenarios can fit on one computational node with four or more cores, whereas our decomposition requires a minimum of one scenario per MPI process. In this case, using one MPI process per core is not possible because of the memory limitations. Therefore, we implement a hybrid programming approach, SMP inside MPI, so that all cores in each node are fully used.

PIPS, our solver, has important applications in diverse optimization tasks arising in electricity markets and operations and planning. In particular, a current challenge faced by smart grid initiatives is effectively mitigating contingencies (line failures) and demand and wind supply uncertainty. A real example of demand and wind power variability is presented in Figure 1. Today this mitigation is done by using conservative reserve margins and nonsystematic procedures. This approach significantly increases prices and nationwide emission levels. Motivated by this situation, researchers are exploring stochastic programming as an enabling technology to mitigate uncertainty. Because of the size of the networks and geographical regions and the number of possible contingencies, the central power operators (ISOs) already rely on high-performance computing. Consequently, a massively parallel stochastic programming solution has practical appeal. In this work we study the particular problem of economic dispatch that balances supply and demand to clear the market in real time over a wide geographical region.

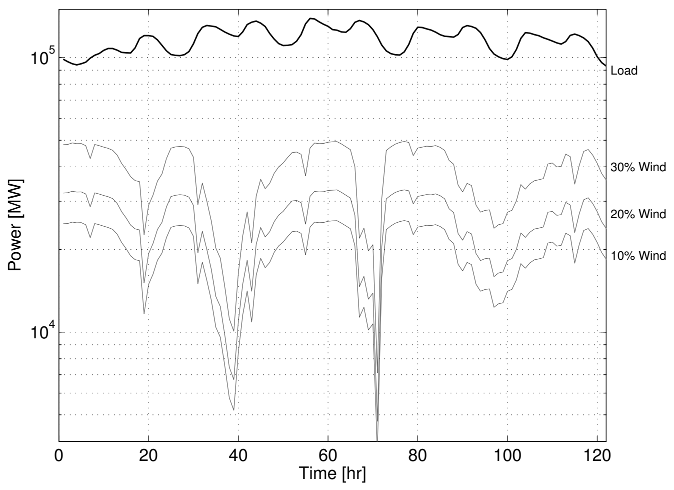

Figure 1: Snapshot of total load and wind supply variability at different adoption levels.

In our numerical tests, we obtain very good (96%) strong scaling performance and solve a large-scale problem to completion. In the previous works mentioned, the scenarios had no more than 500 variables per stage, whereas in our case we solve a problem with 4,700 variables in the first stage and 14,000 variables in the second stage scenarios. The largest linear system we solve directly is of order $\sim$1.4 billion. We will also note the further developments needed to solve even larger problems with important applications that have over 100,000 variables in the second stage.

2. STOCHASTIC PROGRAMMING FOR ENERGY MARKETS AND OPERATIONS

Fast and uncertain fluctuations of supply and demand cannot be effectively managed in the current power grid. In Figure 1 we illustrate the magnitude and frequency of wind supply fluctuations under different adoption levels compared with a typical total load profile in Illinois. Since electricity cannot be currently stored at a large scale, the grid is operated by using conservative and expensive reserve levels to mitigate operational risk associated with these fluctuations and some other uncertain factors such as generator and transmission line failures. Expanding the geographical domain, time resolution, and forecast horizon is critical to enhance proactiveness, responsiveness, and market efficiency under the more volatile environments expected in the next-generation grid resulting from high levels of renewable supply and more elastic demands.

Stochastic programming can be used to effectively mitigate uncertainty and reduce reserves in next-generation grid environments. The nationwide economic and carbon footprint reduction potentials can be substantial; however, advances in algorithms and computing platforms are necessary to reach these goals. An important application of stochastic programming is time-dependent market operational tasks such as unit commitment and economic dispatch [21]. These tasks set the prices in day-ahead and real-time markets, respectively. In practice, the day-ahead market uses hourly forecasts of demand and renewable supply to compute the optimal on/off schedule of the generators and generation levels that match the demands across a given geographical region and satisfy physical Kirchoff's laws and transmission generation ramp constraints. The real-time market corrects the generation levels at a higher frequency (5 minutes) to account for forecast errors in the day-ahead market. A poor forecast and management of uncertainty in the day-ahead market can lead to high real-time prices (twice the magnitude of day-ahead prices) as a result of insufficient ramping and transmission capacity. Poor management of uncertainty in the real-time market can lead to dynamic instability. This is particularly critical during peaking conditions as those observed in the 2003 blackout in the Eastern interconnect and during summer conditions in regions such as California.

