Stochastic Nonlinear Programming with Pyomo and PySP

SAND2011-2648C

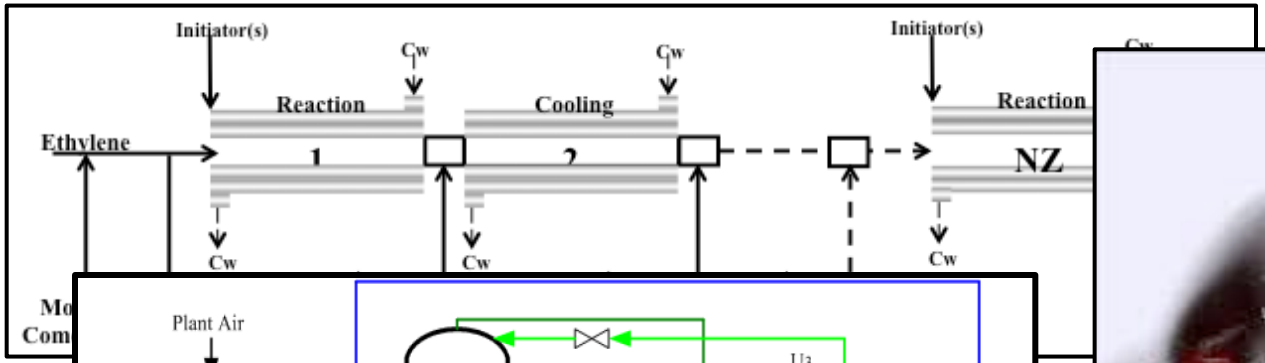
Carl D. Laird, Assistant Professor &
William and Ruth Neely Faculty Fellow
Artie McFerrin Department of Chemical Engineering
Texas A&M University, College Station, TX

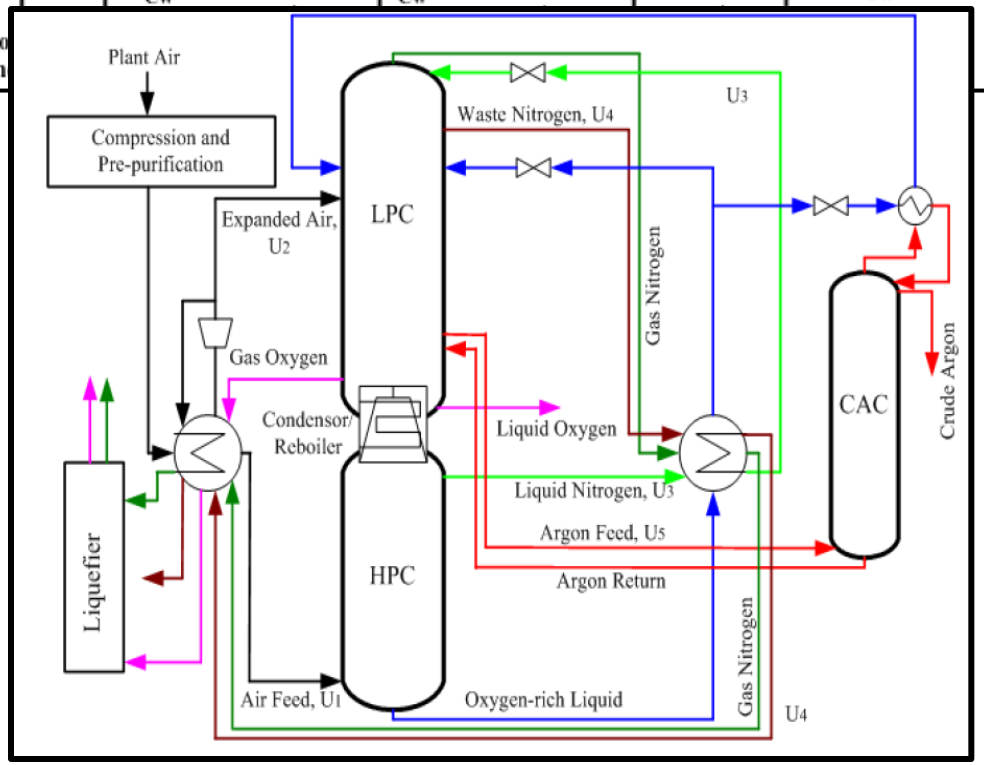
Jean-Paul Watson
Principal Member of Technical Staff
Discrete Math and Complex Systems Department
Sandia National Laboratories, Albuquerque, NM

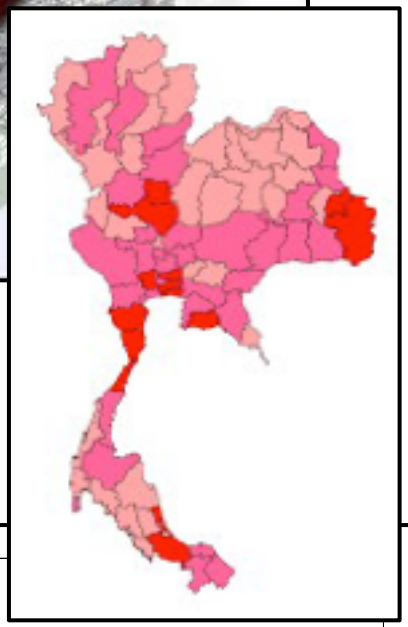
This work was funded by the DOE-Office of Science

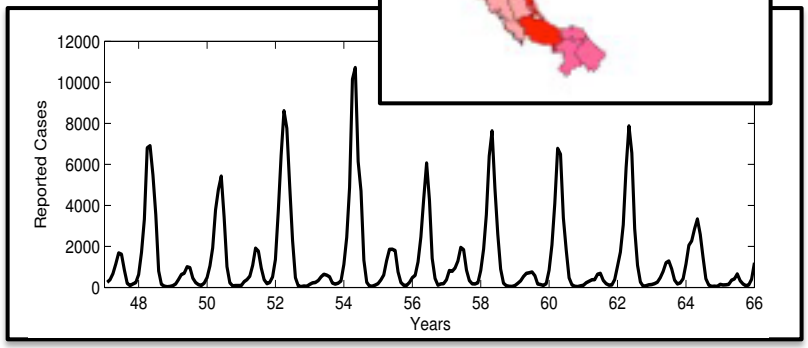
Artie McFerrin Department of
Chemical Engineering
Texas A&M University

Application Landscape





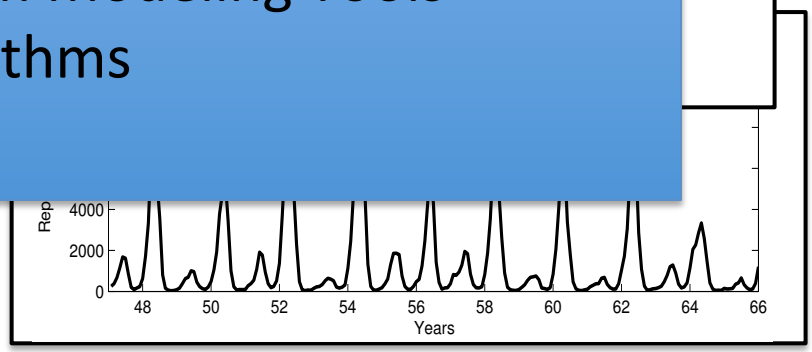




Application Landscape

- Large-Scale Nonlinear Problems, Often Many Scenarios
- Hardware Manufacturers Focusing on Multi-core Architectures
- Need for Parallel Tools to Solve Stochastic NLPs

- Challenges for the Application Developer:
  - Lack of Support for Effective Modeling and Parallel Model Evaluation in Modeling Tools
  - Lack of Off-the-shelf Algorithms



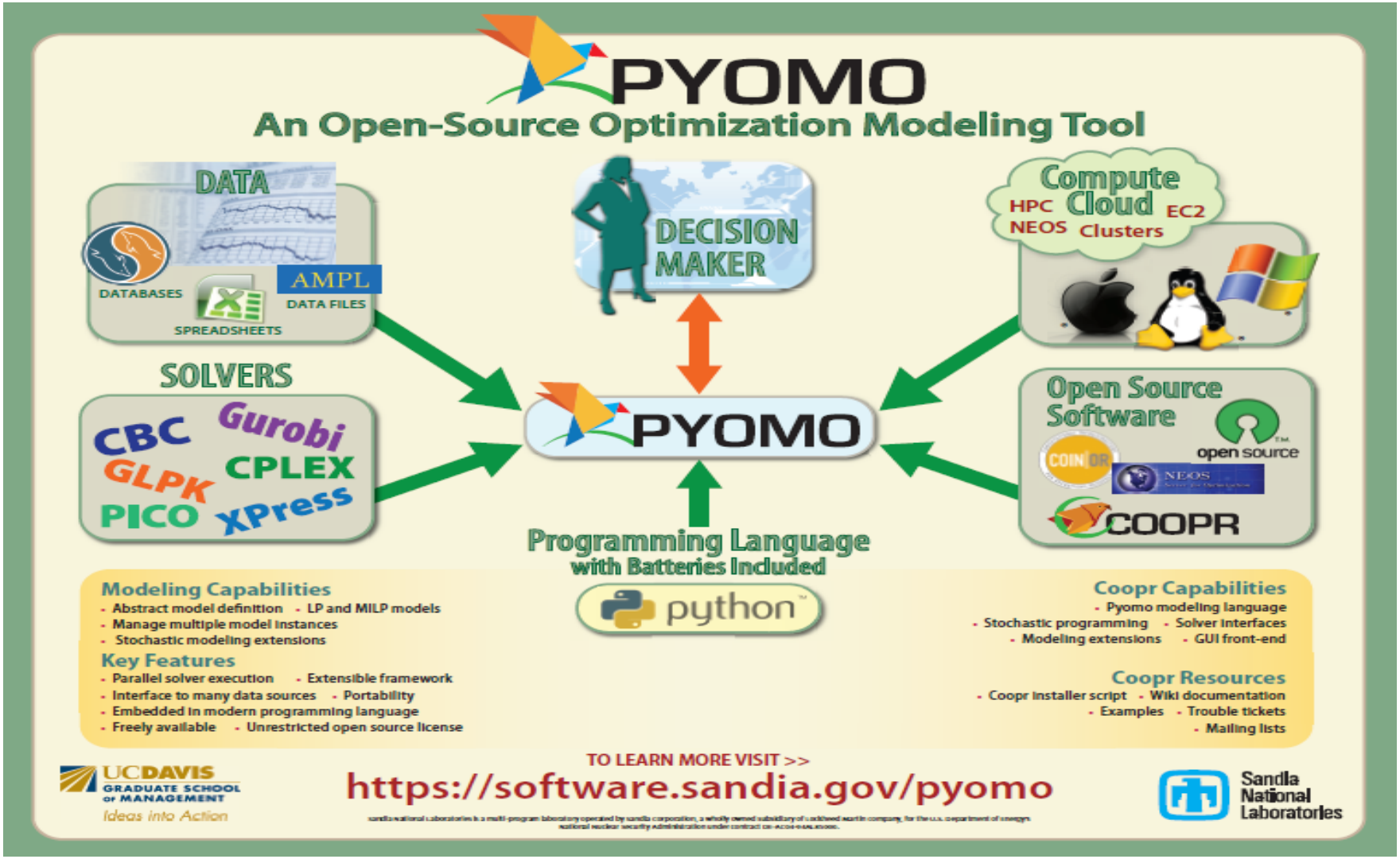
Parallel Solution of Stochastic NLP Problems

- PYOMO: Python Modeling Language for Optimization
  - Rich Environment for Tailored Modeling Extensions
  - Effective Development of Decomposition/Hybrid Algorithms

- PySP
  - Provides Modeling Extension to Represent Stochastic Programming Problems
  - Provides Decomposition Approaches for Efficient Parallel Solution of Stochastic Programming Problems
    - Progressive Hedging, Experimental Bender's

- Internal Linear Decomposition
  - Interior-Point Method (IPOPT)
  - Problem Tailored Linear Algebra (Decomposition of KKT System)
  - Schur-complement Decomposition
  - PCG/BFGS: Iterative Solution of SchurComplement

PYOMO: PYthon Optimization Modeling Objects



PySP: Motivation

- Numerous stochastic programming extensions to Algebraic Modeling Languages (AMLs) have been proposed over the last decade
  - Useful and necessary, especially for creating extensive forms

- Modeling is not our objective here, but rather a necessary pre-requisite

- Our goals

1. Break down the barrier between modeling languages and solvers

2. Provide model-agnostic stochastic programming algorithms

3. Facilitate rapid prototyping, development, and extension of algorithms

Sandia National Laboratories

Step #1: Formulate the Deterministic Model (1)

```
from coopr.pyomo import *
model=Model()

# Parameters
model.CROPS=Set()
model.TOTAL_ACREAGE=Param(within=PositiveReals)
model.PriceQuota=Param(model.CROPS, within=PositiveReals)
model.SubQuotaSellingPrice=Param(model.CROPS, within=PositiveReals)
model.SuperQuotaSellingPrice=Param(model.CROPS)
model.CattleFeedRequirement=Param(model.CROPS, \
                                  within=NonNegativeReals)
model.PurchasePrice=Param(model.CROPS, within=PositiveReals)
model.PlantingCostPerAcre=Param(model.CROPS, within=PositiveReals)
model.Yield=Param(model.CROPS, within=NonNegativeReals)

# Variables
model.DevotedAcreage=Var(model.CROPS, \
                         bounds=(0.0, model.TOTAL_ACREAGE))

model.QuantitySubQuotaSold=Var(model.CROPS, bounds=(0.0, None))
model.QuantitySuperQuotaSold=Var(model.CROPS, bounds=(0.0, None))

model.QuantityPurchased=Var(model.CROPS, bounds=(0.0, None))

model.FirstStageCost=Var()
model.SecondStageCost=Var()
```

ReferenceModel.py

Birge and Louveaux's (1997) Farmer

Sandia National Laboratories

Step #1: Formulate the Deterministic Model (2)

```
# Constraints
def total_acreage_rule(model):
    return summation(model.DevotedAcreage) <= model.TOTAL_ACREAGE
model.ConstrainTotalAcreage=Constraint(rule=total_acreage_rule)

def cattle_feed_rule(i, model):
    return model.CattleFeedRequirement[i] <= \
           (model.Yield[i] * model.DevotedAcreage[i]) + \
            model.QuantityPurchased[i] - \
            model.QuantitySubQuotaSold[i] - \
            model.QuantitySuperQuotaSold[i]
model.EnforceCattleFeedRequirement=Constraint(model.CROPS, \
                                              rule=cattle_feed_rule)

def limit_amount_sold_rule(i, model):
    return model.QuantitySubQuotaSold[i] + \
           model.QuantitySuperQuotaSold[i] <= \
           (model.Yield[i] * model.DevotedAcreage[i])
model.LimitAmountSold=Constraint(model.CROPS, \
                                 rule=limit_amount_sold_rule)

def enforce_quotas_rule(i, model):
    return (0.0, model.QuantitySubQuotaSold[i], model.PriceQuota[i])
model.EnforceQuotas=Constraint(model.CROPS, \
                               rule=enforce_quotas_rule)
```


