Parallel distributed-memory simplex for large-scale stochastic LP problems

Miles Lubin

with Julian Hall (University of Edinburgh),

Cosmin Petra, and Mihai Anitescu

Mathematics and Computer Science Division

Argonne National Laboratory, USA

ERGO Seminar

June 26th 2012

Overview

- Block-angular structure
- Motivation: stochastic programming and the power grid
- Parallelization of the simplex algorithm for block-angular linear programs

Large-scale (dual) block-angular LPs

$$
\begin{array}{lccccccccccc}
\min & c_0^T x_0 & + & c_1^T x_1 & + & c_2^T x_2 & + & \dots & + & c_N^T x_N & & \\
\text{s.t.} & A x_0 & & & & & & & & & = & b_0, \\
& T_1 x_0 & + & W_1 x_1 & & & & & & & = & b_1, \\
& T_2 x_0 & & & + & W_2 x_2 & & & & & = & b_2, \\
& \vdots & & & & & & \ddots & & & & \vdots \\
& T_N x_0 & & & & & & & + & W_N x_N & = & b_N, \\
& x_0 \geq 0, & & x_1 \geq 0, & & x_2 \geq 0, & & \dots, & & x_N \geq 0. & &
\end{array}
$$

- In terminology of stochastic LPs:
  - First-stage variables (decision now): x_0
  - Second-stage variables (recourse decision): $x_1, \dots, x_N$
  - Each diagonal block is a realization of a random variable (scenario)

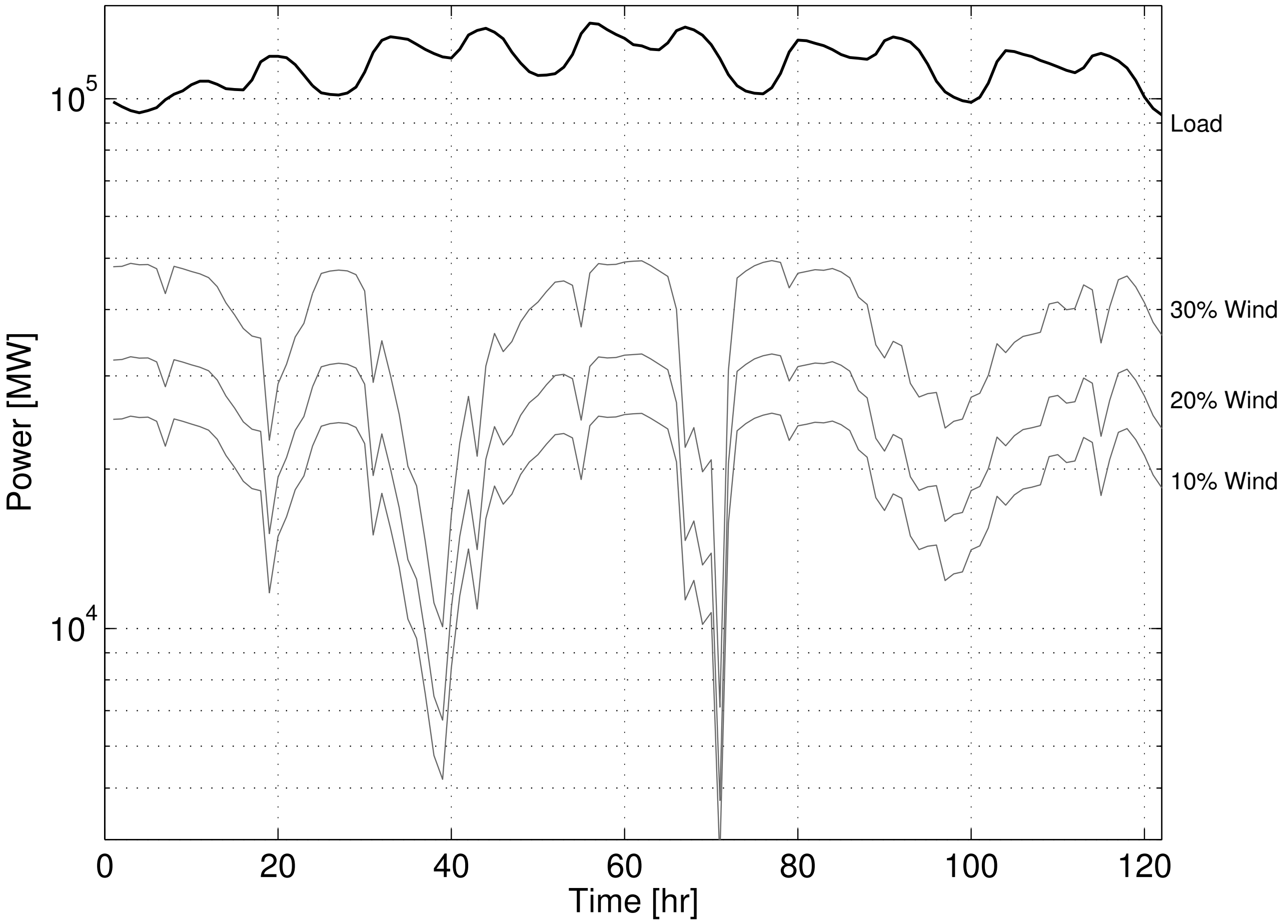
Why?