In our analysis, we consider a two-stage stochastic programming formulation for economic dispatch. The problem has the following structure [25]:

$$\min \left(\sum_{j\in\mathcal{G}} c_j \cdot G_{0,\ell,j}\right) + \frac{1}{N}\sum_{s\in\mathcal{N}}\left(\sum_{k=\ell+1}^{\ell+T}\sum_{j\in\mathcal{G}} c_j \cdot G_{s,k,j}\right) \tag{3a}$$

$$\text{s.t. } G_{s,k+1,j} = G_{s,k,j} + \Delta G_{s,k,j},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{G} \tag{3b}$$

$$\sum_{(i,j)\in\mathcal{L}_j} P_{s,k,i,j} + \sum_{i\in\mathcal{G}_j} G_{s,k,i} = \sum_{i\in\mathcal{D}_j} D_{s,k,i} - \sum_{i\in\mathcal{W}_j} W_{s,k,i},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{B} \quad (p_{s,k,j}) \tag{3c}$$

$$P_{s,k,i,j} = b_{i,j}(\theta_{s,k,i} - \theta_{s,k,j}),\ s\in\mathcal{N}, k\in\mathcal{T}, (i,j)\in\mathcal{L} \tag{3d}$$

$$0 \le G_{s,k,j} \le G_j^{max},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{G} \tag{3e}$$

$$|\Delta G_{s,k,j}| \le \Delta G_j^{max},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{G} \tag{3f}$$

$$|P_{s,k,i,j}| \le P_{i,j}^{max},\ s\in\mathcal{N}, k\in\mathcal{T}, (i,j)\in\mathcal{L} \tag{3g}$$

$$|\theta_{s,k,j}| \le \theta_j^{max},\ s\in\mathcal{N}, k\in\mathcal{T}, j\in\mathcal{B} \tag{3h}$$

$$G_{s,\ell,j} = G_{0,\ell,j},\ s\in\mathcal{N}, j\in\mathcal{G}. \tag{3i}$$

Here, $\mathcal{G}, \mathcal{L}, \mathcal{B}$ are the sets of generators, lines, and buses in the network in the geographical region, respectively. $\mathcal{D}_j, \mathcal{W}_j$ are the sets of demand and wind supply nodes connected to bus j, respectively. Symbol $\mathcal{N}$ denotes the set of uncertain scenarios for the wind and demands over the time horizon $\mathcal{T} := \{\ell, ..., \ell+T\}$ starting at the current instant ℓ and where T is the horizon length. Variables $G_{s,k,j}$ are the generator supply levels for scenario s, time instant k, and bus j. Following a similar notation, $P_{s,k,j}$ are the transmission line power flows, $\theta_{s,k,j}$ are the bus angles, $W_{s,k,i}$ are the wind supply flows, and $D_{s,k,i}$ are the demand levels. Constraint (3c) is Kirchoff's law, which holds at each scenario, time, and bus and which implicitly contains the structure of the network. The multipliers of Kirchoff's law are the locational marginal prices (LMPs) $p_{s,k,j}$ for each scenario, time instant, and node. Constraints (3f) are the ramp constraints. The objective costs can also be modeled as piecewise linear and quadratic costs of the form $c_j \cdot G_{s,\ell,j} + \frac{1}{2}h_j G_{s,\ell,j}^2$. In most energy applications, the cost function is a positive definite function with block-diagonal Q_i matrices.

We note the presence of the nonanticipativity constraints (3i). These complicating constraints indicate that a unique, implementable signal (scenario-independent) for the generator levels $G_{k,j}$ must be computed by using the current wind supply and demand information at time $k = \ell$ (which are known). This implicitly sets unique LMPs $p_{k,j}$ for the current time $k = \ell$ that are implemented in the market. The nonanticipativity constraints give rise to the first-stage (wait-and-see) variables in the stochastic programming formulation (1). In this formulation, the total number of first-stage variables is given by the number of generators. In our implementation, nonanticipativity constraints are handled implicitly by defining complicating variables x_0 as in formulation (1).

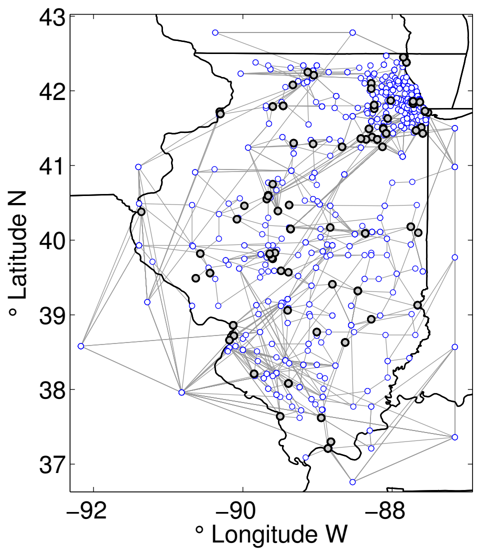

Figure 2: Illinois grid system.

The economic dispatch must run in real time over a receding horizon that recursively updates the wind power supply and demands. Both day-ahead and real-time optimization problems have similar structures; the key difference is that unit commitment makes integer decisions to turn generators on/off under a coarse time resolution, whereas economic dispatch makes mostly continuous decisions under a fine time resolution. At the core of both problems, however, a continuous and highly structured stochastic programming such as (3) is solved. Since these are operational tasks, solution time is critical in these environments.