ReferenceModel.py

Sandia National Laboratories

Step #1: Formulate the Deterministic Model (3)

```
# Stage-specific cost computations
def first_stage_cost_rule(model):
    return model.FirstStageCost == \
        summation(model.PlantingCostPerAcre, model.DevotedAcreage)
model.ComputeFirstStageCost=Constraint(rule=first_stage_cost_rule)

def second_stage_cost_rule(model):
    expr=summation(model.PurchasePrice, model.QuantityPurchased)
    expr -= summation(model.SubQuotaSellingPrice, \
                      model.QuantitySubQuotaSold)
    expr -= summation(model.SuperQuotaSellingPrice, \
                      model.QuantitySuperQuotaSold)
    return (model.SecondStageCost - expr) == 0.0
model.ComputeSecondStageCost=Constraint(rule=second_stage_cost_rule)

# Objective
def total_cost_rule(model):
    return (model.FirstStageCost + model.SecondStageCost)
model.Total_Cost_Objective=Objective(rule=total_cost_rule, \
                                     sense=minimize)
```

ReferenceModel.py

Sandia National Laboratories

Step #2: Specify the Deterministic Model Data

```
set CROPS := WHEAT CORN SUGAR_BEETS ;

param TOTAL_ACREAGE := 500 ;

param PriceQuota := WHEAT 100000 CORN 100000 SUGAR_BEETS 6000 ;

param SubQuotaSellingPrice := WHEAT 170 CORN 150 SUGAR_BEETS 36 ;

param SuperQuotaSellingPrice := WHEAT 0 CORN 0 SUGAR_BEETS 10 ;

param CattleFeedRequirement := WHEAT 200 CORN 240 SUGAR_BEETS 0 ;

param PurchasePrice := WHEAT 238 CORN 210 SUGAR_BEETS 100000 ;

param PlantingCostPerAcre := WHEAT 150 CORN 230 SUGAR_BEETS 260 ;

param Yield := WHEAT 3.0 CORN 3.6 SUGAR_BEETS 24 ;
```

ReferenceModel.dat

- Can initialize an instance from
  1. An AMPL .dat file
  2. Excel
  3. Raw Python

Sandia National Laboratories

Step #3: Specify the Scenario Tree

```
set Stages := FirstStage SecondStage ;

set Nodes := RootNode
             BelowAverageNode
             AverageNode
             AboveAverageNode ;

param NodeStage := RootNode           FirstStage
                   BelowAverageNode SecondStage
                   AverageNode        SecondStage
                   AboveAverageNode SecondStage ;

set Children[RootNode] := BelowAverageNode
                          AverageNode
                          AboveAverageNode ;

param ConditionalProbability := RootNode           1.0
                                BelowAverageNode   0.33333333
                                AverageNode        0.33333334
                                AboveAverageNode   0.33333333 ;

set Scenarios := BelowAverageScenario
                 AverageScenario
                 AboveAverageScenario ;

param ScenarioLeafNode := BelowAverageScenario BelowAverageNode
                          AverageScenario      AverageNode
                          AboveAverageScenario AboveAverageNode ;

set StageVariables[FirstStage] :=  DevotedAcreage[*] ;
set StageVariables[SecondStage] := QuantitySubQuotaSold[*]
                                   QuantitySuperQuotaSold[*]
                                   QuantityPurchased[*] ;

param StageCostVariable := FirstStage  FirstStageCost
                           SecondStage SecondStageCost ;
```


ScenarioStructure.dat

Sandia National Laboratories

Step #4: Specify the Scenario Instance Data

- Two methods are available to specify scenario-specific data
  - Scenario-based
  - Node-based

- In the scenario-based approach, a single and complete .dat file is specified for each individual scenario
  - Redundant, but straightforward if computer-generated

- In the node-based approach, a single .dat file is specified for each node in the scenario tree
  - Maximally compact, but requires some book-keeping

Sandia National Laboratories

Writing and Solving the Extensive Form (1)

- Now that you have a stochastic programming model in PySP…

- Step #1: Write the extensive form and pray that CPLEX can solve it
  - Fantastic if it works
  - But often it doesn't

- In PySP, the ***runef*** script is provided to both write and solve the extensive form of a stochastic programming model

- The basic command-line:

```
runef --model-directory=models \\
      --instance-directory=scenariodata \\
      --solve
```


Sandia National Laboratories

Writing and Solving the Extensive Form (2)

- After solution, you get (in addition to other information):

```
Tree Nodes:

        Name=RootNode
        Stage=FirstStage
        Variables:
                DevotedAcreage[CORN]=80.0
                DevotedAcreage[SUGAR_BEETS]=250.0
                DevotedAcreage[WHEAT]=170.0

        Name=AboveAverageNode
        Stage=SecondStage
        Variables:
                QuantitySubQuotaSold[CORN]=48.0
                QuantitySubQuotaSold[SUGAR_BEETS]=6000.0
                QuantitySubQuotaSold[WHEAT]=310.0

        Name=AverageNode
        Stage=SecondStage
        Variables:
                QuantitySubQuotaSold[SUGAR_BEETS]=5000.0
                QuantitySubQuotaSold[WHEAT]=225.0

        Name=BelowAverageNode
        Stage=SecondStage
        Variables:
                QuantitySubQuotaSold[SUGAR_BEETS]=4000.0
```


Sandia National Laboratories

What Happens if the Extensive Form is Too Difficult?

- *We use decomposition!*

Sandia National Laboratories

Progressive Hedging: A Review and/or Introduction

1. $k := 0$
2. For all $s \in \mathcal{S}$, $x_s^{(k)} := \text{argmin}_x(c \cdot x + f_s \cdot y_s) : (x, y_s) \in \mathcal{Q}_s$
3. $\bar{x}^k := (\sum_{s\in\mathcal{S}} p_s\, d_s x_s^{(k)}) / \sum_{s\in\mathcal{S}} p_s\, d_s$
4. For all $s \in \mathcal{S}$, $w_s^{(k)} := \rho(x_s^{(k)} - \bar{x}^{(k)})$
5. $k := k + 1$
6. For all $s \in \mathcal{S}$, $x_s^{(k)} := \text{argmin}_x(c \cdot x + w_s^{(k-1)} x + \rho/2 \left\|x - \bar{x}^{(k-1)}\right\|^2 + f_s \cdot y_s) : (x, y_s) \in \mathcal{Q}_s$
7. $\bar{x}^{(k)} := (\sum_{s\in\mathcal{S}} p_s\, d_s x_s^{(k)}) / \sum_{s\in\mathcal{S}} p_s\, d_s$
8. For all $s \in \mathcal{S}$, $w_s^{(k)} := w_s^{(k-1)} + \rho\left(x_s^{(k)} - \bar{x}^{(k)}\right)$
9. $g^{(k)} := \frac{(1-\alpha)|\mathcal{S}|}{\sum_{s\in\mathcal{S}} p_s\, d_s} \sum_{s\in\mathcal{S}} \left\|x^{(k)} - \bar{x}^{(k)}\right\|$
10. If $g^{(k)} < \epsilon$, then go to step 5. Otherwise, terminate.

Rockafellar and Wets (1991)

Sandia National Laboratories

PySP: Generic Progressive Hedging (1)

- If you don't care about the value of the penalty parameter ρ, you are willing to take chances, and/or you have time to kill:

```
runph --model-directory=models --instance-directory=scenariodata
```

- If you think a global value of the penalty parameter will work:
  - Add the argument "--default-rho=your-favorite-value"
- More likely, you want to implement variable-specific strategies:
  - Add the argument "--rho-cfgfile=myrhostrategy.cfg"

myrhostrategy.cfg:

```
model_instance = self._model_instance # syntatic sugar

for i in model_instance.ProductSizes:
   self.setRhoAllScenarios(model_instance.ProduceSizeFirstStage[i], \\
                           model_instance.SetupCosts[i] * 0.001)
   self.setRhoAllScenarios(model_instance.NumProducedFirstStage[i], \\
                           model_instance.UnitProductionCosts[i] * 0.001)
   for j in model_instance.ProductSizes:
      if j <= i:
         self.setRhoAllScenarios(model_instance.NumUnitsCutFirstStage[i,j], \\
                                 model_instance.UnitReductionCost * 0.001)
```

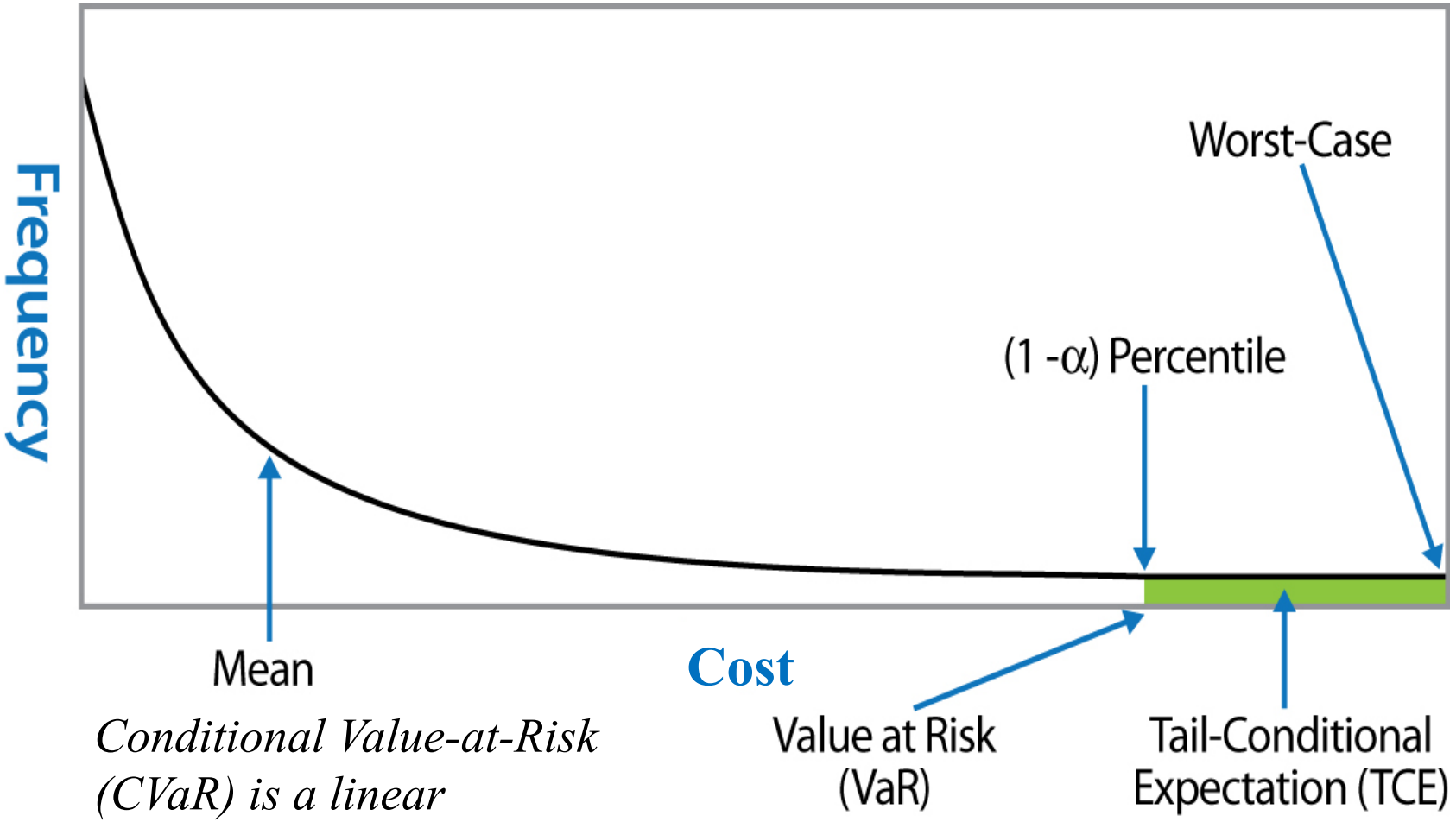

Sandia National Laboratories

PySP, Distributed Computation, and Progressive Hedging

- Decomposition algorithms for solving multi-stage stochastic mixed-integer programs are "naturally" parallelizable
  - L-shaped method and Progressive Hedging are particularly amenable
- PySP supports simple master-slave parallelism
  - Python pickle module for serialization
  - PYRO: Python Remote Objects
- Scalability to $O(1000)$ scenarios and processors
  - Academics don't have commercial solver license issues!
  - For non-academics, prototype EC2/Gurobi deployment

Sandia National Laboratories

Mean versus Risk? A Matter of Taste!