- Block-angular structure one of the first structures identified in linear programming
  - Specialized solution procedures dating to late 1950s
- Many, many applications
- We're interested in two-stage stochastic LP problems with a finite number of scenarios
  - Optimization under uncertainty
  - Power-grid control under uncertainty

Stochastic Optimization and the Power Grid

- **Unit Commitment**: Determine optimal on/off schedule of thermal (coal, natural gas, nuclear) generators. Day-ahead market prices. (hourly)
  - Mixed-integer
- **Economic Dispatch**: Set real-time market prices. (every 5-10 min.)
  - Continuous Linear/Quadratic
- **Challenge**: Integrate energy produced by highly variable renewable sources into these control systems.
  - Minimize operating costs, subject to:
    - Physical generation and transmission constraints
    - Reserve levels
    - Demand
    - …

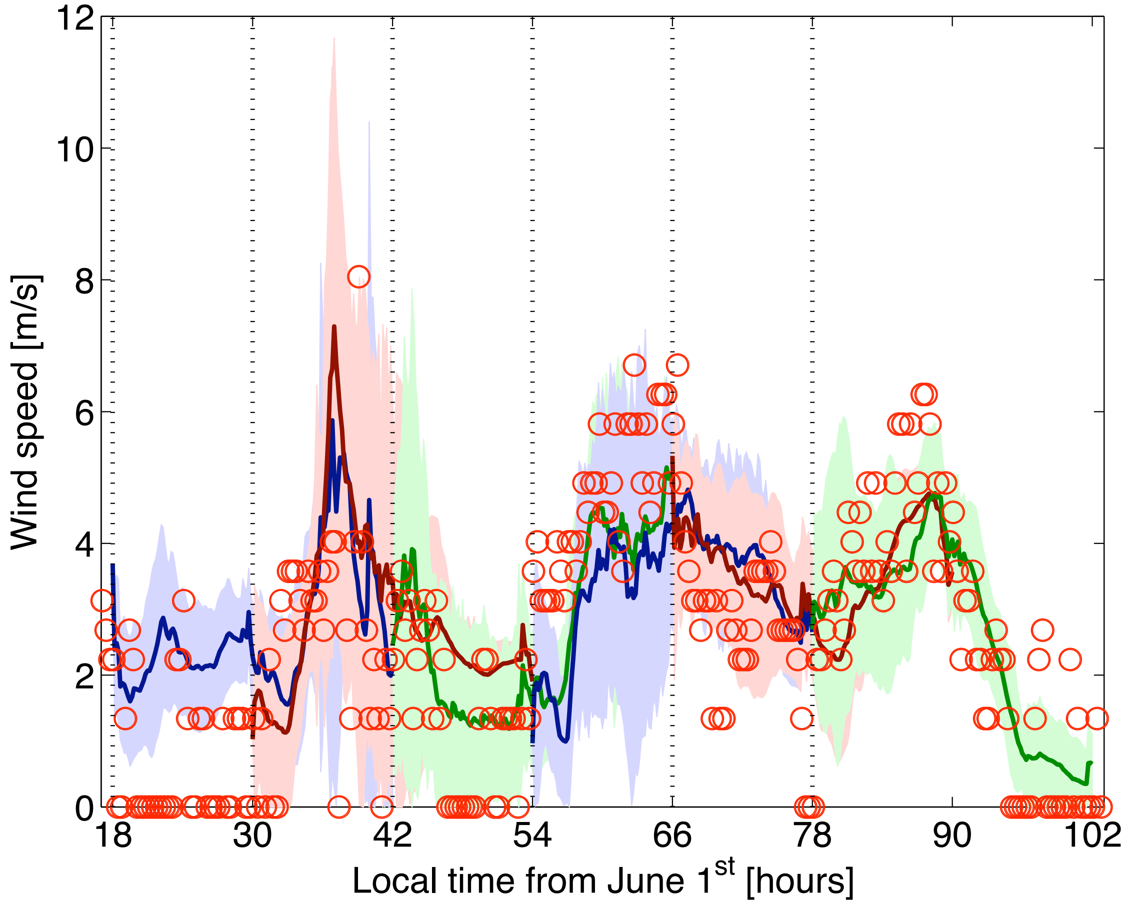
Variability in Wind Energy



Deterministic vs. Stochastic Approach

- To schedule generation, need to know how much wind energy there will be.
- **Deterministic**:
  - Run weather model once, obtain simple predicted values for wind. Plug into optimization problem.
- **Stochastic**:
  - Run ensemble of weather models to generate range of possible wind scenarios. Plug into stochastic optimization problem.
  - These are given to us (the optimizers) as input.