For an idea of the magnitude and challenges posed by these problems, the Illinois grid contains around 2,000 transmission nodes, 2,500 transmission lines, 900 demand nodes, and 300 generation nodes. Time horizons of 24 to 36 hours are currently used in operations, and these are expected to increase as wind power and smart grid programs are adopted. In Figure 2 we illustrate the complexity of the Illinois grid. A deterministic economic dispatch formulation over this geographical region can have as many as 100,000 variables and inequality constraints. In a stochastic formulation, this number of variables is effectively multiplied by the number of uncertain scenarios. With thousands of scenarios, the problems can easily have dozens of millions of variables. A problem over the entire Midwest region can similarly reach hundreds of millions of variables. In addition to the size of the problem, the tight connectivity of the network and the large number of wait-and-see decisions related to real-time enforced generation levels severely complicate decomposition, memory storage, and scalability.

In this work we use PIPS to analyze the scalability and real time solution capability of stochastic programming in unit commitment and economic dispatch problems using real problem data and dimensions. This has enabled us to pinpoint scalability deficiencies arising from physical systems and to critically assess performance by using end-user specifications. The analysis is important not only to demonstrate the improvements in our developments but also to pave the way for future computational research challenges associated with the next-generation power grid.

3. DECOMPOSITION OF STOCHASTIC PROGRAMMING PROBLEMS

We first describe the linear algebra behind the scenario-based decomposition. We then present our specialized factorization procedure for the dense Schur complement.

3.1 Schur complement decomposition of SAA problems

In this section we present the linear algebra needed to solve convex quadratic SAA problems of form (1) by interior-point methods. We refer the reader to [11], [12], [13], or [18] for more details on how the linear algebra is derived.

The deterministic SAA problem (1) has an arrow-shaped structure that can be exploited to produce highly parallelizable linear algebra. The matrix of the linear system that needs to be solved at each iteration of the interior-point algorithm can be simplified to the form

$$K := \begin{bmatrix} K_1 & & & B_1 \\ & \ddots & & \vdots \\ & & K_N & B_N \\ B_1^T & \dots & B_N^T & K_0 \end{bmatrix}. \tag{4}$$

The main computational effort at each iteration of the interior-point algorithm is solving linear systems of the form $K\Delta z = r$. The structure of K allows us to express an explicit block LDL^T factorization, where L is a unit lower triangular matrix and D is a diagonal matrix. One can easily verify that L and D have the following particular structures,

$$L = \begin{bmatrix} L_1 & & & \\ & \ddots & & \\ & & L_N & \\ L_{N1} & \dots & L_{NN} & L_c \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & & & \\ & \ddots & & \\ & & D_N & \\ & & & D_c \end{bmatrix},$$

where

$$\begin{aligned} L_i D_i L_i^T &= K_i, \ i = 1, \dots, N, & (5)\\ L_{Ni} &= B_i^T L_i^{-T} D_i^{-1}, \ i = 1 \dots, N, & (6)\\ C &= K_0 - \sum_{i=1}^N B_i^T K_i^{-1} B_i, & (7)\\ L_c D_c L_c^T &= C. & (8) \end{aligned}$$

We note that C defined by (7) is the Schur complement of the first-stage Hessian block K_0 in the entire Hessian matrix K.

Let $\Delta z_i = \begin{bmatrix} \Delta x_i^T & \Delta y_i^T \end{bmatrix}^T$, $i = 0, 1, \dots, N$, $\Delta z = \begin{bmatrix} \Delta z_1^T & \dots & \Delta z_N^T & \Delta z_0^T \end{bmatrix}^T$, and let r be of the form $\begin{bmatrix} r_1^T & \dots & r_N^T & r_0^T \end{bmatrix}^T$. To solve the linear system $K\Delta z = r$, we take the following steps:

$$\begin{aligned} w_i &= L_i^{-1} r_i, \ i = 1, \dots, N, & (9)\\ w_0 &= L_c^{-1}\left(r_0 - \sum_{i=1}^N L_{Ni} w_i\right), & (10)\\ v_0 &= D_0^{-1} w, \ v_i = D_i^{-1} w_i, \ i = 1, \dots, N, & (11)\\ \Delta z_0 &= L_c^{-1} v_0, & (12)\\ \Delta z_i &= L_i^{-T}(v_i - L_{Ni}\Delta z_0), \ i = 1, \dots, N. & (13) \end{aligned}$$

These operations can be divided into four phases:

- *Factorization*: steps (5), (6), and (8);
- *Computation of the Schur complement matrix*: step (7);
- *Forward substitution*: steps (9) and (10);
- *Diagonal solve*: step (11);
- *Back substitution*: steps (12) and (13).

The computation of the Schur complement matrix, that is, forming $B_i^T K_i^{-1} B_i$ for each scenario, is by far the most expensive phase. This phase, which we call the "backsolve" phase, is embarrassingly parallel, since the calculations can be performed independently. There are, however, communication costs to sum the contributions and explicitly form C. In general, in any phase, operations related to the second stage can be done independently for each scenario, yielding the so-called scenario-based decomposition.

The factorization and triangular solves involving the dense Schur complement matrix C are performed by using distributed dense linear algebra, with the specialized factorization described below.

3.2 Specialized LDL^T factorization for the Schur-complement matrix

It has been shown [18] that the dense Schur-complement system C has the saddle-point structure

$$C = \begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix}, \tag{14}$$

where Q is positive definite and A is full rank. Although using a general LU (Gaussian elimination) routine to factorize C presents a practicable solution, it is not ideal. We would expect to be able to gain a 2x increase in performance by using an algorithm that at least exploited the symmetric structure. We describe below a specialized LDL^T factorization that exploits both the symmetric and the saddle-point structure of C. This approach was first presented in [16].