Conditional Value-at-Risk (CVaR) is a linear approximation of TCE

Sandia National Laboratories

Progressive Hedging and Conditional Value-at-Risk

- Scenario-based decomposition of Conditional Value-at-Risk models is conceptually straightforward (Schultz and Tiedemann 2006)

Proposition 5.1. *Assume that μ is discrete with finitely many scenarios $h_1, \ldots, h_J$ and corresponding probabilities $\pi_1, \ldots, \pi_J$. Let $\alpha \in (0, 1)$. Then the stochastic program*

$$\min\{Q_{CVaR_\alpha}(x) \ : \ x \in X\} \tag{11}$$

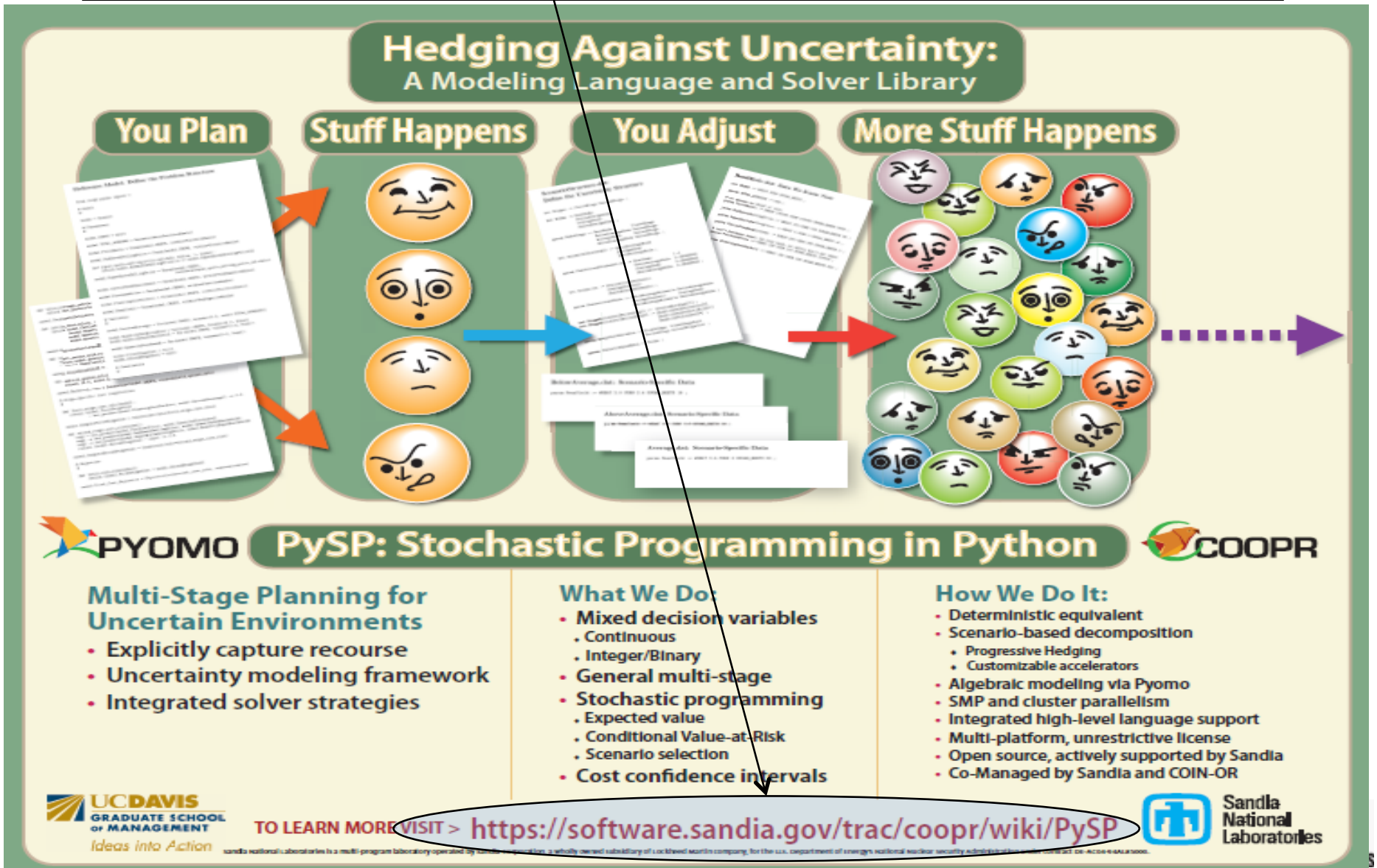
can be equivalently restated as

$$\begin{aligned} \min_{x,y,y',v,\eta} \Bigg\{ \eta + \frac{1}{1-\alpha} \sum_{j=1}^{J} \pi_j v_j \ : \ & W y_j + W' y'_j = h_j - T x, \\ & v_j \geq c^\top x + q^\top y_j + q'^\top y'_j - \eta, \\ & x \in X, \quad \eta \in \mathbb{R}, \quad y_j \in \mathbb{Z}_+^{\bar{m}}, \\ & y'_j \in \mathbb{R}_+^{m'}, \ v_j \in \mathbb{R}_+, \quad j = 1, \ldots, J \Bigg\}. \end{aligned} \tag{12}$$

- But
- Computational issues are largely unexplored

Sandia National Laboratories

PySP: For More Information…!



Extension to Nonlinear Stochastic Programs

- PYOMO and PySP Provide an Effective Modeling Environment
- Nonlinear Extensions to Pyomo
  - NL Writer – Interfaced to all AMPL Solvers (IPOPT, COUENNE)
  - AMPL Solver Library (ASL) provides derivatives
- Nonlinear Progressive Hedging Implemented
- Interface to Internal Decomposition Algorithm in Progress

Internal Decomposition with Interior-Point Methods

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned} \longrightarrow \begin{aligned} \min_x \quad & f(x) - \mu \cdot \sum_i ln(x_i) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned}$$

$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_c I \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta \lambda \end{pmatrix} = - \begin{bmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k) \ \lambda_k \\ c(x_k) \end{bmatrix}$$

$$\left(W_k = \nabla^2_{xx}\mathcal{L} = \nabla^2_{xx} f(x_k) + \nabla^2_{xx} c(x_k)\lambda\right) (\delta_w, \delta_c \geq 0) \ \left(\Sigma_k = Z_k X_k^{-1}\right)$$

- Dominant Computational Expense: Solution of Linear System at Each Iteration

Internal Decomposition with Interior-Point Methods

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned} \longrightarrow \begin{aligned} \min_x \quad & f(x) - \mu \cdot \sum_i ln(x_i) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned}$$

$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_c I \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta \lambda \end{pmatrix} = - \begin{bmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k) \ \lambda_k \\ c(x_k) \end{bmatrix}$$

$$\left(W_k = \nabla^2_{xx}\mathcal{L} = \nabla^2_{xx} f(x_k) + \nabla^2_{xx} c(x_k)\lambda\right) \left(\delta_w, \delta_c \geq 0\right) \left(\Sigma_k = Z_k X_k^{-1}\right)$$

- Dominant Computational Expense: Solution of Linear System at Each Iteration
- Structure in the Problem Definition Induces Structure in the Linear System
- All Scale-Dependent Linear Operations Should be Parallelized

Explicit Schur-Complement Approach

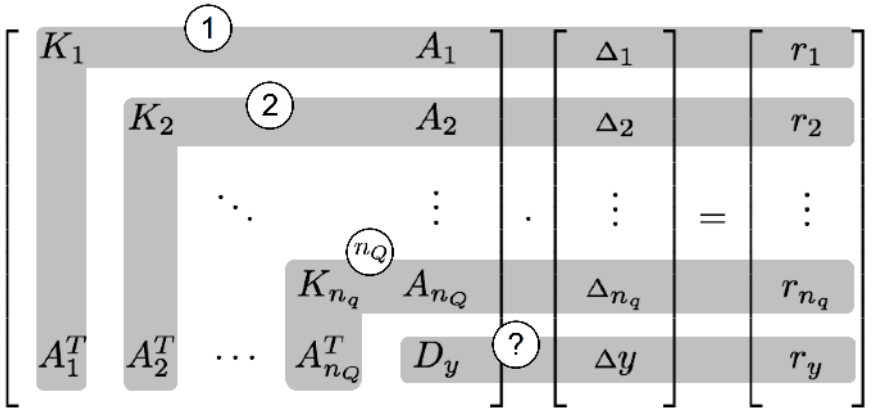
$$\min_{x_q, y} \quad \sum_{q \in \mathcal{Q}} f_q(x_q)$$

$$\begin{aligned} \text{s.t.} \qquad c_q(x_q) &= 0 \\ d_q^L \le d_q(x_q) &\le d_q^U \\ x_q^L \le x_q &\le x_q^U \qquad \forall q \in \mathcal{Q} \\ L_q^x x_q - L_q^y y &= 0 \end{aligned}$$

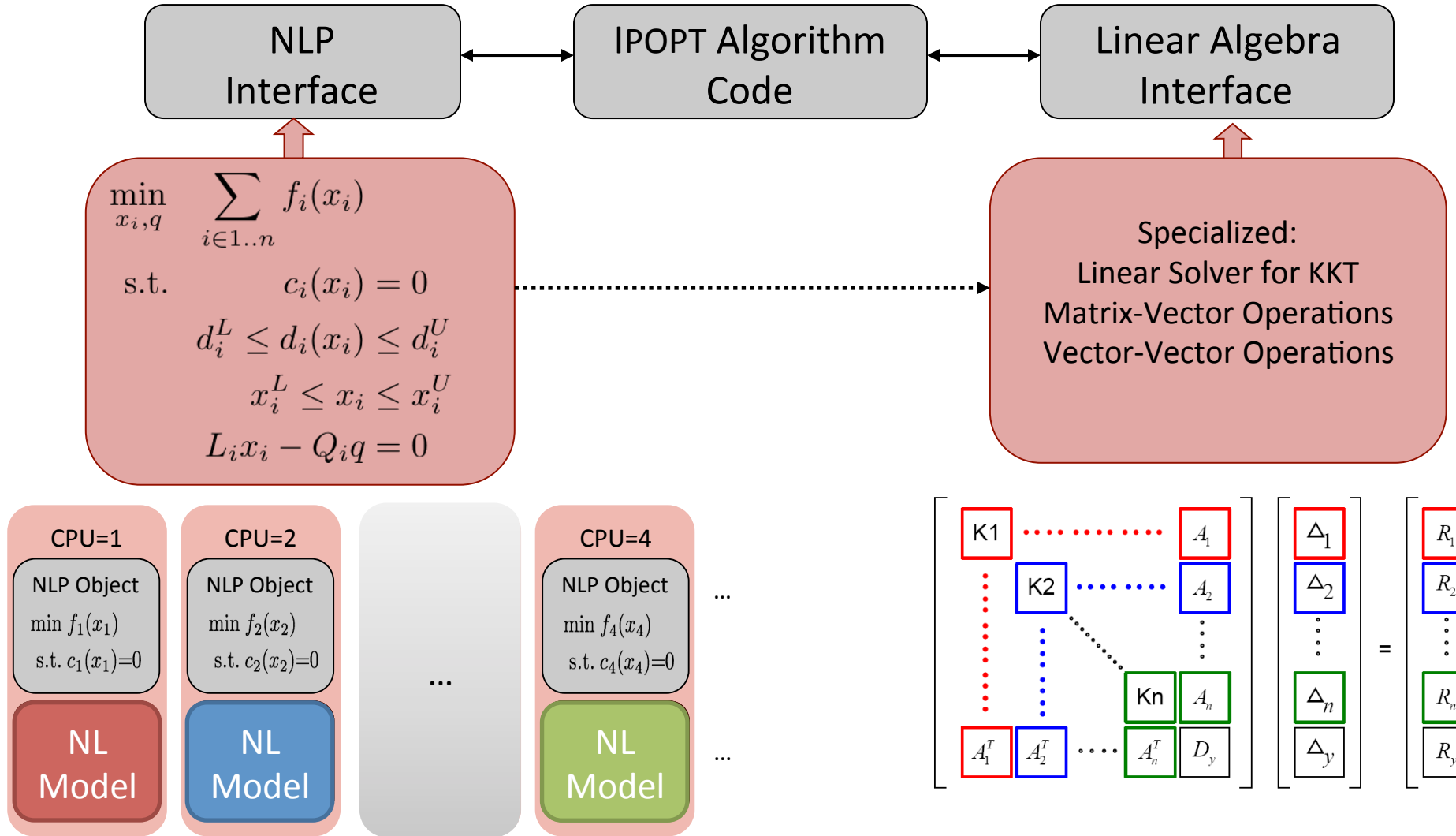
$$\begin{bmatrix} K_1 & & & & A_1 \\ & K_2 & & & A_2 \\ & & \ddots & & \vdots \\ & & & K_{n_q} & A_{nQ} \\ A_1^T & A_2^T & \cdots & A_{nQ}^T & D_y \end{bmatrix} \begin{bmatrix} \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_{n_q} \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n_q} \\ r_y \end{bmatrix}$$

$$\left[D_y - \sum_{q \in Q} A_q^T K_q^{-1} A_q \right] \Delta y = r_y - \sum_{q \in Q} A_q^T K_q^{-1} r_q$$