Deterministic vs. Stochastic Approach



- Single predictions may be very inaccurate, but truth usually falls within range of scenarios.
  - Uncertainty Quantification (Constantinescu, et al. 2010)

Stochastic Formulation

$$
\begin{aligned}
\min_{x \in \mathbb{R}^{n_1}} \quad & c^T x + \mathbb{E}_\xi[Q(x,\xi)] \\
\text{s.t.}\ & Ax = b, \\
& x \geq 0,
\end{aligned}
$$

where

$$
\begin{aligned}
Q(x,\xi) = \min_{y \in \mathbb{R}^{n_2}} \quad & q_\xi^T y \\
\text{s.t.}\ & T_\xi x + W y = h_\xi, \\
& y \geq 0.
\end{aligned}
$$

(some x, y integer)

- Discrete distribution leads to block-angular (MI)LP

Large-scale (dual) block-angular LPs

$$\begin{array}{rcccccccccccl}
\min & c_0^T x_0 & + & c_1^T x_1 & + & c_2^T x_2 & + & \ldots & + & c_N^T x_N & & \\
\text{s.t.} & A x_0 & & & & & & & & & = & b_0, \\
& T_1 x_0 & + & W_1 x_1 & & & & & & & = & b_1, \\
& T_2 x_0 & & & + & W_2 x_2 & & & & & = & b_2, \\
& \vdots & & & & & & \ddots & & & & \vdots \\
& T_N x_0 & & & & & & & + & W_N x_N & = & b_N, \\
& \multicolumn{12}{l}{x_0 \geq 0, \quad x_1 \geq 0, \quad x_2 \geq 0, \quad \ldots, \quad x_N \geq 0.}
\end{array}$$

- In terminology of stochastic LPs:
  - First-stage variables (decision now): x_0
  - Second-stage variables (recourse decision): $x_1, \ldots, x_N$
  - Each diagonal block is a realization of a random variable (scenario)

Difficulties

- May require many scenarios (100s, 1,000s, 10,000s …) to accurately model uncertainty
- "Large" scenarios (W_i up to 100,000 x 100,000)
- "Large" 1st stage (1,000s, 10,000s of variables)
- Easy to build a practical instance that requires 100+ GB of RAM to solve
  - → Requires distributed memory

Plus

- Integer constraints

Existing parallel solution methods

- Based on Benders decomposition
  - Classical approach
  - Asynchronous work by Linderoth and Wright (2003)
- Linear-algebra decomposition inside interior-point methods
  - OOPS (Gondzio and Grothey, 2009)
  - PIPS-IPM (Petra, et al.)
  - Demonstrated capability to efficiently solve large problems from scratch

Focus on warm starts

- With integer constraints, warm starts necessary inside branch and bound
- Real-time control (rolling horizons)
- Neither Benders or IPM approaches particularly suitable …
  - Benders somewhat warm-startable using regularization
  - IPM warm start possible but limited to ~50% speedup
- But we know an algorithm that is…

Idea

- Apply the (revised) simplex method directly to the large block-angular LP
- Parallelize its operations based on the special structure
- Many practitioners and simplex experts (attendees excluded) would say that this won't work

Overview of remainder

- The simplex algorithm
- Computational components of the revised simplex method
- Our parallel decomposition for dual block-angular LPs
- Numerical results
- First experiments with integer constraints

LP in standard form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Given a basis, projected LP

Given

$$A = \begin{bmatrix} B & N \end{bmatrix}$$

$$c = \begin{bmatrix} c_B & c_N \end{bmatrix}$$

$$x = \begin{bmatrix} x_B & x_N \end{bmatrix}$$

$$\begin{array}{ll} \min & c_B^T B^{-1} b + (c_N^T - c_B^T B^{-1} N) x_N \\ \text{s.t.} & B^{-1}(b - N x_N) \geq 0 \\ & x_N \geq 0 \end{array}$$

Idea of primal simplex

- Given a basis, define current iterates as

$$\hat{x}_B := B^{-1}b$$

$$\hat{x}_N := 0$$

$$\hat{s}_N := c_N - N^T B^{-T} c_B$$

- Assume $\hat{x}_B \geq 0$ (primal feasibility)
- If a component of $\hat{s}_N$ (reduced costs) is negative, increasing the corresponding component of $\hat{x}_N$ will decrease the objective, so long as feasibility is maintained.

Mathematical algorithm

- Given a basis and current iterates, identify index q such that $\hat{s}_q < 0$. **(Edge selection)**
  - If none exists, terminate with an optimal solution.
- Determine maximum step length θ^P such that $\hat{x}_B - \theta^P B^{-1} N e_q \geq 0$. **(Ratio test)**
  - Let p be the blocking index with $(\hat{x}_B - \theta^P B^{-1} N e_q)_p = 0$.
  - If none exists, problem is unbounded.
- Replace the pth variable in the basis with variable q. Repeat.

Computational algorithm

- Computational concerns:
  - Inverting basis matrix
  - Solving linear systems with basis matrix
  - Matrix-vector products
  - Updating basis inverse and iterates after basis change
  - **Sparsity**
  - Numerical stability
  - Degeneracy
  - …
- A modern simplex implementation is over 100k lines of C++ code.
- Will review key components.