The standard approach to solving symmetric indefinite systems is to use an LDL^T decomposition, where L is lower triangular and D is a block-diagonal matrix with blocks of size 1 or 2. This is used in the Bunch-Kaufman [6] and Bunch-Parlett [7] methods, which are numerically stable.

However, in the present case where the Q block is positive definite and the matrix A is full rank, one can write an explicit block form of the LDL^T decomposition where D is strictly diagonal:

$$C = \begin{bmatrix} M & 0 \\ AM^{-T} & \widetilde{M} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} M^T & M^{-1}A^T \\ 0 & \widetilde{M}^T \end{bmatrix}, \quad (15)$$

where M and $\widetilde{M}$ are lower triangular Cholesky factors satisfying $MM^T = Q$ and $\widetilde{M}\widetilde{M}^T = AQ^{-1}A^T$. These factors necessarily exist because Q is positive definite, and therefore $AQ^{-1}A^T$ is positive definite as well because A has full rank.

While not directly practical in the sparse case, in the dense case the factorization (15) can be formed explicitly in a sequence of operations similar to a step of blocked right-looking Cholesky factorization [23], which requires half the Flops of an LU decomposition. The factorization may be formed *in place* on a distributed matrix. We describe our implementation of this procedure in Section 4.1.

4. IMPLEMENTATION

The three important parts of the implementation, in terms of execution time, are the first-stage linear algebra, second-stage linear algebra, and assembly of the Schur complement. We describe these in Sections 4.1, 4.2, and 4.3, respectively.

4.1 First stage linear systems

The primary linear algebra operation related to the first stage is solving a dense linear system by using the specialized LDL^T factorization procedure described in Section 3.2. We use Elemental [19], a distributed dense linear algebra library currently under active development.

Elemental uses an *element-cyclic* matrix distribution for load balancing. Given an $n_p \times m_p$ MPI process grid, the element-cyclic distribution arises by assigning element (i, j) to process $(i \mod n_p, j \mod m_p)$. Each process has a single column-oriented local storage buffer, where the local elements are stored in their original shape, *as if* there were no elements of the matrix between them. In comparison, ScaLAPACK [5], a widely used distributed dense linear algebra package, uses larger storage blocks, called *block-cyclic* storage. The practical significance of the different storage methods in our case is that element-cyclic distribution makes it easy to perform operations on arbitrarily sized subblocks, a feature critical for the specialized LDL^T factorization routine.

Listing 1 contains the raw code to perform the factorization, annotated with the mathematical description. On input, the distributed matrix B contains the Schur-complement C, and on output it contains the lower triangular L factor as in (15). Full explanation of the procedure is given in [16]. Readers familiar with ScaLAPACK may be surprised by the simplicity of the Elemental API, which uses objects to refer to matrix subblocks instead of requiring complicated indexing arguments.

```
using namespace elemental;
void factorLDLT(DistMatrix<double,MC,MR> &B,
     int nx)
{
  const Grid &g = B.Grid();
  DistMatrix<double,MC,MR> BTL(g), BTR(g),
       BBL(g), BBR(g);
  // get references to subblocks
  // e.g. BTL (B top left) is nx × nx
  PartitionDownDiagonal(B,
                            BTL, BTR,
                            BBL, BBR,
                            nx);
  // BTL := Chol(Q) = M
  lapack::Chol(Lower, BTL);
  // BBL := AM^{-T}
  blas::Trsm(Right, Lower, Transpose,
       NonUnit, 1, BTL, BBL);
  // BBR := (AM^{-T})(AM^{-T})^T = AQ^{-1}A^T
  blas::Syrk(Lower, Normal, 1, BBL, 0, BBR);
  // BBR := Chol(AQ^{-1}A^T) = M̃
  lapack::Chol(Lower, BBR);
}
```

Listing 1: LDL^T factorization code using Elemental.

Elemental has a special branch for Blue Gene/P systems (Elemental-BG/P), whose primary feature is the ability to align the MPI process grid with the topology of the 3D torus. SMP parallelism is attained through the use of parallel BLAS (IBM's ESSL-SMP) together with OpenMP for packing and unpacking data.

4.2 Second stage linear systems

Recall that the primary computation for each second-stage scenario is the "backsolve" phase, that is, forming $B_i^T K_i^{-1} B_i$. Note that the K_i matrices are sparse and symmetric indefinite. We perform the calculation by solving the sparse system $K_i X_i = B_i$ for X_i (hence *backsolve*), then forming $B_i^T X_i = B_i^T K_i^{-1} B_i$, which is *dense* because of the matrix inversion. In the actual implementation, we compute blocks of columns of $B_i^T K_i^{-1} B_i$ and interlace the computation with the assembly of C, as described in Section 4.3.