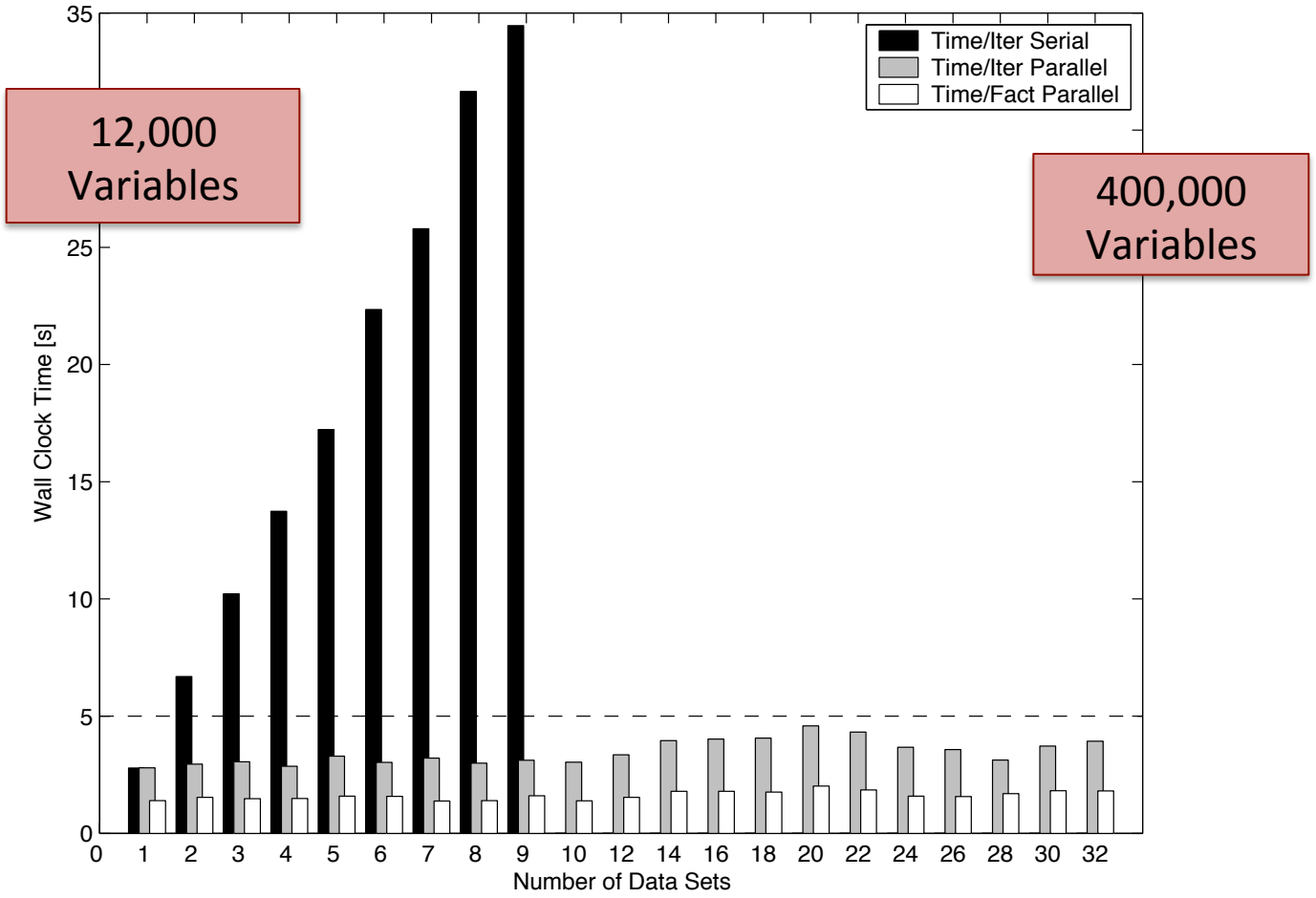
$$K_q \Delta_q = r_q - A_q \Delta y$$



Internal Decomposition: Implementation

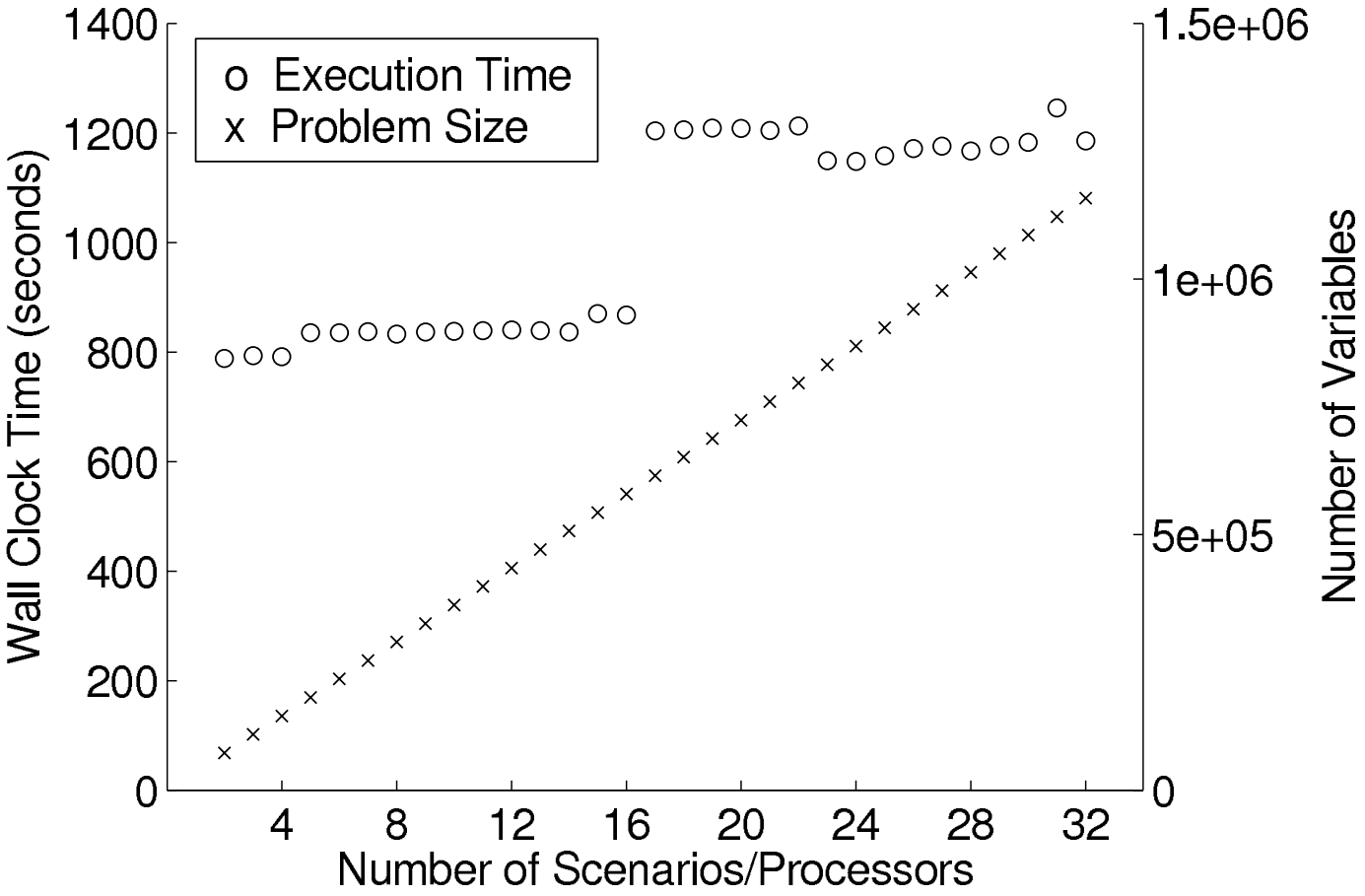


Parameter Estimation of LDPE Plant (MIMD Cluster)



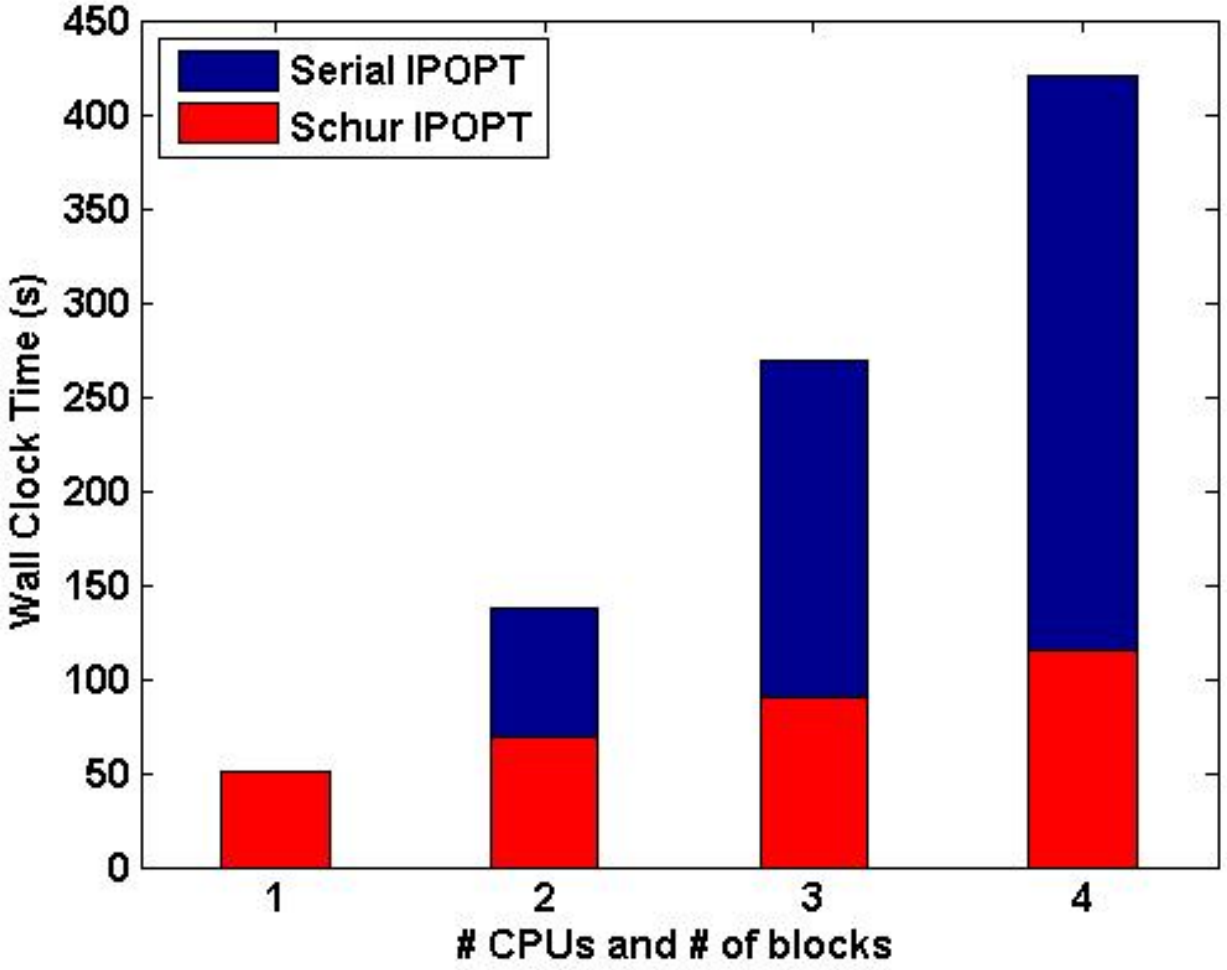
Zavala, V.M., Laird, C.D., Biegler, L.T. "Interior-Point Decomposition Approaches for Parallel Solution of Large-Scale Nonlinear Parameter Estimation Problems", Chemical Engineering Science, 63 (19), pp. 4834-4845, 2008.

Parallel Solution: Water Network Inversion

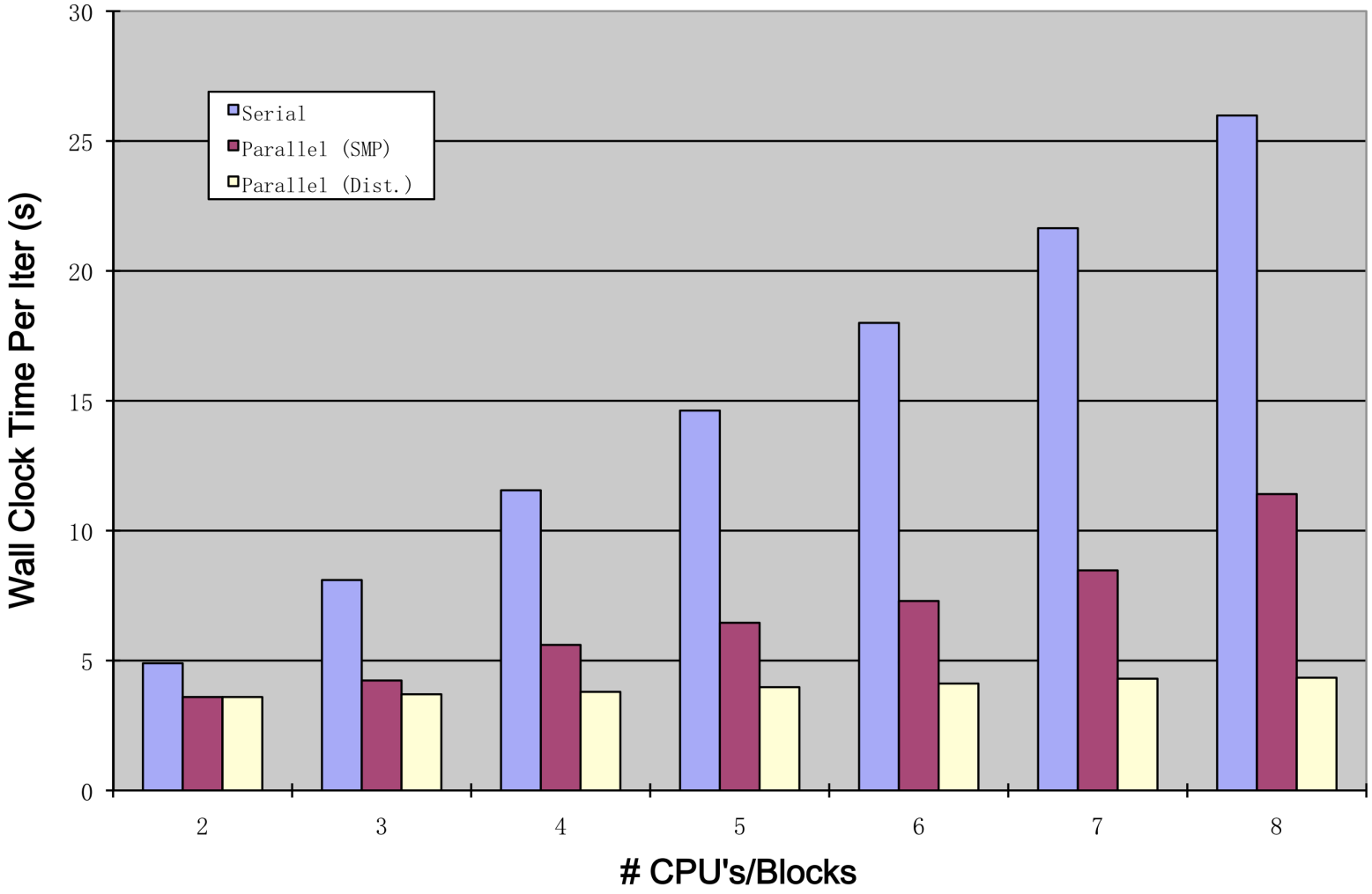


C. D. Laird and L. T. Biegler, "Large-Scale Nonlinear Programming for Multi-scenario Optimization," pp. 323-336, in Modeling, Simulation and Optimization of Complex Processes, H. G. Bock, E. Kostina, H-X Phu, R. Ranacher (eds.), Springer (2008)

Optimal Operation of Air Sep. Process



Dynamic Optimization Under Uncertainty



Stochastic Nonlinear Programming with Pyomo

- Hardware Manufacturers Focusing on Multi-core Architectures
- Problems are Increasing in Size & Complexity (Stoch. Prog.)
- Effective Parallel Solution of Large-Scale Problems Requires:
  - Modeling Environments that Support Parallel Execution
  - Efficient Parallel Algorithms
- PYOMO and PySP Extension Provide Modeling
- PySp: Progressive Hedging
  - Provides Problem-Level Decomposition Approach
- Parallel Solution of Structured KKT: Interior-Point Methods
  - Explicit Schur-Complement Techniques :Excellent Scalability for Large-Scale Problems with Limited Coupling (100's)
  - PCG/BFGS Schur-Complement Approach: 10X Improvement: Allows for Substantial Coupling (1000's)
- Framework Provides Environment
  - Application Development
  - Hybrid Algorithm Prototyping