Computational algorithm (Primal Simplex)

`CHUZC:` Scan $\hat{s}_N$ for a good candidate q to enter the basis.
`FTRAN:` Form the pivotal column $\hat{a}_q = B^{-1} a_q$, where a_q is column q of A.
`CHUZR:` Scan the ratios $(\hat{x}_B)_i / \hat{a}_{iq}$ for the row p of a good candidate to leave the basis.
Update $\hat{x}_B := \hat{x}_B - \theta^P \hat{a}_q$, where $\theta^P = (\hat{x}_B)_p / \hat{a}_{pq}$.
`BTRAN:` Form $\pi_p = B^{-T} e_p$.
`PRICE:` Form the pivotal row $\hat{a}_p = N^T \pi_p$.
Update reduced costs $\hat{s}_N := \hat{s}_N - \theta^D \hat{a}_p$, where $\theta^D = \hat{s}_q / \hat{a}_{pq}$.
`If` $\{$growth in representation of $B^{-1}\}$ `then`
    `INVERT:` Form a new representation of B^{-1}.
`else`
    `UPDATE:` Update the representation of B^{-1} corresponding to the basis change.
`end if`

Edge selection

- Choice in how to select edge to step along
  - Rule used has significant effect on the number of iterations
- Dantzig rule ("most negative reduced cost") is suboptimal
- In practice, *edge weights* used, choosing

$$q = \operatorname{argmax}_{\hat{s}_j < 0} |\hat{s}_j| / w_j .$$

  - Exact "steepest edge" (Forrest and Goldfarb, 1992)
  - DEVEX heuristic (Harris, 1973)
- Extra computational cost to maintain weights, but large decrease in number of iterations

Ratio test

- Also have choice in the ratio test
- "Textbook" ratio test: $\theta^P = \min_i \, (\hat{x}_B)_i / \hat{a}_{iq}$
  - Small values of $\hat{a}_{iq}$ cause numerical instability
  - Fails on practical problems
- Instead, use *two-pass* ratio test
  - Allow small infeasibilities in order improve numerical stability
  - See EXPAND (Gill et al., 1989)

Basis inversion and linear solves

- Typically, Markowitz (1957)-type procedure used to form sparse LU factorization of basis matrix
  - LU factorization before "LU factorization" existed
  - Gaussian elimination with pivotal row and column chosen dynamically to reduce fill-in of non-zero elements
  - Uncommon factorization outside of simplex; best for special structure of basis matrices (e.g. many columns of the identity, highly unsymmetric)
- Need to exploit sparsity in right-hand sides when solving linear systems (*hyper-sparsity*, see Hall and McKinnon, 2005)

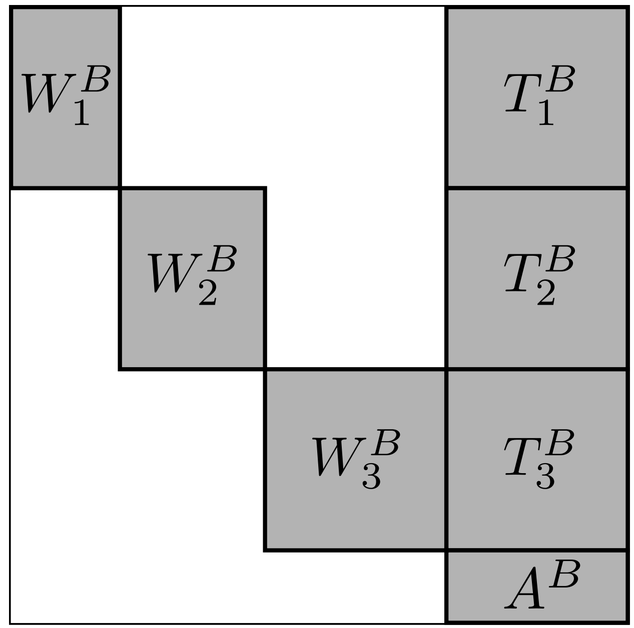
Basis updates

- At every iteration, a column of the basis matrix is replaced.
  - Inefficient to recompute factorization from scratch each time.
- Product-form update: (earliest form, Dantzig and Or-H, 1954)

$$\begin{aligned}\overline{B} &= B + (a_q - Be_p)e_p^T \\ &= B(I + (\hat{a}_q - e_p)e_p^T), \hat{a}_q = B^{-1}a_q. \\ E &:= (I + (\hat{a}_q - e_p)e_p^T)^{-1} = (I + \eta e_p^T). \\ \rightarrow \overline{B}^{-1} &= EB^{-1}\end{aligned}$$

- Originally used to invert the basis matrix! (column by column)
- Today, LU factors updated instead (e.g, Forrest and Tomlin, 1972)

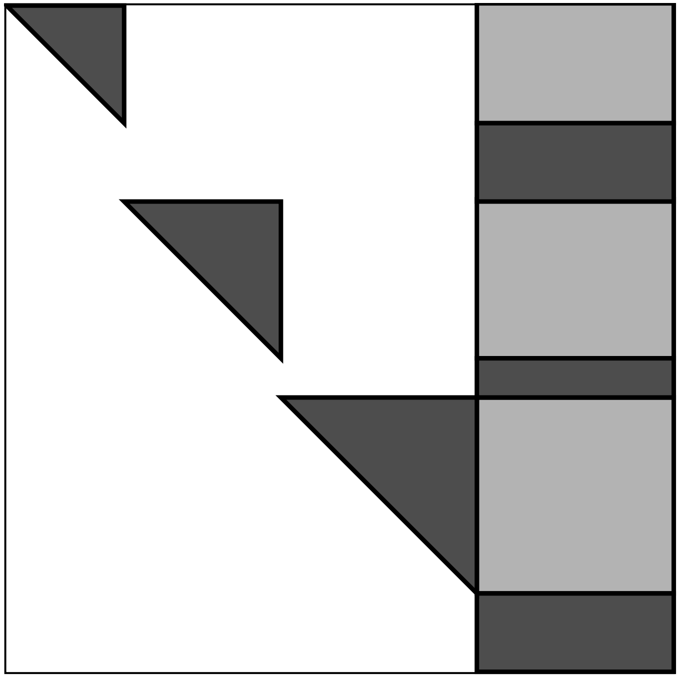
Decomposition – Structure of the basis matrix