In our current decomposition, each scenario is assigned to exactly one MPI process. To avoid load-balancing issues, we consider only cases where the number of MPI processes divides the number of scenarios, but this is not an intrinsic restriction. Second-stage calculations are performed by using SMP. Therefore, we require a pure-SMP sparse linear algebra library with support for solving symmetric indefinite systems with multiple right hand sides. We chose WSMP [14], a proprietary library by Anshul Gupta of IBM, because it has all the required features. The most viable alternative is PARDISO [20]; however, a binary could not be obtained for the Blue Gene platform. MUMPS [1, 2] is a primarily MPI-based parallel sparse library, with initial work still being performed to add hybrid MPI/SMP and pure SMP capabilities [8].

WSMP manages its own pthreads and requires serial BLAS. This is a different SMP paradigm from Elemental and requires setting the environmental variable `BG_APPTHREADDEPTH` = 2 so that OpenMP and WSMP threads may coexist (WSMP threads are never destroyed once the library is initialized). We also call `omp_set_num_threads(1)` before WSMP calls, to force serial BLAS, and again after the calls with argument 4 to restore parallel BLAS and full OpenMP.

4.3 Assembling the Schur complement

The assembly of the Schur-complement C as a distributed dense matrix in the element-cyclic distribution as required by Elemental can be a costly operation, possibly more costly than the factorization itself. This operation must be streamlined to obtain acceptable large-scale performance.

We present a simplified version of the summation at Step (7) in Section 3.1 to form C. To avoid confusion, in this simplified version let B refer to the distributed matrix. Let $\mathcal{P}$ be the set of nodes. The distribution operation we must perform can be described as

$$B := \sum_{p \in \mathcal{P}} M_p, \qquad (16)$$

where M_p is calculated locally on node p and B is stored across MPI processes in the element-cyclic distribution.

In the case where B is instead stored in entirety on each node and linear algebra is performed by using LAPACK (an approach we previously used, viable only for small B, see [16]), this operation maps directly to an `MPI_Allreduce`. In the current case, however, where B is distributed, two important considerations make the distribution problem significantly more complicated:

- M_p itself is too large to fit entirely in a node's local memory in general.
- Every node owns different, noncontiguous elements in B; however, all nodes contribute to all elements.

To address these issues, we calculate M_p in blocks of columns, and distribute these blocks as they are calculated, interweaving calculation and assembly. The communication pattern required in fact maps closely to an `MPI_Reduce_scatter`, in which a large array is reduced (summed) across all nodes and then its pieces are partitioned and sent to their destination. However, `MPI_Reduce_scatter` requires that each node receive a single contiguous part of the send buffer. Considering the distribution of the matrix across nodes, a contiguous column of the matrix cannot be partitioned such that the elements belonging to a given node are in contiguous memory. Intermediate packing and unpacking steps are therefore necessary.

A final complication is that for the LDL^T factorization, we would be performing unnecessary work by distributing the entire symmetric matrix. In initial experiments, we noticed that the communication time can be even more significant than the factorization time. We therefore developed and present a procedure that can effectively cut communication costs in half by distributing only the lower triangular part.

The distribution procedure comprises three steps: **Pack**, **Reduce_scatter**, and **Unpack**. For a fixed block of columns, the procedure is as follows. The local backsolve calculations are performed to fill a column buffer. This is not considered part of the time taken for the reduce. In the **Pack** step, the send buffer is filled with the lower triangular elements according to the distribution required by `MPI_Reduce_scatter`. We then call `MPI_Reduce_scatter` and **Unpack** the results into the local matrix storage. This is repeated for the next block of columns until the matrix has been completely formed. The procedure is illustrated in Figure 3.

An important question is choosing the number of columns to send at each iteration. Note that the backsolve procedure generates full columns, whereas only the lower triangular elements are sent. Clearly, if the number of columns is fixed at each iteration, the `MPI_Reduce_scatter` calls will send successively less and less data, leading to larger and larger communication overhead. The solution is to fix the size of the send buffer (100 MiB in our tests) and at each iteration calculate exactly as many columns as whose lower triangular elements fit in the send buffer. Calculating this number of columns reduces to solving a simple quadratic equation. We may avoid an explosion in the size of the column buffer by storing only the lower triangular elements from the full columns as they are generated.

While the procedure is generally straightforward, special care is needed at some points to ensure an efficient implementation. In order to use `MPI_Reduce_scatter`, the send buffer must be arranged such that the elements destined for a node are in a single, contiguous block and the blocks must be ordered according to MPI rank. For fast unpacking, we also require that inside a block, the order of elements match their order in the local matrix storage. We have fully specified a one-to-one map between the location of the elements in the column buffer and their location in the send buffer, and theoretically only a permutation of the column buffer is necessary. An in-place permutation is not practical because of poor cache performance, so we allocate a separate array for the send buffer and copy the elements into their positions. The copy procedure is streamlined and accelerated with OpenMP, taking care to avoid expensive division and modulus operations that might naïvely be used to calculate the required positions of the elements. There is a small overhead to calculate offsets of the lower triangular elements.