Acknowledgements / Support

Explicit Schur-Complement Approach

N: Number of Blocks, M: Number of Coupling Variables

Form the Schur-Complement
- ~~N~~ 1 Factorizations of K-blocks
- ~~N*M+M~~ 1 Backsolves of K-blocks

$$SC = [D_y - \sum_{q \in Q} A_q^T K_q^{-1} A_q]$$

$$r_{sc} = r_y - \sum_{q \in Q} A_q K_q^{-1} r_q$$

Solve the Schur-Complement
for Step in Common Variables

- Single Dense Linear Solve of M*M Matrix

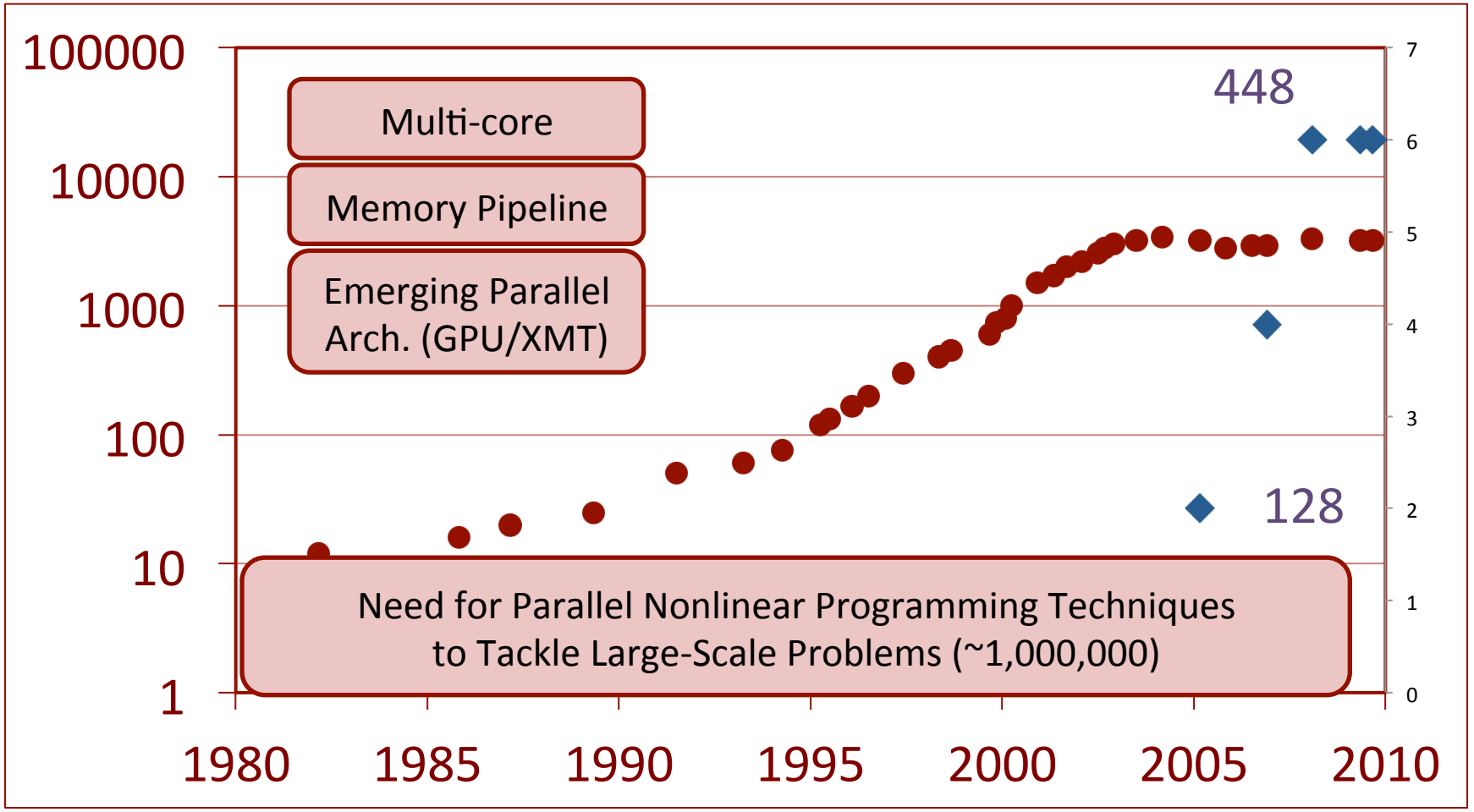
$$[SC]\, \Delta y = r_{SC}$$

Solve Remaining K-blocks for Step in other Variables

- ~~N~~ 1 Backsolves of K-blocks

$$K_q \Delta_q = r_q - A_q \Delta y$$

Changing Hardware Landscape



Parallel Solution of NLP Problems

- Shift in Hardware Focus Requires Parallel Algorithms
- Desire to Increase Problem Size Requires Parallel Tools
  - Stochastic Programming
- Challenges for Application Developers
  - Absence of Off-The-Shelf Solvers
  - Lack of Support in Modeling tools
    - Require: Parallel Evaluation of Functions and Gradients

Inversion and Model Calibration for LNG Dispersion

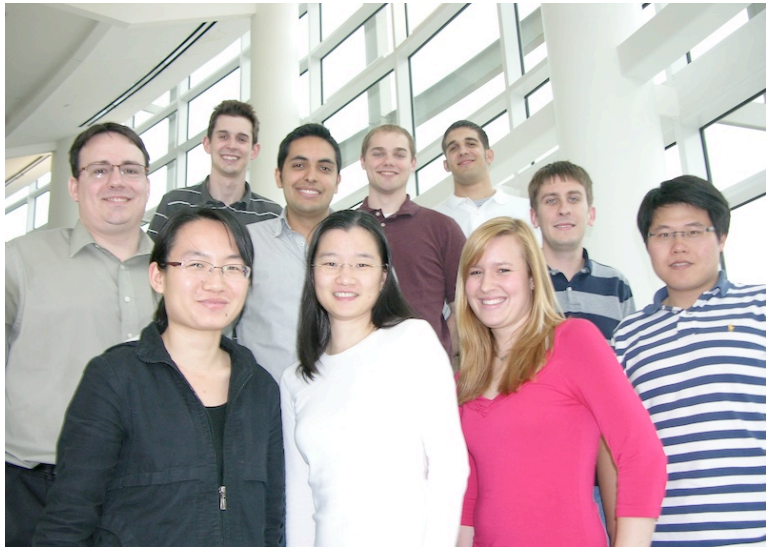
MKOPSC

Parallel NLP Techniques in Rapid Therapeutics Manufacturing

NSF CAREER

Risk Management and Planning in Complex Networks

DOE ASCR/Sandia



Parameter Estimation in Signal Transduction Across Populations

NSF CDI-II

Optimization of Complex Processes Under Uncertainty

TAMU

Real-time Response Management for WDS

PUB Singapore/Sandia

Inversion and Model Calibration for LNG Dispersion

MKOPSC

Parallel NLP Techniques in Rapid Therapeutics Manufacturing

NSF CAREER

Risk Management and Planning in Complex Networks

DOE ASCR/Sandia

- Large-Scale NLP Problems
  - Millions of variables
  - Sparse, Usually Structured
  - Not Solvable off-the-shelf

Parameter Estimation in Signal Transduction Across Populations

NSF CDI-II

Real-time Response Management for WDS

PUB Singapore/Sandia

Optimization of Complex Processes Under Uncertainty

TAMU

Parallel Solution on MIMD

Large-Scale Parameter Estimation

Multi-scenario Optimization Under Uncertainty

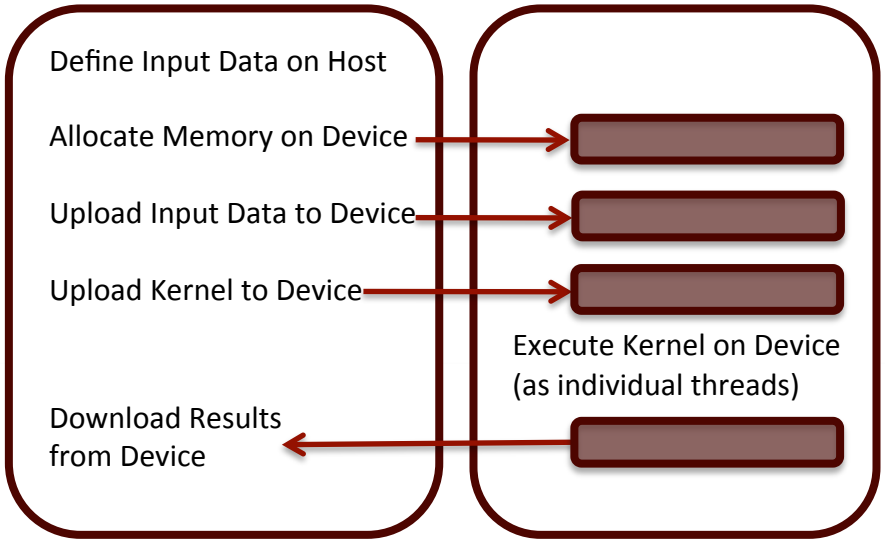
Spatial Decomposition

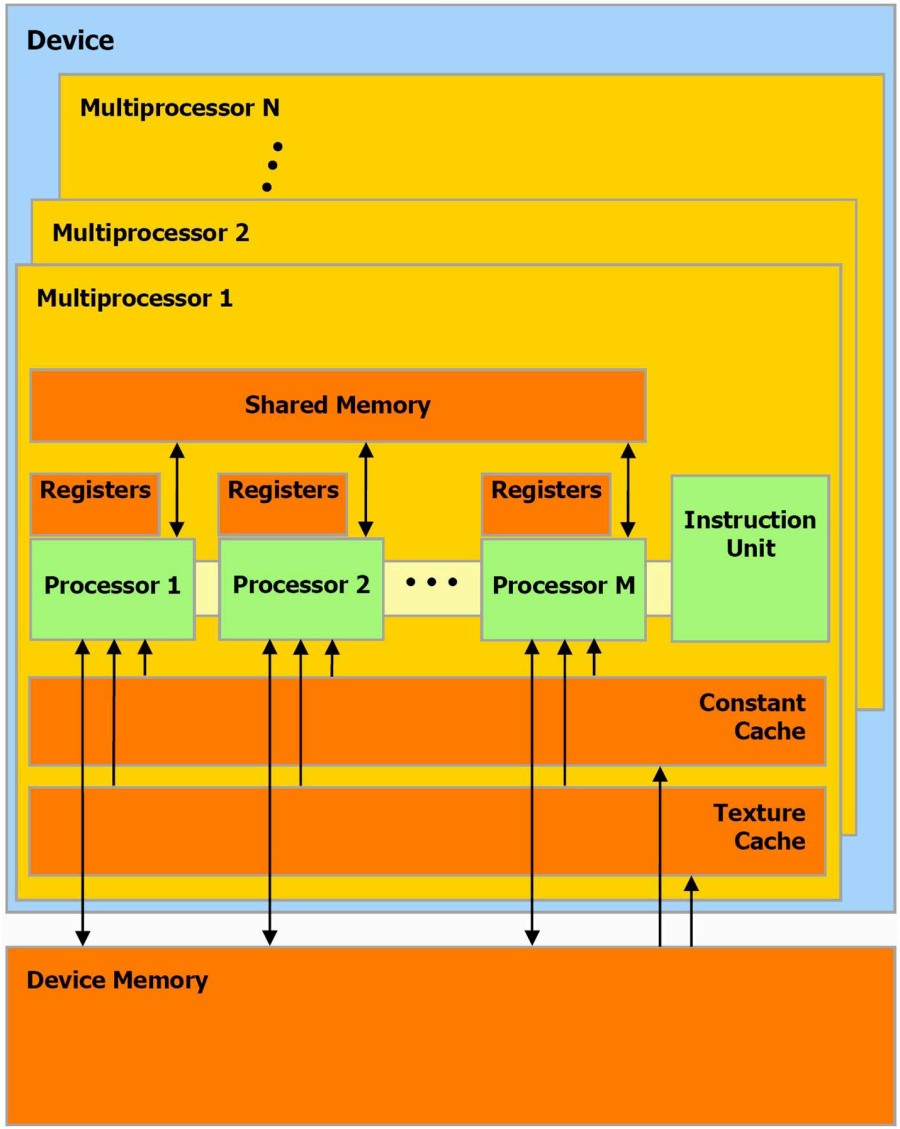
- Excellent Scalability for Problems with Few Common Variables (hundred)
- BFGS Based Preconditioned CG Removes Forming of Schur-Complement (Current Research)

Other Parallel Architectures? (GPU, Cray XMT)

Structure of the GPU

- NVidia Tesla C870 (2 Gen. Old)
  - 16 Multiprocessors
  - 8 Stream Processor
  - 128 Cores
- SIMD Architecture
- Complex Memory Structure
  - Global Memory (High Latency)
  - Shared Memory (Fast, Small)
  - Constant Memory (Read-Only)





Opportunities for Parallel NLP on GPU

Opportunities for Direct Decomposition?

Opportunities for use of Parallel Iterative Linear Solvers?

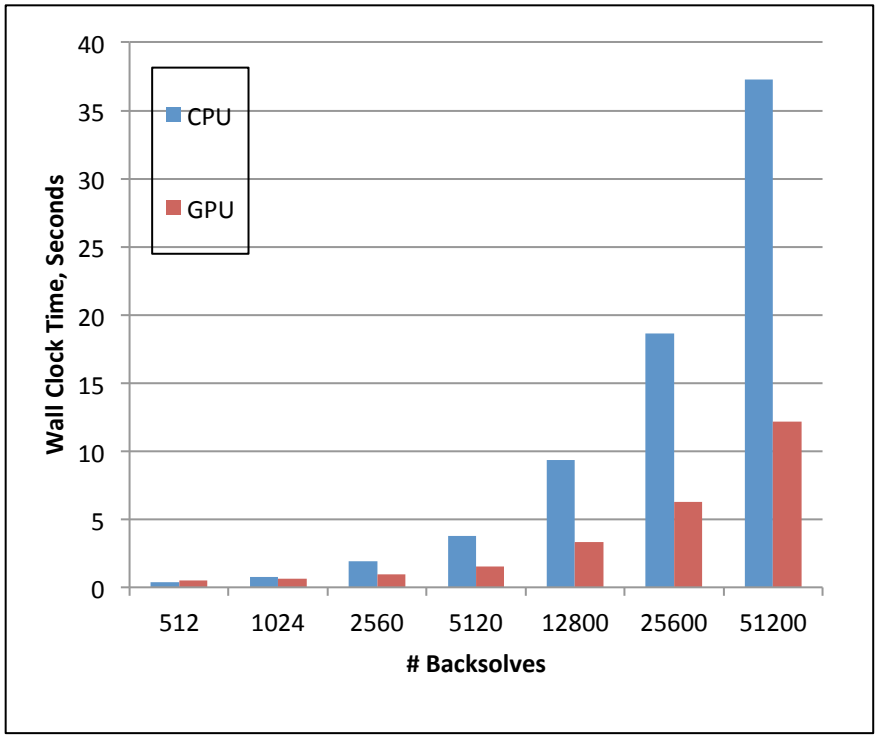
- GPU is a (hybrid) SIMD Architecture
  - Each Stream Processor Running the Same Instruction
  - Cannot Simply Run Separate Linear Solvers on Different Blocks (Execution Flow Differs)
    - Fixed-Pivot Strategies for Direct Decomposition (Code Generation)
  - Very Appropriate for Simple Linear Algebra Operations (Matrix-Vector, Vector-Vector)
    - Iterative Techniques