$$
\begin{array}{rccccccccccc}
\min & c_0^T x_0 & + & c_1^T x_1 & + & c_2^T x_2 & + & \dots & + & c_N^T x_N & & \\
\text{s.t.} & A x_0 & & & & & & & & & = & b_0, \\
 & T_1 x_0 & + & W_1 x_1 & & & & & & & = & b_1, \\
 & T_2 x_0 & & & + & W_2 x_2 & & & & & = & b_2, \\
 & \vdots & & & & & & \ddots & & & & \vdots \\
 & T_N x_0 & & & & & & & + & W_N x_N & = & b_N, \\
 & x_0 \geq 0, & & x_1 \geq 0, & & x_2 \geq 0, & & \dots, & & x_N \geq 0. & &
\end{array}
$$

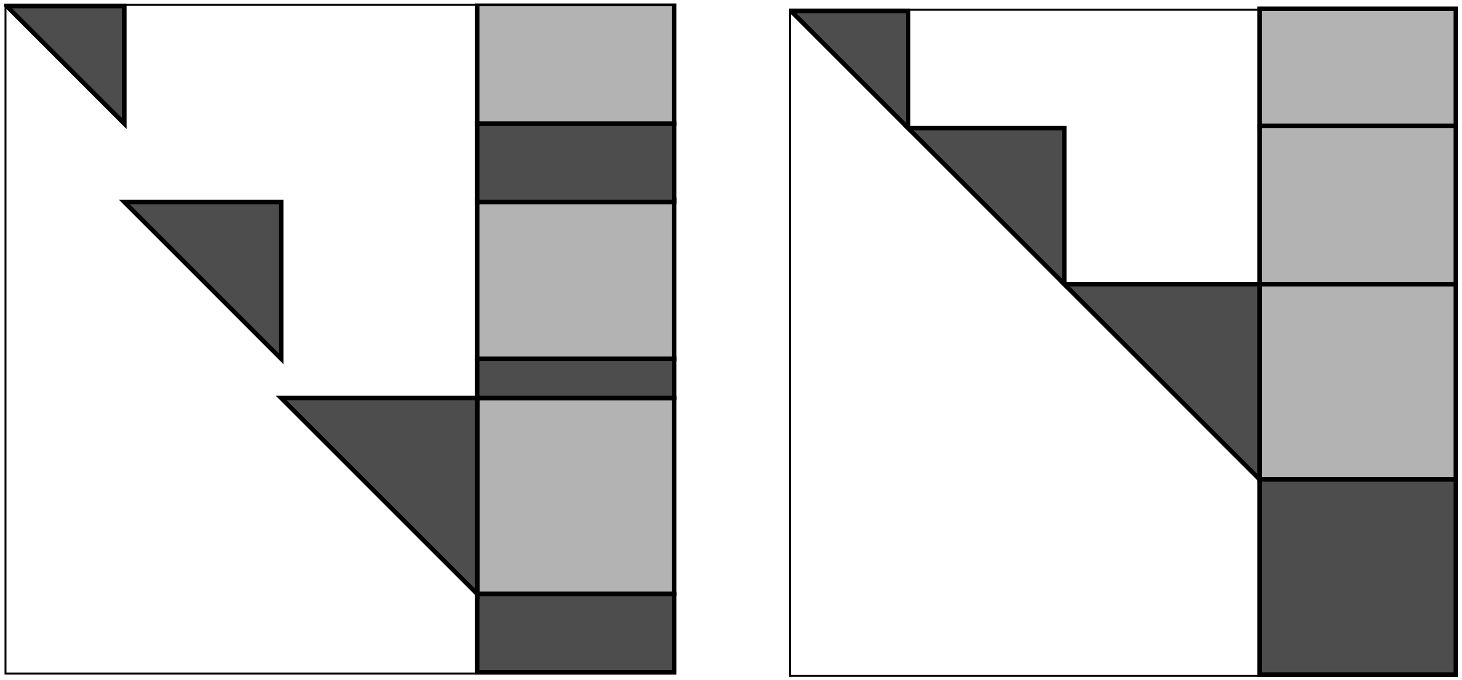
Key linear algebra

- Observation: Eliminating lower-triangular elements in diagonal blocks causes no structure-breaking fill-in
- Observation: May be performed in parallel



Key linear algebra – Implicit LU factorization

1. Factor diagonal blocks in parallel
2. Collect rows of square bottom-right *first-stage* system
3. Factor first-stage system



Implementation

- New codebase "PIPS-S"
  - C++, MPI
  - Reuses many primitives (vectors, matrices) from open-source CoinUtils
  - Algorithmic implementation written from scratch
  - Implements both primal and dual simplex