On Blue Gene/P, `MPI_Reduce_scatter` is implemented as an `MPI_Reduce` followed by `MPI_Scatterv`. We use the `MPI_COMM_WORLD` communicator to ensure that `MPI_Reduce` uses the collective network. In the **Unpack** step, simple `memcpy` operations copy the the lower triangular blocks into the columns of the local matrix storage. This is also ac-

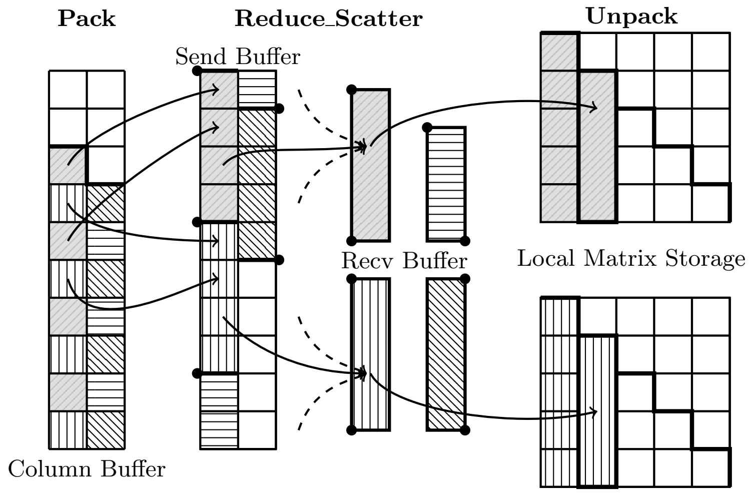

Figure 3: Illustration of a step of the "lower triangular reduce" procedure. The 3rd and 4th columns are sent of a 10×10 distributed matrix on a 2×2 processor grid. The lower triangular elements are indicated. Dashed lines indicate communication from other processes. Dots indicate the partitions of the column-major send buffer. In the illustrated case, only two columns fit in the send buffer. Note that in general, not all MPI processes will receive an equal number of elements, because of the properties of the matrix distribution.

celerated with OpenMP. MPI types could alternatively be used here to directly receive into the local matrix storage; however, `MPI_Scatterv` on Blue Gene/P is optimized for contiguous data types. In our experience, the **Pack** and **Unpack** steps take approximately 10% of the total distribution time.

Note that there is a necessary synchronization point between all nodes for each `MPI_Reduce_scatter` call, which can magnify possible load imbalances in the backsolve phase. We are following the proposals for nonblocking collective operations for the future MPI-3 standard. Our communication pattern may be a good example of an application for a potential `MPI_Ireduce_scatter`.

5. NUMERICAL EXPERIMENTS ON BG/P

In the following sections we investigate the scaling and time-to-solution properties of PIPS for realistically sized problems. Finally, we note the sustained performance of the solver.

5.1 Strong Scaling

We test the strong scaling ability of PIPS using a problem with a fixed size and number of second-stage scenarios. For ease of notation, we take k = 1024. We fix a problem with 32k scenarios and perform five interior-point iterations with 4k, 8k, 16k, and 32k nodes, corresponding to 8, 4, 2, and 1 scenarios per node, respectively. Scenarios for the uncertainty due to weather are obtained by calibrating and sampling a hidden Markov model whose dynamics is described by the physics of the atmosphere, as implemented in the weather research forecast code WRF [9]. For testing purposes only, scenarios are replicated by using normal perturbations. We use a 4 hour time horizon with the economic dispatch problem described in Section 2. The first stage has 4,711 variables and 4,431 equality constraints, leading to a relatively small (9142×9142) dense Schur-complement matrix. The sparse second stage matrices K_i (4) are of size 44124×44124.

As shown in figure 4, from 4k to 8k, 16k, and 32k nodes, we obtain strong scaling efficiencies of 99%, 98%, and 96%, respectively. Total times for the five iterations are 125, 63, 32, and 16 minutes, respectively.

Figure 5 gives a breakdown of the three most important components of the execution time: the backsolves, the factorization, and the distribution of the Schur-complement matrix previously described. The distribution and factorization steps are communication intensive but, compared with the backsolves, take little time. The backsolves, which are the algorithmic bottleneck, are embarrassingly parallel, explaining the nearly ideal strong scaling results observed.

We note that the factorization times increase slightly with the number of nodes. If the Schur-complement matrix were larger, we would also observe strong scaling with the factorization, as we investigated in [16]. In this case, however, communication and not calculation is the bottleneck.

Surprisingly, the time to distribute the Schur complement *decreases* slightly as the number of nodes increases. This behavior is most likely due to a slight increase in load imbalance. We avoid large load imbalance issues by assigning each node the same number of scenarios, and therefore each node performs approximately the same number of calculations per iteration. However, there is still some variation in the backsolve times per scenario, which can be compounded when there are multiple scenarios per node. This imbal-

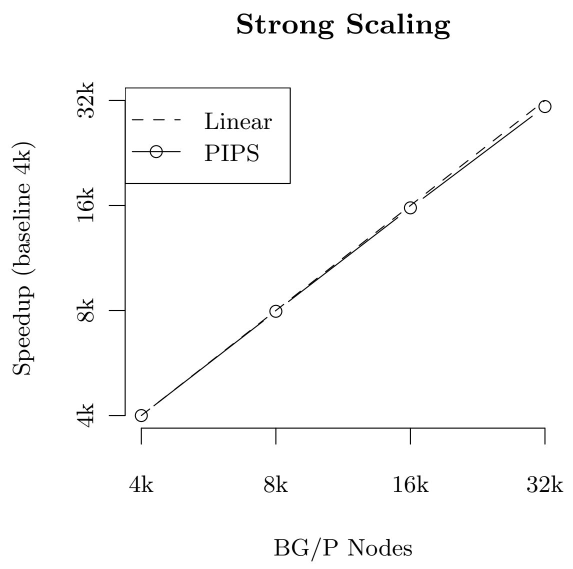

Figure 4: Parallel speedup for total walltime, 5 iterations. k = 1024.