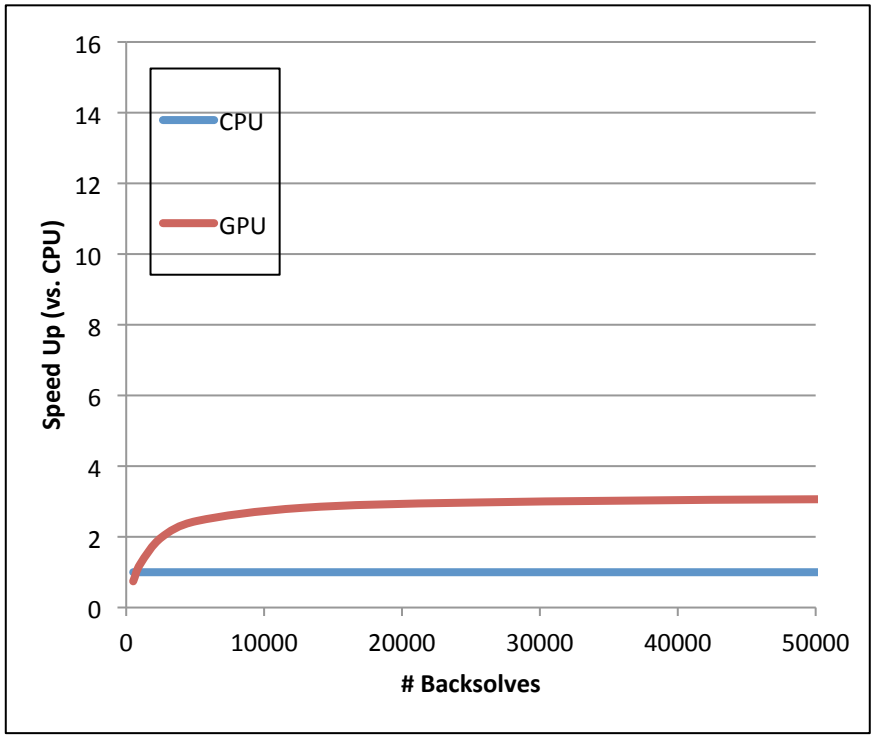
Direct Decomposition on the GPU

Fixed-Pivot Strategy – i.e. Code Generation

- Preprocessing
  - Solve a Single Representative K-block
  - Write Fixed-Pivot Factorization Kernel (CUDA)
  - Write Backsolve Kernel (CUDA)
- During Solve, GPU Uses Fixed-Pivot Strategy
- Limitations:
  - No Active Pivot Selection
  - Possible Numerical Stability Issues
  - Each Block Must be the Same Structure

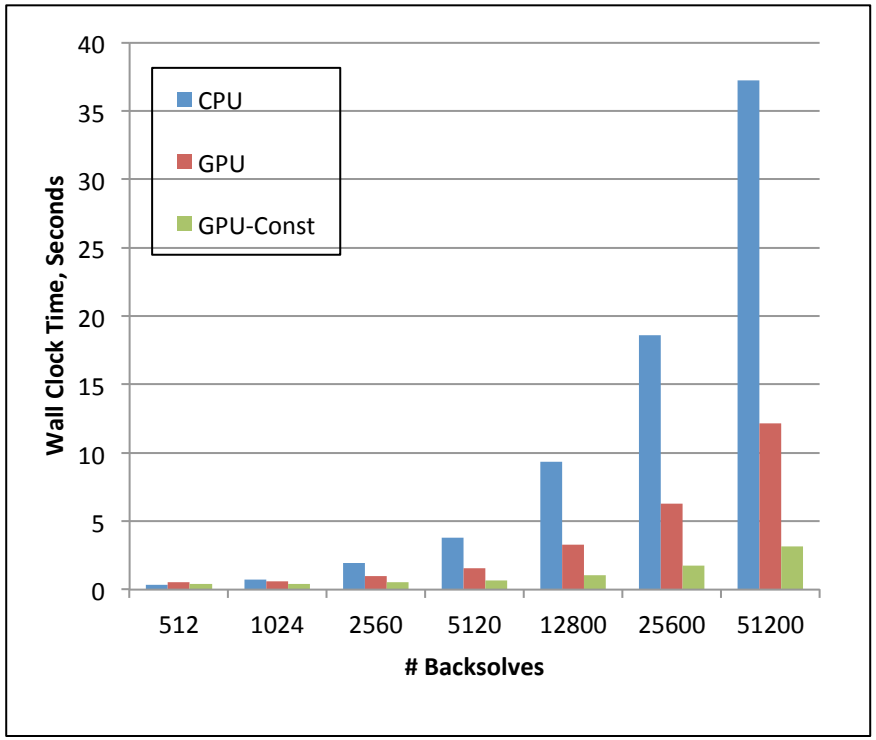
Speedup: Forming Schur-Complement

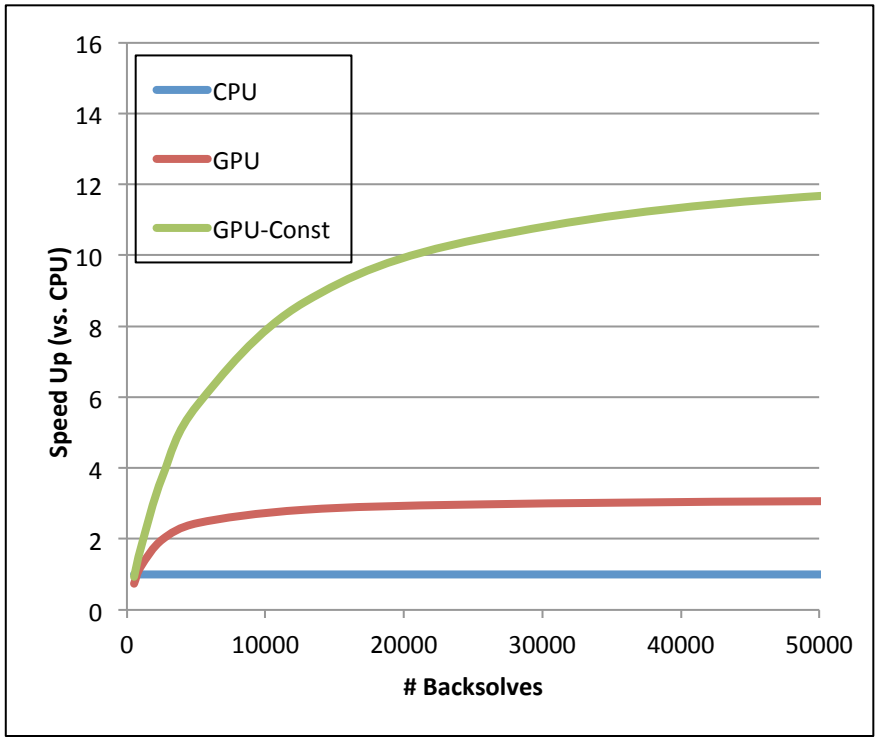




- Initial Implementation: Speedup of ~3

Speedup: Forming Schur-Complement





- Tune Parameters for Problem Structure/GPU Architecture
  - Different Memory Classes (Constant, Shared)
  - Speedup: Order of Magnitude
- Emerging SIMD Architectures Ideal for Simple Linear Operations

Iterative Techniques for KKT Systems

- Iterative Techniques (PCG, GMRES)
  - Matrix-Vector Products
  - Requires Preconditioner
  - PCG Requ
- KKT Systems
  - Ill-conditi
  - Requires
  - Linear Sys
- Dollar et al.
  - Constrai
  - Requires
- Forsgren, Gill
  - Doubly Augmented System
  - Positive Definite if Inertia is Correct (PCG)
  - Allows Detection of Incorrect Inertia
  - Allows Approximate solution
  - Effective Preconditioners?

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & \geq 0 \end{aligned}$$

$$\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} r_x \\ r_z \end{bmatrix}$$

$$\begin{bmatrix} {}^{-1}A & A^T \\ & D \end{bmatrix}$$

$$P = \begin{bmatrix} M + 2A^T D^{-1} A & A^T \\ A & D \end{bmatrix}$$

$$\begin{bmatrix} M & -A^T \\ -A & -D \end{bmatrix}$$

Methods for Iterative Solution of the KKT System Exists for Interior-Point Methods

Rely on Matrix-Vector Products (Appropriate for SIMD)

Nvidia C2050: 440 Cores, 3 GB RAM

Speedup Factor of 60 Times

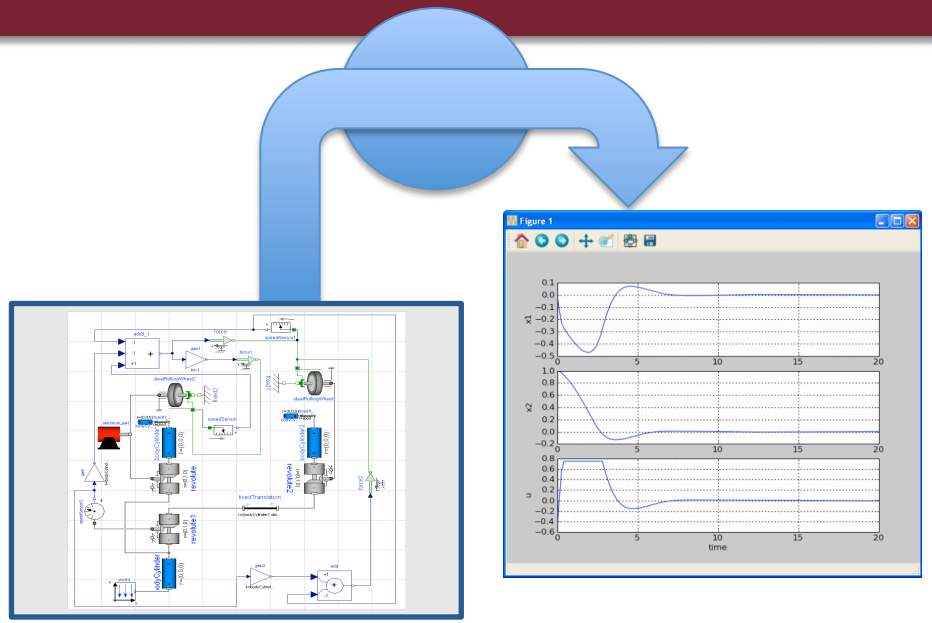
Off-the-Shelf Advances in Nonlinear Programming

- Modeling Tools:
  - Provide Algebraic / Object-Oriented Modeling Environments (AMPL, GAMS, GPROMS, Jmodelica, PYOMO, etc.)
  - Often Provide Fast First and Second Derivative Information with AD

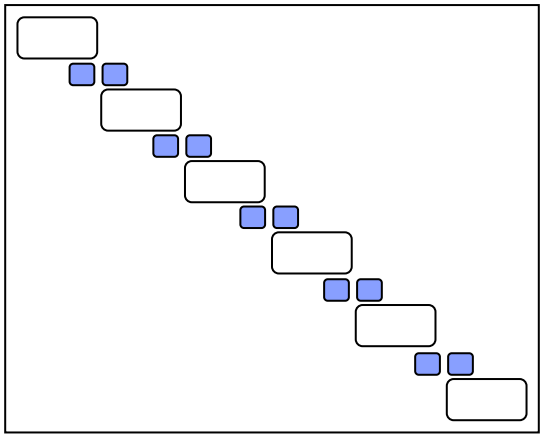
- Efficient Off-the-shelf Algorithms & Software for Large-Scale Problems. Examples:
  - Real-time Response Planning for Water Distribution Systems
    - 2.6 Million Variables → Few Minutes (QP)
  - Dynamic Parameter Estimation for Infectious Disease Models
    - 20,000 Variables → Under 30 Seconds
  - Design Under Uncertainty (Complex Air Separation Process)
    - 8000 Variables → Under 15 Seconds
    - 675,000 Variables → ~20 Minutes

Temporal Decomposition in Dynamic Optimization

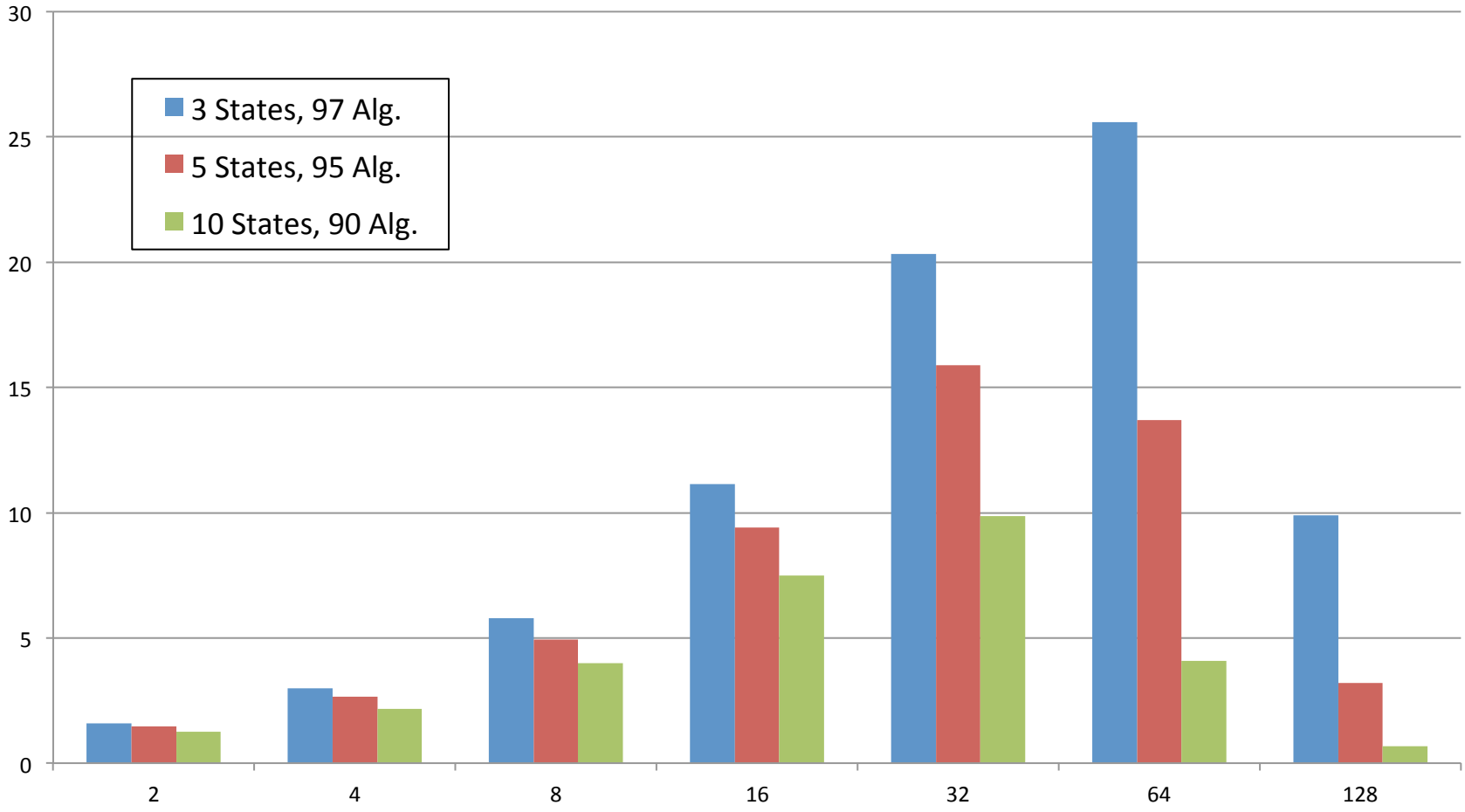
- JModelica/JOptimica
  - Developed at Lund University by Johan Akesson
  - DAE Modeled Using Modelica
  - Object-Oriented Language
  - Simultaneous Approach
  - IPOPT (NLP Solver)