Implementation - Distribution of data

- Before reviewing operations, important to keep in mind distribution of data
- Targeting distributed-memory architectures (MPI) in order to solve large problems.
- Given P MPI processes and $N\,(\geq P)$ second-stage scenarios, assign each scenario to one MPI process.
- Second-stage data and iterates only stored on respective process. ➔Scalable
- First-stage data and iterates duplicated in each process.

$$
\begin{array}{rccccccccccl}
\min & c_0^T x_0 & + & c_1^T x_1 & + & c_2^T x_2 & + & \dots & + & c_N^T x_N & & \\
\text{s.t.} & A x_0 & & & & & & & & & = & b_0, \\
 & T_1 x_0 & + & W_1 x_1 & & & & & & & = & b_1, \\
 & T_2 x_0 & & & + & W_2 x_2 & & & & & = & b_2, \\
 & \vdots & & & & & & \ddots & & & & \vdots \\
 & T_N x_0 & & & & & & & + & W_N x_N & = & b_N, \\
 & \multicolumn{11}{l}{x_0 \geq 0, \quad x_1 \geq 0, \quad x_2 \geq 0, \quad \dots, \quad x_N \geq 0.}
\end{array}
$$

Computational algorithm (Primal Simplex)

`CHUZC:` Scan $\hat{s}_N$ for a good candidate q to enter the basis.

`FTRAN:` Form the pivotal column $\hat{a}_q = B^{-1}a_q$, where a_q is column q of A.

`CHUZR:` Scan the ratios $(\hat{x}_B)_i/\hat{a}_{iq}$ for the row p of a good candidate to leave the basis.
Update $\hat{x}_B := \hat{x}_B - \theta^P\hat{a}_q$, where $\theta^P = (\hat{x}_B)_p/\hat{a}_{pq}$.

`BTRAN:` Form $\pi_p = B^{-T}e_p$.

`PRICE:` Form the pivotal row $\hat{a}_p = N^T\pi_p$.
Update reduced costs $\hat{s}_N := \hat{s}_N - \theta^D\hat{a}_p$, where $\theta^D = \hat{s}_q/\hat{a}_{pq}$.

`If` $\{$growth in representation of $B^{-1}\}$ `then`

  `INVERT:` Form a new representation of B^{-1}.

`else`

  `UPDATE:` Update the representation of B^{-1} corresponding to the basis change.

`end if`

Implementation - Basis Inversion (INVERT)

- Want to reduce non-zero fill-in both in diagonal blocks and on the border
  - Determined by choice of row/column permutations
- Modify existing LU factorization to handle this, by giving as input the augmented system
$$\begin{bmatrix} W_i^B & T_i^B \end{bmatrix},$$
and restricting column pivots to the W_i^B block.
- Implemented by modifying `CoinFactorization` (John Forrest) of open-source CoinUtils package.
- Collect non-pivotal rows from each process, forming first-stage system. Factor first-stage system identically in each MPI process.

Implementation – Linear systems with basis matrix (FTRAN)

- Obtain procedure to solve linear systems with basis matrix by following math for inversion procedure; overview below:

1. Triangular solve for each scenario (parallel)
2. Gather result from each process (communication)
3. Solve first-stage system (serial)
4. Matrix-vector product and triangular solve for each scenario (parallel)

Implementation - Linear systems with basis transpose (BTRAN)

1. Triangular solve and matrix-vector product for each scenario (parallel)
2. Sum contributions from each process (communication)
3. Solve first-stage system (serial)
4. Triangular solve for each scenario (parallel)

Implementation - Matrix-vector product with non-basic columns (PRICE)

$$\begin{bmatrix} W_1^N & & & & T_1^N \\ & W_2^N & & & T_2^N \\ & & \ddots & & \vdots \\ & & & W_N^N & T_N^N \\ & & & & A^N \end{bmatrix}^T \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_N \\ \pi_0 \end{bmatrix} = \begin{bmatrix} (W_1^N)^T \pi_1 \\ (W_2^N)^T \pi_2 \\ \vdots \\ (W_N^N)^T \pi_N \\ (A^N)^T \pi_0 + \sum_{i=1}^N (T_i^N)^T \pi_i \end{bmatrix}$$

- Parallel procedure evident from above:

1. Compute $(W_i^N)^T \pi_i, (T_i^N)^T \pi_i$ terms (parallel)
2. Form $\sum_{i=1}^N (T_i^N)^T \pi_i$ (communication, `MPI_Allreduce`)
3. Form $(A^N)^T \pi_0$ (serial)

Implementation - Edge selection and ratio test

- Straightforward parallelization
- Each process scans through its local variables, then `MPI_Allreduce` determines the maximum/minimum across processes and its corresponding owner

Implementation - Basis updates

$$\overline{B}^{-1} = E_k \ldots E_2 E_1 B^{-1}$$

$$E_i = (I + \eta_i e_{p_i}^T)$$

- Consider operations to apply "eta" matrix to a right-hand side:

$$E_i x = (I + \eta_i e_{p_i}^T)x = (x + x_{p_i}\eta)$$

- What if *pivotal element* x_{p_i} is only stored on one MPI process?
  - Would need to perform a broadcast operation for every eta matrix; huge communication overhead
- Developed a procedure that requires only one communication per sequence of eta matrices.