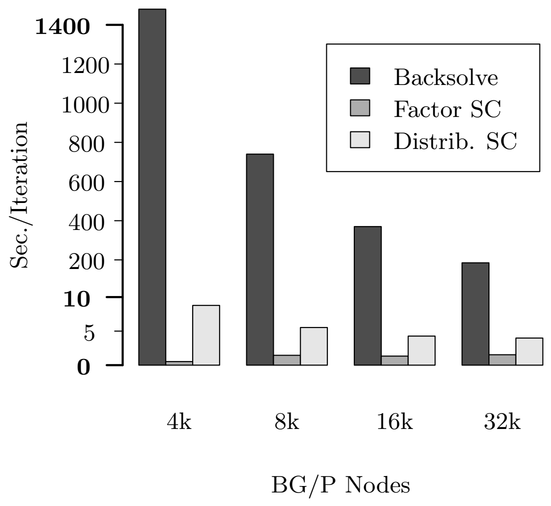

Figure 5: Components of execution time, averaged over 5 iterations. Note the break in axis scale; height differences between backsolves and others are greater than they appear. Backsolves scale in an embarrassingly parallel manner with the number of scenarios per node. Time to factorize and distribute the Schur-complement (SC) matrix increases slightly as a result of increased communication overhead, but remains very small compared with backsolve time.

ance is absorbed into the distribution time, which includes synchronization (no explicit barriers are used).

For the given problem, the strong scaling results indicate that the ideal configuration is in fact with the minimum one scenario per node. This is the algorithmic limit of the decomposition. Before implementation of the SMP model, the limit was a minimum of one scenario per core.

5.2 Solve to completion

In Section 5.1 we examined strong scaling by performing five interior-point iterations. This is a reasonable estimate of strong scaling for any number of iterations, since iteration time is generally constant. Of course, we are interested in the solution to the problem, which requires many more than five iterations.

Because of CPU time restrictions, we were unable to solve the 32k scenario problem to completion. Instead, we solved the same problem with 8k scenarios, on 8k nodes. This is a problem with 115 million total variables. Seventy-four iterations were required to reduce the duality gap by about 10^{15} with a barrier parameter $\mu = 2 \times 10^{-7}$, one of the standard termination criteria in interior-point practice [24]. Total execution time was 4 hours 10 minutes, including 7 minutes for initialization.

Larger problems, either with more scenarios or larger scenarios, would likely require more iterations. Nevertheless, interior-point experience tends to be that the number of iterations does not vary hugely inside a problem class, even for different size instances [24]. We therefore expect that the number of iterations required to observe similar reductions will be no larger than on the order of 100, even as the number of scenarios and time horizons radically increases. We may use this rationale to estimate the CPU time needed to complete an instance, which is essential in today's allocation-limited high-performance computing centers. Clearly, more numerical evidence is needed to support this subjective estimate, which is nonetheless done in light of substantial experience with such problems.

5.3 24+ hour scenarios

The ultimate goal is to solve problems with 24+ hour scenarios in *real time*, meaning a solution in under an hour. For 24-hour scenarios, the sparse second-stage matrices K_i are of size 338284×338284, and there are over 100,000 decision variables, for a total of about 3.5 billion variables. In preliminary tests, we determined that the backsolve phase takes ∼21 minutes per iteration on 32K nodes (compared with ∼3 minutes for 4-hour scenarios). Unfortunately, this is far too much for the size of BG/P allocation available to us. In its current form, it is also too much in terms of our real-time goal, where it would take about a day to compute using all of the IBM Blue Gene/P "Intrepid" system at Argonne National Laboratory (an estimate based on a 100 iterations of interior point as described above). We believe that several algorithmic and resource improvements would vastly improve the computational time and put us in a position to solve such problems in real time (we cannot offer that proof at this time, given both computational and development time limitations). They include splitting scenarios across nodes, increases in computing power (particularly in light of the coming vast increase in number of cores per node), refinements in the linear algebra, and exploring warm- and hot-starting for the interior point method, since rolling horizon problem must be solved repeatedly.

We note that strong scaling performance in the case of larger scenarios would be even greater than before, since the factorization and distribution times remain effectively constant. Conversely, speedups in the backsolves will decrease strong scaling performance but will also decrease total solution time.

5.4 Sustained performance

We measured total Flops of the entire solver (5 iterations, 4-hour scenarios) using the BG/P Universal Performance Counter (UPC) unit and obtained $\sim$1% of peak performance, about 4.4 TFlops on 32 racks. We caution, however, that sustained performance is in many senses not a reasonable measure of efficiency for sparse linear algebra, which is the primary operation in the solver (that is, in the backsolve phase), because the number of operations required to solve a sparse system is not fixed. A faster time to solution thanks to algorithmic improvement could easily decrease the Flops measured. Additionally, sparse operations by definition are difficult to vectorize, and often memory access is a bottleneck.