- Different Structure
  - Coupling Constraints
  - Not Coupling Variables
  - Size of Schur-Complement Not Only Dependent on # Variables
  - Size of Schur-Complement Dependent on # Processors



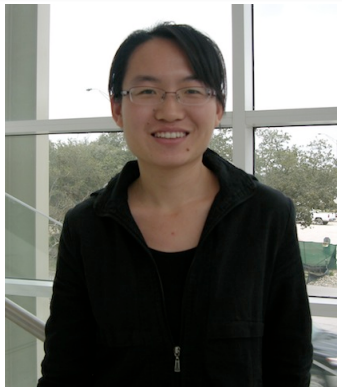
Explicit Schur-Complement: Temporal



Efficient Parallel Optimization on Emerging Architectures

Carl Laird, Assistant Professor
Jia Kang, Ph.D. Candidate (NSF CDI Type II)

Artie McFerrin Department of Chemical Engineering
Texas A&M University, College Station, TX



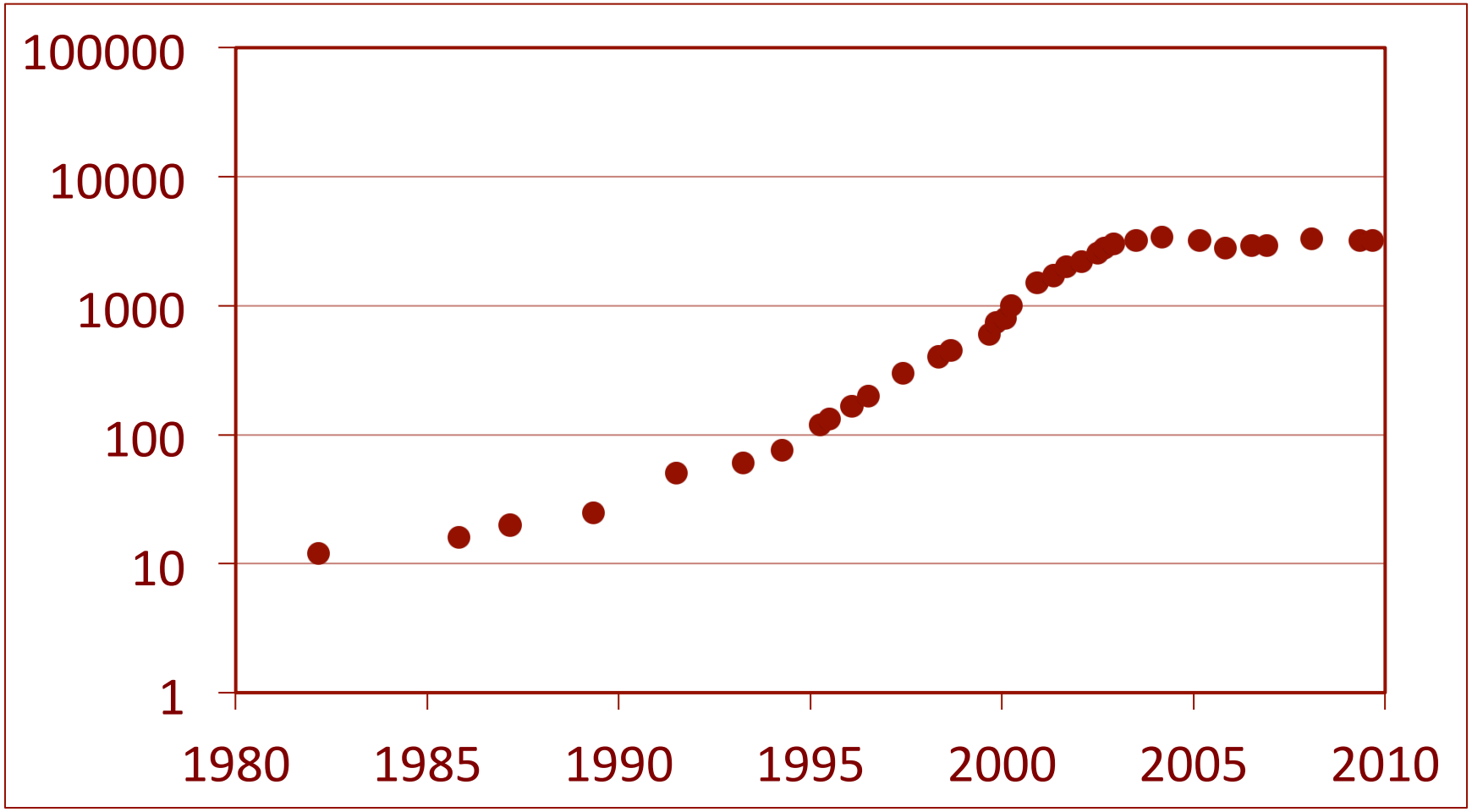
Johan Akesson, Assistant Professor

Department of Automatic Control
Lund University, Lund, Sweden

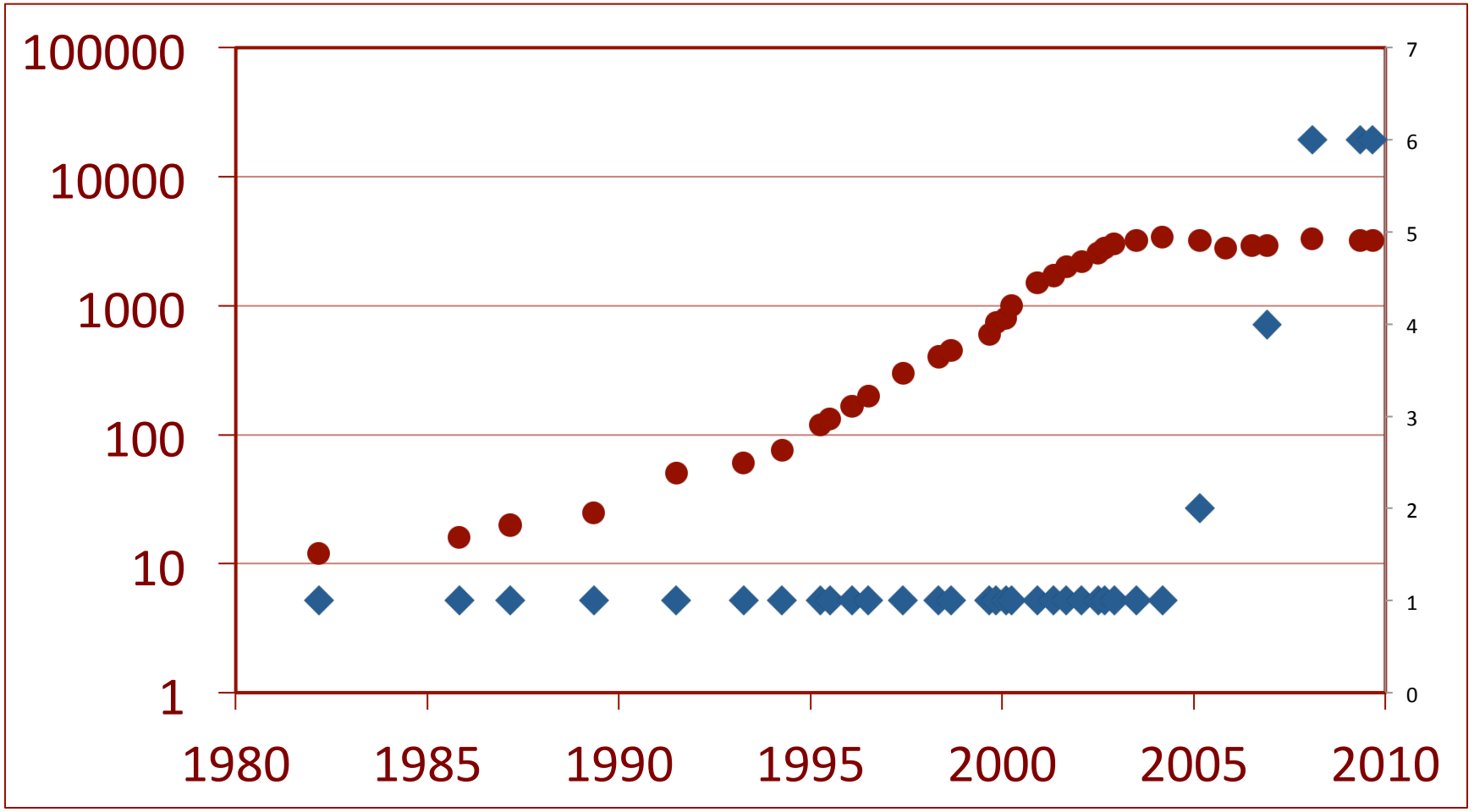
Parallel Nonlinear Programming

Architecture	Structure	Algorithm
All Parallel Architectures are Not Created Equal	Problem Structure Determines Opportunities for Parallelism	Algorithm Dictates Possibilities for Parallelism
• Distributed Cluster • Multi-core Desktop • Streaming Cores (GPU)	• Block Structure from Problem Class (scenario, temporal, etc.) • Element Level	• Problem Decomp. (PH) • Internal Decomp. • Iterative Linear Alg.

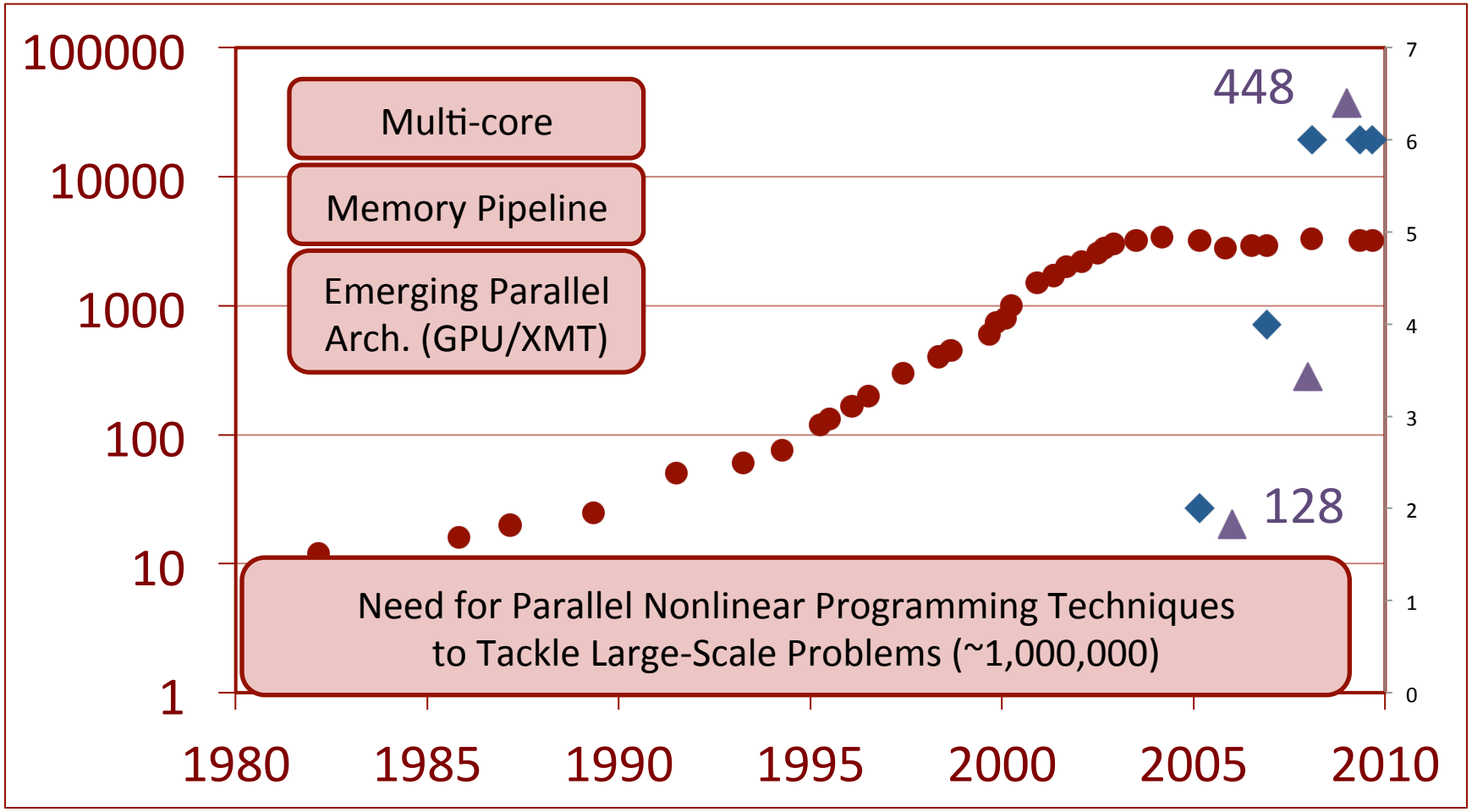
Intel: CPU Clock Rate



Intel: Number of Cores



Nvidia GPU: Number of Cores



Challenges with Explicit Schur-Complement

N: Number of Blocks, M: Number of Coupling Variables

Form the Schur-Complement
- N Factorizations of K-blocks
- N*M*M Backsolves of K-blocks