Numerical Experiments

- Comparisons with highly-efficient serial solver Clp
- Presolve and internal rescaling disabled (not implemented in PIPS-S)
- 10^{-6} feasibility tolerances used
- Preview of conclusions before the numbers:
  - Clp 2-4x faster in serial
  - Significant speedups (up to 100x, typically less) over Clp in parallel
  - Solves problems that don't fit in memory on a single machine

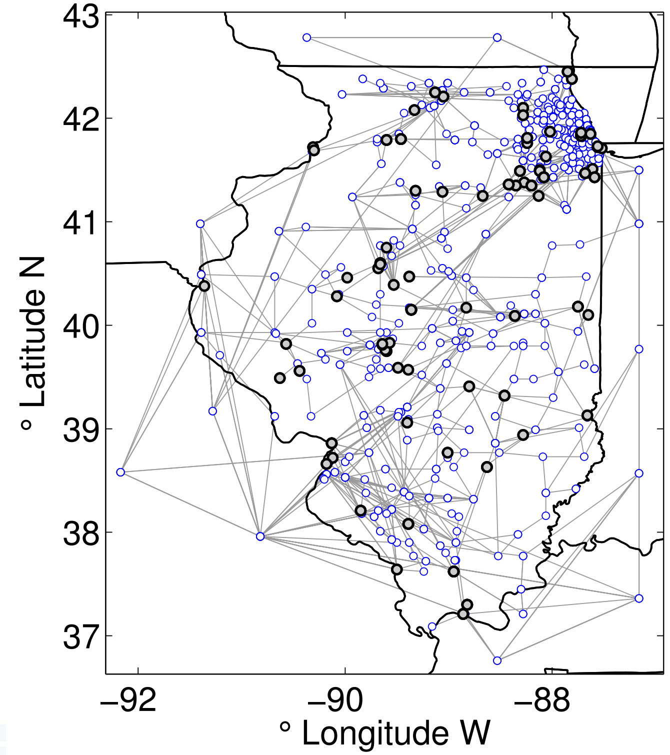
Test problems

Test Problem	1st Stage Vars.	1st Stage Cons.	2nd-Stage Scenario Vars.	2nd-Stage Scenario Cons.	Nonzero Elements A	Nonzero Elements W_i	Nonzero Elements T_i
Storm	121	185	1,259	528	696	3,220	121
SSN	89	1	706	175	89	2,284	89
UC12	3,132	0	56,532	59,436	0	163,839	3,132
UC24	6,264	0	113,064	118,872	0	327,939	6,264

- Storm and SSN used by Linderoth and Wright
- UC12 and UC24 developed by Victor Zavala
- Scenarios generated by Monte-Carlo sampling

UC12 and UC24

- Stochastic Unit Commitment models with 12-hour and 24-hour planning horizons over the state of Illinois.
- Includes (DC) transmission constraints.



Architectures

- “Fusion” high-performance cluster at Argonne
  - 320 nodes
  - InfiniBand QDR interconnect
  - Two 2.6 Ghz Xeon processors per node (total 8 cores)
  - Most nodes have 36 GB of RAM, some have 96 GB
- “Intrepid” Blue Gene/P supercomputer
  - 40,960 nodes
  - Custom interconnect
  - Each node has quad-core 850 Mhz PowerPC processor, 2 GB RAM

Large problems with advanced starts

- Solves "from scratch" not particularly of interest
- Consider large problems that require "high-memory" (96GB) nodes of Fusion cluster
  - 20-40 Million total variables/constraints
- Advanced starting bases in the context of:
  - Using solution to subproblem with a subset of scenarios to generate a starting basis for extensive form
    - Storm and SSN
    - Not included in time to solution
  - Simulate branch and bound (reoptimize after modifying bounds)
    - UC12 and UC24

Storm and SSN – 32,768 scenarios

Test Problem	Solver	Nodes	Cores	Iter./ Sec.
Storm	Clp	1	1	2.2
	PIPS-S	1	1	1.3
	"	1	4	10.0
	"	1	8	22.4
	"	2	16	47.6
	"	4	32	93.9
	"	8	64	158.8
	"	16	128	216.6
	"	32	256	260.4
SSN	Clp	1	1	2.0
	PIPS-S	1	1	0.8
	"	1	4	4.1
	"	1	8	10.5
	"	2	16	22.9
	"	4	32	46.8
	"	8	64	92.8
	"	16	128	143.3
	"	32	256	180.0

UC12 (512 scenarios) and UC24 (256 scenarios)

Test Problem	Solver	Nodes	Cores	Avg. Iter./Sec
UC12	Clp	1	1	0.73
	PIPS-S	1	1	0.34
	"	1	8	2.5
	"	2	16	4.7
	"	4	32	8.8
	"	8	64	14.9
	"	16	128	20.9
	"	32	256	25.8
UC24	Clp	1	1	0.87
	PIPS-S	1	1	0.36
	"	1	8	2.4
	"	2	16	4.4
	"	4	32	8.2
	"	8	64	14.8
	"	16	128	23.2
	"	32	256	28.7

Very big instance

- UC12 with 8,192 scenarios
  - 463,113,276 variables and 486,899,712 constraints
- Advanced starting basis from solution to problem with 4,096 scenarios
- Solved to optimal basis in 86,439 iterations (4.6 hours) on 4,096 nodes of Blue Gene/P (2 MPI processes per node)
- Would require ~1TB of RAM to solve in serial (so no comparison with Clp)

Performance analysis

- Simple performance model for execution time of an operation:

$$\max_p \{t_p\} + c + t_0,$$

where t_p is the time spent by process p on its local second-stage calculations, c is the communication cost, and t_0 is the time spent on the first-stage calculations.