6. CONCLUSIONS

We conclude that the computational pattern of stochastic programming, while far from trivially parallel because of the synchronization bottleneck required by the first-stage decision variable computations, can nonetheless be suitable for massive parallel computing of the type provided by BG/P and similar systems. This statement is based on scalability results that we have obtained for up to 128k BG/P cores. Key to our accomplishment is a judicious use of linear algebra and efficient communication patterns.

Moreover, with algorithmic improvements, we expect that we will be able to solve such problems faster on existing architectures and with comparable scalability profiles on massively multicore architectures. In addition, we have demonstrated—for the first time on this scale, to our knowledge—that certain classes of power grid problems, namely, energy dispatch problems, can significantly benefit from existing and emerging very high-performance computing architectures. Such problems are likely to provide continuing challenges, since emerging engineering practice in energy dispatch may need three times longer horizons (California ISO is experimenting with 72-hour time horizons), 10 times more frequent temporal decisions due to evolution of energy market structure, 10 times larger spatial network (we are considering only Illinois at this time; the proper size for an ISO is about the size of the entire Midwest), and a yet-undetermined appropriate increase in the number of scenarios. While this is a daunting proposition, at stake is the proper usage of hundreds of gigawatts. In comparison, the power needed by the supercomputer to solve the optimization problem—a popular metric—would be essentially insignificant.

Acknowledgments

We thank Mike Papka for allocation support at the Argonne Leadership Computing Facility. We thank Anshul Gupta for timely support using WSMP and Jack Poulson, primary developer of Elemental, for continued support. We also thank Darius Buntinas for a conversation which helped with our analysis of the communication pattern. This work was supported by the U.S. Department of Energy under Contract DE-AC02-06CH11357.

7. REFERENCES

[1] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.*, 23:15–41, January 2001.

[2] P. R. Amestoy, A. Guermouche, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32:136–156, 2006.

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

[4] J. R. Birge and L. Qi. Computing block-angular Karmarkar projections with applications to stochastic programming. *Manage. Sci.*, 34(12):1472–1479, 1988.

[5] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[6] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31(137):163–179, 1977.

[7] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8(4):639–655, Dec. 1971.

[8] I. Chowdhury and J.-Y. L'Excellent. Some experiments and issues to exploit multicore parallelism in a distributed-memory parallel sparse direct solver. Technical Report RR-7411, INRIA, October 2010.

[9] E. M. Constantinescu, V. M. 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 Transactions on Power Systems*, 26(1):431–441, 2010.

[10] J. Gondzio and A. Grothey. Direct solution of linear systems of size 10^9 arising in optimization with interior point methods. In *PPAM*, pages 513–525, 2005.

[11] J. Gondzio and A. Grothey. Parallel interior-point solver for structured quadratic programs: Application to financial planning problems. *Annals of Operations Research*, 152(1):319–339, July 2007.

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

[13] J. Gondzio and R. Sarkissian. Parallel interior point solver for structured linear programs. *Mathematical Programming*, 96:561–584, 2003.

[14] A. Gupta. WSMP: Watson Sparse Matrix Package.

Technical report, IBM Research Report, 2000.

[15] J. Linderoth and S. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Comput. Optim. Appl.*, 24(2-3):207–250, 2003.

[16] M. Lubin, C. Petra, and M. Anitescu. On the parallel solution of dense saddle-point linear systems arising in stochastic programming. *Optimization Methods and Software, submitted*, 2010.

[17] S. Mehrotra and M. G. Ozevin. Decomposition based interior point methods for two-stage stochastic convex quadratic programs with recourse. *Oper. Res.*, 57(4):964–974, 2009.

[18] C. G. Petra and M. Anitescu. A preconditioning technique for Schur complement systems arising in stochastic optimization. Technical report, Preprint ANL/MCS-P1748-0510, Argonne National Laboratory,, 2010.

[19] J. Poulson, B. Marker, and R. A. van de Geijn. Elemental: A new framework for distributed memory dense matrix computations (FLAME Working Note #44). Technical report, Institute for Computational Engineering and Sciences, The University of Texas at Austin, June 2010.

[20] O. Schenk and L. Gartner. On fast factorization pivoting methods for symmetric indefinite systems. *Elec. Trans. Numer. Anal.*, 23:158–179, 2006.

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

[22] A. Shapiro, D. Dentcheva, and A. Ruszczyński. *Lectures on Stochastic Programming: Modeling and Theory.* MPS/SIAM Series on Optimization 9, Philadelphia, PA, 2009.

[23] R. A. van de Geijn. *Using PLAPACK.* MIT Press, March 1997.

[24] S. J. Wright. *Primal-Dual Interior-Point Methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[25] V. M. Zavala, A. Botterud, E. M. Constantinescu, and J. Wang. Computational and economic limitations of dispatch operations in the next-generation power grid. *IEEE Conference on Innovative Technologies for an Efficient and Reliable Power Supply*, 2010.

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