$$SC = [D_y - \sum_{q \in Q} A_q^T K_q^{-1} A_q]$$

$$r_{sc} = r_y - \sum_{q \in Q} A_q K_q^{-1} r_q$$

Solve the Schur-Complement
for Step in Common Variables

- Single Dense Linear Solve of M*M Matrix

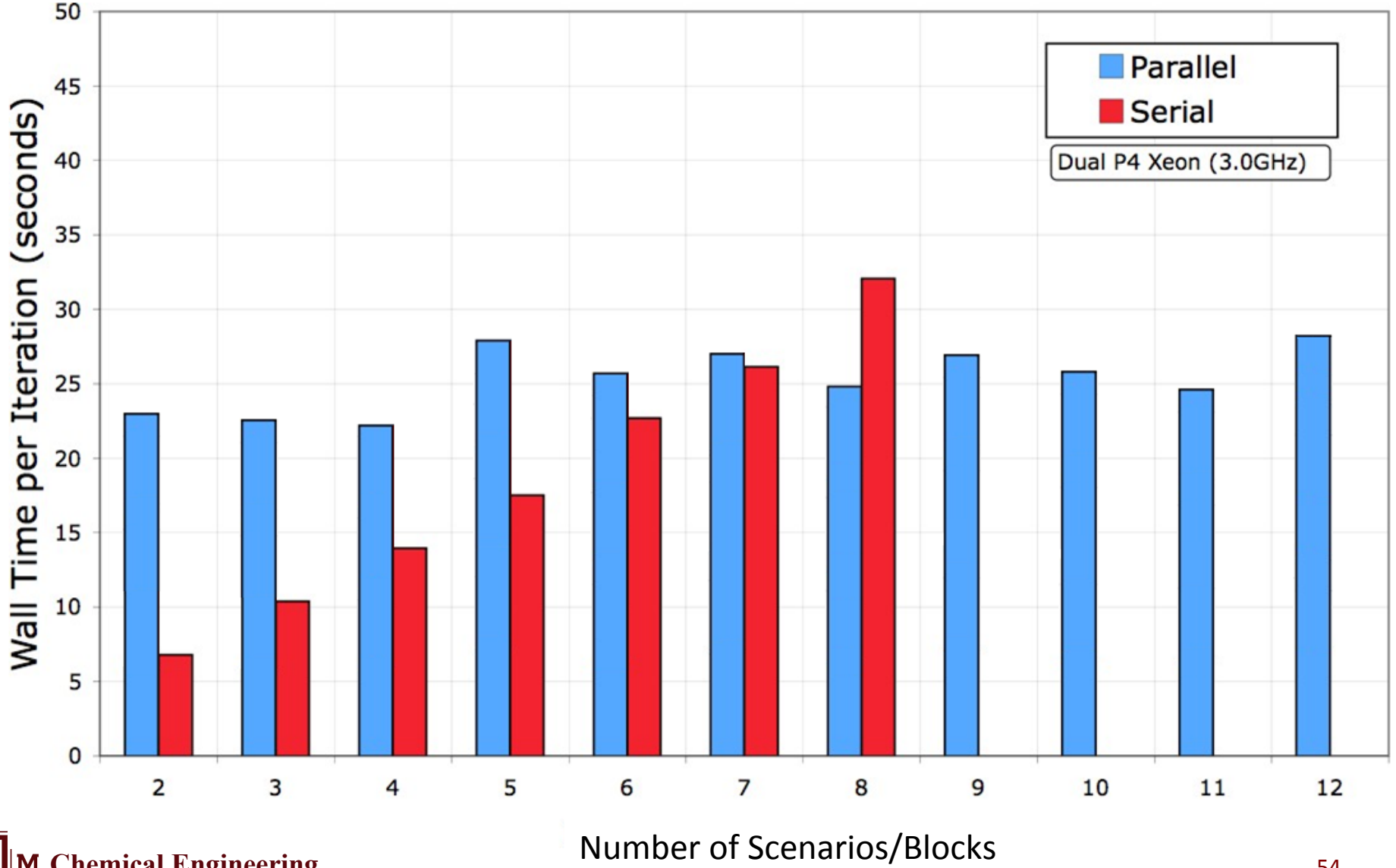
$$[SC]\,\Delta y = r_{SC}$$

Solve Remaining K-blocks for Step in other Variables

- N Backsolves of K-blocks

$$K_q \Delta_q = r_q - A_q \Delta y$$

Challenges with Explicit Schur-Complement



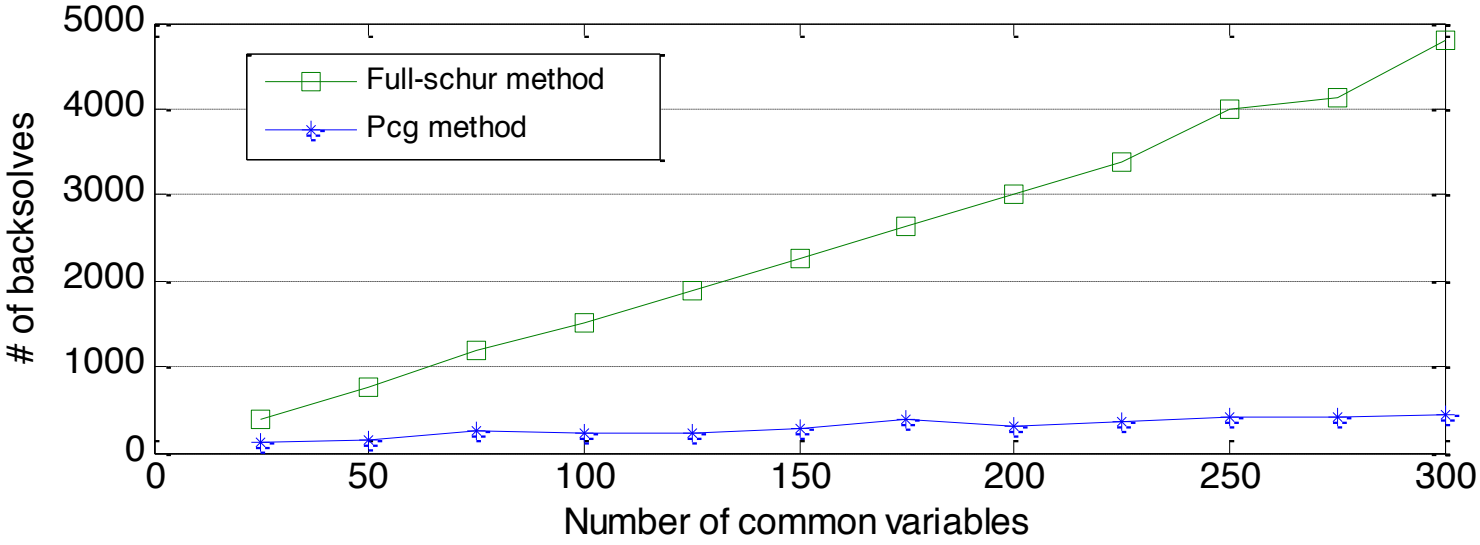
PCG/BFGS Schur-Complement Approach

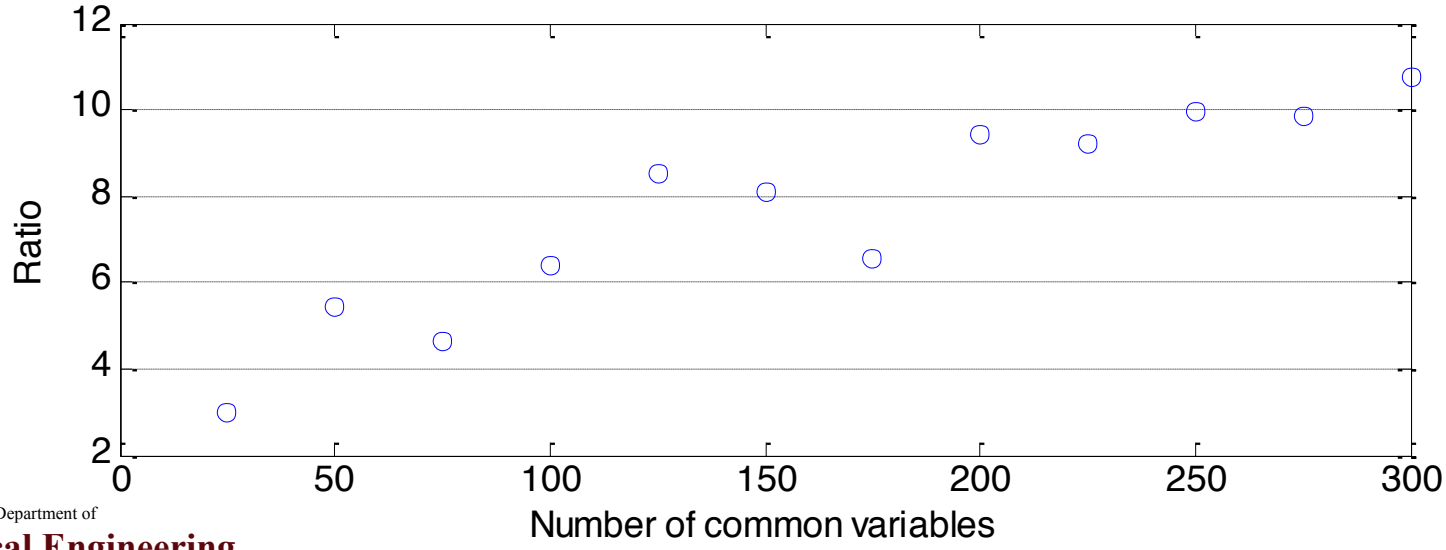
- Dominant Cost of Explicit Schur-Complement Approach
  - Forming the Schur-Complement (Increasing M)

- Never Explicitly Form the Schur-Complement
- Solve the Schur-Complement using PCG
  - 1 Backsolve of K-blocks for right hand side
  - 1 Backsolve of K-blocks per PCG Iteration
- BFGS Update for Inverse Preconditioner
  - Initial Preconditioner Calculated Explicitly

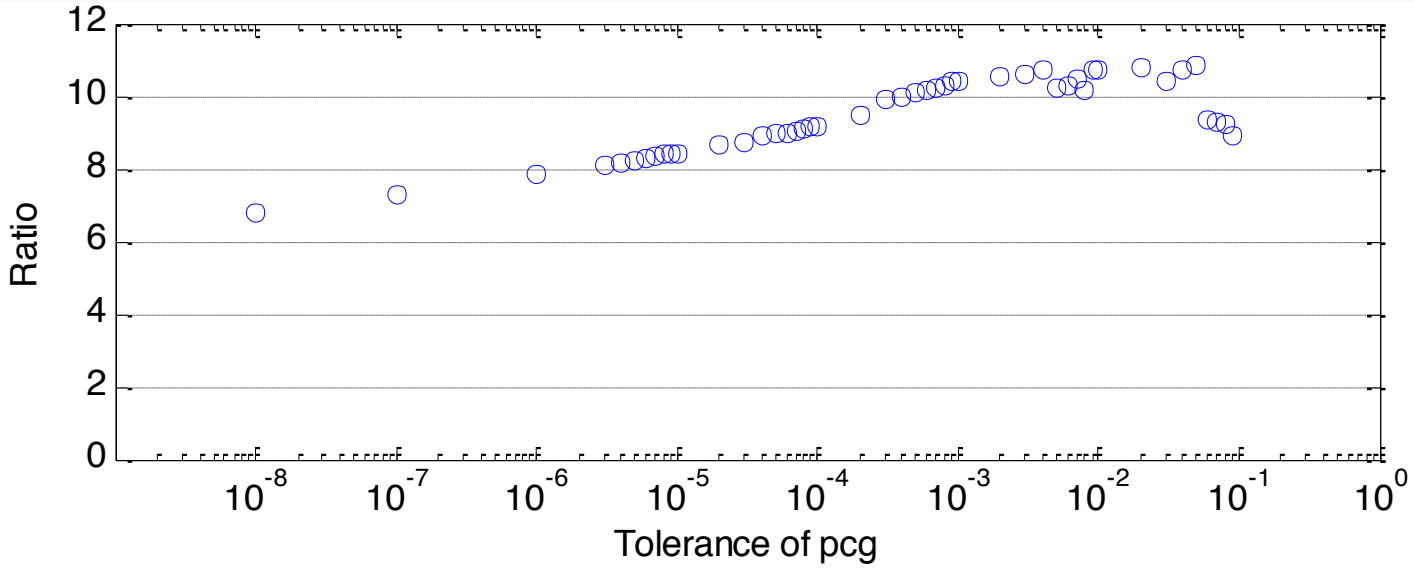
- PCG/BFGS Compared Against Explicit Schur-Complement
  - Number of Backsolves as M increases
  - Randomly Generated Parameter Estimation Problems

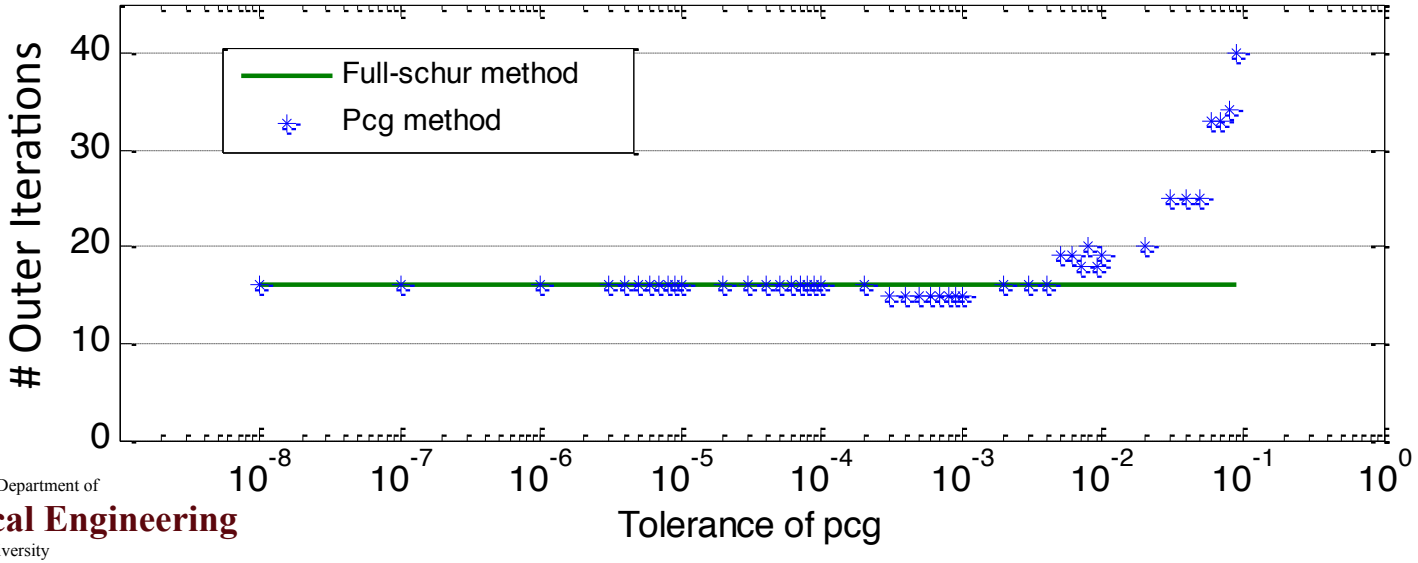
Explicit-SC vs PCG/BFGS





PCG/BFGS: Sensitivity to PCG Tolerance





Parallel Architectures

Classified by Flynn's Taxonomy

	Single Data	Multiple Data
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

Emerging Architectures (SIMD/Hybrid)

Graphics Processing Unit (GPU)

- Affordable, 1000's of Processors
- Specialized Compilers/Languages
  - CUDA, Brook++, OpenCL
- Several Complexities & Limitations

Beowulf Cluster (MIMD)

- Scalable Distributed Computing
  100s of Processors
- Standard Compilers (MPI)
- Network Communication (Ethernet)
- Communication Bottleneck

Desktop Multi-core (MIMD)

- Affordable Hardware
  Low Number of Processors
- Standard Compilers (Threads, OpenMP)
- Fast Communication (No Network)
- Memory Access/# CPU Bottleneck