- Limits to scalability:
  - Load imbalance: $\max_p\{t_p\} - \frac{1}{P}\sum_{i=1}^{P} t_p$
  - Communication cost: c
  - Serial bottleneck: t_0
- Instrumented matrix-vector product (PRICE) to compute these quantities

Matrix-vector product with non-basic columns (PRICE)

$$\begin{bmatrix} W_1^N & & & & T_1^N \\ & W_2^N & & & T_2^N \\ & & \ddots & & \vdots \\ & & & W_N^N & T_N^N \\ & & & & A^N \end{bmatrix}^T \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_N \\ \pi_0 \end{bmatrix} = \begin{bmatrix} (W_1^N)^T \pi_1 \\ (W_2^N)^T \pi_2 \\ \vdots \\ (W_N^N)^T \pi_N \\ (A^N)^T \pi_0 + \sum_{i=1}^N (T_i^N)^T \pi_i \end{bmatrix}$$

1. Compute $(W_i^N)^T \pi_i, (T_i^N)^T \pi_i$ terms (parallel)
2. Form $\sum_{i=1}^N (T_i^N)^T \pi_i$ (communication, `MPI_Allreduce`)
3. Form $(A^N)^T \pi_0$ (serial)

Performance analysis – "Large" instances

Test Problem	Nodes	Cores	Load Imbal. (μs)	Comm. Cost (μs)	Serial Bottleneck (μs)	Total Time/Iter. (μs)
Storm	1	1	0	0	1.0	13,243
	1	8	88	33	0.8	1,635
	2	16	40	68	0.9	856
	4	32	25	105	0.9	512
	8	64	26	112	1.0	326
	16	128	11	102	0.9	205
	32	256	34	253	0.8	333
SSN	1	1	0	0	0.8	2,229
	1	8	18	23	0.8	305
	2	16	25	54	0.8	203
	4	32	14	68	0.7	133
	8	64	12	65	0.7	100
	16	128	10	87	0.6	106
	32	256	8	122	0.6	135

Performance analysis – "Large" instances

Test Problem	Nodes	Cores	Load Imbal. (μs)	Comm. Cost (μs)	Serial Bottleneck (μs)	Total Time/Iter. (μs)
UC12	1	1	0	0	6.8	24,291
	1	8	510	183	6.0	4,785
	2	16	554	274	6.0	2,879
	4	32	563	327	6.0	1,921
	8	64	542	355	6.0	1,418
	16	128	523	547	6.0	1,335
	32	256	519	668	5.8	1,323
UC24	1	1	0	0	11.0	28,890
	1	8	553	259	9.8	5,983
	2	16	543	315	9.7	3,436
	4	32	551	386	9.6	2,248
	8	64	509	367	9.5	1,536
	16	128	538	718	9.5	1,593
	32	256	584	1413	9.5	2,170

Performance analysis

- First-stage calculation bottleneck relatively insignificant
- Load imbalance depends on problem
  - Caused by exploiting hyper-sparsity
- Communication cost significant, but small enough to allow for significant speedups
  - Speedups on Fusion unexpected
  - High-performance interconnects (Infiniband)

Back to what we wanted to solve - Preliminary results

- First-stage variables in UC12 are binary on/off generator states
- With 64 scenarios (3,621,180 vars., 3,744,468 cons., 3,132 binary)
  - LP Relaxation: 939,208
  - LP Relaxation + `CglProbing` cuts: 939,626
  - Feasible solution from rounding: 942,237
  - Optimality Gap: 0.27% (0.5% is acceptable in practice)
  - Starting with optimal LP basis:
    - 1 hour with PIPS-S on 4 nodes (64 cores) of Fusion
    - 4.75 hours with Clp in serial
- Further decrease in gap by better primal heuristics and more cut generators
- UC12 can be “solved” at the root node!
  - Reported in literature for similar deterministic model

Conclusions

- Simplex method is parallelizable for dual block-angular LPs
- Significant speedups over highly-efficient serial solvers possible on a high-performance cluster on appropriately sized problems
- Sequences of large-scale block-angular LPs can now be solved efficiently in parallel
- Path forward for block-angular MILPs
  - Solve stochastic unit commitment problem at root node?
  - Parallel simplex inside parallel branch and bound?

Conclusions

- Communication intensive optimization algorithms can successfully scale on today's high-performance clusters
  - Each simplex iteration has ~10 collective (broadcast/all-to-all) communication operations.
  - Observed 100s of iterations per second.
  - Communication cost is order of 10s/100s of *micro*seconds
    - Used to be order of milliseconds

Thank you!